Ατομική Διπλωματική Εργασία

ΜΕΘΟΔΟΙ ΚΑΙ ΤΕΧΝΙΚΕΣ ΒΕΤΛΙΩΣΗΣ ΤΗΣ ΕΚΠΑΙΔΕΥΣΗΣ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ ΑΜΦΙΔΡΟΜΗΣ ΑΝΑΔΡΑΣΗΣ ΓΙΑ ΠΡΟΒΛΕΨΗ ΔΕΥΤΕΡΟΤΑΓΟΥΣ ΔΟΜΗΣ ΠΡΩΤΕΪΝΩΝ

Συμεών Χατζηκώστας

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μέθοδοι Και Τεχνικές Βελτίωσης Της Εκπαίδευσης Νευρωνικών Δικτύων Αμφίδρομης Ανάδρασης Για Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών

Συμεών Χατζηκώστας

Επιβλέπων Καθηγητής

Δρ. Χρίστος Χριστοδούλου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2010

Ευχαριστίες

Η φοίτηση μου στο Πανεπιστήμιο Κύπρου ολοκληρώνεται με τη παράδοση αυτής της Διπλωματικής Εργασίας, που αποτελεί τη δουλεία μιας ολόκληρης χρονιάς. Μιας χρονιάς, που αποτέλεσε ίσως τη πιο δύσκολη συναισθηματικά και ψυχολογικά, χρόνια της ζωής μου.

Το μόνο θετικό της χρονιάς αυτής ήταν το γεγονός ότι μου δόθηκε η ευκαιρία να είμαι μέρος μιας ομάδας που αποτελείτο από άτομα που με βοήθησαν σε όλη τη διαδικασία ανάπτυξης της διπλωματικής εργασίας μου και που έδειξαν κατανόηση και υπομονή στις δύσκολές μέρες που πέρασα.

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, Δρ. Χρίστο Χριστοδούλου που μου έδωσε την ευκαιρία να είμαι μέρος αυτή της ομάδας καθώς επίσης και για τις πολύτιμες συμβουλές και χρήσιμες πληροφορίες που μου παρείχε στα διάφορα στάδια της διπλωματικής εργασία που είχαν σαν αποτέλεσμα την επιτυχή ολοκλήρωσης της. Επίσης θα ήθελα να ευχαριστήσω τον Δρ. Βασίλειο Προμπονά, Λέκτορα του τμήματος Βιολογικών Επιστημών του Πανεπιστημίου Κύπρου, του οποίου οι γνώσεις και οι ιδέες του στον τομέα της βιολογίας ήταν πολύ σημαντικές για την εργασία μου.

Εν συνεχεία, θα ήθελα να ευχαριστήσω τα υπόλοιπα μέρη της ομάδας, το διδακτορικό φοιτητή Μιχάλη Αγαθοκλέους, το Δρ. Πέτρο Κουντούρη και το διδακτορικό φοιτητή Βασίλη Βασιλειάδη, για όλη τη βοήθεια που μου πρόσφεραν όταν χρειάστηκα τις γνώσεις και τις εμπειρίες τους.

Τέλος θα ήθελα να αφιερώσω αυτή τη Διπλωματική Εργασία στο άνθρωπο που με ώθησε στη κατεύθυνση της Πληροφορικής και ο οποίος δεν βρίσκεται πλέον μαζί μας, το πατέρα μου, Κύπρο Χατζηκώστας.

Σας ευχαριστώ,

Χατζηκώστας Συμεών

Περίληψη

Οι πρωτεΐνες στους οργανισμούς αποτελούν πολύ σημαντικό και απαραίτητο παράγοντα στη σωστή λειτουργία του. Οι πρωτεΐνες προσδιορίζονται από το DNA, στο οποίο βρίσκονται αποθηκευμένες όλες οι πληροφορίες που είναι απαραίτητες για τις λειτουργίες του κύτταρου του οργανισμού. Οι πρωτεΐνες είναι υπεύθυνες για τα βασικά δομικά και λειτουργικά χαρακτηριστικά των κυττάρων.

Οι λειτουργίες των πρωτεϊνών, όμως καθορίζονται από την τρισδιάστατη δομή τους, δηλαδή τη δομή που έχει η πρωτεΐνη στο χώρο. Για να φτάσουμε όμως στη τριτοταγή δομή πρέπει να γνωρίζουμε πρώτα τη δευτεροταγή και τη πρωτοταγή δομή τους. Με το όρο πρωτοταγής δομής εννοούμε βασικά την ακολουθία των αμινοξέων (20 διαφορετικά) από τα οποία αποτελείται η συγκριμένη πρωτεΐνη. Η δευτεροταγούς δομή είναι ουσιαστικά μια αναπαράσταση της τριτοταγούς δομής με τη χρήση των $\hat{\alpha}$-helix, β-sheet, loops και coils, των τοπικών δηλαδή διαμορφώσεων τις πολυπεπτιδικής αλυσίδας (αμινοξέων). Η σύνθεση των helixes και sheets στο χώρο δίνει τελικά τη τριτοταγή δομή.

Σκοπός της συγκεκριμένης εργασίας είναι να προβλέψουμε σωστά τη δευτεροταγή δομή μιας πρωτεΐνης, γνωρίζοντας την αλυσίδα των αμινοξέων από την οποία αποτελείται. Αυτό μπορεί να θεωρηθεί ένα αρχικό στάδιο ώστε να πάρουμε τελικά την τριτοταγή δομή που περιέχει την πληροφορία της πρωτεΐνης στο χώρο που είναι αυτό που μας ενδιαφέρει. Η εργασία βασίστηκε σε ένα σύστημα BRNN που υλοποιήθηκε αρχικά από τον Αγαθοκλέους (Αγαθοκλέους, 2009), τροποποιήθηκε από τη Χριστοδούλου (Χριστοδούλου, 2010), και βασίζεται στην αρχιτεκτονική που εισήγαγε πρώτος στον χώρο ο Baldi και οι συνεργάτες του (Baldi et al., 1999, Baldi et al., 2000).

Στη συγκεκριμένη εργασία εφαρμόζονται μέθοδοι και μηχανικές μάθησης σε διάφορα επίπεδα της ανάπτυξης του συστήματος (Preprocessing-Processing-Post processing), μέσα από τις οποίες προσπαθούμε να αυξήσουμε το ποσοστό επιτυχούς πρόβλεψης. Επίσης προσπαθούμε να αναπτύξουμε πτυχές του προβλήματος που πριν δεν είχαν επεξεργαστεί ούτως ώστε το σύστημα να επεξεργάζεται περισσότερη πληροφορία και άρα να δίνει καλύτερη πρόβλεψη. Χρησιμοποιώντας RBF (Radial Basis Function) δίκτυο το οποίο πρώτο είχε επεξεργαστεί τα δεδομένα με τη χρήση ενός Self Organized Map για να φιλτράρισμα της εξόδου το Q3 αυξάνεται μέχρι και 2%. Επιπρόσθετα με την εφαρμογή των Evolutionary Strategy έχοντας ως fitness function to SOV Score , το ποσοστό επιτυχίας σε SOV (Segment Overlap) score έφτασε μέχρι και 74.02, συγκρίσιμα με αυτά που αναφέρονται στη σύγχρονη Βιβλιογραφία.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Βιολογικό υπόβαθρο

- 1.1.1 Πρωτεΐνες και αμινοξέα
- 1.1.2 Δομή πρωτεϊνών
- 1.1.3 Ρόλος πρωτεϊνών
- 1.1.4 Πρόβλημα *protein folding*

1.2 Σχετική έρευνα

1.3 Νευρωνικά Δίκτυα

- 1.3.1 Αρχιτεκτονική Νευρωνικού Δικτύου Επιβλεπόμενη Μάθησης
- 1.3.2 Πολυστρωματικά δίκτυα perceptron εμπρόσθιου περάσματος
- 1.3.3 Πολυστρωματικά δίκτυα perceptron με ανάδραση
- 1.3.4 Αλγόριθμος μάθησης με ανάστροφη μετάδοση λάθους
- 1.3.5 Ensemble of ANN
- 1.3.6 Μη-Επιβλεπόμενη Μάθηση
- 1.3.7 Radial Basis Function Νευρωνικά Δίκτυα

1.4 Εξελικτικοί Αλγόριθμοι

- 1.4.1 Γενετικοί Αλγόριθμοι
- 1.4.2 Εξελικτικοί Αλγόριθμοί

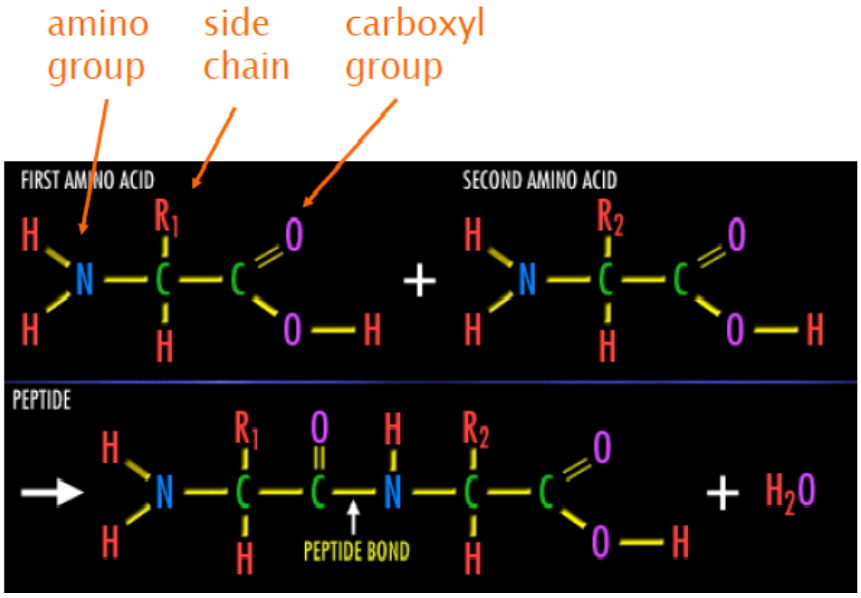
1.1 Βιολογικό υπόβαθρο

1.1.1 Πρωτεΐνες και αμινοξέα

Οι πρωτεΐνες είναι μεγάλα βιολογικά μακρομόρια, που αποτελούνται από μια ή περισσότερες πεπτιδικές αλυσίδες (πολυπεπτίδια). Ένα πολυπεπτίδιο αποτελείται από μια σειρά από αμινοξέα, τα οποία είναι τα βασικά χημικά δομικά στοιχεία που το συνθέτουν.

Από τα διάφορα αμινοξέα που απαντούν στη φύση, μόνο είκοσι χρησιμοποιούνται κατά κανόνα για τη σύνθεση των πρωτεϊνών. Τα αμινοξέα συμβολίζονται με τον κωδικό του ενός ή των τριών γραμμάτων (Αγαθοκλέους, 2009). Επομένως, η αμινοξική ακολουθία ενός πολυπεπτιδίου μπορεί να αναπαρασταθεί ως συμβολοσειρά. Τα αμινοξέα συνδέονται ομοιοπολικά μεταξύ τους με πεπτιδικούς δεσμούς, σχηματίζοντας κατ' αυτόν τον τρόπο μια γραμμική αλυσίδα (πολυπεπτιδική αλυσίδα). Κάθε αμινοξύ, αποτελείται από μια κοινή «ραχοκοκαλιά», η οποία περιλαμβάνει μια αμινο-ομάδα (-NH2) και μια ομάδα καρβοξυλίου (-COOH) ομοιοπολικά συνδεδεμένα με ένα ασύμμετρο άτομο άνθρακα (Αγαθοκλέους, 2009), εκτός από την περίπτωση της γλυκίνης (Gly). Αυτό που τα διαφοροποιεί μεταξύ τους είναι η πλευρική αλυσίδα (side chain), οι ιδιότητες της οποίας καθορίζουν τις ιδιότητες που έχουν διαφορετικά αμινοξέα (Mount, 2001).

Όπως βλέπουμε και από το σχήμα 1.1, η πλευρική αλυσίδα (R), εν γένει διαφέρει σε κάθε αμινοξύ, ενώ η υπόλοιπη αλυσίδα είναι η ίδια για όλα τα αμινοξέα. Τα R1 και R2, προσδιορίζουν χαρακτηριστικά τα συγκεκριμένα αμινοξέα. Μετά τη σύνδεσή τους για το σχηματισμό της πολυπεπτιδικής αλυσίδας αναφερόμαστε σε αυτά ως «αμινοξικά κατάλοιπα». Τα side chains (και κατά συνέπεια και τα αντίστοιχα κατάλοιπα), διαφέρουν στις διάφορες φυσικοχημικές τους ιδιότητες (π.χ. μέγεθος, σχήμα, υδροφοβικότητα).



Σχήμα 1.1: *Η πλευρική αλυσίδα (side chain), προσδίδει χαρακτηριστικές ιδιότητες σε κάθε αμινοξύ/κατάλοιπο (Από Mount, 2001).*

Τα αμινοξέα, μπορούν να διαχωριστούν σε διάφορες ομάδες, ανάλογα με τις φυσικοχημικές τους ιδιότητες. Ταξινομούνται δηλαδή κυρίως βάσει της ομάδας R, αφού αυτή συγκεκριμενοποιεί τα χαρακτηριστικά του αμινοξέως. Έχουμε για παράδειγμα τα πολωμένα, τα φορτισμένα και τα υδρόφοβα αμινοξέα (Mount, 2001). Σημαντικό είναι να αναφέρουμε ότι αυτές οι ιδιότητες των αμινοξέων παίζουν πολύ σημαντικό ρόλο στην μορφή της τριτοταγούς δομής της πρωτεΐνης στον χώρο, όπως θα δούμε και στο υπόλοιπο μέρος του κεφαλαίου.

Η αλληλουχία των διάφορων πρωτεϊνών βρίσκεται αποθηκευμένη στο γενετικό υλικό - Deoxyribonucleic acid (DNA). Το γενετικό υλικό είναι η καθορισμένη σειρά αζωτούχων βάσεων (νουκλεϊνικά οξέα) του DNA και το οποίο εμπεριέχει τα γονίδια των κυττάρων. Η μετατροπή της πληροφορίας του DNA σε πρωτεΐνες γίνεται με τις διαδικασίες μεταγραφής και μετάφρασης, που αποτελούν και το κεντρικό δόγμα της μοριακής βιολογίας. Συγκεκριμένα, η χαρακτηριστική διαδικασία η οποία αντιστοιχίζει τα νουκλεϊνικά οξέα με τα διάφορα αμινοξέα λέγεται πρωτεϊνοσύνθεση και επιτυγχάνεται με τον κώδικα τριπλέτας, όπου κάθε τρεις βάσεις κωδικοποιούν ένα αμινοξύ. Ένα παράδειγμα μιας πρωτεΐνης είναι η γνωστή

αιμοσφαιρίνη, η οποία αποτελείται από τέσσερα πολυπεπτίδια και είναι υπεύθυνη για την μεταφορά οξυγόνου στα ερυθρά αιμοσφαίρια του οργανισμού.

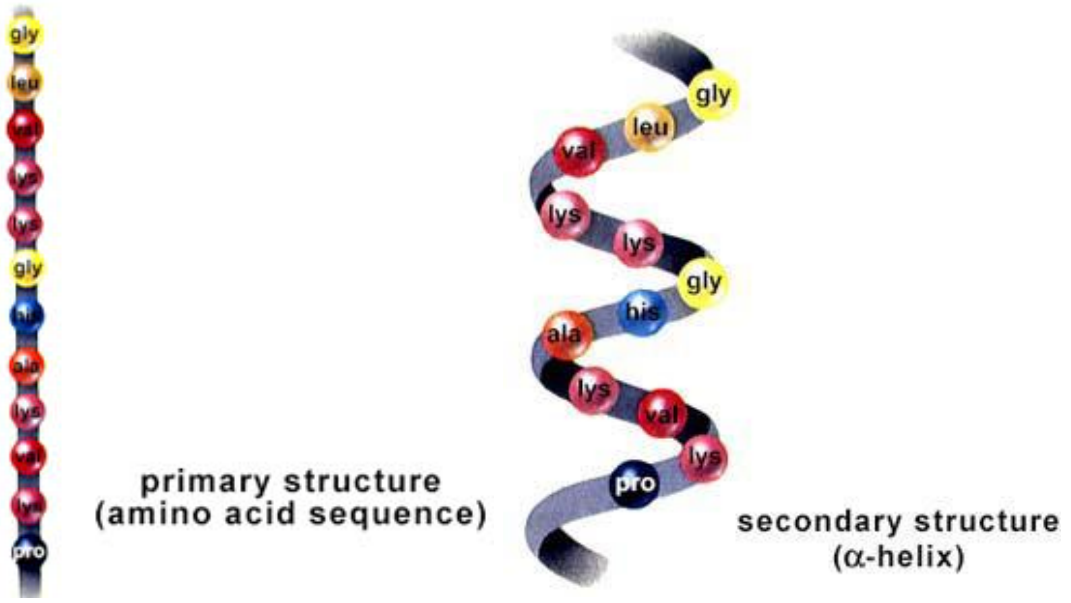
1.1.2 Δομή πρωτεϊνών

Η *τριτοταγής δομή* μιας πρωτεΐνης είναι η τρισδιάστατη μορφή που έχει η πρωτεΐνη κάτω από συγκεκριμένες συνθήκες, όπως είναι η θερμοκρασία, το pH, ο διαλύτης (π.χ. νερό) και τα υπόλοιπα διαλυμένα σε αυτόν μόρια. Η τριτοταγής δομή, είναι διαφορετική για κάθε πρωτεΐνη, και προσδιορίζει την λειτουργικότητα της. Η τριτοταγής δομή λοιπόν είναι σημαντική, επειδή οι ιδιότητες και ο τρόπος δράσης των πρωτεϊνών εξαρτώνται από τις λεπτομέρειες της τρισδιάστατης δομής τους.

Τι είναι όμως αυτό που καθορίζει τη δομή των πρωτεϊνών; Πώς η πρωτεΐνη «αναδιπλώνεται» στον χώρο σχηματίζοντας μια συγκεκριμένη δομή και τι δίνει στις πρωτεΐνες την ευστάθεια τους;

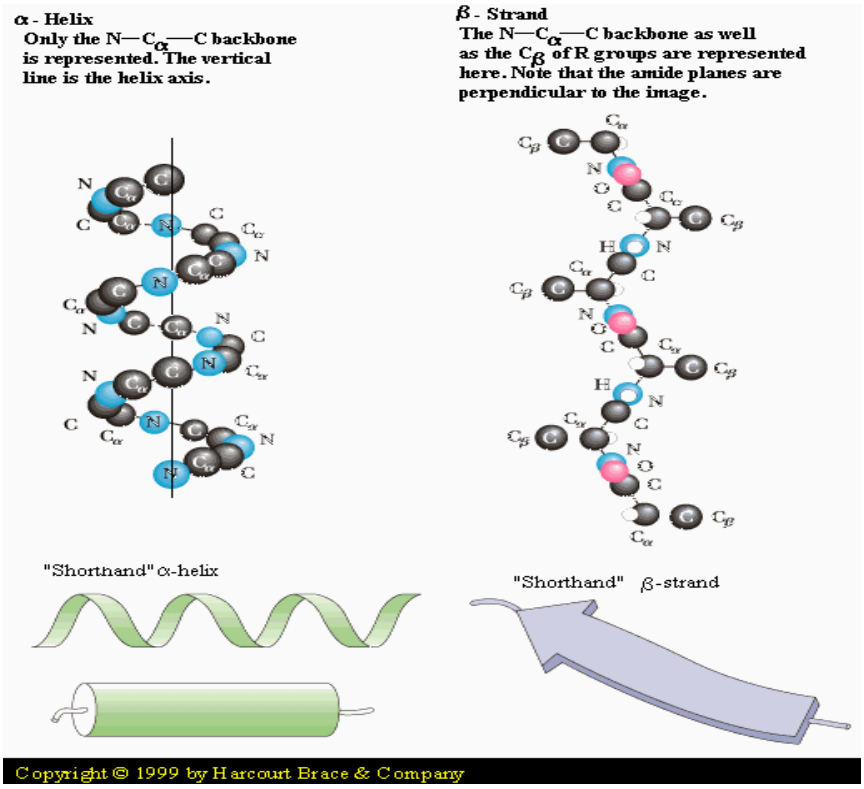
Είναι γενικά αποδεκτό ότι η ακολουθία των διαφόρων αμινοξέων από τα οποία αποτελείται η πρωτεΐνη περιέχει όλη εκείνη την πληροφορία που απαιτείται ώστε να αποκτήσει η πρωτεΐνη την τρισδιάστατη δομή της σε συγκεκριμένες συνθήκες (Mount 2001). Οι πιθανές αλληλεπιδράσεις ανάμεσα σε αυτά τα αμινοξέα είναι πολλών τύπων, συμπεριλαμβανομένων ηλεκτροστατικών αλληλεπιδράσεων, δυνάμεων Van der Waals, δεσμών υδρογόνου και υδρόφοβων αλληλεπιδράσεων. Τα υδρόφοβα αμινοξέα τείνουν να είναι εμφωλιασμένα στο εσωτερικό της δομής της πρωτεΐνης, για να μην έρχονται σε επαφή με το νερό.

Η δομή των πρωτεϊνών μπορεί να περιγραφεί με ιεραρχικό τρόπο: Η ***πρωτοταγής*** δομή μιας πρωτεΐνης (σχήμα 1.2 αριστερά) είναι η πολυπεπτιδική αλυσίδα εάν την δούμε σαν μια σειρά από αμινοξέα, δηλαδή εάν «ξεδιπλώσουμε» την τρισδιάστατη της δομή.



Σχήμα 1.2: *Αριστερά: Η πρωτοταγής δομή των πρωτεϊνών είναι η αλληλουχία των αμινοξέων που την συνθέτουν. Δεξιά: Η δευτεροταγής δομή είναι οι τοπικές διαμορφώσεις της πολυπεπτιδικής αλυσίδας. (Από Mount, 2001)*

Η ***δευτεροταγής*** δομή μιας πρωτεΐνης περιγράφει κατά κύριο λόγο τοπικές κανονικές διαμορφώσεις της πολυπεπτιδικής αλυσίδας στην τρισδιάστατη δομή της (σχήμα 1.2 δεξιά). Οι τοπικές αυτές διαμορφώσεις μπορούν να διαχωριστούν κατά κύριο λόγο σε α-έλικες (α-helices) και εκτεταμένους β-κλώνους (β-strands) οι οποίοι συχνά σχηματίζουν β-πτυχωτές επιφάνειες (Αγαθοκλέους 2009), όπως φαίνεται από το σχήμα 1.3. Αυτό γίνεται γιατί όπως προαναφέρθηκε στο υποκεφάλαιο 1.1.1, τα αμινοξέα έχουν πολλές φυσικοχημικές ιδιότητες οι οποίες τα οδηγούν σε αλληλεπιδράσεις με άλλα αμινοξέα της αλυσίδας. Η διαφορά με την τριτοταγή δομή είναι ότι η ***τριτοταγής*** δομή μπορεί να θεωρηθεί σαν η σύνθεση αυτών των τοπικών διαμορφώσεων, και η συμπερίληψη όλων των δομικών λεπτομερειών που περιγράφουν λεπτομερώς την τελική μορφή της πρωτεΐνης στον χώρο. Όταν περισσότερες από μια τριτοταγείς δομές συνενωθούν μεταξύ τους, τότε αναφερόμαστε στην ***τεταρτοταγή*** δομή των πρωτεϊνών σε πρωτεϊνικά σύμπλοκα.



Σχήμα 1.3: *Η δευτεροταγής δομή των πρωτεϊνών. Αριστερά φαίνεται η τοπική διαμόρφωση που αναπαριστά την ομάδα των Helix, και δεξιά η τοπική διαμόρφωση που αναπαριστά την ομάδα των Strands (Από Mount, 2001).*

1.1.3 Ρόλος πρωτεϊνών

Η τριτοταγής δομή κάθε πρωτεΐνης είναι πολύ σημαντική επειδή καθορίζει την λειτουργικότητά της. Ο ρόλος των πρωτεϊνών είναι πολυδιάστατος. Σχεδόν το σύνολο των λειτουργιών ενός κυττάρου οφείλεται στις πρωτεΐνες. Ο πιο γνωστός ρόλος τους είναι ότι λειτουργούν ως ένζυμα μέσα στα διάφορα κύτταρα. Τα ένζυμα λειτουργούν σαν *καταλύτες* οι οποίοι κατευθύνουν και επιταχύνουν διάφορες χημικές αντιδράσεις, βασικές και αναγκαίες λειτουργίες ενός οργανισμού (πχ μεταβολισμός). Επίσης, συχνά οι πρωτεΐνες έχουν δομικό ρόλο, δηλαδή συμβάλλουν στη διαμόρφωση του σχήματος και της δομής του κυττάρου και των ιστών. Ακόμη, υπάρχουν ρυθμιστικές πρωτεΐνες, οι οποίες μπορούν να τροποποιούν την έκφραση της γενετικής πληροφορίας του DNA ανάλογα με τις περιστάσεις και τις ανάγκες του κυττάρου. Οι σημαντικές λειτουργίες των πρωτεϊνών είναι

αποτέλεσμα της τρισδιάστατης δομής τους και για το λόγο αυτό αποτελούν, εκτός των άλλων, μόρια με μεγάλο βιοτεχνολογικό ενδιαφέρον (Mount, 2001).

1.1.4 Πρόβλημα *protein folding*

Από τα παραπάνω είναι εμφανές ότι η γνώση της τρισδιάστατης δομής των πρωτεϊνών σε ατομική λεπτομέρεια μπορεί (α) να δώσει σημαντικές πληροφορίες για τον τρόπο δράσης της και (β) να καθοδηγήσει την επινόηση νέων πρωτεϊνών με επιθυμητές ιδιότητες. Ανάλογα με τον τρόπο που αναδιπλώνεται στο χώρο η κάθε πρωτεΐνη προσδίδει και την λειτουργικότητα τους. Πολλά πειράματα και έρευνες γίνονται πάνω στη μελέτη της τρισδιάστατης δομής τους έτσι ώστε να *διαφανεί* η λειτουργικότητα της πρωτεΐνης και να χρησιμοποιηθεί σε πολλούς τομείς. Τέτοιες μέθοδοι είναι η κρυσταλλογραφία ακτίνων–Χ και η μέθοδος πυρηνικού μαγνητικού συντονισμού *Nuclear Magnetic Resonance* (NMR). Αξίζει να σημειωθεί όμως, ότι αυτές οι διαδικασίες είναι πολύπλοκες, δαπανηρές και χρονοβόρες ενώ παράλληλα κάθε μέρα ανακαλύπτονται και καινούριες πρωτεΐνες. Κατά συνέπεια, από το τεράστιο πλήθος πρωτεϊνικών ακολουθιών που γνωρίζουμε, μόνο για ένα πολύ μικρό ποσοστό έχουμε πειραματικά προσδιορισμένη τη δομή τους.

Με δεδομένο ότι η πρωτεϊνική δομή κωδικοποιείται στην ακολουθία των αμινοξέων στην πολυπεπτιδική αλυσίδα, αναμένεται ότι είναι δυνατόν να προβλέψουμε το δίπλωμα της πρωτεΐνης με βάση την πρωτοταγή δομή και μόνο. Μια μέθοδος που μπορεί να ακολουθηθεί για την μετάβαση από την πρωτοταγή δομή στην τριτοταγή δομή μιας ακολουθίας, ώστε να έχουμε στα χέρια μας την πολύ χρήσιμη λειτουργικότητα της πρωτεΐνης, είναι μέσω της δευτεροταγούς δομής. Η σωστή πρόβλεψη της δευτεροταγούς δομής, μπορεί εύκολα να μας οδηγήσει στη τριτοταγή δομή, αφού επιβάλλει τοπικούς δομικούς περιορισμούς. Έχουν γίνει και συνεχίζονται να γίνονται πολλές ερευνητικές προσπάθειες από διαφόρους επιστημονικούς κλάδους για τη σωστή πρόβλεψη της δευτεροταγούς δομής. Παρά τη πρόοδο που σημειώνεται τις τελευταίες δεκαετίες, το πρόβλημα αυτό εξακολουθεί να υπάρχει και να παραμένει άλυτο.

Στόχος αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μεθοδολογίας για την πρόγνωση της δευτεροταγούς δομής πρωτεϊνών από την αμινοξική τους ακολουθία χρησιμοποιώντας διαφορές αλγοριθμικές μεθόδους .

1.2 Σχετική έρευνα

Κατά την διάρκεια των 40 τελευταίων χρόνων, αναπτύχθηκαν πολλές μέθοδοι μελέτης και πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών.

Σε μία πρώτη προσπάθεια χρησιμοποιήθηκαν οι στατιστικές μέθοδοι για τη πρόβλεψη της δευτεροταγούς δομής πρωτεϊνών. Οι μέθοδοι αυτοί είναι βασισμένες σε στατιστικούς κανόνες και αναπτύχθηκαν κυρίως από τους Chou and Fasman (Chou and Fasman, 1974) πρώτα και βελτιώθηκαν στη συνέχεια από με τη μέθοδο GOR (Garnier – Osguthorpe – Robson) (Garnier et al., 1978). Οι κανόνες αυτοί εκφράζουν την πιθανότητα για κάθε αμινοξύ να αποτελεί μέρος κάποιας δευτεροταγούς δομής και προσδιορίζονται μέσα από στατιστικές αναλύσεις χρησιμοποιώντας πρωτεΐνες γνωστής τριτοταγούς δομής (Zvelebil and Baum, 2008). Οι Chou και Fasman ανάλυσαν τη συχνότητα του κάθε αμινοξέως σε δευτεροταγείς δομές (α-helices, β-sheets και turns) και στη συνέχεια κατασκεύασαν ένα πίνακα πιθανοτήτων για την εμφάνιση του κάθε αμινοξέως σε κάθε τύπο δευτεροταγούς δομής. Αυτές οι πιθανότητες χρησιμοποιήθηκαν για να προβλεφθεί η δευτεροταγής δομής, δεδομένης μιας αμινοξικής ακολουθίας. Η μέθοδος αυτή είναι περίπου 50% – 60% ακριβής με βάση το σκορ Q3 στην πρόβλεψη της δευτεροταγούς δομής μιας άγνωστης πρωτεΐνης.

Εν συνέχεια, όπως προαναφέρθηκε, αναπτύχθηκε η στατιστική μέθοδος GOR. Η μέθοδος αυτή είναι παρόμοια με αυτή που ανάπτυξαν οι Chou and Fasman, αλλά είναι πιο ακριβής γιατί υπολογίζει τις πιθανότητες για κάθε αμινοξύ λαμβάνοντας υπόψη και την πιθανότητα το αμινοξύ να δώσει μια συγκεκριμένη δευτεροταγή δομή, δεδομένου των γειτονικών του αμινοξέων. Χρησιμοποιήθηκε γι’ αυτό το σκοπό ένα παράθυρο 17 αμινοξέων (Zvelebil and Baum, 2008). Η ακρίβεια πρόβλεψης της δευτεροταγούς δομής με την μέθοδο αυτή όσον αφορά το Q3 σκορ είναι γύρω στο 65%. Να σημειωθεί ότι το αρχικό GOR σύστημα έχει βελτιωθεί,

συνεπώς αναπτύχθηκαν νέες επεκτάσεις του συστήματος με αποτελέσματα για SOV 70.7% και Q3 μέχρι και 73.5% (Zvelebil and Baum, 2008).

Οι μέθοδοι που παρουσιάστηκαν αργότερα, ήταν σε γενικό επίπεδο στατιστικές μέθοδοι, αλλά πιο βελτιωμένες από τις προηγούμενες, αφού λάμβαναν υπόψη τους επιπρόσθετες πληροφορίες για την πρωτεΐνη. Οι πληροφορίες αυτές αφορούσαν κυρίως σχετική γνώση για την δομή της πρωτεΐνης, όπως το σχήμα και το μέγεθος της, αλλά και τις φυσικοχημικές ιδιότητες του κάθε αμινοξέως ξεχωριστά. Μια αντιπροσωπευτική μέθοδος που ανήκει σε αυτήν την κατηγορία είναι το PREDATOR (Frishman and Argos, 1996). Το PREDATOR δεν λαμβάνει υπόψη τις τοπικές αλληλεπιδράσεις όπως κάνει το GOR, αλλά, λαμβάνει υπόψη μεγάλα μήκη της ακολουθίας, άρα μεγάλες αλληλεπιδράσεις των αμινοξέων. Συγκρίνει ζευγάρια αμινοξέων ώστε α) να βρει γειτονικά αμινοξέα που αλληλεπιδρούν μεταξύ τους με υδρογονικούς δεσμούς για πρόβλεψη β-strands, και b) υδρογονικούς δεσμούς μεταξύ αμινοξέων i και i+4 για helices (Zvelebil and Baum, 2008). Το PREDATOR χρησιμοποιεί ένα σύνολο από ομόλογες ακολουθίες, και όχι ακολουθιών πολλαπλής στοίχισης. Παρόλα αυτά το PREDATOR μπορεί να δώσει ποσοστά Q3 ακρίβειας μέχρι και 75% (Frishman and Argos, 1997).

Οι πιο σημαντικές όμως μέθοδοι που χρησιμοποιήθηκαν ήταν τα ΤΝΔ (Υποκεφάλαιο 1.3), Support Vector Machines και HMM (Zvelebil and Baum, 2008). Οι μέθοδοι αυτοί ανήκουν στις υπολογιστικές και όχι τις στατιστικές μεθόδους. Τα ΤΝΔ χρησιμοποιούν ένα σύνολο δεδομένων με τα οποία εκπαιδεύονται, στα οποία παρουσιάζονται ακολουθίες γνωστών πρωτεϊνών, καθώς και η δευτεροταγής δομή τους. Σε αντίθεση με τις στατιστικές μεθόδους, κατά την διάρκεια της εκπαίδευσης τους, *μαθαίνουν* την αντιστοιχία της πρωτοταγούς με δευτεροταγούς δομής. Αυτό γίνεται προσαρμόζοντας κάθε φορά τα συναπτικά βάρη τους κατάλληλα, ώστε να μειώνεται το ολικό σφάλμα πρόβλεψης του ΤΝΔ. Αυτές οι μέθοδοι παρουσιάζουν σημαντικά αποτελέσματα σε σύγκριση με τις προηγούμενες μεθόδους.

Τέτοιες μέθοδοι είναι για παράδειγμα ένα δίκτυο εμπρόσθιου περάσματος και ενός κρυφού επιπέδου (Qian and Sejnowski, 1988), το οποίο χρησιμοποιεί ένα παράθυρο εισόδου συνήθως 13 αμινοξέων και προβλέπει για κάθε κεντρικό αμινοξύ την

κατηγορία του (Helix, Strand ή Coil). Στο δίκτυο αυτό χρησιμοποιήθηκε και ένα δεύτερο το οποίο διόρθωνε τα αποτελέσματα του πρώτου δικτύου. Η μέθοδος αυτή αντιμετώπιζε προβλήματα υπερεκπαίδευσης.

Το PHD δίκτυο (Rost and Sander, 1993), ακολουθεί το ίδιο σχεδιασμό δικτύου όπως την προηγούμενη μέθοδο, απλά εφαρμόζονται κάποιες τεχνικές για επίλυση του προβλήματος της υπερεκπαίδευσης. Χρησιμοποιήθηκαν επίσης τεχνικές όπως η πολλαπλή στοίχιση ακολουθιών.

Το NNSSP (Salamov and Soloveyev, 1995) είναι ένα ΤΝΔ που χρησιμοποιεί την μέθοδο του κοντινότερου γείτονα, για να ομαδοποιήσει ακολουθίες βάσει της ομοιότητας τους και να τις συγκρίνει με άλλες ακολουθίες γνωστής δευτεροταγούς δομής. Το ποσοστό ακρίβειας της μεθόδου αυτής είναι μέχρι και 68% (Q3), ενώ χρησιμοποιώντας πολλαπλή στοίχιση ακολουθιών τα αποτελέσματα βελτιώνονται σε 72.2%. Επιπρόσθετη βελτίωση του NNSSP αλγορίθμου ήταν το γεγονός ότι χρησιμοποιήθηκαν διαφορετικές πρωτεΐνες για εκπαίδευση και διαφορετικές πρωτεΐνες για επαλήθευση, ούτως ώστε να μην επικαλύπτονται μεταξύ τους πρωτεΐνες από το σύνολο εκπαίδευσης και από το σύνολο επαλήθευσης. Το αποτέλεσμα σε αυτήν την περίπτωση ήταν 73.5% (Salamov and Soloveyev, 1997).

Ο αλγόριθμος DSC (Discrimination of protein Secondary structure Class) (King and Sternberg, 1996), ομαδοποιεί σε διάφορες κατηγορίες τα αποτελέσματα εξόδου του δικτύου και προσπαθεί με απλές γραμμικές και στατιστικές μεθόδους να προσεγγίσει την ακριβή δευτεροταγή δομή των πρωτεϊνών. Το ποσοστό ακρίβειας του χρησιμοποιώντας το Q3 σκορ είναι 71.95%.

Το BRNN (Bidirectional Recurrent Neural Network) (Baldi et al., 1999, 2000), είναι ΤΝΔ το οποίο είχε την καλύτερη απόδοση στην πρόβλεψη δευτεροταγούς δομής πρωτεϊνών. Το ΤΝΔ δέχεται μια ακολουθία από αμινοξέα μέσω ενός κινητού παραθύρου και προσπαθεί να προβλέψει την δευτεροταγή δομή του κεντρικού αμινοξέως. Η σημασία του σχεδιασμού του δικτύου αυτού είναι μεγάλη αφού λαμβάνει υπόψη τα αμινοξέα που προηγούνται και έπονται του κεντρικού αμινοξέως χρησιμοποιώντας ΤΝΔ με αμφίδρομη ανάδραση. Ο Baldi και οι

συνεργάτες του επιτυγχάνουν ακρίβεια ίση με 73.6% για το Q3 σκορ με την χρήση της πολλαπλής στοίχισης ακολουθιών στο επίπεδο εισόδου. Χρησιμοποιώντας όμως ένα συνδυασμό από έξι τέτοια δίκτυα το Q3 ποσοστό αυτό αυξάνεται σε 76%.

Πιο πρόσφατες μέθοδοι είναι το LAD (Blacewicz et al., 2005), όπου υλοποιεί ένα αλγόριθμο μηχανικής μάθησης, όπου μελετά και λαμβάνει υπόψη τις ιδιότητες και την δομή των αμινοξέων. Η μέθοδος αυτή είχε καλά αποτελέσματα (Q3 με 70.6%) και οδήγησε σε συμπεράσματα για τις σχέσεις των αμινοξέων μεταξύ τους, και πως επηρεάζεται η πρόβλεψη της δευτεροταγούς δομής βάση των ιδιοτήτων αυτών

Αξιοσημείωτη είναι και συνεισφορά των Γενετικών Αλγορίθμων στη επίλυση του προβλήματος. Οι KJ Won, T Hamelryck (2005) σε μια προσπάθεια να αυξήσουν το ποσοστό επιτυχίας εκπαίδευσαν τα Hidden Markov Models (HMM) ενσωματώνοντας τους Γενετικού Αλγορίθμους Το ποσοστό επιτυχίας που πέτυχαν σε Q3 ήταν περίπου 75% .

Από το 2001, νέες μέθοδοι με *Support Vector Machines* (SVM) (Ward et al., 2003) (Kim and Park, 2003), εφαρμόστηκαν για την πρόβλεψη της δευτεροταγούς δομής πρωτεϊνών.

Σημαντική μέθοδος πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, είναι η πρόβλεψη των δίεδρων γωνιών σε συνδυασμό με την πρόβλεψη της δευτεροταγούς δομής. Οι δίεδρες γωνίες, είναι οι γωνίες της κοινής *ραχοκοκαλιάς* από την οποία αποτελούνται τα αμινοξέα, είναι άμεσα συσχετιζόμενες με την δευτεροταγή δομή τους και συνεπώς μπορούν να δώσουν σημαντικές πληροφορίες για την τριτοταγή δομή της πρωτεΐνης. Οι Kountouris and Hirst (Kountouris and Hirst, 2009), εφαρμόζουν την πιο πάνω ιδέα: προβλέπουν ξεχωριστά και την δευτεροταγή δομή της πρωτοταγούς ακολουθίας, αλλά και τις δίεδρες γωνίες της ραχοκοκαλιάς των αμινοξέων με την χρήση δύο μοντέλων *Support Vector Machines* (SVM). Στο πρώτο μοντέλο, προβλέπεται η επιλογή της κατηγορίας στην οποία ανήκει το κάθε αμινοξύ (H, E, L) και παρόμοια στο δεύτερο μοντέλο προβλέπεται η κατηγορία δίεδρης γωνίας στην οποία ανήκει. Σε κάθε εκτέλεση, τα αποτελέσματα κάθε μοντέλου χρησιμοποιούνται για να *αυξήσουν* την είσοδο του άλλου μοντέλου για την επόμενη εκτέλεση. Η μέθοδος αυτή δίνει ένα Q3 ποσοστό 80%.

Το *Porter,* είναι ένα νέο σύστημα πρόβλεψης δευτεροταγούς δομής πρωτεινών με μεγάλη ακρίβεια (Pollastri and McLysaght, 2004). Είναι βασισμένο σε αρχιτεκτονική ΤΝΔ με αμφίδρομη ανάδραση (BRNN), εκπαιδεύεται με δεδομένα από την πολλαπλή στοίχιση ακολουθιών και εφαρμόζει *φιλτράρισμα* των προβλεπόμενων ακολουθιών του δικτύου, χρησιμοποιώντας ΤΝΔ με ανάδραση. Τέλος, για να πάρει την τελική προβλεπόμενη ακολουθία, εκπαιδεύει ένα σύνολο από τέτοια δίκτυα (BRNN), ξεχωριστά όπου η τελική ακολουθία υπολογίζεται από τον μέσο όρο τους. Το Q3 ποσοστό ακρίβειας του *Porter*, είναι 79%.

Μια από τις πιο πρόσφατες υλοποιήσεις ΤΝΔ, αφορά την δημιουργία δυο επιπέδων ΤΝΔ (Cascaded Bidirectional Recurrent Neural Network) (Chen and Chaudhari, 2007). Με δυο επίπεδα ΤΝΔ, συμπεριλαμβάνουν υπόψη τους τις *μακρινές* αλληλεπιδράσεις μεταξύ των αμινοξέων, αφού παίζουν σημαντικό ρόλο στην αναδίπλωση της πρωτεΐνης. Με τέτοια αρχιτεκτονική τα αποτελέσματα του πρώτου ΤΝΔ (sequence to structure) είναι η είσοδος του δεύτερου ΤΝΔ (structure to structure) με αποτέλεσμα το δεύτερο δίκτυο να *φιλτράρει* τα δεδομένα εξόδου του πρώτου δικτύου, καταλήγοντας σε πιο ακριβή αποτελέσματα. Τα Q3 ποσοστά φτάνουν μέχρι και 74.38% ενώ το SOV Score (Υποκεφάλαιο 3.4.2) είναι πιο ακριβές σε σύγκριση με άλλες μεθόδους (66,0).

1.3 Νευρωνικά Δίκτυα

Τα Τεχνητά Νευρωνικά Δίκτυα αποτελούν κομμάτι του κλάδου της Πληροφορικής τα οποία εφαρμόζουν θεωρίες του τομέα της Τεχνητής Νοημοσύνης και τον Υπολογιστικών Συστημάτων. Κύριος σκοπός των ΤΝΔ είναι να προβλέψουν ορθά το αποτέλεσμα μιας μαθηματικής συνάρτησης. Μετά τη δεκαετία του 80 υπήρξε μεγάλη ανάπτυξη των ΤΝΔ και μέχρι σήμερα εφαρμόζονται με μεγάλη επιτυχία σε πολλά και διάφορα πρακτικά προβλήματα. Αυτή η μεγάλη απήχηση των ΤΝΔ οφείλεται κυρίως στο γεγονός ότι μοντελοποιεί τον τρόπο λειτουργίας του εγκεφάλου.

Ο ανθρώπινος εγκέφαλος αποτελεί ένα από τα μεγαλύτερα μυστήρια και περιβάλλεται από ένα μεγάλο ερωτηματικό ως προς τον τρόπο λειτουργίας του. Πολλοί μελετητές και πειραματιστές ασχολήθηκαν και προσπάθησαν να κατανοήσουν τον τρόπο που σκεφτόμαστε. Έχει γίνει μεγάλη πρόοδος χωρίς όμως να καταλάβουμε πλήρως τον τρόπο λειτουργίας του. Ο ανθρώπινος εγκέφαλος, όπως γνωρίζουμε πλέον, αποτελείται από εκατομμύρια νευρώνες, οι οποίοι είναι και τα βασικά κύτταρα του εγκεφάλου. Κάθε βιολογικός νευρώνας αποτελείται από το σώμα, το πυρήνα, τον άξονα, τους δενδρίτες και τις συνάψεις. Κάθε βιολογικός νευρώνας επικοινωνεί με άλλους νευρώνες μέσω των συνάψεων. Αφού υπάρξει κάποιο ερέθισμα, πολλά συναπτικά δυναμικά σήματα, (διεγερτικά ή ανασταλτικά), φτάνουν στους δενδρίτες των νευρώνων. Ο βιολογικός νευρώνας παίρνει τα σήματα αυτά, και τα επεξεργάζεται μέσα στο σώμα του. Για να τα επεξεργαστεί, πρέπει να τα συναθροίσει στον χώρο και στον χρόνο, ώστε να πάρει κάποια αντιπροσωπευτική τιμή για το σύνολό τους. Εάν η τιμή αυτή είναι ικανή να ξεπεράσει το κατώφλι του νευρώνα, τότε δημιουργείται το δυναμικό ενεργείας, δηλαδή ο νευρώνας πυροβολεί. Εάν όμως η τιμή δεν καταφέρει να ξεπεράσει την τιμή κατωφλίου του νευρώνα, τότε δεν πυροβολεί. Στην συνέχεια, η πληροφορία του νευρώνα μεταδίδεται σε άλλους νευρώνες μέσα από τις συνάψεις. Όταν η πληροφορία φτάσει στις απολήξεις ενός νευροάξονα, προκαλεί την απελευθέρωση μιας χημικής ουσίας (νευροδιαβιβαστής) στη συναπτική σχισμή. Οι νευροδιαβιβαστές μετακινούνται διά μέσου της συναπτικής σχισμής στο επόμενο νευρώνα όπου αναγνωρίζεται και δεσμεύεται από ειδικευμένους υποδοχείς της

εξωτερικής επιφάνειας της κυτταρικής μεμβράνης. Οι συνάψεις αποτελούν το πιο σημαντικό σημείο της επικοινωνίας των νευρώνων, γιατί ουσιαστικά η σύναψη περιέχει τη πληροφορία που μπορεί να αναγνωρίσει ο μετασυναπτικός νευρώνας. Ο αριθμός των νευρώνων στων ανθρώπινων εγκέφαλο είναι σταθερός και δημιουργείται στον άνθρωπό από τη γέννηση του και είναι περίπου 100 δισεκατομμύρια νευρωνικά κύτταρα. Αυτό που αλλάζει στη πορεία του ανθρώπου στη ζωή είναι οι συνδέσεις μεταξύ των νευρώνων. Οι συνδέσεις επιτυγχάνονται μέσα από τις εμπειρίες του και τη μάθηση που αποκτά. Η δύναμη των συνάψεων αυτών, τροποποιείται συνεχώς με την μάθηση, γι αυτό και η κατάλληλη μάθηση μπορεί να προσδιορίσει τα κατάλληλα βάρη των συνάψεων ώστε να περνούν ή όχι συγκεκριμένες πληροφορίες. Πολλοί νευρώνες μαζί δημιουργούν ένα νευρωνικό δίκτυο.

Τα ΤΝΔ αποτελούν ένα υπολογιστικό μοντέλο που προσπαθεί να προσομοιάσει κατά κάποιο τρόπο, τον τρόπο λειτουργίας του εγκεφάλου. Το μεγάλο πλεονέκτημα των ΤΝΔ είναι ότι όπως στον ανθρώπινο εγκέφαλο, διάφορες λειτουργίες εκτελούνται παράλληλα και άρα μπορεί να αντιμετωπίσει τα διάφορα προβλήματα πολύ γρήγορα και με πολύ εξειδικευμένο τρόπο Με τον τρόπο αυτό μπορούμε να λύσουμε προβλήματα μεγάλου μεγέθους, ασαφή και πολύπλοκα σε αντίθεση με τον κλασσικό προγραμματισμό όπου η σειριακή εκτέλεση των υπολογιστικών κανόνων δεν επιτρέπει την γρήγορη και αποδοτική επίλυση το προβλημάτων.

Ακόμη ένα μεγάλο πλεονέκτημα των ΤΝΔ είναι το γεγονός ότι μέσα από τη πληροφορία που το δίνεται το δίκτυο μαθαίνει και προσαρμόζεται ανάλογα και μέσα από τη μάθηση μειώνει το λάθος. Αυτό το κάνει να ευέλικτο, ανεκτικό σε λάθη και του προσδίδει την ικανότητα να γενικεύει, αφού εκπαιδευτεί σε κάποιο σύνολο δεδομένων.

Όπως φαίνεται το πιο σημαντικό κομμάτι των ΤΝΔ είναι η *μάθηση*. Μέσα από τη μάθηση το δίκτυο εκπαιδεύεται και μαθαίνει να προβλέπει σωστά το αποτέλεσμα μίας άγνωστης προς αυτό ακολουθίας δεδομένων. Υπάρχουν τρία είδη μάθησης και συνεπώς τρία διαφορετικά είδη ΤΝΔ. Οι τρεις αυτές κατηγορίες είναι η επιβλεπόμενη, η μη επιβλεπόμενη και η ενισχυτική μάθηση.

Στην *επιβλεπόμενη* μάθηση έχουμε το ρόλο ενός δασκάλου. Ο δάσκαλος είναι υπεύθυνος να εκπαιδεύσει το σύστημα πάνω σε ένα σύνολο δεδομένων (training set) όπου η επιθυμητή έξοδος (αποτέλεσμα δικτύου) είναι γνωστή ούτως ώστε να μάθει, αλλά επίσης και σε κάθε λάθος που κάνει, τον διορθώνει λέγοντας του ποια είναι η σωστή απάντηση. Όταν τελειώσει η φάση της εκπαίδευσης του δίκτυο, δίνεται επίσης ένα σύνολο δεδομένων επαλήθευσης, το οποίο αποτελείται από πληροφορίες που δεν έχει ξαναδεί το σύστημα, αλλά που έχει μάθει παρόμοιές τους από την φάση εκπαίδευσης του. Με αυτό το σύνολο θα αξιολογούμε την ικανότητα μάθησης του δικτύου (γενίκευση).

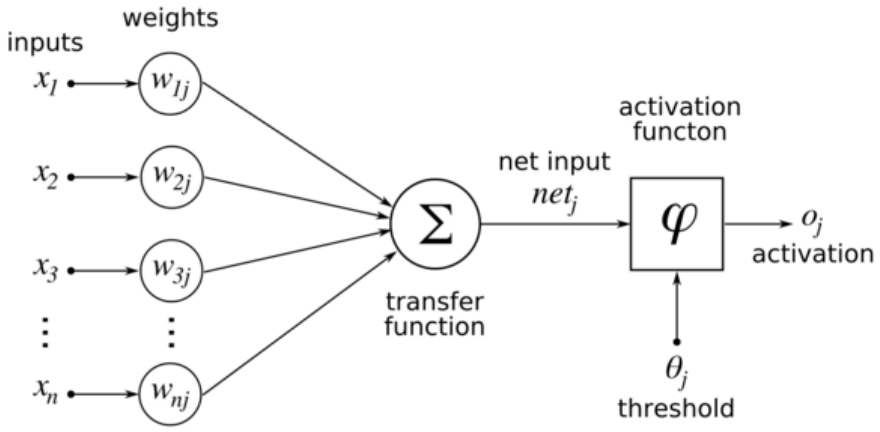
Στην ενισχυτική μάθηση δεν έχουμε δάσκαλο αλλά κριτή, ο οποίος κρίνει τις επιλογές του δικτύου για μια σειρά δεδομένων που του δίνονται. Δηλαδή σε κάθε ενέργεια του το δίκτυο ανταμείβεται αν πήρε τη σωστή απόφαση και τιμωρείται αν πήρε τη λάθος απόφαση. Με αυτό τον τρόπο το δίκτυο μαθαίνει στην ουσία μέσα από τη ανταμοιβή, ποιες είναι οι σωστές και ποιες λάθος ενέργειες στην επίλυση ενός συγκεκριμένου προβλήματος.

Στην μη επιβλεπόμενη μάθηση δεν έχουμε ούτε το ρόλο του δάσκαλου αλλά ούτε το ρόλο του κριτή. Επίσης η επιθυμητή έξοδος του δικτύου δεν είναι γνώστη. Στόχος αυτής της μάθησης είναι να ομαδοποιήσει τα δεδομένα εισόδου που του δίνονται με βάση τα κοινά τους χαρακτηριστικά. Χρησιμοποιείται κυρίως για προβλήματα κατηγοριοποίησης, όπου δεδομένα με παρόμοιες ιδιότητες τοποθετούνται μαζί δημιουργώντας ομάδες.

Στη συγκεκριμένη διπλωματική εργασία θα ασχοληθούμε κυρίως με τη επιβλεπόμενη μάθηση, αφού θα χρησιμοποιήσουμε ένα Bidirectional Recurrent Neural Network (Υποκεφάλαιο 2.1) για εκπαίδευση και δίκτυο Radial Basis Function (Υποκεφάλαιο 4.3.2) για φιλτράρισμα, όπως επίσης και κατά ένα μέρος με τη μη επιβλεπόμενη στα πρώτο στάδιο του φιλτραρίσματος όπου θα χρησιμοποιήσουμε χάρτη αυτοδιοργάνωσης Kohonen (Υποκεφάλαιο 4.3.1).

1.3.1 Αρχιτεκτονική Νευρωνικού Δικτύου Επιβλεπόμενη Μάθησης

Μια αρχική μοντελοποίηση ενός βιολογικού νευρώνα, η οποία προσομοιώνει τον τρόπο λειτουργίας του με τον τρόπο που εξηγήθηκε στην προηγούμενη σελίδα, ώστε να δημιουργούν δίκτυα νευρώνων, τα οποία είναι ικανά να μάθουν και να γενικεύσουν πληροφορίες παρουσιάστηκε από τους McCulloch & Pitts (McCulloch and Pitts, 1943), οι οποίοι παρουσίασαν ένα τυπικό μοντέλο ενός τεχνητού νευρώνα, όπως φαίνεται και από το σχήμα 1.4.



Σχήμα 1.4: *Το μοντέλο τεχνητού νευρώνα McCulloch & Pitts (Από McCulloch and Pitts, 1943).*

Οι διάφορες είσοδοι του νευρώνα παίρνουν τις πληροφορίες εισόδου. Κάθε είσοδος του νευρώνα, αντιστοιχίζεται με κάποιο συναπτικό βάρος, δηλαδή μια τιμή, η οποία αντιπροσωπεύει την δύναμη της σύναψης ενός βιολογικού νευρώνα. Αφού ο νευρώνας πάρει όλες τις εισόδους, τότε αυτές συναθροίζονται μέσα στο σώμα του νευρώνα με βάση την εξίσωση 1.1, και ανάλογα με το βάρος που έχει η κάθε είσοδος. Με τέτοιο τρόπο μοντελοποιείται η χρονική συνάθροιση *ανασταλτικών* και *διεγερτικών* σημάτων που έρχονται σε ένα βιολογικό νευρώνα.

Η εκτίμηση των τιμών των συναπτικών βαρών ενός ΤΝΔ ουσιαστικά δίνει ένα βάρος με το οποίο θα ληφθεί υπόψη η συγκεκριμένη είσοδος στην χρονική συνάθροιση των σημάτων, δηλαδή μοντελοποιεί την έννοια των διεγερτικών και ανασταλτικών

σημάτων. Η *επιλεκτικότητα* αυτή του νευρώνα, είναι χαρακτηριστικό των βιολογικών νευρώνων και μέσω αυτής επιτυγχάνεται μάθηση.

$$u = \sum_{i=1}^{n} w_i x_i$$

Εξίσωση 1.1: *Εξίσωση εισόδου στον τεχνητό νευρώνα, όπου x_i η τιμή εισόδου και w_i η τιμή του βάρους του νευρώνα i. Η εξίσωση αυτή αντιστοιχεί στη transfer functiontου σχήματος 1.4.*

Όμως, αφότου γίνει η συνάθροιση, πρέπει να ελεγχθεί κατά πόσο η τιμή αυτή ξεπερνά ή όχι το κατώφλι του νευρώνα, ώστε να πυροβολήσει ή όχι αντίστοιχα. Αυτό μοντελοποιείται με την βοήθεια μιας συνάρτησης ενεργοποίηση.

$$y = \begin{cases} 1 & \text{if } u \geq \theta \\ 0 & \text{if } u < \theta \end{cases}$$

Εξίσωση 1.2: *Η συνάρτηση κατωφλίου, όπου ϑ η τιμή του κατωφλίου και u η έξοδος του τεχνητού νευρώνα.*

Η πιο απλή συνάρτηση ενεργοποίησης είναι η συνάρτηση κατωφλίου, που ρυθμίζει την έξοδο ενός νευρώνα ανάλογα με ένα κατώφλι θ (Εξίσωση 1.2). Αν η τιμή u είναι πάνω από τη τιμή θ, τότε ο νευρώνας πυροβολεί, αν όχι τότε δεν πυροβολεί. Όμως η πιο πάνω συνάρτηση παρουσίασε πολλά σοβαρά προβλήματα χρήσης με πιο σημαντικό το γεγονός ότι δεν μπορεί να λυσει μη γραμμικά προβλήματα. Εναλλακτικά χρησιμοποιούμε συνήθως την σιγμοειδή συνάρτηση, ή κάποια άλλη παραγωγίσιμη συνάρτηση φ για ενεργοποίηση του νευρώνα (Εξίσωση 1.3).

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i\right)$$

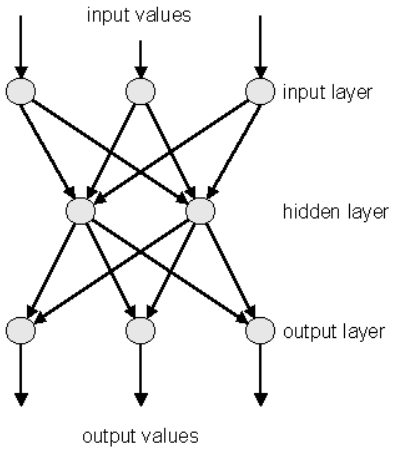
Εξίσωση 1.3: *Το αποτέλεσμα εξόδου ενός νευρώνα, όπου φ η παραγωγίσιμη συνάρτηση ενεργοποίησης.*

Για τον σχηματισμό ενός νευρωνικού δικτύου, ενώνουμε τεχνητούς νευρώνες τύπου McCulloch and Pitts (ή κάποιο άλλο μοντέλο νευρώνα) μεταξύ τους, ώστε να σχηματιστούν επίπεδα από νευρώνες, όπου ο κάθε νευρώνας έχει τις δικές του εισόδους, τα δικά του βάρη και την δική του έξοδο. Τέτοια δίκτυα είναι τα *perceptrons* και υλοποιούνται με μη γραμμικούς νευρώνες τύπου McCulloch and Pitts. Υπάρχουν τρία είδη αρχιτεκτονικών ΤΝΔ που μπορούν να σχηματιστούν και η κάθε αρχιτεκτονική μπορεί να χρησιμοποιηθεί σε διαφορετικά προβλήματα, ανάλογα με την φύση του προβλήματος, τις εισόδους και τις εξόδους του.

Οι νευρώνες των ΤΝΔ, όπως έχουμε προαναφέρει, είναι υπολογιστικές μονάδες, οι οποίες υπολογίζουν μια συνάρτηση εξόδου βάσει της εισόδου τους για κάποια συγκεκριμένη στιγμή. Το ΤΝΔ σαν σύνολο, είναι μια συνάθροιση των συναρτήσεων εξόδου του κάθε νευρώνα, για κάθε είσοδο του δικτύου, ώστε να μετασχηματίζει τις εισόδους σε μια συνάρτηση εξόδου η οποία λέγεται και συνάρτηση δικτύου (Rojas 1996). Η συνάρτηση αυτή μπορεί να προσεγγιστεί μέσω της εκπαίδευσης, χρησιμοποιώντας τα σύνολα εκπαίδευσης. Το πρόβλημα μάθησης, είναι το πρόβλημα της βελτιστοποίησης των βαρών μεταξύ των νευρώνων του δικτύου, ώστε να προσεγγίζουν όσο το δυνατόν καλύτερα την συνάρτηση αυτή. Έτσι, όταν κάποια είσοδος παρουσιαστεί στο δίκτυο, η οποία δεν βρισκόταν στο σύνολο εκπαίδευσης και άρα το δίκτυο δεν την έχει μάθει, πρέπει το δίκτυο να είναι ικανό να προσεγγίσει κατάλληλα την έξοδο της νέας αυτής εισόδου, χρησιμοποιώντας την συνάρτηση του δικτύου. Συγκεκριμένα, αυτό που θέλουμε να κάνουμε είναι να ελαχιστοποιήσουμε την συνάρτηση λάθους του δικτύου ελαχιστοποιώντας το σφάλμα που προκαλεί κάθε είσοδος στο δίκτυο. Συνοπτικά, όλα τα προβλήματα πρόβλεψης τα οποία καλείται να λύσει ένα ΤΝΔ, λύνονται με την πιο πάνω μέθοδο, αφού καλούνται να βελτιστοποιήσουν κάποια συνάρτηση (προβλήματα ***παλινδρόμησης, κατηγοριοποίησης)***.

1.3.2 Πολυστρωματικά δίκτυα perceptron εμπρόσθιου περάσματος

Σε ένα πολυστρωματικό ΤΝΔ εμπρόσθιου περάσματος, κάθε νευρώνας του ΤΝΔ ενώνεται μόνο με νευρώνα του επόμενου επιπέδου. Στο Σχήμα 1.5 μπορούμε να δούμε ότι κάθε νευρώνας αναπαριστάται από ένα κύκλο και κάθε ακμή συμβολίζει το βάρος του συγκεκριμένου νευρώνα. Επίσης μπορούμε να δούμε ότι υπάρχουν τρία επίπεδα, το επίπεδο εισόδου, το επίπεδο εξόδου και το κρυφό επίπεδο.



Σχήμα 1.5: *Ένα πολυστρωματικό δίκτυο perceptron εμπρόσθιου περάσματος με ένα κρυφό επίπεδο.*

Τα πολυστρωματικά δίκτυα εμπρόσθιου περάσματος αποτελούνται από το επίπεδο εισόδου, το οποίο δεν είναι ενεργό επίπεδο αφού απλά μεταφέρει τα δεδομένα εισόδου στο δίκτυο, ένα ενεργό επίπεδο εξόδου και ένα ή περισσότερα «κρυφά» επίπεδα. Να σημειωθεί ότι τα κρυφά επίπεδα είναι ενεργά επίπεδα που βρίσκονται μεταξύ των επιπέδων εξόδου και εισόδου. Σε ένα δίκτυο εμπρόσθιου περάσματος, η πληροφορία και η επεξεργασία αρχίζουν από το επίπεδο εισόδου και μεταδίδονται μπροστά σε κάθε επίπεδο του δικτύου. Άρα η είσοδος κάθε νευρώνα εξαρτάται μόνο από την έξοδο των προηγούμενων της νευρώνων, ενώ το αποτέλεσμα του δικτύου εξαρτάται από τους νευρώνες εξόδου. Η αρχιτεκτονική κάποιου ΤΝΔ εξαρτάται άμεσα από το πρόβλημα που θέλει να λύσει. Ανάλογα με τη πολυπλοκότητα του προβλήματος πρέπει να κάνουμε σωστή επιλογή αριθμών

κρυφών επιπέδων, όπως επίσης και σωστή επιλογή αριθμού νευρώνων. Μεγάλος αριθμός νευρώνων ή κρυφών επίπεδων μπορεί να προσδώσει τεράστια πολυπλοκότητα και να μην καταφέρει να λύσει το πρόβλημα. Άκρως σημαντική διαδικασία είναι η κατάλληλη προεργασία των εισόδων αυτών. Αυτό γίνεται με την κατάλληλη κωδικοποίηση και κανονικοποίηση ανάλογα με το είδος του προβλήματος το οποίο καλούμαστε να λύσουμε.

Η εκπαίδευση των πιο πάνω ΤΝΔ, μπορεί να γίνει με τον αλγόριθμο μάθησης που φαίνεται στο Σχήμα 1.6, δεδομένου ότι έχουμε επιβλεπόμενη μάθηση.

1. Αρχικοποίηση βαρών και κατωφλίου με μικρές τυχαίες τιμές.

2. Παρουσίαση εισόδου και επιθυμητού αποτελέσματος.
   Είσοδος: *x0, x1, x2, ... , xn* Επιθυμητή έξοδος: *d(t)*

3. Υπολογισμός πραγματικής τιμής εξόδου.
   $y(t) = f\,[w0(t)x0(t) + w1(t)x1(t) + w2(t)x2(t) + ... + wn(t)xn(t)]$
   όπου f() η συνάρτηση ενεργοποίησης του δίκτυο

4. Προσαρμογή των βαρών με τον κανόνα δέλτα.
   $wi(t+1) = wi(t) + n\Delta xi(t)$, *όπου* $\Delta = d(t) - y(t)$

5. Επανάληψη από το βήμα 2 ενόσω:
   a) Ο αλγόριθμος δεν έχει συγκλίνει ή
   b) Υπάρχουν ακόμη δεδομένα για εκπαίδευση.

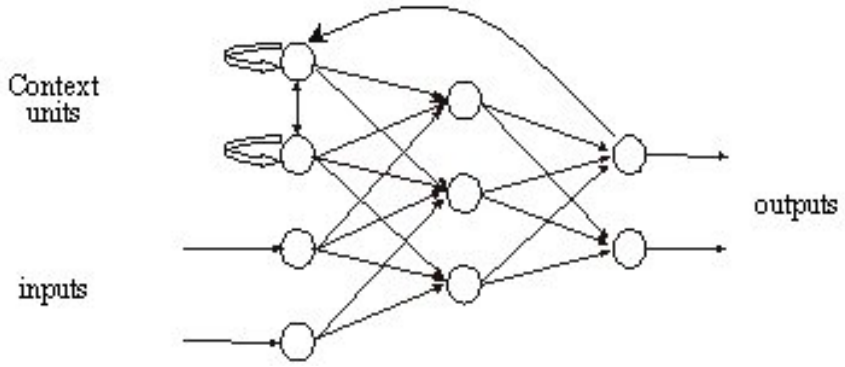
Σχήμα 1.6: *Αλγόριθμος μάθησης perceptron, όπου* x_i *η είσοδος στον νευρώνα,* w_i *το βάρος που αντιστοιχεί στην είσοδο* x_i.

1.3.3 Πολυστρωματικά δίκτυα perceptron με ανάδραση

Μια δεύτερη αρχιτεκτονική πολυεπίπεδων δικτύων perceptron είναι τα δίκτυα με ανάδραση, και διαφέρουν από τα δίκτυα εμπρόσθιου περάσματος στο γεγονός ότι υπάρχουν κάποιες συνδέσεις μεταξύ των επιπέδων του δικτύου οι οποίες δημιουργούν κατευθυνόμενους κύκλους. Τα δίκτυα με ανάδραση είναι ένα είδος νευρωνικών δικτύων ειδικά για *δυναμικές* εφαρμογές.

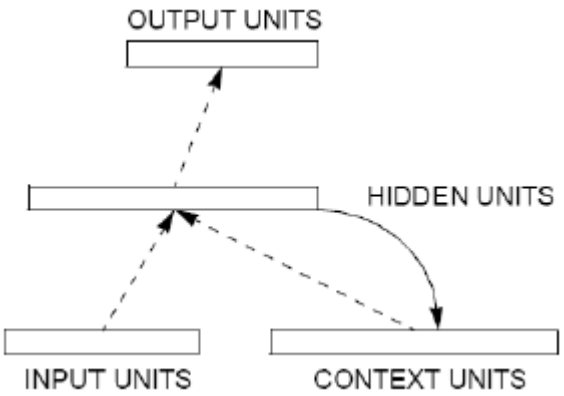
Στα δίκτυα με ανάδραση, γίνεται μοντελοποίηση της αίσθησης του *χρόνου*. Η αρχιτεκτονική τους είναι τέτοια ώστε η είσοδος του δικτύου σε κάποια χρονική στιγμή, να αποτελείται από τα χαρακτηριστικά δεδομένα εισόδου, και από τις εξόδους κάποιων νευρώνων μιας προηγούμενης χρονικής στιγμής. Άρα το δίκτυο μπορεί να μοντελοποιεί την έννοια της μνήμης, αφού κάθε νέα είσοδος επεξεργάζεται έχοντας υπόψη τα αποτελέσματα της προηγούμενης στιγμής.

Ένα παράδειγμα μιας τέτοιας τυπικής αρχιτεκτονικής δημιουργήθηκε από τον Jordan (Jordan, 1986) όπως φαίνεται και στο σχήμα 1.7. Που όμως υπάρχει η βραχυπρόθεσμη μνήμη; Η μνήμη μοντελοποιείται με την εφαρμογή των *context units*, οι οποίοι είναι υπεύθυνοι να κρατούν την πληροφορία εξόδου της προηγούμενης στιγμής, και συνεπώς είναι ίσοι με τον αριθμό των νευρώνων εξόδου του δικτύου. Φυσικά, στα context units, υπάρχει η έννοια της τοπικής ανάδρασης, δηλαδή η πληροφορία πολλαπλασιάζεται με μια χρονική σταθερά. Όσο πιο μεγάλη είναι η χρονική σταθερά, τόσο πιο πολύ οι *context units* κρατούν την συγκεκριμένη πληροφορία. Εκτός των τοπικών αναδράσεων, υπάρχουν κάποια «βάρη» στις αναδράσεις από το επίπεδο εξόδου στα context units. Τα βάρη αυτά είναι σταθερά, ώστε να μην αλλάζουν την έξοδο. Συνεπώς, στα δίκτυα Jordan η έξοδος κάποιας χρονικής στιγμής δεν εξαρτάται μόνο από την τρέχουσα είσοδο του δικτύου, αλλά και από την προηγούμενη έξοδο.



Σχήμα 1.7: *Jordan δίκτυο με ανάδραση. Παρατηρείται ότι υπάρχει η τοπική ανάδραση των context units (Από Jordan, 1986).*

Το δίκτυο με ανάδραση του Jordan, έχει ένα μειονέκτημα. Το επίπεδο εξόδου είναι περιορισμένο από τα επιθυμητά αποτελέσματα. Μια κάπως καλύτερη αρχιτεκτονική είναι η αρχιτεκτονική νευρωνικού δικτύου με ανάδραση από τον Elman (Elman, 1990), η οποία φαίνεται στο σχήμα 1.8. Σε αυτήν την αρχιτεκτονική, υπάρχει και πάλι το context layer, και η ανάδραση, αντί να ξεκινά από την έξοδο ξεκινά από το κρυφό επίπεδο και πηγαίνει πίσω στο context layer. Η ανάδραση λοιπόν γίνεται εντός δικτύου, με αποτέλεσμα οι νευρώνες εξόδου να είναι «ελεύθεροι». Όσον αφορά το ποια αρχιτεκτονική χρησιμοποιείται, έχει να κάνει με το είδος του προβλήματος και την σχεδίαση του δικτύου.



Σχήμα 1.8: *Elman δίκτυο με ανάδραση (Από Elman, 1990).*

1.3.4 Αλγόριθμος μάθησης με ανάστροφη μετάδοση λάθους

Ο αλγόριθμος μάθησης ανάστροφης μετάδοσης λάθους είναι ένας αλγόριθμος για εκπαίδευση ΤΝΔ. Είναι μια μέθοδος επιβλεπόμενης μάθησης και μια υλοποίηση του γνωστού κανόνα δέλτα. Ο αλγόριθμος περιγράφεται μέσα από δύο φάσεις. Η πρώτη φάση λέγεται *εμπρόσθια*. Κατά την φάση αυτή ένα ένα από τα δεδομένα για εκπαίδευση του προβλήματος μπαίνουν στο δίκτυο. Να σημειωθεί ότι για τα συγκεκριμένα δεδομένα εισόδου, η επιθυμητή έξοδος είναι γνωστή (επιβλεπόμενη μάθηση). Αφού οι υπολογισμοί γίνουν σε όλα τα κρυφά επίπεδα του δικτύου υπολογίζονται οι τιμές των νευρώνων εξόδου για την τρέχουσα εκπαίδευση. Στην συνέχεια παρουσιάζεται η επιθυμητή έξοδος του δικτύου για την συγκεκριμένη είσοδο, συγκρίνονται οι τιμές και με βάση την στατιστική μέθοδο των μέσων τετραγωνικών ελαχίστων (Least Mean Squares, LMS), υπολογίζεται το αντίστοιχο σφάλμα (εξίσωση 1.4).

$$E = 1/2 \sum_{j} (t_{pj} - o_{pj})^2$$

Εξίσωση 1.4: *Το μέσο τετραγωνικό σφάλμα, όπου* t_{pj} *η επιθυμητή έξοδος του νευρώνα εξόδου, και* o_{pj} *η πραγματική έξοδος του νευρώνα.*

Κατά την δεύτερη φάση, η οποία λέγεται *ανάστροφη μετάδοση σφάλματος*, γίνεται ένα πέρασμα του δικτύου προς τα πίσω, ώστε το λάθος που υπολογίστηκε στους νευρώνες εξόδου να μεταφερθεί προς τα πίσω, και ανάλογα με το λάθος να γίνει και η αντίστοιχη προσαρμογή των βαρών του δικτύου. Η προσαρμογή των βαρών του δικτύου γίνεται με τον γενικευμένο κανόνα δέλτα (Werbos, 1974, Rumelhart, 1986). Οι εξισώσεις που χρησιμοποιούνται για την διόρθωση των βαρών του δικτύου υπολογίζονται με βάση την μέθοδο κατάβασης κλίσης, τον κανόνα δέλτα, την μέθοδο τετραγωνικού σφάλματος και την συνάρτηση ενεργοποίησης (Αγαθοκλέους, υποκεφάλαιο 3.3.2, 2009). Ο τύπος για την αλλαγή των βαρών των νευρώνων εξόδου φαίνεται στην εξίσωση 1.5, ενώ ο τύπος αλλαγής των βαρών των νευρώνων του κρυφού επιπέδου, φαίνεται στην εξίσωση 1.6.

$$\delta_{pj} = O_{pj} (1 - O_{pj}) (t_{pj} - O_{pj})$$

Εξίσωση 1.5: *Υπολογισμός του σφάλματος για τους νευρώνες εξόδου, όπου Opj η πραγματική έξοδος του νευρώνα j (εξίσωση 1.3) και t_{pj} η επιθυμητή τιμή εξόδου.*

$$\delta_{pj} = O_{pj} (1 - O_{pj}) \Sigma \delta_{pk} W_{jk}$$

Εξίσωση 1.6: *Υπολογισμός του σφάλματος για τους νευρώνες κρυφών επιπέδων, όπου Opj η πραγματική έξοδος του νευρώνα j, δρκ το σφάλμα από τον νευρώνα κ ο οποίος είναι συνδεδεμένος με την έξοδο του j και δίδεται από την εξίσωση 1.5. Wjκ είναι το βάρος μεταξύ των δύο αυτών νευρώνων.*

1.3.5 Ensemble of Neural Network

Ένας άλλος αλγόριθμος μάθησης ΤΝΔ είναι ο συλλογικός αλγόριθμός (ensemble). Ένα ensemble ΤΝΔ ορίζεται ως μια συλλογή από ανεξάρτητα νευρωνικά δίκτυα τα οποία συνεργάζονται με σκοπό την αύξηση της γενίκευση σε σχέση με αυτή που μπορεί να επιτευχθεί με ένα και μόνο δίκτυο. Ο αλγόριθμος αυτός προσημειώνει στην ουσία την έννοια της συνεργασίας και της συλλογικής προσπάθειας. Στην πραγματική ζωή, όταν έχουμε να λύσουμε κάποιο σοβαρό πρόβλημα κάνουμε συνεδρία όπου ακούονται οι γνώμες όλων των μελών και στο τέλος παίρνεται μια ομόφωνη απάντηση, που το πιο πιθανόν θα είναι και καλύτερη από την απόφαση ενός ατόμου.

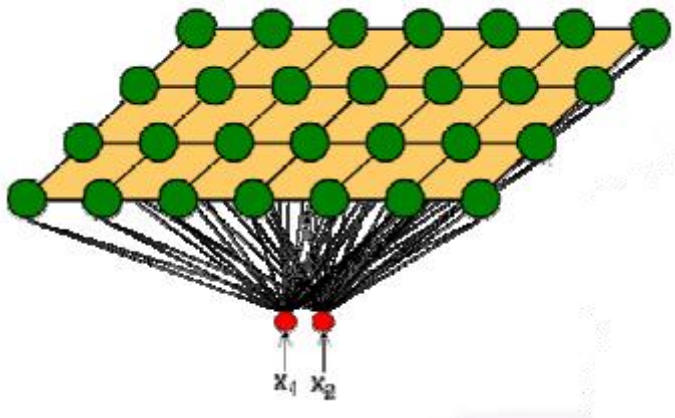
Έτσι σύμφωνα με το αλγόριθμο αυτό συνδυάζουμε πολλά νευρωνικά δίκτυα μαζί δημιουργώντας ένα πιο μεγάλο νευρωνικό δίκτυο που το αποτέλεσμα του θα υπολογίζεται λαμβάνοντας υπόψη τα αποτελέσματα όλων των ΤΝΔ συλλογικά. Με το συνδυασμό αυτό μπορούμε να βελτιώσουμε και να διορθώσουμε το ποσοστό επιτυχίας που θα είχε μόνο ένα νευρωνικό δίκτυο.

Υπάρχουν διάφοροι τρόποι για να συνδυαστούν πολλά νευρωνικά δίκτυα μαζί. Τα δίκτυα μπορούν να έχουν τη ιδία αρχιτεκτονική ή να διαφέρουν ανάλογα πάντα με το πρόβλημα που έχουν να επιλύσουν. Επίσης η τελική απόφαση του συλλογικού νευρωνικού δικτύου μπορεί να επιλεχθεί με πολλούς τρόπους. Τέτοιοι τρόποι είναι οι average, weighted average, Bayesian averaging, voting, rank-based που θα αναλυθούν και θα χρησιμοποιηθούν στα επόμενα κεφάλαια

1.3.5 Μη επιβλεπόμενη μάθηση

Όπως προαναφέρθηκε στη μη επιβλεπόμενη μάθηση δεν έχουμε κάποιο δάσκαλο που μαθαίνει στο δίκτυο ένα σύνολο δεδομένων, γιατί στην ουσία δεν γνωρίζουμε την επιθυμητή έξοδο. Αντίθετα μέσα από την εκπαίδευση του προσπαθεί να οργανώσει τα δεδομένα σε κατηγορίες ανάλογα με τα κοινά τους χαρακτηριστικά, ούτως ώστε όταν του δοθεί ένα καινούριο, άγνωστο σύνολο δεδομένων, να προσπαθήσει να το ορίσει ανάλογα με αυτά που ήδη γνωρίζει, τοποθετώντας κοντά σε μια ομάδα που έχει κοινά στοιχεία. Σε αυτή τη κατηγορία ανήκούν και αυτοοργανούμενοι χάρτες Kohonen (Haykin Simon, 1999).

Οι αυτοοργανούμενοι χάρτες Kohonen (Self-Organizing Maps SOMs) παράγουν μια αντιστοιχία από τον πολυδιάστατο χώρο σε ένα δίκτυο νευρώνων (Σχήμα 1.9). Το κύριο χαρακτηριστικό των SOM είναι ότι διατηρούν την τοπολογία, οπότε γειτονικοί νευρώνες αντιστοιχούν σε παρόμοια πρότυπα. Οι SOM οργανώνονται ως μονοδιάστατα ή δισδιάστατα δίκτυα. Αντίθετα με τα MLP δίκτυα που εκπαιδεύονται με τον αλγόριθμο back-propagation, οι SOM έχουν νευροβιολογική βάση. Στον εγκέφαλο των θηλαστικών τα οπτικά, ακουστικά και αφής ερεθίσματα χαρτογραφούνται σε επίπεδα κυττάρων. Η τοπολογία διατηρείται: αν αγγίξουμε μέρη του σώματος που βρίσκονται κοντά, θα ενεργοποιηθούν ομάδες κυττάρων που βρίσκονται επίσης κοντά.



Σχήμα 1.9: *Χάρτης Αυτοδιοργάνωσης Kohonen*

Οι Kohonen SOM είναι αποτέλεσμα της συνεργίας τριών βασικών διαδικασιών: Ανταγωνισμός, Συνεργασία, Ανταμοιβή.

Στη φάση του ανταγωνισμού, σε κάθε νευρώνα του SOM αποδίδεται ένα διανυσματικό βάρος διαστάσεων N ίσης με με διάσταση του δειγματοχώρου. Κάθε πρότυπο εισόδου συγκρίνεται με το βάρος κάθε νευρώνα και ο νευρώνας με το πλησιέστερο διανυσματικό βάρος (ελάχιστη ευκλείδεια απόσταση) ανακηρύσσεται νικητής.

Στη φάση της συνεργασίας η ενεργοποίηση του νευρώνα-νικητή διαχέεται στους νευρώνες της γειτονιάς του. Αυτό επιτρέπει η τοπολογία κοντινών νευρώνων να γίνει ευαίσθητη σε παρόμοια πρότυπα. Η απόσταση από το νικητή στην τοπολογία του δικτύου ορίζεται ως συνάρτηση του πλήθους των συνδέσεων με το νικητή. Το μέγεθος της γειτονιάς είναι αρχικά μεγάλο αλλά συρρικνώνεται με το χρόνο καθώς μεγάλη γειτονιά σημαίνει διατήρηση της τοπολογίας, ενώ μικρότερη επιτρέπει εξειδίκευση των νευρώνων.

Στη φάση της ανταμοιβής κατά την εκπαίδευση, ο νικητής και οι γείτονες του προσαρμόζουν τα βάρη τους να μοιάσουν πιο πολύ στο πρότυπο εισαγωγής. Ο κανόνας προσαρμογής είναι παρόμοιος αυτού της ανταγωνιστικής μάθησης. Οι νευρώνες που βρίσκονται πιο κοντά στο νικητή προσαρμόζουν τα βάρη τους περισσότερο από τους πιο μακρινούς και σύμφωνα με την εξίσωση 1.7. Το μέγεθος προσαρμογής ελέγχεται από το συντελεστή μάθησης που εξασθενεί με το χρόνο για να εξασφαλίσει σύγκλιση του SOM.

Η εκπαίδευση των πιο πάνω ΤΝΔ, μπορεί να γίνει με τον αλγόριθμο μάθησης που φαίνεται στο σχήμα 1.6, δεδομένου ότι έχουμε επιβλεπόμενη μάθηση.

1. Αρχικοποίηση βαρών και κατωφλίου με μικρές τυχαίες τιμές.
2. Επανέλαβε ωσότου υπάρξει σύγκλιση.

    2α. Επέλεξε πρότυπο εισόδου x_n

    (i). Επέλεξε τη μονάδα που μοιάζει περισσότερο στο x_n (μικρότερη ευκλείδεια απόσταση)

    (ii). Προσάρμοσε τα βάρη του νικητή και των γειτόνων του σύμφωνα με την εξίσωση 1.7

    2b. Μείωσε τον τελεστή μάθησης η(t) (εξίσωση 1.8)

    2c. Μείωσε τη γειτονιά σ(t) (εξίσωση 1.9)

Σχήμα 1.10: *Αλγόριθμος εκπαίδευσης SOM, όπου x_i η είσοδος στον νευρώνα, w_i το βάρος που αντιστοιχεί στην είσοδο x_i.*

$$h_{ik}(t) = \exp\left(-\frac{d_{ik}^2}{2\sigma^2(t)}\right)$$

Όπου d_{ik}=απόσταση μεταξύ w_i και w_k.

Εξίσωση 1.7: *Συνάρτηση ορισμού της γειτονιάς.*

$$\eta(t) = \eta_0 \exp\left(-\frac{t}{T_1}\right)$$

όπου n_0=αρχικός συντελεστής μάθηση $T_1=N_o Iter/log(Radius)$

Radius=MapSize/2

Εξίσωση 1.8: *Κανόνας μείωσης του συντελεστή μάθησης.*

$$\sigma(t)=Radius*exp(-\frac{t}{T1})$$

όπου ***k****=αριθμός επανάληψης*

Εξίσωση 1.9: *Κανόνας μείωσης της γειτονίας.*

1.3.7 Radial Basis Function Δίκτυα

Τα Radial Basis Function δίκτυα (RBF) είναι ένα είδος ΤΝΔ που χρησιμοποιεί radial basis functions (Συναρτήσεις Αξονικής Βάσης) ως συνάρτηση ενεργοποίησης. Τα RBF δίκτυα συνήθως αποτελούνται από τρία επίπεδα. Το επίπεδο εισόδου, ένα κρυφό επίπεδο με μία μη γραμμική συνάρτηση ενεργοποίησης, που συνήθως είναι Gaussian, και το επίπεδο εξόδου που είναι γραμμικό. Το επίπεδο εισόδου κατανέμει όλα τα δεδομένα εισόδου σε όλους τους κόμβους του κρυφού επιπέδου. Στο κρυφό επίπεδο κάθε νευρώνας αποτελεί ένα κέντρο. Τα κέντρα αντικατοπτρίζουν την κατανομή των δεδομένων στο χώρο. Τα δεδομένα χωρίζονται σε ομάδες για να κατανεμηθούν στα κέντρα, συνήθως με τη χρήση μη επιβλεπόμενης μάθησης π.χ. με τη χρήση SOM. Τα κέντρα προσπαθούν να καλύψουν ένα μεγάλο αριθμό δεδομένων που παρουσιάζουν κοινά χαρακτηριστικά. Η εκπαίδευση του δικτύου συνίσταται στην ταυτοποίηση των κέντρων. Το κρυφό στρώμα νευρώνων είναι υπεύθυνο να ανάγει το πρόβλημα που έχει να επιλύσει το δίκτυο σε ψηλότερες διαστάσεις, δηλαδή μετατρέπει το πρόβλημα από μη γραμμικά διαχωρίσιμο σε γραμμικό. Αυτή η μετατροπή επιτυγχάνεται όπως είπαμε με τη χρήση της Gaussian συνάρτησης. Τέλος, στο επίπεδο εξόδου αποτελεί στη ουσία ένα απλό δίκτυο Percepton. Πολλαπλασιάζει την έξοδο κάθε νευρώνα του κρυφού επιπέδου με ένα βάρος και το άθροισμα όλων μαζί αποτελεί το αποτέλεσμα του δικτύου. Τα RBF δίκτυα χρησιμοποιούνται κυρίως για ομαδοποίηση δεδομένων και για επίλυση προβλήματος παλινδρόμησης.

Υπάρχουν δύο είδη RBF δικτύων, αυτά με σταθερά κέντρα και αυτά με μεταβλητά κέντρα. Τα RBF δίκτυα με σταθερά κέντρα, κρατούν σταθερή τη θέση των κέντρων στο χώρο και μεταβάλλουν μόνο τα βάρη για να εκπαιδευτεί το σύστημα. Αυτή η μέθοδος χρησιμοποιείται κυρίως όταν θέλουμε να λύσουμε προβλήματα

παρεμβολής. Τα RBF δίκτυα με μεταβλητά μπορούμε να μεταβάλουμε και τα τρία σύνολο παραμέτρων του δικτύου μας, τα βάρη (συντελεστές), τα κέντρα και τη τυπική απόκλιση σ της Gaussian συνάρτησης. Αυτή η κατηγορία δικτύων χρησιμοποιείται για προσεγγιστική επίλυση προβλημάτων, όπου δεν γνωρίζουμε ποια είναι η βέλτιστη θέση των κέντρων.

Η εκπαίδευση ενός RBF Δικτύου με μεταβλητά κέντρα, μπορεί να γίνει με τον αλγόριθμο μάθησης που φαίνεται στο σχήμα 1.11

Σχετικές Κωδικοποιήσεις

Βάρη/Συντελεστές	**w**	Κέντρο	**R**
Επιθυμητή Έξοδος	**t**	Iterator κέντρου	**h**
Πραγματική Έξοδος	**y**	Αριθμός Κέντρων	**m**
Πρότυπο Εισόδου	**x**	Iterator Εξόδου	**o**
Input Dimension Iterator	**i**	Αριθμός Κέντρων	**k**
Input Dimension	**d**	Τυπική Απόκλιση	**σ**
Αριθμός Εισόδων	**n**	Learning Rate	$\mathbf{n_R}$, $\mathbf{n_\sigma}$, $\mathbf{n_w}$
Iterator Πρότυπου Εισόδου	**p**	Bias unit	**b**

Πίνακας 1.1: *Πίνακας με τις κωδικοποιήσεις των παραμέτρων που χρησιμοποιεί το RBF δίκτυο.*

1. Επέλεξε τον αριθμό (m) και αρχικοποίησε τις συντεταγμένες των κέντρων (R) των RBF συναρτήσεων.
2. Επέλεξε την αρχική τιμή της παραμέτρου (σ) για κάθε κέντρο R.
3. Αρχικοποίησε τις τιμές των βαρών w με μικρές τυχαίες τιμές [-1,1].
4. Επανάλαβε ωσότου υπάρξει σύγκλιση
5. Για κάθε πρότυπο εισόδου $x^{(p)}$
6. Υπολόγισε την έξοδο y για κάθε κόμβο ο χρησιμοποιώντας την εξίσωση 1.10.
7. Μετάβαλλε τις παραμέτρους (w,R,σ) χρησιμοποιώντας τις εξισώσεις 1.15,1.16,1.17,1.18
8. Τέλος για p=n

Σχήμα 1.11: *Αλγόριθμος εκπαίδευσης RBF.*

Εξισώσεις RBF

$$y_o^{(p)} = w_{bo} + \sum_{h=1}^{m} w_{ho}\phi(x^{(p)}, R_h, \sigma_h)$$

Εξίσωση 1.10: *Η έξοδος για κάθε κόμβο ο γία κάθε πρότυπο p*

$$\phi(x^{(p)}, R_h, \sigma_h) = exp\left(-\frac{||x^{(p)} - R_h||^2}{2\sigma_h^2}\right)$$

Εξίσωση 1.11:*Η Gaussian Συνάρτηση*

$$||x^{(p)} - R_h||^2 = \sum_{i=1}^{d} \left(x_i^{(p)} - R_{hi}\right)^2$$

Εξίσωση 1.12:*Εξίσωση ευκλείδειας απόστασης.*

$$E = \frac{1}{2} \sum_{p=1}^{n} \sum_{o=1}^{k} \left[\varepsilon_o^{(p)}\right]^2$$

Εξίσωση 1.13:*Εξίσωση Ολικού Σφάλματος*

$$\varepsilon_o^{(p)} = t_o^{(p)} - y_o^{(p)}$$

Εξίσωση 1.14:*Εξίσωση σφάλματος νευρώνα ο του προτύπου p*

$$w_{ho}^{'} = w_{ho} + \eta_w \varepsilon_o^{(p)} \phi(x^{(p)}, R_h, \sigma_h)$$

Εξίσωση 1.15:*Εξίσωση αλλαγής βαρών/συντελεστών (από το κέντρων h μέχρι το νευρώνα εξόδου ο)*

$$w_{bo}^{'} = w_{bo} + \eta_w \varepsilon_o^{(p)}$$

Εξίσωση 1.16:*Εξίσωση αλλαγής βαρών/συντελεστών (από το bias b μέχρι το νευρώνα εξόδου ο)*

$$R_{hi}^{'} = R_{hi} + \eta_R \sum_{o=1}^{k} \varepsilon_o^{(p)} w_{ho} \phi(x^{(p)}, R_h, \sigma_h) \frac{x_i^{(p)} - R_{hi}}{\sigma_h^2}$$

Εξίσωση 1.17:*Εξίσωση αλλαγής θέσεων κέντρων*

$$\sigma_h^{'} = \sigma_h + \eta_\sigma \sum_{o=1}^{k} \varepsilon_o^{(p)} w_{ho} \phi(x^{(p)}, R_h, \sigma_h) \frac{||x^{(p)} - R_h||^2}{\sigma_h^3}$$

Εξίσωση 1.18:*Εξίσωση αλλαγής θέσεων πλάτους της διασποράς σ της Gaussian συνάρτησης του κέντρου h.*

1.4 Εξελικτικοί Αλγόριθμοι (Evolutionary Algorithms)

Οι Εξελικτικοί Αλγόριθμοί αποτελούν περιοχή της Υπολογιστικής και Τεχνητής Νοημοσύνης που συμπεριλαμβάνει τους:

Γενετικούς Αλγόριθμους (GA)

Εξελικτικές Στρατηγικές (ES)

Εξελισσόμενο Προγραμματισμό (EP)

Γενετικό Προγραμματισμό (GP)

Συστήματα Μάθησης Πρωτοτύπων (LCS)

Εξελισσόμενο Υλισμικό (EHW)

Στα επόμενα περιγράφουμε τους ΓΑ και ΕΣ που θα χρησιμοποιήσουμε στα πλαίσια της εργασίας.

1.4.1 Γενετικοί Αλγόριθμοι

Οι Γενετικοί Αλγόριθμοι (ΓΑ) είναι μια κατηγορία Εξελικτικών Αλγορίθμων, οι οποίοι είναι μια εξελισσόμενη περιοχή της Τεχνητής Νοημοσύνης. Οι Εξελικτικοί Αλγόριθμοι είναι αλγόριθμοι επίλυσης προβλημάτων που βασίζονται στις αρχές της βιολογικής εξέλιξης, δηλαδή της φυσικής επιλογής και της επικράτησης του ισχυρότερου. Οι Γενετικοί Αλγόριθμοι, είναι αλγόριθμοι αναζήτησης και βελτιστοποίησης, βασίζονται στις αρχές της εξέλιξης που παρατηρείται στην φύση, και γίνονται όλο και περισσότερο γνωστοί χάριν της ικανότητας τους να ανακαλύπτουν γρήγορα και αξιόπιστα τις καλές λύσεις δύσκολων και μεγάλης διάστασης προβλημάτων. Οι γενετικοί αλγόριθμοι είναι χρήσιμοι όταν ο χώρος αναζήτησης είναι μεγάλος ή σύνθετος, όταν καμία μαθηματική ανάλυση δεν είναι διαθέσιμη, ή όταν οι παραδοσιακές μέθοδοι αναζήτησης αποτυγχάνουν. Επιπρόσθετα είναι εύκολα επεκτάσιμοι και εξελίξιμοι και μπορούν να χρησιμοποιηθούν σε υβριδικές μορφές μαζί με άλλες μεθόδους.

Ποια είναι όμως τα κύρια χαρακτηριστικά των ΓΑ που τους προδίδουν αυτήν την ισχύ στο να λύνουν προβλήματα γρήγορα και αποδοτικά; Κατ’ αρχήν οι ΓΑ

δουλεύουν με μια κωδικοποίηση ενός συνόλου τιμών που μπορούν να λάβουν οι μεταβλητές, και όχι με τις ίδιες τις μεταβλητές του προβλήματος. Η κωδικοποίηση στους Γενετικούς Αλγόριθμους, γίνεται με τη χρήση της δυαδική κωδικοποίηση. Οι ΓΑ, κάνουν αναζήτηση σε πολλά σημεία του χώρου ταυτόχρονα και όχι μόνο σε ένα. Επιπλέον, μπορούν να χρησιμοποιούν μόνο την αντικειμενική συνάρτηση σαν πληροφορία της απόδοσης του αλγορίθμου, και καμία άλλη πληροφορία. Τέλος οι ΓΑ, χρησιμοποιούν πιθανοθεωρητικούς κανόνες αναζήτησης και όχι ντετερμινιστικούς, κάτι το οποίο γίνεται και στην φύση (Λυκοθανάσης, 2001).

Πως όμως λειτουργούν οι ΓΑ; Για να το καταλάβουμε αυτό, πρέπει να δούμε μερικές έννοιες που αφορούν την βιολογία, οι οποίες συσχετίζονται άμεσα με την λειτουργία των ΓΑ. Γνωρίζουμε ότι κάθε ζωντανός οργανισμός αποτελείται από κύτταρα, μέσα στα οποία βρίσκεται το γενετικό υλικό (DNA). Το DNA, περιέχει τις γενετικές πληροφορίες που καθορίζουν τα χαρακτηριστικά του κάθε οργανισμού και έχει την μορφή διπλής έλικας. Τα χρωμοσώματα είναι τμήματα του DNA και αποτελούνται από γονίδια, όπου το κάθε γονίδιο ανάλογα με την θέση του κωδικοποιεί ένα συγκεκριμένο χαρακτηριστικό του οργανισμού. Κατά την λειτουργία της αναπαραγωγής, τα άτομα διασταυρώνονται και συνεπώς γίνεται η διασταύρωση των γονιδίων τους (Crossover). Το νέο άτομο έχει χαρακτηριστικά τα οποία κληρονόμησε και από τους δυο του τους γονείς. Όμως, υπάρχει και η πιθανότητα της μετάλλαξης (Mutation), κατά την οποία κάποιο στοιχείο του DNA έχει αλλάξει, και αυτό οφείλεται συνήθως σε λάθος αντιγραφή των γονιδίων από τους γονείς. Μια συνοπτική περιγραφή των λειτουργιών τους φαίνεται στο σχήμα 1.12.

1. Κωδικοποίηση (Coding)
2. Επιλογή αντικειμενικής συνάρτησης (Initialization)
3. Αποκωδικοποίηση (Decoding)
4. Υπολογισμός ικανότητας ή αξιολόγηση (Fitness evaluation)
5. Επιλογή (Selection)
6. Αναπαραγωγή (Reproduction)
7. Διασταύρωση (Crossover)
8. Μετάλλαξη (Mutation)

Επανάληψη από το βήμα (2) μέχρι να ικανοποιηθεί το κριτήριο τερματισμού

Σχήμα 1.12: *Ένας απλός γενετικός αλγόριθμος (Λυκοθανάσης, 2001).*

Κατ αρχήν γίνεται η κωδικοποίηση των πιθανών λύσεων του προβλήματος σε χρωμοσώματα (αρχικός πληθυσμός) καθώς και η επιλογή της συνάρτησης καταλληλότητας που αντιπροσωπεύει το πρόβλημα. Μετά γίνεται η αρχικοποίηση του αρχικού αυτού πληθυσμού. Έπειτα, γίνεται η αποκωδικοποίηση των ατόμων της γενεάς (φαινότυπος), και ο υπολογισμός της ικανότητας για επιβίωση του κάθε χρωμοσώματος (συνάρτηση αξιολόγησης). Αυτό μας δίνει μια τιμή ανάλογα με το πόσο καλά λύνει το πρόβλημα το συγκεκριμένο χρωμόσωμα. Στην συνέχεια λαμβάνει μέρος η διαδικασία της επιλογής, που με βάση κάποια πιθανότητα επιλέγει τα «καλύτερα» άτομα της γενεάς τα οποία θα διασταυρωθούν και θα δώσουν πιθανώς νέα καλύτερα άτομα, άρα νέες περιοχές του χώρου αναζήτησης οι οποίες μπορούν να δώσουν καλύτερες λύσεις. Έπειτα γίνεται η αναπαραγωγή (ανά δυο άτομα μεταξύ τους) των ατόμων που επιλέχθηκαν στο προηγούμενο στάδιο. Η διασταύρωση είναι μια πολύ σημαντική διαδικασία, επειδή ανακατευθύνει τον αλγόριθμο σε νέα μονοπάτια του χώρου αναζήτησης. Ακολουθεί η διαδικασία της μετάλλαξης, όπου κάποια στοιχεία των χρωμοσωμάτων αλλάζουν ώστε ο αλγόριθμος να αποκλείσει μονοπάτι ψαξίματος. Οι διαδικασίες αυτές γίνονται με κάποια πιθανοτικά κριτήρια Pm (πιθανότητα για μετάλλαξη), Pc (πιθανότητα για διασταύρωση). Κριτήρια τερματισμού των ΓΑ υπάρχουν πολλά, όπως για παράδειγμα ο αριθμός των μέγιστων γενεών (Λυκοθανάσης, 2001). Συνολικά οι παράμετροι που αφορούν έναν γενετικό αλγόριθμο συμπεριλαμβανομένου των πιθανοτικών κριτηρίων φαίνονται στον πίνακα 1.2.

Παράμετρος	Επεξήγηση παραμέτρου
Pop_number	Αριθμός πληθυσμού
Pc	Πιθανότητα για διασταύρωση
Pm	Πιθανότητα για μετάλλαξη
Num_generations	Αριθμός γενεών (τερματικό κριτήριο)

Πίνακας 1.2: *Παράμετροι ενός τυπικού Γενετικού Αλγόριθμου.*

1.4.2 Evolutionary Strategies

Μια άλλη κατηγορία Εξελικτικών Αλγορίθμων είναι οι Evolutionary Strategies (Εξελικτικές Στρατηγικές). Η ΕΣ τεχνική δημιουργήθηκε γύρω στη δεκαετία του 60 and και επεκτάθηκε περισσότερο τη δεκαετία το 70 από τους Ingo Rechenberg και Hans-Paul Schwefel. Η κατηγορία αυτή αναπτύχθηκε για να επιλύσει προβλήματα που δεν μπορούσαν να λύσουν αποτελεσματικά οι Γενετικοί Αλγόριθμοι. Λόγω της δυαδικής κωδικοποίησης οι Γενετικοί Αλγόριθμοι παρουσιάζουνε μεγάλη δυσκολία στο να λύσουν προβλήματα που έχουν να κάνουν με δεκαδικές τιμές. Μπορούν μεν να κωδικοποιήσουν ένα δεκαδικό αριθμό, όμως όσο αυξάνεται η ακρίβεια του δεκαδικού, τόσο μεγαλώνει και το μήκος του χρωματοσώματός. Άρα αυξάνεται η πολυπλοκότητα του προβλήματος και επομένως πολλές φορές καταστείτε αδύνατη η επίλυση κάποιων προβλημάτων.

Ποιες είναι όμως οι κύριες διαφορές σε σχέση με του Γενετικούς Αλγόριθμούς; Στις εξελικτικές στρατηγικές όπως προαναφέρθηκε η κωδικοποίηση δε γίνεται δυαδικά, αλλά με δεκαδικούς αριθμούς. Δηλαδή ένα χρωμόσωμα αποτελείται από ένα πίνακα που περιέχει δεκαδικούς αριθμούς και κάθε θέση του πίνακα αποτελεί και μια συγκεκριμένη παράμετρο του προβλήματος που θέλουμε να επιλύσουμε.

Άλλη πολύ σημαντική διαφορά σε σχέση με τους Γενετικούς Αλγόριθμους είναι ότι εδώ κύριος τρόπος αναπαραγωγής δεν είναι πλέον η διασταύρωση αλλά η μετάλλαξη. Μπορεί φυσικά να εκτελεστεί διασταύρωση αλλά δεν θα δώσει τα επιθυμητά αποτελέσματα.

Όπως αναφέραμε στη πιο πάνω παράγραφο κύριος τρόπος αναπαραγωγής είναι η μετάλλαξη. Στις εξελικτικές στρατηγικές δεν υπάρχει ένα μικρό ποσοστό χρωμοσωμάτων που θα υποστούν μετάλλαξη αλλά όλες οι θέσεις έχουν πιθανότητα να μεταλλαχτούν. Η μετάλλαξη γίνεται με τη χρήση της Gaussian κατανομής η οποία θα μας δώσει απόγονους οι οποίοι θα είναι σχετικά κοντά στους γονείς.

Υπάρχουν διάφοροι τρόποι επιλογής της επομένης γενεάς οι χρησιμοποιούνται επίσης και στους ΓΑ που αναφέρθηκαν πιο πάνω. Οι πιο διαδομένοι είναι οι Tournament Selection, Roulette Selection, Truncation Selection. Στο Truncation Selection τα άτομα ταξινομούνται βάση του fitness function και μια αναλογία. Αυτοί που συγκέντρωσαν το ψηλότερο ποσοστό επιλέγονται να συνεχίσουν στην επόμενη γενεά. Στο Tournament Selection επιλέγουμε τυχαία κάποια άτομα από το πληθυσμό και για τα συγκεκριμένα άτομα βρίσκουμε τα fitness τους για διάφορες δοκιμές. Στην ουσία δημιουργούμε κάποιου είδους τουρνουά όπου οι νικητές κάθε τουρνουά συνεχίζουν στην επόμενη γενεά. Στο Roulette Selection, κάθε άτομα αποκτά ένα μερίδιο σε μια “ρουλέτα”, ανάλογα με το fitness του. Όσο μεγάλο είναι το fitness, τόσο πιο μεγάλο είναι το μερίδιο. Στη συνέχεια “γυρίζουμε” τη ρουλέτα και τα άτομα που θα μας δείξει συνεχίζουν την επόμενη γενεά. Κάθε μέθοδος έχει τα πλεονεκτήματα και τα μειονεκτήματα. Ανάλογα με το πρόβλημα που έχουμε να επιλύσουμε διαλέγουμε τη κατάλληλη. Στην εργασία μας θα επιλέξουμε το Truncation Selection.

1. Κωδικοποίηση (Coding)
2. Επιλογή αντικειμενικής συνάρτησης (Initialization)
3. Αποκωδικοποίηση (Decoding)
4. Δημιουργία προσωρινού πληθυσμού
   i) Μετάλλαξη κάθε ατόμου για δημιουργία του offspring
   ii) Υπολογισμός ικανότητας ή αξιολόγηση (Fitness evaluation)
5. Επιλογή (Selection)

Επανάληψη από το βήμα (2) μέχρι να ικανοποιηθεί το κριτήριο τερματισμού της ΕΣ.

Σχήμα 1.13: *Ένας απλός αλγόριθμος Εξελικτικής Στρατηγικής.*

Κεφάλαιο 2

Προηγούμενη Εργασία

2.1 Αρχιτεκτονική BRNN

2.2 Αρχικοποίηση BRNN

2.3 Υλοποίηση BRNN

2.4 Τροποποιήσεις και Βελτιστοποίησης BRNN

2.4.1 Εισαγωγή Πολλαπλής Στοίχισης Ακολουθιών

2.4.2 Ενσωμάτωση SOV Score

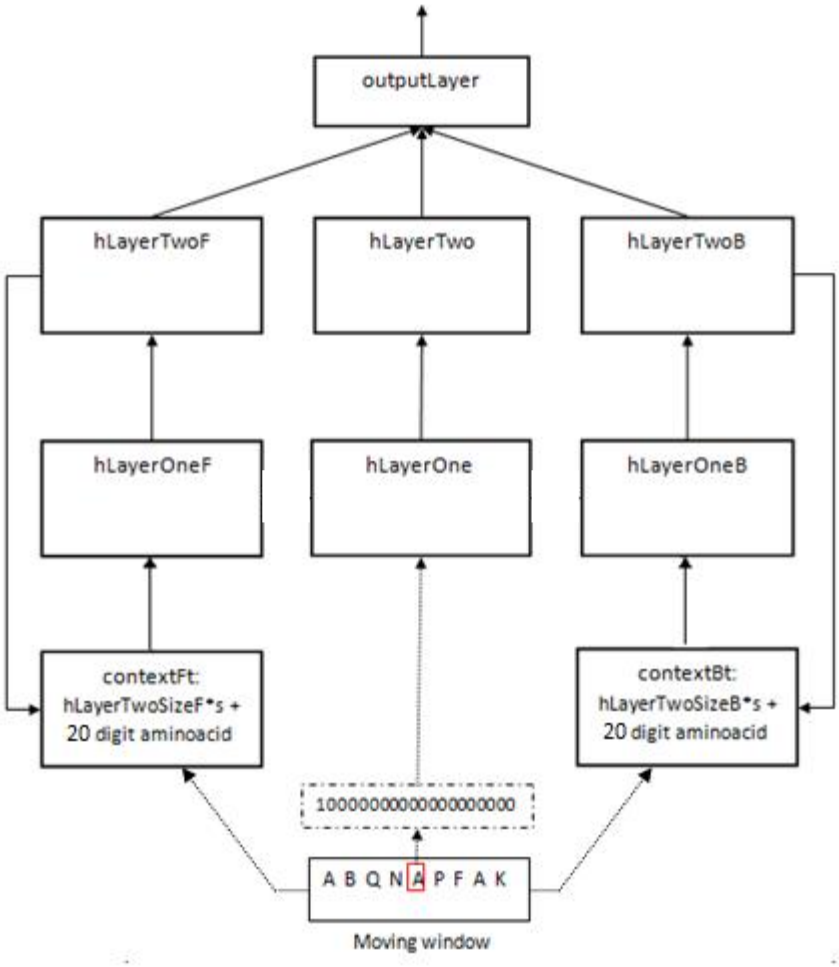
Με το συγκεκριμένο θέμα της παρούσας εργασίας είχαν ασχοληθεί πιο πριν η Γεωργία Χριστοδούλου (2010) και ο Μιχάλης Αγαθοκλέους (2009). Στόχος της παρούσας εργασίας είναι να βελτιώσουμε και να επεκτείνουμε τα αποτελέσματα που πέτυχαν προηγουμένως οι δύο πιο πάνω φοιτητές, ενσωματώνοντας νέες μεθόδους και αλγορίθμους, προσπαθώντας να αυξήσουμε το ποσοστό επιτυχίας του προγράμματος τόσο σε Q3 επίπεδο, όσο και σε SOV. Για να το επιτύχουμε αυτό πρέπει πρώτα να μελετήσουμε και να κατανοήσουμε το σύστημα που είχε δημιουργήσει πρώτα ο Αγαθοκλέους (2009) και το οποίο ήταν ένα BRNN δίκτυο και τροποποίησε στη συνέχεια η Χριστοδούλου (2010). Σαν αρχική ιδέα, για το σύστημα χρησιμοποιήθηκε η αρχιτεκτονική που πρότεινε ο Baldi και οι συνεργάτες του (Baldi et al., 1999), αλλά η υλοποίηση έγινε με αρκετές διαφορές ώστε να προσεγγιστεί το θέμα από μια διαφορετική σκοπιά.

2.1 Αρχιτεκτονική BRNN

Αρχικά θα αναλύσουμε το δίκτυο που είχε σχεδιάσει ο Αγαθοκλέους και το οποίο είναι και η βασική δομή του συστήματος που θα μελετήσουμε και θα επεκτείνουμε. Όπως και το δίκτυο του Baldi (1999), έτσι και το τρέχον δίκτυο ακολουθεί την ίδια ακριβώς αρχιτεκτονική. Αποτελείται από δύο ανεξάρτητα νευρωνικά δίκτυα αμφίδρομης ανάδρασης και ένα νευρωνικό δίκτυο εμπρόσθιου περάσματος τα οποία συσχετίζουν τις εξόδους τους ώστε να δώσουν το τελικό αποτέλεσμα. Για την είσοδο των αμινοξέων χρησιμοποιείται ένα κινητό παράθυρο το οποίο περνά πάνω απ’ όλη την ακολουθία της πρωτεΐνης, και κάθε φορά υπολογίζεται το αποτέλεσμα για το κεντρικό αμινοξύ, συσχετίζοντας τα αμινοξέα που το ακολουθούσαν και τα αμινοξέα που έπονται του κεντρικού.

Πιο συγκεκριμένα, το αριστερό νευρωνικό δίκτυο Ft (σχήμα 2.1), επεξεργάζεται τα αμινοξέα που προηγούνται του αμινοξέως που βρίσκεται στο κέντρο του παραθύρου. Παρόμοια, το δεξιό νευρωνικό δίκτυο Bt επεξεργάζεται τα αμινοξέα που έπονται του κεντρικού αμινοξέως του κινητού παραθύρου. Αυτό είναι πολύ σημαντικό επειδή λόγω της φύσης των πρωτεϊνών, η δευτεροταγής δομή που σχηματίζουν εξαρτάται από τις αλληλεπιδράσεις των αμινοξέων μεταξύ τους. Είναι απαραίτητο λοιπόν για το σύστημα να συσχετίζει το σύνολο της αλληλουχίας που

προηγείται ή έπεται ενός αμινοξέως. Αυτό μοντελοποιείται από την φύση των δικτύων με ανάδραση, ώστε σε κάθε νέα τους είσοδο να έχουν μια πληροφορία για τα προηγούμενα και τα επόμενα αμινοξέα. Το κεντρικό νευρωνικό δίκτυο, παίρνει σαν είσοδο και επεξεργάζεται το αμινοξύ που βρίσκεται στο κέντρο του παραθύρου, του οποίου αναζητούμε να προβλέψουμε την δευτεροταγή δομή. Το *outputLayer* στο τέλος θα δώσει την δευτεροταγή δομή του αμινοξέως αυτού.



Σχήμα 2.1*: Το τρέχον σύστημα, το οποίο υλοποιήθηκε από τον Αγαθοκλέους (Αγαθοκλέους, 2009).*

Η διαδικασία αυτή γίνεται για όλα τα αμινοξέα μιας πρωτεΐνης. Όλα με την σειρά τους, θα βρεθούν κάποια στιγμή στο κέντρο του κινούμενου παραθύρου και θα προβλεφθεί έτσι η δευτεροταγής δομή τους. Αυτό γίνεται για κάθε πρωτεΐνη του συνόλου εκπαίδευσης και έπειτα, αφού με τους κατάλληλους αλγόριθμους

εκπαιδευτεί το δίκτυο, προβλέπει και την δευτεροταγή δομή των πρωτεϊνών που ανήκουν στο σύνολο επαλήθευσης έτσι ώστε να αποφανθεί εάν όντως εκπαιδεύτηκε. Αυτή είναι η γενική ιδέα του όλου δικτύου (σχήμα 2.1).

Επεξεργασία Δεδομένων

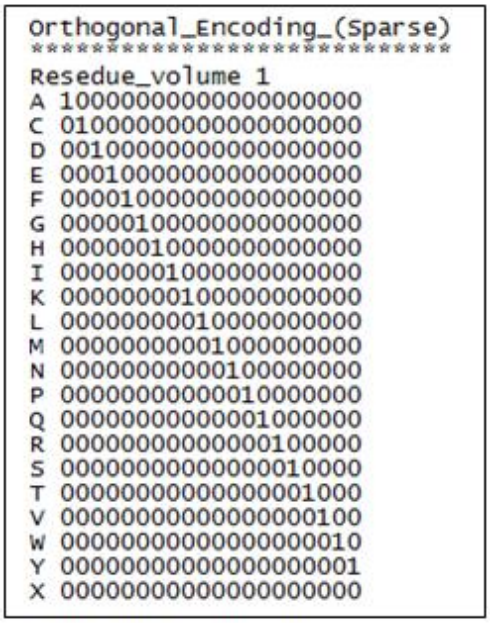
Για τη κατανόηση του τρόπου λειτουργίας του συστήματος πρέπει να μελετήσουμε τόσο τον τρόπο που έχει αναπτυχθεί το σύστημα όσο και τον τρόπο εισαγωγής και εξαγωγής των δεδομένων. Θα ξεκινήσουμε με τον τρόπο που διαβάζονται και κωδικοποιούνται τα δεδομένα στο δίκτυο.

Το δίκτυο στο επίπεδο εξόδου έχει τρείς εξόδους για να μπορέσει να κωδικοποιήσει τις τρεις πιθανές εξόδους που αντιπροσωπεύουν την δευτεροταγή δομή των πρωτεϊνών. Η έξοδος του συστήματος για κάθε αμινοξύ είναι ή Helix ή Extended (Strand) ή Loop τα οποία στο αρχείο συμβολίζονται με H, E και L αντίστοιχα. Χρησιμοποιώντας το αρχείο threeClasses.txt (σχήμα 2.2) μπορούμε να παρατηρήσουμε ότι γίνεται μια συσχέτιση των ομάδων δευτεροταγούς δομής G, H, E, B, I, T, S, L σε μόνο τρεις ομάδες, H, E, L και τον τρόπο κωδικοποίηση των τριων πιθανών εξόδου.

```
Three_Classes
*************
Classes_volume 3
*****************
H G,H
E E,B
L I,T,S,L
Output_Encoding
***************
H 100
E 010
L 001
```

Σχήμα 2.2*: Το αρχείο threeClasses.txt, το οποίο προσδιορίζει την κωδικοποίηση εξόδου του κάθε αμινοξέως και την τυποποίηση τους .*

Στο σύστημα του Αγαθοκλέους παρατηρούμε ότι η κωδικοποίηση των 20 αμινοξέων γίνεται με τη χρήση του αρχείου sparse.txt (σχήμα 2.3) και αυτό αποτελεί και τον τρόπο εισαγωγής των δεδομένων στο δίκτυο.



Σχήμα 2.3*: Το αρχείο «sparse.txt», το οποίο προσδιορίζει την κωδικοποίηση του κάθε αμινοξέως .*

Επίσης υπάρχει το αρχείο parameters.dat το οποίο περιέχει όλα τα στοιχεία αρχικοποιήσεις των δικτύων που θα χρησιμοποιηθούν (Σχήμα 2.4).). Η περιγραφή των παραμέτρων αυτών φαίνεται στον πίνακα 2.1 που δημιουργήθηκε από τον Αγαθοκλέους (Αγαθοκλέους, 2009)

```
Hidden_layer_one_size 12
Hidden_layer_two_size 12
Hidden_layer_one_of_Backward_size 13
Hidden_layer_two_of_Backward_size 12
Hidden_layer_one_of_Forward_size 13
Hidden_layer_two_of_Forward_size 12
Activation_Function_Type 1
Learning_Rate 0.01
Momentum 0.5
Window_size 15
q_minus_one 0.7
q_plus_one 0.7
Error_Function_Type
s 3
Maximum_Iterations 200
input_Profile sparce.txt
output_Profile threeClasses.txt
train_File msaProteinsTrainBigDataset_afterProcess.txt
test_File msaProteinsTestBigDataset_afterProcess.txt
```

Σχήμα 2.4*: Το αρχείο «parameters.dat», το οποίο προσδιορίζει τις παραμέτρους του συστήματος .*

Όνομα Παραμέτρου	Εξήγηση Λειτουργίας
Hidden_layer_one_size	Μέγεθος πρώτου κρυφού επιπέδου δικτύου N
Hidden_layer_two_size	Μέγεθος δεύτερου κρυφού επιπέδου δικτύου N
Hidden_layer_one_of_Backward_size	Μέγεθος πρώτου κρυφού επιπέδου δικτύου B
Hidden_layer_two_of_Backward_size	Μέγεθος δεύτερου κρυφού επιπέδου δικτύου B
Hidden_layer_one_of_Forward_size	Μέγεθος πρώτου κρυφού επιπέδου δικτύου F
Hidden_layer_two_of_Forward_size	Μέγεθος δεύτερου κρυφού επιπέδου δικτύου F
Activation_Function_Type	Επιλογή Συνάρτησης ενεργοποίησης
Learning_Rate	Επιλογή ρυθμού μάθησης
Momentum	Επιλογή ορμής
Window_size	Μέγεθος κινούμενου παραθύρου
q_minus_one	Σταθερά context δικτύου F
q_plus_one	Σταθερά context δικτύου B
Error_Function_Type	Επιλογή συνάρτησης σφάλματος
s	Είσοδος δικτύου για vanishing gradient
Maximum_Iterations	Αριθμός επαναλήψεων
input_Profile	Όνομα αρχείου για κωδικοποίηση εισόδου
output_Profile	Όνομα αρχείου για κωδικοποίηση εξόδου
train_File	Όνομα αρχείου με σύνολο δεδομένων εκπαίδευσης
test_File	Όνομα αρχείου με σύνολο δεδομένων επαλήθευσης

Πίνακας 2.1*: Επεξήγηση του αρχείου «parameters.dat» (Αγαθοκλέους, 2009).*

2.3 Υλοποίηση BRNN

Όπως προαναφέρθηκε το άλλο σημαντικό να κατανοήσουμε είναι ο τρόπος που έχει υλοποιηθεί το δίκτυο BRNN και πως λειτουργεί.

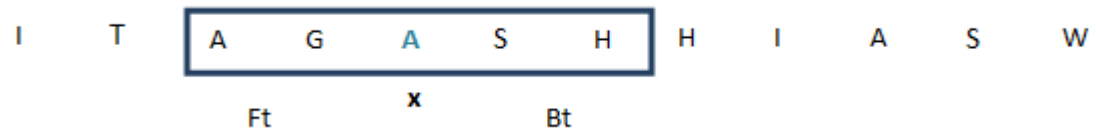
Η κύρια συνάρτηση του όλου συστήματος βρίσκεται στην κλάση *pssp_ucy.cpp*: σε αυτήν βρίσκεται ο κύριος βρόγχος που ελέγχει ολόκληρο το σύστημα. Γίνεται η αρχικοποίηση των διάφορων παραμέτρων του δικτύου από τα αντίστοιχα αρχεία και η κατασκευή του όλου νευρωνικού δικτύου του σχήματος 2.1 με βάση αυτές τις παραμέτρους.

Στο εσωτερικό του κύριου βρόγχου παρατηρούμε ότι εκτελούνται δυο εσωτερικοί βρόγχοι (Σχήμα 2.5), ένας ο οποίος εκπαιδεύει το δίκτυο χρησιμοποιώντας το σύνολο πρωτεϊνών εκπαίδευσης, και ο δεύτερος είναι ένας βρόγχος ο οποίος απλώς επαληθεύει το τρέχον δίκτυο χρησιμοποιώντας το σύνολο πρωτεϊνών επαλήθευσης.

```
While (epoch<maxEpoch)
{
        While (υπάρχουν πρωτεΐνες στο training dataset)
        {
                Πάρε την επόμενη πρωτεΐνη x
                Για κάθε αμινοξύ t της πρωτεΐνης x
                {
                        doFeedForward (t)
                        doBackPropagation (t)
                }
        }
        While (υπάρχουν πρωτεΐνες στο testing dataset)
        {
                Πάρε την επόμενη πρωτεΐνη x
                Για κάθε αμινοξύ t της πρωτεΐνης x
                {
                        doFeedForward (t)
                }
        }
}
```

Σχήμα 2.5: *Αλγόριθμος Εκπαίδευσης BRNN*

Για κάθε αμινοξύ μιας πρωτεΐνης λοιπόν, δημιουργείται ένα κινούμενο παράθυρο ώστε το αμινοξύ να βρίσκεται στο κέντρο του παραθύρου αυτού. Έπειτα καλείται η συνάρτηση *doFeedForward()* η οποία είναι υπεύθυνη να μεταφέρει την πληροφορία των καταλοίπων που περιέχονται στο κινούμενο παράθυρο, στο δίκτυο για επεξεργασία. Στην συνέχεια υπολογίζεται η έξοδος του δικτύου για το συγκεκριμένο κεντρικό αμινοξύ. Τέλος, ακολουθεί η συνάρτηση *doBackPropagation()* για το κεντρικό αμινοξύ. Η συνάρτηση αυτή υπολογίζει το σφάλμα του δικτύου για το συγκεκριμένο αμινοξύ και διορθώνει απευθείας τα βάρη των επιπέδων του δικτύου. Παρατηρούμε ότι αφού η διόρθωση των βαρών γίνεται σε κάθε αμινοξύ κάθε πρωτεΐνης, αυτό σημαίνει ότι στο δίκτυο αυτό δίνεται μια σημαντική ποσότητα πληροφορίας κάθε φορά.

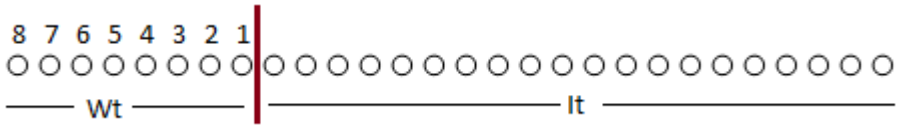


Σχήμα 2.6*: Αναπαράσταση της κλάσης doFeedForward().*

Η *doFeedForward()* είναι η μέθοδος η οποία είναι υπεύθυνη να κάνει το εμπρόσθιο πέρασμα των δεδομένων και στα τρία δίκτυα που συναποτελούν το BRNN με τη βοήθεια το κινητού παραθύρου που διαχωρίζει τα αμινοξέα ανάλογα με το ποια έπονται και ποια προηγούνται του κεντρικού αμινοξέως. Όπως φαίνεται στο Σχήμα 2.6, το κινητό παράθυρο είναι μέγεθος 5 και άρα θα επεξεργαστούμε την πληροφορία 5 αμινοξέων. Το A,G,A είναι τα αμινοξέα που θα επεξεργαστεί το αριστερό νευρωνικό δίκτυο από αριστερά προς δεξιά, τα A,S,H είναι τα αμινοξέα τα οποία θα επεξεργαστεί το δεξιό νευρωνικό δίκτυο από δεξιά προς αριστερά. Το κεντρικό αμινοξύ είναι το A και είναι αυτό που θα επεξεργαστεί από το κεντρικό δίκτυο. Σημαντικό να σημειωθεί ότι για το κεντρικό δίκτυο, στο υπάρχον σύστημα δεν έχουμε κάποιο παράθυρο. Προσοχή δίνεται στο ότι το δεξιό και αριστερό νευρωνικό δίκτυο αμφίδρομης ανάδρασης δίνουν τα αποτελέσματά τους μόνο μια φορά για κάθε κινούμενο παράθυρο και όχι για κάθε αμινοξύ το οποίο επεξεργάζονται. Όμως η πληροφορία από τα προηγούμενα αμινοξέα δεν χάνεται, επειδή το δίκτυο έχει την ιδιότητα να κρατά στην μνήμη τις προηγούμενες

πληροφορίες. Αυτό φυσικά, εξαρτάται από δύο παράγοντες, την χρονική μεταβλητή q και την μεταβλητή s. Η μεταβλητή s δηλώνει πόσο χρόνο θα μείνει η τρέχουσα έξοδος του δικτύου με ανάδραση σαν μέρος της νέας εισόδου στο δίκτυο. Όσο μεγαλύτερο είναι το s, τόσο περισσότερο χρόνο θα παραμένει σαν τμήμα της επόμενης εισόδου του δικτύου με ανάδραση. Μια τυπική είσοδος στο context layer του F δικτύου, φαίνεται στο σχήμα 2.7. Το It είναι η κωδικοποίηση του συγκεκριμένου αμινοξέως που εξετάζει την στιγμή εκείνη, ενώ το Wt αφορά την μεταβλητή s. Η μεταβλητή αυτή καθορίζει πόσα στοιχεία από τα προηγούμενα αποτελέσματα των αμινοξέων θα κρατήσει ώστε να τα επεξεργαστεί μαζί με το νέο αμινοξύ. Εάν για παράδειγμα το s ισούται με 1, τότε στην μνήμη θα κρατηθούν μόνο τα αποτελέσματα των νευρώνων εξόδου που αφορούσαν το αμέσως προηγούμενο αμινοξύ. Εάν όμως το s είναι μεγαλύτερο από 1, τότε στην μνήμη κρατούνται τα αποτελέσματα των νευρώνων εξόδου για τα s προηγούμενα αμινοξέα. Το ίδιο ακριβώς γίνεται και για το context layer του Bt δικτύου. Το s είναι μια σημαντική παράμετρος που πρέπει να προσεχθεί ιδιαίτερα γιατί όσο πιο μεγάλο είναι το s ναι μεν θυμάται περισσότερο αλλά αυτό σημαίνει επίσης ότι επηρεάζεται και εξαρτάται περισσότερο από το τι προηγήθηκε και τι έπεται. Αυτό μπορεί να οδηγήσει σε χαμηλό ποσοστό πρόβλεψης ενώ επίσης προσθέτει περισσότερο χρόνο επεξεργασίας στο σύστημα και άρα το κάνει πιο αργό.

Φυσικά, τα αποτελέσματα της εξόδου του δικτύου με ανάδραση δεν μεταφέρονται αναλλοίωτα πίσω στην είσοδο. Υπάρχει η χρονική σταθερά, q η οποία πολλαπλασιάζει τα αποτελέσματα αυτά με μια τιμή ώστε να εξυπηρετείται η αλλαγή τους στον χρόνο.



Σχήμα 2.7*: Αναπαράσταση της εισόδου του Ft δικτύου, η οποία σχηματίζεται στο context layer. Το It είναι η 20 ψηφίων κωδικοποίηση του αμινοξέως εισόδου, και το Wt είναι ο αριθμός νευρώνων εξόδου για κάθε προηγούμενο αμινοξύ που θέλουμε να κρατηθεί στην μνήμη. Στην περίπτωση αυτή το Wt είναι οχτώ.*

Τέλος, στην επεξεργασία του κεντρικού αμινοξέως, το δίκτυο χρησιμοποιεί μια *step function* στο επίπεδο εξόδου (Σχήμα 2.8), ώστε να τροποποιήσει τα αποτελέσματα σε τιμές 0 και 1. Έτσι σχηματίζεται η κωδικοποίηση του αποτελέσματος (δευτεροταγούς δομής) και είναι έτσι συγκρίσιμο με την αρχική κωδικοποίηση που δόθηκε μέσω του αρχείου «threeClasses.txt» για το επίπεδο εξόδου.

```
For each output neuron x
{
        If (output of x>0.5) {output of x=1}
        else {output of x = 0}
}
```

Σχήμα 2.8*: Αναπαράσταση της step function.*

Ο αλγόριθμος μάθησης που χρησιμοποιήθηκε για εκπαίδευση του δικτύου είναι ο αλγόριθμος ανάστροφης μετάδοσης λάθους. Δηλαδή, για κάθε αμινοξύ, υπολογίζεται η έξοδος του όπως είχαμε προαναφέρει, αφού περάσει για επεξεργασία το κινητό παράθυρο στου οποίου βρίσκεται στην κεντρική θέση, από την κλάση *doFeedForward()*. Στην συνέχεια εκτελείται για το συγκεκριμένο αμινοξύ η κλάση *doBackPropagation()*. Σκοπός της είναι να υπολογίσει το σφάλμα στον κάθε νευρώνα εξόδου, χρησιμοποιώντας τα επιθυμητά αποτελέσματα της δευτεροταγούς δομής που θα έπρεπε να έχει το αμινοξύ, και να το μεταδώσει στα προηγούμενα επίπεδα, ώστε να επιτευχθεί η κατάλληλη ενημέρωση των βαρών του δικτύου σε κάθε επίπεδο. Αυτό γίνεται επειδή θέλουμε να ελαχιστοποιήσουμε το συνολικό σφάλμα σε συνάρτηση με το κάθε βάρος του δικτύου. Η συγκεκριμένη διαδικασία επιτυγχάνεται με την μέθοδο κατάβασης κλίσης. Κατά την διάρκεια του back propagation για το συγκεκριμένο αμινοξύ, το σφάλμα μεταδίδεται τόσο στο Ft

όσο και στο Bt δίκτυο αμφίδρομης ανάδρασης. Τα βάρη λοιπόν ενημερώνονται ακριβώς μια φορά για κάθε αμινοξύ, κάθε πρωτεΐνης.

Ας προχωρήσουμε λοιπόν να μελετήσουμε την υλοποίηση αλλά και τις σημαντικές λεπτομέρειες όσον αφορά το υπόλοιπο σύστημα. Οι διαδικασίες που αφορούν τους νευρώνες του δικτύου, διαχειρίζονται από την κλάση «Neuron.cpp». Εκεί υπάρχουν πολλές μέθοδοι που υλοποιούν στοιχεία του κάθε νευρώνα, αλλά και διαδικασίες που αφορούν τον αλγόριθμο μάθησης των νευρώνων (σχήμα 2.9). Σημαντικό είναι ότι η συνάρτηση ενεργοποίησης που χρησιμοποιείται για την έξοδο όλων των νευρώνων είναι η σιγμοειδής συνάρτηση. Στο αρχείο παραμέτρων, υπάρχει σχετική παράμετρος που μπορεί να τροποποιηθεί επιτρέποντας στο σύστημα να εκπαιδευτεί χρησιμοποιώντας μια διαφορετική συνάρτηση ενεργοποίησης. Εκτός αυτού, υπάρχει και μια παράμετρος η οποία αφορά τον τρόπο υπολογισμού του λάθους ενός νευρώνα και συμβαδίζει με την συνάρτηση ενεργοποίησης (πίνακας 2.1). Αυτό εξαρτάται και από το είδος του νευρώνα. Για τους εξωτερικούς νευρώνες, το λάθος όταν η συνάρτηση ενεργοποίησης είναι η σιγμοειδής αποτελείται από την παράγωγο της συνάρτησης ενεργοποίησης και το γενικό σφάλμα του νευρώνα. Το γενικό σφάλμα του νευρώνα είναι διαφορά της επιθυμητή με την πραγματική έξοδο του νευρώνα. Όσον αφορά τους κρυφούς νευρώνες το σφάλμα αποτελείται από την παράγωγο της συνάρτησης ενεργοποίησης του νευρώνα και το άθροισμα των γινομένων των βαρών επί το σφάλμα των νευρώνων του προηγούμενου επιπέδου.

```
Neuron ()
{
        Αρχικοποίησε τα βάρη του νευρώνα
        Αρχικοποίησε τις μεταβλητές του νευρώνα
}
calculateInput()
{
        Υπολόγισε την συνολική είσοδο του νευρώνα
}
callActivationFunction()
{
        Βρες την έξοδο του νευρώνα χρησιμοποιώντας την σιγμοειδή συνάρτηση
}
getOutput()
{
        Πάρε τις νέες εισόδους για τον νευρώνα
        calculateInput()
        callActivationFunction()
}
calculateOutputLayerError()
{
        Υπολόγισε το σφάλμα του νευρώνα αν είναι εξωτερικός
}
calculateHiddenLayerError()
{
        Υπολόγισε το σφάλμα του νευρώνα αν είναι κρυφός
}
adjustDw()
{
        Προσαρμογή βαρών του νευρώνα
}
```

Σχήμα 2.9*: Μερικές σημαντικές μέθοδοι που χρησιμοποιούνται στην κλάση «Neuron.cpp».*

2.4 Τροποποιήσεις και Βελτιστοποιήσεις BRNN

Σε μετέπειτα φάση το σύστημα επεξεργάστηκε και τροποποιήθηκε από τη Γεωργία Χριστοδούλου (2010), σαν μέρος της διπλωματικής εργασίας της. Η Χριστοδούλου έκανε κάποιες σημαντικές αλλαγές στο υπάρχων σύστημα που βοήθησαν στην καλύτερη πρόγνωση και βελτιστοποίησαν τα αποτελέσματα του Αγαθοκλέους. Ίσως η πιο σημαντική τροποποίηση ήταν η δυνατότητας εισαγωγής δεδομένων Πολλαπλής Στοίχιση Ακολουθιών (MSA) στο σύστημα και με αυτή θα απασχοληθούμε πρώτα.

2.4.1 Εισαγωγή Πολλαπλής Στοίχισης Ακολουθιών

Η Χριστοδούλου θέλησε να τροποποιήσει το δίκτυο ώστε να εκπαιδεύεται από πρωτεΐνες οι οποίες στοιχήθηκαν με κάποιες άλλες, έδωσαν κάποια αποτελέσματα και δημιούργησαν τέλος, ένα νέο προφίλ με τα χαρακτηριστικά της οικογένειας αυτής των πρωτεϊνών που ευθυγραμμίστηκε. Το μόνο δηλαδή καινούργιο στοιχείο που θα προστεθεί στο τρέχον δίκτυο θα είναι η δυνατότητα να επεξεργάζεται προφίλ πρωτεϊνών που πάρθηκαν με την χρήση MSA (Rost and Sander, 1993). Κατ' ακρίβεια αυτό που θα αλλάξει δεν θα είναι το σύνολο των πρωτεϊνών που χρησιμοποιήθηκαν μέχρι τώρα για εκπαίδευση και επαλήθευση, αλλά ο τρόπος κωδικοποίησης τους. Η μέχρι τώρα κωδικοποίηση των αμινοξέων που είχε χρησιμοποιήσει ο Αγαθοκλέους ήταν σταθερή για κάθε αμινοξύ και λέγεται *orthogonal encoding* (σχήμα 2.3). Προστέθηκε λοιπόν μια νέα επιλογή στις παραμέτρους του δικτύου που να του δίνει την δυνατότητα να εκπαιδεύεται με ένα διαφορετικό τρόπο κωδικοποίησης των αμινοξέων των πρωτεϊνών που θα δούμε πιο κάτω.

Η νέα κωδικοποίηση που πρέπει να παίρνει το δίκτυο, δεν είναι τόσο αυστηρή όσο η πρώτη, και παράλληλα προσθέτει περισσότερη πληροφορία στο δίκτυο. Αυτός είναι και ο κύριος λόγος που χρησιμοποιούμε την μέθοδο της πολλαπλής στοίχισης και των MSA profiles για εκπαίδευση του δικτύου.

Πρώτα γίνεται η πολλαπλή στοίχιση της πρωτεΐνης που εξετάζουμε με άλλες που έχουν σημαντική ομοιότητα με αυτήν. Υπενθυμίζουμε ότι ευθυγραμμίζουμε

ακολουθίες εξελικτικά σχετιζόμενων πρωτεϊνών, γιατί πιστεύεται ότι αφού έχουν εξελικτική σχέση, θα έχουν την ίδια δομή και στον χώρο (Rost and Sander, 1993). Έπειτα, σαν αποτέλεσμα αυτής της στοίχισης δημιουργείται το λεγόμενο MSA profile που χαρακτηρίζει την στοίχιση των συγκεκριμένων πρωτεϊνών, και γενικότερα μπορούμε να πούμε ότι χαρακτηρίζει την δομή τους. Άρα το MSA profile για κάθε οικογένεια πρωτεϊνών, αντιστοιχεί σε ένα σύνολο πρωτεϊνών που παρόλο που οι ακολουθίες τους διαφέρουν, δίνουν σε γενικές γραμμές την ίδια δομή στον χώρο.

Για την νέα κωδικοποίηση χρησιμοποιείται το MSA profile μιας οικογένειας πρωτεϊνών. Συγκεκριμένα, κωδικοποιείται η θέση του αμινοξέως και όχι το ίδιο το αμινοξύ, άρα η κωδικοποίηση του κάθε αμινοξέως σε κάθε πιθανή θέση της ακολουθίας αλλάζει. Η κωδικοποίηση της θέσης του αμινοξέως δείχνει την πιθανότητα εμφάνισης των 20 αμινοξέων σε εκείνη την θέση συμπεριλαμβανομένου και του αμινοξέως αυτού. Έτσι για κάθε θέση της ακολουθίας δίνουμε μια είσοδο πάλι 20 στοιχείων (κάθε γραμμή του profile), όπου κάθε αριθμητική τιμή αντιπροσωπεύει τον αριθμό εμφάνισης κάθε αμινοξέως. Η μεγαλύτερη τιμή κάθε στήλης είναι η στήλη που αντιστοιχεί στο αμινοξύ της ακολουθίας. Ένα πρότυπο της νέας κωδικοποίησης φαίνεται στο σχήμα 2.10.

	V	**L**	**I**	**M**	**F**	**W**	**Y**	**G**	**A**	**P**	**S**	**T**	**C**	**H**	**R**	**K**	**Q**	**E**	**N**	**D**
N	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	16	7	16	40	10
K	1	1	1	1	0	0	0	4	0	0	0	4	0	1	5	77	1	0	3	0
C	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0
P	1	1	0	0	1	0	2	22	5	48	4	2	1	7	1	2	0	1	1	1

Σχήμα 2.10*: Έστω ότι έχουμε την πρωτεΐνη 1bdoa_77-156. Η πρώτη γραμμή παρουσιάζει τα 20 αμινοξέα, και η πρώτη στήλη με bold γράμματα παρουσιάζει τα αμινοξέα της πρωτεΐνης όπως τα βρίσκουμε στην πρωτοταγή δομή. Η δεύτερη γραμμή παρουσιάζει τα αμινοξέα που πιθανόν να παρατηρηθούν στην πρώτη θέση, στην θέση δηλαδή του N αμινοξέως και άρα κωδικοποιεί την θέση του πρώτου αμινοξέως. Επομένως μπορούμε να δούμε για παράδειγμα στην δεύτερη γραμμή ότι τα αμινοξέα S,K,E,N παρουσιάστηκαν με το N να έχει την πλειοψηφία εμφανίσεων Το ίδιο γίνεται και για τις υπόλοιπες θέσεις των αμινοξέων.*

Αρχικά για την υλοποίηση των MSA τοποθετήθηκε ο φάκελος με τις νέες κωδικοποιήσεις των πρωτεϊνών (MSA profiles), τις οποίες πήρε μέσω του προγράμματος «msaProduction.pl» στον *EncodingProfiles* φάκελο του προγράμματος. Έτσι αντί να διαβάζει το αρχείο *sparse.txt* με την παλιά κωδικοποίηση, θα διαβάζει από τον φάκελο *msaProfiles* (βρίσκεται στο *EncodingProfiles* φάκελο), την κωδικοποίηση της πρωτεΐνης που ψάχνει κάθε φορά. Επιπρόσθετα, εκτός από τα αρχεία των MSA profiles, χρησιμοποίησε και ένα αρχείο *msa.txt* που θα δίνει πληροφορίες για το πρόγραμμα σχετικά με τον αριθμό των αμινοξέων που χρησιμοποιούνται (20 αμινοξέα, το X δεν χρειάζεται) και κατά πόσο η κωδικοποίηση αφορά το κάθε αμινοξύ ξεχωριστά (*residue_volume*). Το αρχείο αυτό θα αντικαθιστά το αρχείο *sparse.txt* όταν θέλουμε να εκπαιδεύσουμε το δίκτυο με την χρήση MSA profiles.

Αρκετές αλλαγές έγιναν και σε επίπεδο κώδικα στις διάφορες κλάσεις που απαρτίζουν το σύστημα. Κατ αρχήν πρόσθεσε ακόμα μια μεταβλητή στο αρχείο parameters.dat η οποία λέγεται *msaEnable*. Η συγκεκριμένη μεταβλητή ειδοποιεί το σύστημα ότι η κωδικοποίηση αλλάζει συνεπώς το σύστημα πρόκειται να εκπαιδευτεί με την χρήση MSA profiles.

Μια σημαντική αλλαγή σε επίπεδο κώδικα έγινε στην κλάση *DataReader.cpp*. Όταν δεν υπήρχε η χρήση MSA profiles και η κωδικοποίηση γινόταν βάση του αρχείου *sparse.txt*, η κλάση αυτή αρχικοποιούσε και κρατούσε σε ένα vector την κωδικοποίηση του κάθε πιθανού γράμματος (αμινοξέως) και η κωδικοποίηση αυτή δεν άλλαζε για κάθε αμινοξύ κάθε πρωτεΐνης. Στην περίπτωση χρήσης MSA profiles όμως, η κωδικοποίηση αυτή πρέπει να αλλάζει για κάθε πρωτεΐνη. Αυτό σημαίνει ότι το vector θα είναι ίσο με το μέγεθος της πρωτεΐνης και θα κρατά για μια τρέχουσα πρωτεΐνη την κωδικοποίηση της κάθε θέσης της, δηλαδή του αμινοξέως που βρίσκεται σε εκείνη την θέση. Αυτά θα παίρνονται από τα MSA αρχεία των πρωτεινών που πήραμε μέσω του προγράμματος «msaProduction.pl» . Έτσι υλοποιήθηκε μια μέθοδος η οποία θα εντοπίζει και θα διαβάζει την κωδικοποίηση κάθε πρωτεΐνης και θα την αποθηκεύει ολόκληρη σαν μια συμβολοσειρά σε αυτό το vector (Σχήμα 2.11).

```
void readEncodingMSA (encoding, nameOfProtein)
{
  Άνοιξε το αρχείο nameOfProtein
  While (x δεν είναι το τέλος του αρχείου)
  {
        Για κάθε αμινοξύ t
        {
          Διάβασε πιθανότητα του t για την θέση x
          Αποθήκευσε την πιθανότητα στο encoding
        }
  }
}
```

Σχήμα 2.11: *Ψευδοκώδικας για την μέθοδο readEncodingMSA.*

Σημαντικές αλλαγές διεκπεραιώθηκαν και στην κύρια κλάση του προγράμματος *BidirectionalRecurrentNeuralNetwork.cpp*, ώστε να μπορεί το δίκτυο να επεξεργάζεται την νέα κωδικοποίηση. Κατ' αρχήν αλλάχτηκε η υλοποίηση της μεθόδου *getInitialInput()*, η οποία είναι υπεύθυνη να αρχικοποιεί από κάποια αρχεία μεταβλητές του προγράμματος όπως είναι το μέγεθος της κωδικοποίησης. Χωρίς την μέθοδο πολλαπλής στοίχισης το μέγεθος της κωδικοποίησης ήταν 21, γιατί είχαμε 20 αμινοξέα συν το *X* αμινοξύ, το οποίο χρησίμευε για την κωδικοποίηση των ορίων μιας πρωτεΐνης. Για να γίνει αυτό χρησιμοποιήθηκε το αρχείο sparse.txt. Εφόσον λοιπόν, προστεθεί η επιλογή της χρήσης MSA profiles στην μέθοδο αυτή, το μέγεθος της κωδικοποίησης ισούται με τα αμινοξέα που λαμβάνουν μέρος σε αυτήν, δηλαδή στην περίπτωσή μας έχουμε 20 διαφορετικές πιθανότητες, μια για κάθε ένα από τα 20 αμινοξέα. Το μέγεθος των αμινοξέων καθορίζεται από το αρχείο *msa.txt* και συγκεκριμένα την παράμετρο *Residue_number*. Η κωδικοποίηση των ορίων της πρωτεΐνης γίνεται ακριβώς με τον ίδιο τρόπο, μέσω του προγράμματος, όπου εντοπίζει πότε έχουμε να κάνουμε με τα όρια της πρωτεΐνης, όπου για την κωδικοποίηση τους, εισαγάγει στο σύστημα μια ακολουθία από 20 μηδενικά.

Επίσης, δημιουργήθηκε η συνάρτηση *getMSAInput()* η οποία είναι υπεύθυνη να καλεί όποτε χρειάζεται μια συνάρτηση για την κωδικοποίηση μιας νέας πρωτεΐνης η οποία θα δίνεται σαν παράμετρος στην συνάρτηση αυτή. Η κωδικοποίηση της πρωτεΐνης δημιουργείται από την μέθοδο *readEncodingMSA()* της κλάσης *DataReader.cpp* (σχήμα 2.11).

Επίσης αναγκαία ήταν και η υλοποίηση της μεθόδου *createMSAtempInput()*, η οποία παίρνει σαν παράμετρο την θέση της πρωτεΐνης που εξετάζουμε και το μήκος της κωδικοποίησης. Το μήκος της κωδικοποίησης είναι πάντα ίσο με 20, όπου 20 είναι το πλήθος των αμινοξέων και έτσι δίνεται μια πιθανότητα για τα 20 αμινοξέα να εμφανιστούν στην συγκεκριμένη θέση. Γνωρίζοντας την κωδικοποίηση όπως την πήραμε από το αρχείο για την συγκεκριμένη θέση, λόγω του ότι είναι μια συμβολοσειρά, πρέπει πρώτα να την τροποποιήσουμε ώστε να πάρουμε τις τιμές από την συμβολοσειρά αυτή και μετά να δώσουμε το νέο vector για επεξεργασία στο δίκτυο. Έτσι, δημιουργεί μια προσωρινή είσοδο που αφορά την συγκεκριμένη θέση της πρωτεΐνης και το μεταδίδει για επεξεργασία. Αυτό θα γίνει για όλες τις θέσεις της πρωτεΐνης, δηλαδή για όλο το μήκος της (σχήμα 2.12).

```
void createMSAtempInput (position, length)
{
  νέα κωδικοποίηση = splitValues (encoding[position])
  Για κάθε αμινοξύ t της νέας κωδικοποίησης
  {
        Βάλε την τιμή στο input vector
  }
}
```

Σχήμα 2.12: *Ψευδοκώδικας για τη μέθοδο createMSAtempInput.*

Τώρα, όσον αφορά το πότε θα γίνεται αυτό, έχει τροποποιηθεί η κλάση *doFeedForward()*. Η *doFeedForward()* στην αρχή δημιουργεί την είσοδο του δικτύου ανάλογα με το παράθυρο των αμινοξέων που πρέπει να ικανοποιεί. όπου αντί να παίρνει σαν είσοδο την αρχική κωδικοποίηση, παίρνει το vector που δημιουργήθηκε, με τις τιμές των 20 αμινοξέων για την συγκεκριμένη θέση της πρωτεΐνης που εξετάζουμε (σχήμα 2.13).

```
Για κάθε αμινοξύ του window
{
  Δημιουργία κωδικοποίησης:
  Για το αμινοξύ position
  {
        αν MSA ==0
              input = κωδικοποίηση position βάση sparse.txt
        αλλιώς
              input = createMSAInput (position,20)
  }
}
```

Σχήμα 2.13: *Ψευδοκώδικας εισαγωγής MSA κωδικοποίησης.*

Μικρές αλλαγές έγιναν στην κλάση *pssp_ucy.cpp*. Χωρίς την χρήση MSA profiles στο σύστημα, η κωδικοποίηση ήταν για κάθε αμινοξύ κάθε πρωτεΐνης σταθερή και έτσι παίρναμε την κωδικοποίηση μια φορά μόνο για όλες τις πρωτεΐνες. Αντιθέτως με την χρήση MSA profiles, πριν να αρχίσει η επεξεργασία των αμινοξέων χρειάζεται να ξέρουμε για κάθε πρωτεΐνη την κωδικοποίηση της. Έτσι λοιπόν προστέθηκε μια εντολή όπου θα πρέπει να παίρνει την κωδικοποίηση και να την αποθηκεύει στο συγκεκριμένο vector. Μετά εκτελούνται οι δύο βασικές συναρτήσεις *doFeedForward()* και *doBackpropagation()* (σχήμα 2.14).

```
While (epoch<maxEpoch)
{
    While (υπάρχουν πρωτεΐνες στο training dataset)
    {
        Πάρε την επόμενη πρωτεΐνη x
        Πάρε την κωδικοποίηση της πρωτεΐνης x
        Για κάθε αμινοξύ t της πρωτεΐνης x
        {
            doFeedForward (t)
            doBackPropagation (t)
        }
      While (υπάρχουν πρωτεΐνες στο testing dataset)
      {
        Πάρε την επόμενη πρωτεΐνη x
        Πάρε την κωδικοποίηση της πρωτεΐνης x
        Για κάθε αμινοξύ t της πρωτεΐνης x
        {
            doFeedForward (t)
        }
    }
}
```

Σχήμα 2.14: *Ψευδοκώδικας με τον οποίο τροποποιείται η κλάση pssp.ucy, ώστε να επεξεργάζεται την χρήση των MSA profiles.*

2.4.2 Εισαγωγή SOV τρόπου βαθμολόγησης του Δικτύου

Μια άλλη σημαντική εισαγωγή στου σύστημα που είχε κάνει η Χριστοδούλου ήταν η ενσωμάτωση του SOV Score.

SOV (Segment Overlap), είναι μια νέα μέθοδος βαθμολόγησης της προβλεπόμενης ακολουθίας της δευτεροταγούς δομής η οποία προτάθηκε αρχικά από τον Rost και τους συνεργάτες του (Rost et al., 1994) και επαναπροσδιορίστηκε από τους Zemla et al (1999).

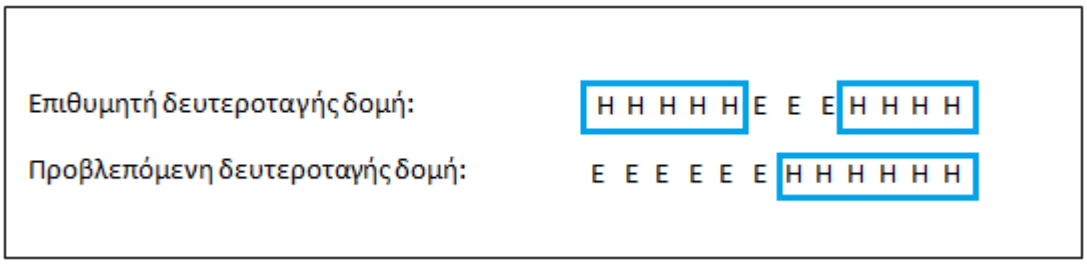
Μέχρι τώρα η μέθοδος βαθμολόγησης της ακρίβειας πρόβλεψης δευτεροταγούς δομής μιας πρωτεΐνης από το υφιστάμενο σύστημα είναι η μέθοδος Q3. Συγκεκριμένα η Q3 είναι μια μέθοδος αξιολόγησης του πρόβλεψης δευτεροταγούς δομής πρωτεϊνών που συγκρίνει ένα προς ένα τα αμινοξέα που συνθέτουν την πρωτεΐνη υπολογίζοντας πόσα από αυτά πρόβλεψε σωστά, χρησιμοποιώντας την Εξίσωση 2.1. Όμως όπως θα δούμε στη συνέχεια αυτή η μέθοδος δεν είναι τόσο σωστή από βιολογικής απόψεως.

$$Q3 = \frac{\text{number of residues correctly predicted}}{\text{number of all residues}} * 100$$

Εξίσωση 2.1: *Η μέθοδος βαθμολόγησης της πρόβλεψης του δικτύου, η οποία βασίζεται στον αριθμό των καταλοίπων που προβλέφτηκαν σωστά.*

Το SOV είναι μια άλλη μέθοδος βαθμολόγησης της προβλεπόμενης ακολουθίας η οποία είναι πολύ πιο βιολογικά ακριβής από το Q3. Αυτό ισχύει επειδή δεν συγκρίνει ένα προς ένα κατάλοιπο, αλλά διαστήματα από κατάλοιπα που επικαλύπτονται στις δύο ακολουθίες της προβλεπόμενης και πραγματικής ακολουθίας αντίστοιχα. Αυτό είναι πολύ σημαντικό, γιατί αυτά τα κομμάτια θέσεων είναι αυτά που καθορίζουν την «δομή» της πρωτεΐνης στον χώρο (τριτοταγούς δομής). Δεν συγκρίνει μια προς μια θέση και αυτή είναι όπως θέσαμε πιο πάνω, η βασική διαφορά με το Q3. Για παράδειγμα, εάν η πρωτεΐνη στον χώρο αποτελείται από δύο κομμάτια Helices ενώ η προβλεπόμενη ακολουθία προβλέπει μόνο ένα

Helix διάστημα (σχήμα 2.15), αυτό σημαίνει ότι υπάρχει πολύ μεγάλη διαφορά στην τριτοταγή δομή, όσον αφορά το σχήμα δηλαδή της πρωτεΐνης. Παρόλα αυτά, με βάση και πάλι το σχήμα 2.15, η προβλεπόμενη δευτεροταγής δομή με την μέθοδο βαθμολόγησης Q3 έχει ένα μεγάλο ποσοστό ακριβείας 75%. Από αυτό καταλαβαίνουμε ότι οι δύο μέθοδοι βαθμολόγησης μπορούν να δώσουν ακραία αποτελέσματα, και αυτό ακριβώς γίνεται επειδή η θεωρία την οποία εφαρμόζουν είναι πολύ διαφορετική. Η νέα όμως μέθοδος βαθμολόγησης, φαίνεται να είναι ρεαλιστικά πιο ακριβής από την Q3.



Σχήμα 2.15: *Η επιθυμητή δευτεροταγής δομή αποτελείται από δύο τμήματα Η, ενώ η προβλεπόμενη δευτεροταγής δομή αποτελείται μόνο από ένα Η. Δομικά λοιπόν υπάρχει μια πολύ μεγάλη διαφορά στις δύο αυτές ακολουθίες.*

Με βάση τις εξισώσεις υπολογισμού SOV (Zemla et al., 1999) όπως φαίνεται στην εξίσωση 2.2, (s1,s2) δηλώνει ένα ζεύγος από τμήματα που εξετάζονται, όπου s1 ανήκει στην επιθυμητή δευτεροταγή δομή και s2 ανήκει στην προβλεπόμενη δευτεροταγή δομή. Η τομή των δύο αυτών τμημάτων πρέπει να περιέχει τουλάχιστον ένα κοινό στοιχείο και η ένδειξη του αριθμού των κοινών στοιχείων καθορίζεται από την τιμή minov(s1,s2). Αντίθετα, η τιμή maxov(s1,s2) καθορίζει την ένωση αυτών των δύο τμημάτων (εξίσωση 2.2). Επιπλέον η τιμή δ(s1,s2) καθορίζει μια θετική τιμή ίση με το αποτέλεσμα της εξίσωσης 2.2 (της δεύτερης εξίσωσης), όπου len(s1) το μήκος του s1 τμήματος και len(s2), το μήκος του s2 τμήματος.

$$\left[\frac{1}{N}\sum_{i\in\{H,E,C\}}\sum_{S(i)}\frac{\text{minov}(s_1, s_2) + \delta(s_1, s_2)}{\text{maxov}(s_1, s_2)} \times len(s_1)\right]$$

$$\delta(s_1, s_2) = \min[(\text{maxov}(s_1, s_2) - \text{minov}(s_1, s_2)),$$
$$\text{minov}(s_1, s_2), \text{int}(\text{len}(s_1)/2), \text{int}(\text{len}(s_2)/2)]$$

Εξίσωση 2.2: *Οι εξισώσεις υπολογισμού της νέας μεθόδου βαθμολόγησης προβλεπόμενων ακολουθιών, SOV (Από Zemla et al., 1999).*

Το εσωτερικό άθροισμα (εξίσωση 2.2) γίνεται για κάθε πιθανό ζεύγος τμημάτων (s1,s2) που αφορούν μια συγκεκριμένη δευτεροταγή δομή. Το εξωτερικό άθροισμα, γίνεται για κάθε δευτεροταγή δομή που μελετούμε, δηλαδή στην προκειμένη περίπτωση, το εσωτερικό άθροισμα θα εκτελεστεί 3 φορές, μια για όλα τα τμήματα που αφορούν Helices (H), μια για όλα τα τμήματα που περιέχουν Strand (E) και μια για όλα τα τμήματα που έχουν Coils ή Loops (C ή L).

$$N(i) = \sum_{S(i)} len(s_1) + \sum_{S'(i)} len(s_1)$$

Εξίσωση 2.3: *Μέθοδος υπολογισμού του όρου N(i) (Από Zemla et al., 1999).*

Η τιμή N (εξίσωση 2.3), για κάποια δευτεροταγή δομή (i), δίνει το άθροισμα όλων των τιμών len(s1) της δευτεροταγής δομής που μελετούμε και που σχηματίζουν επικαλυπτόμενα τμήματα με την προβλεπόμενη ακολουθία, αλλά και την τιμή len(s1) τμημάτων της επιθυμητής ακολουθίας, που δεν σχηματίζουν επικαλυπτόμενα τμήματα με την προβλεπόμενη ακολουθία.

Για να δούμε την σημαντικότητα αυτής της νέας μετρικής ας δούμε το πιο πάνω παράδειγμα του σχήματος 2.16, το οποίο μελετά το SOV για τα Helices. Έστω η ακολουθία έχει το πιο πάνω τμήμα από Helices. Στην πρώτη πρόβλεψη, 5 από τα 10 στοιχεία της ακολουθίας έχουν προβλεφτεί σωστά, και λογικά, με την Q3 μετρική αναμένουμε ένα σκορ ανάλογο με αυτό που βλέπουμε. Όμως, με την μέθοδο SOV, το σκορ αυτό μειώνεται κατά πολύ. Αντιθέτως, στην δεύτερη πρόβλεψη το Q3 ποσοστό είναι το ίδιο αφού η ποσότητα των στοιχείων που έχουν προβλεφτεί σωστά είναι η ίδια με την προηγούμενη πρόβλεψη, όμως το SOV Score αλλάζει δραματικά. Αυτό οφείλεται στο ότι το SOV δείχνει ότι η δομή της πρωτεΐνης με βάση την δεύτερη πρόβλεψη είναι πιο κοντά στην πραγματική, επειδή υπάρχει μόνο ένα τμήμα Helix, όπως συμβαίνει στην πραγματικότητα, και όχι πέντε όπως προβλέπει η πρώτη πρόβλεψη. Με το ίδιο σκεπτικό μπορούμε να κρίνουμε τα αποτελέσματα Q3 και SOV για τις υπόλοιπες προβλέψεις. Η πέμπτη πρόβλεψη, δίνει το υψηλότερο ποσοστό όσον αφορά την SOV μετρική, επειδή εκτός του ότι προβλέπει ένα τμήμα Helix, έχει και σημαντική ομοιότητα με τα στοιχεία του, ενώ το Q3 δίνει τα υψηλότερα ποσοστά για την τρίτη και τέταρτη πρόβλεψη όπου μπορεί η ποσότητα των στοιχείων που προβλέφτηκαν σωστά να είναι μεγαλύτερη από τις άλλες προβλέψεις, αλλά, η δομή της πρωτεΐνης είναι πολύ διαφορετική από την πραγματικότητα.

		Sov	Q_3
Observed	CHHHHHHHHHHC		
Prediction 1	CHCHCHCHCHCC	12.5	58.3
Prediction 2	CCCHHHHHCCCC	63.2	58.3
Prediction 3	CHHHCHHHCHHC	40.6	83.3
Prediction 4	CHHCCHHHHHCC	52.3	75.0
Prediction 5	CCCHHHHHHCCC	80.6	66.7

Σχήμα 2.16: *Τυπικό παράδειγμα σύγκρισης δύο μεθόδων βαθμολόγησης πρόβλεψης πρωτεϊνικών ακολουθιών (Από Zemla et al., 1999).*

Το πρόγραμμα το οποίο υπολογίζει αυτήν την νέα μετρική SOV, είναι ήδη υλοποιημένο και διαθέσιμο (SOV, 23 April 2010), και χρησιμοποιήθηκε χωρίς καμία αλλαγή. Το μόνο που αλλάχτηκε είναι η μορφή των αρχείων που δίνει το τρέχον σύστημα, ώστε να είναι συμβατά με την μορφή των αρχείων που παίρνει σαν είσοδο το πρόγραμμα SOV.

Το πρόγραμμα SOV, παίρνει σαν είσοδο δύο ακολουθίες, μια ακολουθία που εκφράζει την προβλεπόμενη δευτεροταγή δομή κάποιας πρωτεΐνης και μια η οποία εκφράζει την πραγματική δευτεροταγή δομή της πρωτεΐνης αυτής. Στην συνέχεια υπολογίζει το SOV ποσοστό για κάθε ένα από τα H, E και L χρησιμοποιώντας τις πιο πάνω μαθηματικές εξισώσεις (εξίσωση 2.2, 2.3), καθώς και το ολικό SOV ποσοστό για την συγκεκριμένη πρωτεΐνη. Επίσης μια επιπλέον πράξη που κάνει είναι ο υπολογισμός του Q3, αλλά υπολογίζει επίσης τα Q_H, Q_E και Q_L ξεχωριστά. Το ανάλογο αρχείο εξόδου που δίνει το συγκεκριμένο πρόγραμμα αποτελείται από τις δύο ακολουθίες εισόδου, και από τις πληροφορίες SOV και Q3 (σχήμα 2.17).

```
 .
 .
 .
 X   H     H       167
 X   H     H       168
 X   H     H       169
 X   C     H       170
 X   C     H       171
 X   H     H       172
 X   H     H       173
 X   H     H       174
 X   H     H       175
 X   C     C       176
 X   C     C       177
 X   C     C       178
 X   C     C       179
-----------------------

SECONDARY STRUCTURE PREDICTION ACCURACY EVALUATION.  N_AA =  179

                                  ALL    HELIX   STRAND     COIL

Q3                          :     64.2    90.1     12.9     50.9

SOV                         :     64.6    83.9     19.4     57.8
```

Σχήμα 2.17: *Τυπικό αρχείο εξόδου για το πρόγραμμα SOV, που δείχνει τις μετρικές Q3 και SOV, και για όλη την πρωτεΐνη, και για τα H, E και L ξεχωριστά.*

Για τις ανάγκες του συστήματος η Χριστοδούλου χρησιμοποίησε επίσης το πρόγραμμα «confusionMatrices.pl» το οποίο τροποποιήθηκε και μετονομάστηκε σε «confusionMatricesAndSOV.pl». Το πρόγραμμα αυτό χρησιμοποιήθηκε για να κατασκευαστούν τα confusion matrices κάθε πρωτεΐνης ούτως ώστε να εξετάσει στατιστικά η ικανότητα πρόβλεψης του δικτύου για όλες τις πιθανές τιμές δευτεροταγούς δομής H, E και L.

Τα confusion matrices είναι ένα είδος πίνακα όπως φαίνεται από το σχήμα 2.18, το οποίο χρησιμοποιείται συχνά για εκτίμηση της εξόδου των συνόλων δεδομένων που εκπαιδεύτηκαν με την χρήση επιβλεπόμενης μάθησης, όταν το σύνολο δεδομένων δεν είναι ισοζυγισμένο. Οριζόντια φαίνεται η προβλεπόμενη τιμή των συνόλων δεδομένων που δίνει το δίκτυο και κάθετα η πραγματική τιμή που έχουν.

Πραγματική τιμή

Προβλεπόμενη τιμή	H	E	L
H	10	2	8
E	1	6	2
L	0	1	1

Σχήμα 2.18: *Ένα τυπικό παράδειγμα ενός confusion matrix πρόβλεψης δευτεροταγούς δομής μιας πρωτεΐνης με χρήση νευρωνικών δικτύων αμφίδρομης ανάδρασης.*

Όσον αφορά το τρέχον σύστημα, ο πίνακας αποτελείται από τα τρία είδη δευτεροταγούς δομής που μελετούμε. Έστω ότι το σχήμα 2.18 δείχνει το confusion matrix μιας πρωτεΐνης της οποίας η δευτεροταγής δομή προβλέφτηκε από το σύστημα. Από τα 11 λοιπόν «H» που υπάρχουν στην πραγματική δευτεροταγή δομή, τα 10 απ' αυτά τα πρόβλεψε σαν H και μόνο το ένα από αυτά προβλέφτηκε σαν «E». Αυτό σημαίνει ότι το δίκτυο μπορεί και προβλέπει σωστά την ομάδα των Helixes. Συνεχίζοντας, από τα 9 «E» που υπάρχουν στην δευτεροταγή ακολουθία το δίκτυο κατάφερε να προβλέψει τα δυο σαν «H» και το ένα σαν «L» και από τα 11

«L» που υπάρχουν μόνο το ένα από αυτά προβλέφτηκε σωστά. Από το τελευταίο αυτό σημείο καταλαβαίνουμε ότι έχουμε πρόβλημα όσον αφορά την τρίτη περίπτωση, δηλαδή την περίπτωση που αφορά την πρόβλεψη του δικτύου για τα «L». Το δίκτυο δεν μπορεί να προβλέψει σωστά την ομάδα «L», και από ότι βλέπουμε στο παράδειγμα, συγχύζει την ομάδα αυτή με την ομάδα των «H».

Επομένως ήταν πολύ σημαντικό να δημιουργηθούν τα confusion matrices για τις πρωτεΐνες που εκπαιδεύονται και επαληθεύονται με το τρέχον σύστημα, γιατί είναι ένας τρόπος αξιολόγησης όσον αφορά τις ξεχωριστές ομάδες της δευτεροταγούς δομής που προβλέπει, και να εντοπιστούν τυχόν συγχύσεις που μπορεί να γίνονται μεταξύ τους.

Για αυτό το λόγο χρησιμοποίησε το πρόγραμμα «confusionMatrices.pl» το οποίο είναι γραμμένο στη γλώσσα PERL και υπολογίζει όλα τα confusion matrices. Επίσης όπως προαναφέραμε το πρόγραμμα τροποποιήθηκε και μετονομάστηκε σε «confusionMatricesAndSOV.pl» ούτως ώστε να υπολογίζει τα confusion matrices και τις απαραίτητες πληροφορίες για τις προβλεπόμενες ακολουθίες του συστήματος για κάθε πρωτεΐνη, χρησιμοποιώντας και τη νέα μέθοδο αξιολόγησης του συστήματος, SOV.

Από το σχήμα 2.19 βλέπουμε ότι στη ουσία πρόσθεσε τρείς διαφορετικές υπορουτίνες στο πρόγραμμα . Η πρώτη υπορουτίνα, *createTempFile* είναι υπεύθυνη για την δημιουργία ενός προσωρινού αρχείου, στην μορφή των αρχείων που δέχεται το πρόγραμμα SOV, για κάθε πρωτεΐνη. Ακολούθως θα τρέχει το πρόγραμμα SOV δίνοντας του σαν είσοδο το αρχείο αυτό. Από αυτό το αρχείο παίρνουμε μόνο τις δύο τελευταίες γραμμές οι οποίες αφορούν τις μετρικές με το Q3 και SOV αντίστοιχα. Αυτές οι τιμές διαβάζονται και αποθηκεύονται στο πρόγραμμα, με βάση την υπορουτίνα *readFromSOV*. Η τελευταία υπορουτίνα που ονομάζεται *printInformationToFile.* είναι υπεύθυνη να τυπώνει τις πληροφορίες αυτές σε δύο διαφορετικά αρχεία. Το πρώτο αρχείο είναι ένα γενικό αρχείο για το συγκεκριμένο σύνολο πρωτεϊνών που μελετούμε, και όλες οι πληροφορίες της κάθε πρωτεΐνης αποθηκεύονται εκεί. Επίσης, δημιουργεί ένα αρχείο για κάθε ξεχωριστή

πρωτεΐνη, όπου και πάλι αποθηκεύει τα χαρακτηριστικά της, το confusion matrix της, αλλά και τις τιμές Q3, SOV ολικό και SOV για κάθε δευτεροταγή δομή.

```
Δώσε όνομα αρχείου εισόδου
Για κάθε πρωτεΐνη x του αρχείου εισόδου
{
        Αποθήκευσε επιθυμητή δευτεροταγή δομή
        Αποθήκευσε πραγματική δευτεροταγή δομή
        Για κάθε θέση i της πρωτεΐνης
        {
                Πάρε δευτεροταγή δομή z της επιθυμητής ακολουθίας
                Πάρε δευτεροταγή δομή k της πραγματικής ακολουθίας
                Σύγκρινε z και k
                Αποθήκευσε το αποτέλεσμα στον πίνακα
        }
        Κάλεσε υπορουτίνα createTempFile()
        Κάλεσε υπορουτίνα readFromSOV()
        Κάλεσε υπορουτίνα printInformationToFile()
}
```

Σχήμα 2.19: *Η αλλαγή στο προηγούμενο πρόγραμμα «confusionMatrices.pl», το οποίο μετονομάστηκε σε «confusionMatricesAndSOV.pl» είναι οι τρείς τελευταίες γραμμές.*

Η μορφή ενός τυπικού αρχείου μιας πρωτεΐνης που δίνει σαν έξοδο το πρόγραμμα «confusionMatricesAndSOV.pl», φαίνεται στο σχήμα 2.20 ενώ η μορφή του γενικού αρχείου για ολόκληρο το σύνολο πρωτεϊνών που σχηματίζεται από το πρόγραμμα, φαίνεται στο σχήμα 2.21. Σημαντικό είναι να αναφέρουμε ότι το πρόγραμμα αυτό στο τέλος της εκτέλεσης του, υπολογίζει τον μέσο όρο των τιμών των confusion matrices για τις πρωτεΐνες του συγκεκριμένου συνόλου πρωτεϊνών που χρησιμοποιήθηκε. Αυτό είναι μια *αντιπροσωπευτική* πληροφορία για το συγκεκριμένο σύνολο δεδομένων το οποίο επεξεργαζόμαστε, και με βάση αυτά τα αποτελέσματα θα δείξουμε τα ποσοστά σε διάφορα πειράματα. Οι πληροφορίες αυτές εμφανίζονται στην γραμμή εντολών στο τέλος της εκτέλεσης του προγράμματος.

```
ProteinName            :1gl2D

PrimaryStructure             :DAGLDALSSIISRQKQMGQEIGNELDEQNEIIDDLANLVENTDEKLRTEARRVI
SecondaryStructure           :LHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHI
PredictedSecondaryStructure :LHHHHHHHHHHHHHHHHHHHLHHHHHHHHHHLLHHHLLLLLLHHHLHHHHLLI


Q3       SOV(all)    SOV(H)      SOV(E)      SOV(L)
-----------------------------------------------
77.8      23.0        22.7        100.0       66.7



Confusion Matrix:   HH    HE    HL       EH    EE    EL       LH    LE    LL
                    ---------------------------------------------------------
                    40    0     12       0     0     0        0     0     2
```

Σχήμα 2.20: *Τυπικό αρχείο εξόδου από το πρόγραμμα «confusionMatricesAndSOV.pl» για κάθε μια από τις πρωτεΐνες εισόδου.*

Protein	Q3	SOV(all)	SOV(H)	SOV(E)	SOV(L)	HH	HE	HL	EH	EE	EL	LH	LE	LL
1z3xA	80.1	76.2	76.0	0.0	77.9	117	1	15	0	0	2	29	0	72
2hjmC	74.4	43.7	42.6	100.0	51.7	60	3	7	0	0	0	13	0	7
2prfA	64.8	56.9	85.8	15.6	63.4	26	0	1	9	4	13	21	0	51
7a3hA	65.0	62.4	71.4	26.0	70.0	84	0	30	27	13	17	29	2	98
2cprA	75.8	58.7	87.5	100.0	45.9	53	0	3	0	0	0	27	0	41
1bshA	55.1	42.8	51.1	26.4	47.7	29	0	0	11	9	18	30	3	38
3eagA	68.5	63.1	91.5	19.0	66.0	89	1	16	19	12	48	18	0	121
2ccmA	80.6	49.6	71.4	100.0	39.2	50	0	6	0	0	0	14	4	50
1kq1A	43.3	38.7	88.9	6.2	56.4	14	0	1	15	2	15	3	0	10
1nkqA	54.3	43.9	55.3	24.8	52.9	37	0	16	30	18	36	30	1	79
2jdxA	62.1	52.5	63.0	22.9	58.8	86	0	27	27	17	39	40	3	120
2dktA	51.7	41.4	54.0	0.0	50.2	8	0	4	9	0	26	30	0	66
2b51C	60.3	43.2	43.3	17.5	48.0	17	0	5	19	5	16	28	1	83
1vyxA	66.7	60.3	54.0	0.0	67.9	8	0	6	2	0	4	8	0	32
1mb6A	48.6	33.1	100.0	9.1	42.5	0	0	0	3	1	7	8	0	16
2ec2E	58.1	45.0	55.6	23.5	40.5	47	0	6	10	6	18	20	0	22
1aj4A	80.6	77.0	78.0	0.0	77.5	78	0	13	1	0	1	16	0	51
1vccA	54.5	53.7	83.5	28.8	59.8	10	0	1	10	5	11	12	1	27
1gl2D	77.8	23.0	22.7	100.0	66.7	40	0	12	0	0	0	0	0	2
1jkwA	83.0	77.3	82.3	100.0	66.7	168	2	17	0	0	0	27	1	62
2f07B	81.1	63.9	71.7	0.0	48.6	129	2	10	1	0	1	23	0	30

Σχήμα 2.21: *Τυπικό γενικό αρχείο εξόδου από το πρόγραμμα «confusionMatricesAndSOV.pl» για όλες τις πρωτεΐνες εισόδου.*

Κεφάλαιο 3

Επεξεργασία Δεδομένων

3.1 Χρήση CB513

3.2 Επεξεργασία δεδομένων Εξόδου του Συστήματος

3.1 Χρήση CB513

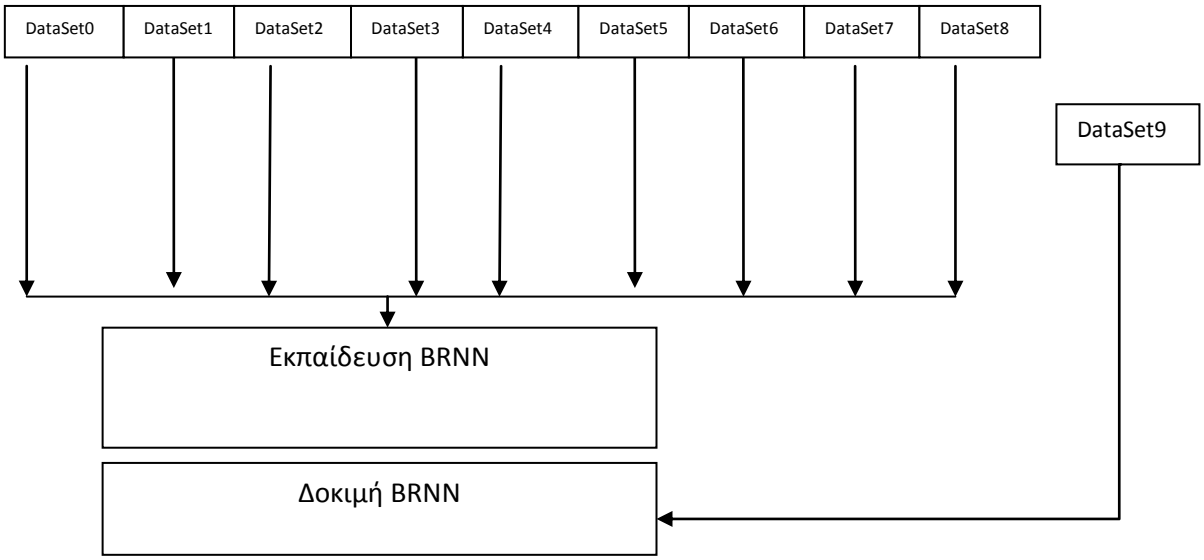
Η πρώτη τροποποίηση που έγινε στο σύστημα αφορούσε το σύνολο των δεδομένων που εισάγονται στο σύστημα για εκπαίδευση και δοκιμή. Τα δεδομένα εισόδου στο υπάρχον σύστημα, όπως προαναφέρθηκ,ε είναι χωρισμένα σε δεδομένα εκπαίδευσης και δεδομένα δοκιμής και τα οποία βρίσκονται στο φάκελο *TrainingSet* και *TestSet*. Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο η επιλογή σωστών συνόλων δεδομένων είναι πολύ σημαντική στη καλή πρόβλεψη του δικτύου. Η επιλογή των πρωτεϊνών που θα χρησιμοποιήσουμε παίζει μεγάλο ρόλο στη μετέπειτα εκπαίδευση. Αν επιλεχθούν πρωτεΐνες που όταν στοιχιστούν παρουσιάζουν μεγάλη ομοιότητα μεταξύ τους , τότε θα έχουμε μειωμένα ποσοστά πρόβλεψης. Μέχρι σήμερα έχουν δημιουργηθεί διάφορα σύνολα δεδομένων που χρησιμοποιήθηκαν για τη πρόβλεψη της δευτεροταγούς δομής πρωτεϊνών. Η χρήση διαφορετικών συνόλων έχει σαν αποτέλεσμα να είναι δύσκολη η σύγκριση μεταξύ διαφορετικών μεθόδων πρόβλεψης. Αποφασίσαμε λοιπόν να χρησιμοποιήσουμε ένα σύνολο το δεδομένων το οποίο είναι ευρεία γνωστό και αναγνωρίσιμο, για να μπορούμε να συγκρίνουμε αποτελεσματικά τα ποσοστό μας με ποσοστά άλλων εφαρμογών. Το σύνολο που αποφασίσαμε να χρησιμοποιήσουμε είναι το *CB513*.

Το *CB513* είναι ένα σύνολο που δημιουργήθηκε από τους Cuff και Barton (1999) και αποτελείται από 513 πρωτεΐνες. Αποτελεί το συνδυασμό των συνόλων RS126 και CB554, τα οποία παρουσίαζαν κάποια σημαντικά μειονεκτήματα. Το RS126 δημιουργήθηκε από τους Rost και Sander (1993) και αποτελείται από 126 πρωτεΐνες. Οι πρωτεΐνες επιλέχθηκαν βάση του κανόνα ότι μεταξύ 2 πρωτεϊνών με περισσότερα από 80 αμινοξέα στα δεδομένα δεν πρέπει να υπάρχει ομοιότητα πέραν του 25%. Όταν όμως χρησιμοποιήθηκαν πιο ακριβείς μέθοδοι σύγκρισης παρατηρήθηκαν ότι οι πρωτεΐνες παρουσιάζουν μεγαλύτερη ομοιότητα. Για αυτό το λόγο οι Cuff και Barton (1999) δημιούργησαν αρχικά ένα σύνολο από πρωτεΐνες που επιλέχθηκαν από τη βάση δεδομένων *3Dee* με τη χρήση ευαίσθητων αλγορίθμων σύγκρισης ακολουθιών και τεχνικών *clustering* αντί της απλής σύγκρισης ομοιότητας. Το σύνολο αυτό ονομάστηκε *CB554* και όπως φαίνεται από το όνομα του αποτελείται από 554 πρωτεΐνες.

Για να επιβεβαιωθεί ότι το *CB554* δεν παρουσιάζει ομοιότητες με το *RS126,* τα δύο σύνολα συνδυάστηκαν και συγκρίθηκαν με τη μέθοδο AMPS (Alignment of Multiple Protein Sequences) (Bardon, 1994), με πίνακα *blosum62* και *penalty gap* 10. Στοιχίσεις με SD Score μεγαλύτερο του 5, θεωρήθηκαν ομόλογες και για αυτό οι συγκεκριμένες πρωτεΐνες αφαιρέθηκαν από το σύνολο. Έτσι δημιουργήθηκε το τελικό dataset, το οποίο αποτελείται απ 513 πρωτεΐνες και ονομάζεται CB513. Στο CB513 υπάρχουν συνολικά 73253 κατάλοιπα.

Για τις ανάγκες του προβλήματος μας πραγματοποιήσαμε 10-fold cross-validation σε 10 υποσύνολά. Έτσι όταν πρόκειται να τρέξουμε ένα πείραμα, εκπαιδεύουμε το σύστημα για τα 9 dataset και δοκιμάζουμε το υπολειπόμενο ένα (Σχήμα 3.1). Η διαδικασία αυτή πρέπει να επαναληφθεί και για τα 10 dataset. Το τελικό ποσοστό πρόβλεψης του δικτύου υπολογίζεται συνδυάζοντας τα αρχεία που χρησιμοποιήθηκαν ως TestSet και στα 10 υποσύνολα. Δηλαδή ενώνοντας τα αποτελέσματα και των 10 folds σε ένα αρχείο για να υπολογιστεί το τελικό αποτέλεσμα.

Έχοντας πλέον ένα διαδεδομένο σύνολο δεδομένων μπορούμε να μιλούμε και να συγκρίνουμε τα αποτελέσματα μας με άλλα σύστημα που έχουν αναπτυχθεί χωρίς να υπάρχει πρόβλημα σύγκρισης.



Σχήμα 3.1: *Παράδειγμα Τρόπου Εκπαίδευση και Δοκιμής του Συστήματός.Στο συγκεκριμένο παράδειγμα εκπαιδεύουμε για τα DataSet από 0 μέχρι 8 και δοκιμάζουμε το DataSet9.*

3.2 Επεξεργασία δεδομένων Εξόδου του Συστήματος

Λόγων των νέων τροποποιήσεων που έγιναν στο πρόγραμμα (Υποκεφάλαιο 4.2) αυξήθηκε η πολυπλοκότητα και άρα ο χρόνος εκπαίδευσης το συστήματος. Για δίκη μας ευκολία τροποποιήσαμε το πρόγραμμα ούτως ώστε να δίνει ο χρήστης ρητά μέσα από το αρχείο παραμέτρων τον όνομα του φακέλου που θα αποθηκεύονται τα αρχεία εξόδου. Από το σχήμα 3.2 παρατηρούμε ότι η τελευταία γραμμή έχει να κάνει με το όνομα το αρχείο που θα αποθηκεύονται.

```
Hidden_layer_one_size 15
Hidden_layer_two_size 0
Hidden_layer_one_of_Backward_size 17
Hidden_layer_two_of_Backward_size 0
Hidden_layer_one_of_Forward_size 17
Hidden_layer_two_of_Forward_size 0
Activation_Function_Type_Hidden 1
Activation_Function_Type_Output 1
Learning_Rate 0.09
Momentum 0.5
Window_size 31
q_minus_one 0.6
q_plus_one 0.6
Error_Function_Type_Hidden 1
Error_Function_Type_Output 1
S 3
Maximum_Iterations 300
/**********InputFiles**********/
input_Profile msa.txt
output_Profile threeClasses.txt
train_File trainSet0
test_File testSet0
/*********OtherParams*********/
msa_Enable 1
randomizeDataset 1
center_window_size 12
nameofOutPutFile Experiment02
```

Σχήμα 3.2: *Περιεχόμενο αρχείο parameters.dat. Παρατηρούμε ότι στη τελευταία γραμμή έχει προστεθεί ακόμη μια παράμετρος που αφορά το όνομα του αρχείου στο οποίο θα αποθηκεύονται τα αποτελέσματα του πειράματος που θα τρέξουμε.*

Αυτή η αλλαγή βοήθησε στην καλύτερη ταξινόμηση και πιο εύκολη αποθήκευση των δεδομένων μας. Στο προηγούμενο σύστημα το όνομα κάθε αρχείο εξόδου στο τέλος είχε την ημερομηνία που δημιουργήθηκε. Μετά την ενσωμάτωση των ensemble στο σύστημα, όπου ο αριθμός των αρχείων εξόδου αυξήθηκε, έγινε πολύ δύσκολο να διακρίνεις τα αρχεία που σχετίζονται για κάποιο συγκεκριμένο πείραμα. Με τη νέα κωδικοποίηση των ονομάτων, διατηρούμε μόνο το όνομα χωρίς την ημερομηνία και όλα τα αρχεία που σχετίζονται για κάποιο πείραμα αποθηκεύονται ξεχωριστά σε δικό τους φάκελο.

Για κάθε πείραμα το σύστημα με τη χρήση των *Ensemble* δημιουργεί 7 αρχεία εξόδου . Η ονομασία κάθε αρχείου είναι *X_Output*.txt, όπου X μπαίνει το γράμμα από A μέχρι F για το κάθε δίκτυο του Ensemble (Σχήμα 3.3). Για το τελικό αρχείο που περιέχει το αποτέλεσμα του Ensemble δίνεται το γράμμα Z. Η δομή του αρχείο εξόδου φαίνεται στο Σχήμα 3.3. Επίσης να αναφέρουμε ότι καταργήθηκε η δημιουργία των αρχείων .html γιατί οι πληροφορίες αυτές δημιουργούνται και αποθηκεύονται από αυτά τα αρχεία.

```
1bdoa_77-156 Correctness Percentage:0%
primaryStructure          :EISGHIVRSPMVGTFYRTPSPDAKAFIEVGQKVNVGDTLCIVEAMKMMNQIEADKSGTVKAILVESGQPVEFDEPLVVIE
secondaryStructure        :LLLLEEEELLLLEEEELLLLLLLLLLLLLEELLLLEEEEEELLEEEEELLLLEEEEELLLLLLEELLLLEEEEL
predictedSecondaryStructure:LLLLLEEELLLLLEEEELLLLLLLEEEELLLEELLLEEEEEEHHHHEHELLLLLLLEEEEEELLLLEEELLLEEEEL
1edna_1-21 Correctness Percentage:0%
primaryStructure          :CSCSSLMDKECVYFCHLDIIW
secondaryStructure        :LLLLLLLHHHLLLLLLLLLL
predictedSecondaryStructure:LLLLLLLLHLEEEEEEHHHEL
```

Σχήμα 3.3: *Δομή αρχείο Z_Output.txt. Για κάθε πείραμα δημιουργούνται 7 αρχεία της μορφής αυτής, ένα για κάθε δίκτυο του Ensemble και ένα για το τελικό συνδυασμένο αποτέλεσμα.*

Ακόμη ένα θετικό στοιχείο της κωδικοποίησης αυτής είναι το γεγονός ότι με αυτό τον τρόπο μπορούμε πλέον να τρέχουμε παράλληλα πολλά πειράματα χωρίς να χρειάζεται να περιμένουμε να τελειώσει κάποιο για να ξεκινήσει το επόμενο. Τα αποτελέσματα κάθε πειράματος στην προηγούμενη κωδικοποίηση αποθηκεύονταν στο φάκελο *'DataOut/'* και επομένως κάθε φορά που ξεκινούσαμε ένα νέο πείραμα αντικαθιστούσε τα προηγούμενα αρχεία. Έτσι με την αλλαγή αυτή μπορούμε πλέον να τρέξουμε πολλά πειράματα ταυτόχρονα στην ίδια μηχανή (Server) και στο ίδιο πρόγραμμα, αφού τα αποτελέσματα θα αποθηκεύονται σε διαφορετικούς φακέλους .

Αυτό είναι πολύ σημαντικό γιατί για να αποδείξουμε ότι την ορθότητα των αποτελεσμάτων των πειραμάτων μας πρέπει να τρέξουμε πολλά πειράματα για επαλήθευση. Το σύστημα στη τελική του μορφή χρειάζεται πολύ χρόνο για εκπαιδευτεί, συνήθως 4 με 5 μέρες. Επομένως ήταν αναγκαίο, να μπορούμε να τρέξουμε πολλά πειράματα ταυτόχρονα, για εξοικονόμηση χρόνου.

Άλλη σημαντική προσθήκη στα αρχεία εξόδου ήταν δημιουργία αρχείων που περιέχουν την ακριβή τιμή εξόδου του κάθε δίκτυού σε κάθε θέση της πρωτεΐνης. Στη φάση του *filtering* που θα αναλυθεί στο Κεφάλαιο 4, χρειαζόμαστε την ακριβή αριθμητική τιμή και των τριών εξόδων για κάθε αμινοξύ κάθε πρωτεΐνης που εκπαιδεύτηκε. Στα αρχεία που περιγράφτηκαν προηγουμένως παρατηρούμε ότι στο αρχείο εξόδου αποθηκεύεται μόνο το γράμμα που αντιστοιχεί στο τύπο της δευτεροταγούς δομής και αφού υπολογιστεί η τελική έξοδος του δικτύου. Εμείς θα δημιουργήσουμε πέραν των 7 αρχείων που υπάρχουν, ακόμη 7 στα όποια για κάθε πρωτεΐνη θα προστεθούν 3 επιπλέον γραμμές. Η κάθε γραμμή θα αντιπροσωπεύει την αριθμητική τιμή του νευρώνα εξόδου που αντιστοιχεί σε κάθε κατάσταση της δευτεροταγούς δομής (H, E, L), για κάθε αμινοξύ της πρωτεΐνης (Σχήμα 3.4). Το όνομα κάθε αρχείου θα είναι της μορφής *X_Filter_Output.txt*, όπου X είναι η ίδια κωδικοποίηση με τα προηγούμενα αρχεία. Επομένως, τώρα θα έχουμε αποθηκευμένες όλες τις πληροφορίες που χρειαζόμαστε για την υλοποίηση του *filtering* (Κεφάλαιο 4).

```
1bdoa_77-156 Correctness Percentage:0%
primaryStructure          :EISGHIVRSPMVGTFYRTPSPDAKAFIEVGQKVNVGDTLCIVEAMKMMNQIEADKSGTVKAILVESGQPVEFDEPLVVIE
secondaryStructure        :LLLLEEEELLLLEEEELLLLLLLLLLLLLLEELLLLEEEEEELLEEEEELLLLEEEEEELLLLLLEELLLLEEEEEL
predictedSecondaryStructure:LLLLLEEELLLLLEEEELLLLLLLEEEELLLEELLLEEEEEEHHHHEHELLLLLLLEEEEEEELLLLEEELLLEEEEEL
Output Value of:
H
0.1062,0.1026,0.07538,0.161,0.07325,0.08788,0.02427,0.04913,0.008473,0.01603,0.08948,0.05163,0.02058,0.01851,0.08522,0.07248,0.1067,0.07638,0.07363,0.05144,0.07217,0.1024,0.10
22,0.06741,0.07939,0.1325,0.08003,0.1271,0.09848,0.01459,0.1143,0.06612,0.07491,0.1122,0.02948,0.05511,0.2852,0.1507,0.2428,0.4174,0.3208,0.2842,0.2379,0.3655,0.4615,0.4046,0.
5047,0.2989,0.325,0.1145,0.06788,0.0743,0.01198,0.04906,0.08483,0.07684,0.01972,0.01846,0.1232,0.03408,0.1171,0.1025,0.3978,0.1063,0.1171,0.0397,0.03084,0.08366,0.07138,0.0272
6,0.06201,0.04211,0.1177,0.2765,0.08752,0.1209,0.1252,0.2284,0.1825,0.0021
L
0.7603,0.9001,0.8632,0.6902,0.6898,0.2161,0.2473,0.3211,0.8848,0.8664,0.8655,0.83,0.548,0.2779,0.04248,0.1386,0.2866,0.6559,0.8376,0.9071,0.8649,0.7832,0.7564,0.6648,0.3349,0.
255,0.1543,0.2634,0.7328,0.9187,0.6585,0.2116,0.2342,0.3203,0.8544,0.9317,0.6789,0.2536,0.04195,0.03236,0.009683,0.03453,0.06249,0.1749,0.3134,0.1808,0.2128,0.2987,0.3128,0.28
,0.4484,0.5461,0.7448,0.7721,0.7171,0.8864,0.6204,0.1636,0.0468,0.07252,0.07972,0.03655,0.1254,0.173,0.502,0.8985,0.9545,0.8237,0.4236,0.2583,0.4637,0.7226,0.8954,0.5243,0.285
1,0.06995,0.04385,0.04488,0.2011,0.8562
E
0.04711,0.01882,0.06738,0.1348,0.1942,0.7732,0.8168,0.7003,0.1589,0.1815,0.05139,0.1135,0.4503,0.8167,0.9242,0.8206,0.5925,0.2653,0.0782,0.04336,0.07041,0.1067,0.1476,0.2678,0
.6473,0.5371,0.7733,0.5182,0.1912,0.07659,0.1579,0.7889,0.6107,0.5931,0.1143,0.02508,0.03249,0.6113,0.6813,0.8019,0.8715,0.6511,0.5625,0.36,0.1433,0.3992,0.1656,0.3208,0.2504,
0.5701,0.4253,0.4216,0.1714,0.1899,0.154,0.06656,0.3827,0.836,0.8371,0.9315,0.8607,0.9309,0.4599,0.696,0.2508,0.08539,0.0192,0.06898,0.5319,0.6553,0.5022,0.1917,0.02312,0.1239
,0.6415,0.7251,0.9305,0.8841,0.551,0.07691
```

Σχήμα 3.4: *Δομή αρχείο Z_Filter_Output.txt. Παρατηρούμε ότι για κάθε θέση της πρωτεΐνη έχουμε και την ακριβή αριθμητική τιμή και για τις τρεις πιθανές καταστάσεις της δευτεροταγούς δομής.*

Κεφάλαιο 4

Σχεδιασμός και Υλοποίηση

4.1 Υλοποίηση Παραθύρου Κεντρικού Δικτύου BRNN

4.2 Ενσωμάτωση Ensembles

4.3 Υλοποίηση Post Processing Filtering

4.3.1 Self Organized Map-Kohonen Algorithm

4.3.2 Radial Basis Function

4.4 Ενσωμάτωση Εξελικτικών Στρατηγικών

4.4.1 Εξέλιξη του Βαθμού Συμμετοχής του κάθε BRNN στο Ensemble

4.4.2 Εξέλιξη των τιμών των βαρών του BRNN

4.4.2α Εξέλιξη των τιμών των βαρών χωρίς τη χρήση του Back-Propagation

4.4.2β Εξέλιξη των τιμών των βαρών μετά τη εκπαίδευση του δικτύου με Back-Propagation

4.4.3 Ενσωμάτωση του SOV ως fitness function

4.1 Υλοποίηση Παραθύρου Κεντρικού Δικτύου BRNN

Κατ' αρχάς η πρώτη τροποποίηση που κάναμε στο σύστημα αφόρα το κεντρικό δίκτυο του BRNN. Όπως επεξηγήθηκε στο κεφάλαιο 2 το BRNN αποτελείται από τρία νευρωνικά δίκτυα (Κεφάλαιο 2, Σχήμα 2.1). Το αριστερό νευρωνικό δίκτυο Ft επεξεργάζεται τα αμινοξέα που προηγούνται του αμινοξέως που βρίσκεται στο κέντρο του παραθύρου ενώ το δεξιό νευρωνικό δίκτυο Bt επεξεργάζεται τα αμινοξέα που έπονται του κεντρικού αμινοξέως του κινητού παραθύρου. Επίσης για κάθε αμινοξύ μιας πρωτεΐνης δημιουργείται ένα κινούμενο παράθυρο για να μπορούμε να γνωρίζουμε τι προηγήθηκε και τι έπεται ενός συγκεκριμένου αμινοξύ. Η *doFeedForward()* συνάρτηση είναι υπεύθυνη να μεταφέρει την πληροφορία των καταλοίπων που περιέχονται στο κινούμενο παράθυρο, στο δίκτυο για επεξεργασία (Κεφάλαιο 2, Σχήμα 2.7).

Μετά από μελέτη στο υπάρχων σύστημα παρατηρήσαμε ότι ενώ για το αριστερά και το δεξιά δίκτυο υπάρχει κινητό παράθυρο, για το κεντρικό δίκτυο δεν υπάρχει υλοποιημένο. Δηλαδή όταν δίνεται ένα αμινοξύ για επεξεργασία στο σύστημα το κεντρικό δίκτυο δεν επεξεργάζεται πληροφορία πέραν του ενός αμινξοέως. Αυτό μπορεί να οδηγήσει σε χαμηλά ποσοστά πρόβλεψης γιατί δεν δίνεται στο δίκτυο η απαραίτητη πληροφορία για επεξεργασία. Η αναδίπλωση των πρωτεϊνών επηρεάζεται άμεσα από τη πληροφορία που εμπεριέχεται στα κατάλοιπα της. Το τι έπεται και τι προηγήθηκε παίζουν σημαντικό ρόλο αφού σύμφωνα με το Ceroni (Ceroni et al., 2005) τα δεδομένα έχουν μια χρονική εξάρτηση μεταξύ τους. Επομένως η μη ύπαρξη κινητού παραθύρου στο κεντρικό δίκτυο μπορεί να προκαλέσει λανθασμένα αποτελέσματα.

Η τροποποίηση που κάναμε αφόρα την κλάση *BidirectionalReccurentNetwork.cpp* και συγκεκριμένα τη συνάρτηση *doFeedForward(),* ενώ επίσης έγινε και μια μικρή αλλαγή στη κεντρική κλάση *pssp_ucy.cpp* και στο αρχείο παραμέτρων. Συγκεκριμένα, προστέθηκε αρχικά στο αρχείο parametes.dat ακόμη μια παράμετρος που αφορά το μέγεθος του κεντρικού παραθύρου. Για παράδειγμα αν η τιμή της παραμέτρου window_size είναι 2, αυτό σημαίνει ότι για το κεντρικό δίκτυο θα γνωρίζουμε τα αμινοξέα των 2 προηγούμενων και των 2 επόμενων θέσεων (2+2=4).

Όσον αφορά τη *BidirectionalReccurentNetwork.cpp* στο κομμάτι που αφορά το κεντρικό παράθυρο πλέον δε θα επεξεργαζόμαστε πληροφορία μόνο για ένα αμινοξύ αλλά για περισσότερα. Επομένως στο συνάρτηση doFeedForward(), στο τμήμα που αφορά το κεντρικό δίκτυο, πρέπει να προστεθεί ακόμη ένα βρόχος επανάληψης για να διαβάζονται τα κατάλοιπα αριστερά και δεξιά του κεντρικού αμινοξέος (Σχήμα 4.1).

```
for(int center=sequencet-center_window_size;
      center<=sequencet+center_window_size; center++)

      for(int p=center-halfInputSize;
            p<(sequencet-halfInputSize+inputSize); p++)
                  Εμπρόσθιο πέρασμα αμινοξέως από κετρικό δίκτυο
```

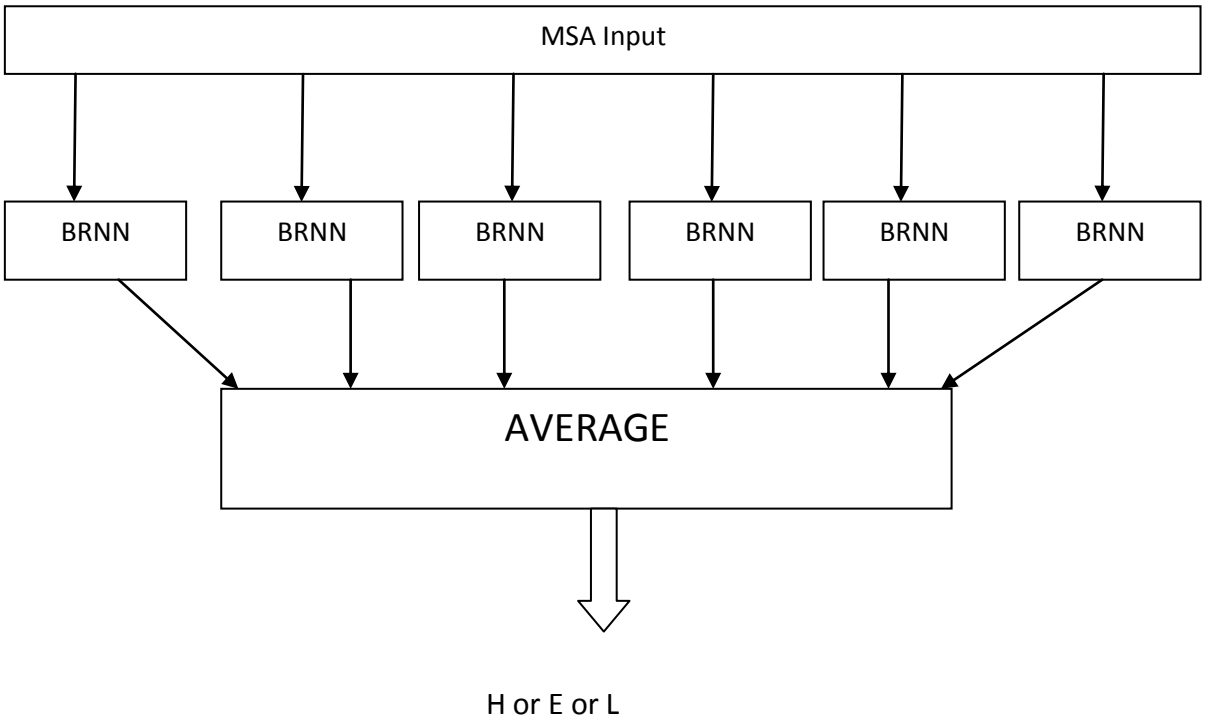
Σχήμα 4.1 : *Η επιπλέον επανάληψη στο κεντρικό δίκτυο στη "BidirectionalReccurentNetwork.cpp" επιτυγχάνεται με το εξωτερικό loop. Από το αμινοξύ που διαβάσαμε θα πάρουμε τη πληροφορία που βρίσκεται "center_window_size" αμινοξέα προηγουμένος του και "center_window_size" αμινοξέα μετά.*

Η εισαγωγή της νέας αυτής παραμέτρου εξυπακούει ότι πρέπει να ξαναγίνουν όλα τα πειράματα για την εύρεση των βέλτιστων τιμών των παραμέτρων (Κεφάλαιο 5). Η πρόβλεψη της δευτεροταγούς δομής οφείλεται σε πρώτο βαθμό στις ποικίλες αλληλεπιδράσεις ανάμεσα στα αμινοξέα. Γι αυτό τον λόγο, πολλές μέθοδοι βασίζονται σε ένα κινητό παράθυρο σταθερού μεγέθους από το οποίο προσπαθούν να προβλέψουν την δευτεροταγή δομή του κεντρικού αμινοξέως, λαμβάνοντας υπόψη τις αλληλεπιδράσεις των αμινοξέων του κινητού παραθύρου που προηγούνται και έπονται του κεντρικού αμινοξέος. Αυτό, δεν είναι αρκετό, γιατί οι αλληλεπιδράσεις ανάμεσα στα αμινοξέα μπορεί να είναι πολλές και ταυτόχρονα μακρινές. Εκτός από αυτό, υπάρχουν και συσχετίσεις ανάμεσα στα προβλεπόμενα στοιχεία της δευτεροταγούς δομής των αμινοξέων αυτών. Επομένως ένα κινητό παράθυρο σταθερού μεγέθους είναι δύσκολο να προσδιορίσει επακριβώς τις μακρινές αλληλεπιδράσεις των αμινοξέων και την δευτεροταγή τους συσχέτιση. Όμως, επιτρέποντας μεγαλύτερο μέγεθος παραθύρου και πάλι δεν αυξάνουμε το ποσοστό πρόβλεψης λόγω υπερεκπαίδευσης. Άρα, η εφαρμογή κινητού παραθύρου στην πρόβλεψη δευτεροταγούς δομής έχει σοβαρούς περιορισμούς για το πρόβλημα (Crooks and Brenner, 2004), (Chen and Chaudhari, 2006, 2007).

4.2 Ενσωμάτωση Ensembles

Η μεθοδολογία των Ensembles χρησιμοποιήθηκε όπως προαναφέρθηκε με την υπόθεση ότι πολλά δίκτυα μαζί μπορούν να εξάγουν καλύτερα αποτελέσματα από ένα. Για την υλοποίηση των ensembles χρησιμοποιήσαμε την ίδια τεχνική που ακολούθησε και ο Baldi (Baldi et al., 1999,Baldi et al., 2000).

Το ensemble που δημιουργήσαμε αποτελείται από έξι νευρωνικά δίκτυα BRNN και ακολουθούν όλα την ίδια αρχιτεκτονική (Σχήμα 4.2). Στην ουσία αντί να έχουμε μόνο ένα αντικείμενο της κλάσης *BidirectionalReccurentNetwork.cpp*, πλέον έχουμε 6 στιγμιότυπα της κλάσης αυτής. Από το σχήμα 4.3 βλέπουμε ότι και τα 6 BRNN δημιουργούνται και αρχικοποιούνται με το ίδιο τρόπο.



Σχήμα 4.2*: Αρχιτεκτονική του Ensemble που δημιουργήθηκε*

```
BidirectionalRecurrentNeuralNetwork BRNN2 =
      BidirectionalRecurrentNeuralNetwork('B',hLayerOneSize,
      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,
      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,
      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,
      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

BidirectionalRecurrentNeuralNetwork BRNN3 =
      BidirectionalRecurrentNeuralNetwork('C',hLayerOneSize,
      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,
      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,
      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,
      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);
```

Σχήμα 4.3: *Παράδειγμα αρχικοποιήσης 2 από τα 6 BRNN.*

Όλες οι λειτουργίες του Ensemble γίνονται στη κύρια κλάση του συστήματος, *pssp_ucy.cpp*. Από το σχήμα 4.4 παρατηρούμε ότι όταν διαβάζεται ένα αμινοξύ, αυτό εκπαιδεύεται και επεξεργάζεται και από τα 6 BRNN. Το τελικό αποτέλεσμα θα αποτελείται από το συνδυασμό των αποτελεσμάτων και των 6 δικτύων. Για αυτό το λόγο ορίσαμε ένα πίνακα για κάθε BRNN στο οποίο θα αποθηκεύεται το αποτέλεσμα του συγκεκριμένου BRNN, ενώ επίσης θα χρειαστούμε και ένα πίνακα στον οποίο θα αποθηκεύονται ο συνδυασμός τους και επομένως το τελικό αποτέλεσμα.

```
while(υπάρχουν πρωτεινες στο training dataSet)
{
      Διάβασε το MSA της επόμενης x πρωτεινης για το BRNN1
      Διάβασε το MSA της επόμενης x πρωτεινης για το BRNN2
      Διάβασε το MSA της επόμενης x πρωτεινης για το BRNN3
      Διάβασε το MSA της επόμενης x πρωτεινης για το BRNN4
      Διάβασε το MSA της επόμενης x πρωτεινης για το BRNN5
      Διάβασε το MSA της επόμενης x πρωτεινης για το BRNN6

      Αρχικοποίηση και πρετοιμασία BRNN1
      Αρχικοποίηση και πρετοιμασία BRNN2
      Αρχικοποίηση και πρετοιμασία BRNN3
      Αρχικοποίηση και πρετοιμασία BRNN4
      Αρχικοποίηση και πρετοιμασία BRNN5
      Αρχικοποίηση και πρετοιμασία BRNN6


      Για κάθε αμινοξύ t της πρωτεινης x
      {
       BRNN1.doFeedForward(t)
       BRNN2.doFeedForward(t)
       BRNN3.doFeedForward(t)
       BRNN4.doFeedForward(t)
       BRNN5.doFeedForward(t)
       BRNN6.doFeedForward(t)

       BRNN1.doBackpropagation(t)
       BRNN2.doBackpropagation(t)
       BRNN3.doBackpropagation(t)
       BRNN4.doBackpropagation(t)
       BRNN5.doBackpropagation(t)
       BRNN6.doBackpropagation(t)
      }

      Αποθήκευσε το SequenceTrainingError για το BRNN1
      Αποθήκευσε το SequenceTrainingError για το BRNN2
      Αποθήκευσε το SequenceTrainingError για το BRNN3
      Αποθήκευσε το SequenceTrainingError για το BRNN4
      Αποθήκευσε το SequenceTrainingError για το BRNN5
      Αποθήκευσε το SequenceTrainingError για το BRNN6

      Αν βρίσκεσαι στη τελευτάια εποχή{
      Τύπωσε το OutputSequence(t) για το BRNN1
      Τύπωσε το OutputSequence(t) για το BRNN2
      Τύπωσε το OutputSequence(t) για το BRNN3
      Τύπωσε το OutputSequence(t) για το BRNN4
      Τύπωσε το OutputSequence(t) για το BRNN5
      Τύπωσε το OutputSequence(t) για το BRNN6
      }

}
```

Σχήμα 4.4: *Οι λειτουργίες του BRNN εκτελούνται και για τα 6 δίκτυα*

Όσον αφορά τον τρόπο που θα συνδυαστούν, δοκιμάσαμε 4 διαφορετικές τεχνικές προσέγγισης για να ανακαλύψουμε ποια έχει τα καλύτερα αποτελέσματα. Πριν αναλύσουμε τις τεχνικές είναι κάλο να υπενθυμίσουμε πως κωδικοποιείται η έξοδος του δικτύου. Το κάθε BRNN για κάθε αμινοξύ που εκπαιδεύεται, επιστρέφει τρεις αριθμητικές τιμές, μια για κάθε κατάσταση που υπάρχει στη δευτεροταγής δομής. Η πιο μεγάλη τιμή αντιπροσωπεύει και τη κατάσταση που προέβλεψε το σύστημα. Επομένως τώρα που εκπαιδεύονται 6 BRNN, η τελική απόφαση θα πρέπει να υπολογίζεται λαμβάνοντας υπόψη και τις 3 εξόδους και των 6 δικτύων.

Οι 4 μέθοδοι που χρησιμοποιήσαμε ήταν το *Average, Weighted Average, Voting* και *Borda Function.*

Σύμφωνα με το *Voting* το αποτέλεσμα που υποστηρίζεται από τη πλειοψηφία των νευρωνικών δικτύων είναι και το αποτέλεσμα του Ensemble (Σχήμα 4.2).

Σύμφωνα με το *Bora Function* δίνουμε κάποιο σκορ (Borda Score) σε κάθε μέλος του Ensemble. Έστω ότι το r^i_k είναι το ranking της $i_{οστή}$ κλάσης του $κ_{στού}$ ταξινομητή, τότε το Borda Score υπολογίζεται αθροίζοντας το M- r^i_k. Επομένως το τελικό αποτέλεσμα υπολογίζεται σύμφωνα με τη εξίσωση 4.1

$$F(e(x)) = \max_{i \in A} \sum_{k=1}^{K} (M - r_k^i(x))$$

Εξίσωση 4.1: *Εξίσωση υπολογισμού της εξόδου του Ensemble δικτύου με τη χρήση του Borda Score.*

Στην μέθοδο το *Average* καθεμία από τις τρεις καταστάσεις εξόδου κάθε BRNN αθροίζεται με τις υπόλοιπες των άλλων BRNN στη συνέχεια βρίσκεται ο μέσος όρος για κάθε κατάσταση. Τέλος επιλέγουμε μια από τις τρεις καταστάσεις (H,E ή L) που έχει τη ψηλότερη τιμή.

Τέλος η μέθοδος *Weighted Average* είναι παρόμοια με την *Average* αλλά η διαφορά της είναι ότι στη έξοδο κάθε BRNN δίνεται μία τιμή βάρους η οποία δείχνει το βαθμό επιρροής του κάθε δικτύου και η οποία υπολογίζεται σύμφωνα με το ποσοστό σφάλματος του κάθε BRNN (Εξίσωση 4.2).

$$w_i = \frac{1 - E_i}{\sum_k (1 - E_k)}$$

Εξίσωση 4.2: *Το w_i ορίζεται ως το βαρός που αποκτά η έξοδος του i BRNN δικτύου και υπολογίζεται σύμφωνα με το ποσοστό σφάλματος Ei.*

Μετά από δοκιμή πολλών πειραμάτων, τα οποία παρουσιάζονται στο Κεφάλαιο 5, παρατηρήσαμε ότι τα καλύτερα αποτελέσματα βελτίωσης του συστήματος ήταν με τη μέθοδο το Average με μικρή διαφορά από τα αποτελέσματα του Weighted Average. Για λόγο αυτό θα κρατήσουμε αυτή τη τακτική και σε όλες τις μεθόδους βελτίωσης και πειραμάτων που θα αναπτυχθούν στη συνέχεια. Ο τρόπος που υπολογίζεται το Average στο πρόγραμμα μας φαίνεται στο πιο κάτω κομμάτι κώδικα (Σχήμα 4.5)

```
Για κάθε κατάσταση K της δευτεροταγούς δομης της πρωτείνης x
{
 ΑποτέλεσμαEnsemble[K]=(ΑποτέλεσμαBRNN1[K]  +
      ΑποτέλεσμαBRNN2[K]  + ΑποτέλεσμαBRNN3[K]  +
      ΑποτέλεσμαBRNN4[K]  + ΑποτέλεσμαBRNN5[K]  +
      ΑποτέλεσμαBRNN6[K]  )/6.0
}
```

Σχήμα 4.5: *Κομμάτι κώδικα από τη κλαση "pssp_ucy.cpp" που υπολογίζει το τελικο αποτέλεσμα του Ensembe με τη χρήση του Average*

4.3 Υλοποίηση Post Processing Filtering

Πολλοί, έχουν προτείνει για το πρόβλημα αυτό ένα δεύτερο δίκτυο το οποίο να *διορθώνει* τα αποτελέσματα του πρώτου (Qian and Sejnowski, 1988), (Rost and Sander, 1993). Παρόμοια, οι Chen and Chaudhari (Chen and Chaudhari, 2007) έχουν προτείνει ένα δεύτερο δίκτυο, επιπρόσθετα στο δίκτυο του Baldi (Baldi et al., 1999), το οποίο θα λαμβάνει υπόψη τις συσχετίσεις μεταξύ των στοιχείων της δευτεροταγούς δομής (structure to structure) που έχει προβλεφτεί για κάθε αμινοξύ από το πρώτο δίκτυο (sequence to structure), αλλά και τις μακρινές αλληλεπιδράσεις των αμινοξέων. Ουσιαστικά, το τι κάνει το δεύτερο δίκτυο είναι να *διορθώνει* τα αποτελέσματα του πρώτου δικτύου, βάση των συσχετίσεων της δευτεροταγούς δομής των αμινοξέων. Το δίκτυο αυτό δίνει ένα ποσοστό επιτυχίας με βάση την Q3 παράμετρο 74.38%, και για την SOV παράμετρο 66.05% (Chen and Chaudhari, 2007).

Μια άλλη τεχνική *φιλτραρίσματος* προβλεπόμενων πρωτεϊνών, θα ακολουθηθεί σε αυτή την τρέχουσα διπλωματική εργασία. Λαμβάνοντας υπόψη τις θεωρίες της Επιβλεπόμενης και μη Επιβλεπόμενης Μάθησης θα χρησιμοποιήσουμε για το *«φιλτράρισμα»* των δεδομένων εξόδου του υφιστάμενου δικτύου πρώτα ένα χάρτη αυτοοργάνωσης (SOM) και στη συνέχεια ένα νευρωνικό δίκτυο τύπου RBF. Το *φιλτράρισμα* αυτό λειτουργεί με τον ίδιο τρόπο όπως και το δεύτερο διορθωτικό δίκτυο των Chen and Chaudhari (Chen and Chaudhari, 2007), δηλαδή, *διορθώνει* την προβλεπόμενη ακολουθία που δίνει το τρέχον σύστημα στο τέλος κάθε εκτέλεσης του, λαμβάνοντας υπόψη τις συσχετίσεις μεταξύ της προβλεπόμενης δευτεροταγούς δομής κάθε αμινοξέως της πρωτεΐνης.

Όπως προαναφέραμε οι έξοδοι του δικτύου είναι τρεις μια για κάθε κατάσταση δευτεροταγής δομής. Επομένως στόχος του *φιλτραρίσματός* είναι να πάρει αυτές τις τιμές και να προσπαθήσει να τις διορθώσει ούτως ώστε να μπορέσει να αυξήσει το ποσοστό σωστής πρόβλεψης. Για να γίνει σωστό φιλτράρισμα των δεδομένων σύμφωνα με το RBF πρέπει πρώτο να γίνει σωστή οργάνωση των δεδομένων μας. Για αυτό πρώτα θα ασχοληθούμε με τη δημιουργία του χάρτη αυτοοργάνωσης.

4.3.1 Self Organized Map-Kohonen Algorithm

Το SOM είναι είδος μη επιβλεπόμενης μάθησης. Στόχος του είναι μέσα από την εκπαίδευση του να οργανώσει τα δεδομένα σε κατηγορίες ανάλογα με τα κοινά τους χαρακτηριστικά. Στη συγκεκριμένη περίπτωση θέλουμε να οργανώσει τα δεδομένα εξόδου σε *περιοχές-γειτονιές*, σύμφωνα με τη δευτεροταγή δομή των πρωτεϊνών, για να μπορέσουμε στη συνέχεια να χρησιμοποιήσουμε κάποιες περιοχές του χάρτη ως κέντρα στο RBF μας.

Για την υλοποίηση του SOM θα χρησιμοποιήσουμε ως δεδομένα εισόδου το *Z_Filter_Output.txt* το οποίο, όπως αναφέραμε στο κεφάλαιο 3, έχει τροποποιηθεί ούτως ώστε για κάθε αμινοξύ της κάθε πρωτεΐνης που περιλαμβάνει, έχουμε και τη δεκαδική τιμή της κάθε εξόδου (H, E ή L)της δευτεροταγής δομής (Υποκεφάλαιο 3.2, Σχήμα 3.4).

Το SOM διαβάζει τα δεδομένα εισόδου με τη βοήθεια των κλάσεων “pssp.cpp,” “pattern.cpp” και ”parameters.cpp”. Το δίκτυο αποτελείται από νευρώνες της κλάσης “neuron.cpp” η οποία αποθηκεύει ένα πίνακα με όλα τα βάρη του χάρτη και ένα χαρακτήρα που αντιπροσωπεύει την τιμή εξόδου. Οι τρεις λειτουργίες του SOM, Ανταγωνισμός, Συνεργασία, Ανταμοιβή γίνονται κατά κύριο λόγο στις κλάσεις “neutralnet.cpp” και “neutralnet.h”.

Στην αρχή κάθε θέση-νευρώνας του χάρτη αρχικοποιείται με μια τυχαία τιμή. Έπειτα κάθε πρωτεΐνη του διαβάζεται και αποθηκεύεται σε ένα πίνακα. Για την εκπαίδευση του χάρτη χρησιμοποιήσαμε όπως και στο BRNN σύστημα ένα κινητό παράθυρο το οποία μας βοηθά να αποθηκεύουμε περισσότερη πληροφορία για μια συγκεκριμένη θέση της πρωτεΐνης. Στη συνέχεια υλοποιείται η εκπαίδευση του δικτύου. Δηλαδή για ένα συγκεκριμένο αμινοξύ με βάση τα βάρη που το καθορίζουν βρίσκουμε το νευρώνα του χάρτη που απέχει τη πιο μικρή απόσταση. Υπολογίζουμε στη ουσία την ευκλείδεια απόσταση της εισόδου με κάθε νευρώνα του χάρτη (Σχήμα 4.6).

```
findWinningNeuron(vector<double> inputs)
{
      Αρχηκοποιήσε το νικητή με το νευρώνα [0][0] του χάρτη
      Για Κάθε νευρώνα του Χάρτη [i]x[j]
      {
            Υπολόγισε την απόσταση του από την έισοδο
            distance = myNeurons[i][j].calculateDistance(inputs);

            if (distance < smallestDistance)
            {
                  Αποθήκευσε τη απόσταση και τις Συντεταγμένεςς του
            Νικητή
            }
      }

      Επέστρεψε το Νικητή
}

calculateDistance(vector<double> inputs)
{
      distance = 0.0;

      Για κάθε βάρος του νευρώνα
            distance += (inputs[i] - w[i]) * (inputs[i] - w[i]);

      return sqrt(distance);
}
```

Σχήμα 4.6: *Ψευδοκώδικας που υπολογίζει το νικητή-νευρώνα βάση της ευκλείδειας απόστασης της εισόδου από κάθε νευρώνα του χάρτη.*

Ο νικητής νευρώνας είναι και αυτός που θα επηρεαστεί. Σε κάθε νευρώνα αντιστοιχεί μια γειτονιά. Δηλαδή όταν θα ανανεώνουμε τα βάρη ενός νευρώνας, επιπρόσθετα θα ανανεώνουμε και τα βάρη των νευρώνων που τον περιτριγυρίζουν. Η ακτίνα της γειτονίας του υπολογίζεται με τη βοήθεια της εξίσωσής 4.3. Από την εξίσωση παρατηρούμε ότι η ακτίνα καθώς περνά ο χρόνος μικραίνει γιατί θέλουμε σε κάθε επανάληψη να μειώνεται η γειτονία που επηρεάζεται για να μην έχουμε συνεχής μετακινήσεις τμημάτων του χάρτη. Στα πρώτα στάδια αναδίπλωσης του χάρτη θέλουμε οι τροποποιήσεις να είναι μεγάλες γιατί τα βάρη έχουν αρχικοποιηθεί τυχαία. Όμως επειδή ο χάρτης συνεχώς εκπαιδεύεται αυτό σημαίνει ότι δεν χρειάζεται να τροποποιούμε μεγάλα τμήματα σε κάθε επανάληψη.

$$\text{ακτίνα} = \text{Μέγεθος Χάρτη} / 2;$$

$$T = \text{Μέγιστος Αριθμός Εποχών} / \log_e(\text{ακτίνα})$$

$$\text{Ακτίνα επιρροής γειτονίας} = \text{ακτίνα} * e^{(-\text{αριθμός εποχής}/T)}$$

Εξίσωση 4.3: *Η εξίσωση που μας δίνει τη γειτονία που θα τροποποιούμε σε κάθε επανάληψη. Η μεταβλητή radius αντιπροσωπεύει την ακτίνα του χάρτη και η T είναι η χρονική σταθερά, ενώ k ορίζεται ως ο αριθμός της επανάληψης. Όπως βλέπουμε από την εξίσωση σε κάθε επανάληψη η ακτίνα που επηρεάζεται μικραίνει για να μην επηρεάζει το κομμάτι του χάρτη που έχει εκπαιδευτεί.*

Η εξίσωση 4.4 Μας δείχνει ότι ο ρυθμός εκμάθησης μειώνεται εκθετικά όσο περνούν οι επαναλήψεις για να μην γίνονται τεράστιες αλλαγές στις τιμές των βαρών. Επίσης σημαντικό ρόλο παίζει και η παράμετρος *influence* (Εξίσωση 4.5). Σύμφωνα με τη παράμετρο αυτή, εξυπακούεται ότι όσο ποιο μακριά βρισκόμαστε από το νευρώνα-νικητή τόσο πιο μικρές θα πρέπει να είναι οι αλλαγές στα βάρη των συγκεκριμένο θέσεων. Αυτό είναι λογικό γιατί οι μακρινοί γείτονες του νευρώνα-νικητή δεν πρέπει να επηρεάζονται το ίδιο με τους πιο κοντινούς.

$$\text{learningRate} = \text{startLearningRate} * e^{(-k/\text{Maxterations})}$$

Εξίσωση 4.4: *Εξίσωσή που μας δίνει το ρυθμό εκμάθησης σε κάθε επανάληψη*

$$\text{DistToNodeSq} = (\text{winningNeuron}_X - X_{pos})^2 + (\text{winningNeuron}_Y - Y_{pos})^2$$
$$\text{WidthSq} = \text{neighbourhoodRadius}^2$$

$$\text{influence} = e^{(-(\text{DistToNodeSq}) / (2*\text{WidthSq}))}$$

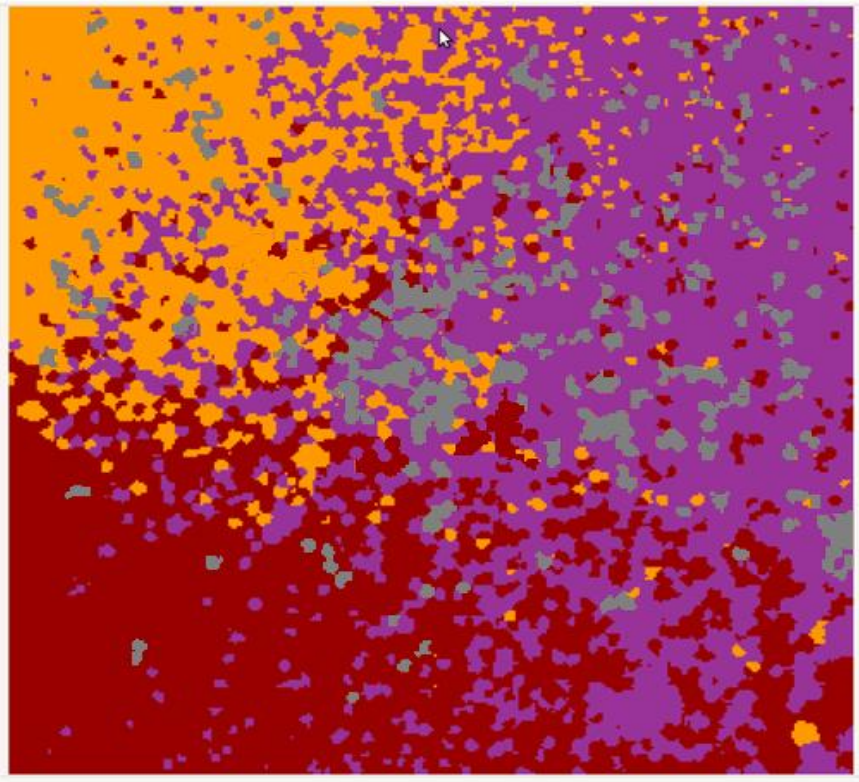
Εξίσωση 4.5:*Εξίσωση που μας δίνει το επιρροή που θα έχει ο νευρώνας νικητής στην αλλαγή των βαρών ενός νευρώνα της γειτονιάς του.*

Όσον αφορά την διόρθωση των βαρών για μείωση του λάθους αυτό επιτυγχάνεται σύμφωνα με τη τμήμα κώδικα που φαίνεται στο σχήμα 4.7.

```
Για κάθε βάρος i που χαρακτηρίζει το νευρώνα w
{
      w[i] += learningRate * influence * (είσοδος[i] - w[i])
}
```

Σχήμα 4.7: *Κώδικας ο οποίος χρησιμοποιείται για τη αλλαγή των βαρών του νευρώνων του SOM.*

Όταν τελειώσει η φάση της εκπαίδευσης, περνάμε στη φάση της δοκιμής όπου δίνεται ένα άγνωστο σύνολο δεδομένων για χαρτογραφηθεί στο SOM. Ένα παράδειγμα του τελικό χάρτη φαίνεται στο σχήμα 4.8.



Σχήμα 4.8: *Εικονική αναπαράσταση της οργάνωσης των δεδομένων σε ομάδες. Ο χάρτης είναι μεγέθους 275*275 και εκπαιδεύτηκε για το σύνολο DataSet1 του CB513. Η αρχική ακτίνα είναι 150 και το μέγεθος του κινητού παραθύρτυ 5(2+2+1). Το μωβ αντιπροσωπεύει τη κωδικοποίηση E , το κόκκινο το L και το κίτρινο το H. Το γκρίζο χρώμα αντιπροσωπεύει τις θέσεις του χάρτη ο οποίες δεν χρησιμοποιήθηκα ποτέ κατά τη διάρκεια εκπαίδευσης του χάρτη και επομένως δεν αντιπροσωπεύουν κάποια συγκεκριμένη κατάσταση (H,E,L).*

Στο Σχήμα 4.8 φαίνεται καθαρά ο τρόπος που έγινε κατηγοριοποίηση. Συγκεκριμένα το κίτρινο χρώμα αντιπροσωπεύει την έξοδο L(Loop), το κόκκινο χρώμα την έξοδο H (Helix) και το μωβ χρώμα την έξοδο E (Extended). Το γκρίζο χρώμα αντιπροσωπεύει τις θέσεις του χάρτη ο οποίες δεν χρησιμοποιήθηκα ποτέ κατά τη διάρκεια εκπαίδευσης του χάρτη και επομένως δεν αντιπροσωπεύουν κάποια συγκεκριμένη κατάσταση (H,E,L). Επίσης από την εικόνα είναι φανερό ότι για τα Helix που αποτελούν το μεγάλο τμήμα στις πρωτεΐνες, παρατηρούμε ότι έχει γίνει καλύτερη ομαδοποίηση. Δεν έχουμε πολλές μικρές ομάδες στο χάρτη σε αντίθεση με τα Extended που είναι πιο δύσκολα να τα ομαδοποιήσει, λόγω της χαμηλής εμφάνισης τους στο σύνολο των πρωτεϊνών.

Αυτό που είναι σημαντικό να αποθηκεύσουμε για να χρησιμοποιήσουμε στο RBF είναι οι τιμές των βαρών του χάρτη όπως έχουν εκπαιδευτεί. Στο πρόγραμμα στη τελευταία επανάληψη, δημιουργεί το αρχείο weights.dat του οποίου το περιεχόμενο φαίνεται στο σχήμα 4.9.

```
Neuron No: 0 , 0
    0.00722648
    0.0100125
    0.996151

Neuron No: 0 , 1
    0.0088144
    0.0120228
    0.994109

Neuron No: 0 , 2
    0.0144101
    0.0190648
    0.989178

Neuron No: 0 , 3
    0.019415
    0.0188003
    0.98765
```

Σχήμα 4.9: *Μορφή του αρχείο weights.dat που δημιουργείται από το SOM και περιέχει μέσα τις τιμές των βαρών των νευρώνων του χάρτη, όπως αυτός οργανώθηκέ*

4.3.2 Radial Basis Function

Στη επόμενη φάση του φιλτραρίσματος δημιουργήσαμε ένα δίκτυο Radial Basis Function το οποίο όπως αναφέραμε αρχικοποιήσαμε σε σχέση με το αποτέλεσμα του SOM. Το RBF που επιλέχθηκε χρησιμοποιεί ένα κρυφό επίπεδο όπως επίσης υλοποιείται και εδώ το κινητό παράθυρο για το ίδιο λόγο. Ακόμη μια σημαντική παράμετρος είναι το γεγονός ότι επιλέξαμε RBF με μεταβλητή θέση κέντρων και τυπική απόκλιση. Λόγω της φύσης του προβλήματος επιλέξαμε τα μεταβλητά κέντρα γιατί δεν έχουμε να κάνουμε με πρόβλημα παρεμβολής αλλά πρόβλημα προσέγγισης. Ο αλγόριθμός που χρησιμοποιήθηκε φαίνεται στο σχήμα 4.10

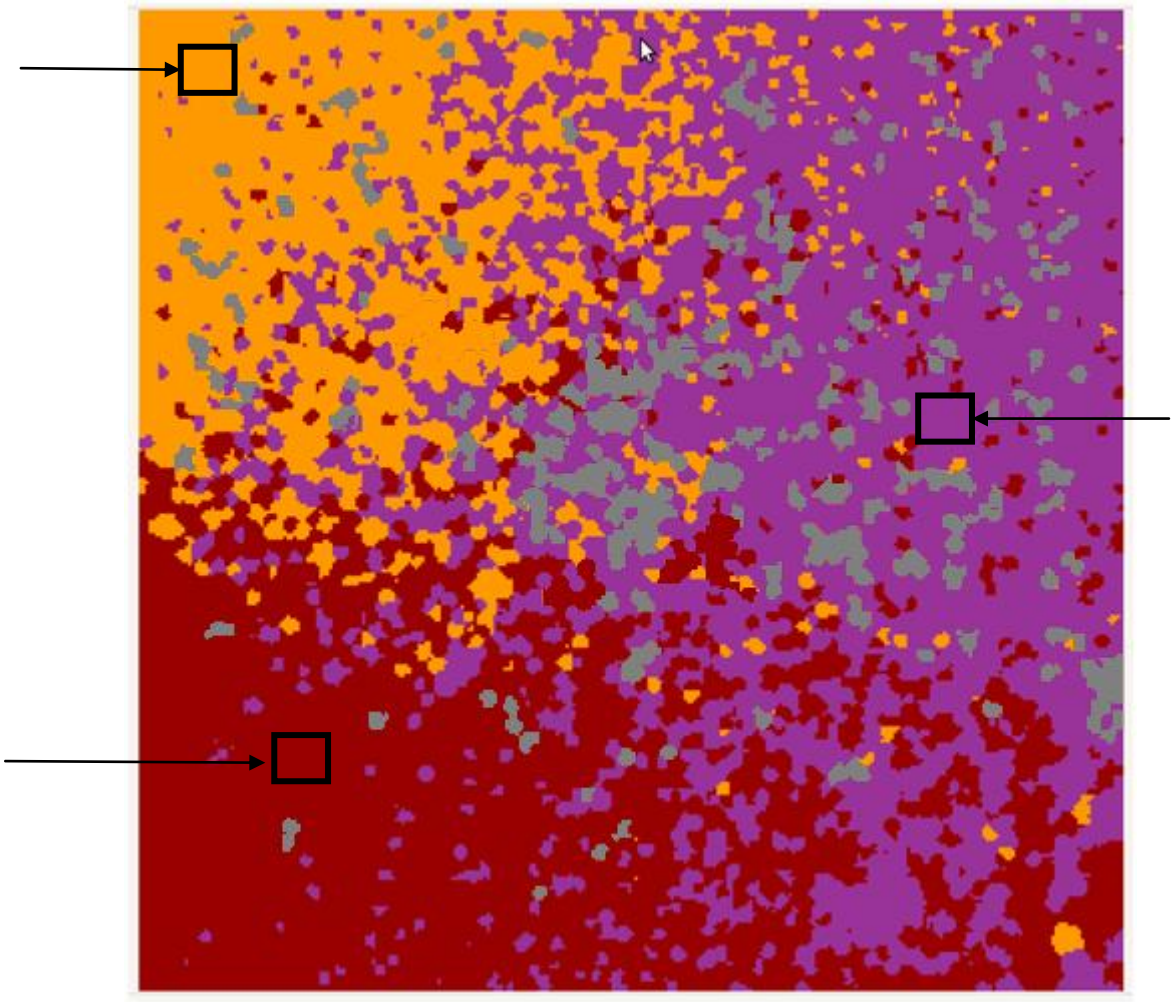
Σχετικές Κωδικοποιήσεις

Βάρη/Συντελεστές	**w**	Κέντρο	**R**
Επιθυμητή Έξοδος	**t**	Iterator κέντρου	**h**
Πραγματική Έξοδος	**y**	Αριθμός Κέντρων	**m**
Πρότυπο Εισόδου	**x**	Iterator Εξόδου	**o**
Input Dimension Iterator	**i**	Αριθμός Κέντρων	**k**
Input Dimension	**d**	Τυπική Απόκλιση	**σ**
Αριθμός Εισόδων	**n**	Learning Rate	$\mathbf{n_R}$, $\mathbf{n_\sigma}$, $\mathbf{n_w}$
Iterator Πρότυπου Εισόδου	**p**	Bias unit	**b**

1. Επέλεξε τον αριθμό (m) και αρχικοποίησε τις συντεταγμένες των κέντρων (R) των RBF συναρτήσεων.
2. Επέλεξε την αρχική τιμή της παραμέτρου (σ) για κάθε κέντρο R.
3. Αρχικοποίησε τις τιμές των βαρών w με μικρές τυχαίες τιμές [-1,1].
4. Για κάθε εποχή
5. Για κάθε πρότυπο εισόδου $x^{(p)}$
6. Υπολόγισε την έξοδο y για κάθε κόμβο ο χρησιμοποιώντας την εξίσωση 1.10.
7. Μετάβαλλε τις παραμέτρους (w,R,σ) χρησιμοποιώντας τις εξισώσεις 1.15,1.16,1.17,1.18
8. Τέλος για p=n
9. Τέλος όταν e=τελευταία εποχή

Σχήμα 4.10: *Αλγόριθμος εκπαίδευσης RBF.*

Το πρώτο βήμα του αλγόριθμου σχετίζεται με το δίκτυο της Μη-επιβλεπόμενης μάθησης που αναπτύξαμε προηγουμένως. Σε ένα RBF δίκτυο είναι πολύ σημαντική η σωστή επιλογή αριθμού αλλά και των θέσεων των κέντρων που θα χρησιμοποιήσουμε. Σε περίπτωση που δεν γίνει σωστή επιλογή τότε είναι πολύ πιθανόν να μην καταφέρουμε να εκπαιδεύσουμε το δίκτυο και να πάρουμε χειρότερα αποτελέσματα από προηγουμένως. Για αυτό άλλωστε χρησιμοποιήσαμε το SOM(Υποκεφάλαιο 4.3.1). Το SOM μας έχει ομαδοποιήσει τις τρεις πιθανές εξόδου της δευτεροταγής δομής με βάση τα κοινά τους χαρακτηριστικά. Έτσι βλέποντας το χάρτη που δημιουργήθηκε μπορούμε να δούμε ποιες περιοχές του χάρτη θα ήταν καλύτερο να χρησιμοποιήσουμε. Επιλέξαμε στη ουσία περιοχές που είναι “καθαρές”, δηλαδή δεν υπάρχουν μέσα ομάδες των άλλων καταστάσεων και εκτείνονται σε σχετικά μεγάλη απόσταση και άρα καλύπτουν μεγάλο αριθμό συνδυασμών που εμπερικλείονται στη συγκεκριμένη κατάσταση. Επομένως θα περιέχουν και καλύτερη πληροφορία για την συγκεκριμένη κατάσταση, κάτι που είναι απαραίτητο να υπάρχει στα κέντρα. Όσον αφόρα το αριθμό των κέντρων, αυτός επιλέχτηκε εμπειρικά και μετά από πολλές δοκιμές σε διάφορα πειράματα (Κεφάλαιο 5). Στο Σχήμα χ.χ φαίνεται η επιλογή των κεντρών που έδωσε τα καλυτερα αποτελέσματα. Όπως μπορούμε να δούμε επιλέξαμε 3 περιοχές που δεν περιέχουν μέσα άλλες καταστάσεις. Κάθε περιοχή είναι περιπου 100 νευρώνες. Επομένως σύνολο χρησιμοποιήσαμε 300 κέντρα. Όπως θα φαίνεται και στα πειράματα του Κεφαλαίο 5, η πιο δύσκολη επιλογή αφορούσε τη κατάσταση Ε, που δεν έγινε τόσο καλή ομαδοποίηση και έπρεπε να δοικμαστούν διάφορες περιοχές μέχρι να βρεθεί η καλύτερη.



Σχήμα 4.11: *Τα τετράγωνα υποδεικνύουν τις περιοχές που επιλέχτηκαν ως κέντρα από το SOM. Κάθε τετράγωνο είναι μεγέθους 100 νευρώνων και αντικατοπτρίζει τις 3 διαφορετικές καταστάσεις της δευτεροταγούς δομής πρωτεϊνών.Ο χάρτης είναι μεγέθους 275*275 και εκπαιδεύτηκε για το σύνολο DataSet1 του CB513. Το μωβ αντιπροσωπεύει τη κωδικοποίηση Ε , το κόκκινο το L και το κίτρινο το Η. Το γκρίζο χρώμα αντιπροσωπεύει τις θέσεις του χάρτη ο οποίες δεν χρησιμοποιήθηκα ποτέ κατά τη διάρκεια εκπαίδευσης του χάρτη και επομένως δεν αντιπροσωπεύουν κάποια συγκεκριμένη κατάσταση (H,E,L).*

Στη συνέχεια όπως βλέπουμε από τον αλγόριθμο (Σχήμα 4.10) το κάθε αμινοξυ μαζί με το παράθυρο που ορίστηκε περνά από το επίπεδο εισόδου στο κρυφό επίπεδο όπου εκεί υπολογίζεται πρώτα η ευκλείδεια απόσταση της εισόδου από κάθε κέντρο (Σχήμα 4.12) και στη συνέχεια με τη βοήθεια της Gaussian Συνάρτηση ανάγουμε το πρόβλημα σε ψηλότερη διάσταση.

```
Επανέλαβε για ένα μέγιστο αριθμό Επαναλήψεων
 {
      Για κάθε πρότυπο εισόδου i που βρίσκεται στο Training Set
      {
       Για κάθε νευρώνα j του Κρυφού Επιπέδου H
       {
          Υπολόγισε Απόσταση H[j] από είσοδο (i) χρησιμοποιώντας την
          Σχήμα 4.13

          Χρησιμοποίησε τη Gaussian Συνάρτησης (Σχημα 4.14) για να
          υπολογίσεις την  έξοδου την Έξοδο κάθε Κρυφού Νευρώνα
       }
      }

      Για κάθε νευρώνα j του Επιπέδου Εξόδου O
       {
          Υπολόγισε το άθροισμα του γινομένου κάθε νευρώνα με το
          βάρος του

          Υπολόγισε το error μεταξύ επιθυμητής και πραγματικής τιμής
       }
}
```

Σχήμα 4.12: *Κομμάτι κώδικα που υπολογίζει πρώτα τη τιμή εξόδου κάθε νευρώνα στο κρυφό επίπεδο βρίσκοντας την απόσταση μιας εισόδου από κάθε κέντρο και στη συνέχεια υπολογίζει στο επίπεδο εξόδου το τελικό αποτέλεσμα του δικτύου.*

```
Για κάθε κεντρο h του Κρυφού Επίπεδου
{
      απόσταση =(Εισοδος[i]-h[i])^2
}
```

Σχήμα 4.13: *Ψευδωκώδικας που υπολογίζει την ευκλείδεια απόσταση μεταξύ της εισόδου i και κάθε κέντρου h του κρυφού επιπέδου.*

```
Για κάθε κεντρο h του Κρυφού Επίπεδου
{
      e^( -Ευκλειδεια Απόσταση(Είσοσδος (i)-Κέντρο Κρυφού Επιπέδου / σ2)
}
```

Σχήμα 4.14: *Ψευδωκώδικας που δείχνει πώς χρησιμοποιείται η Gaussian για να υπολογίσουμε το αποτέλεσμα εξόδου του κρυφού Επιπέδου.*

Στο τελευταίο στάδιο αφού υπολογίσουμε τη έξοδο του δικτύου αλλάζουμε τα βάρη(Εξίσωση 1.15) , τη θέση των κέντρων(Εξίσωση 1.17) και τη τιμή της τυπικής διασποράς(Εξίσωση 1.18) τη Gaussians Συνάρτησης.

4.4 Ενσωμάτωση εξελικτικών στρατηγικών

Μια γενική αναφορά στους Εξελικτικούς Αλγορίθμους (ΕΑ), έγινε στο υποκεφάλαιο 1.3. Αφού λοιπόν οι ΕΑ είναι τόσο χρήσιμοι και μπορούν να λύνουν πολύπλοκα προβλήματα με πολλές παραμέτρους που απαιτούν πολλούς συνδυασμούς, μπορούν να χρησιμοποιηθούν και στο συγκεκριμένο σύστημα. Η Χριστοδούλου(2010) χρησιμοποίησε τη κατηγορία των Γενετικών Αλγορίθμων (ΓΑ) και προσπάθησε να βρει ένα συνδυασμό παραμέτρων για το δίκτυο (πίνακας 4.1) ώστε να μπορέσει να βρει ένα βέλτιστο ποσοστό απόδοσης. Η αντικειμενική συνάρτηση την οποία προσπάθησε να βελτιστοποιήσει είναι η παράμετρος απόδοσης Q3. Εμείς στη τρέχουσα εργασία θα προσεγγίσουμε το θέμα από διαφορετική σκοπιά και χρησιμοποιώντας διαφορετικούς αλγόριθμους.

ΠΑΡΑΜΕΤΡΟΣ
Hidden_layer_one_size
Hidden_layer_two_size
Hidden_layer_one_of_Backward_size
Hidden_layer_two_of_Backward_size
Hidden_layer_one_of_Forward_size
Hidden_layer_two_of_Forward_size
Learning_Rate
Momentum
Window_size
q_minus_1
q_plus_1

Πίνακας 4.1: *Οι παράμετροι του δικτύου για τις οποίες η Χριστοδούλου(2010) προσπάθησε να βρει ένα συνδυασμό τιμών πουν να βελτιστοποιούν το ποσοστό Q3.*

Οι γενετικοί αλγόριθμοι που χρησιμοποίησε η Χριστοδούλου (2010) ακολουθούν την ίδια αρχιτεκτονική των ΓΑ που σχεδιάστηκαν και υλοποιήθηκαν από τον Αγαθοκλέους (2009). Χωρίζονται σε δύο κλάσεις (Chromosome.cpp, GeneticAlgorithm.cpp), οι οποίες υλοποιούν τον αλγόριθμο των γενετικών όπως είδαμε και στο Κεφάλαιο 1, οι οποίες ενσωματώνονται στο πρόγραμμα.

Η κωδικοποίηση του χρωμοσώματος γίνεται με δυαδική κωδικοποίηση και αφορά τις παραμέτρους του πίνακα 4.1, ανάλογα φυσικά και με το εύρος των τιμών της κάθε μιας. Αφού ολοκληρωθεί μια γενεά (αριθμός χρωμοσωμάτων), επιλέγεται το καλύτερο χρωμόσωμα για την συγκεκριμένη γενεά. Κύριος τρόπος αναπαραγωγής στην υλοποίηση αυτή είναι η μέθοδος της διασταύρωσης. Μετά ακολουθούν οι διαδικασίες της επιλογής, μετάλλαξης, και το δίκτυο προχωρεί στην επόμενη γενεά. Το κριτήριο τερματισμού είναι ένας αριθμός γενεών που πρέπει να εκτελεστούν.

Λόγω της φύσης του προβλήματος και της μεγάλης πολυπλοκότητας η μέθοδος αυτή δεν βόηθησε το σύστημα και δεν επέφερε τα επιθυμητά αποτελέσματα. Το σύστημα στη τελική του κατάσταση χρειάζεται μεγάλη χρονική διάρκεια για να εκπαιδευτεί, περίπου 4 μέρες. Επομένως για να επιτύχουμε βελτιστοποιήσεις με τη μέθοδο αυτή θα πρέπει να τρέξουμε το πρόγραμμα για πολλές επαναλήψεις-γενεές αφού οι παράμετροι που προσπαθεί να βελτιστοποιήσει είναι πολλές. Αυτό σημαίνει ότι το πρόγραμμα θα τρέχει για εβδομάδες, ίσως και μήνα, κάτι που δεν είναι καθόλου βολικό γιατί έχουμε 10 σύνολα δεδομένων που θέλουμε να τρέξουμε για να πάρουμε ολικά αποτελέσματα. Επίσης η υλοποίηση αυτή παρουσίαζε κάποια παράξενη συμπεριφορά και έδινε αποτελέσματα που δεν μπορούσαν εξηγηθούν. Έγινα κάποιες αλλαγές από τη Χριστοδούλου για την επίλυση των προβλημάτων αυτών αλλά και πάλι οι ΓΑ δεν αύξησαν το πόστο επιτυχίας.

Στη τρέχουσα εργασία όπως αναφέραμε ενσωματώσαμε μια άλλη κατηγορία Εξελικτικών Αλγορίθμων και προσεγγίσαμε το πρόβλημα από διαφορετική πλευρά ελπίζοντας ότι θα επιτύχουμε καλύτερα αποτελέσματα από πριν και σε λιγότερο χρόνο.

Η Χριστοδούλου προσπάθησε να βελτιστοποιήσει με τη χρήση των ΓΑ όλες τις παραμέτρους του προβλήματος (Πίνακας 4.1). Εμείς αντίθετα θα προσπαθήσουμε να βελτιστοποιήσουμε δύο παραμέτρους του προβλήματος, αλλά κάθε μία ξεχωριστά σε διαφορετικά συστήματα . Η μία παράμετρος που πιστεύουμε ότι αν εξελιχτεί θα δώσουν καλύτερα αποτελέσματα είναι η τιμή των βαρών του BRNN ενώ η άλλη είναι η επιρροή του κάθε δικτύου του Ensemble στο τελικό αποτέλεσμα. Από τις παραμέτρους που θα εξελίξουμε είναι φανερό ότι δεν μπορούμε να χρησιμοποιήσουμε πλέον τους ΓΑ, γιατί έχουμε να κάνουμε με δεκαδικές τιμές. Μπορούν φυσικά να χρησιμοποιηθούν για δεκαδική κωδικοποίηση αλλά όσο αυξάνεται η ακρίβεια του δεκαδικού τμήματος τόσο αυξάνεται και η πολυπλοκότητας τους. Για αυτό το λόγο θα χρησιμοποιήσουμε τις Εξελικτικές Στρατηγικές που δημιουργήθηκαν για την επίλυση και κωδικοποίηση τέτοιων προβλημάτων.

4.4.1 Εξέλιξη του Βαθμού Συμμετοχής του κάθε BRNN στο Ensemble

Θα ξεκινήσουμε με τη περιγραφή της δεύτερης προσέγγισης που αναφέρθηκε και η οποία είναι σχετικά πιο απλή. Σε αυτή τη προσέγγιση θα προσπαθήσουμε να αλλάξουμε το ρόλο που διαδραματίζει κάθε μέλος του Ensemble στο τελικό αποτέλεσμα. Όπως εξηγήθηκε στο υποκεφάλαιο 4.2 τα καλύτερα αποτελέσματα για Ensemble πάρθηκαν με τη μέθοδο Average. Σύμφωνα με τη μέθοδο αυτή, αφού έχουμε 6 BRNN καθένα από αυτά λαμβάνει μέρος στο τελικό αποτέλεσμα σε αναλόγια 1/6. Όμως αυτό δεν είναι πάντα καλό γιατί για κάποια πρότυπα κάποιο από τα έξι δίκτυα ίσως εκπαιδεύτηκε καλύτερα από τα άλλα. Άρα θα πρέπει να έχει μεγαλύτερο ποσοστό αναλογίας στο τελικό αποτέλεσμα. Σημαντικό να σημειωθεί ότι το άθροισμα της αναλογίας όλων των BRNN πρέπει να ισούται με 1.

Η διαδικασία εκτέλεσης του αλγόριθμου φαίνεται που ακολουθείται φαίνεται στο πιο κάτω Σχήμα 4.15.

1. Κωδικοποίηση (Coding)
2. Επιλογή αντικειμενικής συνάρτησης (Initialization)
3. Δημιουργία πληθυσμού.
4. Ανάγνωση δεδομένων εισόδου για καθένα από τα έξι δίκτυα BRNN και αποθήκευση τους.
5. Για κάθε πρωτεΐνη
    a. Για κάθε αμινοξύ
        i. Διάβασε τις τιμές εξόδου κάθε δικτύου BRNN.
        ii. Υπολόγισε τη τιμή εξόδου
        iii. Αν η πρόβλεψη ήταν σωστή αύξησε το rank του συγκεκριμένου ατόμου.
6. Δημιουργία προσωρινού πληθυσμού
    i) Μετάλλαξη κάθε ατόμου για δημιουργία του offspring
    ii) Επανέλαβε το βήμα 5
7. Επιλογή (Selection)
    a. Κατάταξε τα άτομο και των δύο πληθυσμών θέτωντας πρώτο αυτό με το ψηλότερο fitness

Επανάληψη από το βήμα (4) μέχρι να ικανοποιηθεί το κριτήριο τερματισμού της ΕΣ.

Σχήμα 4.15: *Ένας απλός αλγόριθμος Εξελικτικής Στρατηγικής.*

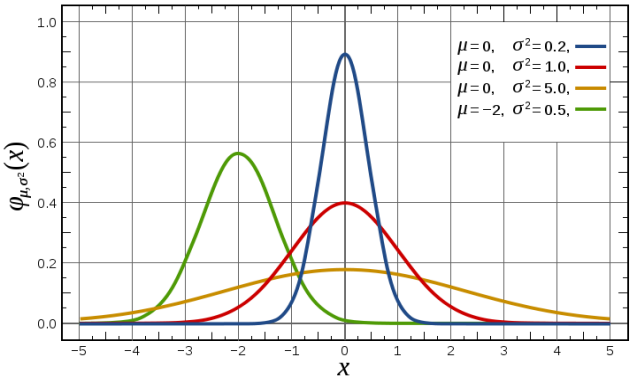
Κάθε BRNN έχει τρεις εξόδους. Αφού έχουμε έξι BRNN, ο συνολικός αριθμός των τιμών που θα εξελίξουμε είναι 18. Επομένως κάθε χρωμόσωμα (*"chromosome.h"*) θα κωδικοποιείται από ένα πίνακα δεκαδικών τιμών, μεγέθους 18. Κάθε χρωμόσωμα έχει μια αξία που υπολογίζεται σύμφωνα με το *fitness function*. Το fitness function το ορίσαμε ως συνάρτηση μεγιστοποίηση γιατί σχετίζεται με το Q3 ποσοστό που επιτεύχθηκε για το συγκεκριμένο σύνολο δεδομένων.

Στη κλάση "*ga.h*" θα δημιουργήσουμε ένα πληθυσμό από χρωμοσώματα, μεγέθους 200, ενώ παράλληλα θα έχουμε και ένα προσωρινό πληθυσμό, στον οποίο θα γίνονται οι μετάλλαξη και η επιλογή. Η διαδικασία της μετάλλαξης στις ΕΣ γίνεται για κάθε άτομο αφού είναι και ο βασικός τρόπος αναπαραγωγής. Επίσης η συνεχής μετάλλαξη αποτρέπει την περίπτωση να κολλήσουμε σε κάποιο τοπικό ελάχιστο. Σε κάθε γενεά το κάθε άτομο του προσωρινού πληθυσμού αποτελεί τη μετάλλαξη ενός ατόμου από το πληθυσμού μας. Η μετάλλαξη ακολουθεί επιτυγχάνεται με τη χρήση της Gaussians η οποία μας εγγυάται ότι το άτομο που θα δημιουργηθεί είναι σχετικά παρόμοιο με το γονιό του. Από το σχήμα 4.16 βλέπουμε ότι το μεταλλαγμένο άτομο είναι μια μικρή πρόσθεση ή αφαίρεση στο γονιό του.

```
Για κάθε παράμετρό που είναι κωδικοποιημένη σε ένα άτομο
{
   g=gaussian(0,1);
   προσωρινός_πλυθησμός[i].άτομο[j]=πλυθησμός[i].άτομο[j]+g
}
```

Σχήμα 4.16:*Ψευδοκώδικας που υλοποιεί τη λειτουργία της μετάλλαξης σε κάθε χρωμόσωμα*

Από τη κυμματομορφή της Gaussian (Σχήμα 4.17) παρατηρούμε ότι ανάλογα με τη μέση τιμή και τη τυπική απόκλιση πέφτει ή απομακρύνεται από το +∞ η -∞ με διαφορετικό ρυθμού. Εμεις θέλουμε οι τυχαίες αλλαγές που θα γίνονται στο άτομο να είναι μικρές και να ακολουθούν την κανονικη κατανομή για να μπορέσουμε να πλησιάσουμε όσο το δύνατό στη βέλτιστη κωδικοποιήση του χρωμοσώματός



Σχήμα 4.17: *Διάφορες Κατανομές της Gaussian Συνάρτησης για διαφορετική μέση τιμη (μ) και τυπική απόκλιση (σ^2).*

Για τη υλοποιήση της Gaussian κατανομή χρησιμοποιήσε τη μέθοδο που φαίνεται στο σχημα 4.18.

```
double ga:: uniform(const double min, const double max)
{
    return min + rand()/(RAND_MAX/(max - min));
}

double ga:: gaussian(double mu, double sigma)
{
    double p = 1.0, p1, p2;
    double result=0;
    while (p >= 1.0)
    {
        p1 = uniform(-1, 1);
        p2 = uniform(-1, 1);
        p = p1 * p1 + p2 * p2;
    }
    result=mu + sigma * p1 * sqrt(-2.0 * log(p) / p);

    return result;
}
```

Σχήμα 4.18: *Κώδικας που υλοποιεί τη Gaussian Κατανομή*

Στο τελικό στάδιο γίνεται η επιλογή του νέου πληθυσμού. Η μέθοδος που χρησιμοποιήσαμε για επιλογή είναι το Tournament Selection. Σύμφωνα με τη μέθοδο αυτή σε κάθε άτομο και από τους δύο πληθυσμούς δίνεται μια θέση κατάταξης σύμφωνα με τιμή του fitness function. Όσο πιο μεγάλη είναι η τιμή του fitness του , τόσο πιο ψηλή θα είναι η θέση στη κατάταξη. Για την επόμενη γενεά θα επιλέξουμε του καλύτερους για να συνεχίσουν, δηλαδή τα 200 άτομα με τη ψηλότερη τιμή fitness(Σχήμα 4.19).

```
void ga :: chooseBestIndividual()
{

      Δημιουργία πλυθησμού temp[POP*2]

      Αποθήκευσε άτομα πλυθησμού και προσωρινού_πλυθησμού στο temp[]
    temp=new chromosome [POP*2];

      Ταξινόμηση temp κατατάσσοντας τα ατόμα σε σειρά βάση το fitness
      function

      Επέστρεψε τα άτομα του temp που βρίσκονται στις POP πρώτες
      θέσεις για να συνεχίσουν στην επόμενη γενεά


}
```

Σχήμα 4.19: *Στο πιο πάνω κομμάτι κώδικα γίνεται κατάταξη των δύο πλυθησμών ανάλογα με τη τιμή του fitness τους. Οι 200 καλύτεροι θα επιλεκτούν για να συνεχίσουν στην επόμενη γενέα γιατί η κωδικοποιήση τους έδωσε τα καλύτερα αποτελέσματα και άρα εμπεριέχουν χρήσιμη πληροφορία.*

Αυτό θα επαναληφθεί μέχρι ένα μέγιστο αριθμό γενεών. Στη τελευταία γενεά το άτομο με το ψηλότερο fitness θα επιλεχτεί για να δοκιμαστεί σε ένα σύνολο δεδομένων δοκιμής για να υπολογιστεί το τελικό ποσοστό επιτυχίας.

4.4.2 Εξέλιξη των τιμών των βαρών του BRNN

Η δεύτερη μέθοδος που χρησιμοποιήσαμε αφόρα τα βάρη του BRNN. Συγκεκριμένα θα προσπαθήσουμε να εξελίξουμε τις τιμές των βαρών του BRNN για να επιτύχουμε καλύτερη πρόβλεψη. Υπολογίσαμε ότι ο αριθμός των βαρών που έχουμε να εξελίξουμε, σύμφωνα με τις παραμέτρους που έδινα τα καλύτερα αποτελέσματα (Κεφάλαιο 5), είναι περίπου 30 000. Ο αριθμός αυτός περιλαμβάνει τα βάρη και των τριών δικτύων που συναποτελούν το BRNN, σε όλα τα επίπεδα.

4.4.2α Εξέλιξη των τιμών των βαρών χωρίς τη χρήση του Back-Propagation

Για την υλοποίηση της μεθόδου αυτή χρησιμοποιήσαμε δυο στρατηγικές. Στην πρώτη περίπτωση αφαιρέσαμε τελείως τη μάθηση με αναστροφή λάθους από την εκπαίδευση του BRNN και η μάθηση πλέον γίνεται μόνο με τη χρήση των ΕΣ. Το κάθε άτομο του πληθυσμού όπως και πριν έχει μια αξία που υπολογίζεται σύμφωνα με το fitness function. Στη συγκεκριμένη περίπτωση το fitness function υπολογίζει το mean square error της πραγματικής τιμής σε σχέση με την επιθυμητή. Επομένως έχουμε να κάνουμε με συνάρτηση ελαχιστοποίησης γιατί όσο πιο μικρό είναι το σφάλμα τόσο πιο ”καλό” είναι το συγκεκριμένο άτομο του πληθυσμού. Οι τιμές των βαρών αρχικοποιούνται όπως και πριν τυχαία.

Οι λειτουργίες της μετάλλαξης και της επιλογής εκτελούνται ακριβώς οι ίδιες όπως και πριν με μόνη διαφορά ότι στη φάση της επιλογής, οι καλύτεροι είναι αυτοί που έχουν τη μικρότερη τιμή στο fitness function.

```
Για κάθε γενεά
{
  Για κάθε άτομο του πληθυσμού
  {
      Αποκωδικοποίηση χρωμοσώματος και αρχικοποίηση βαρών BRNN βάση του χρωμοσώματος
      Όσον υπάρχουν πρωτεΐνες στο training dataset
      {
        Πάρε την κωδικοποίηση της επόμενης πρωτεΐνη x βάση του MSA
        Για κάθε αμινοξύ t της πρωτεΐνης x
            doFeedForward(t)
        }
  Αποθήκευσε το EpochError ως τιμή του Fitness Function
}
  Μετάλλαξη χρωμοσωμάτων με τη χρήση τη Gaussian
  Για κάθε άτομο του προσωρινού πληθυσμού
  {
      Αποκωδικοποίηση χρωμοσώματος και αρχικοποίηση βαρών BRNN βάση του χρωμοσώματος
      Όσον υπάρχουν πρωτεΐνες στο training dataset
      {
        Πάρε την κωδικοποίηση της επόμενης πρωτεΐνη x βάση του MSA
        Για κάθε αμινοξύ t της πρωτεΐνης x
            doFeedForward(t)
       }
  Αποθήκευσε το EpochError ως τιμή του Fitness Function
}
Επιλογή 200 καλύτερων ατόμων βάση mean square error
Εάν η γενεά είναι η τελευταία τότε επέλεξε το καλύτερο άτομο και δοκίμασε το TestSet
```

Σχήμα 4.20: *Ψευδοκώδικας που υλοποιεί την εξέλιξη των τιμών των βαρών του BRNN χωρίς τη χρήση Back-Propagation.*

4.4.2β Εξέλιξη των τιμών των βαρών μετά τη εκπαίδευση του δικτύου με Back-Propagation

Στη δεύτερη περίπτωση η αρχιτεκτονική των ΕΣ είναι η ιδία όπως φαίνεται στο σχήμα 4.23 αλλά εφαρμόζεται αφότου γίνει η εκπαίδευση του δικτύου. Δηλαδή το BRNN εξακολουθεί να εκπαιδεύεται κανονικά με το Back-Propagation (Υποκεφάλαιο 2.3, Σχήμα 2.5) και μετά το τέλος της εκπαίδευσης του εφαρμόζεται ο Εξελικτικός Αλγόριθμος (Σχήμα 4.20). Προσπαθούμε στη ουσία να βελτιώσουμε τα βάρη του δικτύου αφότου αυτά εκπαιδεύτηκαν από το BRNN.

Το BRNN με Back-Propagation γνωρίζουμε ότι επιφέρει καλά αποτελέσματα. Επομένως αν γίνει η εκπαίδευση και μετά εφαρμοστεί η ΕΣ είμαστε σίγουροι, σε σχέση με τη προηγούμενη υλοποίηση, ότι το ποσοστό επιτυχίας θα φθάσει στα επιθήματα ποσοστά και με τις μικρές βελτιώσεις που θα εφαρμοστούν στα βάρη του δικτύου με τους ΕΣ ευελπιστούμε ότι τα θα αυξήσουν το ποσοστό επιτυχίας.

4.4.3 Ενσωμάτωση του SOV ως *fitness function*

Από τα πειράματα που έγιναν (Κεφάλαιο 5) παρατηρήσαμε ότι κάποια αισθητή βελτίωση φάνηκε στην τελευταία υλοποίηση (Σχήμα 4.20). Όπως αναφέραμε στο Υποκεφάλαιο 1.4 ένα από τα πιο σημαντικά πλεονεκτήματα το Εξελικτικών Αλγορίθμων είναι το η ιδιότητα τους να εφαρμόζεται και να τροποποιούνται με μεγάλη ευκολία. Αφού τα πειράματα έδειξαν κάποια βελτίωση σκεφτήκαμε να κάνουμε μια μικρή τροποποίηση στον αλγόριθμο αφού η μορφή του είναι τέτοια που επιτρέπεται. Η τροποποίηση που θα κάνουμε αφόρα τον τρόπο αξιολόγησης του fitness function. Στις πιο πάνω περιπτώσεις το fitness υπολογίζεται είτε με το mean square error είτα βάση του Q3. Σκεφτήκαμε λοιπόν να δοκιμάσουμε ΕΣ στην χ υλοποίηση αλλά το fitness function του θα υπολογίζεται βάση του SOV, κάτι που δεν έχει ξαναδοκιμαστεί προηγουμένως.

Ο τρόπος εύρεσης του SOV Score (Υποκεφάλαιο 2.4.2) είναι σχετικά περίπλοκος και για να υπολογιστεί χρειάζεται τη βοήθεια του προγράμματος sov.c. Οι προηγούμενες υλοποιήσεις του BRNN δεν μπορούσαν να εκπαιδεύσουν το δίκτυο βάση του SOV αλλά αντίθετα υπολογιζόταν στο τέλος της εκπαίδευσης ως ένα κριτήριο αξιολόγησης του BRNN. Αντίθετα με του ΕΣ μπορούμε εύκολα να το ενσωματώσουμε απλά αλλάζοντας το fitness function. Η δομή του είναι τέτοια που είναι εύκολα επεκτάσιμοι.

Για κάθε άτομο του πληθυσμού δημιουργούμε ένα αρχείο στο φάκελο SOV/ . η μορφή του αρχείο είναι αυτή που μπορεί να αναγνωρίσει το πρόγραμμα sov.c και φαίνεται στο σχήμα 4.24. Έτσι για κάθε γενεά αποθηκεύουμε το αποτέλεσμα της δευτεροταγή δομής για κάθε πρωτεΐνη στο αντίστοιχο αρχείο. Το ίδιο επαναλαμβάνεται και για το προσωρινό πληθυσμό που δημιουργείται με μετάλλαξη. Συνεπώς για πληθυσμό 200 ατόμων θα δημιουργήσουμε 400 αρχεία.

```
>OSEC
LLLLEEEELLLLEEEELLLLLLLLLLLLLLLEELLLLEEEEEEELLEEEEEE
LLLLEEEEEELLLLLLEELLLLEEEEEL
>PSEC
LLLLLEEELLLLLEEEELLLLLLLEEEELLLEEELLLEEEEEEHHHHEHELL
LLLLLEEEEEEELLLLEEELLLEEEEEL
>OSEC
LLEEEEELLLLEEEEELLLLLEEEELLLLLHHHLEEEEELLLLEEEEELLLL
EEEEELLLLLEEEELLLLHHHLEEEEELLLLLEEEEELLLLLLELLELLLLLE
LLHHHLLLLLHHHLEEEEEL
>PSEC
LLLEEEEELLLEEEELLLLLELLLLLLLLLLLEEEEEELLLEEEEELEELL
EEEEELLLLEEEEELLLLLLLLLEHEHLLLLLLHHHLLLLLLEEEEELLLLL
LLLLLLLLLLLLLLLLLLL
>OSEC
LLLLLLLHHHLLLLLLLLLL
>PSEC
LLLLLLLLHLEEEEEEHHHEL
```

Σχήμα 4.21: *Παράδειγμα Μορφής Αρχείου SovtempscriptX.txt όπου X είναι ο αριθμός του ατόμου του πλυθησμού.*

Στο τέλος της γενεάς τρέχουμε το πρόγραμμα sov.c για κάθε αρχείο για να δημιουργηθεί το αρχείο *"OtX.txt"* που περιέχει μέσα το ποσοστό σε SOV. Αφού υπολογίσουμε το SOV μπορούμε να προχωρήσουμε στη φάση της επιλογή όπου θα επιλεχτούν τα καλυτέρα, αυτά δηλαδή με το ψηλότερο ποσοστό σε SOV.

```
-----------------------

SECONDARY STRUCTURE PREDICTION ACCURACY EVALUATION.  N_AA =  212

                                ALL    HELIX   STRAND     COIL

Q3                         :    72.2     63.6     64.4     79.5

SOV                        :    55.4     72.7     65.6     47.2

-----------------------
```

Σχήμα 4.22*: Τμήμα του αρχείο OtX.txt, όπου X ο αριθμός ατόμου του πληθυσμού. Από το τμήμα αυτό θα πάρουμε το SOV για να το αποθηκεύσουμε ως αποτέλεσμα του fitness function.*

Από τη δομή του προγράμματος μπορούμε να δούμε ότι χρειάζεται πολύ χρόνο για να τελειώσει. Πέραν από το χρόνο που χρειάζεται γα να εκπαιδευτεί το σύστημα με Back-Propagation τώρα προστίθεται και ο χρόνος που απαιτείται για να εξελιχτούν τα βάρη. Η διαδικασία του ανοίγματος και εγγραφής σε αρχεία για κάθε άτομο του πληθυσμό για κάθε γενεάς αυξάνει εκθετικά το χρόνο εκπλήρωσης του προγράμματός. Όμως από τα αποτελέσματα που φαίνονται στο κεφάλαιο 5 παρατηρούμε το ποσοστό βελτίωσης σε SOV Score φτάνει μέχρι και το 74%.

Η υλοποίηση του της μεθόδου αυτή φαίνεται στο πιο κάτω Σχήμα 4.23.

```
Εκπαίδευσε κανονικά το BRNN με τη χρήση του Back-Propagation
Για κάθε γενεά
{
  Για κάθε άτομο του πληθυσμού
  {
        Αποκωδικοποίηση χρωμοσώματος και αρχικοποίηση βαρών BRNN βάση του χρωμοσώματος
        Δημιουργία Αρχείων, ένα για κάθε άτομο.
        Όσον υπάρχουν πρωτεΐνες στο training dataset
        {
          Πάρε την κωδικοποίηση της επόμενης πρωτεΐνη x βάση του MSA
          Για κάθε αμινοξύ t της πρωτεΐνης x
                doFeedForward(t)
          }
  Αποθήκευσε στο αρχείο του ατόμου τη κατάσταση που προβλέφθηκε στη κατάλληλη μορφή
}
  Μετάλλαξη χρωμοσωμάτων με τη χρήση τη Gaussian
  Για κάθε άτομο του προσωρινού πληθυσμού
  {
        Αποκωδικοποίηση χρωμοσώματος και αρχικοποίηση βαρών BRNN βάση του χρωμοσώματος
        Όσον υπάρχουν πρωτεΐνες στο training dataset
        {
          Πάρε την κωδικοποίηση της επόμενης πρωτεΐνη x βάση του MSA
          Για κάθε αμινοξύ t της πρωτεΐνης x
                doFeedForward(t)
          }
  Αποθήκευσε στο αρχείο του ατόμου τη κατάσταση που προβλέφθηκε στη κατάλληλη μορφή
}
        Για κάθε άτομο
        Εύρεση του SOV Score με τη βοήθεια των αρχείων που δημιουργήθηκαν και τη χρήση του προγράμματος sov.c
        Αποθηκεύση του SOV ως τιμή του fitness function
Επιλογή των 200 καλύτερων ατόμων με τη χρήση του Tournament Selection
Εάν η γενεά είναι η τελευταία τότε επέλεξε το καλύτερο άτομο και δοκίμασε το TestSet
```

Κεφάλαιο 5

Πειράματα και ανάλυση αποτελεσμάτων

Στο υπάρχον σύστημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, χρειάστηκε να γίνουν κάποιες αλλαγές, όπως αναφέρθηκε και στο κεφάλαιο 3 και 4, ούτως ώστε να διασφαλιστεί η σωστή λειτουργία του. Αρχικά, τροποποιήσαμε το σύνολο δεδομένων με το οποίο εκπαιδεύουμε το δίκτυο μας, αφού πλέον χρησιμοποιούμε το σύνολο CB513 (Κεφάλαιο 3). Επιπλέον ενσωματώσαμε το παράθυρο για το κεντρικό δίκτυο του BRNN το οποίο αλλάζει το ποσό πληροφορίας που επεξεργάζεται το δίκτυο για κάθε αμινοξύ. Αυτό εξυπακούει ότι πρέπει να ξαναγίνουνε τα πειράματα για την εύρεση των βέλτιστων τιμών των παραμέτρων αφού.

Γενικά, μετά από οποιαδήποτε αλλαγή στο σύστημα, πρέπει να ελέγχουμε ξανά την απόδοση του δικτύου, ώστε να καταλάβουμε εάν όντως έχει επηρεαστεί και βασικά, εάν έχει βελτιωθεί το αποτέλεσμα σε σύγκριση με τα αποτελέσματα χωρίς τις τροποποιήσεις αυτές. Δε θα γίνει λεπτομερής σύγκριση με τα αποτελέσματα της Χριστοδούλου (2010) για το λόγο ότι έχουν γίνει αλλαγές και έχουν χρησιμοποιηθεί διαφορετικά σύνολα δεδομένων εκπαίδευσης, οι οποίες καταστούν δύσκολή την ακριβής σύγκριση. Θα γίνει όμως μια τελική σύγκριση για να δούμε κατά πόσο οι νέες μέθοδοι και τεχνικές που χρησιμοποιήθηκαν αύξησαν το ποσοστό επιτυχίας σε σχέση με το τελικό ποσοστό που είχε επιτύχει. Η Χριστοδούλου (2010) με τις τροποποιήσεις που εισήγαγε πέτυχε Q3 ποσοστό μέχρι και 76%, ενώ χρησιμοποιώντας *Hidden Markov Models* (HMM) για φιλτράρισμα της εξόδου, το SOV ποσοστό αυξάνεται μέχρι και 70%.

Γενικά όμως θα δημιουργήσουμε εκ νέου πειράματα ώστε να μελετήσουμε την μεταβολή της απόδοσης του δικτύου με τις διάφορες αλλαγές στις παραμέτρους σε σχέση κάθε φορά με το προηγούμενο ποσοστό που επιτύχαμε.

Όπως προαναφέρθηκε και επεξηγήθηκε στο Κεφάλαιο 3 χρησιμοποιήσαμε το CB513 για την εκπαίδευση του δικτύου. Το CB513 είναι ένα ευρέως διαδεδομένο και αναγνωρισμένο σύνολο από πρωτεΐνες. Επομένως τα αποτελέσματα μας θα είναι συγκρίσιμα με τα αποτελέσματα άλλων εφαρμογών που αναπτύχθηκαν για την επίλυση του συγκεκριμένου προβλήματος.

5.1 Πειράματα για εύρεση βέλτιστων τιμών παραμέτρων που αφορούν το κεντρικό Δίκτυο του BRNN

Οι παράμετροι με τις οποίες ξεκινάμε να εκπαιδεύουμε το δίκτυο φαίνονται στον πίνακα 5.1. Οι παράμετροι αυτές πάρθηκαν από τις παραμέτρους που έδωσαν τα καλύτερα αποτελέσματα στα πειράματα που έτρεξε η Χριστοδούλου (2010). Το καλύτερο ποσοστό ακριβείας με βάση τις τιμές αυτές, είναι 76.0% χρησιμοποιώντας την παράμετρο Q3 και 70.0% για τη παράμετρο SOV. Πιστεύουμε ότι οι αρχικές αυτές τιμές των παραμέτρων, είναι μια καλή αρχή για να ξεκινήσουμε τα νέα πειράματα, για να δούμε αν η εισαγωγή των νέων τεχνικών και βελτιστοποιήσεων επιφέρει καλύτερα αποτελέσματα με αυτές τις τιμές η πρέπει να αλλαχτούν. Μια σημαντική διαφορά είναι όπως βλέπουμε στο Πίνακα 5.1 η εισαγωγή της νέας παραμέτρου 14 (“center_window_size”)που αφορά το μέγεθος του κεντρικού παραθύρου.

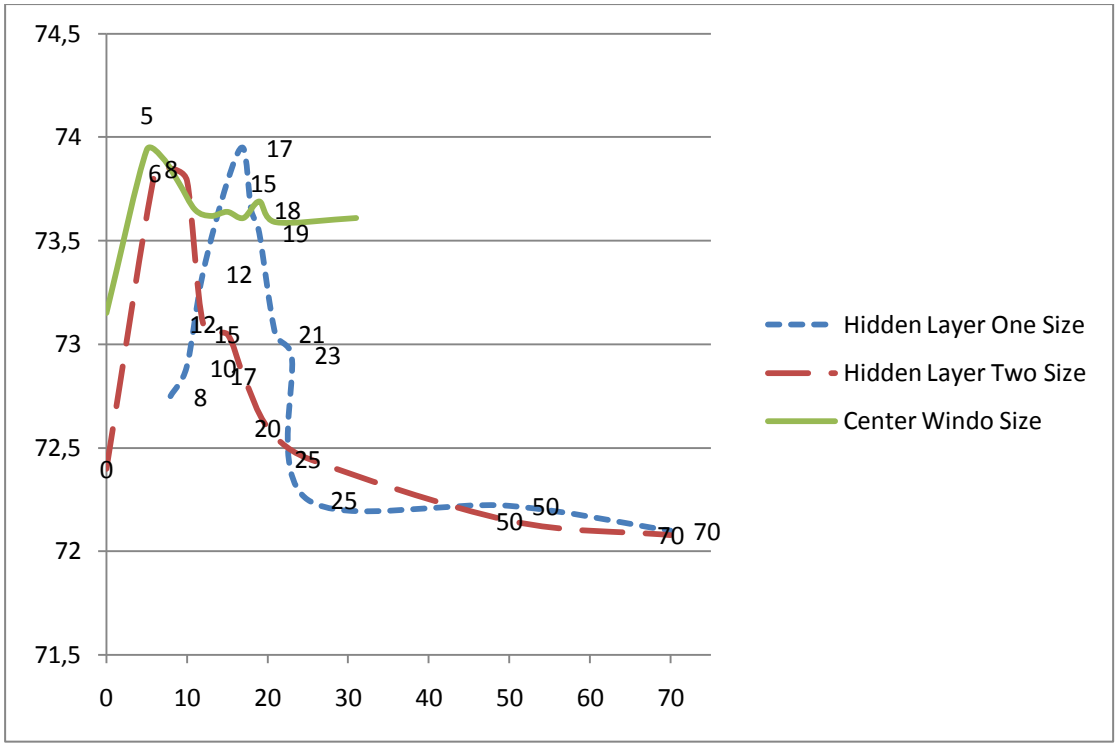
No	ΠΑΡΑΜΕΤΡΟΣ	ΤΙΜΗ
1	Hidden_layer_one_size	17
2	Hidden_layer_two_size	0
3	Hidden_layer_one_of_Backward_size	15
4	Hidden_layer_two_of_Backward_size	0
5	Hidden_layer_one_of_Forward_size	15
6	Hidden_layer_two_of_Forward_size	0
7	Learning_Rate	0.5
8	Momentum	0.1
9	Window_size	15
10	q_minus_1	0.5
11	q_plus_1	0.5
12	S	3
13	Maximum_Iterations	200
14	Center_window_size	5

Πίνακας 5.1: *Αρχικές τιμές για τις παραμέτρους του δικτύου.*

Για τα διάφορα πειράματα που ακολουθούν, κάθε φορά που δοκιμαζόταν μια νέα τιμή κρατούσαμε τις υπόλοιπες παραμέτρους του δικτύου που οδήγησαν τη Χριστοδούλου (Χριστοδούλου, 2010) στο αποτέλεσμα 76.0%, σταθερές (Πίνακας 5.1). Με αυτόν τον τρόπο θα δούμε εύκολα συνδυασμούς τιμών στις παραμέτρους που επηρεάζουν, αλλά και σε πιο βαθμό, την εκπαίδευση και κατά συνέπεια, την απόδοση του συστήματος. Να σημειωθεί ότι έγιναν πολλά πειράματα και δοκιμάστηκαν πολλοί συνδυασμοί , κρατώντας σταθερά τα βάρη, για να βρεθεί το καλύτερο αποτέλεσμα στις τιμές των παραμέτρων. Επομένως οι τιμές των παραμέτρων που επιλέξαμε αποτελούν το πιο πιθανόν το "βέλτιστο" συνδυασμό για το δίκτυο μας. Ίσως κάποιος άλλος συνδυασμός να δώσει καλύτερο αποτέλεσμα, όμως το αποτέλεσμα θα είναι ελάχιστα διαφορετικό και ίσως τυχαίο. Στα πρώτα πειράματα που έγιναν για την εύρεση των βέλτιστων τιμών προτιμήσαμε να επιλέξουμε ως μέτρο σύγκρισης το Q3 που δίνει πιο ψηλά αποτελέσματα, ενώ το SOV χρησιμοποιείται στη συνέχεια όπου γίνεται το filtering και ενσωμάτωση των ES. Στη γραφική αναπαράσταση των αποτελεσμάτων μας εμφανίζουμε τις τιμές οι οποίες ήταν κομβικές και είναι εμφανής η διαφορά μεταξύ των πειραμάτων.

Κατ αρχήν, εισάγαμε το όρο του κεντρικού παράθυρου. Για το λόγο αυτό θα πειραματιστούμε με το μέγεθος των κρυφών επιπέδων του κεντρικού δικτύου για να βρούμε σε ποια τιμή έχουν το καλύτερο ποσοστό. Επίσης θα δοκιμάσαμε διάφορες τιμές για το κεντρικό παράθυρο που ενσωματώσαμε.

Όπως αναφέραμε στο Κεφάλαιο 3 τα δεδομένα μας είναι χωρισμένα σε 10 folds και κάθε υποσύνολο χρειάζεται πολλή ώρα για να τρέξει. Για το λόγο αυτό θα βρούμε πρώτα τις βέλτιστες τιμές των παραμέτρων μόνο για ένα υποσύνολο και στη συνέχεια τις καλύτερες θα τις δοκιμάσουμε για όλα τα υποσύνολα και θα βρούμε το μέσο όρο. Στα πειράματα που παρουσιάζονται πιο κάτω χρησιμοποιήσαμε το *DataSet1* το οποίο είναι σχετικά "δύσκολο", γιατί το ποσοστό επιτυχής πρόβλεψης του είναι μικρότερο συγκρινόμενο με τα άλλα.



Σχήμα 5.1: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με διάφορες τιμές για τις παραμέτρους Hidden_layer_one_size, Hidden_layer_one_size και window_center_size. Για τον υπολογισμό κάθε παραμέτρου κρατήσαμε σταθερές τις άλλες δύο και ίσες με τις τιμές που υπολογίστηκαν βάση του Πίνακα 5.1*

Στη πιο πάνω γραφική βλέπουμε πως κυμαίνονται οι διάφορες τιμές των τριών παραμέτρων σε σχέση με το Q3 ποσοστό. Όπως φαίνεται από τη γραφική παράσταση οι βέλτιστες τιμές για τις παραμέτρους *Hidden_layer_one_size*, *Hidden_layer_two_size* και *center_window_size* είναι στο 17, 8, και 5 αντίστοιχα. Παρατηρούμε επίσης ότι τα ποσοστά πρόβλεψης με μεγάλο μέγεθος κινητού παραθύρου είναι σχετικά μικρότερα από τα ποσοστά πρόβλεψης χρησιμοποιώντας μικρότερο μέγεθος κινητού παραθύρου. Όταν το μέγεθος του παραθύρου είναι 5, το αποτέλεσμα παίρνει την ψηλότερη του τιμή, 73.8%. Για να είμαστε σίγουροι ότι είναι πράγματι οι καλύτερες τιμές δοκίμασε διάφορες τιμές που κυμαίνονται κοντά σε αυτές τις τιμές και οι οποίες είχαν και αυτές καλά αποτελέσματα. Τα συγκριτικά αποτελέσματα φαίνονται στο Πινάκα 5.2.

Q3	Architecture layer by layer for Centre Network	Window Size	Architecture layer by layer for Fr –Br	Learning Rate	Momentum
73.54%	17 - 6	5	17-0-17-0	0.09	0.5
73.58%	17 - 6	7	17-0-17-0	0.09	0.5
73.93%	17 - 8	5	17-0-17-0	0.09	0.5
73.78%	17 – 8	7	17-0-17-0	0.09	0.5
73.71%	17 – 10	5	17-0-17-0	0.09	0.5
73.65%	17 – 10	7	17-0-17-0	0.09	0.5
73.83%	15 – 6	5	17-0-17-0	0.09	0.5
73.71%	15 – 6	7	17-0-17-0	0.09	0.5
73.79%	15 – 8	5	17-0-17-0	0.09	0.5
73.67%	15 – 8	7	17-0-17-0	0.09	0.5
73.7%	15 – 10	5	17-0-17-0	0.09	0.5
73.55%	15 – 10	7	17-0-17-0	0.09	0.5

Πίνακας 5.2: *Στην δεύτερη στήλη φαίνονται οι τιμές για τον αριθμό των νευρώνων σε κάθε επίπεδο του κεντρικού Δικτύου του BRNN. Η επόμενη στήλη αφορά το μέγεθος του κινητού παραθύρου του κεντρικού δικτύου, ενώ οι υπόλοιπες στήλες έχουν τις τιμές των άλλων παραμέτρων που κρατήθηκαν σταθερές και ίδιες με αυτές που πέτυχε τα καλύτερα αποτελέσματα η Χριστοδούλου (2010).*

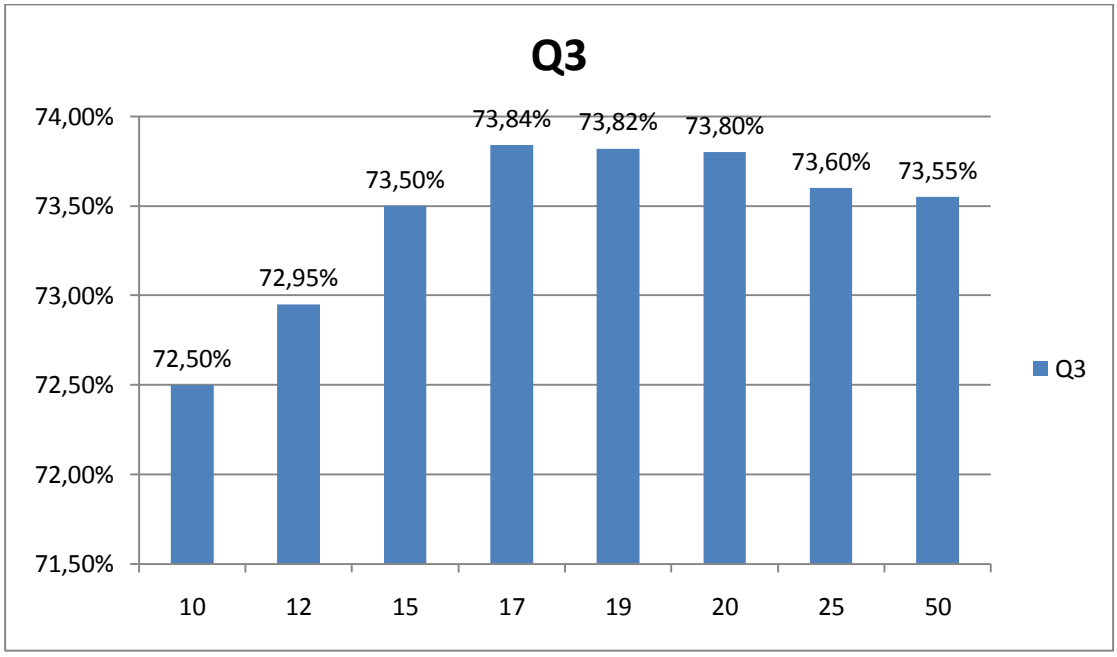
Στον Πίνακα 5.2 βλέπουμε τι γίνεται εάν αλλάξουμε την αρχιτεκτονική του δικτύου εφαρμόζοντας διαφορετικό αριθμό νευρώνων όσον αφορά το κεντρικό δίκτυο και το κεντρικό παράθυρο μόνο. Ο αριθμός των νευρώνων των άλλων δύο δικτύων, το μέγεθος του κινητού παραθύρου, καθώς και οι χρονικές μεταβλητές, ο ρυθμός μάθησης, και ο όρος της ορμής παραμένουν σταθερά για να βρούμε πως επηρεάζει η αρχιτεκτονική του κεντρικού δικτύου την εκπαίδευση του συστήματος. Όπως βλέπουμε τα αποτελέσματα του συστήματος είναι πολύ καλά όταν η αρχιτεκτονική είναι 17-8 και το κεντρικό παράθυρο μένει μικρό και ίσο με 5. Παρόλο που η

διαφορά όταν η αρχιτεκτονική είναι 15-6 είναι μικρή εμείς θα κρατήσουμε την πρώτη αρχιτεκτονική γιατί έστω και μικρή η διάφορα, υπάρχει.

5.1.2 Πειράματα και αποτελέσματα με αλλαγή τιμών των παραμέτρων του δικτύου

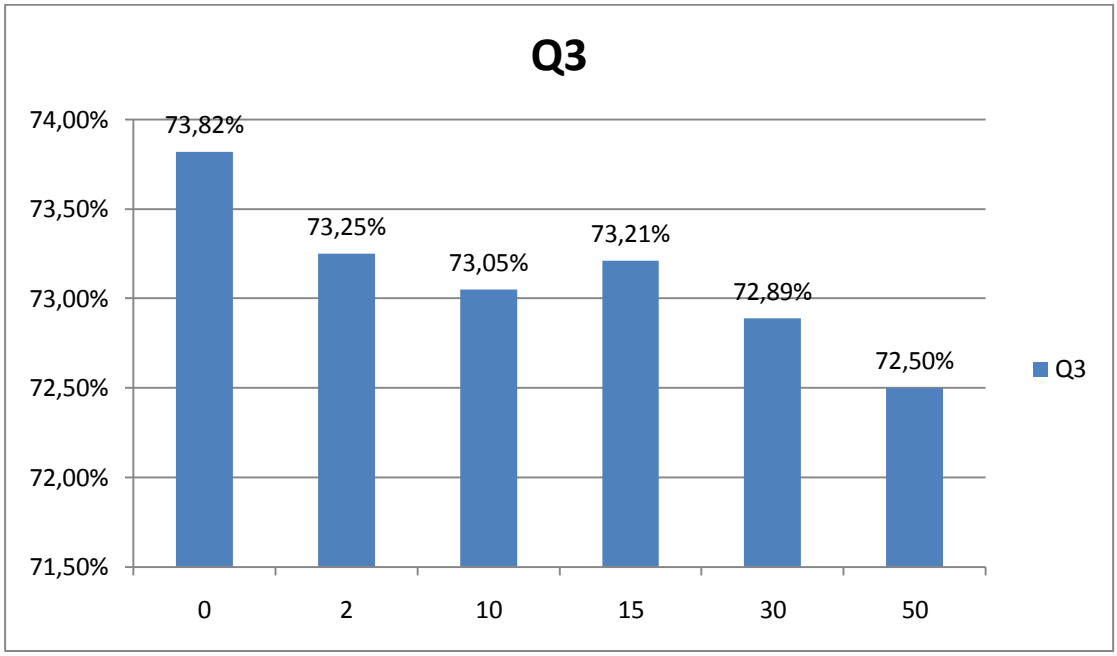
Στη συνέχεια θα τροποποιήσουμε τον αριθμό των νευρώνων των άλλων δύο δικτύων, το εμπρόσθιο (Fr) και το πίσω (Br). Λογικά επειδή οι αλλαγές που έγιναν μέχρι τώρα δεν επηρεάζουν τα άλλα δύο δίκτυα αυτό σημαίνει ότι η βέλτιστη τιμή είναι αυτή που βρήκε η Χριστοδούλου (2010) και είναι ίση με 17 για το πρώτο κρυφό επίπεδο και 0 για το δεύτερο κρυφό επίπεδο και για τα δύο δίκτυα. Όμως για να εξακριβώσουμε την υπόθεση μας ότι πράγματι αυτή είναι η καλύτερη τιμή, ξαναδοκιμάσαμε κάποια πειράματα τροποποιώντας τις 2 παραμέτρους αυτές και τα αποτελέσματα φαίνεται πιο κάτω.

Στο Σχήμα 5.2 πιο κάτω φαίνονται τα αποτελέσματα από τα πειράματα για τιμές που αφορούν τα κρυφά επίπεδα του Backward (Br) δικτύου.



Σχήμα 5.2: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με το μέγεθος του πρώτου κρυφού επιπέδου στο δίκτυο Br του BRNN.Το σύνολο εκπαίδευσης που χρησιμοποιήθηκε ήταν το DataSet1 του CB513.*

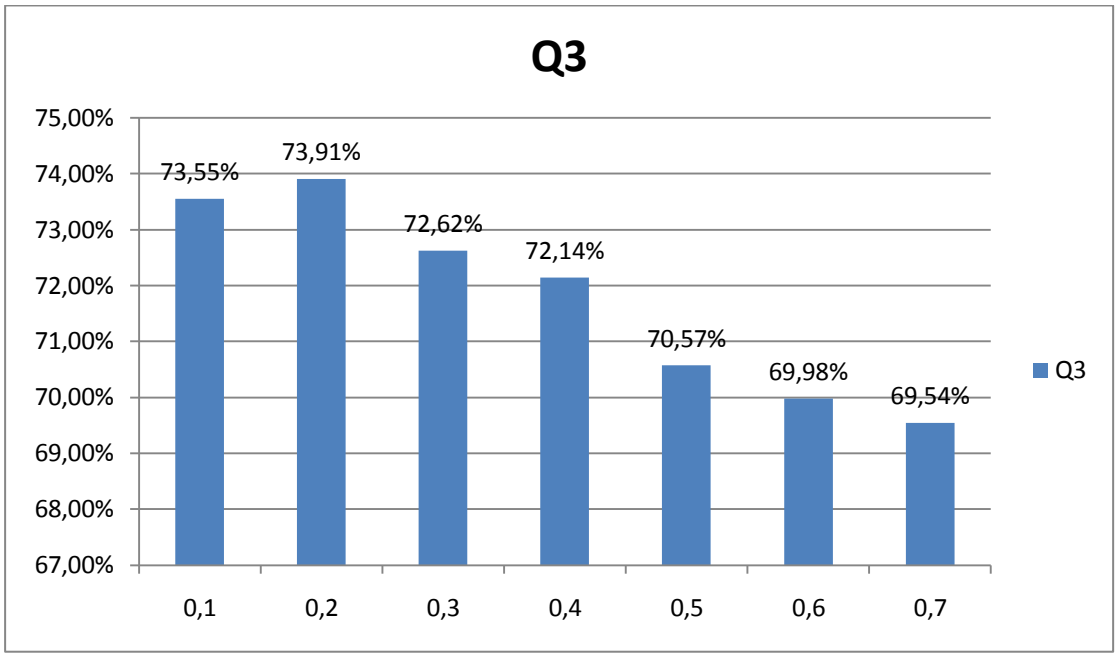
Στη πιο πάνω γραφική παράσταση κρατήσαμε σταθερό το μέγεθος το δεύτερου κρυφού επιπέδου και ίσο 0. Επίσης χρησιμοποιήσαμε τις βέλτιστες παραμέτρους που μόλις βρήκαμε για το κεντρικό παράθυρο. Πράγματι από τη γραφική βλέπουμε ότι η τιμή 17 δίνει ποσοστό επιτυχίας 73,84% που είναι και το πιο ψηλό. Επίσης από τη γραφική παράσταση στο Σχήμα 5.3 φαίνεται ότι όταν δεν χρησιμοποιήσουμε δεύτερο κρυφό επίπεδο παίρνουμε καλύτερα αποτελέσματα.



Σχήμα 5.3: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με το μέγεθος του δεύτερου κρυφού επιπέδου στο δίκτυο Br του BRNN. Το σύνολο εκπαίδευσης που χρησιμοποιήθηκε ήταν το 2ο fold (DataSet1) του CB513.*

Τα αποτελέσματα των πειραμάτων για το εμπρόσθιο δίκτυο (Fr) δεν θα αναλυθούν γιατί παρουσίασαν παρόμοια συμπεριφορά με το Backward Δίκτυο. Αναφέρουμε όμως ότι και εδώ ο αριθμός των νευρώνων στο πρώτο κρυφό επίπεδο είναι 17 και στο δεύτερο 0.

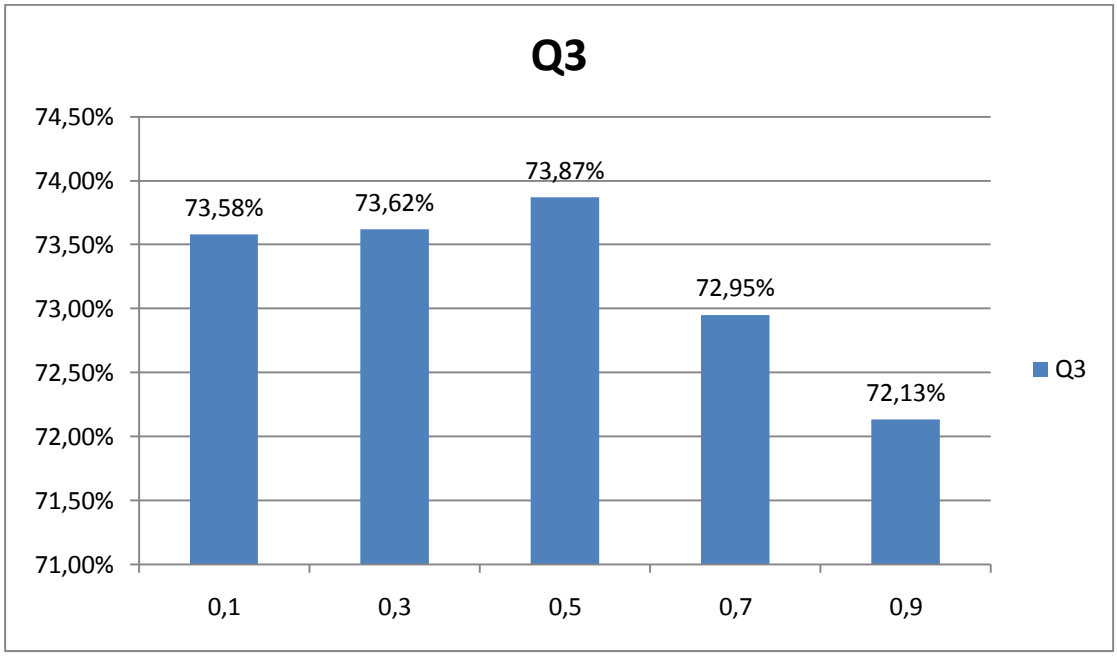
Το επόμενο πείραμα, αφορά την εναλλαγή διάφορων τιμών για την παράμετρο 7 του πίνακα 5.1, που αφορά τον ρυθμό μάθησης του συστήματος. Η Χριστοδούλου στην υλοποίηση της είχε υπολογίσει ότι το καλύτερη τιμή για ρυθμό μάθησης έπρεπε να είναι ίση 0,09. Όμως μετά από δοκιμές που κάναμε στη δική μας υλοποίηση παρατηρήσαμε ότι διαφορετικές τιμές επέφεραν καλυτέρα αποτελέσματα.



Σχήμα 5.4: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με το ρυθμός μάθησης που χρησιμοποιήσαμε. Το σύνολο εκπαίδευσης που χρησιμοποιήθηκε ήταν το 2ο fold (DataSet1) του CB513.*

Απ' ότι βλέπουμε από την γραφική παράσταση του σχήματος 5.4, η απόδοση του συστήματος εφαρμόζοντας στον ρυθμό μάθησης τιμές μεγαλύτερες από 0.4 είναι χαμηλότερη σε σύγκριση με τα αποτελέσματα της απόδοσης του δικτύου σε τιμές από 0.1 μέχρι 0.2, κάτι που είχε παρατηρήσει και η Χριστοδούλου (2010). Το μεγαλύτερο όμως Q3 ποσοστό απόδοσης ήταν 73,91% και επιτεύχθηκε όταν ο ρυθμός μάθησης ήταν 0.2. Αυτή η αλλαγή στην τιμή του ρυθμού μάθησης σε σχέση με αυτή της Χριστοδούλου ίσως να οφείλεται κυρίως στο γεγονός ότι χρησιμοποιήσαμε διαφορετικό σύνολο δεδομένων.

Η επόμενη παράμετρος που ίσως να χρίζει διερεύνησης, είναι η ορμή (παράμετρος 8 του πίνακα 5.1). Έτσι λοιπόν, ορίσαμε διάφορες τιμές για τον όρο της ορμής και κάναμε τα πειράματα βάση των τιμών αυτών. Όπως φαίνεται και από τα αποτελέσματα στο σχήμα 5.5 παρατηρούμε ότι η τιμή της ορμής δεν χρειάζεται αλλαγή αφού με τιμή 0,5 δίνει τα καλύτερα αποτελέσματα.



Σχήμα 5.4: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με το ρυθμός μάθησης που χρησιμοποιήσαμε. Το σύνολο εκπαίδευσης που χρησιμοποιήθηκε ήταν το 2ο fold (DataSet1) του CB513.*

Κλείνοντας τον κύκλο των πειραμάτων που αφορούν τις παραμέτρους , μπορούμε να δούμε ότι όντως η μεθοδολογία που ακολουθήθηκε και τα πειράματα που έγιναν για το σύνολο δεδομένων για ένα fold του CB513, έδωσαν κάποιες τοπικά «βέλτιστες» τιμές για τις παραμέτρους του πίνακα 5.1, και ένα ποσοστό πρόβλεψης 73,91%. Οι τιμές αυτές φαίνονται συνοπτικά στον πίνακα 5.2. Επίσης μπορούμε να δούμε ότι η ενσωμάτωση του όρου του κεντρικού παραθύρου αύξησε εν μέρει το ποσοστό επιτυχίας του συστήματος αλλά όχι στο βαθμό που υπολογίζαμε. Συνοπτικά το κεντρικό παράθυρο στη βέλτιστη τιμή του που είναι ίση με 5, πρόσθεσε αύξηση της τάξεως του 0,8%

No	ΠΑΡΑΜΕΤΡΟΣ	ΤΙΜΗ
1	Hidden_layer_one_size	17
2	Hidden_layer_two_size	8
3	Hidden_layer_one_of_Backward_size	17
4	Hidden_layer_two_of_Backward_size	0
5	Hidden_layer_one_of_Forward_size	17
6	Hidden_layer_two_of_Forward_size	0
7	Learning_Rate	0.2
8	Momentum	0.1
9	Window_size	31
10	q_minus_1	0.5
11	q_plus_1	0.5
12	S	3
13	Maximum_Iterations	200
14	Center_window_size	5 (2+2+1)

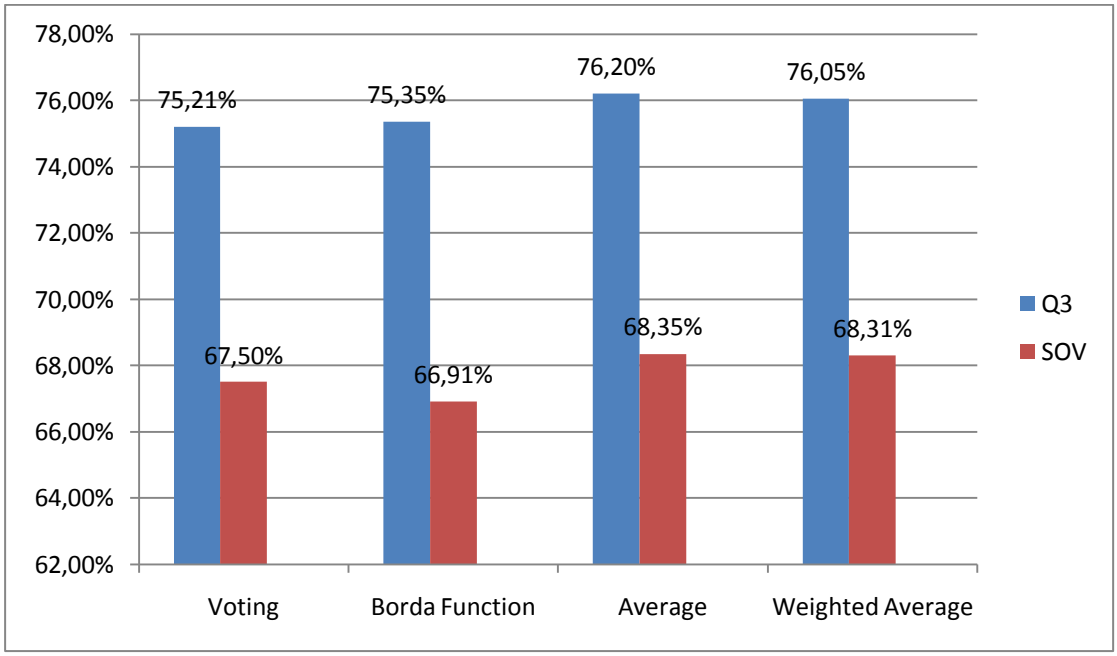
Πίνακας 5.3: *Τελικές τιμές για τις παραμέτρους του δικτύου όπως τις ορίσαμε και οι οποίες είναι οι βέλτιστες.*

5.2 Πειράματα με την χρήση Ensemble

Αφού βρήκαμε τις βέλτιστες τιμές για τις παραμέτρους (Πίνακας 5.3) για ένα υποσύνολο, κανονικά στη συνέχεια πρέπει να τρέξουμε και τα 10 folds που αποτελούν το CB513 για να βρούμε το τελικό αποτέλεσμα με τη χρήση του Μέσου Όρου. Όμως αντί να γίνει αυτό, λόγω του μεγάλου χρόνου εκτέλεσης του προγράμματος επιλέξαμε να τρέξουμε και τα 10 folds αφού πρώτα ενσωματώσουμε και το μέθοδο του Ensemble. Επειδή τα πειράματα θα επαναληφθούν και για την εύρεση της καλύτερης μεθόδου Ensemble, προτιμήσαμε να τρέξουμε ένα υποσύνολο και σε αυτή την τροποποίηση του συστήματος. Αφού βρεθεί η καλύτερη μέθοδος τότε θα γίνουν πειράματα σε όλο το CB513.

Όπως αναφέραμε στο Υποκεφάλαιο 4.2 η μέθοδος του Ensemble υλοποιήθηκε με την υπόθεση ότι πολλά δίκτυα μαζί μπορούν να εξάγουν καλύτερα αποτελέσματα

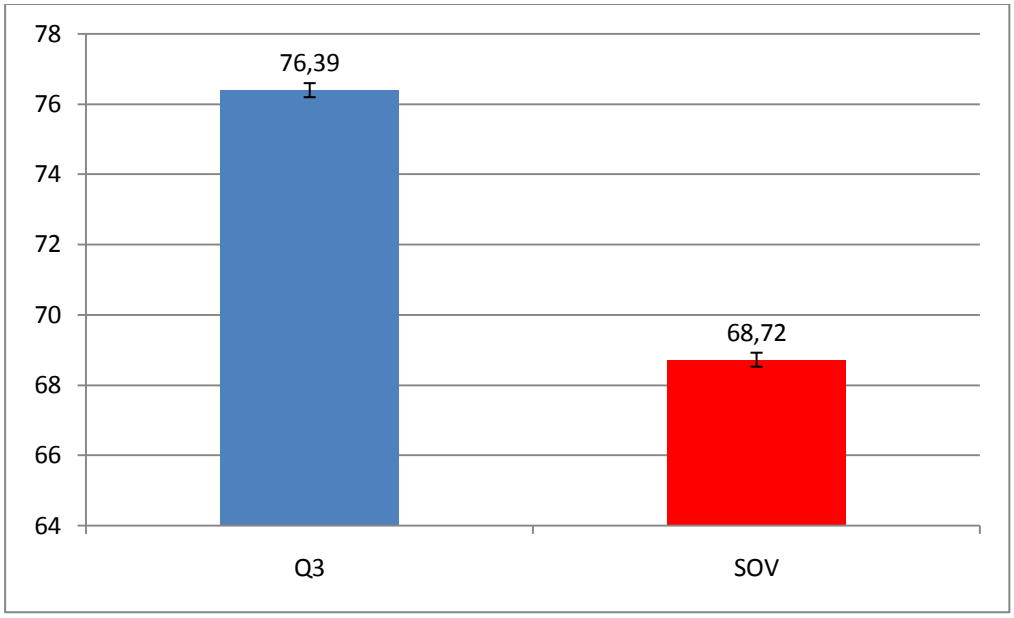
από ένα. Στα πειράματα που κάναμε το σύστημα αποτελείται από 6 δίκτυα BRNN και δοκιμάσαμε 4 διαφορετικές μεθόδους, Το Voting, το Borda Function, το Average και το Weighted Average που επεξηγήθηκαν στο Υποκεφάλαιο 4.2. Τα αποτελέσματα των πειραμάτων δείχνουν το ποσοστό επιτυχίας σε Q3 και SOV και φαίνονται στο πιο κάτω Σχήμα 5.5.



Σχήμα 5.5: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV για τις 4 μεθόδους που χρησιμοποιήθηκαν για την υλοποίηση του Ensemble. Το σύνολο εκπαίδευσης που χρησιμοποιήθηκε ήταν το 2ο fold (DataSet1) του CB513.*

Όπως φαίνεται από τα πειράματα η μέθοδος του Ensemble επιφέρει πράγματι καλύτερα αποτελέσματα. Και με τις 4 μεθόδους καταφέραμε να αυξήσουμε το ποσοστό σωστής πρόβλεψης από 73,91% μέχρι και 76,20% σε Q3. Τα καλύτερα αποτελέσματα τόσο σε Q3 όσο και σε SOV πάρθηκαν από το μέθοδο Average με μικρή διαφορά από τη μέθοδο Weighted Average. Επιλέχθηκε επομένως η Average μέθοδο η οποία είναι και η πιο απλή και έχει τη λιγότερη πολυπλοκότητα ως αύτη με την οποία θα γίνουν τα πειράματα πάνω σε ολόκληρο το CB513. Συνδυάζοντας τις βέλτιστες τιμές των παραμέτρων που πήραμε προηγουμένως μαζί με το *Ensemble*, θα τρέξουμε και τα 10 folds του CB513. Το τελικό ποσοστό επιτυχίας του συστήματος θα παρθεί αφού ενώσουμε τα 10 αρχεία εξόδου που θα δημιουργηθούν και υπολογίζοντας το μέσο όρο τους. Για τη επαλήθευση των

αποτελεσμάτων η όλη διαδικασία επαναλήφθηκε 3 φορές. Τα αποτελέσματα φαίνονται στο σχήμα 5.6.



Σχήμα 5.6: *Γραφική παράσταση που παρουσιάζει το μέσο όρο του ποσοστά επιτυχίας σε Q3 και SOV για τρείς εκτελέσεις με την χρήση της μεθόδου Average στη δημιουργία των Ensemble. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

Όπως φαίνεται από τη γραφική παράσταση στο Σχήμα 5.6 το μεγαλύτερο ποσοστό επιτυχίας έφτασε το 76,40% σε Q3 και 68,74 σε SOV. Συγκριτικά με το τελικό ποσοστό που είχε επιτύχει η Χριστοδούλου (2010), παρατηρούμε ότι η τρέχουσα υλοποίηση του συστήματος πέτυχε σχεδόν 0,5% αύξηση στο Q3 ενώ το SOV, από 70% που ήταν πριν τώρα μειώθηκε κατά 1,3%. Στη συνέχεια θα δούμε πως μπορούμε να αυξήσουμε περισσότερο το ποσοστό με διάφορες άλλες μεθόδους.

5.3 Φιλτράρισμα προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών

Μέχρι τώρα είδαμε ότι το τελικό ποσοστό του συστήματος έφτασε μέχρι το 76,40% σε Q3 και 68,74% σε SOV. Για την επίτευξη καλύτερων αποτελεσμάτων όπως εξηγήθηκε στο Υποκεφάλαιο 4.3 ενσωματώσαμε post-processing filtering στο σύστημα μας. Η τεχνική αυτή χρησιμοποιώντας την έξοδο του BRNN (τρείς καταστάσεις H,E,L), προσπαθεί να *διορθώσει* την προβλεπόμενη ακολουθία που

δίνει το τρέχον σύστημα στο τέλος κάθε εκτέλεσης του, λαμβάνοντας υπόψη τις συσχετίσεις μεταξύ της προβλεπόμενης δευτεροταγούς δομής κάθε αμινοξέως της πρωτεΐνης. Η μέθοδος που ακολουθήθηκε για το *φιλτράρισμα* ήταν με τη χρήση δικτύου RBF (Υποκεφάλαιο 4.3.2). Για τη σωστή επιλογή των κέντρων και αρχικοποίηση του RBF χρησιμοποιήσαμε πρώτα ένα χάρτη αυτοοργάνωσης Kohonen για τη κατηγοριοποίηση των δεδομένων μας. Επομένως πρώτα θα αναφερθούμε στα αποτελέσματα του SOM.

5.3.2 Αποτελέσματα φιλτραρίσματος προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών με χρήση SOM

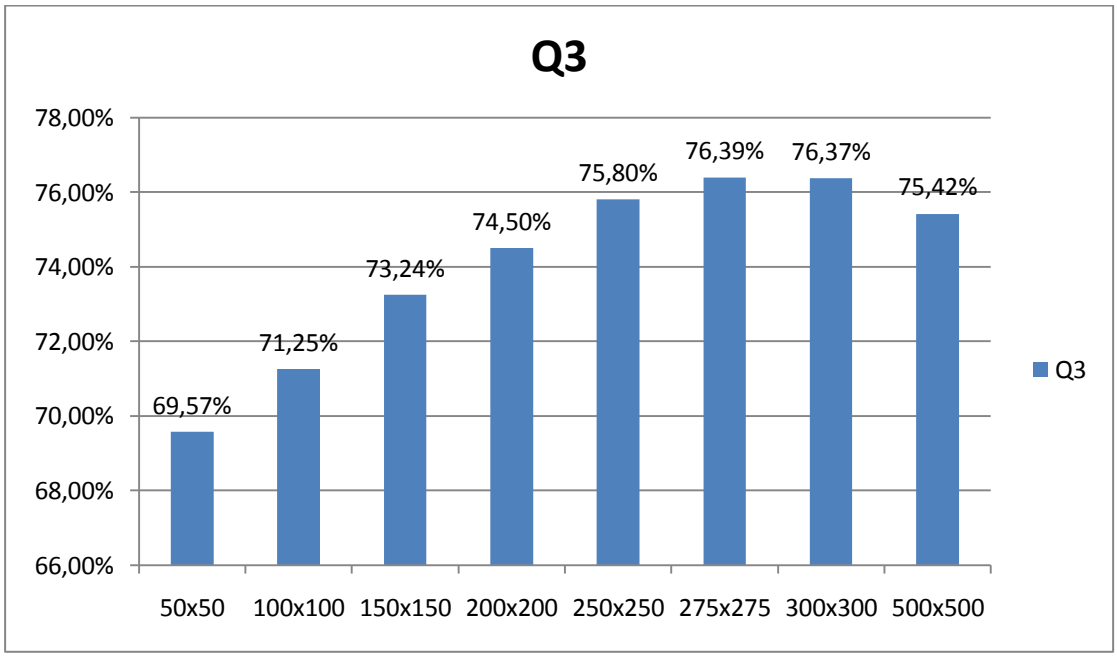
Για την υλοποίηση του SOM θα χρησιμοποιήσουμε ως δεδομένα εισόδου το *Z_Filter_Output.txt* το οποίο, όπως αναφέραμε στο κεφάλαιο 3, έχει τροποποιηθεί ούτως ώστε για κάθε αμινοξύ της κάθε πρωτεΐνης που περιλαμβάνει, έχουμε και τη δεκαδική τιμή της κάθε εξόδου (H, E ή L) της δευτεροταγούς δομής. Το μέγεθος του χάρτη που δημιουργήσαμε όπως και οι υπόλοιπες παράμετροι φαίνονται στο Πίνακα 5.4

No	ΠΑΡΑΜΕΤΡΟΣ	ΤΙΜΗ
1	mapSideSize	275
2	learningRate neighborhood	0.2
3	Radius	50
4	maxIterations	10
5	trainFile Z_Filter.txt	-
6	testFile Z_Filter_Output.txt	-
13	Window_size	7

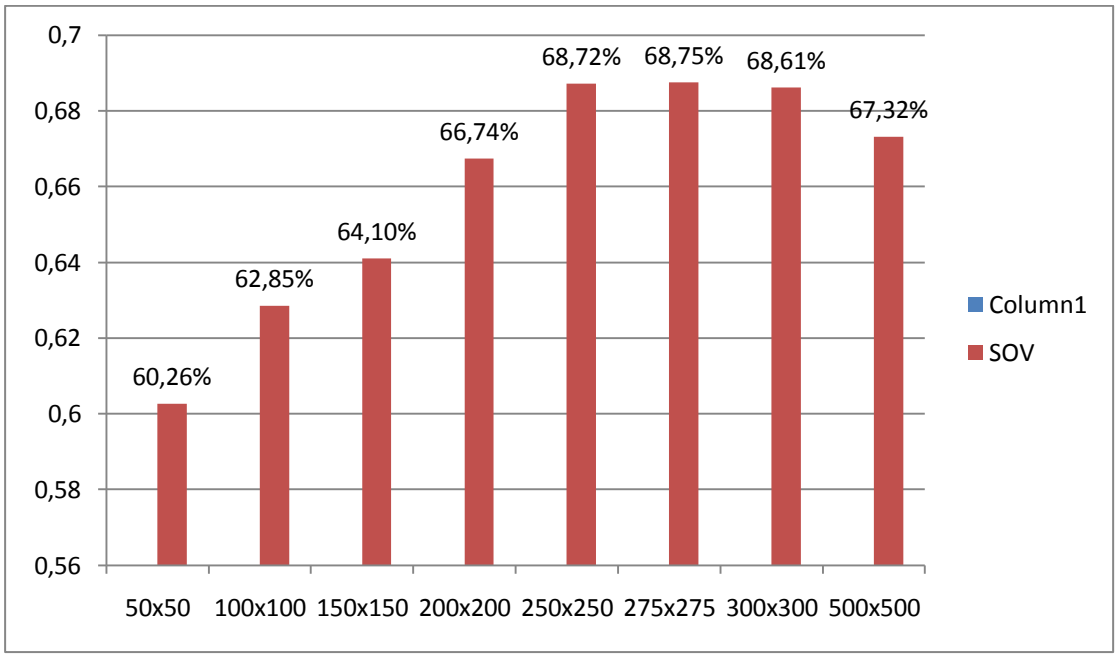
Πίνακας 5.4: *Τελικές τιμές για τις παραμέτρους του χάρτη αυτοοργάνωσης Kohonen (SOM).*

Για την επιλογή των σωστών παραμέτρων έγιναν όπως και πριν πειράματα για την εύρεση των βέλτιστων τιμών. Για κάθε πείραμα που εκτελέσαμε εκπαιδεύαμε το δίκτυο στα 9 ανεξάρτητα υποσύνολα του CB513 και δοκιμάζαμε σε 1. Αυτή η διαδικασία επαναλήφθηκε 10 φορές για να δοκιμάσουμε και τα 10 υποσύνολα. Το τελικό αποτέλεσμα είναι ίσο με το μέσο όρο των 10 συνόλων. Υπό κανονικές συνθήκες το SOM χρησιμοποιήθηκε για την εύρεση των κέντρων της τοπολογίας του RBF. Όμως αφού η όλη διαδικασία θα γίνει για την οργάνωση των δεομένων μας σε ομάδες, σκεφτήκαμε να χρησιμοποιήσουμε το SOM και ως ανεξάρτητο είδος filtering, για να δούμε πως αντιδρά στο συγκεκριμένο πρόβλημα. Ουσιαστικά αυτό αποτελεί και το τρόπο εύρεσης και των βέλτιστων τιμών των παραμέτρων. Για να θεωρηθεί καλή η τιμή μιας παραμέτρου πρέπει βασικά το αποτέλεσμα ενός πειράματος να έχει περίπου τα ίδια αποτελέσματα με πριν. Δε πρέπει να έχουμε μείωση στο Q3, ούτε στο SOV. Αν μια τοπολογία επιφέρει χαμηλότερα ποσοστά από πριν αυτό σημαίνει ότι το πιο πιθανό αν τη χρησιμοποιήσουμε στην επομένη φάση, θα έχουμε χαμηλά ποσοστά.

Το πρώτο πείραμα που έγινε αφορά το μέγεθος του χάρτη. Δοκιμάστηκαν διάφορα μεγέθη από 50x50 μέχρι 300x300. Τα πιο σημαντικά αποτελέσματα φαίνονται πιο κάτω (Σχήμα 5.7 και 5.8).



Σχήμα 5.7: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 για διάφορα μεγέθη του χάρτη που θα οργανώσουμε. Οι υπόλοιπες παράμετροι διατηρούνται σταθερές και ίσες με αυτέ που φαίνονται στο Πίνακα 5.4*

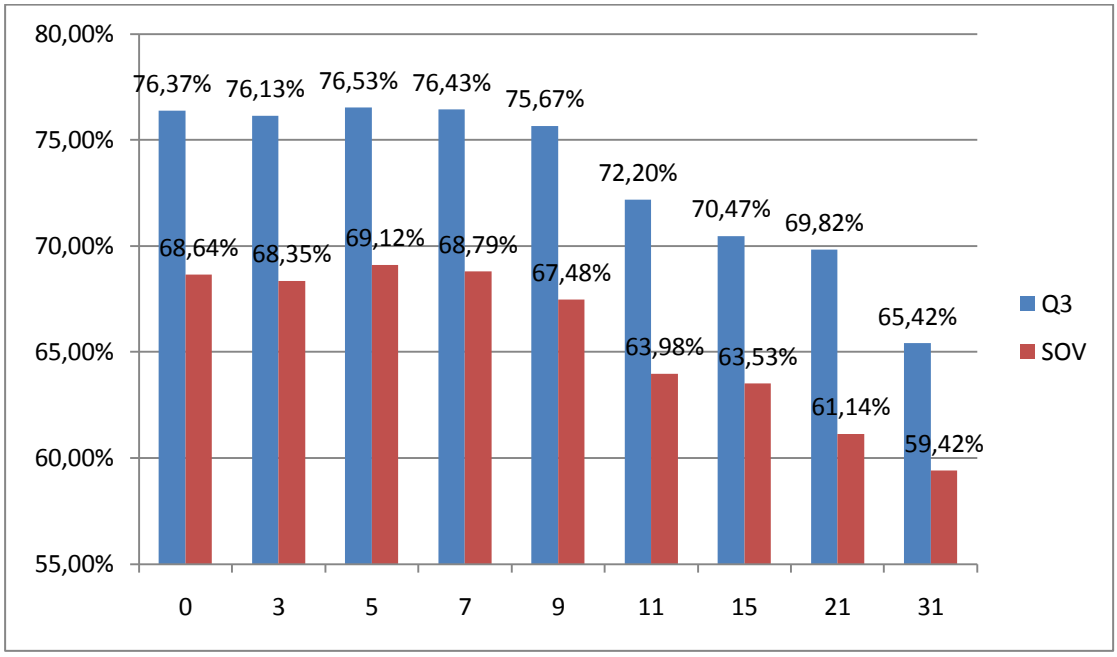


Σχήμα 5.8: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε SOV Score για διάφορα μεγέθη του χάρτη που θα οργανώσουμε. Οι υπόλοιπες παράμετροι διατηρούνται σταθερές και ίσες με αυτέ που φαίνονται στο Πίνακα 5.4*

Από τη γραφική παράσταση μπορούμε να δούμε ότι τα καλύτερα αποτελέσματα επιτυγχάνονται από χάρτη 275x275, δηλαδή 75625. Αυτό συμβαίνει γιατί ο αριθμός των αμινοξέων (residues) που έχουμε να εκπαιδεύσουμε συνολικά φτάνει περίπου τις 73000. Επομένως αυτό το μέγεθος χάρτη στην ουσία μπορεί να χαρτογραφήσει σχεδόν κάθε αμινοξύ σε κάθε θέση. Μικρότερου μεγέθους χάρτη παρατηρούμε ότι έχουν μικρότερο ποσοστό επιτυχίας ενώ μεγαλύτεροι δεν επιφέρουν κάποια καλυτέρα αποτελέσματα, ενώ επιπρόσθετα αυξάνουν τη πολυπλοκότητα του συστήματος.

Μια ακόμα σημαντική παράμετρος που πρέπει να εξετασθεί προσεκτικά είναι το μέγεθος του κινητού παράθυρου που θα χρησιμοποιήσουμε. Όπως προαναφέρθηκε το μέγεθος του κινούμενου παραθύρου που θα χρησιμοποιήσουμε παίζει πολύ

σημαντικό ρόλο, αφού η αναδίπλωση των πρωτεϊνών επηρεάζεται άμεσα από τη πληροφορία που εμπεριέχεται στα κατάλοιπα της. Τα αποτελέσματα με τις δοκιμές του μεγέθους του παραθύρου φαίνεται στο πιο κάτω Σχήμα 5.9.

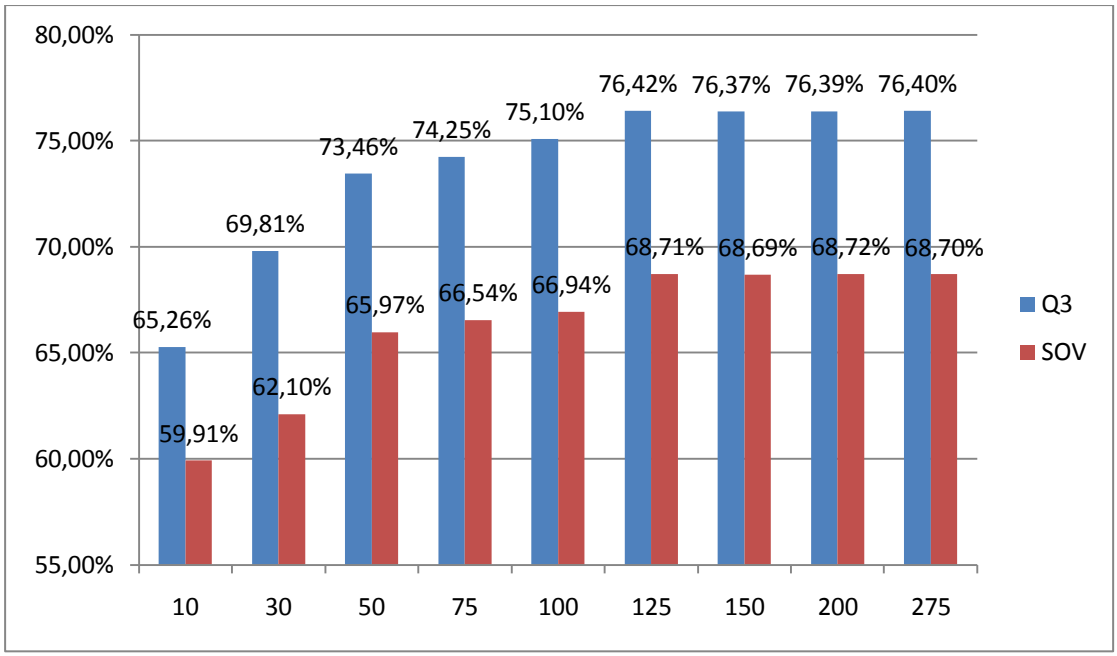


Σχήμα 5.9: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV για διάφορα μεγέθη του κινητού παραθύρου που χρησιμοποιήσαμε. Οι υπόλοιπες παράμετροι διατηρούνται σταθερές και ίσες με αυτέ που φαίνονται στο Πίνακα 5.4*

Από τη γραφική παράσταση στο Σχήμα 5.9 μπορούμε να δούμε ότι το παράθυρο πράγματι επηρεάζει το τελικό αποτέλεσμα κάποιες φορές θετικά και κάποιες φορές αρνητικά. Είναι φανερό ότι όταν έχουμε μικρό κινητό παράθυρο ή καθόλου το ποσοστό επιτυχίας είναι πιο ψηλό, ενώ όταν το παράθυρο είναι μεγάλο προκαλεί χειρότερα αποτελέσματα. Αυτό οφείλεται στο γεγονός ότι όταν έχουμε μεγάλο παράθυρο σημαίνει ότι προσπαθεί να χαρτογραφήσει μεγάλη πληροφορία σε κάποιο τμήμα του χάρτη και αυτό έχει σαν αποτέλεσμα να μην μπορεί να γίνει σωστή χαρτογράφηση.

Στην συνέχεια δοκιμάσαμε να αλλάξουμε τα ρυθμό μάθησης και την ακτίνα της γειτονιάς. Από τα αποτελέσματα των πειραμάτων που φαίνονται στον Σχήμα 5.10 παρατηρούμε ότι τα καλύτερα ποσοστά επιτυγχάνονται με ρυθμό μάθηση 0,2 και αρχική ακτίνα 125. Η τιμή της ακτίνας μπορεί να εξηγηθεί από το γεγονός ότι καθώς

περνούν οι εποχές η ακτίνα μικραίνει. Επομένως μεγάλη ακτίνα στην αρχή έχει καλύτερα αποτελέσματα γιατί στα αρχικά στάδια της οργάνωσης του χάρτη θέλουμε να τροποποιούνται μεγάλα τμήματα, μέχρις ότου ο χάρτης πάρει κάποια μορφή.

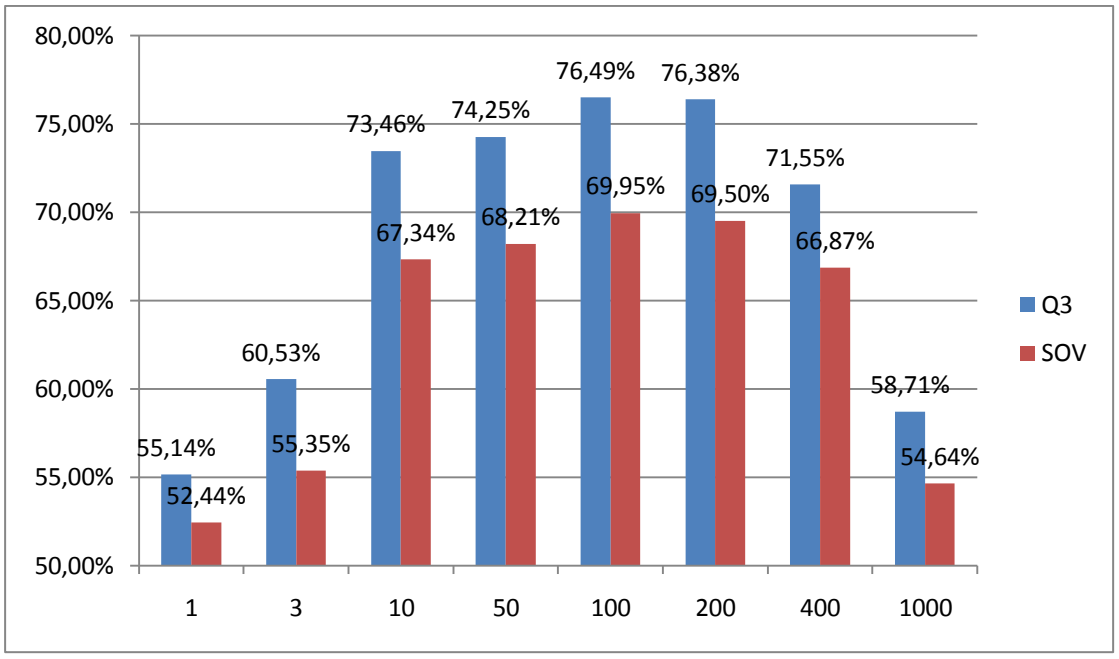


Σχήμα 5.10: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV για διάφορα μεγέθη ακτίνας που χρησιμοποιήσαμε. Οι υπόλοιπες παράμετροι διατηρούνται σταθερές και ίσες με αυτέ που φαίνονται στο Πίνακα 5.4.*

5.3.2 Αποτελέσματα φιλτραρίσματος προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών με χρήση Radial Basis Function

Αφού έχουμε δημιουργήσει το χάρτη μας είμαστε σε θέση να αρχικοποιήσουμε το δίκτυο RBF μας. Όπως αναφέρθηκε στο Υποκεφάλαιο 4.3.2 πολύ σημαντική παράμετρος είναι η σωστή επιλογή κέντρων. Αν δεν γίνει σωστή επιλογή κέντρων τότε δεν θα έχουμε τα επιθυμητά αποτελέσματα. Η άλλη παράμετρος που πρέπει να ελεχθεί προσεχτικά είναι το μέγεθος του κινητού παραθύρου. Θα ξεκινήσουμε τη περιγραφή των πειραμάτων μας με τη πρώτη παράμετρο και αρχικά δε θα χρησιμοποιήσουμε κάποιο κινητό παράθυρό.

Η επιλογή του αριθμού των κέντρων θα ξεκινήσει από 1 και θα αυξάνεται. Όπως γνωρίζουμε στα RBF δίκτυα όσο αυξάνεται ο αριθμός των κέντρων, οδηγούμαστε σε παρεμβολή δηλαδή το δίκτυο προσπαθεί να μάθει "απέξω" κάποιες συγκεκριμένες γνωστές καταστάσεις. Όσον αφορά την επιλογή της θέσης των κέντρων αυτή έγινε εμπειρικά. Δηλαδή βλέποντας πώς έχει χαρτογραφηθεί ο χάρτης στο SOM, επιλέγουμε περιοχές οι οποίες είναι «καθαρές» δηλαδή δεν περιέχουν τμήματα από τις άλλες δύο καταστάσεις. Οι τιμές των βαρών των νευρώνων στις θέσεις αυτές θα αποτελέσουν και τις τιμές των κέντρων που θα χρησιμοποιήσουμε. Τα αποτελέσματα για διάφορα πειράματα με διαφορετικό αριθμό κέντρων φαίνονται στο Σχήμα 5.11.

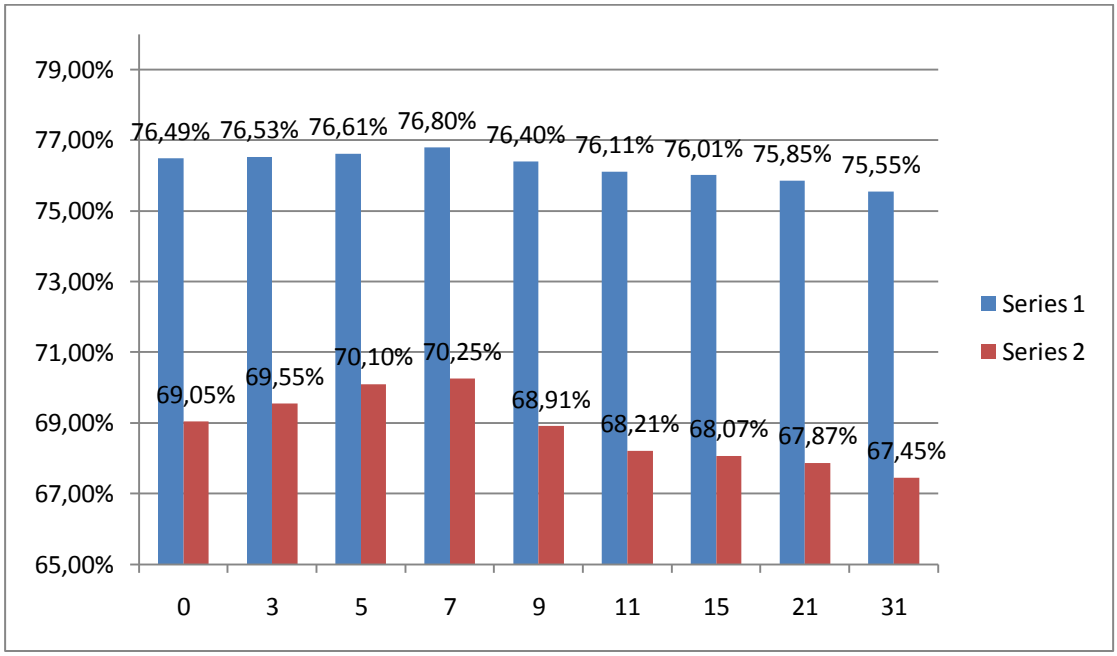


Σχήμα 5.11: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV για διάφορους αριθμούς κέντρων που χρησιμοποιήθηκαν στην εκπαίδευση του RBF.*

Από τα αποτελέσματα της γραφικής παράσταση στο Σχήμα 5.11 μπορούμε να δούμε ότι το καλύτερο ποσοστό επιτυγχάνεται όταν χρησιμοποιούμε περίπου 100 κέντρα. Με το αριθμό αυτό καταφέραμε να αυξήσουμε το SOV score κατά 1% ενώ το Q3 έμεινε σχετικά σταθερό. Επίσης από της γραφική παράσταση είναι φανερό ότι όταν χρησιμοποιήσουμε μεγάλο αριθμό κέντρων οδηγούμαστε σε πιο χαμηλά ποσοστά από ότι χωρίς τη μέθοδο του filtering γιατί ουσιαστικά το δίκτυο μαθαίνει

"απέξω" κάποιες καταστάσεις και όταν του δοθεί το υποσύνολο με τα δεδομένα δοκιμής που περιέχει μέσα πρωτεΐνες που δεν έχει ξαναδεί δεν καταφέρνει να προβλέψει σωστά τη δευτεροταγή δομή τους.

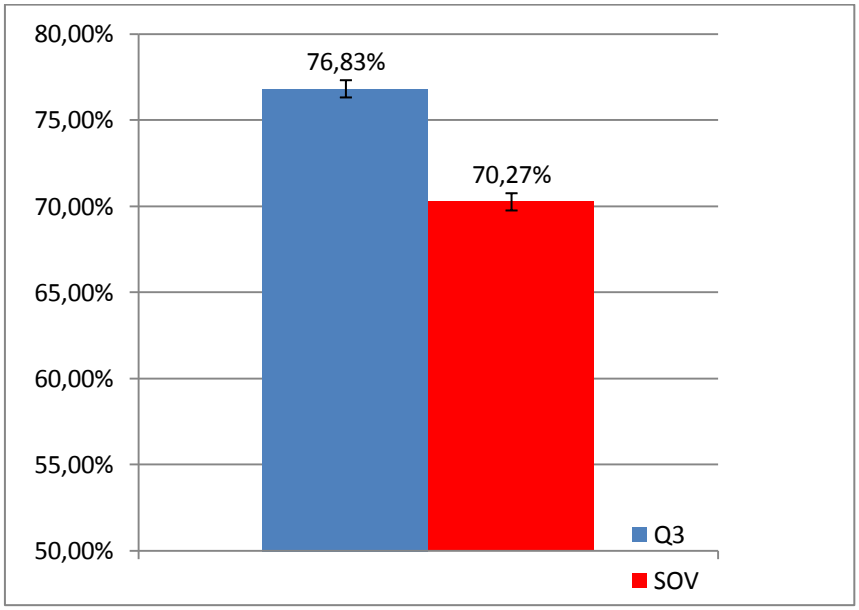
Η επόμενη παράμετρος του RBF που θα ελέγξουμε είναι το μέγεθος του κινητού παράθυρου, μια παράμετρος που ελέχθηκε και κατά τη διάρκεια της εκπαίδευση του SOM. Τα αποτελέσματα για διάφορα μεγέθη παραθύρου φαίνονται στη πιο κάτω γραφική του Σχήματος 5.12.



Σχήμα 5.11: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV για διάφορα μεγέθη κινητού παράθυρου που χρησιμοποιήθηκαν στην εκπαίδευση του RBF.*

Όπως μπορούμε να δούμε η εισαγωγή του κινητού παραθύρου επέφερε όπως και πριν καλύτερα αποτελέσματα. Όταν το κινητού παράθυρο είναι ίσο με 7 (3+1+3) δηλαδή αποθηκεύουμε πληροφορία πέραν από το κεντρικό αμινοξύ για ακόμα 2 κατάλοιπα δεξιά και 2 αριστερά το ποσοστό επιτυχίας ανέρχεται σε 76,80% και 70,25 για SOV. Επομένως, παρατηρούμε μια σημαντική αύξησης κυρίως στο SOV της τάξης του 2,5%. Από τη γραφική παράσταση μπορούμε να δούμε επίσης ότι όταν το παράθυρο αυξηθεί κι άλλο τότε το ποσοστό επιτυχίας μειώνεται και έχουμε χαμηλότερα ποσοστό από ότι εάν δεν χρησιμοποιούσαμε filtering.

Τα καλά αποτελέσματα που επιτεύχθηκαν προηγουμένως μας οδήγησαν να κάνουμε περισσότερες δοκιμές και συγκρίσεις, καθώς επίσης και πιο αναλυτική περιγραφή της πρόβλεψης του συστήματος.



Σχήμα 5.12: *Γραφική παράσταση που παρουσιάζει το μέσο όρο του ποσοστά επιτυχίας σε Q3 και SOV για τέσσερις εκτελέσεις της μεθόδου filtering με τη χρήση RBF δικτύου. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

Η γραφική παράσταση στο σχήμα 5.12 αντιπροσωπεύει και τη τελικό ποσοστό επιτυχίας που επιτεύχθηκε με τη μέθοδο του filtering. Από τα συγκεντρωτικά αποτελέσματα μπορούμε να δούμε ότι πράγματι με το *φιλτράρισμα* των αποτελεσμάτων του δικτύου BRNN παίρνουμε αισθητά καλύτερα αποτελέσματα κυρίως για SOV Score. Η χρήση το *filtering* με τη μέθοδο αυτή αύξησε το Q3 κατά 0,4% ενώ το SOV κατά 2,5, μια σχετικά πολύ καλή βελτίωση.

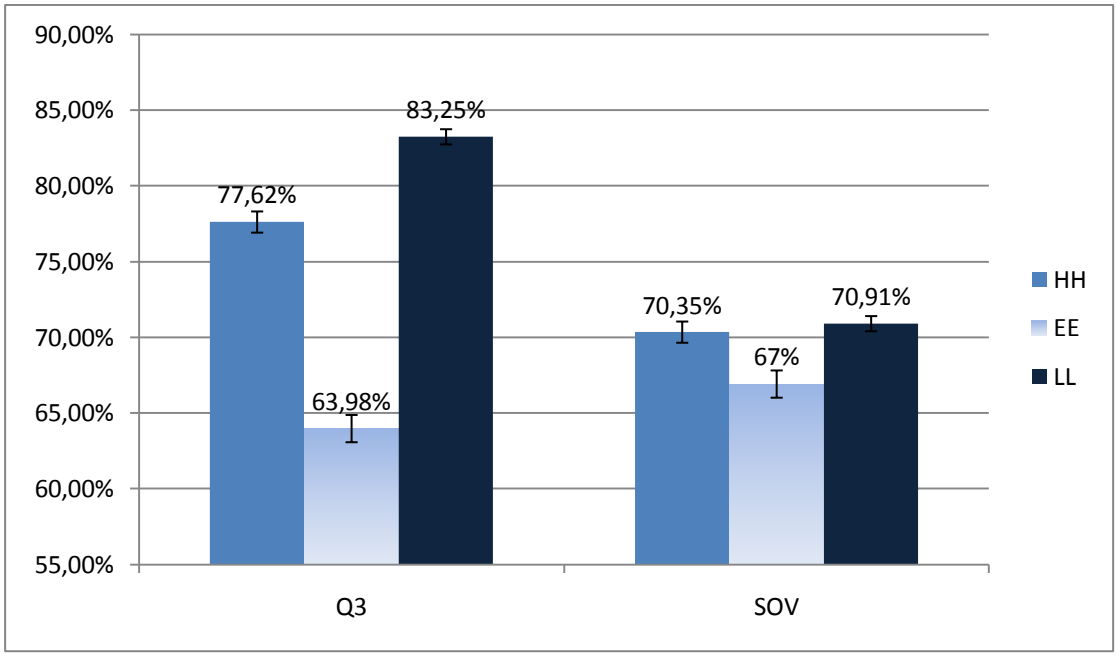
Ο μεταδιδακτορικός ερευνητής Δρ. Πέτρος Κουντούρης για το ίδιο πρόβλημα δοκίμασε διαφορές τεχνικές φιλτραρίσματός με τη χρήση του προγράμματος WEKA. Τα αποτελέσματα του, όπως και τα αποτελέσματα που επιτύχαμε φαίνονται αναλυτικά στο πιο κάτω Πίνακα 5.5.

Filtering method	w	Q_3 (%)	Q_H (%)	Q_E (%)	Q_L (%)	SOV	SOV_H	SOV_E	SOV_L	C_H	C_E	C_L	SEL
LibSVM	19	**77.04**	78.02	65.81	82.40	72.54	71.92	68.64	70.80	0.718	0.635	0.583	74.79
Logistic	19	76.93	78.69	67.33	80.76	**72.83**	72.43	68.74	71.31	0.716	0.633	0.582	**74.88**
RBF (SOM)	7	76.80	77.62	63.98	83.18	70.25	70.35	66.91	70.91	0.713	0.625	0.586	73.53
MLP	5	76.75	77.94	66.02	81.68	71.75	70.72	68.77	70.17	0.717	0.626	0.579	74.25
Simple Cart	5	76.65	79.06	66.78	80.11	70.60	70.91	67.57	69.66	0.712	0.625	0.580	73.63
SS-filt	–	76.43	75.98	62.23	84.58	71.25	70.53	66.92	70.56	0.711	0.622	0.578	73.84
No filtering	–	76.39	77.12	63.53	82.87	68.74	68.75	66.07	69.63	0.706	0.618	0.580	72.57
WH-filt	–	76.24	74.77	62.33	**85.06**	69.43	67.31	65.90	70.35	0.710	0.616	0.577	72.84
RBF (k-Means)	1	76.23	**81.52**	69.88	75.44	69.30	71.73	68.84	66.86	0.705	0.618	0.578	72.77
Naive Bayes	3	76.10	78.68	**71.99**	76.27	71.75	71.37	70.46	68.57	0.710	0.629	0.561	73.92
Viterbi	1	75.98	77.77	63.93	81.14	69.59	69.58	65.57	67.53	0.705	0.619	0.562	72.79
J48	3	75.98	78.97	65.87	79.10	68.53	69.33	66.84	67.84	0.704	0.610	0.569	72.25
Random Forest	19	75.19	79.64	68.58	75.23	66.76	68.58	66.68	64.94	0.696	0.608	0.550	70.98
IBk (k=3)	13	72.03	78.67	62.64	71.81	62.66	67.98	62.71	60.86	0.640	0.566	0.499	67.34

Πίνακας 5.5: *Πίνακας που δείχνει συγκεντρωτικά και αναλυτικά το ποσοστά πρόβλεψης του συστήματος τόσο σε Q3 όσο και σε SOV, σε σχέση με άλλες τεχνικές που χρησιμοποιήθηκαν από το post-doct Πέτρο Κουντούρη για filtering. Στη μέση του Πίνακα έχουμε τα ποσοστά χωρίς την χρήση οποιουδήποτε filtering, ενώ από τη μέση και πάνω έχουμε τα ποσοστά των τεχνικών που κατάφεραν να αυξήσουν το ποσοστό επιτυχίας.*

Συγκρίνοντας τα αποτελέσματα μας με αυτά του Δρ. Πέτρου Κουντούρη μπορούμε να δούμε αναλυτικά ότι τεχνική που αναπτύξαμε επιφέρει τα τρίτα καλύτερα αποτελέσματα από όλες τις τεχνικές που χρησιμοποιήθηκαν. Οι δύο πρώτες τεχνικές που είχαν τα καλύτερα ποσοστά είναι LibSVM και Logistic Function. Μπορούμε επίσης από τον Πίνακα 5.5 να δούμε ότι και στις τεχνικές που χρησιμοποίησε ο Πέτρος Κουντούρης υπήρξε περισσότερη αύξηση στο SOV Score. Ακόμη αν δούμε αναλυτικά την πρόβλεψη κάθε κατάστασης είναι φανερό ότι η δική μας τεχνική υστερεί κυρίως στη σωστή πρόβλεψη της κατάστασης E (Extended β-strands) αφού στο ποσοστό σωστή πρόβλεψης είναι 63,98% για Q3 και 66,91 για SOV σε σχέση με το 65,78% σε Q3 και 68,61 σε SOV της μεθόδου LibSVM. Αντίθετα όμως όσον αφορά τη κατάσταση L (Loop) παρατηρούμε ότι η δική μας μέθοδο ξεπέρασε το ποσοστό του Q3 τόσο του LibSVM όσο και του Logistic αφού έφτασε το 83,18% σε σχέση με το 82,42% και 80,76% αντίστοιχα.

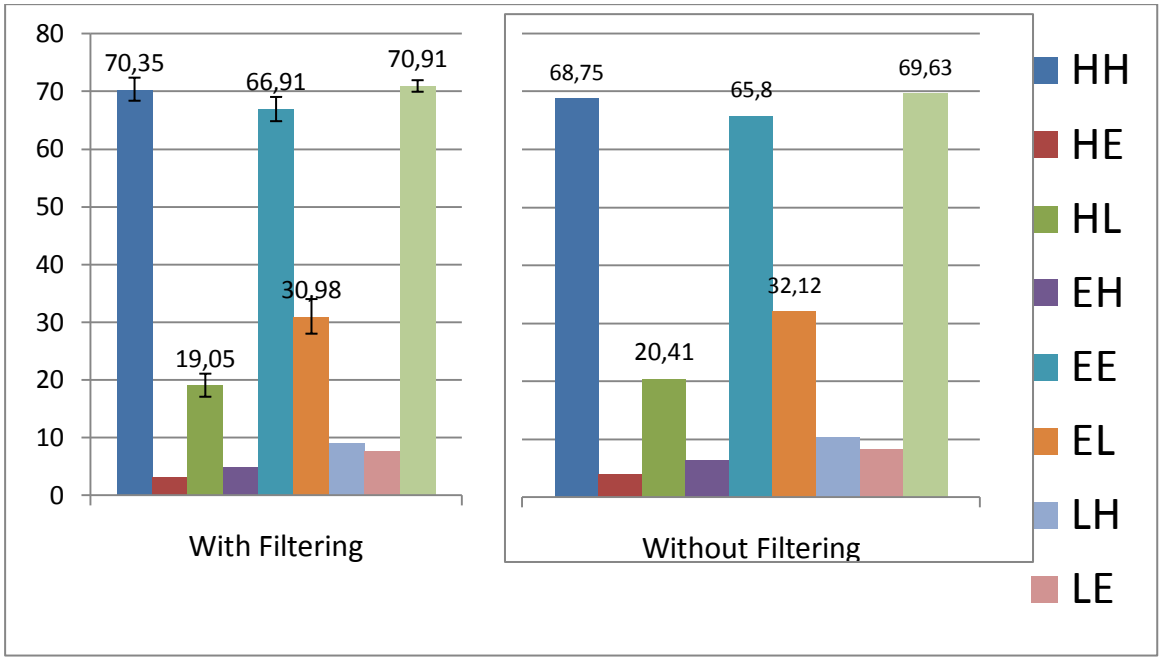
Όπως βλέπουμε από την γραφική παράσταση του σχήματος 5.13, η ομάδα των Strands (E), δεν προβλέπεται σωστά σε μεγάλο βαθμό. Βλέπουμε ότι περίπου το 0,64 κατά μέσο όρο (E) προβλέπονται σωστά από το δίκτυο. Η ομάδα των (L) κατά μέσο όρο προβλέπεται σωστά από το δίκτυο, όπως παρατηρούμε και από την τρέχουσα γραφική. Τέλος η ομάδα των (H), προβλέπεται σε ικανοποιητικό βαθμό από το δίκτυο. Το πρόβλημα είναι με την ομάδα (E). Αυτό σημαίνει ότι το δίκτυο μας εξακολουθεί να παρουσιάζει μειωμένα ποσοστά στη σωστή πρόβλεψη της κατάστασης E.



Σχήμα 5.13: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα όσον αφορά το Q3 και το SOV. Τα αποτελέσματα που παρουσιάζονται είναι συνοπτικά για 4 εκτελέσεις. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

Αν κοιτάξουμε τα αναλυτικά ποσοστά της κάθε κατάστασης και για το SOV Score (Σχήμα 5.13), μπορούμε να δούμε ότι και πάλι η κατάσταση E παρουσιάζει τη χαμηλότερη βελτίωση. Αντίθετα οι άλλες δύο καταστάσεις παρατηρούμε ότι αυξήθηκαν σχεδόν 2%, η αύξηση για τη κατάσταση E είναι μόλις 0,9%.

Αφού είδαμε τα αποτελέσματα όσον αφορά την σωστή πρόβλεψη του δικτύου, δηλαδή τα EE, HH, και LL, παρατηρήσαμε ότι έχουμε κάποιο πρόβλημα γιατί ο αριθμός των Strands (E) που προβλέπονται κανονικά σε Strands (E) εξακολουθεί να είναι χαμηλός. Θα επικεντρωθούμε κυρίως στα ποσοστά σε SOV που είναι και το μέτρο σύγκρισης που είχε την περισσότερη βελτίωση. Προφανώς, το πρόβλημα αυτό, ίσως να οφείλεται σε μια «σύγχυση» κάποιων ομάδων μεταξύ τους. Για να το δούμε αυτό, πρέπει να δούμε τα αποτελέσματα για τα υπόλοιπα στοιχεία, δηλαδή EH, EL και άλλα (σχήμα 5.14). Επίσης είναι καλό να δούμε πόσο είναι το ποσοστό αναλυτικά για κάθε fold του CB513, για εξακριβώσουμε αν το πρόβλημα είναι τοπικό σε κάποιο υποσύνολό. Επίσης στο Σχήμα 5.14 φαίνονται συγκριτικά τα αναλυτικά αποτελέσματα σε σχέση με αυτά στα οποία δεν έχει γίνει filtering.



Σχήμα 5.14: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L με filtering και χωρίς filtering, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα καθώς και τις λανθασμένες προβλέψεις HE, HL, EH, EL, LE, LH. Για παράδειγμα η κατάσταση LE σημαίνει ότι η επιθυμητή έξοδος είναι L και εμείς προβλέψαμε E. Τα αποτελέσματα αποτελούν το συνολικό αποτέλεσμα όλων των folds του CB513, ενώ η απόκλιση αντιπροσωπεύει το διαφορά του αποτελέσματος μεταξύ των folds.*

Βλέπουμε στο σχήμα 5.14 πιο καθαρά, τα κατά μέσο όρο ποσοστά σε SOV για το συνολικό αποτέλεσμα όλων των folds του CB513, ενώ η απόκλιση αντιπροσωπεύει το διαφορά του αποτελέσματος μεταξύ των folds. Παρατηρούμε ότι η πρόβλεψη (HH), η οποία είναι μια από τις τρεις που θέλουμε, είναι και η πιο *σωστή* πρόβλεψη έναντι της λανθασμένης κατάταξης (HE) και (HL). Άρα το δίκτυο μπορεί σωστά να προβλέψει σε μεγάλο βαθμό την ομάδα (H). Όσον αφορά την ομάδα (E), όπως βλέπουμε από τα αποτελέσματα, δεν μπορεί να προβλεφτεί σωστά γιατί συγχύζεται με την ομάδα των Loops (L). Τέλος η ομάδα των (L) είναι και η πιο σωστά προβλεπόμενη ομάδα, αφού σε μεγάλο βαθμό τα δεδομένα της προβλέπονται σωστά (LL).

Το πρόβλημα αυτό μπορεί να υφίσταται για πολλούς λόγους: α) από την φύση τους, ο αριθμός των Strands (E) που κωδικοποιούν ένα συγκεκριμένο τμήμα έχουν γενικά μικρότερο μήκος από τα τμήματα που κωδικοποιούν Helices και Loops, γι' αυτό πιθανόν τα Strands να *χάνονται* στο μέγεθος του παραθύρου που χρησιμοποιούμε, β) πολύ πιθανόν μέρος του προβλήματος να οφείλεται στο σύνολο δεδομένων που χρησιμοποιούμε. Μπορεί το σύνολο δεδομένων να μην αντιπροσωπεύει ικανοποιητικά την ομάδα των Strands (E), γ) η τρέχουσα κωδικοποίηση του επιπέδου εξόδου. Στο επίπεδο εξόδου έχουμε οχτώ πιθανούς συνδυασμούς που κωδικοποιούν κάποια δευτεροταγή δομή. Επειδή όμως όπως είναι υλοποιημένο το σύστημα, η ομάδα των (L), έχει 6 συνδυασμούς που την αναπαριστούν, αυτό μπορεί να αποτελεί πρόβλημα, γιατί με αυτόν τον τρόπο πιθανόν να δίνεται μεγαλύτερο βάρος στην ομάδα αυτή.

Επίσης αν συγκρίνουμε τα τελικά ποσοστά με filtering και χωρίς (Σχήμα 5.14) μπορούμε να δούμε ότι σχεδόν σε όλες τα καταστάσεις λανθασμένης πρόβλεψης τα ποσοστά έχουν μειωθεί. Για αυτό και το συγκεντρωτικό SOV ανέβηκε στο 70,25. Παρόλο που το ολικό SOV ανέβηκε εντούτοις το ποσοστό λανθασμένης πρόβλεψης της κατάστασης έχει πάλι το πιο μεγάλο ποσοστό λανθασμένης πρόβλεψης. Συνολικά ποσοστό 30,9 συγχέεται με τη κατάσταση L σε σχέση με το 32,12 που ήταν πριν. Ακόμα μια λανθασμένη κατάσταση που παρουσιάζει κάποιο πρόβλημα αλλά σε μικρότερο βαθμό είναι η HL, όπου το ποσοστό είναι 19,05 με filtering. Αυτό

φυσικά δεν είναι τόσο σοβαρό γιατί η ορθή πρόβλεψη σε H και L είναι αρκετά μεγάλη.

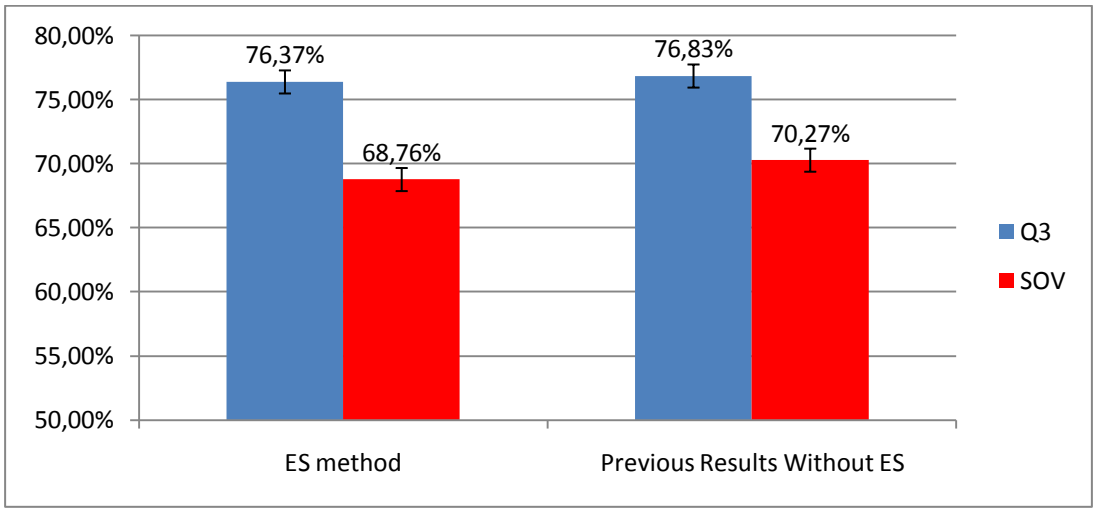
Συμπερασματικά μπορούμε να αντιληφθούμε ότι πράγματι η μέθοδος του post-processing filtering βοηθά αισθητά στην επίλυση του προβλήματος πρόβλεψης της δευτεροταγούς δομής πρωτεϊνών και μπορεί να δώσει καλύτερα αποτελέσματα σωστής πρόβλεψης, κυρίως όσο αφόρα το SOV score. Σε αυτό σημαντικό παράγοντα έπαιξε και η εισαγωγή του κινητού παραθύρου. Με το να επεξεργαζόμαστε περισσότερη πληροφορία για κάθε θέση μιας πρωτεΐνης επέφερε ακόμα πιο ψηλά ποσοστά στο filtering.Τα τελικά αποτελέσματα έδειξαν ότι το filtering βοήθησε το SOV για τη κατάσταση H να αυξηθεί σχεδόν κατά 2%, του L κατά 1,2% και σε πιο λίγο βαθμό το E κατά 0,9%. Επίσης μπορούμε να δούμε ότι με μέθοδο αυτή καταφέραμε να μειώσουμε το ποσοστό της λανθασμένης πρόβλεψης για ELτο οποίο είναι ένα από τα προβλήματα που είχε δυσκολέψει ιδιαίτερα και τη Χριστοδούλου (2010).

5.4 Αποτελέσματα Εξελικτικών Στρατηγικών

Στο επόμενο στάδιο της ανάπτυξης της συγκεκριμένης Διπλωματικής Εργασίας προσπαθήσαμε να ενσωματώσουμε διάφορες Εξελικτικές Μεθόδους στο σύστημα. Όπως αναφέρθηκε στο Υποκεφάλαιο 4.4 παρόμοια προσπάθεια είχε γίνει και από τη Χριστοδούλου (2010). Η Χριστοδούλου χρησιμοποίησε Γενετικούς Αλγόριθμούς (ΓΑ) για την εξέλιξη των τιμών των παραμέτρων του BRNN. Παρόλο που οι ΓΑ έχουν την ικανότητα να ενσωματώνονται σε προβλήματα τεχνητών νευρωνικών δικτύων και να δίνουν λύσεις αποδοτικές, εντούτοις στο συγκεκριμένο πρόβλημα δεν βοήθησαν σχεδόν καθόλου. Ένας πιθανώς λόγος για το γεγονός αυτό είναι ότι ίσως δεν προσεγγίστηκαν και δεν χρησιμοποιήθηκαν σωστά στη επίλυση του συγκεκριμένου προβλήματος. Για αυτό το λόγο στα πρόθυρα αυτής της Διπλωματικής Εργασίας χρησιμοποιήθηκε η ομάδα των Εξελικτικών Στρατηγικών (ΕΣ) οι περιγράφτηκαν αναλυτικά στο Υποκεφάλαιο 4.4.

5.4.1 Αποτελέσματα Εξέλιξης Τιμών του Ensemble Δικτύου

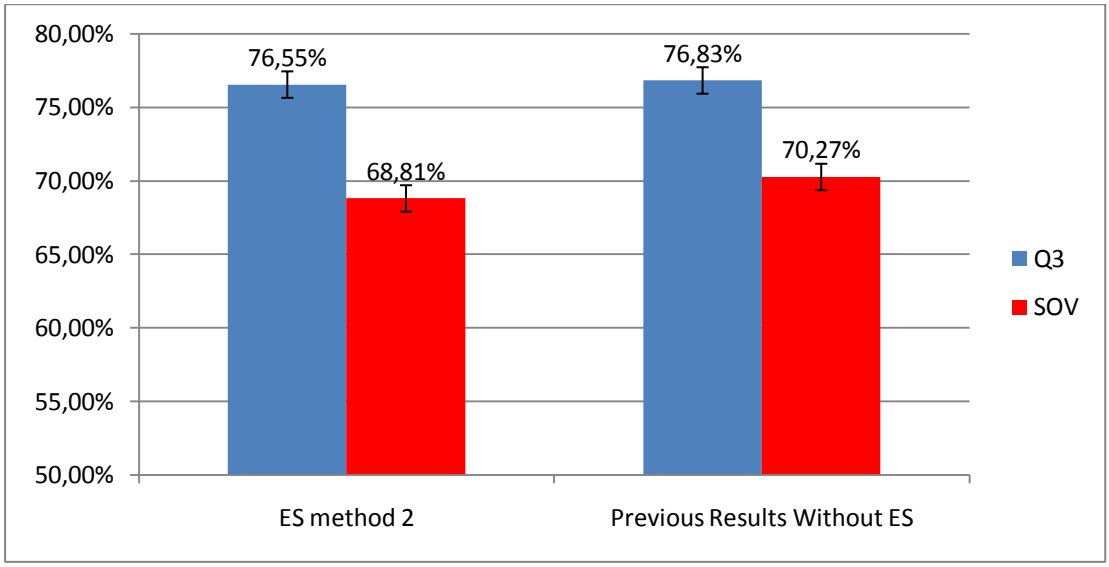
Η πρώτη προσπάθεια που έγινε ήταν να εξελίξουμε τις τιμές συμμετοχής του κάθε BRNN του Ensemble στο τελικό αποτέλεσμα. Σε αυτή τη προσέγγιση θα προσπαθήσουμε να αλλάξουμε το ρόλο που διαδραματίζει κάθε μέλος του Ensemble στο τελικό αποτέλεσμα. Όπως εξηγήθηκε στο Υποκεφάλαιο 4.2 τα καλύτερα αποτελέσματα για Ensemble πάρθηκαν με τη μέθοδο Average. Σύμφωνα με τη μέθοδο αυτή, αφού έχουμε 6 BRNN καθένα από αυτά λαμβάνει μέρος στο τελικό αποτέλεσμα σε αναλόγια 1/6. Εμείς θα εξελίξουμε την τιμή συμμετοχής κάθε BRNN ούτως ώστε όταν κάποιο από αυτά έχει καλύτερα αποτελέσματα τότε θα συμμετέχει περισσότερο στο τελικό αποτέλεσμα. Στη πρώτα πειράματα που εκτελέσαμε κρατήσαμε σταθερό στη υλοποίηση μας, το γεγονός ότι το άθροισμα της συμμετοχής κάθε BRNN πρέπει να ισούται με 1. Τα αποτελέσματα φαίνονται στο πιο κάτω Σχήμα 5.15.



Σχήμα 5.15: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV, συνολικά για τέσσερις εκτελέσεις της μεθόδου εξέλιξης των τιμών συμμετοχής κάθε BRNN στο τελικό αποτέλεσμα. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

Από τα πειράματα που εκτελέσαμε (Σχήμα 5.15) μπορούμε να δούμε ότι η υλοποίηση αυτή δεν επέφερε κάποια αισθητή διαφορά στο ποσοστό επιτυχίας, πέραν από μία περίπτωση όπου υπήρξε μια μικρή αύξηση από 76,39% σε 76,43% σε Q3 η οποία όμως μπορεί να θεωρηθεί και τυχαία αφού δεν επαναλήφθηκε.

Στη συνέχεια δοκιμάσαμε να αφήσουμε τις τιμές να ξεπεράσουν τη τιμή του ενός και να πάρουν τυχαίες τιμές είτα θετικές, είτε αρνητικές για να δούμε πώς θα αντιδράσει το σύστημα.



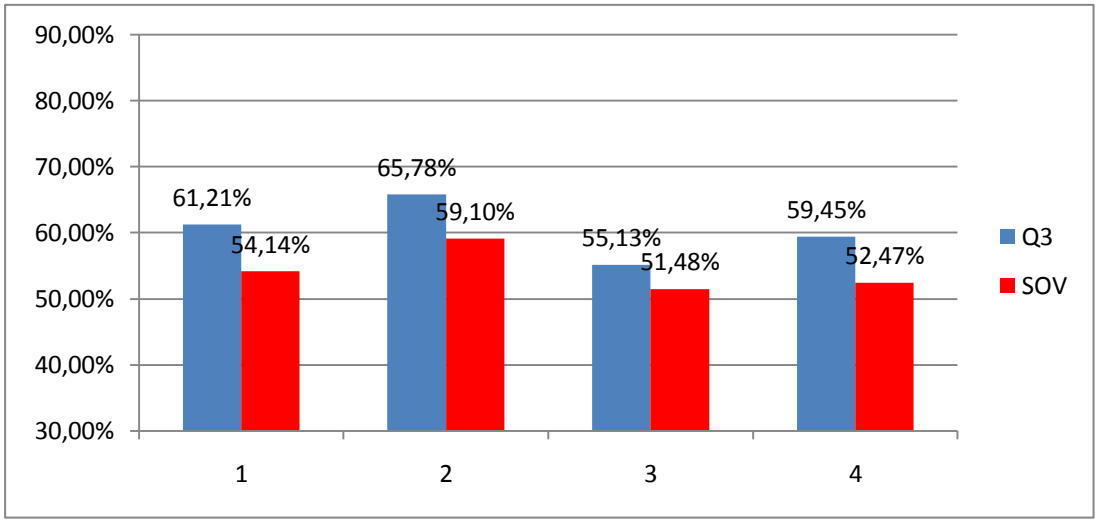
Σχήμα 5.16: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV, συνολικά για τέσσερις εκτελέσεις της μεθόδου εξέλιξης των τιμών συμμετοχής κάθε BRNN στο τελικό αποτέλεσμα όταν το άθροισμα τους ξεπερνά το 1. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

Τα αποτελέσματα που φαίνονται στο σχήμα 5.16 έδειξαν κάποια μικρή αύξηση κυρίως στη Q3 μέτρηση. Όπως μπορούμε να δούμε το Q3 αυξήθηκε από το 76,39% σε 76,55%, δηλαδή περίπου 0,15% . Αυτή η αύξηση εμφανίζεται σε όλα τις δοκιμές που κάναμε, επομένως δε μπορεί να θεωρηθεί τυχαία. Και πάλι όμως δε κατάφερε να ξεπεράσει το ποσοστό που είχαμε πετύχει με τη μέθοδο του filtering το οποίο έφτασε σε 76,8% για Q3 και 70,25 για SOV. Επίσης αυτό αποδεικνύει ότι η μέθοδος του Ensemble που υλοποιήθηκε στο Υποκεφάλαιο 5.2 είναι αρκετά ακριβής και αξιόπιστη και οποιαδήποτε εξέλιξη της δεν επιφέρει κάποια αισθητή βελτίωση.

5.4.2 Αποτελέσματα Εξέλιξης Τιμών των Βαρών του BRNN

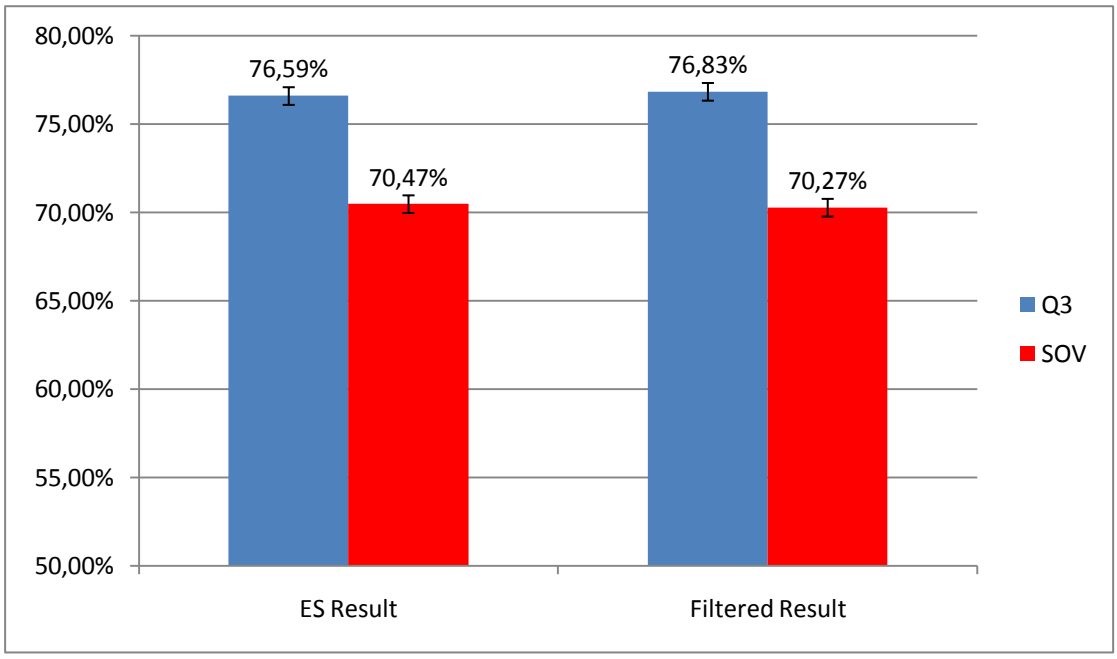
Η δεύτερη προσπάθεια που έγινε ήταν να εξελίξουμε τις τιμές των βαρών όλων των νευρώνων του BRNN. Για τη μέθοδο αυτή προσεγγίσαμε δύο λύσεις. Στην πρώτη περίπτωση αφαιρέσαμε τελείως τη μάθηση με αναστροφή λάθους από την εκπαίδευση του BRNN και η μάθηση πλέον γίνεται μόνο με τη χρήση των ΕΣ, ενώ στη δεύτερη περίπτωση το BRNN θα εκπαιδευτεί κανονικά με Back-Propagation (Υποκεφάλαιο 2.3, Σχήμα 2.5) και μετά το τέλος της εκπαίδευσης του εφαρμόζεται ο Εξελικτικός Αλγόριθμος (Σχήμα 4.23).

Λόγω της μεγάλης διαρκείας εκτέλεσης του προγράμματος για την ενσωμάτωση των μεθόδων αυτών απενεργοποιήθηκαν η χρήση του Ensemble που είχε προσθέσει πολύ χρόνος στο πρόγραμμα. Και στις δύο περιπτώσεις θα ακολουθήσουμε την ίδια αρχιτεκτονική και τις ίδιες τιμές των παραμέτρων που βρήκαμε ότι είναι οι βέλτιστες. (Πίνακας 5.3). Από τις παραμέτρους αυτές μπορούμε εύκολα να υπολογίσουμε ότι ο αριθμός των βαρών που έχουμε να εξελίξουμε είναι 3089 και επομένως κάθε χρωμόσωμα θα έχει μήκος 3089. Επίσης ο αρχικός πληθυσμός που θα χρησιμοποιήσουμε είναι 200 άτομα ενώ το ίδιο μέγεθος έχει και ο προσωρινός πληθυσμός. Τα αποτελέσματα της πρώτης τεχνικής φαίνονται στο Σχήμα5.17.



Σχήμα 5.17: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV για τέσσερις εκτελέσεις της μεθόδου εξέλιξης των τιμών των βαρών του BRNN χωρίς τη χρήση του Back-Propagation για εκπαίδευση. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

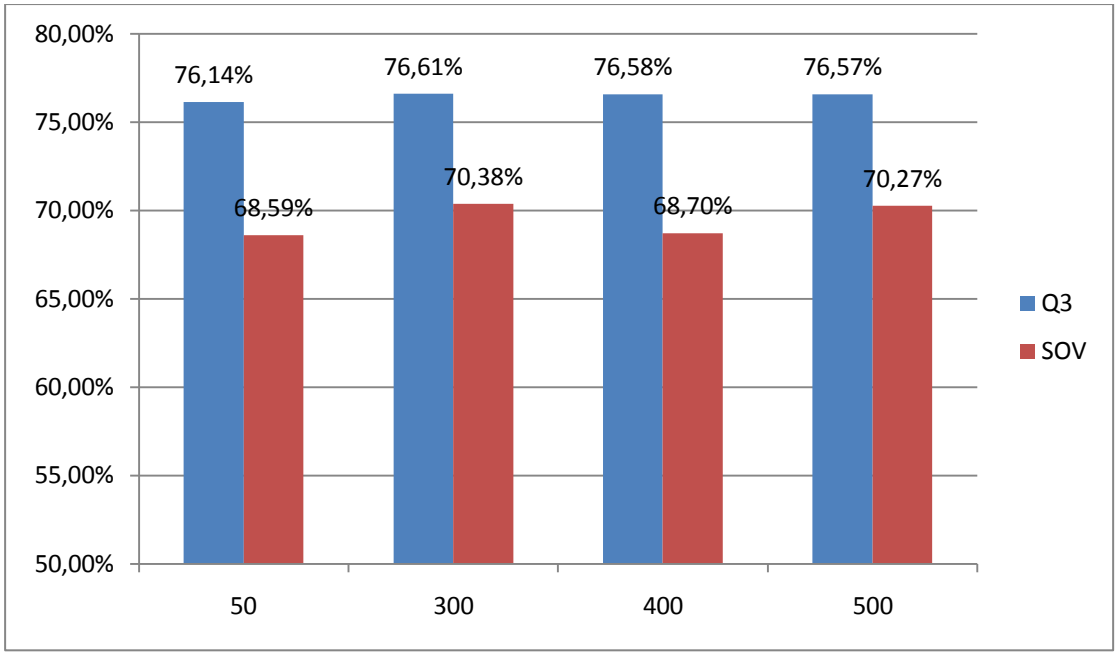
Από τα αποτελέσματα της γραφικής παράστασης μπορούμε να δούμε ότι η τεχνική αυτή δεν επέφερε καλύτερα αποτελέσματα αλλά αντίθετα παρατηρούμε μια πολύ χαμηλή πρόβλεψη και καθόλου ομοιόμορφη. Αυτό το πιο πιθανόν να οφείλεται στη αρχικοποίηση των ατόμων του πληθυσμού ή στον αριθμό των γενεών. Αυτό στην ουσία ήταν κάτι που το περιμέναμε αφού οι Εξελικτικοί Αλγόριθμοι δεν μπορούν να επιλύσουν τόσο αποδοτικά το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής πρωτεϊνών από μόνοι τους. Για αυτό το λόγο πρέπει να συνδυαστούν με τα ΤΝΔ για να επιφέρουν καλύτερα αποτελέσματα. Για αυτό άλλωστε υλοποιήσαμε τη δεύτερη μέθοδο που προαναφέραμε και τα αποτελέσματα της οποίας φαίνονται στο πιο κάτω Σχήμα 5.18.



Σχήμα 5.18: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV, συνολικά για τρεις εκτελέσεις της μεθόδου εξέλιξης των τιμών των βαρών του BRNN αφού γίνει εκπαίδευση του δικτύου με τη χρήση του Back-Propagation. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

Οι εξελικτικοί αλγόριθμοι όπως αναφέραμε στο Υποκεφάλαιο 4.4 εξαρτώνται κυρίως από την τυχαία μετάλλαξη. Για αυτό το πρόγραμμα τρέξαμε το πρόγραμμα 4 φορές για να επιβεβαιώσουμε τα αποτελέσματα μας. Η μεγάλη διάρκεια εκτέλεσης του προγράμματος δεν βοηθούσε στην εκτέλεση περισσότερων επαναλήψεων. Από τα αποτελέσματα της γραφικής παράστασης (Σχήμα 5.18) μπορούμε να δούμε ότι αυτή η μέθοδος συντέλεσε στην αύξηση του ποσοστού επιτυχίας σε Q3 αλλά κυρίως σε SOV. Η αύξηση αυτή κατάφερε να ξεπεράσει τα επίπεδα που είχαμε πετύχει τόσο με τη μέθοδο του Ensemble, όσο και με τη μέθοδο του filtering για SOV, αφού έφτασε στο 70,38. Αντίθετα το Q3 παρατηρούμε ότι έμεινε σε πιο χαμηλά ποσοστά αφού έφτασε μέχρι 76,61%, ποσοστό που ξεπερνά αυτό του Ensemble αλλά όχι του filtering.

Για να δούμε αν ο αριθμός των γενεών παίζει ρόλο ή θα επιφέρει κάτι καλύτερο, σκεφτήκαμε να αλλάξουμε το μέγεθος του. Αξίζει να σημειώσουμε ότι όσο αυξάνονται οι γενεές αυξάνεται και ο χρόνος εκτέλεσης. Για παράδειγμα όταν ο γενεές ήταν 300 άτομα ο χρόνος εκτέλεσης ήταν σχεδόν 8 μέρες. Για αυτό κάναμε μόνο δύο δοκιμές με μεγάλο αριθμό γενεών. Τα αποτελέσματα με διάφορους πληθυσμούς φαίνονται στο πιο κάτω Σχήμα 5.19.



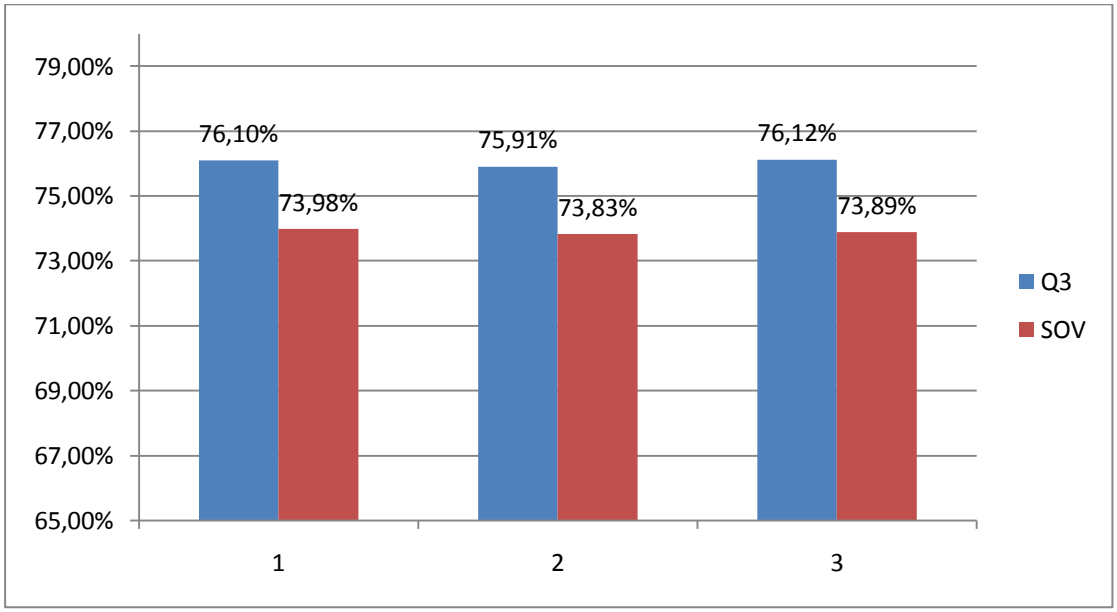
Σχήμα 5.19: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV χρησιμοποιώντας τη μεθόδου εξέλιξης των τιμών των βαρών του BRNN με διαφορετικούς αριθμούς γενεών και αφού γίνει εκπαίδευση του δικτύου με τη χρήση του Back-Propagation. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

Τα αποτελέσματα έδειξαν ότι περισσότερες γενεές δεν βοήθησαν περισσότερο την εφαρμογή αυτή. Παρατηρούμε ότι περίπου 200 επαναλήψεις είναι αρκετές για να συγκλίνει ο αλγόριθμος.

Επομένως συνοπτικά μπορεί οι μέθοδοι των Εξελικτικών Στρατηγικών να μην κατάφεραν να ξεπεράσουν το ποσοστό Q3 που πετύχαμε με τη μέθοδο του filtering, όμως τα αποτελέσματα της τελευταίας τεχνικής έδειξαν ότι η ενσωμάτωση τους μπορεί να βοηθήσει στην αύξηση του ποσοστού SOV. Αυτό που μπορούμε να πούμε είναι η τελευταία τεχνική που περιγράφτηκε μπορεί να επιτελέσει το ίδιο, ίσως και καλύτερο, ρόλο με την εφαρμογή του Ensemble με τη μόνη σημαντική διαφορά το χρόνο εκτέλεσης. Η μέθοδος του Ensemble χρειαζόταν 3 με 4 μέρες για τρέξει όλο το CB513, ενώ οι ΕΣ χρειάστηκαν μέχρι και 8 μέρες για το ίδιο σύνολο.

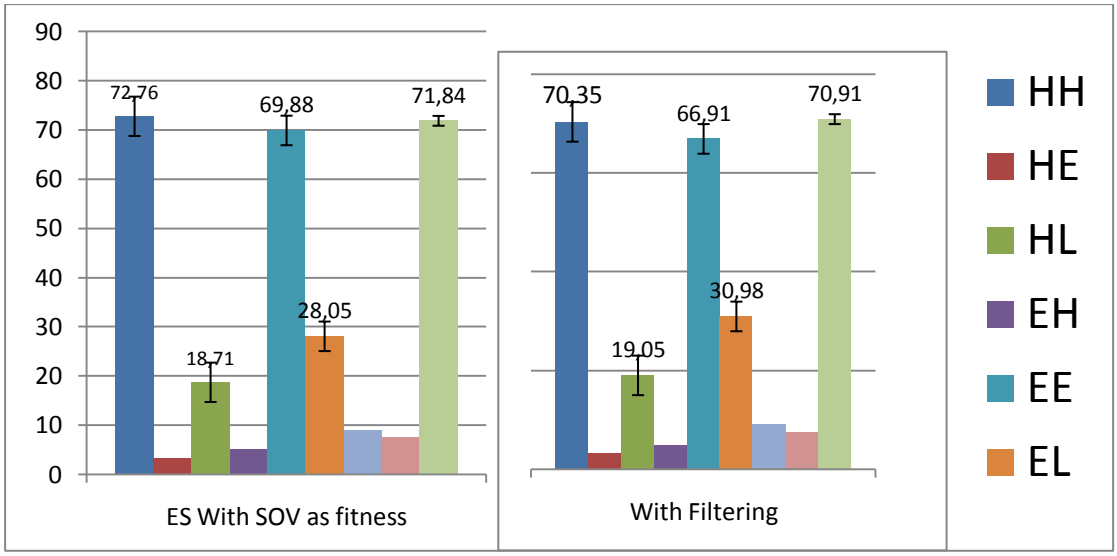
5.4.3 Αποτελέσματα Εξέλιξης Τιμών με τη χρήση του SOV Score

Σκεφτήκαμε να τροποποιήσουμε τον αλγόριθμο της τελευταίας τεχνικής που αναφέρθηκε στην προηγούμενη παράγραφο και ο οποίος είχε τα καλύτερα αποτελέσματα. Αντί να χρησιμοποιούμε το mean square error ως κριτήριο του *fitness function*, χρησιμοποιήσαμε το SOV Score. Η δομή των ΕΣ είναι τέτοια που καταστούσε εύκολη τη τροποποίηση και ενσωμάτωση του κριτηρίου SOV. Ένα μεγάλο μειονέκτημα της μεθόδου αυτής είναι και πάλι ο χρόνος εκτέλεσης, ο οποίος αυξάνεται ακόμη περισσότερο. Πάλι ο τελικός χρόνος ολόκληρου του CB513 έφτασε τις 10 μέρες. Για τον υπολογισμό του SOV όπως αναφέρθηκε στο Υποκεφάλαιο 4.4.2.β χρειάστηκε να χρησιμοποιήσουμε το πρόγραμμα *sov.c* το οποίο γράφει και διαβάζει από αρχεία, διαδικασίες οι οποίες είναι αρκετά αργές. Η δομή των ΕΣ που χρησιμοποιήσαμε προϋποθέτει όπως και πριν την ύπαρξη πληθυσμού 200 ατόμων καθώς επίσης και προσωρινού πληθυσμού ακόμη 200 ατόμων. Επομένως σε κάθε γενεά πρέπει να χρησιμοποιούμε 400 αρχεία, κάτι που είναι πολύ χρονοβόρο. Παρόλα αυτά τρέξαμε το πρόγραμμα για όλα τα folds του CB513, 3 φορές και τα αποτελέσματα εμφανίζονται στο Σχήμα 5.20.



Σχήμα 5.20: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας σε Q3 και SOV, συνολικά για τρεις εκτελέσεις της μεθόδου εξέλιξης των τιμών των βαρών του BRNN όταν το κριτήριο του fitness function είναι το SOV. Κάθε εκτέλεση αναφέρεται και στα 10 folds του CB513, όπου το τελικό αποτέλεσμα ορίζεται ως η συνένωση των αποτελεσμάτων των 10 ανεξάρτητων folds.*

Είναι φανερό ότι έχουμε επιτύχει μεγάλη αύξηση στο SOV αφού έφτασε σχεδόν στο 74%, ενώ το Q3 όπως βλέπουμε τις πλείστες περιπτώσεις έχει μια μικρή μείωση. Αν δούμε την αναλυτική πρόβλεψη για SOV για κάθε κατάσταση μπορούμε να δούμε σε ποιους τομείς βοήθησε περισσότερο η μέθοδος αυτή. Σχήμα 5.21



Σχήμα 5.21: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, με τη χρήση ES για εξέλιξη εξέλιξης των τιμών των βαρών του BRNN όταν το κριτήριο του fitness function είναι το SOV καθώς επίσης και τα αποτελέσματα με filtering για τρεις εκτελέσεις, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα καθώς και τις λανθασμένες προβλέψεις HE, HL, EH, EL, LE, LH για SOV. Για παράδειγμα η κατάσταση LE σημαίνει ότι η επιθυμητή έξοδος είναι L και εμείς προβλέψαμε E. Τα αποτελέσματα αποτελούν το συνολικό αποτέλεσμα όλων των folds του CB513, ενώ η απόκλιση αντιπροσωπεύει το διαφορά του αποτελέσματος μεταξύ των folds.*

Στο πειράματα που κάναμε με filtering είδαμε ότι στο Σχήμα 5.14 η πρόβλεψη των E συγχεόταν με την κλάση των L με αποτέλεσμα το 31,0 σχεδόν των E να προβλέπεται σαν άλλη κατάσταση. Αυτό το γεγονός παρατηρούμε ότι βελτιώνεται αισθητά με τη χρήση των ΕΣ. Όπως παρατηρούμε και από τις τιμές στο Σχήμα 5.23 το ποσοστό των E που προβλέπεται σαν L είναι περίπου 27,95, που είναι μια πολύ σημαντική

ενίσχυση για το σύστημα ενώ το ποσοστό σωστής πρόβλεψης της κατάστασης E έφτασε σχεδόν 70,0. Αυτό η άνοδος παρουσιάστηκε και για τις άλλες δύο καταστάσεις, όμως δεν ήταν τόσο αισθητή. Όπως μπορούμε να δούμε στο σχήμα 5.23 οι καταστάσεις L και H δεν είχε την ίδια άνοδο αφού έφτασαν το 73,0 και 71,98 αντίστοιχα. Είναι φανερό όμως το γεγονός ότι αυτή η υλοποίηση που παίρνει σαν πρώτο κριτήριο το SOV φαίνεται ότι βοήθησε το σύστημα να αυξήσει σε κάποιο βαθμό το συνολικό SOV Score έφτασε στο 74%.

Δεν πρέπει να παραλείψουμε να πούμε ότι αυτή η υλοποίηση παρουσιάζει και κάποια αρνητικά αποτελέσματα. Από τα Σχήμα 5.20 όπως είδαμε το Q3 παρουσιάζει χαμηλότερα ποσοστά. Παρόμοια συμπεριφορά είχε παρατήρησε και η Χριστοδούλου (2010), όταν είχε χρησιμοποιήσει τη μέθοδο του *viterbi* με Hidden Markov Models για φιλτράρισμα. Συνοπτικά, μπορούμε να δούμε ότι με αυτή τη υλοποίηση των ΕΣ, μπορεί μεν να μειώθηκε το Q3 ποσοστό, όμως η μείωση δεν είναι τόσο μεγάλη, όσο η άνοδος σε SOV που ήταν αρκετά αισθητή. Επομένως αν σκοπός μας είναι η αύξηση του SOV, δηλαδή αύξηση της σωστής πρόβλεψης για διαστήματα από κατάλοιπα που επικαλύπτονται στις δύο ακολουθίες της προβλεπόμενης και πραγματικής ακολουθίας αντίστοιχα, τότε η μέθοδος αυτή όπου χρησιμοποιεί το SOV ως κριτήριο στο fitness function επιφέρει πολύ καλές βελτιώσεις.

5.5 Γενικές παρατηρήσεις και συζήτηση

Κλείνοντας το κεφάλαιο το πειραμάτων, όπως είχαμε αναφέρει στην αρχή του κεφαλαίου αυτού, θα κάνουμε μια απλή σύγκριση με τα αποτελέσματα που είχε πετύχει η Χριστοδούλου (2010). Η Χριστοδούλου στα τελικά στάδια είχε καταφέρει να πετύχει ποσοστό 76% για Q3 και 70,0 για SOV. Εμείς καταφέραμε να πάρουμε ποσοστό 76,4% για Q3 και 68.75 για SOV χωρίς τη χρήση οποιασδήποτε εφαρμογής filtering ή Εξελικτικών Αλγορίθμων. Παρατηρούμε ότι πετύχαμε κάποια αύξηση στο Q3 αφού ανέβηκε 0,4%, όμως αντίθετα είχαμε αισθητή πτώση της τάξεως 1,25 για το SOV. Όμως εφαρμόζοντας τη τεχνική του filtering με RBF δίκτυο είδαμε ότι αυξήσαμε το ποσοστό ακρίβεια αφού έφτασε το 76,8% για Q3 και 70,25 για SOV. Επιπρόσθετα με την εφαρμογή των Εξελικτικών Στρατηγικών που χρησιμοποιούσε ως *fitness function* τη παράμετρο SOV, είδαμε ότι το SOV ανέβηκε μέχρι και 74,0 ενώ το Q3 πέφτει στο 76,1%.

Επομένως συνοπτικά βλέπουμε ότι τα ποσοστά αυξήθηκαν σε σχέση με τις μετρήσεις που είχε κάνει η Χριστοδούλου (2010). Συνεπώς οι μέθοδοι και τεχνικές που ενσωματώθηκαν μπορούν να θεωρηθούν επιτυχημένες, αφού επέφεραν αύξηση στο τελικό ποσοστό πρόβλεψης δευτεροταγούς δομής πρωτεϊνών. Όμως είναι καλό να αναφέρουμε ξανά ότι δεν έγιναν αναλυτικές συγκρίσεις με τα αποτελέσματα τις Χριστοδούλου (2010) γιατί όπως αναφέραμε και προηγουμένως δεν χρησιμοποιήσαμε το ίδιο σύνολο δεδομένων εκπαίδευσης, μια παράμετρος που παίζει καθοριστικό ρόλο στη τελικό αποτέλεσμα των ΤΝΔ.

Κεφάλαιο 6

Συμπεράσματα και μελλοντική εργασία

6.1 Συμπεράσματα

Αρχικά, θα αναφερθούμε στις τροποποιήσεις που αφορούν την αρχιτεκτονική και τις παραμέτρους του BRNN. Όπως είδαμε από τα πειράματα αλλαγές σε κάποιες από τις τιμές των παραμέτρων του BRNN πέτυχαν μια μικρή βελτίωση των ποσοστών ακρίβειας του συστήματος. Ίσως η πιο σημαντική αλλαγή ήταν η εισαγωγή του κινητού παράθυρου για το κεντρικό δίκτυο. Η αλλαγή αυτή αύξησε μεν το ποσοστό αλλά όχι στο βαθμό που περιμέναμε, αφού έδωσε μια αύξηση της τάξεως το 1.7% σε Q3. Αξίζει να σημειωθεί ότι ενώ για τα άλλα δυο δίκτυα του BRNN το μέγεθός του κινητού παράθυρου κατά μέσο όρο όσο αυξανόταν έδινε καλύτερα αποτελέσματα, αντίθετα στη περίπτωση του κεντρικού δικτύου έδινε καλύτερα αποτελέσματα όσο αυτό διατηρείται ήταν μικρό. Αυτό οφείλεται στο γεγονός ότι τα μεγάλα παράθυρα των άλλων δύο δικτύων είναι αρκετά και μπορούν να συσχετίσουν μια μεγαλύτερη περιοχή από κατάλοιπα μιας πρωτεΐνης μεταξύ τους. Έτσι, με τα μεγάλα κινητά παράθυρα των Forward και Backward δικτύου καλύπτουμε σε μεγάλο βαθμό τις μακρινές πληροφορίες που αφορούν τα αμινοξέα. Συνοψίζοντας, ένα κινητό παράθυρο σταθερού μεγέθους για το κεντρικό δίκτυο μπορεί να μη συντέλεσε στο βαθμό που περιμέναμε, όμως, επιτρέποντας μεγαλύτερο μέγεθος παραθύρου και πάλι δεν αυξάνουμε το ποσοστό πρόβλεψης λόγω υπερεκπαίδευσης.

Αντίθετα αισθητή αύξηση επέφερε η εισαγωγή της μεθόδου Ensemble. Η χρήση πολλών BRNN τα οποία αποφασίζουν συλλογικά το τελικό αποτέλεσμα έδωσε πράγματι την δυνατότητα στο σύστημα να προβλέπει με μεγαλύτερη ακρίβεια την δευτεροταγή δομή της πρωτεΐνης. Και οι τέσσερις μέθοδοι που χρησιμοποιήθηκαν έδωσαν καλυτέρα αποτελέσματα, αλλά τελικά επιλέχθηκε η μέθοδος Average που έδινε τα πιο μεγάλα ποσοστό, αφού έφτασε το σύστημα μέχρι 76,4% για Q3 και 68,74 για SOV.

Σημαντική βελτίωση του ποσοστού πρόβλεψης των πρωτεϊνών του συστήματος, είχε και η πρόσθεση του post-processing filtering. Το φιλτράρισμα αποσκοπεί να διορθώσει την προβλεπόμενη ακολουθίας αφού λαμβάνει υπόψη τις συσχετίσεις μεταξύ των στοιχείων της δευτεροταγούς δομής (structure to structure) που έχει

προβλεφτεί για κάθε αμινοξύ από το πρώτο δίκτυο (sequence to structure). Η μέθοδος του filtering εφαρμόστηκε πάνω στις καλύτερες προβλέψεις που είχαμε μέχρι τώρα, δηλαδή 76,4% για Q3 και 68,75 για SOV. Για το φιλτράρισμα των αποτελέσματος μας δοκιμάσαμε ένα δίκτυο RBF του οποίου τα κέντρα αρχικοποιήθηκε βάση ενός χάρτη αυτοοργάνωσης Kohonen. Η παρουσία γραμμικής συνάρτησης ενεργοποίησης στο επίπεδο εξόδου και της μη γραμμικής (Gaussian) στο κρυφό επίπεδο συντέλεσε στη αύξηση της σωστής πρόβλεψης. Με τη χρήση του δικτύου αυτού πετύχαμε σημαντική αύξηση στο SOV που έφτασε το 70,25 ενώ το Q3 ανέβηκε στο 76,8%. Μια σημαντική παρατήρηση είναι ότι για την επίτευξη των ποσοστών αυτών χρησιμοποιήσαμε και πάλι κινητό παράθυρο. Για άλλη μια φορά φαίνεται καθαρά ότι με τη χρήση κινητού παράθυρου καλύπτουμε τις μακρινές πληροφορίες και αλληλεπιδράσεις που αφορούν τα αμινοξέα και επιτυγχάνουμε καλύτερα αποτελέσματα.

Βάσει των αποτελεσμάτων του υποκεφαλαίου 5.3.1, παρατηρούμε το πρόβλημα με τη χαμηλή πρόβλεψη των Strands (E) εξακολουθεί να υπάρχει αλλά σε μικρότερο βαθμό. Το δίκτυο δεν μπορεί να προβλέψει τόσο καλά την ομάδα των (E), όπως τις άλλες δύο καταστάσεις (Σχήματος 5.13).

Αυτό θεωρητικά μπορεί να γίνεται για πολλούς λόγους: α) οφείλεται στην φύση των β – strands. Τα β – strands (E) τμήματα, έχουν γενικά μικρότερο μήκος από τα τμήματα που κωδικοποιούν Helices και Loops. Ένα τέτοιο τμήμα αποτελείται από περίπου 5 – 10 (E), τα οποία βρίσκονται σε ένα μέρος της αλυσίδας, με ένα άλλο τμήμα με 5 – 10 συνεχόμενα (E) να ακολουθεί του πρώτου σε κάποια απόσταση. Τα δύο αυτά τμήματα των (E), είτε διαχωρίζονται με ένα μικρό τμήμα από Loops, ή βρίσκονται αρκετά μακριά το ένα από το άλλο, με διάφορα άλλα είδη δευτεροταγούς δομής ανάμεσά τους (Mount, 2001). Έτσι είναι πιθανόν τα Strands να *χάνονται* στο μέγεθος του παραθύρου που χρησιμοποιούμε. β) Ένας άλλος πιθανός λόγος για την χαμηλή πρόβλεψη των (E) ίσως είναι το σύνολο δεδομένων που χρησιμοποιούμε. Μπορεί το σύνολο δεδομένων να μην αντιπροσωπεύει ικανοποιητικά την ομάδα των Strands σε σχέση με τις άλλες ομάδες. γ) Η τρέχουσα αρχιτεκτονική του δικτύου ίσως οφείλεται κατά ένα μέρος για το πιο πάνω πρόβλημα. Το επίπεδο εξόδου του συστήματος αποτελείται από τρεις νευρώνες, οι

οποίοι ανάλογα με τον τριαδικό *binary* συνδυασμό κωδικοποιούν μια ομάδα δευτεροταγούς δομής (παράρτημα Z). Τι γίνονται όμως οι υπόλοιποι συνδυασμοί; Με βάση την υλοποίηση του δικτύου, οι υπόλοιποι συνδυασμοί καταγράφονται σαν (L). Αυτό μπορεί να δικαιολογήσει πολλά από τα πιο πάνω ερωτήματα, διότι ίσως εδώ να οφείλεται η μεγάλη πρόβλεψη των Loops που έχει το δίκτυο.

Στην συνέχεια εφαρμόσαμε διαφορές μεθόδους Εξελικτικών Αλγορίθμων. Αυτές κατάφεραν να πετύχουν τελικά και τα καλύτερα αποτελέσματα από όλες τις τεχνικές που χρησιμοποιούσαμε όσον αφορά το SOV. Συγκεκριμένα μετά την εκπαίδευση του δικτύου με Back-Propagation, προσπαθήσαμε να εξελίξουμε τις τιμές των βαρών του BRNN, με τη χρήση Εξελικτικών Στρατηγικών, όπου SOV αποτελεί το κριτήριο του fitness function. Προσπαθήσαμε δηλαδή να κάνουμε μικρές αλλαγές στις τιμές του, που θα επιφέρουν καλύτερα ποσοστά. Με την τεχνική αυτή, τα αποτελέσματα πρόβλεψης με βάση την SOV παράμετρο αυξήθηκαν μέχρι και 74.0 ενώ με βάσει την Q3 έπεσαν γύρω στο 76,1%. Επίσης με τη μέθοδο αυτή το πρόβλημα της χαμηλής πρόβλεψης των (E) έχει παρουσιάσει μείωση σε κάποιο βαθμό. Παρόλα αυτά συνεχίζει να υφίστανται .

Τέλος θα αναφερθούμε στο πιο σημαντικό πρόβλημα που αντιμετωπίσαμε σε όλες τις φάσεις της Διπλωματικής Εργασίας. Το πρόβλημα αυτό ήταν ο χρόνος εκτέλεσης. Οποιεσδήποτε αλλαγές κάναμε, πρόσθεταν επιπλέον χρόνο στο ήδη αργό σύστημα. Πιο χαρακτηριστικό παράδειγμα στη φάση υλοποίησης ΕΣ στο Υποκεφάλαιο 5.4.3 όπου ο χρόνος εκτέλεσης για όλο το CB513 έφτασε τις 10 μέρες περίπου. Ακόμη κάποιες άλλες φορές, παρουσιάστηκαν προβλήματα και με τις μηχανές που χρησιμοποιούσαμε για τα πειράματα μας. Λόγω του ότι γέμιζε η RAM η μηχανή δεν μπορούσε να τελειώσει κάποια από τα πειράματα που δοκιμάζαμε. Αυτός ήταν και ο λόγος που δεν μπορέσαμε να κάνουμε περισσότερα πειράματα στις υλοποιήσεις με Εξελικτικούς Αλγόριθμους.

6.1 Μελλοντική Εργασία

Όπως είχαμε δει στο Υποκεφάλαιο 5.4.3 η ενσωμάτωση Εξελικτικών Στρατηγικών που θέτουν ως κριτήριο το SOV στο fitness function, επέφερε καλύτερα ποσοστά όσον αφορά το SOV. Επομένως θα ήταν μια καλή ιδέα να δοκιμάζαμε να εκπαιδεύαμε από την αρχή το BRNN, αλλά αντί με το mean square error μεταξύ επιθυμητής και πραγματικής εξόδου, με βάση το SOV. Ακόμη καλύτερα θα μπορούσαμε να συνδυάσουμε και τις δυο μετρικές που χρησιμοποιούμε για να υπολογίσουμε το ποσοστό σωστή πρόβλεψης του δικτύου, δηλαδή το Q3 και το SOV. Το SOV μπορεί να μην θεωρείται επί τις εκατό ποσοστό όμως μπορεί να συνδυαστεί με το Q3 και θα μπορούσαμε να ορίσουμε ένα μέτρο σύγκρισης όπως αυτό που φαίνεται στην Εξίσωση 6.1. Με αυτό τον τρόπο θα λαμβάνουμε υπόψη και τις δύο μετρήσεις στο τελικό αποτέλεσμα.

$$SEL= (Q3+SOV) \ / 2$$

Εξίσωση 6.1: *Εξίσωση που συνδυάζει και τις δύο μετρικές που χρησιμοποιούμε για την εύρεση του ποσοστού επιτυχής πρόβλεψης δευτεροταγούς δομής πρωτεϊνών.*

Όσον αφορά τη χρήση των Ensemble, είχαν χρησιμοποιηθεί 4 διαφορετικές μέθοδοι για να συνδυάσουμε το αποτέλεσμα των 6 BRNN που συναποτελούν το σύστημα. Υπάρχουν όμως και άλλες μέθοδοι, πέραν από αυτών που αναπτύχθηκαν, που ίσως να δουλεύουν πιο αποτελεσματικά στη επίλυση του συγκεκριμένο προβλήματος και να επιφέρουν καλύτερο ποσοστό πρόβλεψης δευτεροταγούς δομής πρωτεϊνών.

Ο δημιουργός του συστήματος *pssp_ucy* που προσπαθήσαμε να βελτιώσουμε, Μιχάλης Αγαθοκλέους, έχει αναλάβει τη δημιουργία ενός νέου συστήματος. Το νέο σύστημα ασχολείται με την επίλυση του ίδιου προβλήματος, αλλά προσπαθεί να βελτιώσει την ικανότητα πρόβλεψης του BRNN, αντικαθιστώντας τον αλγόριθμο μάθησης Back-Propagation με ένα *second-order learning* αλγόριθμο. Είναι αποδεχτό ότι αυτού του είδους αλγόριθμοι είναι ανώτεροι στην εκπαίδευση

πολυστρωματικών νευρωνικών δικτύων. Συγκεκριμένα ο αλγόριθμος που υλοποίησε είναι ο Scaled-conjugate-gradient (SGC), ο οποίος φαίνεται να είναι πολύ πιο αποδοτικός και αποτελεσματικός όταν εφαρμόζεται σε μεγάλα σύνολα δεδομένων, έναντι του Back-Propagation through Time. Επομένως σκοπός του νέου συστήματος είναι να ενσωματώσει το SCG στο BRNN για την επίλυση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής. Η ενσωμάτωση αυτή αποσκοπεί στην εξάλειψη των σημαντικών μειονεκτημάτων που παρουσίαζε το υπάρχων σύστημα και σχετίζονται κυρίως με το χρόνο εκτέλεσης και την επίδοση του συστήματος στην ορθή πρόβλεψης της δευτεροταγούς δομής πρωτεϊνών.

Κάποια πρώτα πειράματα έχουν δείξει θετικά αποτελέσματα. Παρατηρήθηκε ότι ο χρόνος εκτέλεσης έχει μειωθεί ραγδαία, αφού τώρα χρειάζεται μόνο κάποια λεπτά για να εκπαιδευτεί. Επίσης φαίνεται να δουλεύει αποτελεσματικά σε κάποια απλά προβλήματα νευρωνικών δικτύων ανάδρασης. Άρα θα ήταν καλή ιδέα να γίνουν σε κάποια φάση κάποιες συγκρίσεις μεταξύ των δύο συστημάτων, σε διάφορες πτυχές. Για παράδειγμα το ποσοστό πρόβλεψης της κατάστασης Ε, το οποίο εξακολουθεί να παραμένει σχετικά χαμηλό στο υπάρχων σύστημα, σε σχέση με τη πρόβλεψη των άλλων δύο καταστάσεων. Με αυτό τον τρόπο μπορούμε να δούμε αν αυτή η συμπεριφορά, είναι αποτέλεσμα της αρχιτεκτονικής του συγκεκριμένου συστήματος ή επηρεάζει και άλλα συστήματα.

Αναφορές

Αγαθοκλέους Μ., “Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών με την χρήση Νευρωνικών Δικτύων Αμφίδρομης Ανάδρασης”, Προπτυχιακή διπλωματική, Πανεπιστήμιο Κύπρου, Λευκωσία, 2009.

Χριστοδούλου Γεωργία, «Πρόβλεψη δευτεροταγούς δομής πρωτεϊνών με τη χρήση νευρωνικών δικτύων αμφίδρομης ανάδρασης», Ατομική Διπλωματική Εργασία, Ιούνιος 2010, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

Agathocleous, M., Christodoulou, G., Promponas, V., Christodoulou, C. Vassiliades, V. and Antoniou, A. (2010). Protein Secondary Structure Prediction with Bidirectional Recurrent Neural Nets: can weight updating for each residue enhance performance? In *AIAI 2010*. H. Papadopoulos, A. S. Andreou and M. Bramer (Eds.), IFIP International Federation for Information Processing AICT, **339**, 128-137

A.E. Eiben and J.E. Smith, “Introduction to Evolutionary Computing.” Springer, Natural Computing Series, chapters 2-3 (2003)

Baldi P., Brunak S., Frasconi P., Soda G. and Pollastri G. “Bidirectional Dynamics for Protein Secondary Structure Prediction”, Sequence Learning, LNAI 1828 (2000) : 80-104.

Baldi P., Brunak S., Frasconi P., Soda G. and Pollastri G. “Exploiting the Past and the Future in Protein Secondary Structure Prediction”, Bioinformatics, Vol. 15, No.11 (1999): 937-946.

Blazewicz J., Hammer L.P. and Lukasiak P. ‘‘Predicting Secondary Structures of Proteins Recognizing Properties of Amino Acids with the Logical Analysis of Data Algorithm”, IEEE Engineering in Medicine and Biology Magazine, pp. 88-94, 2005.

Ceroni A., Frasconi P. and Pollastri G. “Learning protein secondary structure from sequential and relational data”, Neural Networks, Volume 18, Issue 8, (2005): 1029-1039.

Chen J. and Chaudhari N. S. "Statistical analysis of long-range interactions in proteins", Proc. Int. Conf. Bioinf. Comput. Biol., pp. 296-302, 2006.

Chen J. and Chaudhari N. S. “Cascaded Bidirectional Recurrent Neural Networks for Protein Secondary Structure Prediction”, IEEE/ACM Transactions on Computational Biology and Bioinformatics, Vol. 14, No. 4, pp. 572-582, 2007.

Chou PY and Fasman GD. “Prediction of protein conformation”. Biochemistry, Vol. 13(2), pp. 222-45, 1974.

Crooks G.E and Brenner S.E. “Protein secondary structure: entropy, correlations and prediction”, Bioinformatics, Vol. 20, No. 10, pp. 1603-1611, 2004.

Durbin, R., Eddy, S. R., Krogh, A. and Mitchison, G. J. “Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids”, Cambridge University Press, Cambridge UK, 1998.

Elman L.J. “Finding Structure in Time”, Cognitive Science, Vol. 14, pp. 179-211, 1990.

Frishman D. and Argos P. “Incorporation of non-local interactions in protein secondary structure prediction from the amino acid sequence” Protein Eng, Vol. 9(2), pp. 133-142, 1996.

Frishman D. and Argos P., “Seventy-five percent accuracy in protein secondary structure prediction”, *Proteins, Vol.* 27, pp. 329-335, 1997.

Garnier J., Osguthorpe DJ. and Robson B., “Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins”, J Mol Biol, Vol. 120, pp. 97-120, 1978.

G. Pollastri, D. Przybylski, B. Rost and P. Baldi, “Three and Eight Classes Using Recurrent Neural Networks and Profiles” (2002)

G. Pollastri, McLysaght A. “*Porter*: a new, accurate server for protein secondary structure prediction”, Bioinformatics, Vol. 21, pp. 1719–1720, 2004.

Haykin S., “Neural Networks: A Comprehensive Foundation”, Second Edition, Prentice Hall, Canada, 1999.

Holland, J. (1992) "Genetic algorithms", Scientific American 267 (1), pp. 66-72.

Kim H, Park H: “**Protein secondary structure prediction based on an improved support vector machines approach”,** *Protein Engineering Design and Selection*, Vol. **16, pp.** 553-560, 2003.

King RD, and Sternberg MJ., “Identification and application of the concepts important for accurate and reliable protein secondary structure prediction”, Protein Sci, Vol. 5(11), pp. 298-310, 1996.

Kountouris P. and Hirst J., “Prediction of backbone dihedral angles and protein secondary structure using support vector machines”, BMC Bioinformatics, Vol. 10, pp. 437, 2009.

LeCun Y., “Generalisation and Network Design Strategies”, Technical Report, University of Toronto Connections Research Group, pp. 7, 1989.

Likothanasis S. “Genetikoi Algorithmoi kai Efarmoges”, Part C, 2001.

Liu Y. and Yao X.*, “Ensemble learning via negative correlation”,* Neural Networks, Vol. 12, pp. 1399–1404, 1999.

McCulloch W.S. and Pitts W. “A Logical Calculus of the Ideas Immanent in Nervous Activity”, Bulletin of Mathematical Biophysics, Vol. 5, pp. 115-133, 1943.

Qian N, Sejnowski TJ, "Predicting the secondary structure of globular proteins using neural network models*", Journal of Molecular Biology*, Vol. 202, pp. 865-884, 1988.

Rojas R. “Neural Networks”, Springer – Verlag, Berlin, 1996.

Rumelhart D. E., Hinton G. E. and Williams R. J. “Learning internal representations by error propagation” *Parallel Distributed Processing*, Vol. 1, pp. 318-362, 1986.

R. Rurbin, S. Eddy, A. Krogh and G. Mitchison (1998). Biological Sequence Analysis, Cambridge: Cambridge University Press.

Rost B. and Sander C. “Prediction of secondary structure at better than 70% accuracy.“ J. Mol. Biol. 232, 584-599 (1993).

Rost B., Sander C. and Schneider, R. “Redefining the goals of protein secondary structure prediction.” J. Mol. Biol. 235, 13-26 (1994).

Rost B. and *Sander* C. “*Improved prediction of protein secondary structure by use of* sequence profiles and neural networks” PNAS, Vol. 90, pp. 7558-7562, 1993.

Rost B. and Sander C. “Combining evolutionary information & neural networks to predict protein secondary structure” Proteins, Vol. 19, pp. 55-72, 1994.

Rost B. and V. A. Eyrich, “EVA: large Scale analysis of secondary structure prediction.” Proteins, vol. 5, pp 192-199, 2001.

Seung-Ik Lee, Joon-Hyun Ahn, and Sung-Bae Cho,” Exploiting Diversity of Neural Ensembles with Speciated Evolution.”, Yonsei University, South Korea

Salamov A. A. and Solovyev V. V., “Prediction of protein secondary structure by combining nearest-neighbor algorithms and multiple sequence alignments” *J. Mol. Biol.* Vol. 247, pp. 11-15, 1995.

Salamov A. A. and Solovyev V. V., “Protein secondary structure prediction using local alignments” *J. Mol. Biol. V*ol. 268, pp. 31-36, 1997.

Schneider R. and Sander C. “The HSSP database of protein structure-sequence alignments”, Nucleic Acids Research, Vol. 24, No. 1, pp. 201–205, 1996.

SOV, April 23, 2010, http://proteinmodel.org/AS2TS/download_area/

Ward JJ, McGuffin LJ, Buxton BF, Jones DT, “Secondary structure prediction with support vector machines”, Bioinformatics, Vol. 19, pp. 1650-1655, 2003.

Warren S. Sarle. “Neural Networks and Statistical Models”, Proceedings of the Nineteenth Annual SAS Users Group International Conference, April, 1994.

Werbos P. J. , “Beyond regression: New tools for prediction and analysis in the behavioral sciences”, Ph.D., Harvard University, Cambridge, MA, 1974.

W.Mount D. “Bioinformatics: Sequence and Genome Analysis”, Second Edition, Cold Spring Harbor Laboratory Press, New York (2004).

Zemla A., Venclovas C., Fidelis K. and Rost B. A modified definition of Sov, a segment-based measure for protein secondary structure prediction assessment. Proteins: Struct. Funct. Genet. 34,220-223 (1999).

Zhi-Hua Zhou , Jianxin Wu and Wei Tang, “Ensembling neural networks: Many could be better than all”, Nanjing University, (2001)

Zoran Vojinovic and Vojislav Kecman, ”Data Assimilation Using Recurrent Radial Basis Function Neural Network Model”, International Symposium on Computational Intelligence for Measurement Systems and Applications, , Switzerland, (2003)

Zhen Zhang and Nan Jing ,”Radial Basis Function Method for Prediction of Protein Secondary Structure”,Seventh International Conference on Machine Learning and Cybernetics, (2008)

Zvelebil M. and Baum J., “Understanding Bioinformatics”, Garland Science, 2008.

Γενική Βιβλιογραφία

Antoniou P. EPL450 notes., Nicosia 2009 – 1010.

Bishop C. M., “Neural Networks for Pattern Recognition”, Oxford UK, 2005.

Christodoulou C. EPL442 notes., Nicosia 2008 – 2009.

Christodoulou C. And Schizas C. EPL444 notes., Nicosia 2009 – 2010.

ΠΑΡΑΡΤΗΜΑ Α

Νευρωνικό Δίκτυο Αμφίδρομης Ανάδρασης

pssp_ucy.cpp

```
//included libraries

#include "DataReader.h"
#include "Neuron.h"
#include "BidirectionalRecurrentNeuralNetwork.h"
#include "OutputData.h"
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>


using namespace std;

int main(int argc, char* argv[])
{
	//initiate random function - the same random values every time
	srand(time(NULL));

	//variables of the main program
	char trainFile[100];
	char testFile[100];
	char inputProfile[100];
	char outputProfile[100];
	char msaFile[100];
	int primaryStructureSize;
	int msaEnable;
	int randomizeDataset;
	float parameters[17];
	int epoch=0;
	bool endOfFile=true;
	vector<double> outputresult1;
	vector<double> outputresult2;
	vector<double> outputresult3;
	vector<double> outputresult4;
	vector<double> outputresult5;
	vector<double> outputresult6;

	vector<vector <double> > vectoroutputresult1;
	vector<vector <double> > vectoroutputresult2;
	vector<vector <double> > vectoroutputresult3;
	vector<vector <double> > vectoroutputresult4;
	vector<vector <double> > vectoroutputresult5;
	vector<vector <double> > vectoroutputresult6;
	vector<vector <double> > vectoroutputresultensemble;


	vector<double> ensambleAverage;
	char secSt;
	double maxVal;
	int maxIndex;
```

```
	double tempadditionprotein;
	vector<double> ensambleResults;

	vector<char> primaryStructure;
	vector<char> secondaryStructure;
	vector<char> predictedSecondaryStructure;
	char proteinID[100];
	time_t rawtime;
	struct tm * timeinfo;
	time ( &rawtime );
	timeinfo = localtime ( &rawtime );
	string tempstr="DataOut_";
	string oFolder=asctime (timeinfo);
	string  oFolder1;
	string  oFolder2;
	string  oFolder3;
	string  oFolder4;
	string  oFolder5;
	char outputFolder[500];
	char scriptcommand[500];
	char scriptcommand2[500];
	char viterbicommand[500];

	int center_window_size;
	char outputfolderName[100];

		for(int i=0;i<500;i++){
					outputfolderName[i]='\0';
					scriptcommand2[i]='\0';
		}
	//object of the DataReader class that reads the program's
parameters from the parameter file
		DataReader *getInput=new DataReader();

	//Parameters' entry
	getInput-
>readParameters(parameters,inputProfile,outputProfile,trainFile,testF
ile,&msaEnable,&randomizeDataset,&center_window_size,outputfolderName
);



	//Create the Outputu Folder
			oFolder1.insert(0,"mkdir ");

	oFolder1.insert(oFolder1.length(),outputfolderName);
			oFolder1.insert(oFolder1.length(),"//");
			for(int i=0;i<oFolder1.length();i++){
				outputFolder[i]=oFolder1[i];
				}
			for(int i=oFolder1.length()-1;i<500;i++){

	outputFolder[i]='\0';
								}

			char temp;

			system(outputFolder);
			oFolder2.insert(0,oFolder1);
			oFolder2.insert(oFolder2.length()-
2,"/ScriptResults//");
```

```
                  for(int i=0;i<oFolder2.length();i++){
                        outputFolder[i]=oFolder2[i];
                  }
                  for(int i=oFolder2.length()-1;i<500;i++){
                        outputFolder[i]='\0';
                        }

                  system(outputFolder);

                  oFolder1.erase(0,6);
                  oFolder2.erase(0,6);
                  oFolder2.erase(oFolder2.length(),6);
                  oFolder3.insert(0,oFolder1);
                  oFolder3.insert(oFolder3.length()-2,"/Z_Output.txt
");

                  oFolder4.insert(0,oFolder1);
                  oFolder4.insert(oFolder4.length()-2,"/Z_Output.txt
"+oFolder1+"/Z_Output.txt >"+oFolder1+"/ScriptResults/viterbi.txt");
                  oFolder4.insert(0,"perl viterbi.pl ");
                  for(int i=0;i<oFolder4.length();i++){
                        viterbicommand[i]=oFolder4[i];
                                                      }
                  for(int i=oFolder4.length()-2;i<500;i++){
                        viterbicommand[i]='\0';
                                                            }


                  oFolder3.insert(oFolder3.length()-2,oFolder2);

                  oFolder3.insert(oFolder3.length()-6,"/results.txt
>"+oFolder2+"info.txt");

                  oFolder3.insert(0,"perl confusionMatricesAndSOV.pl
");

                  for(int i=0;i<oFolder3.length();i++){

      scriptcommand[i]=oFolder3[i];
                                    }
                  for(int i=oFolder3.length()-6;i<500;i++){
                                          scriptcommand[i]='\0';
                                          }


                  for(int i=0;i<500;i++){
                        if(outputfolderName[i]=='\0'){
                              outputfolderName[i]='/';
                              break;
                        }
                                                            }
      OutputData* saveOutputData=new
OutputData('Z',outputfolderName);

    OutputData* OutputDataForFilter=new
OutputData("Z_Filter",outputfolderName);

      saveOutputData->createEnsembleOutputFile();

      for(int i=0;i<3;i++)
```

```
	{
		outputresult1.push_back(0);
		outputresult2.push_back(0);
		outputresult3.push_back(0);
		outputresult4.push_back(0);
		outputresult5.push_back(0);
		outputresult6.push_back(0);
		ensambleAverage.push_back(0);
	}


	//initiate the program's parameter
	int hLayerOneSize = parameters[0];
	int hLayerTwoSize = parameters[1];
	int hLayerOneSizeB = parameters[2];
	int hLayerTwoSizeB = parameters[3];
	int hLayerOneSizeF = parameters[4];
	int hLayerTwoSizeF = parameters[5];
	int activationFuncTypeHidden = parameters[6];
	int activationFuncTypeOutput = parameters[7];
	double initialLearningRate = parameters[8];
	double momentum = parameters[9];
	int windowSize = parameters[10];
	double qMinus1 = parameters[11];
	double qPlus1 = parameters[12];
	int errorFunctionTypeHidden = parameters[13];
	int errorFunctionTypeOutput = parameters[14];
	int s = parameters[15];
	int maxIterations = parameters[16];
	double learningRate=initialLearningRate;


	//creation of the Bidirectional Recurrent Neural Network with
the specific parameters
	//takes the details of the msa file
	BidirectionalRecurrentNeuralNetwork BRNN1 =
BidirectionalRecurrentNeuralNetwork('A',hLayerOneSize,

	hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

	hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

	qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

	inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

	BidirectionalRecurrentNeuralNetwork BRNN2 =
BidirectionalRecurrentNeuralNetwork('B',hLayerOneSize,

	hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

	hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

	qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,
```

```
      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      BidirectionalRecurrentNeuralNetwork BRNN3 =
BidirectionalRecurrentNeuralNetwork('C',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      BidirectionalRecurrentNeuralNetwork BRNN4 =
BidirectionalRecurrentNeuralNetwork('D',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      BidirectionalRecurrentNeuralNetwork BRNN5 =
BidirectionalRecurrentNeuralNetwork('E',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      BidirectionalRecurrentNeuralNetwork BRNN6 =
BidirectionalRecurrentNeuralNetwork('F',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      //the main loop of the program.
```

```
	tempadditionprotein=0.0;//for ensamble per protein
	while(epoch<maxIterations)
	{
		tempadditionprotein=0.0;//for ensamble per protein

		//open pointers to the output files
		BRNN1.openFolders();
		BRNN2.openFolders();
		BRNN3.openFolders();
		BRNN4.openFolders();
		BRNN5.openFolders();
		BRNN6.openFolders();

		//takes the first protein of the training set
		endOfFile=BRNN1.initTrainingInput(&primaryStructureSize);
		BRNN2.initTrainingInput(&primaryStructureSize);
		BRNN3.initTrainingInput(&primaryStructureSize);
		BRNN4.initTrainingInput(&primaryStructureSize);
		BRNN5.initTrainingInput(&primaryStructureSize);
		BRNN6.initTrainingInput(&primaryStructureSize);

		learningRate = initialLearningRate * exp(-
(double)epoch/(double)(108.57));//prosthesi apo ton Antoni

		//
		while(endOfFile)
		{
			//takes the new msa encoding of this protein
			if(msaEnable)
			{
				BRNN1.getMSAInput();
				BRNN2.getMSAInput();
				BRNN3.getMSAInput();
				BRNN4.getMSAInput();
				BRNN5.getMSAInput();
				BRNN6.getMSAInput();
			}

			BRNN1.initProtainQuest();
			BRNN2.initProtainQuest();
			BRNN3.initProtainQuest();
			BRNN4.initProtainQuest();
			BRNN5.initProtainQuest();
			BRNN6.initProtainQuest();


			for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
			{
				//processing
				//cout<<primaryStructureSize<<endl;
				//cout<<sequencet<<endl;

	BRNN1.doFeedForward(sequencet,1,epoch,center_window_size);

BRNN2.doFeedForward(sequencet,1,epoch,center_window_size);

	BRNN3.doFeedForward(sequencet,1,epoch,center_window_size);

	BRNN4.doFeedForward(sequencet,1,epoch,center_window_size);
```

```
	BRNN5.doFeedForward(sequencet,1,epoch,center_window_size);

	BRNN6.doFeedForward(sequencet,1,epoch,center_window_size);

	BRNN1.doBackpropagation(sequencet,learningRate,1);

	BRNN2.doBackpropagation(sequencet,learningRate,1);

	BRNN3.doBackpropagation(sequencet,learningRate,1);

	BRNN4.doBackpropagation(sequencet,learningRate,1);

	BRNN5.doBackpropagation(sequencet,learningRate,1);

	BRNN6.doBackpropagation(sequencet,learningRate,1);
			}

			/*for(int i=1;i<OtH.size();i++){
				cout<<OtH.at(i)<<", ";
			*/

			//pernoume to error gia tin protein pou isoute me
to sinoliko lathos/ton arithmo twn aa pou exei i proteini
			BRNN1.getSequenceTrainingError();
			BRNN2.getSequenceTrainingError();
			BRNN3.getSequenceTrainingError();
			BRNN4.getSequenceTrainingError();
			BRNN5.getSequenceTrainingError();
			BRNN6.getSequenceTrainingError();

			if(epoch==maxIterations-1){

	BRNN1.printOutputSequence(1,vectoroutputresult1,0);

	BRNN2.printOutputSequence(1,vectoroutputresult2,0);

	BRNN3.printOutputSequence(1,vectoroutputresult3,0);

	BRNN4.printOutputSequence(1,vectoroutputresult4,0);

	BRNN5.printOutputSequence(1,vectoroutputresult5,0);

	BRNN6.printOutputSequence(1,vectoroutputresult6,0);
			}


			//takes the next protein from the training file

	endOfFile=BRNN1.initTrainingInput(&primaryStructureSize);
			BRNN2.initTrainingInput(&primaryStructureSize);
			BRNN3.initTrainingInput(&primaryStructureSize);
			BRNN4.initTrainingInput(&primaryStructureSize);
			BRNN5.initTrainingInput(&primaryStructureSize);
			BRNN6.initTrainingInput(&primaryStructureSize);

		}

		endOfFile=BRNN1.initTestInput(&primaryStructureSize);
		BRNN2.initTestInput(&primaryStructureSize);
		BRNN3.initTestInput(&primaryStructureSize);
```

```
		BRNN4.initTestInput(&primaryStructureSize);
		BRNN5.initTestInput(&primaryStructureSize);
		BRNN6.initTestInput(&primaryStructureSize);

		//
		ensambleResults.clear();
		while(endOfFile)
		{

			vectoroutputresult1.clear();
			vectoroutputresult2.clear();
			vectoroutputresult3.clear();
			vectoroutputresult4.clear();
			vectoroutputresult5.clear();
			vectoroutputresult6.clear();
			vectoroutputresultensemble.clear();

			//takes the new msa encoding of this protein
			if(msaEnable)
			{
				BRNN1.getMSAInput();
				BRNN2.getMSAInput();
				BRNN3.getMSAInput();
				BRNN4.getMSAInput();
				BRNN5.getMSAInput();
				BRNN6.getMSAInput();
			}

			BRNN1.initProtainQuest();
			BRNN2.initProtainQuest();
			BRNN3.initProtainQuest();
			BRNN4.initProtainQuest();
			BRNN5.initProtainQuest();
			BRNN6.initProtainQuest();

			predictedSecondaryStructure.clear();


			for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
			{


	BRNN1.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN2.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN3.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN4.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN5.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN6.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN1.retOutput(sequencet,outputresult1,&secSt);

	BRNN2.retOutput(sequencet,outputresult2,&secSt);

	BRNN3.retOutput(sequencet,outputresult3,&secSt);
```

```
	BRNN4.retOutput(sequencet,outputresult4,&secSt);

	BRNN5.retOutput(sequencet,outputresult5,&secSt);

	BRNN6.retOutput(sequencet,outputresult6,&secSt);

                        vectoroutputresult1.push_back(outputresult1);
                        vectoroutputresult2.push_back(outputresult2);
                        vectoroutputresult3.push_back(outputresult3);
                        vectoroutputresult4.push_back(outputresult4);
                        vectoroutputresult5.push_back(outputresult5);
                        vectoroutputresult6.push_back(outputresult6);


                        //calculate the average ensemble output
                        for(int ensambleI=0;ensambleI<3;ensambleI++)
                        {


	ensambleAverage[ensambleI]=(outputresult1[ensambleI]+outputresu
lt2[ensambleI]+outputresult3[ensambleI]+outputresult4[ensambleI]+outp
utresult5[ensambleI]+outputresult6[ensambleI])/6.0;

                        }


	vectoroutputresultensemble.push_back(ensambleAverage);

                        maxVal=ensambleAverage[0];
                        maxIndex=0;

                        for(int position=0;position<3;position++)
                        {

                              if (maxVal<=ensambleAverage[position])
                              {
                                    maxVal=ensambleAverage[position];
                                    maxIndex=position;
                              }
                        }

                        if((maxIndex==0 && secSt=='H') ||
(maxIndex==1 && secSt=='L') || (maxIndex==2 && secSt=='E'))

	tempadditionprotein=tempadditionprotein+1.0;

                        if(maxIndex==0)

	predictedSecondaryStructure.push_back('H');
                        else if(maxIndex==1)

	predictedSecondaryStructure.push_back('L');
                        else if(maxIndex==2)

	predictedSecondaryStructure.push_back('E');



                  }
```

```
                    ensambleResults.push_back(primaryStructureSize);

                    BRNN1.getSequenceTestingError();
                    BRNN2.getSequenceTestingError();
                    BRNN3.getSequenceTestingError();
                    BRNN4.getSequenceTestingError();
                    BRNN5.getSequenceTestingError();
                    BRNN6.getSequenceTestingError();

                    if(epoch==maxIterations-1)
                    {

      BRNN1.printOutputSequence(2,vectoroutputresult1,0);

      BRNN2.printOutputSequence(2,vectoroutputresult2,0);

      BRNN3.printOutputSequence(2,vectoroutputresult3,0);

      BRNN4.printOutputSequence(2,vectoroutputresult4,0);

      BRNN5.printOutputSequence(2,vectoroutputresult5,0);

      BRNN6.printOutputSequence(2,vectoroutputresult6,0);


      BRNN1.returnData(primaryStructure,secondaryStructure,proteinID)
;

                        //create the ensemble output file
                        saveOutputData-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecond
aryStructure,proteinID,0,2,vectoroutputresult1,1);
                OutputDataForFilter-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecond
aryStructure,proteinID,0,2,vectoroutputresultensemble,0);

                saveOutputData-
>printEnsembleOutputFile(primaryStructure,secondaryStructure,predicte
dSecondaryStructure,proteinID,vectoroutputresult1,vectoroutputresult2
,vectoroutputresult3,vectoroutputresult4,vectoroutputresult5,vectorou
tputresult6,vectoroutputresultensemble);

                    }


      endOfFile=BRNN1.initTestInput(&primaryStructureSize);
                    BRNN2.initTestInput(&primaryStructureSize);
                    BRNN3.initTestInput(&primaryStructureSize);
                    BRNN4.initTestInput(&primaryStructureSize);
                    BRNN5.initTestInput(&primaryStructureSize);
                    BRNN6.initTestInput(&primaryStructureSize);
              }


              double count=0;
              for(int i=0;i<ensambleResults.size();i++)
              {
                    count+=ensambleResults[i];
              }
```

```
            cout<<epoch<<"
"<<(tempadditionprotein*100)/count<<"%"<<endl;

            //
            BRNN1.getEpochError();
            BRNN2.getEpochError();
            BRNN3.getEpochError();
            BRNN4.getEpochError();
            BRNN5.getEpochError();
            BRNN6.getEpochError();

            //
            BRNN1.closeFolders();
            BRNN2.closeFolders();
            BRNN3.closeFolders();
            BRNN4.closeFolders();
            BRNN5.closeFolders();
            BRNN6.closeFolders();

            epoch++;
      }


      BRNN1.printOutputs();
      BRNN2.printOutputs();
      BRNN3.printOutputs();
      BRNN4.printOutputs();
      BRNN5.printOutputs();
      BRNN6.printOutputs();
      BRNN1.printNetworkSpecification();
      BRNN2.printNetworkSpecification();
      BRNN3.printNetworkSpecification();
      BRNN4.printNetworkSpecification();
      BRNN5.printNetworkSpecification();
      BRNN6.printNetworkSpecification();

      system(scriptcommand);
      system(viterbicommand);

      saveOutputData->closeEnsembleOutputFile();

      return 0;
}
```

BidirectionalRecurrentNeuralNetwork.h

```
#include "Neuron.h"
#include "DataReader.h"
#include "OutputData.h"
#include "targetver.h"
#include <cstdio>
#include <cstdlib>
#include <cstring>
//#include <fstream>
#include <iostream>
#include <cmath>
#include <cctype>
#include <vector>
#include <time.h>
#include <string>

using namespace std;

#ifndef BidirectionalRecurrentNeuralNetwork_h
#define BidirectionalRecurrentNeuralNetwork_h

//*****************************************************************
*********************
//class BidirectionalRecurrentNeuralNetwork: This class illustrates
all the functions that
//a Bidirectional Recurent Neural Network needs to be trained and
tested
//*****************************************************************
*********************
class BidirectionalRecurrentNeuralNetwork
{

private:

	//members of BidirectionalRecurrentNeuralNetwork class
	vector<string> encodingT;
	vector<string> encodingOutput;
	vector< vector<char> > outputClassification;
	vector<char> currentInput;
	vector<char> primaryStructure;
	vector<char> primaryStructureRead;
	vector<char> secondaryStructure;
	vector<char> secondaryStructureRead;
	vector<char> predictedSecondaryStructure;
	vector<char> predictedSecondaryStructureTrain;
	vector<double> weightsReturn;

	int msa;
	int msaLineLength;
	int inputSizeK;
	int inputSize;
	int halfWindow;
	int halfInputSize;
	int outputSize;
	int sizeOfEachLetter;

	char trainFile[100];
	char testFile[100];
	char inputProfile[100];
```

```
char outputProfile[100];
char proteinName[100];
char proteinID[100];

int hLayerOneSize;
int hLayerTwoSize;
int hLayerOneSizeB;
int hLayerTwoSizeB;
int hLayerOneSizeF;
int hLayerTwoSizeF;
int activationFuncTypeOutput;
int activationFuncTypeHidden;
int windowSize;
int errorFunctionTypeHidden;
int errorFunctionTypeOutput;
int s;
int randomizeTheDataset;
int maxIterations;
int trainingSetAdditionVectorTempSize;
int testSetAdditionVectorTempSize;
int percentageCounter;int percentageCounterTrain;
int prevExcl;

double learningRate;
double momentum;
double qMinus1;
double qPlus1;
double percentage;double percentageTrain;
double trainingSetAddition;
double testSetAddition;
double trainingSetAdditionEpoch;
double testSetAdditionEpoch;
int ss;
char outputLetter;
int tempSizeStart;
int tempSizeEnd;

vector<Neuron> hLayerOne;
vector<Neuron> hLayerTwo;
vector<Neuron> hLayerOneB;
vector<Neuron> hLayerTwoB;
vector<Neuron> hLayerOneF;
vector<Neuron> hLayerTwoF;
vector<Neuron> outputLayer;

vector <int> tempOt;

vector<double> Ot;
vector<double> It;
vector<double> contextFt;
vector<double> contextBt;
vector<double> hLayerOneOutput;
vector<double> hLayerTwoOutput;
vector<double> hLayerOneBOutput;
vector<double> hLayerTwoBOutput;
vector<double> hLayerOneFOutput;
vector<double> hLayerTwoFOutput;
vector<double> inputhLayerOneF;
vector<double> inputhLayerOneB;
vector<double> tempInput;
vector<double> inputOutputLayer;
```

```
        vector<double> targetOutputs;
        vector<double> tempVector;
        vector<double> trainingSetAdditionVector;
        vector<double> trainingSetAdditionVectorTemp;
        vector<double> testSetAdditionVector;
        vector<double> testSetAdditionVectorTemp;
        vector<double> trainingSetAdditionEpochVector;
        vector<double> testSetAdditionEpochVector;
        vector<double> msaInput;

        //objects of input,output to a file
        DataReader* getInputData;
        OutputData* saveOutputData;
          OutputData* OutputData_Filter;
        DataReader* getInput;

        //private method of BidirectionalRecurrentNeuralNetwork class
        void getInitialInput(void);

public:

        BidirectionalRecurrentNeuralNetwork(char id,int
hLayerOneSize,int hLayerTwoSize,int hLayerOneSizeB,int
hLayerTwoSizeB,int hLayerOneSizeF, int hLayerTwoSizeF,int
activationFuncTypeHidden,int activationFuncTypeOutput,double
learningRate,double momentum,int windowSize, double qMinus1,double
qPlus1,int errorFunctionTypeHidden,int errorFunctionTypeOutput,int
s,int epochN,char trainFile[100],char testFile[100], char
inputProfile[100],char outputProfile[100], int msaEnable,int
randomizeDataset,char outputFolder[100]);

        ~BidirectionalRecurrentNeuralNetwork(void);

        //Methods of BidirectionalRecurrentNeuralNetwork class
        bool initTrainingInput(int *primaryStructureSize);
        void retOutput(int seqt,vector<double> &outputresult, char
*secSt);
        bool initTestInput(int *primaryStructureSize);
        void doFeedForward(int sequencet,int isTrainOrTest,int
epoch,int center_window_size);
        void doBackpropagation(int sequencet,double learningRate,int
enable);
        void openFolders(void);
        void closeFolders(void);
        void getSequenceTrainingError();
        void getSequenceTestingError();
        void getEpochError();
        void printOutputs();
        void printOutputSequence(int c, vector<vector <double> >
vectoroutputresult,int flag);
        void getTargetOutputs(int sequencet);
        void printNetworkSpecification();
        void getMSAInput();
        void createMSAtempInput (int p, int q, int size);
        void returnData(vector<char> &primaryStructure,vector<char>
&secondaryStructure,char proteinID[100]);
        void initProtainQuest();

};

#endif
```

BidirectionalRecurrentNeuralNetwork.cpp

```
#include "DataReader.h"
#include "BidirectionalRecurrentNeuralNetwork.h"
#include <cstdio>
#include <cstdlib>
#include <cstring>
//#include <fstream>
#include <iostream>
#include <cmath>
#include <cctype>
#include <vector>
#include <time.h>
#include <string>


//using namespace std;

//*****************************************************************
*********************
//Constructor
BidirectionalRecurrentNeuralNetwork::BidirectionalRecurrentNeuralNetw
ork
//                (int hLayerOneSizeN,int hLayerTwoSizeN,
//                int hLayerOneSizeBN,int hLayerTwoSizeBN,int
hLayerOneSizeFN,int hLayerTwoSizeFN,
//                int activationFuncTypeN,double learningRateN,double
momentumN,int windowSizeN,
//                double qMinus1N,double qPlus1N,int
errorFunctionTypeN,int sN,int epochN,
//                char trainFileN[100],char testFileN[100],char
inputProfileN[100],char outputProfileN[100]):
//                initiates the BidirectionalRecurrentNeuralNetwork
//Parameters:
//                int hLayerOneSizeN                  :hidden layer
one size
//                int hLayerTwoSizeN                  :hidden layer
two size
//                int hLayerOneSizeBN                 :Backward
hidden layer one size
//                int hLayerTwoSizeBN                 :Backward
hidden layer two size
//                int hLayerOneSizeFN                 :Forward hidden
layer one size
//                int hLayerTwoSizeFN                 :Foeward hidden
layer two size
//                int activationFuncTypeN       :number that
corresponds to an activation function
//                double learningRateN          :learning rate
//                double momentumN              :momentum
//                int windowSizeN                     :the window
size for each specific residue
//                double qMinus1N                     :the q operator
for forward recurrent neural network
//                double qPlus1N                      :the q operator
for backward recurrent neural network
//                int errorFunctionTypeN        :number that
corresponds to an error function
//                int sN                                    :the
operator s
```

```
//                int epochN                          :number of
iterations
//                char trainFileN[100]          :the file name of
training set
//                char testFileN[100]                 :the file name
of testing set
//                char inputProfileN[100]       :the file name of
input profile
//                char outputProfileN[100]      :the file name of
output profile
//*****************************************************************
*******************
BidirectionalRecurrentNeuralNetwork::BidirectionalRecurrentNeuralNetw
ork(char id,int hLayerOneSizeN,int hLayerTwoSizeN,
                  int hLayerOneSizeBN,int hLayerTwoSizeBN,int
hLayerOneSizeFN,int hLayerTwoSizeFN,
                  int activationFuncTypeHiddenN,int
activationFuncTypeOutputN,double learningRateN,double momentumN,int
windowSizeN,
                  double qMinus1N,double qPlus1N,int
errorFunctionTypeHiddenN,int errorFunctionTypeOutputN,int sN,int
epochN,char trainFileN[100],char testFileN[100],char
inputProfileN[100],char outputProfileN[100],int msaEnable,int
randomizeDataset,char outputFolder[100])
{
      //private objects of BidirectionalRecurrentNeuralNetwork to
read from files and write to files
      getInputData=new DataReader();

      saveOutputData=new OutputData(id,outputFolder);
        //OutputData_Filter=new OutputData(id,outputFolder);
      getInput=new DataReader();

      //variables of class
      inputSizeK=0;
      inputSize=0;
      halfWindow=0;
      halfInputSize=0;
      outputSize=0;
      percentageCounter=0;percentageCounterTrain=0;
      percentage=0.0;percentageTrain=0.0;
      trainingSetAddition=0;
      testSetAddition=0;
      trainingSetAdditionEpoch=0;
      testSetAddition=0;

      //assign parameters of this class
      hLayerOneSize = hLayerOneSizeN;
      hLayerTwoSize = hLayerTwoSizeN;
      hLayerOneSizeB = hLayerOneSizeBN;
      hLayerTwoSizeB = hLayerTwoSizeBN;
      hLayerOneSizeF = hLayerOneSizeFN;
      hLayerTwoSizeF = hLayerTwoSizeFN;
      activationFuncTypeHidden = activationFuncTypeHiddenN;
      activationFuncTypeOutput = activationFuncTypeOutputN;
      learningRate = learningRateN;
      momentum = momentumN;
      windowSize = windowSizeN;
      msa = msaEnable;

      qMinus1 = (1.0 / qMinus1N);         //change applied by Georgia
```

```
	qPlus1 = qPlus1N;
	errorFunctionTypeHidden = errorFunctionTypeHiddenN;
	errorFunctionTypeOutput = errorFunctionTypeOutputN;
	maxIterations=epochN;
	s = sN;
	randomizeTheDataset = randomizeDataset;

	//assign parameters
	for(int i=0;i<100;i++)
	{
		trainFile[i] = trainFileN[i];
		testFile[i] = testFileN[i];
		inputProfile[i] = inputProfileN[i];
		outputProfile[i] = outputProfileN[i];
		proteinID[i] = '\0';
	}

	//printValuesOfOutputNeurons(-1,0);

	//private method that is called to read info about the input
and output encoding
	BidirectionalRecurrentNeuralNetwork::getInitialInput();

	//create variables for RNNs
	halfWindow=floor((float)windowSize/2);
	halfInputSize=floor((float)inputSize/2); //inputSize is the
residue volume -- almost always = 1


	//create vector for the target output of each simulation
	for(int i=0;i<outputSize;i++)
	{
		targetOutputs.push_back(0);
	}

	//create a vector for the current input of the network
	for(int i=0;i<inputSize;i++)
	{
		currentInput.push_back('\0');
	}

	//create a vector for the current input of the network -- is 21
	for(int i=0;i<inputSizeK;i++)
	{
		It.push_back(0);
	}

	//create the hidden layer one of the network
	for(int i=0;i<hLayerOneSize;i++)
	{

	hLayerOne.push_back(Neuron(inputSizeK,momentum,learningRate,act
ivationFuncTypeHidden,activationFuncTypeOutput,errorFunctionTypeHidde
n,errorFunctionTypeOutput,1));
		hLayerOneOutput.push_back(0);

	}

	//create the hidden layer two of the network
	for(int i=0;i<hLayerTwoSize;i++)
```

```
	{

	hLayerTwo.push_back(Neuron(hLayerOneSize,momentum,learningRate,
activationFuncTypeHidden,activationFuncTypeOutput,errorFunctionTypeHi
dden,errorFunctionTypeOutput,1));
		hLayerTwoOutput.push_back(0);
	}

	//create the contex layer of forward recurrent neural network
	if(hLayerTwoSizeF==0)
	{
		for(int i=0;i<(hLayerOneSizeF*s);i++)
		{
			contextFt.push_back(0);
		}
	}else
	{
		for(int i=0;i<(hLayerTwoSizeF*s);i++)
		{
			contextFt.push_back(0);
		}
	}

	//create the contex layer of backward recurrent neural network
	if(hLayerTwoSizeB==0)
	{
		for(int i=0;i<(hLayerOneSizeB*s);i++)
		{
			contextBt.push_back(0);
		}
	}else
	{
		for(int i=0;i<(hLayerTwoSizeB*s);i++)
		{
			contextBt.push_back(0);
		}
	}

	//create the input vector of hidden layer one
	for(int i=0;i<(inputSizeK+contextFt.size());i++)
	{
		inputhLayerOneF.push_back(0);
	}

	//create the input vector of hidden layer two
	for(int i=0;i<(inputSizeK+contextBt.size());i++)
	{
		inputhLayerOneB.push_back(0);
	}

	//create the hidden layer one of forward recurrent neural
network
	for(int i=0;i<hLayerOneSizeF;i++)
	{

	hLayerOneF.push_back(Neuron(inputSizeK+contextFt.size(),momentu
m,learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,errorFunctionTypeOutput,1));
		hLayerOneFOutput.push_back(0);
	}
```

```
	//create the hidden layer two of forward recurrent neural
network
	for(int i=0;i<hLayerTwoSizeF;i++)
	{

	hLayerTwoF.push_back(Neuron(hLayerOneSizeF,momentum,learningRat
e,activationFuncTypeHidden,activationFuncTypeOutput,errorFunctionType
Hidden,errorFunctionTypeOutput,1));
		hLayerTwoFOutput.push_back(0);
	}

	//create the hidden layer one of backward recurrent neural
network
	for(int i=0;i<hLayerOneSizeB;i++)
	{

	hLayerOneB.push_back(Neuron(inputSizeK+contextBt.size(),momentu
m,learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,errorFunctionTypeOutput,1));
		hLayerOneBOutput.push_back(0);
	}

	//create the hidden layer two of backward recurrent neural
network
	for(int i=0;i<hLayerTwoSizeB;i++)
	{

	hLayerTwoB.push_back(Neuron(hLayerOneSizeB,momentum,learningRat
e,activationFuncTypeHidden,activationFuncTypeOutput,errorFunctionType
Hidden,errorFunctionTypeOutput,1));
		hLayerTwoBOutput.push_back(0);
	}

	//creates the output layer with input and output vectors of
this layer
	//the size of the vector depends from the size and existence of
previous layers
	if(hLayerTwoSize==0 && hLayerTwoSizeF==0 && hLayerTwoSizeB==0)
	{
		for(int i=0;i<outputSize;i++)
		{

	outputLayer.push_back(Neuron(hLayerOneSize+hLayerOneSizeF+hLaye
rOneSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerOneSize+hLayerOneSizeF+hLayerOneSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
	}else if(hLayerTwoSize==0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB!=0)
	{
		for(int i=0;i<outputSize;i++)
		{
```

```
	outputLayer.push_back(Neuron(hLayerOneSize+hLayerOneSizeF+hLaye
rTwoSizeB,momentum,learningRate,activationFuncTypeHidden,activationFu
ncTypeOutput,errorFunctionTypeHidden,errorFunctionTypeOutput,0));
				Ot.push_back(0);
				tempOt.push_back(0);
			}

			for(int
i=0;i<hLayerOneSize+hLayerOneSizeF+hLayerTwoSizeB;i++)
			{
				inputOutputLayer.push_back(0);
			}

	}else if(hLayerTwoSize==0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB==0)
	{
			for(int i=0;i<outputSize;i++)
			{

	outputLayer.push_back(Neuron(hLayerOneSize+hLayerTwoSizeF+hLaye
rOneSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
				Ot.push_back(0);
				tempOt.push_back(0);
			}

			for(int
i=0;i<hLayerOneSize+hLayerTwoSizeF+hLayerOneSizeB;i++)
			{
				inputOutputLayer.push_back(0);
			}
	}else if(hLayerTwoSize==0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB!=0)
	{
			for(int i=0;i<outputSize;i++)
			{

	outputLayer.push_back(Neuron(hLayerOneSize+hLayerTwoSizeF+hLaye
rTwoSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
				Ot.push_back(0);
				tempOt.push_back(0);
			}

			for(int
i=0;i<hLayerOneSize+hLayerTwoSizeF+hLayerTwoSizeB;i++)
			{
				inputOutputLayer.push_back(0);
			}
	}else if(hLayerTwoSize!=0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB==0)
	{
			for(int i=0;i<outputSize;i++)
			{
```

```
	outputLayer.push_back(Neuron(hLayerTwoSize+hLayerOneSizeF+hLaye
rOneSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerTwoSize+hLayerOneSizeF+hLayerOneSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
	}else if(hLayerTwoSize!=0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB!=0)
	{
		for(int i=0;i<outputSize;i++)
		{

	outputLayer.push_back(Neuron(hLayerTwoSize+hLayerOneSizeF+hLaye
rTwoSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerTwoSize+hLayerOneSizeF+hLayerTwoSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
	}else if(hLayerTwoSize!=0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB==0)
	{
		for(int i=0;i<outputSize;i++)
		{

	outputLayer.push_back(Neuron(hLayerTwoSize+hLayerTwoSizeF+hLaye
rOneSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerTwoSize+hLayerTwoSizeF+hLayerOneSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
	}else if(hLayerTwoSize!=0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB!=0)
	{
		for(int i=0;i<outputSize;i++)
		{
```

```
	outputLayer.push_back(Neuron(hLayerTwoSize+hLayerTwoSizeF+hLaye
rTwoSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerTwoSize+hLayerTwoSizeF+hLayerTwoSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
	}
}

//*********************************************************************
*********************
//Deconstructor BidirectionalRecurrentNeuralNetwork(void)
//*********************************************************************
*********************
BidirectionalRecurrentNeuralNetwork::~BidirectionalRecurrentNeuralNet
work(void)
{

}

//executes this once
void BidirectionalRecurrentNeuralNetwork::getInitialInput()
{

	if (msa == 0)
	{	//to msaLineLength den xreiazete otan msa=0, gi afto den
xrisimopiite pouthenaF
		getInput-
>readEncoding(encodingT,&inputSizeK,inputProfile,msa,&msaLineLength);
		inputSize=inputSizeK;
		sizeOfEachLetter=encodingT[0].size();
		inputSizeK=inputSizeK*sizeOfEachLetter;
	}
	else //extra code by Georgia
	{
		//reads the detail lines from the information file
		getInput-
>readEncoding(encodingT,&inputSizeK,inputProfile,msa,&msaLineLength);
		inputSize=inputSizeK;
		sizeOfEachLetter=msaLineLength;
		inputSizeK=inputSizeK*sizeOfEachLetter;
	}


	getInput-
>readOutputEncoding(encodingOutput,outputClassification,&outputSize,o
utputProfile);
}
//
void BidirectionalRecurrentNeuralNetwork::getMSAInput()
{
	//get the whole msa encoding of the protein
```

```
	getInput->readEncodingMSA(encodingT,proteinID);

}

void BidirectionalRecurrentNeuralNetwork::openFolders(void)
{
	getInputData->initiateDataPointers(trainFile,testFile);
}

//
void BidirectionalRecurrentNeuralNetwork::closeFolders(void)
{
	getInputData->closeDataPointers(trainFile,testFile);

	//randomize dataset
	if(randomizeTheDataset==1) {int k = system("perl
randomizeDataset.pl");}
}

bool BidirectionalRecurrentNeuralNetwork::initTrainingInput(int
*primaryStructureSize)
{
	bool endOfFile;

	//the proteinID is the name of the current protein -- reads it
from training set
	endOfFile=getInputData-
>readTrainingInput(primaryStructureRead,secondaryStructureRead,protei
nID);

	primaryStructure.clear();
	secondaryStructure.clear();

	for(int i=0;i<primaryStructureRead.size();i++)
	{
		if(primaryStructureRead[i]!='!')
		{

	primaryStructure.push_back(primaryStructureRead[i]);

	secondaryStructure.push_back(secondaryStructureRead[i]);
		}
	}

	if(endOfFile){

		getInputData-
>translateSecondaryStructure(outputClassification,secondaryStructure)
;
		*primaryStructureSize=primaryStructure.size();

		return endOfFile;
	}
	else { return endOfFile;}
}

bool BidirectionalRecurrentNeuralNetwork::initTestInput(int
*primaryStructureSize)
{
	bool endOfFile;
```

```
	endOfFile=getInputData-
>readTestInput(primaryStructureRead,secondaryStructureRead,proteinID)
;

	primaryStructure.clear();
	secondaryStructure.clear();

	for(int i=0;i<primaryStructureRead.size();i++)
	{
		if(primaryStructureRead[i]!='!')
		{

	primaryStructure.push_back(primaryStructureRead[i]);

	secondaryStructure.push_back(secondaryStructureRead[i]);
		}
	}

	ss=0;
	if(endOfFile){
		getInputData-
>translateSecondaryStructure(outputClassification,secondaryStructure)
;
		*primaryStructureSize=primaryStructure.size();

		return endOfFile;
	}
	else
	{ return endOfFile; }
}
//
void BidirectionalRecurrentNeuralNetwork::createMSAtempInput (int p,
int q, int size)
{
	//msaInput is a double values vector
	msaInput.clear();

	//initialize and split msaInput --encodingT is a vector<string>
	//primaryStructure[p] gives a letter
	msaInput = getInput->splitValues(encodingT[p]);

	for (int i=0;i<size; i++){
	tempInput.push_back(msaInput[i]);

	}
}

void BidirectionalRecurrentNeuralNetwork::initProtainQuest()
{

	prevExcl=0;
	tempSizeStart=0;
	tempSizeEnd=primaryStructure.size();

	for(int i=0;i<primaryStructureRead.size();i++)
	{
		if(primaryStructureRead[i]=='!')
		{
			tempSizeEnd=i;
			break;
		}
```

```
	}

}


void BidirectionalRecurrentNeuralNetwork::doFeedForward(int
sequencet,int isTrainOrTest,int epoch,int center_window_size)
{


	if(sequencet>=tempSizeEnd)
	{

		prevExcl++;

		tempSizeStart=tempSizeEnd;


		tempSizeEnd=primaryStructure.size();
		for(int
i=tempSizeStart+2;i<primaryStructureRead.size();i++)
		{
			if(primaryStructureRead[i]=='!')
			{
				tempSizeEnd=i-prevExcl;
				break;
			}
		}

	}

	//cout<<primaryStructure[sequencet]<<endl;

	if(sequencet==2)
	{
		//cout<<endl;
	}

	//RNNF
	for(int forwardt=sequencet-
halfWindow;forwardt<=sequencet;forwardt++)
	{
		tempInput.clear();

		for(int p=forwardt-halfInputSize;p<(forwardt-
halfInputSize+inputSize);p++)
		{
			if(p<tempSizeStart || p>=tempSizeEnd){
				for(int q=0;q<sizeOfEachLetter;q++)
				{
					tempInput.push_back(0);
				}
			}
			//if to p ine apo 0 mexri to tempsize-1
			else
			{
				for(int q=0;q<sizeOfEachLetter;q++)
				{
					//if(sequencet==185)
					//{
```

```
                                    //    cout<<p<<"
"<<q<<sizeOfEachLetter<<endl;
                                    //}
                                    if (msa == 0)

      tempInput.push_back((double)encodingT[primaryStructure[p]-
65][q]-48);
                                    else{//change by Georgia
                                          //take the input in the sequencet
place -- the p variable express the position
                                          if(sequencet==2)
                                          {
                                                //cout<<p<<" "<<q<<endl;
                                          }

      createMSAtempInput(p,q,sizeOfEachLetter);
                                          break;
                                    }
                              }
                        }
                  }



                  for(int p=0;p<inputhLayerOneF.size();p++)
                  {
                        if(p<contextFt.size())
                              inputhLayerOneF[p]=contextFt[p];
                        else{
                              inputhLayerOneF[p]=tempInput[p-
contextFt.size()];

                        }
                  }

                  for(int p=0;p<hLayerOneF.size();p++)
                  {

      hLayerOneFOutput[p]=hLayerOneF[p].getOutput(inputhLayerOneF);
                  }

                  if(hLayerTwoSizeF==0)
                  {

                        int temp=contextFt.size()-hLayerOneFOutput.size();
                        int newPlace=contextFt.size()/s;
                        for(int p=0;p<temp;p++)
                        {
                              contextFt[p]=contextFt[p+newPlace];
                        }
                        for(int p=temp,i=0;p<contextFt.size();p++,i++)
                        {
                              contextFt[p]=hLayerOneFOutput[i]*qMinus1;
                        }
                  }else
                  {
                        for(int p=0;p<hLayerTwoF.size();p++)
                        {

      hLayerTwoFOutput[p]=hLayerTwoF[p].getOutput(hLayerOneFOutput);
                        }
```

```
                  int temp=contextFt.size()-hLayerTwoFOutput.size();
                  int newPlace=contextFt.size()/s;
                  for(int p=0;p<temp;p++)
                  {
                        contextFt[p]=contextFt[p+newPlace];
                  }
                  for(int p=temp,i=0;p<contextFt.size();p++,i++)
                  {
                        contextFt[p]=hLayerTwoFOutput[i]*qMinus1;
                  }
            }
      }

      for(int i=0;i<contextFt.size();i++)
      {
            contextFt[i]=0;
      }

      It.clear();
      for(int center=sequencet-
center_window_size;center<=sequencet+center_window_size;center++){
      for(int p=center-halfInputSize;p<(sequencet-
halfInputSize+inputSize);p++)
      {
            if(p<0)
            {
                  for(int q=0;q<sizeOfEachLetter;q++)
                  {
                        It.push_back(0);
                  }
            }
            else if(p>=primaryStructure.size())
            {
                  for(int q=0;q<sizeOfEachLetter;q++)
                  {
                        It.push_back(0);
                  }
            }
            else
            {
                  for(int q=0;q<sizeOfEachLetter;q++)
                  {
                        if (msa == 0)

      It.push_back((double)encodingT[primaryStructure[p]-65][q]-48);
                        else{
                              msaInput.clear();
                              //initialize and split msaInput --
encodingT is a vector<string>
                              //primaryStructure[p] gives a letter
                              msaInput = getInput-
>splitValues(encodingT[p]);
                              for (int i=0;i<sizeOfEachLetter; i++){
                                    It.push_back(msaInput[i]);
                              }
                        break;
                        }
                  }
            }
```

```
	}
	}

	//BNNF
	for(int
backwardt=sequencet+halfWindow;backwardt>=sequencet;backwardt--)
	{
		tempInput.clear();
		for(int p=backwardt-halfInputSize;p<(backwardt-
halfInputSize+inputSize);p++)
		{
			if(p<tempSizeStart || p>=tempSizeEnd)
			{
				for(int q=0;q<sizeOfEachLetter;q++)
				{
					tempInput.push_back(0);
				}
			}
			else
			{
				for(int q=0;q<sizeOfEachLetter;q++)
				{
					if (msa == 0)

	tempInput.push_back((double)encodingT[primaryStructure[p]-
65][q]-48);
					else{//change by Georgia
						if(sequencet==268)
						{
							///cout<<p<<" "<<q<<endl;
						}

	createMSAtempInput(p,q,sizeOfEachLetter);
						break;
					}
				}
			}
		}

		for(int p=0;p<inputhLayerOneB.size();p++)
		{
			if(p<tempInput.size())
			{
				inputhLayerOneB[p]=tempInput[p];
			}
			else
			{
				inputhLayerOneB[p]=contextBt[p-
tempInput.size()];
			}
		}

		for(int p=0;p<hLayerOneB.size();p++)
		{

	hLayerOneBOutput[p]=hLayerOneB[p].getOutput(inputhLayerOneB);
		}

		if(hLayerTwoSizeB==0)
		{
			int temp=contextBt.size()-hLayerOneBOutput.size();
```

```
                  int newPlace=contextBt.size()/s;
                  for(int
p=contextBt.size();p>=hLayerOneBOutput.size();p--)
                  {
                        contextBt[p]=contextBt[p-newPlace];
                  }
                  for(int p=0;p<hLayerOneBOutput.size();p++)
                  {
                        contextBt[p]=hLayerOneBOutput[p]*qPlus1;
                  }
            }else
            {
                  for(int p=0;p<hLayerTwoB.size();p++)
                  {

      hLayerTwoBOutput[p]=hLayerTwoB[p].getOutput(hLayerOneBOutput);
                  }

                  int temp=contextBt.size()-hLayerTwoBOutput.size();
                  int newPlace=contextBt.size()/s;
                  for(int p=contextBt.size()-
1;p>=hLayerTwoBOutput.size();p--)
                  {
                        contextBt[p]=contextBt[p-newPlace];
                  }
                  for(int p=0;p<hLayerTwoBOutput.size();p++)
                  {
                        contextBt[p]=hLayerTwoBOutput[p]*qPlus1;
                  }
            }
      }
                        //telos BRNN
      for(int i=0;i<contextBt.size();i++)
      {
            contextBt[i]=0;
      }

      for(int p=0;p<hLayerOne.size();p++)
      {
            hLayerOneOutput[p]=hLayerOne[p].getOutput(It);
      }

      if(hLayerTwoSize!=0)
      {
            for(int p=0;p<hLayerTwo.size();p++)
            {

      hLayerTwoOutput[p]=hLayerTwo[p].getOutput(hLayerOneOutput);
            }
      }


      if(hLayerTwoSize==0 && hLayerTwoSizeF==0 && hLayerTwoSizeB==0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerOneFOutput.size())
                        inputOutputLayer[p]=hLayerOneFOutput[p];
                  else
if(p<(hLayerOneFOutput.size()+hLayerOneOutput.size()))
```

```
                        inputOutputLayer[p]=hLayerOneOutput[p-
hLayerOneFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerOneBOutput[p-
hLayerOneFOutput.size()-hLayerOneOutput.size()];
            }
      }else if(hLayerTwoSize==0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB!=0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerOneFOutput.size())
                        inputOutputLayer[p]=hLayerOneFOutput[p];
                  else
if(p<(hLayerOneFOutput.size()+hLayerOneOutput.size()))
                        inputOutputLayer[p]=hLayerOneOutput[p-
hLayerOneFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerTwoBOutput[p-
hLayerOneFOutput.size()-hLayerOneOutput.size()];
            }
      }else if(hLayerTwoSize==0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB==0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerTwoFOutput.size())
                        inputOutputLayer[p]=hLayerTwoFOutput[p];
                  else
if(p<(hLayerTwoFOutput.size()+hLayerOneOutput.size()))
                        inputOutputLayer[p]=hLayerOneOutput[p-
hLayerTwoFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerOneBOutput[p-
hLayerTwoFOutput.size()-hLayerOneOutput.size()];
                  }
      }else if(hLayerTwoSize==0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB!=0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerTwoFOutput.size())
                        inputOutputLayer[p]=hLayerTwoFOutput[p];
                  else
if(p<(hLayerTwoFOutput.size()+hLayerOneOutput.size()))
                        inputOutputLayer[p]=hLayerOneOutput[p-
hLayerTwoFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerTwoBOutput[p-
hLayerTwoFOutput.size()-hLayerOneOutput.size()];
                  }
      }else if(hLayerTwoSize!=0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB==0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerOneFOutput.size())
                        inputOutputLayer[p]=hLayerOneFOutput[p];
                  else
if(p<(hLayerOneFOutput.size()+hLayerTwoOutput.size()))
```

```
                        inputOutputLayer[p]=hLayerTwoOutput[p-
hLayerOneFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerOneBOutput[p-
hLayerOneFOutput.size()-hLayerTwoOutput.size()];
                  }
      }else if(hLayerTwoSize!=0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB!=0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerOneFOutput.size())
                        inputOutputLayer[p]=hLayerOneFOutput[p];
                  else
if(p<(hLayerOneFOutput.size()+hLayerTwoOutput.size()))
                        inputOutputLayer[p]=hLayerTwoOutput[p-
hLayerOneFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerTwoBOutput[p-
hLayerOneFOutput.size()-hLayerTwoOutput.size()];
                  }
      }else if(hLayerTwoSize!=0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB==0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerTwoFOutput.size())
                        inputOutputLayer[p]=hLayerTwoFOutput[p];
                  else
if(p<(hLayerTwoFOutput.size()+hLayerTwoOutput.size()))
                        inputOutputLayer[p]=hLayerTwoOutput[p-
hLayerTwoFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerOneBOutput[p-
hLayerTwoFOutput.size()-hLayerTwoOutput.size()];
            }
      }else if(hLayerTwoSize!=0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB!=0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerTwoFOutput.size())
                        inputOutputLayer[p]=hLayerTwoFOutput[p];
                  else
if(p<(hLayerTwoFOutput.size()+hLayerTwoOutput.size()))
                        inputOutputLayer[p]=hLayerTwoOutput[p-
hLayerTwoFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerTwoBOutput[p-
hLayerTwoFOutput.size()-hLayerTwoOutput.size()];
            }
      }

      //gia kathe output neuron vazw sto Ot tin real timi tou
      for(int p=0;p<outputLayer.size();p++)
      {
            Ot[p]=outputLayer[p].getOutput(inputOutputLayer);
      }

      //function that takes the target outputs of the output neurons
      getTargetOutputs(sequencet);
```

```
	//1 == is train
	if(isTrainOrTest==1)
	{
		for(int p=0;p<Ot.size();p++)
		{
			trainingSetAddition+=(targetOutputs[p]-
Ot[p])*(targetOutputs[p]-Ot[p]);
		}


		//we want to see what were the real outputs of the
training set
		int i=0;
		int counterEnc=0;
		int x;

		double maxVal=Ot[0];
		int maxIndex =0;

		for(int i=0;i<Ot.size();i++)
		{

			if (maxVal <= Ot[i])
			{
				maxVal = Ot[i];
				maxIndex = i;
			}
		}


		for (int i=0;i<tempOt.size();i++)
		{
			if (i==maxIndex)
			{
				tempOt[i]=1;
			}
			else
			{
				if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
					tempOt[i]=-1;
				else
					tempOt[i]=0;
			}
		}

		for(int i=0;i<encodingOutput.size();i++)
		{
			counterEnc++;
			//if the current position has a letter
			if(encodingOutput[i][0]!='\0')
			{
				int counter;
				int desiredOutput;


	for(counter=0;counter<tempOt.size();counter++)
				{
					desiredOutput =
((int)encodingOutput[i][counter]-48);
```

```
                                    //changes the lower bound if tanh is
used for output neurons
                                    if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
                                    {
                                          if(desiredOutput==0)
                                                desiredOutput = -1;
                                    }

                                    if(desiredOutput!=tempOt[counter])
                                          break;

                              }

                              if(counter==tempOt.size())
                              {
                                    outputLetter=i+65;
                                    break;
                              }
                        }

                  }

                  x=encodingOutput.size();
                  if((int)x == (int)counterEnc)
                        outputLetter='L';

                  if(epoch==this->maxIterations-1){

            predictedSecondaryStructureTrain.push_back(outputLetter);



                  }

            }


            //2 == is test -- no backward
            else if(isTrainOrTest==2)

            {
                  for(int p=0;p<Ot.size();p++)
                  {
                        testSetAddition+=(targetOutputs[p]-
Ot[p])*(targetOutputs[p]-Ot[p]);
                  }

                  int i=0;
                  int counterEnc=0;
                  int x;

                  //step function
                  /*for(int i=0;i<Ot.size();i++)
                  {
                        //if the activation function is tanh
                        if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
                        {
                              if(Ot[i]>=0.0)
```

```
                              tempOt[i]= 1;
                        else
                              tempOt[i]= -1;
                  }
                  else
                  {
                        if(Ot[i]>=0.5)          //kanonika edw ine
0.5 an exw sigmoid/linear
                              tempOt[i]= 1;
                        else
                              tempOt[i]= 0;
                  }
            }*/

            //enable this for "winner takes all" and remove step
function
            double maxVal=Ot[0];
            int maxIndex =0;

            for(int i=0;i<Ot.size();i++)
            {

                  if (maxVal <= Ot[i])
                  {
                        maxVal = Ot[i];
                        maxIndex = i;
                  }
            }


            for (int i=0;i<tempOt.size();i++)
            {
                  if (i==maxIndex)
                  {
                        tempOt[i]=1;
                  }
                  else
                  {
                        if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
                              tempOt[i]=-1;
                        else
                              tempOt[i]=0;
                  }
            }


            for(int i=0;i<encodingOutput.size();i++)
            {
                  counterEnc++;
                  //if the current position has a letter
                  if(encodingOutput[i][0]!='\0')
                  {
                        int counter;
                        int desiredOutput;


      for(counter=0;counter<tempOt.size();counter++)
                        {
                              desiredOutput =
((int)encodingOutput[i][counter]-48);
```

```
                                    //changes the lower bound if tanh is
used for output neurons
                                    if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
                                    {
                                          if(desiredOutput==0)
                                                desiredOutput = -1;
                                    }

                                    if(desiredOutput!=tempOt[counter])
                                          break;

                              }

                              if(counter==tempOt.size())
                              {
                                    outputLetter=i+65;
                                    break;
                              }
                        }

                  }

                  x=encodingOutput.size();
                  if((int)x == (int)counterEnc)
                        //outputLetter='X';
                        outputLetter='L';


                  if(epoch==this->maxIterations-1){

            predictedSecondaryStructure.push_back(outputLetter);
                  }


            }
}

void BidirectionalRecurrentNeuralNetwork::getTargetOutputs(int
sequencet)
{
      //always save 1 or 0 in the tagetOutputs vector -- if
activation function is sigmoid/linear it does not need any other
processing
      for(int p=0;p<Ot.size();p++)
      {

      targetOutputs[p]=encodingOutput[secondaryStructure[sequencet]-
65][p]-48;
      }

      //if the output neuron's activation function is tanh changes
those target values from 1 and 0 to 1 and -1 respectively
      if(activationFuncTypeOutput==2 && errorFunctionTypeOutput==2)
      {
            for(int p=0;p<Ot.size();p++)
            {
                  //change only the 0 to -1
                  if((targetOutputs[p]) == 0 )
                  {
```

```
                        targetOutputs[p] = -1;
                    }
                }
            }
}


void BidirectionalRecurrentNeuralNetwork::getSequenceTrainingError(){

      trainingSetAdditionVector.push_back(sqrt(trainingSetAddition)/(
double)Ot.size()/primaryStructure.size());
      trainingSetAdditionVectorTemp.push_back(sqrt(trainingSetAdditio
n)/(double)Ot.size()/primaryStructure.size());
      trainingSetAdditionVectorTempSize=trainingSetAdditionVectorTemp
.size();
      trainingSetAddition=0;
}

void BidirectionalRecurrentNeuralNetwork::getSequenceTestingError(){

      testSetAdditionVector.push_back(sqrt(testSetAddition)/(double)O
t.size()/primaryStructure.size());
      testSetAdditionVectorTemp.push_back(sqrt(testSetAddition)/(doub
le)Ot.size()/primaryStructure.size());
      testSetAdditionVectorTempSize=testSetAdditionVectorTemp.size();
      testSetAddition=0;
}


void BidirectionalRecurrentNeuralNetwork::getEpochError(){

      double tempValue;

      tempValue=0;

      for(int i=0;i<trainingSetAdditionVectorTempSize;i++)
      {
            tempValue+=trainingSetAdditionVectorTemp[i];
      }

      trainingSetAdditionEpochVector.push_back(tempValue/(double)trai
ningSetAdditionVectorTempSize);

      tempValue=0;

      for(int i=0;i<testSetAdditionVectorTempSize;i++)
      {
            tempValue+=testSetAdditionVectorTemp[i];
      }

      testSetAdditionEpochVector.push_back(tempValue/(double)testSetA
dditionVectorTempSize);

      trainingSetAdditionVectorTemp.clear();
      testSetAdditionVectorTemp.clear();
}


void BidirectionalRecurrentNeuralNetwork::printOutputSequence(int
trainOrTest,vector<vector <double> > vectoroutputresult,int flag)
{
```

```
	double tempPercentage;
	int counter=0;

	if(trainOrTest==2) //testing
	{for(int i=0;i<primaryStructure.size();i++)
	{
		if(secondaryStructure[i]==predictedSecondaryStructure[i])
			counter++;
	}

	tempPercentage=(double)counter*100.0/(double)secondaryStructure
.size();
	saveOutputData-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecond
aryStructure,proteinID,tempPercentage,2,vectoroutputresult,flag);
	percentageCounter++;
	percentage+=tempPercentage;
	predictedSecondaryStructure.clear();}
	else //training
	{

	for(int i=0;i<primaryStructure.size();i++)
	{

	if(secondaryStructure[i]==predictedSecondaryStructureTrain[i])
			counter++;
	}

	tempPercentage=(double)counter*100.0/(double)secondaryStructure
.size();
	saveOutputData-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecond
aryStructureTrain,proteinID,tempPercentage,1,vectoroutputresult,flag)
;
	percentageCounterTrain++;
	percentageTrain+=tempPercentage;
	predictedSecondaryStructureTrain.clear();
	}
}

void BidirectionalRecurrentNeuralNetwork::printNetworkSpecification()
{

	saveOutputData-
>createNetworkFile(hLayerOneSize,hLayerTwoSize,hLayerOneSizeB,hLayerT
woSizeB,hLayerOneSizeF,

	hLayerTwoSizeF,activationFuncTypeHidden,learningRate,momentum,w
indowSize,

	qMinus1,qPlus1,errorFunctionTypeHidden,s,maxIterations,trainFil
e,testFile,

	inputProfile,outputProfile,percentage/(double)percentageCounter
);

}

void BidirectionalRecurrentNeuralNetwork::printOutputs()
{
```

```
	saveOutputData-
>createErrorFiles(trainingSetAdditionEpochVector,testSetAdditionEpoch
Vector,

	trainingSetAdditionVector,testSetAdditionVector); //change by
Georgia - last variable
	saveOutputData->closeAllPointers();
}


void BidirectionalRecurrentNeuralNetwork::doBackpropagation(int
sequencet, double learningRate,int enable)
{

	for(int p=0;p<outputLayer.size();p++)
	{

	outputLayer[p].calculateOutputLayerError(targetOutputs[p]);
		outputLayer[p].calculateErrorPropagation();
		outputLayer[p].adjustDw(learningRate);
	}
//
	if(hLayerTwoSizeF==0 && hLayerTwoSize==0 && hLayerTwoSizeB==0)
	{

		tempVector.clear();
		for(int i=0;i<hLayerOneF.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i);
				}
			}
		}


		for(int i=0;i<hLayerOneF.size();i++){


	hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOneF[i].calculateErrorPropagation();
			hLayerOneF[i].adjustDw(learningRate);
		}




		tempVector.clear();
		for(int i=0;i<hLayerOne.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
);
```

```
                        }
                  }
            }


            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
+hLayerOneSize);
                        }
                  }
            }


            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }


      }else if(hLayerTwoSizeF==0 && hLayerTwoSize==0 &&
hLayerTwoSizeB!=0)
      {

            tempVector.clear();
            for(int i=0;i<hLayerOneF.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                  }
            }


            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
```

```
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }



            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
);
                        }
                  }
            }



            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerTwoB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
+hLayerOneSize);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoB.size();i++){


      hLayerTwoB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoB[i].calculateErrorPropagation();
                  hLayerTwoB[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<hLayerTwoB.size();j++){
                        if(j==0){
```

```
	tempVector.push_back(hLayerTwoB[j].getSpecificError(i));
                        }else{

	tempVector[i]+=hLayerTwoB[j].getSpecificError(i);
                        }
		}
		}

		for(int i=0;i<hLayerOneB.size();i++){


	hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                hLayerOneB[i].calculateErrorPropagation();
                hLayerOneB[i].adjustDw(learningRate);
		}



	}else if(hLayerTwoSizeF==0 && hLayerTwoSize!=0 &&
hLayerTwoSizeB==0)
	{

		tempVector.clear();
		for(int i=0;i<hLayerOneF.size();i++){
                for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

	tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                }
		}


		for(int i=0;i<hLayerOneF.size();i++){


	hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                hLayerOneF[i].calculateErrorPropagation();
                hLayerOneF[i].adjustDw(learningRate);
		}



		tempVector.clear();
		for(int i=0;i<hLayerTwo.size();i++){
                for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF));
                        }else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
);
                        }
```

```
                  }
            }

            for(int i=0;i<hLayerTwo.size();i++){


      hLayerTwo[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwo[i].calculateErrorPropagation();
                  hLayerTwo[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<hLayerTwo.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwo[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwo[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
+hLayerTwoSize);
                        }
                  }
            }


            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }



      }else if(hLayerTwoSizeF==0 && hLayerTwoSize!=0 &&
hLayerTwoSizeB!=0)
```

```
	{

		tempVector.clear();
		for(int i=0;i<hLayerOneF.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i);
				}
			}
		}


		for(int i=0;i<hLayerOneF.size();i++){


	hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOneF[i].calculateErrorPropagation();
			hLayerOneF[i].adjustDw(learningRate);
		}




		tempVector.clear();
		for(int i=0;i<hLayerTwo.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
);
				}
			}
		}

		for(int i=0;i<hLayerTwo.size();i++){


	hLayerTwo[i].calculateHiddenLayerError(tempVector[i]);
			hLayerTwo[i].calculateErrorPropagation();
			hLayerTwo[i].adjustDw(learningRate);
		}

		tempVector.clear();
		for(int i=0;i<hLayerOne.size();i++){
			for(int j=0;j<hLayerTwo.size();j++){
				if(j==0){

	tempVector.push_back(hLayerTwo[j].getSpecificError(i));
				}else{

	tempVector[i]+=hLayerTwo[j].getSpecificError(i);
				}
		}
```

```
            }

            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerTwoB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
+hLayerTwoSize);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoB.size();i++){


      hLayerTwoB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoB[i].calculateErrorPropagation();
                  hLayerTwoB[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<hLayerTwoB.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoB[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoB[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }




      }else if(hLayerTwoSizeF!=0 && hLayerTwoSize==0 &&
hLayerTwoSizeB==0)
```

```
	{

		tempVector.clear();
		for(int i=0;i<hLayerTwoF.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i);
				}
			}
		}

		for(int i=0;i<hLayerTwoF.size();i++){


	hLayerTwoF[i].calculateHiddenLayerError(tempVector[i]);
			hLayerTwoF[i].calculateErrorPropagation();
			hLayerTwoF[i].adjustDw(learningRate);
		}


		tempVector.clear();
		for(int i=0;i<hLayerOneF.size();i++){
			for(int j=0;j<hLayerTwoF.size();j++){
				if(j==0){

	tempVector.push_back(hLayerTwoF[j].getSpecificError(i));
				}else{

	tempVector[i]+=hLayerTwoF[j].getSpecificError(i);
				}
		}
		}

		for(int i=0;i<hLayerOneF.size();i++){


	hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOneF[i].calculateErrorPropagation();
			hLayerOneF[i].adjustDw(learningRate);
		}




		tempVector.clear();
		for(int i=0;i<hLayerOne.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
);
				}
			}
```

```
            }


            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
+hLayerOneSize);
                        }
                  }
            }


            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }







      }else if(hLayerTwoSizeF!=0 && hLayerTwoSize==0 &&
hLayerTwoSizeB!=0)
      {

            tempVector.clear();
            for(int i=0;i<hLayerTwoF.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoF.size();i++){


      hLayerTwoF[i].calculateHiddenLayerError(tempVector[i]);
```

```
                  hLayerTwoF[i].calculateErrorPropagation();
                  hLayerTwoF[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneF.size();i++){
                  for(int j=0;j<hLayerTwoF.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoF[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoF[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
);
                        }
                  }
            }


            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerTwoB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF+hLayerOneSize));
                        }else{
```

```
      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
+hLayerOneSize);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoB.size();i++){


      hLayerTwoB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoB[i].calculateErrorPropagation();
                  hLayerTwoB[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<hLayerTwoB.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoB[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoB[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }



      }else if(hLayerTwoSizeF!=0 && hLayerTwoSize!=0 &&
hLayerTwoSizeB==0)
      {

            tempVector.clear();
            for(int i=0;i<hLayerTwoF.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoF.size();i++){


      hLayerTwoF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoF[i].calculateErrorPropagation();
```

```
                  hLayerTwoF[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneF.size();i++){
                  for(int j=0;j<hLayerTwoF.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoF[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoF[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerTwo.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
);
                        }
                  }
            }

            for(int i=0;i<hLayerTwo.size();i++){


      hLayerTwo[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwo[i].calculateErrorPropagation();
                  hLayerTwo[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<hLayerTwo.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwo[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwo[j].getSpecificError(i);
                        }
```

```
            }
            }

            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
+hLayerTwoSize);
                        }
                  }
            }

            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }




      }else if(hLayerTwoSizeF!=0 && hLayerTwoSize!=0 &&
hLayerTwoSizeB!=0)
      {

            tempVector.clear();
            for(int i=0;i<hLayerTwoF.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoF.size();i++){


      hLayerTwoF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoF[i].calculateErrorPropagation();
                  hLayerTwoF[i].adjustDw(learningRate);
            }
```

```
            tempVector.clear();
            for(int i=0;i<hLayerOneF.size();i++){
                  for(int j=0;j<hLayerTwoF.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoF[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoF[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerTwo.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
);
                        }
                  }
            }

            for(int i=0;i<hLayerTwo.size();i++){


      hLayerTwo[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwo[i].calculateErrorPropagation();
                  hLayerTwo[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<hLayerTwo.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwo[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwo[j].getSpecificError(i);
                        }
            }
            }
```

```
            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerTwoB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
+hLayerTwoSize);
                        }
                  }
            }///

            for(int i=0;i<hLayerTwoB.size();i++){


      hLayerTwoB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoB[i].calculateErrorPropagation();
                  hLayerTwoB[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<hLayerTwoB.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoB[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoB[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }




      }
```

```
}

void BidirectionalRecurrentNeuralNetwork::retOutput(int
seqt,vector<double> &outputresult,char *secSt)
{

	outputresult=Ot;
	*secSt=secondaryStructure[seqt];

}

void BidirectionalRecurrentNeuralNetwork::returnData(vector<char>
&primaryStructureReturn,vector<char> &secondaryStructureReturn,char
proteinIDReturn[100])
{

	primaryStructureReturn.clear();
	secondaryStructureReturn.clear();

	for(int i=0;i<100;i++)
	{
		proteinIDReturn[i]=proteinID[i];
	}


	for(int i=0;i<primaryStructure.size();i++)
	{
		primaryStructureReturn.push_back(primaryStructure[i]);

	secondaryStructureReturn.push_back(secondaryStructure[i]);
	}


}
```

Neuron.h

```
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>

using namespace std;

#ifndef Neuron_h
#define Neuron_h

//*********************************************************************
********************
//class Neuron: This class illustrates all the functions of a neuron
in a Bidirectional
//Recurent Neural Network that is trained by a variation of
Backpropagation Algorithm
//through time
//*********************************************************************
********************
class Neuron
{

private:

	//members of Neuron class
	vector<double> inputVector;
	vector<double> weightsVector;
	vector<double> weightMulError;
	vector<double> previousAdjustment;
	vector<double> Dweights;

	int sizeOfInput;
	int activationFunctionTypeOutput;
	int activationFunctionTypeHidden;
	int errorFunctionTypeHidden;
	int errorFunctionTypeOutput;
	int neuronType;

	double output;
	double error;
	double bias;
	double Dbias;
	double inputNet;
	double momentum;
	double learningRate;
	double biasPreviousAdjustment;
	double slope;
	double amplitude;
	double fmin;
	double fmax;
```

```
        void callActivationFunction(void);
        void calculateInput(void);

public:

        Neuron(int size,double momentumN,double learningRateN,int
functionTypeHiddens,int functionTypeOutput,int errorTypeHidden,int
errorTypeOutput,int neuronType);
        ~Neuron(void);

        //methods of Neuron class
        double getOutput(vector<double> getInput);
        void calculateOutputLayerError(double targetOutput);
        void calculateErrorPropagation();
        void calculateHiddenLayerError(double prevWeightSum);
        void adjustDw(double learningRateNew);
        double getSpecificError(int x);
        void returnWeightVector(vector<double> *weihtsReturn);
        void printTest();

};
#endif
```

Neuron.cpp

```
#include "Neuron.h"
#include "Services.h"
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>

using namespace std;


//*****************************************************************
*********************
//Constructor Neuron(int size,double momentumN,double
learningRateN,int functionType):
//initiates the Neuron
//Parameters:
//          int size                        :Specifies the input size
//          double momentumN        :Specifies the initial momentum
of the neuron
//          double learningRateN    :Specifies the initial learning
rate of the neuron
//          int functionType        Specifies the activation function
of the neuron
//*****************************************************************
*********************
Neuron::Neuron(int size,double momentumN,double learningRateN,int
functionTypeHiddens,int functionTypeOutput,int errorTypeHidden,int
errorTypeOutput,int neuronTypeN)
{

      Services newServices=Services();

      for(int i=0;i<size;i++){
            inputVector.push_back(0);
            weightsVector.push_back(newServices.rand_numbers());
            weightMulError.push_back(0);
            Dweights.push_back(0.0);
            previousAdjustment.push_back(0.0);  //the previous
weights are saved in this variables
      }


      output=0;
      error=0;
      Dbias=0;
      inputNet=0;
      biasPreviousAdjustment=0;

      bias=newServices.rand_numbers();

      sizeOfInput=size;
```

```
	momentum=momentumN;
	learningRate=learningRateN;
	errorFunctionTypeOutput=errorTypeOutput;
	errorFunctionTypeHidden=errorTypeHidden;
	activationFunctionTypeHidden=functionTypeHiddens;
	activationFunctionTypeOutput=functionTypeOutput;

	neuronType=neuronTypeN;

	//parameters for tanh() activation function
	slope = (2.0/3.0);
	amplitude = 1.7159;

	//sigmoid bounds
	fmin = 0.0;
	fmax = 1.0;

}

//*****************************************************************
*********************
//Deconstructor Neuron(void)
//*****************************************************************
*********************
Neuron::~Neuron(void)
{
}

//*****************************************************************
*********************
//void calculateInput(void): private method that is called to
calculate the
//neuron's input. Specifically, it multiplies each input(output of a
neuron
//from the previous layer) with the appropriate  weight and then sum
all the
//results together to calculate the new neuron's input
//*****************************************************************
*********************
void Neuron :: calculateInput(void)
{

	inputNet=0;

	for(int i=0;i<sizeOfInput;i++){
		inputNet+=(inputVector[i]*weightsVector[i]);
	}

	inputNet=(inputNet+(-bias));

}

//*****************************************************************
*********************
//void activationFunction(void): private method that is called to
calculate the
//neuron's output through an activation function
//*****************************************************************
*********************
void Neuron :: callActivationFunction(void)
{
```

```
	//hidden neurons
	if(neuronType==1)
	{
		if(activationFunctionTypeHidden==1) //using sigmoid
		{
			output=(1/(1+pow(2.718281828,-inputNet)));
		}
		else if (activationFunctionTypeHidden==2) //using tanh
		{
			output = amplitude * tanh(inputNet*slope);
		}
		else if (activationFunctionTypeHidden==3) //using linear
		{
			output = inputNet * ((fmax - fmin) + fmin);
		}
	}
	//output neurons (neuronType==0)
	else
	{
		if(activationFunctionTypeOutput==1) //using sigmoid
		{
			output=(1/(1+pow(2.718281828,-inputNet)));
		}
		else if (activationFunctionTypeOutput==2) //using tanh
		{
			output = amplitude * tanh(inputNet*slope);
		}
		else if (activationFunctionTypeOutput==3) //using linear
		{
			output = inputNet * ((fmax - fmin) + fmin);
		}
	}

}

//*********************************************************************
*********************
//double getOutput(vector <double> getInputs): public method that is
called to calculate
//the output of the neuron
//Parameters:
//		vector <double> getInputs	:the outputs of the
previous layer that are given as
//							inputs to this
neuron and are used to calculate the
//					net input
//Return:
//		double output			:the output of the
neuron
//*********************************************************************
*********************
double Neuron :: getOutput(vector<double> getInputs)
{

	inputVector=getInputs;

	calculateInput();

	callActivationFunction();
```

```
	return output;

}

//*****************************************************************
*********************
//void calculateOutputLayerError(double targetOutput): public method
that is called to
//calculate the output error of the neurons of output Layer
//Parameters:
//		double targetOutput		:the specific,target output
of the neuron for each
//									time step
//*****************************************************************
*********************
void Neuron :: calculateOutputLayerError(double targetOutput)
{
	//sigmoid function
	if (errorFunctionTypeOutput==1)
	{
		error = output*(1-output)*(targetOutput-output);
	}
	//tanh function
	else if (errorFunctionTypeOutput==2)
	{
		error = (slope/amplitude) * (targetOutput-output) *
(amplitude-output) * (amplitude+output);
	}
	//linear function
	else if (errorFunctionTypeOutput==3)
	{
		error = 1.0 * (targetOutput-output);
	}
}

//*****************************************************************
*********************
//void calculateErrorPropagation(): public method that is called to
calculate the error
//that must be propagated back to the weights that are connected to
the specific neuron
//by multiply all the weights of the neuron with the epecific error
//*****************************************************************
*********************
void Neuron :: calculateErrorPropagation(){

	for(int i=0;i<weightMulError.size();i++){
		weightMulError[i]= error * (weightsVector[i]);
	}
}

//*****************************************************************
*********************
//void calculateHiddenLayerError(double prevWeightSum): public method
that is called to
//calculate the output error of the neurons of Hidden Layers
//Parameters:
//		double prevWeightSum		:the sum of the neurons'
error that are connectet
//									with the output
of the specific neuron
```

```
//*****************************************************************
*********************
void Neuron :: calculateHiddenLayerError(double prevWeightSum)
{
	//sigmoid function
	if (errorFunctionTypeHidden==1)
	{
		error = output*(1-output)*prevWeightSum;
	}
	//tanh function
	else if (errorFunctionTypeHidden==2)
	{
		error = (slope/amplitude) * (amplitude-output) *
(amplitude+output) * prevWeightSum;
	}
	//linear function
	else if (errorFunctionTypeHidden==3)
	{
		error = 1.0 * prevWeightSum;
	}

}

//*****************************************************************
*********************
//void getSpecificError(int position): public method that is called
to return the
//result of the multiplication of a specific weight with the neurons
error
//Parameters:
//		double position			:the position of the weight
in the weight's vector
//*****************************************************************
*********************
double Neuron :: getSpecificError(int position){

	return weightMulError[position];

}

//*****************************************************************
*********************
//void adjustDw(): public method that is called to calculate the
adjustment of each
//weight and bias of the specific neuron. The bias is adjusted like a
neuron's weight
//*****************************************************************
*********************
void Neuron :: adjustDw(double learningRateNew){

	for(int i=0;i<Dweights.size();i++)
	{

	/*Dweights[i]+=learningRate*inputVector[i]*error+momentum*previ
ousAdjustment[i];

	previousAdjustment[i]=learningRate*inputVector[i]*error+momentu
m*previousAdjustment[i];
		weightsVector[i] = weightsVector[i] + Dweights[i];*/
```

```
            Dweights[i] =
learningRateNew*inputVector[i]*error+momentum*(weightsVector[i]-
previousAdjustment[i]);
            previousAdjustment[i] = weightsVector[i];
            weightsVector[i] = weightsVector[i] + Dweights[i];

            //Dweights[i]=0;
            //previousAdjustment[i]=0;
      }

      Dbias = learningRateNew*1*error+momentum*(bias-
biasPreviousAdjustment);
      biasPreviousAdjustment = bias;
      bias = - bias + Dbias;
      bias = -bias;

}
void Neuron :: printTest(){

      for (int i=0;i<inputVector.size();i++)
      {
            cout<<weightsVector[i]<<" ";
      }cout<<endl;



}
```

DataReader.h

```
#include <fstream>
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>

using namespace std;

#ifndef DataReader_h
#define DataReader_h

//*****************************************************************************
*********************
//class DataReader: This class illustrates all the functions that the
Bidirectional
//Recurent Neural Network needs to read from files for its inputs and
parameters
//*****************************************************************************
*********************
class DataReader
{

private:

	//members of DataReader class
	ifstream inTrainFile;
	ifstream inTestFile;
	ifstream encodingFile;
	int beginning;
	int numberOfResidues;
	ifstream inMsaFile;

public:

	DataReader(void);

	~DataReader(void);

	//methods of DataReader class
	void readParameters(float parameters[17],char
inputProfile[100],char outputProfile[100],
			char trainFile[100],char testFile[100], int*
msaEnable,int* randomizeDataset,int *center_w_size,char
outputFolder[100]);
	void readEncoding(vector<string> &encodingT,int
*inputSizeK,const char inputProfile[100], int msa, int* length);
	void initiateDataPointers(char trainFile[100],char
testFile[100]);
	bool readTrainingInput(vector<char>
&primaryStructure,vector<char> &secondaryStructure,char name[100]);
	bool readTestInput(vector<char> &primaryStructure,vector<char>
&secondaryStructure,char name[100]);
```

```
      void readOutputEncoding(vector<string> &encodingT,vector<
vector<char> > &outputClassification,int *outputSize,
                  const char outputProfile[100]);
      void translateSecondaryStructure(vector< vector<char> >
&outputClassification,vector<char> &secondaryStructure);
      void closeDataPointers(char trainFile[100],char testFile[100]);
      void readEncodingMSA (vector<string> &encodingT,char
proteinID[100]);
      vector<double> splitValues (string stringEncoding);

};
#endif
```

DataReader.cpp

```
#include "DataReader.h"
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>

using namespace std;


//*********************************************************************
*********************
//Constructor DataReader(void):initiates the DataReader
//*********************************************************************
*********************
DataReader::DataReader(void)
{

}

//*********************************************************************
*********************
//Deconstructor DataReader(void)
//*********************************************************************
*********************
DataReader::~DataReader(void)
{

}

//*********************************************************************
*********************
//void readParameters(float parameters[15],char
inputProfile[100],char outputProfile[100],
//                 char trainFile[100],char testFile[100]): public
method that is called to
//get the parameters of the neural network from the parameters.dat
file
//Parameters:
//          float parameters[17]          :this table returns alla
the numeric parameters for
//                                                BRNN. Each
position illustrates a parameter and which are
//                                                depended from
their order in the parameters.dat
//                                                file
//          char inputProfile[100]        :this table returns the
input profile file name
```

```
//          char outputProfile[100]       :this table returns the
output profile file name
//          char trainFile[100]                 :this table returns
the file name which contains the
//                                                    training set
//          char testFile[100]                  :this table returns
the file name which contains the
//                                                    testing set
//*****************************************************************
*******************
//*****************************************************************
*******************
void DataReader::readParameters(float parameters[17],char
inputProfile[100],char outputProfile[100],
                  char trainFile[100],char testFile[100], int*
msaEnable,int* randomizeDataset,int *center_w_size,char
outputFolder[100])
{
      //variables
      char temp[100];
      char state[100];
      int counter=0;

      //opens the DataIn\\parameters.dat file to read the parameters
      ifstream inFile;
      inFile.open("DataIn//parameters.dat");

      if (!inFile) {
            cerr << "Unable to open file parameters.dat";
            int x=0;
            cin>>x;
            exit(1);
      }

      //takes all the network parameters with specific order
      for(int i=0;i<27;i++){
            //reads the general parameters
            if(i<17)
            {
                  inFile >> temp;
                  inFile >> parameters[i];
            }else if(i==18)
            {
                  inFile >> temp;
                  inFile >> inputProfile;
            }else if(i==19)
            {
                  inFile >> temp;
                  inFile >> outputProfile;
            }else if(i==20)
            {
                  inFile >> temp;
                  inFile >> trainFile;
            }else if(i==21)
            {
                  inFile >> temp;
                  inFile >> testFile;
            }else if(i==23)
            {
                  inFile >> temp;
                  inFile >> *msaEnable;
```

```
		}
		else if(i==24)
		{
			inFile >> temp;
			inFile >> *randomizeDataset;
		}else if(i==25)
		{
			inFile >> temp;
			inFile >> *center_w_size;
		}else if(i==26)
		{
			inFile >> temp;
			inFile >> outputFolder;
		}
		else if((i==17)||(i==22))
		{
			inFile >> temp;
		}

	}

	inFile.close();
}


//*****************************************************************
*********************
//void readEncoding(vector<string> &encodingT,int *inputSizeK,const
char inputProfile[100]):
//public method that is called to read the file with input encoding
of a protein's primary
//structure. The file must have a specific layout. The encoding of
each letter is saved in
//a vector to the position that the letter has in the English
alphabet.
//Parameters:
//		vector<string> &encodingT	:returns a vector of
strings that illustrates the encoding
//							of each letter.
The vector has size of 26 strings which
//							portrays the
place of each letter
//		int *inputSizeK				:returns the size of
each input(how many residues)
//		const char inputProfile[100]:the name of input profile
file name
//*****************************************************************
*********************
void DataReader::readEncoding(vector<string> &encodingT,int
*inputSizeK,const char inputProfile[100], int msa, int* length)
{

	//variables
	char temp[100];
	char tempChar;
	int tempK;
	string tempString="\0";

	for(int i=0;i<100;i++)
		temp[i]='\0';
```

```
	//create the path string
	strcat(temp,"EncodingProfiles/");
	strcat(temp,inputProfile);

	//points to the file
	encodingFile.open(temp);

	if (!encodingFile)
	{
		cerr << "Unable to open file "<<inputProfile;
		int x=0;
		cin>>x;
		exit(1);
	}

	if(msa == 0)
	{
	//create the size of those vectors
	for(int i=0;i<26;i++)
	encodingT.push_back(tempString);


		for(int i=0;i<24;i++)
		{
			if(i<2)
			{
				encodingFile >> temp;
			}
			else if(i==2)
			{
				encodingFile >> temp;
				encodingFile >> tempK;
				*inputSizeK = tempK; //residue volume

			}
			else if(i<24)
			{
				encodingFile >> tempChar;
				encodingFile >> tempString;
				//The encoding of each letter is saved in a
vector to the position that
				//the letter has in the English alphabet
				encodingT[tempChar-65]+=tempString;

			}
		}
	}

	//add by Georgia
	/* extra code for the msa.txt file*/
	else
	{
		for(int i=0;i<6;i++)
		{
			if(i<2)
			{
				encodingFile >> temp;
			}
			else if(i==2)
			{
				encodingFile >> temp;
```

```
                        encodingFile >> tempK;
                        *inputSizeK = tempK; //residue volume
                  }
                  else if(i==3)
                  {
                        //the **************
                        encodingFile >> temp;
                  }
                  else if(i==4)
                  {
                        encodingFile >> temp;
                        encodingFile >> numberOfResidues;
                        *length = numberOfResidues;
                  }
                  else if(i==5)
                  {
                        //the **************
                        encodingFile >> temp;
                  }

            }

      }

      //close the pointer
      encodingFile.close();
}

void DataReader::readOutputEncoding(vector<string> &encodingT,vector<
vector<char> > &outputClassification,
                  int *outputSize,const char outputProfile[100])
{

      //pointer to a file
      ifstream inFile;

      //variables
      int tempK;
      int counter;
      char temp[100];
      char state[100];
      char tempChar;
      vector<char> tempCharV;
      string tempString="\0";

      //create the size of those vectors
      for(int i=0;i<26;i++)
            encodingT.push_back(tempString);

      for(int i=0;i<100;i++)
            temp[i]='\0';

      //create the path string
      strcat(temp,"OutputProfiles/");
      strcat(temp,outputProfile);

      //point to the file
      inFile.open(temp);

      if (!inFile) {
            cerr << "Unable to open file "<<outputProfile;
```

```
			int x=0;
			cin>>x;
			exit(1);
		}

		//read alla data from the file with a specific order
		for(int i=0;i<100;i++){
			if(i<2)
			{
				inFile >> temp;
			}else if(i==2)
			{
				inFile >> temp;
				inFile >> tempK;
				*outputSize = tempK;
			}else if(i<4)
			{
				inFile >> temp;
			}else if(i<(4+tempK))
			{
				//reads from the file all the secondary structure's
letters and their class
				tempCharV.clear();
				inFile >> state;
				tempCharV.push_back(state[0]);
				inFile >> state;
				counter=0;

				//remove alla the commas
				while(state[counter]!='\0')
				{
					if(state[counter]!=',')
					{
						tempCharV.push_back(state[counter]);
					}
					counter++;
				}
				//Each position of the vector of vector holds
				//the characteristic letter of the class which is
				//followed by the secondary structure letters of
				//that class.
				outputClassification.push_back(tempCharV);

			}else if(i<(4+tempK+2))
			{
				inFile >> temp;
			}else if(i<(6+tempK+tempK))
			{
				inFile >> tempChar;
				inFile >> tempString;
				//The encoding of each letter is saved in a vector
to the position that
				//the letter has in the English alphabet
				encodingT[tempChar-65]+=tempString;
			}
		}
		inFile.close();
}

//*****************************************************************
*********************
```

```
//void initiateDataPointers(char trainFile[100],char testFile[100]):
//public method that is called to initiate tha pointers to the files
which contain the training
//and testing sets.
//Parameters:
//          char trainFile[100]                         :the file name
of training sets
//          char testFile[100]                          :the file name
of testing sets
//*****************************************************************
*********************
void DataReader::initiateDataPointers(char trainFile[100],char
testFile[100])
{
      //variables
      char temp[100];

      //create the size of those vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      //create the path string
      strcat(temp,"TrainingSets/");
      strcat(temp,trainFile);

      //points to the file
      inTrainFile.open(temp);

      if (!inTrainFile) {
            cerr << "Unable to open file "<<temp;
            int x=0;
            cin>>x;
            exit(1);
      }

      //create the size of those vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      //create the path string
      strcat(temp,"TestSets/");
      strcat(temp,testFile);

      //points to the file
      inTestFile.open(temp);

      if (!inTestFile) {
            cerr << "Unable to open file "<<temp;
            int x=0;
            cin>>x;
            exit(1);
      }
}

//*****************************************************************
*********************
//void closeDataPointers(char trainFile[100],char testFile[100]):
//public method that is called to close tha pointers to the files
which contain the training
//and testing sets.
//Parameters:
```

```
//          char trainFile[100]                         :the file name
of training sets
//          char testFile[100]                          :the file name
of testing sets
//*************************************************************************
*******************
void DataReader::closeDataPointers(char trainFile[100],char
testFile[100])
{

      //variables
      char temp[100];

      //creates the size of those vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      //create the path string
      strcat(temp,"TrainingSets/");
      strcat(temp,trainFile);

      //close the pointer
      inTrainFile.clear();
      inTrainFile.close();

      //create the size of those vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      //creates the path string
      strcat(temp,"TestSets/");
      strcat(temp,testFile);
      //close the pointer
      inTestFile.clear();
      inTestFile.close();
}

//*************************************************************************
*******************
//bool readTrainingInput(vector<char> &primaryStructure,vector<char>
&secondaryStructure):
//public method that is called to read from a file the primary and
secondary structure of
//proteins that included in training sets. the pointers to the files
are initiated with
//initiateDataPointers method and closed with closeDataPointers.The
file must have a
//specific layout.
//Parameters:
//          vector<char> &primaryStructure      :returns the primary
structure of the protein
//          vector<char> &secondaryStructure:returns the secondary
structure of the protein
//Return:
//          true                                        :if there are
more sequences
//          false                                       :if the pointer
points to eof
//*************************************************************************
*******************
```

```
bool DataReader::readTrainingInput(vector<char>
&primaryStructure,vector<char> &secondaryStructure,char
proteinID[100])
{
	//variables
	char name[100];
	char temp[1000];

	primaryStructure.clear();

	inTrainFile >> name;

	for(int i=0;i<100;i++)
	{
		proteinID[i]='\0';
	}

	for(int i=0;i<1000;i++)
	{
		temp[i]='\0';
	}

	strcat(proteinID,name);

	//if end of file returns false
	if(inTrainFile.eof()) { return false;}

	inTrainFile.getline(temp,1000);
	inTrainFile.getline(temp,1000);
	//inTrainFile>> temp;

	//reads all the letters of primary structure until the "."
	//while(temp!='.')
	//{
		//primaryStructure.push_back(temp);
		//inTrainFile >> temp;
	//}

	for(int i=0;(i<1000) && (temp[i]!='\0');i++)
	{
		primaryStructure.push_back(temp[i]);
	}


	for(int i=0;i<1000;i++)
	{
		temp[i]='\0';
	}

	inTrainFile.getline(temp,1000);

	secondaryStructure.clear();

	//inTrainFile >> temp;

	//reads all the letters of secondary structure until the "."
	//while(temp!='.')
	//{
	//	secondaryStructure.push_back(temp);
	//	inTrainFile >> temp;
	//}
```

```
	for(int i=0;(i<1000) && (temp[i]!='\0');i++)
	{
		secondaryStructure.push_back(temp[i]);
	}

	return true;
}

//*****************************************************************
*********************
//bool readTestInput(vector<char> &primaryStructure,vector<char>
&secondaryStructure,
//                 char name[100]):
//public method that is called to read from a file the primary and
secondary structure of
//proteins that included in testing sets. The pointers to the files
are initiated with
//initiateDataPointers method and closed with closeDataPointers.The
file must have a
//specific layout.
//Parameters:
//          vector<char> &primaryStructure      :returns the primary
structure of the protein
//          vector<char> &secondaryStructure:returns the secondary
structure of the protein
//          char name[100]                              :the name of
the protein
//Return:
//          true                                        :if there are
more sequences
//          false                                       :if the pointer
points to eof
//*****************************************************************
*********************
bool DataReader::readTestInput(vector<char>
&primaryStructure,vector<char> &secondaryStructure,char
proteinID[100])
{
	/*
	//variables
	char temp;
	char name[100];

	primaryStructure.clear();

	inTestFile >> name;

	for(int i=0;i<100;i++) {proteinID[i]='\0';}

	strcat(proteinID,name);


	//if end of file returns false
	if(inTestFile.eof()){ return false;}

	inTestFile >> temp;

	//reads all the letters of primary structure until the "."
	while(temp!='.')
	{
```

```
	primaryStructure.push_back(temp);
	inTestFile >> temp;
}

//if there in low case letter replace it with capital case
for(int i=0;i<primaryStructure.size();i++)
{
	if(primaryStructure[i]>='a' && primaryStructure[i]<='z')
		primaryStructure[i]=primaryStructure[i]-32;
}


secondaryStructure.clear();

inTestFile >> temp;

//reads all the letters of secondary structure until the "."
while(temp!='.')
{
	secondaryStructure.push_back(temp);
	inTestFile >> temp;
}

return true;
*/
//variables
char name[100];
char temp[1000];

primaryStructure.clear();

inTestFile >> name;

for(int i=0;i<100;i++)
{
	proteinID[i]='\0';
}

for(int i=0;i<1000;i++)
{
	temp[i]='\0';
}

strcat(proteinID,name);

//if end of file returns false
if(inTestFile.eof()) { return false;}

inTestFile.getline(temp,1000);
inTestFile.getline(temp,1000);
//inTrainFile>> temp;

//reads all the letters of primary structure until the "."
//while(temp!='.')
//{
	//primaryStructure.push_back(temp);
	//inTrainFile >> temp;
//}

for(int i=0;(i<1000) && (temp[i]!='\0');i++)
{
```

```
			primaryStructure.push_back(temp[i]);
		}

		//if there in low case letter replace it with capital case
		for(int i=0;i<primaryStructure.size();i++)
		{
			if(primaryStructure[i]>='a' && primaryStructure[i]<='z')
				primaryStructure[i]=primaryStructure[i]-32;
		}

		for(int i=0;i<1000;i++)
		{
			temp[i]='\0';
		}

		inTestFile.getline(temp,1000);

		secondaryStructure.clear();

		//inTrainFile >> temp;

		//reads all the letters of secondary structure until the "."
		//while(temp!='.')
		//{
		//	secondaryStructure.push_back(temp);
		//	inTrainFile >> temp;
		//}

		for(int i=0;(i<1000) && (temp[i]!='\0');i++)
		{
			secondaryStructure.push_back(temp[i]);
		}

		return true;
}

//*****************************************************************
*********************
//void translateSecondaryStructure(vector<vector<char>>
&outputClassification,
//				vector<char> &secondaryStructure):
//public method that is called to replace all the letters of
secondary structure with
//the specific letter of their class
//Parameters:
//		vector<vector<char>> &outputClassification	:contains
a vector with the output classes of the
//		network. Each position of the vector of vector holds
//		the characteristic letter of the class which is
//		followed by the secondary structure letters of
//		that class
//		vector<char> &secondaryStructure	:contains the
secondary structure of the protein
//*****************************************************************
*********************
void DataReader::translateSecondaryStructure(vector< vector<char> >
&outputClassification,vector<char> &secondaryStructure)
{
		for(int i=0;i<secondaryStructure.size();i++)
		{
			for(int j=0;j<outputClassification.size();j++)
```

```
		{
			for(int k=0;k<outputClassification[j].size();k++)
			{

	if(secondaryStructure[i]==outputClassification[j][k])
				{

	secondaryStructure[i]=outputClassification[j][0];
					break;
				}
			}
		}
	}
}

//Created by Georgia - MSA
//*****************************************************************
*******************
//This function reads the msa dataset for each protein -- use only
with msa
//*****************************************************************
*******************
//encodingT contains the same vector of strings, each string
represents an aminoacid
//*****************************************************************
*******************
void DataReader::readEncodingMSA(vector<string> &encodingT,char
proteinID[100])
{
	//pointer to protein's file
	ifstream inFile;

	//variables
	string tempString="\0";
	string tempLine="\0";
	int aminoacid = 0;
	int digit = 0;
	char temp[100];

	//create the size of those vectors
	for(int i=0;i<100;i++) {temp[i]='\0';}


	//create the path string
	strcat(temp,"EncodingProfiles/msaFiles/");



	char c;
	for(int i=0;i<100;i++)
	{

		proteinID[i]=tolower(proteinID[i]);

	}


	strcat(temp,proteinID);
	//strcat(temp,".txt");
```

```
	strcat(temp,".hssp");

	//tolower(temp);



	//cout<<proteinID<<endl;

	//points to the file
	inFile.open(temp);

	if (!inFile) {
		cerr << "Unable to open file of protein "<<temp;
		int x=0;
		cin>>x;
		exit(1);
	}

	//clear this vector for the next protein
	encodingT.clear();

	//read msa encoding for each protein:

	while ( !inFile.eof() )
	{
		//read the next encoding digit
		inFile >> tempString;

		if(digit != numberOfResidues )
		{
			//store the previous digit
			tempLine=tempLine+tempString+" ";

			digit++;

			if(digit == numberOfResidues)
			{
				digit = 0;
				encodingT.push_back(tempLine);
				tempLine = "\0";

			}

		}
	}

}

//Created by Georgia - MSA
//*****************************************************************
*********************
//This function takes a vector of string values and tranforms it into
a vector of
//double numbers.
//*****************************************************************
*********************
//stringEncoding: to string pou periexei to line
//doubleEncoding: to vector pou tha exei ta values
//*****************************************************************
*********************
```

```
vector<double> DataReader::splitValues (string stringEncoding)
{
	vector<double> doubleEncoding;
	istringstream iss(stringEncoding);

	copy(istream_iterator<double>(iss),
		istream_iterator<double>(),
		 back_inserter<vector<double> >(doubleEncoding));

	for(int i=0;i<doubleEncoding.size();i++)
		doubleEncoding[i]=doubleEncoding[i]/100.0;


	return doubleEncoding;
}
```

OutputData.h

```
#include "targetver.h"

#include <stdio.h>

#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>



using namespace std;
#ifndef OutputData_h
#define OutputData_h

//*****************************************************************************
*********************
//class OutputData: This class illustrates all the functions that the
Bidirectional
//Recurent Neural Network needs to write to the output files
//*****************************************************************************
*********************
class OutputData
{

private:

	//members of OutputData class
	ofstream printError;
	ofstream printErrorP;
	ofstream printOutput;
	ofstream printOutputTrain;
	ofstream printNetwork;
	ofstream printNetworkHTML;
	ofstream printEnsembleResults;
	char folderName[100];
	char HTMLName[100];
	char networkId[100];
	char ensembleResultsName[100];
	int ensembleOutputCounter;

public:

	OutputData(char id,char outputFolder[100]);
	  OutputData(char *id,char outputFolder[100]);
	~OutputData(void);
	char nameOfSimulation[200];

	//methods of OutputData class
	void createErrorFiles(vector<double>
trainingError,vector<double> testingError,
			vector<double>
proteinTrainingError,vector<double> proteinTestingError);
```

```
      void createOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
                        vector<char> predictedSecondaryStructure,char
proteinName[100],double percentage,int c,vector<vector <double> >
vectoroutputresult,int flag);
      void createZOutputFileforFilter(vector<char>
primaryStructure,vector<char> secondaryStructure,
                        vector<char> predictedSecondaryStructure,char
proteinName[100],double percentage,int c,vector<vector <double> >
vectoroutputresult);
      void createNetworkFile(int hLayerOneSizeN,int
hLayerTwoSizeN,int hLayerOneSizeBN,
                        int hLayerTwoSizeBN,int hLayerOneSizeFN,int
hLayerTwoSizeFN,int activationFuncTypeN,
                        double learningRateN,double momentumN,int
windowSizeN,double qMinus1N,
                        double qPlus1N,int errorFunctionTypeN,int
temporaryN,int epochN,char trainFileN[100],
                        char testFileN[100],char
inputProfileN[100],char outputProfileN[100],double percentage);
      void closeOutputPointers(void);
        void closeAllPointers(void);
      void createEnsembleOutputFile(void);
      void closeEnsembleOutputFile(void);
      void printEnsembleOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
                  vector<char> predictedSecondaryStructure,char
proteinName[100], vector<vector <double> > vectoroutputresult1,
                  vector<vector <double> > vectoroutputresult2,
vector<vector <double> > vectoroutputresult3,
                  vector<vector <double> > vectoroutputresult4,
vector<vector <double> > vectoroutputresult5,
                  vector<vector <double> > vectoroutputresult6,
vector<vector <double> > vectoroutputresultensemble);
};
#endif
```

OutpuData.cpp

```
#include "OutputData.h"
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>



using namespace std;

//
//**********************************************************************
*********************
//Constructor DataReader(void):initiates the DataReader
//**********************************************************************
*********************
OutputData::OutputData(char id,char outputFolder[100])
{
	//it takes the time from the system and creates a folder in the
DataOut folder
	//this folder will contain all tha output files of each
simulation
	//Also initiate the pointer to the output file and HTML output
file
	time_t rawtime;
	struct tm * timeinfo;


	networkId[0]=id;

	networkId[1]='\0';

	//cout<<networkId[0]<<endl;

    time ( &rawtime );
	timeinfo = localtime ( &rawtime );



	char temp[200];char temp3[200];

	//cout<<networkId<<endl;

	for(int i=0;i<200;i++)
	{
		temp[i]='\0';
		temp3[i]='\0';

	}

	for(int i=0;i<100;i++)
	{
```

```
		folderName[i]='\0';
		HTMLName[i]='\0';
		nameOfSimulation[i]='\0';
	}

	//cout<<networkId<<endl;

	strcat(temp,outputFolder);
	//strcat(temp,asctime (timeinfo));
	strcat(temp3,outputFolder);
	//strcat(temp3,asctime (timeinfo));
	//strcat(HTMLName,asctime(timeinfo));
	strcat(nameOfSimulation,outputFolder);


	//cout<<networkId<<endl;


	//replace some characters
	/*for(int i=0;i<100;i++){
		if(temp[i]=='\n')
		{
			temp[i]='\0';
		}
		if(temp[i]==' ')
		{
			temp[i]='-';
		}
		if(temp[i]==':')
		{
			temp[i]='_';
		}
	}

	for(int i=0;i<100;i++){
		if(temp3[i]=='\n')
		{
			temp3[i]='\0';
		}
		if(temp3[i]==' ')
		{
			temp3[i]='-';
		}
		if(temp3[i]==':')
		{
			temp3[i]='_';
		}
	}*/

	/*for(int i=0;i<100;i++){
		if(HTMLName[i]=='\n')
		{
			HTMLName[i]='\0';
		}
		if(HTMLName[i]==' ')
		{
			HTMLName[i]='_';;
		}
		if(HTMLName[i]==':')
		{
			HTMLName[i]='_';
```

```
	}
}*/

//create folder
//system("mkdir temp");

//create the string path
//strcat(temp,"\\");
//strcat(temp,asctime (timeinfo));
//strcat(temp3,"\\");
//strcat(temp3,asctime (timeinfo));

/*for(int i=0;i<100;i++)
{
	if(temp[i]=='\n')
	{
		temp[i]='\0';
	}
	if(temp[i]==' ')
	{
		temp[i]='-';
	}
	if(temp[i]==':')
	{
		temp[i]='_';
	}
}

for(int i=0;i<100;i++)
{
	if(temp3[i]=='\n')
	{
		temp3[i]='\0';
	}
	if(temp3[i]==' ')
	{
		temp3[i]='-';
	}
	if(temp3[i]==':')
	{
		temp3[i]='_';
	}
}*/

for(int i=0;i<100;i++)
{
	folderName[i]=temp[i];
}



strcat(temp,networkId);
strcat(temp3,networkId);
strcat(temp,"_Output.txt");
strcat(temp3,"_OutputForTrain.txt");



//initiate pointer of the output file for test
```

```
	printOutput.open(temp);
	printOutputTrain.open(temp3);

	ensembleOutputCounter=0;



}
OutputData::OutputData(char *id,char outputFolder[100])
{

	  strcpy(networkId,id);

	char temp[200];

	//cout<<networkId<<endl;

	for(int i=0;i<200;i++)
	{
		temp[i]='\0';


	}

	for(int i=0;i<100;i++)
	{
		folderName[i]='\0';

	}

	strcat(temp,outputFolder);

	  for(int i=0;i<100;i++)
		{
		folderName[i]=temp[i];
		}

	strcat(temp,networkId);
	strcat(temp,"_Output.txt");
	  printOutput.open(temp);


}

//*********************************************************************
*********************
//Deconstructor DataReader(void)
//*********************************************************************
*********************
OutputData::~OutputData(void)
{

}

//*********************************************************************
*********************
//void createErrorFiles(vector<double> trainingError,vector<double>
testingError,
```

```
//                vector<double> proteinTrainingError,vector<double>
proteinTestingError):
//public method that is called to create the files that contains the
training and testing
//error per protein and per epoch
//Parameters:
//                vector<double> trainingError        :vector with
training errors per epoch
//                vector<double> testingError               :vector
with testing errors per epoch
//                vector<double> proteinTrainingError :vector with
training errors per protein
//                vector<double> proteinTestingError  :vector with
testing errors per protein
//*****************************************************************
*********************
void OutputData::createErrorFiles(vector<double>
trainingError,vector<double> testingError,
                  vector<double> proteinTrainingError,vector<double>
proteinTestingError)
{
      //variables
      char temp[100];
      char temp1[100];

      //create vectors
      for(int i=0;i<100;i++){
            temp[i]='\0';
            temp1[i]='\0';
      }

      //create the names of output files
      for(int i=0;i<100;i++)
      {
            temp[i]=folderName[i];
            temp1[i]=folderName[i];
      }

      //creates the string path and opens the output file with errors
per epoch
      strcat(temp,networkId);


      strcat(temp,"_error_per_epoch.txt");
      printError.open(temp);

      //write all the data to the file
      printError<<"No          Training Error          Test
Error\n";

printError<<"***********************************************\n\n";

      for(int i=0;i<trainingError.size();i++)
      {

      printError<<i<<".\t\t"<<trainingError[i]<<"\t\t"<<testingError[
i]<<endl;
      }

      //creates the string path and opens the output file with errors
per protein
```

```
	strcat(temp1,networkId);
	strcat(temp1,"_error_per_protein.txt");
	printErrorP.open(temp1);

	//write all the data to the file
	printErrorP<<"No		Training Error		Test
Error\n";

	printErrorP<<"*********************************************
\n\n";

	if(proteinTrainingError.size()<=20000)
	 for(int i=0;i<proteinTrainingError.size();i++)
	 {

	printErrorP<<i<<".\t\t"<<proteinTrainingError[i]<<"\t\t"<<prote
inTestingError[i]<<endl;

	 }
}

//*****************************************************************
*******************
//void OutputData::createOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
//		vector<char> predictedSecondaryStructure,char
proteinName[100],double percentage)):
//public method that is called  at the end of each epoch to write in
the output file the name of
//a protein, its primary structure, its secondary structure, its
predicted secondary structure and
//the percentage of succes for the specific simulation
//Parameters:
//		vector<char> primaryStructure
	:protein's primary structure
//		vector<char> secondaryStructure
	:protein's secondary structure
//		vector<char> predictedSecondaryStructure
	:protein's predicted secondary structure
//		char proteinName[100]
	:protein's name
//		double percentage
	:persentage of succes for a simulation
//*****************************************************************
*******************
void OutputData::createOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
		vector<char> predictedSecondaryStructure,char
proteinName[100],double percentage,int trainOrTest,
			vector<vector <double> >
vectoroutputresult,int flag)
{
	if(trainOrTest==2)
	{
	//writes the name of a protein and the percentage of success
	printOutput<<proteinName<<" Correctness
Percentage:"<<percentage<<"%"<<endl;
	//cout<<proteinName<<endl;
	//writes the primary structure
	printOutput<<"primaryStructure		:";
	for(int i=0;i<primaryStructure.size();i++)
```

```
	{
		printOutput<<primaryStructure[i];
	}
	printOutput<<endl;

	//writes the secondary structure
	printOutput<<"secondaryStructure         :";
	for(int i=0;i<secondaryStructure.size();i++)
	{
		printOutput<<secondaryStructure[i];
	}
	printOutput<<endl;

	//writes the predicted secondary structure
	printOutput<<"predictedSecondaryStructure:";
	for(int i=0;i<predictedSecondaryStructure.size();i++)
	{
		printOutput<<predictedSecondaryStructure[i];
	}
	printOutput<<endl;

	  //writes the
	//printOutput<<"predictedSecondaryStructure:";
	  if(flag==0){
	   printOutput<<"Output Value of:"<<endl;
	  for(int i=0;i<vectoroutputresult.at(1).size();i++){

		if(i==0)
		    printOutput<<"H\t";
		else if(i==1)
		    printOutput<<"L\t";
		else if(i==2)
		    printOutput<<"E\t";
	for(int k=0;k<vectoroutputresult.size();k++)
	{
		    printOutput.precision(4);
		printOutput<<vectoroutputresult.at(k).at(i);
		    if(k!=vectoroutputresult.size()-1)
		        printOutput<<",";
	}
	  printOutput<<endl;
	  }
	  }
	  /*else
	printOutput<<endl;*/
	}
	else
	{
	//writes the name of a protein and the percentage of success
	printOutputTrain<<proteinName<<" Correctness
Percentage:"<<percentage<<"%"<<endl;
	//cout<<proteinName<<endl;
	//writes the primary structure
	printOutputTrain<<"primaryStructure           :";
	for(int i=0;i<primaryStructure.size();i++)
	{
		printOutputTrain<<primaryStructure[i];
	}
	printOutputTrain<<endl;

	//writes the secondary structure
```

```
	printOutputTrain<<"secondaryStructure         :";
	for(int i=0;i<secondaryStructure.size();i++)
	{
		printOutputTrain<<secondaryStructure[i];
	}
	printOutputTrain<<endl;

	//writes the predicted secondary structure
	printOutputTrain<<"predictedSecondaryStructure:";
	for(int i=0;i<predictedSecondaryStructure.size();i++)
	{
		printOutputTrain<<predictedSecondaryStructure[i];
	}
	printOutputTrain<<endl;
	  /*for(int i=1;i<OtH.size();i++)
	{
		printOutput<<OtH.at(i)<<",";
	}
	printOutput<<endl;*/
	}
}




//*****************************************************************
*********************
//void createNetworkFile(int hLayerOneSizeN,int hLayerTwoSizeN,int
hLayerOneSizeBN,
//                int hLayerTwoSizeBN,int hLayerOneSizeFN, int
hLayerTwoSizeFN,int activationFuncTypeN,
//                double learningRateN,double momentumN,int
windowSizeN,double qMinus1N,double qPlus1N,
//                int errorFunctionTypeN,int temporaryN,int
epochN,char trainFileN[100],char testFileN[100],
//                char inputProfileN[100],char
outputProfileN[100],double percentage):
//public method that is called to create a file with the parameters
of the network and an HTML file
//with information about the simulation
//Parameters:
//                int hLayerOneSizeN                  :hidden layer
one size
//                int hLayerTwoSizeN                  :hidden layer
two size
//                int hLayerOneSizeBN                 :Backward
hidden layer one size
//                int hLayerTwoSizeBN                 :Backward
hidden layer two size
//                int hLayerOneSizeFN                 :Forward hidden
layer one size
//                int hLayerTwoSizeFN                 :Foeward hidden
layer two size
//                int activationFuncTypeN       :number that
corresponds to an activation function
//                double learningRateN          :learning rate
//                double momentumN              :momentum
//                int windowSizeN                     :the window
size for each specific residue
```

```
//                double qMinus1N                     :the q operator
for forward recurrent neural network
//                double qPlus1N                      :the q operator
for backward recurrent neural network
//                int errorFunctionTypeN        :number that
corresponds to an error function
//                int sN                                    :the
operator s
//                int epochN                          :number of
iterations
//                char trainFileN[100]          :the file name of
training set
//                char testFileN[100]                 :the file name
of testing set
//                char inputProfileN[100]       :the file name of
input profile
//                char outputProfileN[100]      :the file name of
output profile
//                double percentage             :the percentage of
succes
//*****************************************************************
*******************
void OutputData::createNetworkFile(int hLayerOneSizeN,int
hLayerTwoSizeN,int hLayerOneSizeBN,
                  int hLayerTwoSizeBN,int hLayerOneSizeFN, int
hLayerTwoSizeFN,int activationFuncTypeN,
                  double learningRateN,double momentumN,int
windowSizeN,double qMinus1N,double qPlus1N,
                  int errorFunctionTypeN,int sN,int epochN,char
trainFileN[100],char testFileN[100],
                  char inputProfileN[100],char
outputProfileN[100],double percentage)
{

      //variables
      char temp[100];
      char temp1[100];

      //create the vectors
      for(int i=0;i<100;i++)
      {
            temp[i]='\0';
      }

      for(int i=0;i<100;i++)
      {
            temp1[i]='\0';
      }

      for(int i=0;i<100;i++)
      {
            temp[i]=folderName[i];
            temp1[i]=folderName[i];

      }

      //creates the string path and opens the output file with
parameters
      strcat(temp,networkId);
      strcat(temp,"_Network_Specifications.txt");
      printNetwork.open(temp);
```

```
	//creates the string path and opens the HTML file with results
	//strcat(temp1,"DataOut//");
	strcat(temp1,HTMLName);
	strcat(temp1,networkId);
	strcat(temp1,"_1.html");
	printNetworkHTML.open(temp1);

	//writes the information to the output file with parameters
	printNetwork<<"Network Specification"<<endl;
	printNetwork<<"**********************"<<endl;
	printNetwork<<"Size of Hidden Layer 1:"<<hLayerOneSizeN<<endl;
	printNetwork<<"Size of Hidden Layer 2:"<<hLayerTwoSizeN<<endl;
	printNetwork<<"Size of Forward Hidden Layer
1:"<<hLayerOneSizeFN<<endl;
	printNetwork<<"Size of Forward Hidden Layer
2:"<<hLayerTwoSizeFN<<endl;
	printNetwork<<"Size of Backward Hidden Layer
1:"<<hLayerOneSizeBN<<endl;
	printNetwork<<"Size of Backward Hidden Layer
2:"<<hLayerTwoSizeBN<<endl;
	printNetwork<<"Type of Activation Function (Hidden
Neurons):"<<activationFuncTypeN<<endl;
	printNetwork<<"Learning Rate:"<<learningRateN<<endl;
	printNetwork<<"Momentum:"<<momentumN<<endl;
	printNetwork<<"Window Size:"<<windowSizeN<<endl;
	printNetwork<<"q-1:"<<qMinus1N<<endl;
	printNetwork<<"q+1:"<<qPlus1N<<endl;
	printNetwork<<"Type of error Function (Hidden
Neurons):"<<errorFunctionTypeN<<endl;
	printNetwork<<"s:"<<sN<<endl;
	printNetwork<<"Number of Iterrations: "<<epochN<<endl;
	printNetwork<<"Name of Training File: "<<trainFileN<<endl;
	printNetwork<<"Name of Testing File: "<<testFileN<<endl;
	printNetwork<<"Name of Input Profile: "<<inputProfileN<<endl;
	printNetwork<<"Name of Output Profile: "<<outputProfileN<<endl;

	//writes the information to the HTML file with the simulation
results
	printNetworkHTML<<"<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML
1.0 Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd\">"<<endl;
	printNetworkHTML<<"<html
xmlns=\"http://www.w3.org/1999/xhtml\">"<<endl;
	printNetworkHTML<<"<head>"<<endl;
	printNetworkHTML<<"<title>Website Title</title>"<<endl;
	printNetworkHTML<<"<meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\" />"<<endl;
	printNetworkHTML<<"<link rel=\"stylesheet\" type=\"text/css\"
href=\"style.css\" media=\"screen\" />"<<endl;
	printNetworkHTML<<"</head>"<<endl;
	printNetworkHTML<<"<body>"<<endl;
	printNetworkHTML<<"<div id=\"content\">"<<endl;
	printNetworkHTML<<"<div id=\"header\">"<<endl;
	printNetworkHTML<<"<h1><a href=\"#\">Protein Secondary
Structure Prediction </a></h1>"<<endl;
	printNetworkHTML<<"<h2>University of Cyprus </h2>"<<endl;
	printNetworkHTML<<"</div>"<<endl;
	printNetworkHTML<<"<div id=\"navigation\">"<<endl;
	printNetworkHTML<<"<ul>"<<endl;
	printNetworkHTML<<"<li><a href=\"#\">Report</a></li>"<<endl;
```

```
	printNetworkHTML<<"<li><a href=\""<< HTMLName
<<"_2.html\">Output Files</a></li>"<<endl;
	printNetworkHTML<<"</ul>"<<endl;
	printNetworkHTML<<"</div>"<<endl;
	printNetworkHTML<<"<div class=\"right\">"<<endl;
	printNetworkHTML<<"<h2>Report of: "<< HTMLName <<"</h2>"<<endl;
	//printNetworkHTML<<"<p>This website illustrates the
Bidirectional Recurrent Neural Network specifications for Protein
Secondary Structure Prediction. The success prediction of this
experiment is: "<<percentage<<"% </a>.</p>"<<endl;
	//printNetworkHTML<<"</div>"<<endl;
	printNetworkHTML<<"<div class=\"right\">"<<endl;
	printNetworkHTML<<"<h2>Network's Specifications: </h2>"<<endl;
	printNetworkHTML<<"<p>Size of Hidden Layer 1: "<<
hLayerOneSizeN<<"</p>"<<endl;
	printNetworkHTML<<"<p>Size of Hidden Layer 2: "<<hLayerTwoSizeN
<<" </p>"<<endl;
	printNetworkHTML<<"<p>Size of Forward Hidden Layer 1:
"<<hLayerOneSizeFN <<" </p>"<<endl;
	printNetworkHTML<<"<p>Size of Forward Hidden Layer 2:
"<<hLayerTwoSizeFN <<" </p>"<<endl;
	printNetworkHTML<<"<p>Size of Backward Hidden Layer 1:
"<<hLayerOneSizeBN <<" </p>"<<endl;
	printNetworkHTML<<"<p>Size of Backward Hidden Layer 2: "<<
hLayerTwoSizeBN<<" </p>"<<endl;
	printNetworkHTML<<"<p>Type of Activation Function (Hidden
Neurons): "<<activationFuncTypeN <<" </p>"<<endl;
	printNetworkHTML<<"<p>Learning Rate: "<<learningRateN <<"
</p>"<<endl;
	printNetworkHTML<<"<p>Momentum: "<<momentumN <<" </p>"<<endl;
	printNetworkHTML<<"<p>Window Size: "<<windowSizeN <<"
</p>"<<endl;
	printNetworkHTML<<"<p>q-1: "<< qMinus1N<<" </p>"<<endl;
	printNetworkHTML<<"<p>q+1: "<<qPlus1N <<" </p>"<<endl;
	printNetworkHTML<<"<p>Type of error Function (Hidden Neurons):
"<<errorFunctionTypeN <<" </p>"<<endl;
	printNetworkHTML<<"<p>s: "<<sN <<" </p>"<<endl;
	printNetworkHTML<<"<p>Number of Iterrations: "<<epochN <<"
</p>"<<endl;
	printNetworkHTML<<"<p>Name of Training File: "<< trainFileN<<"
</p>"<<endl;
	printNetworkHTML<<"<p>Name of Testing File: "<< testFileN<<"
</p>"<<endl;
	printNetworkHTML<<"<p>Name of Input Profile: "<<inputProfileN
<<" </p>"<<endl;
	printNetworkHTML<<"<p>Name of Output Profile: "<<outputProfileN
<<" </p>"<<endl;
	printNetworkHTML<<"<h2> </h2>"<<endl;
	printNetworkHTML<<"<a href=\"#\" title=\"Link Title\"><img
src=\"pic1.jpg\" alt=\"Something\" width=\"695\" height=\"367\"
style=\"border: 3px solid #ddd;\" /></a>"<<endl;
	printNetworkHTML<<"</div>"<<endl;
	printNetworkHTML<<"<div style=\"clear: both;\"> </div>"<<endl;
	printNetworkHTML<<"<div id=\"footer\">"<<endl;
	printNetworkHTML<<"&copy; Copyright by <a href=\"#\">University
of Cyprus</a> | Designed by <a href=\"#\">Michalis
Agathocleous</a></a>"<<endl;
	printNetworkHTML<<"</div>"<<endl;
	printNetworkHTML<<"</div>"<<endl;
	printNetworkHTML<<"</body>"<<endl;
	printNetworkHTML<<"</html>"<<endl;
```

```
	//close the pointers
	printNetwork.close();
	printNetworkHTML.close();

}

//*********************************************************************
*********************
//void closeAllPointers(): public method that is called to close all
the pointers to
//specific files
//*********************************************************************
*********************
void OutputData::closeAllPointers(){

	printError.close();
	printErrorP.close();
	printOutput.close();
}


void OutputData::closeOutputPointers(){
    printOutput.close();
}


void OutputData::createEnsembleOutputFile(){

	printEnsembleResults.open("ensembleResults.txt");

	printEnsembleResults<<"ID\t\tpdbCode\t\tchain\t\tAA\t\tObsSS\t\
tPredSS\t\tH\t\tE\t\tL\t\tBRNN1_H\t\tBRNN1_E\t\tBRNN1_L\t\tBRNN2_H\t\
tBRNN2_E\t\tBRNN2_L\t\tBRNN3_H\t\tBRNN3_E\t\tBRNN3_L\t\tBRNN4_H\t\tBR
NN4_E\t\tBRNN4_L\t\tBRNN5_H\t\tBRNN5_E\t\tBRNN5_L\t\tBRNN6_H\t\tBRNN6
_E\t\tBRNN6_L"<<endl;
	printEnsembleResults<<"*************************************
*********************************************************************
*********************************************************************
*********************************************************************
************"<<endl;

}



void OutputData::closeEnsembleOutputFile(){

	printEnsembleResults.close();

}


void OutputData::printEnsembleOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
		vector<char> predictedSecondaryStructure,char
proteinName[100], vector<vector <double> > vectoroutputresult1,
		vector<vector <double> > vectoroutputresult2,
vector<vector <double> > vectoroutputresult3,
		vector<vector <double> > vectoroutputresult4,
vector<vector <double> > vectoroutputresult5,
```

```
            vector<vector <double> > vectoroutputresult6,
vector<vector <double> > vectoroutputresultensemble){



      //cout<<primaryStructure.size()<<"
"<<predictedSecondaryStructure.size()<<endl;
      for(int i=0;i<primaryStructure.size();i++)
      {
            ensembleOutputCounter++;

      printEnsembleResults<<ensembleOutputCounter<<"\t\t"<<proteinNam
e[0]<<proteinName[1]<<proteinName[2]<<proteinName[3]<<"\t\t"<<protein
Name[4]<<"\t\t"<<primaryStructure[i]<<"\t\t"<<secondaryStructure[i]<<
"\t\t"<<predictedSecondaryStructure[i]<<"\t\t"<<vectoroutputresultens
emble[i][0]<<"\t\t"<<vectoroutputresultensemble[i][2]<<"\t\t"<<vector
outputresultensemble[i][1]<<"\t\t"<<vectoroutputresult1[i][0]<<"\t\t"
<<vectoroutputresult1[i][2]<<"\t\t"<<vectoroutputresult1[i][1]<<vecto
routputresult2[i][0]<<"\t\t"<<vectoroutputresult2[i][2]<<"\t\t"<<vect
oroutputresult2[i][1]<<vectoroutputresult3[i][0]<<"\t\t"<<vectoroutpu
tresult3[i][2]<<"\t\t"<<vectoroutputresult3[i][1]<<vectoroutputresult
4[i][0]<<"\t\t"<<vectoroutputresult4[i][2]<<"\t\t"<<vectoroutputresul
t4[i][1]<<vectoroutputresult5[i][0]<<"\t\t"<<vectoroutputresult5[i][2
]<<"\t\t"<<vectoroutputresult5[i][1]<<vectoroutputresult6[i][0]<<"\t\
t"<<vectoroutputresult6[i][2]<<"\t\t"<<vectoroutputresult6[i][1]<<end
l;
      }

}
```

ΠΑΡΑΡΤΗΜΑ Β

Αρχείο Παραμέτρων Αρχικοποίησης

Hidden_layer_one_size 15

Hidden_layer_two_size 0

Hidden_layer_one_of_Backward_size 17

Hidden_layer_two_of_Backward_size 0

Hidden_layer_one_of_Forward_size 17

Hidden_layer_two_of_Forward_size 0

Activation_Function_Type_Hidden 1

Activation_Function_Type_Output 1

Learning_Rate 0.09

Momentum 0.5

Window_size 31

q_minus_one 0.6

q_plus_one 0.6

Error_Function_Type_Hidden 1

Error_Function_Type_Output 1

s 3

Maximum_Iterations 1

/**********InputFiles**********/

input_Profile msa.txt

output_Profile threeClasses.txt

train_File trainSet0

test_File testSet0

/*********OtherParams*********/

msa_Enable 1

randomizeDataset 1

center_window_size 12

nameofOutPutFile testingName

ΠΑΡΑΡΤΗΜΑ Γ

Αρχείο Δεδομένων Εισόδου

1bujA

AVINTFDGVADYLIRYKRLPDNYITKSQASALGWVASKGNLAEVAPGKSIGGDVFSNREGRLPSASGRTW
READINYVSGFRNADRLVYSSDWLIYKTTDHYATFTRIR.

LLLLSHHHHHHHHHHHSSLLTTEELHHHHHHHHTLLSSSSLHHHHSTTLEEEEEELLLLSSSSLLSSSLLEEEEE
SSLSSSSLLLSEEEEETTLLEEEESSSSSSLEELL.

2el8A

GSSGSSGEVLAKEEARRALETPSCFLKVSRLEAQLLLERYPECGNLLLRPSGDGADGVSVTTRQMHNGTH
VVRHYKVKREGPKYVIDVEQPFSCTSLDAVVNYFVSHTKKALVPFLLD.

LLLLLLLLLLLSLLLLLLSSSLTTLLLLLHHHHHHHHHHSSTTLSBEEEELLSSSSSEEEEELLLBTTBLLLEEELBLL
BTTBLLBSSSSLLLLSSHHHHHHHHHHHSSSLLLBLLLL.

1co4A

MVVINGVKYACDSCIKSHKAAQCEHNDRPLKILKPRGRPPTT.

LEEETTEEEEETTTTTTSGGGGLLLLSSLEEEELSLLLLSLL.

1edsA

TTLYTSLHGYFVFGPTGCNLEGFFATLGGEI.

LLLSSTTSSLGGGLSSTTLIIIIIIIIIILL.

2byeA

GEERKCLQTHRVTVHGVPGPEPFTVFTINGGTKAKQLLQQILTNEQDIKPVTTDYFLMEEKYFISKEKNEC
RKQPFQRAIGPEEEIMQILSSWFPEEGYMGRIVLKTQQE.

LLLSSLLLLLEEEESSLSSSSSSEEEELLTTLLHHHHHHHHHHLLSSSSLLLLLSEEEEEELLLLSSSSLSLTTSLLEEE
ELSSSLHHHHHHHLLTTTTTLLLEEEEESLL.

1hf9A

ALKKHHENEISHHAKEIERLQKEIERHKQSIKKLKQSEDDD.

LLSSHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHLL.

ΠΑΡΑΡΤΗΜΑ Δ

Αρχείο Δεδομένων Εξόδου

2e5tA Correctness Percentage:78.2609%

primaryStructure :IDVLRAKAAKERAERRLQSQQDDIDFKRAELALKRAMNRLSVAEMK

secondaryStructure :LLHHHHHHHHHHHHHHHHLLLLLLLLHHHHHHHHHHHHHHHHHHHLL

predictedSecondaryStructure:LLLHHHHHHHHHHLHLLLLLLLLLHHLHHHHHHHHHHHHHHLLLLLL

2e60A Correctness Percentage:84.1584%

primaryStructure
:GSSGSSGVAPLGLSVPSDVELPPTAKMHAIIERTASFVCRQGAQFEIMLKAKQARNSQFDFLRFDHYLNP
YYKFIQKAMKEGRYTVLAENKSDEKKKSGVS

secondaryStructure
:LLLLLLLLLLLLLLLLLLLLLLLHHHHHHHHHHHHHHHHHLLHHHHHHHHHHLLLLLLLHHHLLLLLHHHHH
HHHHHHHHHLLLLLLLLLLLLLLLLLL

predictedSecondaryStructure:LLLLLLLLLLLLLLLLLLLLLLLHHHHHHHHHLLLLEHLLLLLHHHHHHHLL
LLLLLLLLELLLLLLLLLHHHHHHHHHHHLLLLLLLLLLLLLLLLLL

2e61A Correctness Percentage:82.6087%

primaryStructure
:GSSGSSGEISGFGQCLVWVQCSFPNCGKWRRLCGNIDPSVLPDNWSCDQNTDVQYNRCDIPEETWTG
LE

secondaryStructure
:LLLLLLLLLLLLLLLLEEELLLLLLLEEELLLLLLLLLLLLLLHHHLLLHHHLLLLLLLLLLLL

predictedSecondaryStructure:LLLLLLLLLLLLLLLLELLLLLLLLLELLLLLLLLLLLLLLELEELLLLLLLLLL
LLLLELLLLL

2e62A Correctness Percentage:72.1311%

primaryStructure
:GSSGSSGNGMDEEQRQKRRRIEVALIEYRETLEEQGMKNPEEIERKVEINRKRLEVDYGLS

secondaryStructure
:LLLLLLLLLLHHHHHHHHHHHHHHHHHHHHHHHHHHHLLLLHHHHHHHHHHHHHHHHHHLLLL

predictedSecondaryStructure:LLLLLLLLLHHHLLHHHHHHHHELLLHHHHLHLLLLLLLLHHHLHHHHH
HLLLHHHLLLLL

ΠΑΡΑΡΤΗΜΑ Ε

Αρχείο Κωδικοποίησης Εισόδου

Κωδικοποίηση Εισόδου με MSA profiles

1α0αA.txt

0.25 0 0.5 0.25 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.27 0.71 0.01 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.78 0.22 0 0 0 0
0.01 0 0.03 0.01 0 0 0 0 0.26 0 0.03 0.43 0 0 0 0 0.01 0.2 0.01 0
0.08 0 0 0 0 0 0 0.16 0.03 0 0.33 0.03 0 0 0 0 0 0 0.37 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0.17 0 0 0 0 0 0 0 0 0 0 0 0 0.83 0 0 0 0
0.07 0.47 0.12 0 0 0 0.01 0 0.01 0 0.01 0 0 0.21 0 0 0 0.09 0 0
0.11 0 0.01 0 0 0 0 0 0.85 0 0.03 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.11 0.04 0.85 0 0 0
0 0 0 0 0 0 0 0.61 0.05 0 0 0.03 0 0 0.07 0.23 0 0.01 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0.16 0 0 0 0 0 0.01 0.01 0 0 0 0.09 0.71 0.01
0 0 0 0 0 0 0 0 0 0 0.09 0 0 0 0.73 0 0 0 0.17 0
0 0.16 0.68 0.16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0.05 0 0 0.03 0 0 0 0.17 0 0 0.75 0
0.07 0.05 0.03 0.08 0.03 0 0 0 0 0 0.08 0.21 0 0 0 0 0.04 0 0.41 0
0 0 0 0 0 0.01 0 0.25 0.71 0 0 0 0.03 0 0 0 0 0 0 0
0.12 0.53 0.17 0 0.17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0.01 0 0 0 0 0 0 0.08 0 0.04 0.03 0 0.07 0.08 0.32 0.19 0 0.01 0.17
0 0 0 0.04 0 0 0 0 0 0 0 0.15 0 0 0.01 0 0.03 0.72 0 0.05
0 0.63 0.35 0.03 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Κωδικοποίηση Εισόδου χωρίς MSA profiles

Orthogonal_Encoding_(Sparse)

****

Resedue_volume 1

A 10000000000000000000
C 01000000000000000000
D 00100000000000000000
E 00010000000000000000
F 00001000000000000000
G 00000100000000000000
H 00000010000000000000
I 00000001000000000000
K 00000000100000000000
L 00000000010000000000
M 00000000001000000000
N 00000000000100000000
P 00000000000010000000
Q 00000000000001000000
R 00000000000000100000
S 00000000000000010000
T 00000000000000001000
V 00000000000000000100
W 00000000000000000010
Y 00000000000000000001
X 00000000000000000000

ΠΑΡΑΡΤΗΜΑ Στ

Post-Processing Filtering

Self-Organized Map Kohonen

main.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sstream>

#include "parameters.h"
#include "pattern.h"
#include "epoch.h"
#include "neuron.h"
#include "neutralnet.h"


using namespace std;
using std::ifstream;



int main(){

	/* initialize random seed: */
	srand ( time(NULL) );

	//declare Neutral Network
	neutralnet myNeutralnet("parameters.dat");

	cout << "Simulation Begin..." << endl;
	//Start Simulation / Make calculations
	myNeutralnet.calculate();
	cout << "Simulation End..." << endl << endl;


	//Run the Test File automatic to see some examples
	myNeutralnet.autorunTestFile();




	return 0;
}
```

Neutralnet.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "neutralnet.h"
#include "pssp.h"

using namespace std;
using std::ifstream;

neutralnet::neutralnet(string paramfile){

	//My Parameters, define and read
	myParamaters = new parameters(paramfile);
	myParamaters->setNumOfInputs(15);
	myParamaters->print(); // print them

	//My Training and Test File
	string trainingFile=myParamaters->getTrainFile();
	string testFile=myParamaters->getTestFile();
	myTraining= new epoch(trainingFile,myParamaters-
>getNumOfInputs());
	myTest= new epoch(testFile,myParamaters->getNumOfInputs());
	  psspFile=new pssp(testFile);
	//other initializes
	learningRate=myParamaters->getLearningRate();

	radius = myParamaters->getMapSideSize()/2;
	T = myParamaters->getMaxIterations()/log(radius);


	//create neurons of the network
	for (int i=0; i<myParamaters->getMapSideSize(); i++)
	{
		myNeurons.push_back( vector<neuron>() );
		for (int j=0; j<myParamaters->getMapSideSize(); j++)
		{
			myNeurons.back().push_back(neuron(myParamaters-
>getNumOfInputs()));
		}
	}



}
```

neutralnet.h

```
#ifndef NEUTRALNET_H_INCLUDED
#define NEUTRALNET_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "parameters.h"
#include "pattern.h"
#include "epoch.h"
#include "neuron.h"
#include "bmp.h"
#include "pssp.h"




#define startLearningRate 0.6

using namespace std;
using std::ifstream;


class neutralnet {

private:
	vector < vector <neuron> > myNeurons;
	neuron* winningNeuron; //store pointer of the winning neuron
	int winningNeuronX; //position
	int winningNeuronY;
	double radius; //radius of the map
	parameters *myParamaters;
	epoch *myTraining;
	epoch *myTest;
	double neighbourhoodRadius; // the radious of neighbourhood
	double influence;
	double learningRate;
	double  T; // time constant

	   pssp *psspFile;

public:
	neutralnet(string paramfile);

	//Set Values


	void calculate()
	{

		//Writing results.dat
		ofstream resultsFile("results.dat",ios::out);
```

```
                ifstream myFile;
                myFile.open("Z_Output1.txt");
                ofstream script("script.txt",ios::out);
                char temp[4000];

            for (int k=0; k<myParamaters->getMaxIterations(); k++)
            {
                  cout << "Itaration : " << k  << endl;

                  resultsFile << k;

                  float error=0.0;

                  //run Training File
                  for (int i=0; i<myTraining->getNumOfPatterns();
i++)
                  {
                              for(int s=0; s<myTraining-
>getNumOfInputsInPattern(i); s++){


                        winningNeuron = findWinningNeuron(myTraining-
>getInputs(i,s));
                        //finding error
                        error+=winningNeuron-
>calculateDistance(myTraining->getInputs(i,s));

                        //calculate radius of neighbourhood
                        neighbourhoodRadius = radius * exp(-
(double)k/T);

                        for (int n=0; n<myParamaters-
>getMapSideSize(); n++)
                        {

                              for (int m=0; m<myParamaters-
>getMapSideSize(); m++)
                              {
                                    //the Euclidean distance
                                    double DistToNodeSq
=pow((winningNeuronX-m),2) + pow((winningNeuronY-n),2) ;

                                    double WidthSq =
pow(neighbourhoodRadius ,2);

                                    //if within the neighbourhood
adjust its weights
                                    if (DistToNodeSq < (WidthSq))
                                    {

                                          //calculate by how much its
weights are adjusted
                                          influence = exp(-
(DistToNodeSq) / (2*WidthSq));


      myNeurons[n][m].changeWs(myTraining->getInputs(i,s),
learningRate,influence);


      myNeurons[n][m].setContent(myTraining->getOutput(i,s));
```

```
                                    }

                                }

                            }

                            }
                            cout << "Itaration : " << k  << endl;
                        }

                        //Final Training Error
                        error/=myTraining->getNumOfPatterns();
                        resultsFile << "\t\t\t" << error;


                        //change learning rate
                        learningRate = startLearningRate * exp(-
(double)k/myParamaters->getMaxIterations());


                        //run Test File
                        for (int i=0; i<myTest->getNumOfPatterns(); i++)
                        {
                                    for(int s=0; s<myTraining-
>getNumOfInputsInPattern(i); s++){
                            winningNeuron = findWinningNeuron(myTest-
>getInputs(i,s));

                            //finding error
                            error+=winningNeuron-
>calculateDistance(myTest->getInputs(i,s));
                                    }

                        }

                        //Final Test Error
                        error/=myTest->getNumOfPatterns();
                        resultsFile << "\t\t\t" << error  << "\n";

                }

                //closing file
                resultsFile.close();

                //writing weights to file
                printWeights();

                //writing chars to file | Clustering
                printCharacterMap();

                //writing chars to file as colors | Clustering with Image
                printImage();
                    int counter=0;
                    for (int i=0; i<psspFile->getNumOfPatterns(); i++)
                      {

                                myFile >> temp;
                                script << temp;
                                script << " ";
                                myFile >> temp;
                                script << temp;
```

```
                                script << " ";
                                myFile >> temp;
                                script << temp;
                                script << endl;

                                myFile >> temp;
                                script << temp;
                                script << "           ";
                                myFile >> temp;
                                script << temp;
                                script << endl;

                                myFile >> temp;
                                script << temp;
                                script << "         ";
                                myFile >> temp;
                                script << temp;
                                script << endl;

                                script<< "predictedSecondaryStructure:";

                        // cout<<"kati"<<endl;
                                for(int k=0;k<psspFile-
>getproteinsize(i);k++){
                         //calculate output in hidden layer neurons
                                    /*vector <double> tem;
                                        tem=psspFile->getInputs(i,k);
                                                for(int
s=0;s<tem.size();s++)
                                                        cout<<
tem.at(s)<<endl;
                                */
                                    int ot=0;
                                    counter++;

                            winningNeuron = findWinningNeuron(psspFile-
>getInputs(i,k));

                            char tempActual=winningNeuron->getContent();
                            //char tempWanted=myTest->getOutput(k);



                                     // if(ot==0)

script<<tempActual;
                                       /*else if (ot==1)

script<<tempActual;
                                       else

script<<tempActual;
                                        */
                                    //error += pow( myTest-
>getInputValueOfPattern(i,j) - myOutputNeurons[j].getValue(), 2);


                                }
                           script<<endl;
                            myFile >> temp;
                        }
```

```
        script.close();
        myFile.close();



    }


    neuron* findWinningNeuron(vector<double> inputs)
    {
        neuron* rv = NULL;


        double smallestDistance =
myNeurons[0][0].calculateDistance(inputs);

        rv=&myNeurons[0][0];
        winningNeuronX=0;
        winningNeuronY=0;


        for (int i=0; i<myParamaters->getMapSideSize(); i++)
        {
            for (int j=0; j<myParamaters->getMapSideSize();
j++)
            {
                //if distance < smallest
                double distance =
myNeurons[i][j].calculateDistance(inputs);
                if (distance < smallestDistance)
                {
                    smallestDistance = distance;
                    rv = &myNeurons[i][j];
                    winningNeuronY=i;
                    winningNeuronX=j;
                }

            }
        }
        return rv;
    }



    void print()
    {
        cout << "This is a NeutralNet!" << endl;
    }

    void printWeights()
    {
        //Writing (weights.dat)
        ofstream weightsFile("weights.dat",ios::out);

        //for each neuron row
        for (int i=0; i<myParamaters->getMapSideSize(); i++)
        {

            //for each neuron
```

```
                  for (int j=0; j<myParamaters->getMapSideSize();
j++)
                  {
                        weightsFile << "Neuron No: " << i << " , " <<
j << endl;


                        /// Print the weights of neuron
                        for (int k=0; k<myNeurons[i][j].getNumOfWs();
k++)
                        {
                              weightsFile << "\t"  <<
myNeurons[i][j].getW(k) << "\n";
                        }
                        weightsFile << endl;
                  }

            }

            weightsFile.close();

      }

      //Print Clustering
      void printCharacterMap()
      {
            //Writing (weights.dat)
            ofstream charMapFile("clustering.dat",ios::out);

            //for each neuron row
            for (int i=0; i<myParamaters->getMapSideSize(); i++)
            {

                  //for each neuron
                  for (int j=0; j<myParamaters->getMapSideSize();
j++)
                  {
                        charMapFile << myNeurons[i][j].getContent()
<< " ";
                  }
                  charMapFile << endl;

            }

            charMapFile.close();

      }

      //auto run the Test File
      void autorunTestFile()
      {
            cout << "Wanted\t\t" << "Actual" << endl;

            int countTrue=0;

            //run Test File
            for (int i=0; i<myTest->getNumOfPatterns(); i++)
            {
                    for(int s=0; s<myTraining-
>getNumOfInputsInPattern(i); s++){
```

```
                  winningNeuron = findWinningNeuron(myTest-
>getInputs(i,s));

                  char tempActual=winningNeuron->getContent();
                  char tempWanted=myTest->getOutput(i,s);

                  if (tempWanted==tempActual)
                  {
                        countTrue++;
                  }

                  cout << tempWanted << "\t\t" << tempActual << endl;
                    }

            }

            cout << "Find True: " << countTrue << "/" << myTest-
>getNumOfPatterns() << endl;

      }

      //print clustering in Image
      void printImage()
      {
            bitmap myResults(myParamaters->getMapSideSize(),20);

            myResults.createBitmap(myNeurons);
      }



};

#endif
```

Neuron.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sstream>
#include "neuron.h"

using namespace std;
using std::ifstream;


neuron::    neuron(int nOfWs){
      numOfWs=nOfWs;

      for (int i=0; i<numOfWs; i++)
      {
            /* generate random number and tranform between 0 and 1:
*/
            double temp = rand() % 10000 + 1;
            temp/=10000;
            //cout << temp << endl;
            w.push_back(temp);
      }
      //cout << endl;
}
```

neuron.h

```
#ifndef NEURON_H_INCLUDED
#define NEURON_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>

using namespace std;
using std::ifstream;

class neuron {

private:
	vector< double> w;
	int numOfWs;
	char content;

public:
	neuron(int nOfWs);

	//Set Values


	void addW(double value){
		w.push_back(value);
	}

	void setW(double value, int position){
		w.at(position)=value;
	}

	char setContent(char c){
		content=c;
	}


	//Get Values


	double getW(int i){
		return w.at(i);
	}

	char getContent(){
		return content;
	}

	int getNumOfWs(){
		return numOfWs;
	}


	/* calulate distance */
	double calculateDistance(vector<double> inputs)
```

```
	{
		double distance = 0.0;


		for (int i=0; i<numOfWs; i++)
		{
			distance += (inputs[i] - w[i]) * (inputs[i] -
w[i]);
		}
		return sqrt(distance);
	}

	/* change weights */
	void changeWs(vector<double> inputs, double learningRate,
double influence)
	{
		for (int i=0; i<inputs.size(); i++)
		{
			w[i] += learningRate * influence * (inputs[i] -
w[i]);
		}
	}



};

#endif
```

pattern.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "pattern.h"

using namespace std;
using std::ifstream;



pattern::pattern ()
{
	//do nothing
}
```

pattern.h

```
#ifndef PATTERN_H_INCLUDED
#define PATTERN_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>

using namespace std;
using std::ifstream;

class pattern {

private:
	vector<double> inputs;
	char output;

	  vector<int> ots;

public:

	pattern ();

	//Set Values
	void addInput(double value)
	{
		inputs.push_back(value);
	}

	void setOutput(char value)
	{
		output=value;
	}

	//Get Values
	double getInput(int position)
	{
		return inputs.at(position);
	}


	int getNumOfInputs()
	{
		return inputs.size();
	}


	char getOutput()
	{
		return output;
	}

	vector <double> getInputs()
	{
		return inputs;
```

```
	}

	void print()
	{
		cout << " -> pattern:" << endl;
		for (int i=0; i<inputs.size(); i++)
			cout << "\t Input" << i << ": " << inputs.at(i) <<
endl;

		cout << "\t Output" << ": " << output << endl;
	}



	   void addOutput(int value)
	{
		ots.push_back(value);
	}
};


#endif
```

pssp.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "epoch.h"
#include <sstream>
#include <algorithm>
#include <iterator>

using namespace std;
using std::ifstream;

#include "pssp.h"

pssp::pssp (string s)
{
	fileName=s;
	int tempI=0;
	  int counter=0;

	char temp[4000];
	  char * split;



	  char secondaryStructure[4000];

	  vector <pattern> protein;


	//Reading File

	ifstream myFile;
	  myFile.open("Z_Filter_Output1.txt");
	  myFile >> temp;
	while(!myFile.eof())
	{

			myFile >> temp;
			myFile >> temp;
			myFile >> temp;
			myFile >> temp;
			myFile >> temp;
			myFile >> temp;
			for(int i=1;i<500;i++){
			    secondaryStructure[i-1]=temp[i];
			}


			for(int i=0;i<500;i++){
			    //counter++;
			    if (secondaryStructure[i]=='\0')
			        break;

			    else{
```

```
                        protein.push_back(pattern());
                        if (secondaryStructure[i]=='H')
                            protein.back().addOutput(1);
                        else
                            protein.back().addOutput(0);
                        if (secondaryStructure[i]=='E')
                            protein.back().addOutput(1);
                        else
                            protein.back().addOutput(0);
                        if (secondaryStructure[i]=='L')
                            protein.back().addOutput(1);
                        else
                            protein.back().addOutput(0);
                        }
                    }
                    //print();
                myFile >> temp;
                    myFile >> temp;
                    myFile >> temp;
                    myFile >> temp;
                    myFile >> temp;
                    myFile >> temp;

                      split = strtok (temp,",");
                      counter=0;
                      while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                          split = strtok (NULL,",");
                          counter++;
                      }

                      myFile >> temp;
                      myFile >> temp;
                      split = strtok (temp,",");
                      counter=0;
                      while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                          split = strtok (NULL,",");
                          counter++;
                      }

                      myFile >> temp;
                      myFile >> temp;
                      split = strtok (temp,",");
                      counter=0;
                      while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                          split = strtok (NULL,",");
                          counter++;
                      }
                      //tempI=counter;
                      sequence.push_back(protein);
                      myFile >> temp;

                      protein.clear();

        }
        myFile.close();
```

```
        //print();
        /*for(int i=0;i<sequence.size();i++)
            if(i==1)
            for(int j=0;j<sequence.at(i).size();j++)
                 for(int
k=0;k<sequence.at(i).at(j).getNumOfInputs();k++)
                cout<< sequence.at(i).at(j).getInput(k)<<endl;
*/




}
```

pssp.h

```
#ifndef PSSP_H_INCLUDED
#define     PSSP_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "pattern.h"

using namespace std;
using std::ifstream;

class pssp {

private:
      vector <vector <pattern> >  sequence;

      string fileName;
      vector <double> minInputsInColumn;
      vector <double> maxInputsInColumn;

public:

      pssp (string s);

        int getNumOfPatterns()
      {
            return sequence.size();
      }

         int getproteinsize(int value)
      {
            return sequence.at(value).size();
      }

         vector <double> getInputs(int i,int j)
      {
            return sequence.at(i).at(j).getInputs();
      }




};
#endif
```

epoch.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "epoch.h"
#include <sstream>
#include <algorithm>
#include <iterator>

using namespace std;
using std::ifstream;



epoch::epoch (string s,int numOfIn)
{
	fileName=s;
	numOfInputs=numOfIn;
	int tempI=0;
	  int counter=0;

	char temp[4000];
	  char * split;



	  char secondaryStructure[4000];
	  vector <pattern> protein;



	//Reading File

	ifstream myFile;
	  myFile.open("Z_Filter.txt");
	  //ofstream resultsFile("dokimi.txt",ios::out);
	  myFile >> temp;
	while(!myFile.eof())
	{
		  myFile >> temp;
		  myFile >> temp;
		  myFile >> temp;
		  myFile >> temp;
		  myFile >> temp;
		  for(int i=0;i<4000;i++){
			temp[i]='\0';
			secondaryStructure[i]='\0';
		   }
		  myFile >> temp;

		  for(int i=1;i<500;i++){
			if (temp[i]=='\0')
			   break;
			secondaryStructure[i-1]=temp[i];
		  }
```

```
                    for(int i=0;i<500;i++){
                       //counter++;
                       if (secondaryStructure[i]=='\0')
                           break;
                       else{
                           protein.push_back(pattern());
                          if (secondaryStructure[i]=='H')
                               protein.back().setOutput('H');
                           else if (secondaryStructure[i]=='E')
                               protein.back().setOutput('E');
                           else if (secondaryStructure[i]=='L')
                               protein.back().setOutput('L');

                           }

                    }

                   myFile >> temp;
                   myFile >> temp;
                   myFile >> temp;
                   myFile >> temp;
                   myFile >> temp;
                   myFile >> temp;

                     split = strtok (temp,",");
                     counter=0;
                     while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                         split = strtok (NULL,",");
                         counter++;
                     }

                     myFile >> temp;
                     myFile >> temp;
                     split = strtok (temp,",");
                     counter=0;
                     while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                         split = strtok (NULL,",");
                         counter++;
                     }

                     myFile >> temp;
                     myFile >> temp;
                     split = strtok (temp,",");
                     counter=0;
                     while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                         split = strtok (NULL,",");
                         counter++;
                     }
                     //tempI=counter;
                     myPatterns.push_back(protein);
                     myFile >> temp;

                     protein.clear();
```

```
      }
        myFile.close();
       /* for(int i=0;i<myPatterns.size();i++)
            if(i==8)
            for(int j=0;j<myPatterns.at(i).size();j++)
                 for(int
k=0;k<myPatterns.at(i).at(j).getNumOfInputs();k++)
               cout<< myPatterns.at(i).at(j).getInput(k)<<endl;*/
       // vector <double> temp;
       int window=2;
          for (int i=0; i<getNumOfPatterns(); i++){

                              for(int s=0;
s<getNumOfInputsInPattern(i); s++){
                                  if(s==0){
                                      for(int x=0; x<window*3; x++){

myPatterns.at(i).at(s).addInput(0);
                                      }
                                  }
                                  else{
                                  vector <double> temp;
                                  temp=getInputs(i,s-1);
                                  for(int x=0; x<window*3; x++){

myPatterns.at(i).at(s).addInput(temp[x]);
                                  }
                                  }

                                  if(myPatterns.at(i).size()-1-
s<window){
                                        for(int x=1;
x<=myPatterns.at(i).size()-1-s; x++){
                                             vector <double> temp;
                                             temp=getInputs(i,s+x);
                                             for(int o=0; o<3; o++)

myPatterns.at(i).at(s).addInput(temp[o]);
                                        }
                                         for(int x=0; x<window-
(myPatterns.at(i).size()-1-s); x++){
                                              for(int o=0; o<3; o++)

myPatterns.at(i).at(s).addInput(0);
                                         }
                                    }
                                  else{
                                  for(int x=1; x<=window; x++){
                                       vector <double> temp;
                                       temp=getInputs(i,s+x);
                                       for(int o=0; o<3; o++)

myPatterns.at(i).at(s).addInput(temp[o]);

                                    }
                                  }
                              }
          }

           /*for(int i=0;i<myPatterns.size();i++)
               if(i==490)
```

```
            for(int j=0;j<myPatterns.at(i).size();j++)
                     if(j==211)
                  for(int
k=0;k<myPatterns.at(i).at(j).getNumOfInputs();k++)
                cout<< myPatterns.at(i).at(j).getInput(k)<<endl;*/









        /*while(!myFile.eof())
      {
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                 for(int i=0;i<4000;i++){
                     temp[i]='\0';
                     secondaryStructure[i]='\0';
                 }
                myFile >> temp;
                for(int i=1;i<500;i++){
                    if (temp[i]=='\0')
                        break;
                    secondaryStructure[i-1]=temp[i];
                    resultsFile << secondaryStructure[i-1];
                }
                   resultsFile << endl;

                for(int i=0;i<500;i++){
                    //counter++;
                    if (secondaryStructure[i]=='\0')
                        break;

                    else{
                        myPatterns.push_back(pattern());
                       if (secondaryStructure[i]=='H')
                            myPatterns.back().setOutput('H');
                        else if (secondaryStructure[i]=='E')
                            myPatterns.back().setOutput('E');
                        else if (secondaryStructure[i]=='L')
                            myPatterns.back().setOutput('L');
                        resultsFile << myPatterns.back().getOutput();
                        }
                }
                resultsFile << endl;
                //resultsFile << endl;
```

```
                //print();
            myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;



                  split = strtok (temp,",");
                  counter=tempI;
                  int count2=0;
                  while(split!=NULL){

myPatterns.at(counter).addInput(strtod(split,NULL));
                      split = strtok (NULL,",");
                      counter++;
                      count2++;
                  }
                  counter=tempI;*/

                  /* for(int d=0;d<count2;d++)
                        for(int s=0;s<2;s++){
                   if(d==0 ||d==1 || d==2)
                      myPatterns.at(d).addInput(0);
                   else{
                       myPatterns.at(d).addInput( myPatterns.at(d-s-
1).getInput());
                   }
                        }*/

               /*   myFile >> temp;
                  myFile >> temp;
                  split = strtok (temp,",");
                  counter=tempI;
                  while(split!=NULL){

myPatterns.at(counter).addInput(strtod(split,NULL));
                      split = strtok (NULL,",");
                      counter++;
                  }

                  myFile >> temp;
                  myFile >> temp;
                  split = strtok (temp,",");
                  counter=tempI;
                  while(split!=NULL){

myPatterns.at(counter).addInput(strtod(split,NULL));
                      split = strtok (NULL,",");
                      counter++;
                  }
                  tempI=counter;
                  myFile >> temp;


      }*/

}
```

epoch.h

```
#ifndef EPOCH_H_INCLUDED
#define EPOCH_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "pattern.h"

using namespace std;
using std::ifstream;

class epoch {

private:
	vector <vector <pattern> > myPatterns;
	  //vector <pattern> myPatterns2;
	string fileName;
	int numOfInputs;

public:

	epoch (string s,int numOfIn);

	void print()
	{
		cout << "Epoch: " << fileName << "\n";
		for (int i=0; i<myPatterns.size(); i++){
			cout << "\t" << i;
			//myPatterns.at(i).print();
		}
	}


	int getNumOfPatterns()
	{
		return myPatterns.size();
	}


	int getNumOfInputsInPattern(int patternPos)
	{
		/*For creating window size we change this one to the next
one*/
		//return myPatterns.at(patternPos).getNumOfInputs();
		    return myPatterns.at(patternPos).size();
	  }

	double getInputValueOfPattern(int patternPos, int inputPos)
	{
		//return myPatterns.at(patternPos).getInput(inputPos);
	}
```

```
        int getOutputValueOfPattern(int patternPos)
        {
              //return myPatterns.at(patternPos).getOutput();
        }

        vector <double> getInputs(int i,int j)
        {
        /*For creating window size we change this one to the next one*/
              //return myPatterns.at(patternPos).getInputs();
                  return myPatterns.at(i).at(j).getInputs();
          }

        char getOutput(int i,int j)
        {
              return myPatterns.at(i).at(j).getOutput();
        }

};

#endif
```

parameters.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "parameters.h"

using namespace std;
using std::ifstream;



parameters::parameters (string s="parameters.dat") {

	//initialize
	mapSideSize=-1;
	learningRate=-1;
	neighborhoodRadius=-1;
	maxIterations=-1;
	trainFile="";
	testFile="";
	fileName=s;
	numOfInputs=-1;

	//tempory Vars
	int tempI;
	double tempD;
	string tempS;

	//Reading Parameters
	ifstream parametersFile(fileName.c_str(),ios::in);
	parametersFile >> tempS; parametersFile >> tempI;
setMapSideSize(tempI);
	parametersFile >> tempS; parametersFile >> tempD;
setLearningRate(tempD);
	parametersFile >> tempS; parametersFile >> tempI;
setNeighborhoodRadius(tempI);
	parametersFile >> tempS; parametersFile >> tempI;
setMaxIterations(tempI);
	parametersFile >> tempS; parametersFile >> tempS;
setTrainFile(tempS);
	parametersFile >> tempS; parametersFile >> tempS;
setTestFile(tempS);
	parametersFile.close();
}
```

paraneters.h

```
#ifndef PARAMETERS_H_INCLUDED
#define PARAMETERS_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>

using namespace std;
using std::ifstream;

class parameters {

private:
	int mapSideSize;
	double learningRate;
	int neighborhoodRadius;
	int maxIterations;
	string trainFile;
	string testFile;
	string fileName;

	int numOfInputs;


public:
	parameters (string s);

	void setMapSideSize(int i)
	{
		mapSideSize=i;
	}

	void setLearningRate(double d)
	{
		learningRate=d;
	}

	void setNeighborhoodRadius(int i)
	{
		neighborhoodRadius=i;
	}

	void setMaxIterations(int i)
	{
		maxIterations=i;
	}

	void setTrainFile(string s)
	{
		trainFile=s;
	}

	void setTestFile(string s)
	{
```

```
            testFile=s;
      }

      void setNumOfInputs(int i)
      {
            numOfInputs=i;
      }


      int getMapSideSize()
      {
            return mapSideSize;
      }

      double getLearningRate()
      {
            return learningRate;
      }

      int getNeighborhoodRadius()
      {
            return neighborhoodRadius;
      }

      int getMaxIterations()
      {
            return maxIterations;
      }

      string getTrainFile()
      {
            return trainFile;
      }

      string getTestFile()
      {
            return testFile;
      }

      int getNumOfInputs()
      {
            return numOfInputs;
      }

      void print()
      {
            cout << "Parameters: " << fileName << endl;
            cout << "\t mapSideSize: " << mapSideSize << endl;
            cout << "\t learningRate: " << learningRate << endl;
            cout << "\t neighborhoodRadius: " << neighborhoodRadius
<< endl;
            cout << "\t maxIterations: " << maxIterations << endl;
            cout << "\t trainFile: " << trainFile << endl;
            cout << "\t testFile: " << testFile << endl;
            cout << "\t numOfInputs: " << numOfInputs << endl;
      }

};
#endif
```

bmp.h

```
#include <fstream>
#include <iostream>
#include "neuron.h"


using namespace std;

class bitmap {

private:


	struct BMP_HEADER
	{

		long size_of_file;
		long reserve;
		long offset_of_pixle_data;
		long size_of_header;
		long width;
		long hight;
		short num_of_colour_plane;
		short num_of_bit_per_pix;
		long compression;
		long size_of_pix_data;
		long h_resolution;
		long v_resolution;
		long num_of_colour_in_palette;
		long important_colours;

	}
	BMP_HEADER;

	struct COLOR_PALLATE
	{
		char blue;
		char green;
		char red;

	}COLOR_PALLATE;

	short padding;
	short BM ;
	int w_in_pix;
	int h_in_pix;
	struct COLOR_PALLATE BGR [26];
	int sizeOfPixel;

	void initializeColors()
	{
		//color 1 mple
		BGR [0].blue = 0xFF;		BGR [0].green = 0x00;
	BGR [0].red = 0x00;
		//color 2 prasino
		BGR [1].blue = 0x00;		BGR [1].green = 0xFF;
	BGR [1].red = 0x00;
		//color 3 kokkino
		BGR [2].blue = 0x00;		BGR [2].green = 0x00;
	BGR [2].red = 0xFF;
```

```
      //color 4 kitrino
      BGR [3].blue = 0x00;          BGR [3].green = 0xFF;
BGR [3].red = 0xFF;
      //color 5 portokali
      BGR [4].blue = 0x00;          BGR [4].green = 0x99;
BGR [4].red = 0xFF;
      //color 6 galazio
      BGR [5].blue = 0xFF;          BGR [5].green = 0xFF;
BGR [5].red = 0x00;
      //color 7 fuxia
      BGR [6].blue = 0xFF;          BGR [6].green = 0x00;
BGR [6].red = 0xFF;
      //color 8 visini
      BGR [7].blue = 0x00;          BGR [7].green = 0x00;
BGR [7].red = 0x99;
      //color 9 kafe
      BGR [8].blue = 0x00;          BGR [8].green = 0x66;
BGR [8].red = 0x99;
      //color 10 prasino skouro
      BGR [9].blue = 0x00;          BGR [9].green = 0x66;
BGR [9].red = 0x33;
      //color 11 portokali skouro
      BGR [10].blue = 0x00;         BGR [10].green = 0x66;
BGR [10].red = 0xFF;
      //color 12 lila
      BGR [11].blue = 0x99;         BGR [11].green = 0x33;
BGR [11].red = 0x99;
      //color 13 mple skouro
      BGR [12].blue = 0x99;         BGR [12].green = 0x00;
BGR [12].red = 0x33;
      //color 14 mple poly skouro
      BGR [13].blue = 0x66;         BGR [13].green = 0x00;
BGR [13].red = 0x00;
      //color 15 lila-galazio
      BGR [14].blue = 0xFF;         BGR [14].green = 0x99;
BGR [14].red = 0x99;
      //color 16 PORTOKALO-KAFE
      BGR [15].blue = 0x00;         BGR [15].green = 0x66;
BGR [15].red = 0xCC;
      //color 17 PRASINO-KITRINO
      BGR [16].blue = 0x00;         BGR [16].green = 0xFF;
BGR [16].red = 0x99;
      //color 18 XAKI
      BGR [17].blue = 0x00;         BGR [17].green = 0x99;
BGR [17].red = 0x99;
      //color 19 KITRINOPRASINO
      BGR [18].blue = 0x00;         BGR [18].green = 0xFF;
BGR [18].red = 0xCC;
      //color 20 XAKI SKOURO
      BGR [19].blue = 0x00;         BGR [19].green = 0x66;
BGR [19].red = 0x66;
      //color 21 LILA ANOIXTO
      BGR [20].blue = 0x99;         BGR [20].green = 0x66;
BGR [20].red = 0x99;
      //color 22 PRASINO-GALAZIO
      BGR [21].blue = 0x99;         BGR [21].green = 0xFF;
BGR [21].red = 0x33;
      //color 23 PRASINO POLY ANOIXTO
      BGR [22].blue = 0x99;         BGR [22].green = 0xFF;
BGR [22].red = 0x99;
      //color 24 ROZ
```

```
            BGR [23].blue = 0xFF;          BGR [23].green = 0x33;
      BGR [23].red = 0xFF;
            //color 25 KAFE ANOIXTO
            BGR [24].blue = 0x00;          BGR [24].green = 0x99;
      BGR [24].red = 0xCC;
            //color 26 LILA SKOURO
            BGR [25].blue = 0x66;          BGR [25].green = 0x33;
      BGR [25].red = 0x66;


      }

public:

      bitmap (int sizeOfSide,int sizeOfP){

            padding = 0x0000;
            BM = 0x4d42;
            sizeOfPixel=sizeOfP;

            w_in_pix = sizeOfSide * sizeOfPixel;
            h_in_pix = sizeOfSide * sizeOfPixel;

            /* colors */
            initializeColors();

      }



      int createBitmap(vector < vector <neuron> > myNeurons)
      {
            BMP_HEADER.size_of_file = sizeof(BMP_HEADER) +
sizeof(BGR) + sizeof(padding) * h_in_pix + 2;
            BMP_HEADER.reserve = 0000;
            BMP_HEADER.offset_of_pixle_data = 54;
            BMP_HEADER.size_of_header = 40;
            BMP_HEADER.width = w_in_pix;
            BMP_HEADER.hight = h_in_pix;
            BMP_HEADER.num_of_colour_plane = 1;
            BMP_HEADER.num_of_bit_per_pix = 24;
            BMP_HEADER.compression = 0;
            BMP_HEADER.size_of_pix_data = sizeof(BGR) +
sizeof(padding) * h_in_pix;
            BMP_HEADER.h_resolution = 2835;
            BMP_HEADER.v_resolution = 2835;
            BMP_HEADER.num_of_colour_in_palette = 0;
            BMP_HEADER.important_colours = 0;

            /* BMP Header */

            ofstream file;
            file.open ("clustering.bmp", ios::out | ios::trunc |
ios::binary);
            file.write ((char*)(&BM), 2);
            file.write ((char*)(&BMP_HEADER), sizeof(BMP_HEADER));


            /* colors */
            int color=0;
            int color2=0;
```

```
            /* print colors ISIA */
            for (int h=h_in_pix/sizeOfPixel-1; h>=0; h--)
            {


                  for (int m=0; m<sizeOfPixel; m++)
                  {

                        color=myNeurons[h][0].getContent() - 'A';

                        for (int w=0; w<w_in_pix/sizeOfPixel; w++)
                        {
                              for (int k=0; k<sizeOfPixel; k++)
                              {
                                    file.write
((char*)(&BGR[(color)].blue), 1);
                                    file.write
((char*)(&BGR[(color)].green), 1);
                                    file.write
((char*)(&BGR[(color)].red), 1);
                              }

                              color=myNeurons[h][w+1].getContent() -
'A';
                        }

                  }

            }



      }


};
```

Radial Basis Function Δίκτυο

main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include <cstdlib>

#include "parameters.h"
#include "pattern.h"
#include "epoch.h"
#include "centerVector.h"
#include "centers.h"
#include "hiddenneuron.h"
#include "outputneuron.h"
#include "neutralnet.h"


using namespace std;
using std::ifstream;



int main(){

	/* initialize random seed: */
	srand ( time(NULL) );

	//declare Neutral Network
	neutralnet myNeutralnet("parameters.txt");


	cout << "Simulation Begin..." << endl;
	//Start Simulation / Make calculations
	myNeutralnet.calculate();
	cout << "Simulation End..." << endl << endl;


	//Run the Test File automatic to see some examples
	//myNeutralnet.autorunTestFile();




	return 0;
}
```

neutralnet.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "neutralnet.h"

using namespace std;
using std::ifstream;

neutralnet::neutralnet(string paramfile){

	//My Parameters, define and read
	myParamaters = new parameters(paramfile);
	myParamaters->print(); // print them


	//My Training and Test File
	string trainingFile=myParamaters->getTrainFile();
	string testFile=myParamaters->getTestFile();
	myTraining= new epoch(trainingFile,myParamaters-
>getNumInputNeurons(),myParamaters->getNumOutputNeurons());
	myTest= new epoch(testFile,myParamaters-
>getNumInputNeurons(),myParamaters->getNumOutputNeurons());
	  psspFile=new pssp(testFile);

	//myTraining->print();
	//myTraining->printMaxMin();


	//My Centers File
	string centersFile=myParamaters->getCentersFile();
	//if the file name is not "none" means that a file exist
	if (centersFile!="none")
		myCenters= new centers(centersFile,myParamaters-
>getNumInputNeurons());
	//else create random centers
	else
		myCenters= new centers(myTraining-
>getMaxInputsInColumn(),myTraining-
>getMinInputsInColumn(),myParamaters-
>getNumInputNeurons(),myParamaters->getNumHiddenLayerNeurons());

	//myCenters->print();



	//other initializes

	  myParamaters->setNumHiddenLayerNeurons(myCenters-
>getNumOfCenterVectors());
	//initialize hiddenNeurons
	for (int i=0; i<myParamaters->getNumHiddenLayerNeurons(); i++)
	{
```

```
            myHiddenNeurons.push_back( hiddenneuron(myParamaters-
>getNumInputNeurons(),myCenters->getInputs(i),myParamaters-
>getSigmas()) );
      }

      //initialize outputNeurons
      for (int i=0; i<myParamaters->getNumOutputNeurons(); i++)
      {
            // +1 is for bias
            myOutputNeurons.push_back( outputneuron(myParamaters-
>getNumHiddenLayerNeurons()+1) );
      }

}
```

neutralnet.h

```
#ifndef NEUTRALNET_H_INCLUDED
#define NEUTRALNET_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "parameters.h"
#include "pattern.h"
#include "epoch.h"
#include "centerVector.h"
#include "centers.h"
#include "hiddenneuron.h"
#include "outputneuron.h"
#include "pssp.h"


#define startLearningRate 0.6

using namespace std;
using std::ifstream;


class neutralnet {

private:
      vector <outputneuron> myOutputNeurons;
      vector <hiddenneuron> myHiddenNeurons;

        pssp *psspFile;
      parameters *myParamaters;
      epoch *myTraining;
      epoch *myTest;
      centers *myCenters;

public:
      neutralnet(string paramfile);

      //Set Values


      void calculate()
      {

            //Writing results.dat
            ofstream resultsFile("results.dat",ios::out);

                ifstream myFile;
                myFile.open("Z_Output.txt");
                ofstream script("script.txt",ios::out);
                char temp[4000];


            for (int k=0; k<myParamaters->getMaxIterations(); k++)
            {
```

```
                    cout << "Itaration : " << k  << endl;

                    resultsFile << k;

                    float error=0.0;

                    //run Training File
                    for (int i=0; i<myTraining->getNumOfPatterns();
i++)
                    {

                          //calculate output in hidden layer neurons
                          for (int j=0; j<myHiddenNeurons.size(); j++)
                          {

      myHiddenNeurons[j].calculate(myTraining->getInputs(i));
                          }


                          //calculate the output in output layer
neurons
                          for (int j=0; j<myOutputNeurons.size(); j++)
                          {

      myOutputNeurons[j].calculate(myHiddenNeurons);
                                //finding error
                                error += pow( myTraining-
>getInputValueOfPattern(i,j) - myOutputNeurons[j].getValue(), 2);
                          }

                          //change Ws
                          for (int j=0; j<myOutputNeurons.size(); j++)
                          {

      myOutputNeurons[j].changeWs(myHiddenNeurons,myParamaters-
>getLearningRateW(),myTraining->getOutputValueOfPattern(i,j));
                          }


                          //change center position and diaspora
                          for (int j=0; j<myHiddenNeurons.size(); j++)
                          {

      myHiddenNeurons[j].changeCenterPos(myParamaters-
>getLearningRateR(),myOutputNeurons,myTraining-
>getInputs(i),myTraining->getOutputs(i));

      myHiddenNeurons[j].changeDiaspora(myParamaters-
>getLearningRateS(),myOutputNeurons,myTraining-
>getInputs(i),myTraining->getOutputs(i));
                          }

                    }


                    //Final Training Error
                    error/=myTraining->getNumOfPatterns();
                    resultsFile << "\t\t\t" << error;

                    error=0.0;
```

```
                    //run Test File
                    for (int i=0; i<myTest->getNumOfPatterns(); i++)
                    {
                          //calculate output in hidden layer neurons
                          for (int j=0; j<myHiddenNeurons.size(); j++)
                          {
                                myHiddenNeurons[j].calculate(myTest-
>getInputs(i));
                          }


                          //calculate the output in output layer
neurons
                          for (int j=0; j<myOutputNeurons.size(); j++)
                          {

      myOutputNeurons[j].calculate(myHiddenNeurons);
                                //finding error
                                error += pow( myTest-
>getInputValueOfPattern(i,j) - myOutputNeurons[j].getValue(), 2);
                          }

                    }

                    //Final Test Error
                    error/=myTest->getNumOfPatterns();
                    resultsFile << "\t\t\t" << error  << "\n";

              }

              //closing file
              resultsFile.close();

              //writing weights to file
              printWeights();

                  for (int i=0; i<psspFile->getNumOfPatterns(); i++)
                    {

                              myFile >> temp;
                              script << temp;
                              script << " ";
                              myFile >> temp;
                              script << temp;
                              script << " ";
                              myFile >> temp;
                              script << temp;
                              script << endl;

                              myFile >> temp;
                              script << temp;
                              script << "           ";
                              myFile >> temp;
                              script << temp;
                              script << endl;

                              myFile >> temp;
                              script << temp;
                              script << "         ";
                              myFile >> temp;
```

```
                                script << temp;
                                script << endl;

                                script<< "predictedSecondaryStructure:";

                        // cout<<"kati"<<endl;
                                for(int k=0;k<psspFile-
>getproteinsize(i);k++){
                         //calculate output in hidden layer neurons
                                    /*vector <double> tem;
                                        tem=psspFile->getInputs(i,k);
                                                for(int
s=0;s<tem.size();s++)
                                                        cout<<
tem.at(s)<<endl;
                               */for (int j=0; j<myHiddenNeurons.size();
j++)
                               {

                                     myHiddenNeurons[j].calculate(psspFile-
>getInputs(i,k));
                               }


                               //calculate the output in output layer
neurons
                                       int ot=0;
                               for (int j=0; j<myOutputNeurons.size(); j++)
                               {

      myOutputNeurons[j].calculate(myHiddenNeurons);


                                                    if(j>0 &&
myOutputNeurons[j].getValue()>myOutputNeurons[j-1].getValue()){
                                                        ot=j;
                                                    }
                                        }
                                          if(ot==0)
                                                            script<<"H";
                                           else if (ot==1)
                                                            script<<"L";
                                           else
                                                            script<<"E";

                                        //error += pow( myTest-
>getInputValueOfPattern(i,j) - myOutputNeurons[j].getValue(), 2);


                                    }
                                script<<endl;
                                 myFile >> temp;
                           }


        script.close();
        myFile.close();



      }
```

```
	void print()
	{
		cout << "This is a NeutralNet!" << endl;
	}


	void printWeights()
	{
		//Writing (weights.dat)
		ofstream weightsFile("weights.dat",ios::out);

		for (int i=0; i<myOutputNeurons.size(); i++)
		{
			weightsFile << "Ouput Neuron No: " << i << endl;

			//for each neuron
			for (int j=0; j<myHiddenNeurons.size(); j++)
			{
				/// Print the weights of neuron
				weightsFile << "\t"  <<
myOutputNeurons[i].getW(j) << "\n";

			}
			weightsFile << endl;

		}

		weightsFile.close();

	}




};

#endif
```

hiddeneuron.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "hiddenneuron.h"

using namespace std;
using std::ifstream;


hiddenneuron::hiddenneuron(int nOfC, vector<double>
tempCenters,double sigma){

	numOfCenters=nOfC;
	value=0.0;
	diaspora=sigma;

	for (int i=0; i<numOfCenters; i++)
	{
		hnCenters.push_back(tempCenters[i]);
	}

}


/* calulate output */
void
hiddenneuron::calculate(vector<double> inputs)
{
	value = calculateGausian(inputs);
}


//calculate gausian
double
hiddenneuron::calculateGausian(vector<double> inputs)
{
	return exp( -calculateDistance(inputs) / pow(diaspora, 2) );
}


//calculate distance
double
hiddenneuron::calculateDistance(vector<double> inputs)
{
	double distance = 0;

	for (int i=0; i<hnCenters.size(); i++)
	{
		distance+= pow((inputs[i] - hnCenters[i]),2);
	}

	return distance;
}
```

```
//change center pos
void
hiddenneuron::changeCenterPos(double learningRate,
vector<outputneuron> outputs, vector<double> inputs, vector<int>
wantedOutputs)
{
	double sum;

	for (int i=0; i<hnCenters.size(); i++)
	{
		sum = 0;
		for (int j=0; j<outputs.size(); j++)
		{
			sum += wantedOutputs[j] - outputs[j].getValue() *
calculateGausian(inputs) * (inputs[i] - hnCenters[i]) / pow(diaspora,
2);
		}
		hnCenters[i] += learningRate * sum;
	}

}

//change Diapora
void
hiddenneuron::changeDiaspora(double learningRate,
vector<outputneuron> outputs, vector<double> inputs, vector<int>
wantedOutputs)
{
	double sum = 0;

	for (int i=0; i<outputs.size(); i++)
	{
		sum += wantedOutputs[i] - outputs[i].getValue() *
calculateGausian(inputs) * (pow(calculateDistance(inputs), 2) /
pow(diaspora, 3));
	}

	diaspora += learningRate * sum;
}
```

hiddenneuron.h

```
#ifndef HIDDENNEURON_H
#define HIDDENNEURON_H

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>

#ifndef OUTPUTNEURON_H
#include "outputneuron.h"
#endif

using namespace std;
using std::ifstream;

class outputneuron;
class hiddenneuron {

private:
	vector<double> hnCenters;
	int numOfCenters;
	double value;
	//gaussian width
	double diaspora;

public:
	hiddenneuron(int nOfC, vector<double> tempCenters, double
sigma);

	//Set Values


	void addCenter(double value){
		hnCenters.push_back(value);
	}

	void setCenter(double value, int position){
		hnCenters.at(position)=value;
	}

	void setValue(double d){
		value=d;
	}

	void setDiaspora(double d){
		diaspora=d;
	}


	//Get Values


	double getCenter(int i){
		return hnCenters.at(i);
	}
```

```
	double getValue(){
		return value;
	}

	double getDiaspora(){
		return diaspora;
	}

	int getNumOfCenters(){
		return numOfCenters;
	}


	void calculate(vector<double> inputs);
	double calculateGausian(vector<double> inputs);
	double calculateDistance(vector<double> inputs);
	void changeCenterPos(double learningRate, vector<outputneuron>
outputs, vector<double> inputs, vector<int> wantedOutputs);
	void changeDiaspora(double learningRate, vector<outputneuron>
outputs, vector<double> inputs, vector<int> wantedOutputs);

};

#endif
```

outputneuron.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sstream>
#include "outputneuron.h"

using namespace std;
using std::ifstream;

outputneuron::outputneuron(int nOfWs){
	numOfWs=nOfWs;

	value=0.0;

	for (int i=0; i<numOfWs; i++)
	{
		/* generate random number and tranform between 0 and 1:
*/
		double temp = rand() % 10000 + 1;
		temp/=10000;
		//cout << temp << endl;
		w.push_back(temp);
	}
	//cout << endl;
}
/* calulate output */
void
outputneuron::calculate(vector<hiddenneuron> hiddenNeurons)
{
	//adding bias
	value = w[0];

	//hiddenNeurons.size() = numOfws
	for (int i=0; i<hiddenNeurons.size(); i++)
	{
		value+=hiddenNeurons[i].getValue()*w[i+1];
	}

}
/* change weights */
void
outputneuron::changeWs(vector<hiddenneuron> hiddenNeurons, double
learningRate, double wantedOutput)
{
	//change bias
	w[0] += learningRate * (wantedOutput - value);

	for (int i=0; i<hiddenNeurons.size(); i++)
	{
		w[i+1] += learningRate * (wantedOutput - value) *
hiddenNeurons[i].getValue();
}

}
```

outputneuron.h

```
#ifndef OUTPUTNEURON_H
#define OUTPUTNEURON_H

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#ifndef HIDDENNEURON_H
#include "hiddenneuron.h"
#endif

using namespace std;
using std::ifstream;

class hiddenneuron;
class outputneuron {

private:
	vector< double> w;
	int numOfWs;
	double value;

public:
	outputneuron(int nOfWs);
	void addW(double value){
		w.push_back(value);
	}

	void setW(double value, int position){
		w.at(position)=value;
	}

	void setValue(double d){
		value=d;
	}
	double getW(int i){
		return w.at(i);
	}

	double getValue(){
		return value;
	}

	int getNumOfWs(){
		return numOfWs;
	}
	void calculate(vector<hiddenneuron> hiddenNeurons);
	void changeWs(vector<hiddenneuron> hiddenNeurons, double
learningRate, double wantedOutput);



};

#endif
```

centerVector.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "centerVector.h"

using namespace std;
using std::ifstream;



centerVector::centerVector ()
{
	//do nothing
}
```

centerVector.h

```
#ifndef CENTERVECTOR_H_INCLUDED
#define CENTERVECTOR_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>

using namespace std;
using std::ifstream;

class centerVector {

private:
      vector<double> inputs;
        int x;
        int y;

public:

      centerVector ();

      //Set Values
      void addInput(double value)
      {
            inputs.push_back(value);
      }


      double setInput(int position,double value)
      {
            inputs.at(position)=value;
      }

        double addX(int value)
      {
            x=value;
      }

        double addY(int value)
      {
            y=value;
      }

      //Get Values
      double getInput(int position)
      {
            return inputs.at(position);
      }


      int getNumOfInputs()
      {
            return inputs.size();
      }
```

```
        vector <double> getInputs()
        {
              return inputs;
        }

           /*vector <double> find(int xpos,int ypos){
               int counter=0;
              for(int i=0;i<300;i++){
                  counter++;
                  if(i==xpos){
                      for(int j=0;j<300;j++)
                          if(j==ypos){
                              return inputs;
                          }
                          else if(j>ypos)
                              break;
                  }
                  else if(i>ypos)
                      break;
              }
           }*/

        void print()
        {
                  cout << "\n ThesiX" << ": " << x << endl;
                  cout << "\t ThesiY" << ": " << y << endl;
              cout << " -> centerVector:" << endl;

              for (int i=0; i<inputs.size(); i++){
                    cout << "\t Input" << i << ": " << inputs.at(i) <<
endl;
                  }
        }


};


#endif
```

center.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "centers.h"
#include <cstdlib>

using namespace std;
using std::ifstream;



centers::centers (string s,int numOfIn)
{
	fileName=s;
	int tempI=0;
	  int counter=0;

	char temp[150];
	  char * split;

	  char temp2[150];


	ifstream myFile;
	  myFile.open("weightsKoh.dat");
	   myFile >> temp;
	while(!myFile.eof())
	{

		   myFile >> temp;
		   myFile >> temp;
		  // myFile >> temp;

		   myCentersFile.push_back(centerVector());
		   myCentersFile.back().addX(atoi(temp));

		   myFile >> temp;
		   myFile >> temp;

		   myCentersFile.back().addY(atoi(temp));

		  myFile >> temp;

		    counter=0;
		    //myCenterVectors.push_back(centerVector());
		     while(counter<3){

			  counter++;

myCentersFile.back().addInput(strtod(temp,NULL));
			  myFile >> temp;

		     }
```

```
                //myFile >> temp;


        }
        myFile.close();

          counter=0;
          for(int i=0;i<100;i++){
              for(int j=0;j<100;j++){
                  if((i==3 && j==10) /*|| (i==150 && j==150)*/){
                       myCenterVectors.push_back(centerVector());
                       myCenterVectors.back().addX(i);
                       myCenterVectors.back().addY(j);
                       vector <double> temp;
                       temp=getInputs2(counter);
                       for(int k=0;k<temp.size();k++)
                          myCenterVectors.back().addInput(temp.at(k));
                       //print();
                  }
                  counter++;
              }
          }

           counter=0;
          for(int i=0;i<100;i++){
              for(int j=0;j<100;j++){
                  if(i==90 && j==30){
                       myCenterVectors.push_back(centerVector());
                       myCenterVectors.back().addX(i);
                       myCenterVectors.back().addY(j);
                       vector <double> temp;
                       temp=getInputs2(counter);
                       for(int k=0;k<temp.size();k++)
                          myCenterVectors.back().addInput(temp.at(k));
                       //print();
                  }
                  counter++;
              }
          }

          counter=0;
          for(int i=0;i<100;i++){
              for(int j=0;j<100;j++){
                  if(i==6 && j==85){
                       myCenterVectors.push_back(centerVector());
                       myCenterVectors.back().addX(i);
                       myCenterVectors.back().addY(j);
                       vector <double> temp;
                       temp=getInputs2(counter);
                       for(int k=0;k<temp.size();k++)
                          myCenterVectors.back().addInput(temp.at(k));
                       //print();
                  }
                  counter++;
              }
          }

}
```

```
centers::centers (vector<double> max,vector<double> min,int
numOfIn,int numOfCV)
{
	for (int p=0; p<numOfCV; p++)
		myCenterVectors.push_back(centerVector());

	//create random centers
	for (int p=0; p<myCenterVectors.size(); p++)
	{

		for (int i=0; i<numOfIn; i++)
		{
			int pedio=(max[i]-min[i]) * 100000;
			double temp = rand() % pedio + 1;
			temp/=100000;
			temp+=min[i];
			myCenterVectors[p].addInput(temp);
			//cout << "Min: " <<  min[i] << "\tRandom: " <<
temp  << "\tMax: " << max[i] << endl;
		}

	}


}
```

center.h

```
#ifndef CENTERS_H_INCLUDED
#define CENTERS_H_INCLUDED
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "centerVector.h"

using namespace std;
using std::ifstream;

class centers {

private:
	vector <centerVector> myCenterVectors;
	  vector <centerVector> myCentersFile;
	string fileName;

public:
	centers (string s,int numOfIn);
	centers (vector<double> max,vector<double> min,int numOfIn,int
numOfCV);

	void print()
	{
		cout << "Center: " << fileName << "\n";
		for (int i=0; i<myCenterVectors.size(); i++){
			cout << "\t" << i;
			myCenterVectors.at(i).print();
		}
	}
	int getNumOfCenterVectors()
	{
		return myCenterVectors.size();
	}
	int getNumOfInputsInCenterVector(int cvPos)
	{
		return myCenterVectors.at(cvPos).getNumOfInputs();
	}

	double getInputValueOfCenterVector(int cvPos, int inputPos)
	{
		return myCenterVectors.at(cvPos).getInput(inputPos);
	}
	vector <double> getInputs(int cvPos)
	{
		return myCenterVectors.at(cvPos).getInputs();
	}

	  vector <double> getInputs2(int cvPos)
	{
		return myCentersFile.at(cvPos).getInputs();
	}
};
#endif
```

pattern.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "pattern.h"

using namespace std;
using std::ifstream;



pattern::pattern ()
{
	//do nothing
}
```

pattern.h

```
#ifndef PATTERN_H_INCLUDED
#define PATTERN_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>

using namespace std;
using std::ifstream;

class pattern {

private:
	vector<double> inputs;
	vector<int> outputs;

public:

	pattern ();

	//Set Values
	void addInput(double value)
	{
		inputs.push_back(value);
	}

	void addOutput(int value)
	{
		outputs.push_back(value);
	}

	double setInput(int position,double value)
	{
		inputs.at(position)=value;
	}

	double setOutput(int position,int value)
	{
		outputs.at(position)=value;
	}

	//Get Values
	double getInput(int position)
	{
		return inputs.at(position);
	}

	double getOutput(int position)
	{
		return outputs.at(position);
	}


	int getNumOfInputs()
```

```
	{
		return inputs.size();
	}


	int getNumOfOutputs()
	{
		return outputs.size();
	}

	vector <double> getInputs()
	{
		return inputs;
	}

	vector <int> getOutputs()
	{
		return outputs;
	}

	void print()
	{
		cout << " -> pattern:" << endl;


		for (int i=0; i<outputs.size(); i++)
			cout << "\t Output" << i << ": " << outputs.at(i)
<< endl;


		for (int i=0; i<inputs.size(); i++)
			cout << "\t Input" << i << ": " << inputs.at(i) <<
endl;
	}


};


#endif
```

pssp.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "epoch.h"
#include <sstream>
#include <algorithm>
#include <iterator>

using namespace std;
using std::ifstream;

#include "pssp.h"

pssp::pssp (string s)
{
	fileName=s;
	int tempI=0;
	  int counter=0;

	char temp[4000];
	  char * split;



	  char secondaryStructure[4000];

	  vector <pattern> protein;

	//Reading File

	ifstream myFile;
	  myFile.open("Z_Filter_Output.txt");
	  myFile >> temp;
	while(!myFile.eof())
	{
		  myFile >> temp;
		  myFile >> temp;
		  myFile >> temp;
		  myFile >> temp;
		  myFile >> temp;
		   for(int i=0;i<4000;i++){
		       temp[i]='\0';
		       secondaryStructure[i]='\0';
		   }
		  myFile >> temp;
		  for(int i=1;i<500;i++){
		      if (temp[i]=='\0')
		          break;
		      secondaryStructure[i-1]=temp[i];
		  }


		  for(int i=0;i<500;i++){
		      //counter++;
```

```
                    if (secondaryStructure[i]=='\0')
                        break;
                    else{
                        protein.push_back(pattern());
                        if (secondaryStructure[i]=='H')
                            protein.back().addOutput(1);
                        else
                            protein.back().addOutput(0);
                        if (secondaryStructure[i]=='E')
                            protein.back().addOutput(1);
                        else
                            protein.back().addOutput(0);
                        if (secondaryStructure[i]=='L')
                            protein.back().addOutput(1);
                        else
                            protein.back().addOutput(0);
                        }
                }
            myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;

                  split = strtok (temp,",");
                  counter=0;
                  while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                      split = strtok (NULL,",");
                      counter++;
                  }
                  myFile >> temp;
                  myFile >> temp;
                  split = strtok (temp,",");
                  counter=0;
                  while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                      split = strtok (NULL,",");
                      counter++;
                  }
                  myFile >> temp;
                  myFile >> temp;
                  split = strtok (temp,",");
                  counter=0;
                  while(split!=NULL){

protein.at(counter).addInput(strtod(split,NULL));
                      split = strtok (NULL,",");
                      counter++;
                  }
                  //tempI=counter;
                  sequence.push_back(protein);
                  myFile >> temp;
                  protein.clear();

      }
      myFile.close();
}
```

pssp.h

```
#ifndef PSSP_H_INCLUDED
#define     PSSP_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "pattern.h"

using namespace std;
using std::ifstream;

class pssp {

private:
      vector <vector <pattern> >  sequence;

      string fileName;
      vector <double> minInputsInColumn;
      vector <double> maxInputsInColumn;

public:

      pssp (string s);

        int getNumOfPatterns()
      {
            return sequence.size();
      }

         int getproteinsize(int value)
      {
            return sequence.at(value).size();
      }

         vector <double> getInputs(int i,int j)
      {
            return sequence.at(i).at(j).getInputs();
      }

      vector <int> getOutputs(int i,int j)
      {
            return sequence.at(i).at(j).getOutputs();
      }


};
#endif
```

epoch.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "epoch.h"
#include <sstream>
#include <algorithm>
#include <iterator>

using namespace std;
using std::ifstream;



epoch::epoch (string s,int numOfIn,int numOfOut)
{
	fileName=s;
	int tempI=0;
	  int counter=0;

	char temp[4000];
	  char * split;

	double stddevIN[numOfIn];
	double stddevOUT[numOfOut];
	double averageIN[numOfIn];
	double averageOUT[numOfOut];

	  char secondaryStructure[4000];


	for (int i=0; i<numOfIn; i++)
	{
		averageIN[i]=0;
		stddevIN[i]=0;
		maxInputsInColumn.push_back(0);
		minInputsInColumn.push_back(0);
	}

	for (int i=0; i<numOfOut; i++)
	{
		averageOUT[i]=0;
		stddevOUT[i]=0;
	}


	//Reading File

	ifstream myFile;
	  myFile.open("Z_Filter_Output.txt");
	  myFile >> temp;
	while(!myFile.eof())
	{
			myFile >> temp;
			myFile >> temp;
```

```
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                for(int i=0;i<4000;i++){
                     temp[i]='\0';
                     secondaryStructure[i]='\0';
                 }
                myFile >> temp;
                for(int i=1;i<500;i++){
                     if (temp[i]=='\0')
                        break;
                    secondaryStructure[i-1]=temp[i];
                }


                for(int i=0;i<500;i++){
                    //counter++;
                    if (secondaryStructure[i]=='\0')
                        break;

                    else{
                        myPatterns.push_back(pattern());
                        if (secondaryStructure[i]=='H')
                            myPatterns.back().addOutput(1);
                        else
                            myPatterns.back().addOutput(0);
                        if (secondaryStructure[i]=='L')
                            myPatterns.back().addOutput(1);
                        else
                            myPatterns.back().addOutput(0);
                        if (secondaryStructure[i]=='E')
                            myPatterns.back().addOutput(1);
                        else
                            myPatterns.back().addOutput(0);
                        }
                }
                //print();
            myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;
                myFile >> temp;

                  split = strtok (temp,",");
                  counter=tempI;
                  while(split!=NULL){

myPatterns.at(counter).addInput(strtod(split,NULL));
                      split = strtok (NULL,",");
                      counter++;
                  }

                  myFile >> temp;
                  myFile >> temp;
                  split = strtok (temp,",");
                  counter=tempI;
                  while(split!=NULL){

myPatterns.at(counter).addInput(strtod(split,NULL));
                      split = strtok (NULL,",");
```

```
                    counter++;
                }

                myFile >> temp;
                myFile >> temp;
                split = strtok (temp,",");
                counter=tempI;
                while(split!=NULL){

myPatterns.at(counter).addInput(strtod(split,NULL));
                    split = strtok (NULL,",");
                    counter++;
                }
                tempI=counter;
                myFile >> temp;


        }
        myFile.close();

}
```

epoch.h

```
#ifndef EPOCH_H_INCLUDED
#define EPOCH_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "pattern.h"

using namespace std;
using std::ifstream;

class epoch {

private:
	vector <pattern> myPatterns;
	string fileName;
	vector <double> minInputsInColumn;
	vector <double> maxInputsInColumn;

public:

	epoch (string s,int numOfIn,int numOfOut);

	void print()
	{
		cout << "Epoch: " << fileName << "\n";
		for (int i=0; i<myPatterns.size(); i++){
			cout << "\t" << i;
			myPatterns.at(i).print();
		}
	}


	void printMaxMin()
	{
		cout << "max: " << "\n";
		for (int i=0; i<maxInputsInColumn.size(); i++){
			cout << "\t" << maxInputsInColumn[i] << endl;
		}
		cout << endl;

		cout << "min: " << "\n";
		for (int i=0; i<minInputsInColumn.size(); i++){
			cout << "\t" << minInputsInColumn[i] << endl;
		}
		cout << endl;
	}


	int getNumOfPatterns()
	{
		return myPatterns.size();
	}
```

```
        int getNumOfInputsInPattern(int patternPos)
        {
              return myPatterns.at(patternPos).getNumOfInputs();
        }

        double getInputValueOfPattern(int patternPos, int inputPos)
        {
              return myPatterns.at(patternPos).getInput(inputPos);
        }

        int getNumOfOutputsInPattern(int patternPos)
        {
              return myPatterns.at(patternPos).getNumOfOutputs();
        }

        double getOutputValueOfPattern(int patternPos, int outputPos)
        {
              return myPatterns.at(patternPos).getOutput(outputPos);
        }



        vector <double> getInputs(int patternPos)
        {
              return myPatterns.at(patternPos).getInputs();
        }

        vector <int> getOutputs(int patternPos)
        {
              return myPatterns.at(patternPos).getOutputs();
        }

        vector <double> getMinInputsInColumn()
        {
              return minInputsInColumn;
        }

        vector <double> getMaxInputsInColumn()
        {
              return maxInputsInColumn;
        }


};

#endif
```

parameters.cpp

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>
#include "parameters.h"

using namespace std;
using std::ifstream;
parameters::parameters (string s="parameters.txt") {

	//initialize
	numHiddenLayerNeurons=-1;
	numInputNeurons=-1;
	numOutputNeurons=-1;
	learningRateW=-1;
	learningRateR=-1;
	learningRateS=-1;
	sigmas=-1;
	maxIterations=-1;
	trainFile="";
	testFile="";
	fileName=s;
	//tempory Vars
	int tempI;
	double tempD;
	string tempS;
	//Reading Parameters
	ifstream parametersFile(fileName.c_str(),ios::in);
	parametersFile >> tempS; parametersFile >> tempI;
setNumHiddenLayerNeurons(tempI);
	parametersFile >> tempS; parametersFile >> tempI;
setNumInputNeurons(tempI);
	parametersFile >> tempS; parametersFile >> tempI;
setNumOutputNeurons(tempI);
	parametersFile >> tempS; parametersFile >> tempD;
setLearningRateW(tempD);
	parametersFile >> tempS; parametersFile >> tempD;
setLearningRateR(tempD);
	parametersFile >> tempS; parametersFile >> tempD;
setLearningRateS(tempD);
	parametersFile >> tempS; parametersFile >> tempD;
setSigmas(tempD);
	parametersFile >> tempS; parametersFile >> tempI;
setMaxIterations(tempI);
	parametersFile >> tempS; parametersFile >> tempS;
setCentersFile(tempS);
	parametersFile >> tempS; parametersFile >> tempS;
setTrainFile(tempS);
	parametersFile >> tempS; parametersFile >> tempS;
setTestFile(tempS);
	parametersFile.close();
}
```

parameters.h

```
#ifndef PARAMETERS_H_INCLUDED
#define PARAMETERS_H_INCLUDED

#include <iostream>
#include <fstream>
#include <string.h>
#include <vector>
#include <stdio.h>
#include <math.h>
#include <sstream>

using namespace std;
using std::ifstream;

class parameters {

private:
	int numHiddenLayerNeurons;
	int numInputNeurons;
	int numOutputNeurons;
	double learningRateW;
	double learningRateR;
	double learningRateS;
	double sigmas;
	int maxIterations;
	string centersFile;
	string trainFile;
	string testFile;
	string fileName;


public:
	parameters (string s);

	//Set Parameters


	void setNumHiddenLayerNeurons(int i)
	{
		numHiddenLayerNeurons=i;
	}

	void setNumInputNeurons(int i)
	{
		numInputNeurons=i;
	}

	void setNumOutputNeurons(int i)
	{
		numOutputNeurons=i;
	}

	void setLearningRateW(double d)
	{
		learningRateW=d;
	}

	void setLearningRateR(double d)
```

```
{
	learningRateR=d;
}

void setLearningRateS(double d)
{
	learningRateS=d;
}

void setSigmas(double d)
{
	sigmas=d;
}

void setMaxIterations(int i)
{
	maxIterations=i;
}

void setCentersFile(string s)
{
	centersFile=s;
}

void setTrainFile(string s)
{
	trainFile=s;
}

void setTestFile(string s)
{
	testFile=s;
}


//Get Parameters


int getNumHiddenLayerNeurons()
{
	return numHiddenLayerNeurons;
}

int getNumInputNeurons()
{
	return numInputNeurons;
}

int getNumOutputNeurons()
{
	return numOutputNeurons;
}

double getLearningRateW()
{
	return learningRateW;
}

double getLearningRateR()
{
	return learningRateR;
```

```
	}

	double getLearningRateS()
	{
		return learningRateS;
	}

	double getSigmas()
	{
		return sigmas;
	}

	int getMaxIterations()
	{
		return maxIterations;
	}

	string getCentersFile()
	{
		return centersFile;
	}

	string getTrainFile()
	{
		return trainFile;
	}

	string getTestFile()
	{
		return testFile;
	}

	void print()
	{
		cout << "Parameters: " << fileName << endl;
		cout << "\t numHiddenLayerNeurons: " <<
numHiddenLayerNeurons << endl;
		cout << "\t numInputNeurons: " << numInputNeurons <<
endl;
		cout << "\t numOutputNeuron: " << numOutputNeurons <<
endl;
		cout << "\t learningRateW: " << learningRateW << endl;
		cout << "\t learningRateR: " << learningRateR << endl;
		cout << "\t learningRateS: " << learningRateS << endl;
		cout << "\t sigmas: " << sigmas << endl;
		cout << "\t maxIterations: " << maxIterations << endl;
		cout << "\t centersFile: " << centersFile << endl;
		cout << "\t trainFile: " << trainFile << endl;
		cout << "\t testFile: " << testFile << endl;
	}

};


#endif
```

Παράρτημα Ζ

Γενετικοί Αλγόριθμοι

pssp_ucy.cpp

```
//included libraries

#include "DataReader.h"
#include "Neuron.h"
#include "BidirectionalRecurrentNeuralNetwork.h"
#include "OutputData.h"
#include "ga.h"
#include "chromosome.h"
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>

#define POP 200
using namespace std;

int main(int argc, char* argv[])
{
	//initiate random function - the same random values every time
	srand(time(NULL));

	//variables of the main program
	char trainFile[100];
	char testFile[100];
	char inputProfile[100];
	char outputProfile[100];
	char msaFile[100];
	int primaryStructureSize;
	int msaEnable;
	int randomizeDataset;
	float parameters[17];
	int epoch=0;
	bool endOfFile=true;
	vector<double> outputresult1;
	vector<double> outputresult2;
	vector<double> outputresult3;
	vector<double> outputresult4;
	vector<double> outputresult5;
	vector<double> outputresult6;

	vector<vector <double> > vectoroutputresult1;
	vector<vector <double> > vectoroutputresult2;
	vector<vector <double> > vectoroutputresult3;
	vector<vector <double> > vectoroutputresult4;
	vector<vector <double> > vectoroutputresult5;
	vector<vector <double> > vectoroutputresult6;
	vector<vector <double> > vectoroutputresultensemble;


	vector<double> ensambleAverage;
	char secSt;
```

```
	double maxVal;
	int maxIndex;
	double tempadditionprotein;
	vector<double> ensambleResults;

	vector<char> primaryStructure;
	vector<char> secondaryStructure;
	vector<char> predictedSecondaryStructure;
	char proteinID[100];
	time_t rawtime;
	struct tm * timeinfo;
	time ( &rawtime );
	timeinfo = localtime ( &rawtime );
	string tempstr="DataOut_";
	string oFolder=asctime (timeinfo);
	string  oFolder1;
	string  oFolder2;
	string  oFolder3;
	string  oFolder4;
	string  oFolder5;
	char outputFolder[500];
	char scriptcommand[500];
	char scriptcommand2[500];
	char viterbicommand[500];

	int center_window_size;
	char outputfolderName[100];

		for(int i=0;i<500;i++){
					outputfolderName[i]='\0';
					scriptcommand2[i]='\0';
		}
	//object of the DataReader class that reads the program's
parameters from the parameter file
		DataReader *getInput=new DataReader();

	//Parameters' entry
	getInput-
>readParameters(parameters,inputProfile,outputProfile,trainFile,testF
ile,&msaEnable,&randomizeDataset,&center_window_size,outputfolderName
);


			//oFolder.insert(0,tempstr);
			//oFolder.insert(oFolder.length()-1,"//");

			oFolder1.insert(0,"mkdir ");

	oFolder1.insert(oFolder1.length(),outputfolderName);
			oFolder1.insert(oFolder1.length(),"//");
			for(int i=0;i<oFolder1.length();i++){
				outputFolder[i]=oFolder1[i];
				}
			for(int i=oFolder1.length()-1;i<500;i++){

	outputFolder[i]='\0';
								}

			char temp;
			/*for(int i=0;i<oFolder.length();i++){
				temp=oFolder[i];
```

```
                        if(temp==' ' && i!=5)
                              outputFolder[i]='-';
                        else
                        outputFolder[i]=oFolder[i];
                  }
                  for(int i=oFolder.length()-1;i<500;i++){
                                    outputFolder[i]='\0';
                              }*/
                  system(outputFolder);
                  oFolder2.insert(0,oFolder1);
                  oFolder2.insert(oFolder2.length()-
2,"/ScriptResults//");

                  for(int i=0;i<oFolder2.length();i++){
                        outputFolder[i]=oFolder2[i];
                  }
                  for(int i=oFolder2.length()-1;i<500;i++){
                        outputFolder[i]='\0';
                        }

                  system(outputFolder);

                  oFolder1.erase(0,6);
                  oFolder2.erase(0,6);
                  oFolder2.erase(oFolder2.length(),6);
                  oFolder3.insert(0,oFolder1);
                  oFolder3.insert(oFolder3.length()-2,"/Z_Output.txt
");

                  oFolder4.insert(0,oFolder1);
                  oFolder4.insert(oFolder4.length()-2,"/Z_Output.txt
"+oFolder1+"/Z_Output.txt >"+oFolder1+"/ScriptResults/viterbi.txt");
                  oFolder4.insert(0,"perl viterbi.pl ");
                  for(int i=0;i<oFolder4.length();i++){
                        viterbicommand[i]=oFolder4[i];
                                                      }
                  for(int i=oFolder4.length()-2;i<500;i++){
                        viterbicommand[i]='\0';
                                                            }
/*************************************
 * confusion Matrices after filtering
 */
                  oFolder5.insert(0,oFolder1);
                  oFolder5.insert(oFolder5.length()-
2,"/ScriptResults/viterbi.txt
"+oFolder1+"/ScriptResults/filtered_res.txt
>"+oFolder1+"/ScriptResults/finalreport.txt");
                  oFolder5.insert(0,"perl
confusionMatricesAndSOVopt.pl ");
                                    for(int
i=0;i<oFolder5.length();i++){

      scriptcommand2[i]=oFolder5[i];

      }
                                    for(int i=oFolder5.length()-
2;i<500;i++){
                                          scriptcommand2[i]='\0';

            }
```

```
                  oFolder3.insert(oFolder3.length()-2,oFolder2);

                  oFolder3.insert(oFolder3.length()-6,"/results.txt
>"+oFolder2+"info.txt");

                  oFolder3.insert(0,"perl confusionMatricesAndSOV.pl
");

                  for(int i=0;i<oFolder3.length();i++){

      scriptcommand[i]=oFolder3[i];
                              }
                  for(int i=oFolder3.length()-6;i<500;i++){
                                    scriptcommand[i]='\0';
                                    }
                  //system(scriptcommand2);



                  //oFolder.erase(0,6);
                  /*for(int i=0;i<oFolder.length();i++){
                              temp=oFolder[i];
                              if(temp==' ' && i!=5)
                                    outputFolder[i]='-';
                              else
                              outputFolder[i]=oFolder[i];
                        }
                        for(int i=oFolder.length()-
1;i<100;i++){
                                          outputFolder[i]='\0';
                              }*/

      for(int i=0;i<500;i++){
                  if(outputfolderName[i]=='\0'){
                        outputfolderName[i]='/';
                        break;
                  }
                                                      }
      OutputData* saveOutputData=new
OutputData('Z',outputfolderName);

       OutputData* OutputDataForFilter=new
OutputData("Z_Filter",outputfolderName);

      saveOutputData->createEnsembleOutputFile();

      for(int i=0;i<3;i++)
      {
            outputresult1.push_back(0);
            outputresult2.push_back(0);
            outputresult3.push_back(0);
            outputresult4.push_back(0);
            outputresult5.push_back(0);
            outputresult6.push_back(0);
            ensambleAverage.push_back(0);
      }


      //initiate the program's parameter
      int hLayerOneSize = parameters[0];
```

```
      int hLayerTwoSize = parameters[1];
      int hLayerOneSizeB = parameters[2];
      int hLayerTwoSizeB = parameters[3];
      int hLayerOneSizeF = parameters[4];
      int hLayerTwoSizeF = parameters[5];
      int activationFuncTypeHidden = parameters[6];
      int activationFuncTypeOutput = parameters[7];
      double initialLearningRate = parameters[8];
      double momentum = parameters[9];
      int windowSize = parameters[10];
      double qMinus1 = parameters[11];
      double qPlus1 = parameters[12];
      int errorFunctionTypeHidden = parameters[13];
      int errorFunctionTypeOutput = parameters[14];
      int s = parameters[15];
      int maxIterations = parameters[16];
      double learningRate=initialLearningRate;

        ga *gAlgorithm=new ga();

      //creation of the Bidirectional Recurrent Neural Network with
the specific parameters
      //takes the details of the msa file
      BidirectionalRecurrentNeuralNetwork BRNN1 =
BidirectionalRecurrentNeuralNetwork('A',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      /*BidirectionalRecurrentNeuralNetwork BRNN2 =
BidirectionalRecurrentNeuralNetwork('B',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      BidirectionalRecurrentNeuralNetwork BRNN3 =
BidirectionalRecurrentNeuralNetwork('C',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,
```

```
      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      BidirectionalRecurrentNeuralNetwork BRNN4 =
BidirectionalRecurrentNeuralNetwork('D',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      BidirectionalRecurrentNeuralNetwork BRNN5 =
BidirectionalRecurrentNeuralNetwork('E',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);

      BidirectionalRecurrentNeuralNetwork BRNN6 =
BidirectionalRecurrentNeuralNetwork('F',hLayerOneSize,

      hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutpu
t,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,
s,maxIterations,trainFile,testFile,

      inputProfile,outputProfile,msaEnable,randomizeDataset,outputfol
derName);*/

      //the main loop of the program.
      tempadditionprotein=0.0;//for ensamble per protein

      while(epoch<maxIterations)//epoch mean generation
      {
            cout << "Generation: ";
            cout<<epoch;
            cout<<endl;
            for(int atom=0;atom<POP;atom++){

                  BRNN1.setWeights_ga(gAlgorithm->population[atom]);

            tempadditionprotein=0.0;//for ensamble per protein

            //open pointers to the output files
```

```
		BRNN1.openFolders();
		/*BRNN2.openFolders();
		BRNN3.openFolders();
		BRNN4.openFolders();
		BRNN5.openFolders();
		BRNN6.openFolders();*/

		//takes the first protein of the training set
		endOfFile=BRNN1.initTrainingInput(&primaryStructureSize);
		/*BRNN2.initTrainingInput(&primaryStructureSize);
		BRNN3.initTrainingInput(&primaryStructureSize);
		BRNN4.initTrainingInput(&primaryStructureSize);
		BRNN5.initTrainingInput(&primaryStructureSize);
		BRNN6.initTrainingInput(&primaryStructureSize);*/

		learningRate = initialLearningRate * exp(-
(double)epoch/(double)(108.57));//prosthesi apo ton Antoni

		//
		while(endOfFile)
		{
			//takes the new msa encoding of this protein
			if(msaEnable)
			{
				BRNN1.getMSAInput();
				/*BRNN2.getMSAInput();
				BRNN3.getMSAInput();
				BRNN4.getMSAInput();
				BRNN5.getMSAInput();
				BRNN6.getMSAInput();*/
			}

			BRNN1.initProtainQuest();
			/*BRNN2.initProtainQuest();
			BRNN3.initProtainQuest();
			BRNN4.initProtainQuest();
			BRNN5.initProtainQuest();
			BRNN6.initProtainQuest();*/


			for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
			{
				//processing
				//cout<<primaryStructureSize<<endl;
				//cout<<sequencet<<endl;

	BRNN1.doFeedForward(sequencet,1,epoch,center_window_size);
					/*
BRNN2.doFeedForward(sequencet,1,epoch,center_window_size);

	BRNN3.doFeedForward(sequencet,1,epoch,center_window_size);

	BRNN4.doFeedForward(sequencet,1,epoch,center_window_size);

	BRNN5.doFeedForward(sequencet,1,epoch,center_window_size);

	BRNN6.doFeedForward(sequencet,1,epoch,center_window_size);*/

	//BRNN1.doBackpropagation(sequencet,learningRate,1);
```

```
	/*BRNN2.doBackpropagation(sequencet,learningRate,1);

	BRNN3.doBackpropagation(sequencet,learningRate,1);

	BRNN4.doBackpropagation(sequencet,learningRate,1);

	BRNN5.doBackpropagation(sequencet,learningRate,1);

	BRNN6.doBackpropagation(sequencet,learningRate,1);*/
			}

				/*for(int i=1;i<OtH.size();i++){
					cout<<OtH.at(i)<<", ";
				}*/

			//pernoume to error gia tin protein pou isoute me
to sinoliko lathos/ton arithmo twn aa pou exei i proteini
			BRNN1.getSequenceTrainingError();
			/*BRNN2.getSequenceTrainingError();
			BRNN3.getSequenceTrainingError();
			BRNN4.getSequenceTrainingError();
			BRNN5.getSequenceTrainingError();
			BRNN6.getSequenceTrainingError();*/

			if(epoch==maxIterations-1){

	BRNN1.printOutputSequence(1,vectoroutputresult1,0);

	/*BRNN2.printOutputSequence(1,vectoroutputresult2,0);

	BRNN3.printOutputSequence(1,vectoroutputresult3,0);

	BRNN4.printOutputSequence(1,vectoroutputresult4,0);

	BRNN5.printOutputSequence(1,vectoroutputresult5,0);

	BRNN6.printOutputSequence(1,vectoroutputresult6,0);*/
			}

			//vector<char> primaryStructure;
			//vector<char> secondaryStructure;
			//vector<char> predictedSecondaryStructure;
			//takes the next protein from the training file

	endOfFile=BRNN1.initTrainingInput(&primaryStructureSize);
			/*BRNN2.initTrainingInput(&primaryStructureSize);
			BRNN3.initTrainingInput(&primaryStructureSize);
			BRNN4.initTrainingInput(&primaryStructureSize);
			BRNN5.initTrainingInput(&primaryStructureSize);
			BRNN6.initTrainingInput(&primaryStructureSize);*/

		}//endofFile


		//



		double count=0;
		/*for(int i=0;i<ensambleResults.size();i++)
```

```
            {
                  count+=ensambleResults[i];
            }

            cout<<epoch<<"
"<<(tempadditionprotein*100)/count<<"%"<<endl;*/

            //

                gAlgorithm-
>population[atom].setEval(BRNN1.getEpochError(0));

            /*BRNN2.getEpochError();
            BRNN3.getEpochError();
            BRNN4.getEpochError();
            BRNN5.getEpochError();
            BRNN6.getEpochError();*/

            //
            BRNN1.closeFolders();
            /*BRNN2.closeFolders();
            BRNN3.closeFolders();
            BRNN4.closeFolders();
            BRNN5.closeFolders();
            BRNN6.closeFolders();*/



        }
            gAlgorithm->mutation();
             for(int atom=0;atom<POP;atom++){
                  BRNN1.setWeights_ga(gAlgorithm->next_gen[atom]);
                 BRNN1.openFolders();

            endOfFile=BRNN1.initTrainingInput(&primaryStructureSize);
            while(endOfFile)
            {
                  //takes the new msa encoding of this protein
                  if(msaEnable)
                  {
                        BRNN1.getMSAInput();
                  }

                  BRNN1.initProtainQuest();

                  for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
                  {


      BRNN1.doFeedForward(sequencet,1,epoch,center_window_size);
                  }

                  BRNN1.getSequenceTrainingError();



      endOfFile=BRNN1.initTrainingInput(&primaryStructureSize);


            }//endofFile
```

```
             BRNN1.closeFolders();
                gAlgorithm-
>next_gen[atom].setEval(BRNN1.getEpochError(0));
            //endOfFile=BRNN1.initTestInput(&primaryStructureSize);
             }


            gAlgorithm->chooseBestIndividual(POP*2);

            if(epoch==maxIterations-1){
                BRNN1.setWeights_ga(gAlgorithm->population[0]);



            //open pointers to the output files
            BRNN1.openFolders();

                endOfFile=BRNN1.initTestInput(&primaryStructureSize);
            /*BRNN2.initTestInput(&primaryStructureSize);
            BRNN3.initTestInput(&primaryStructureSize);
            BRNN4.initTestInput(&primaryStructureSize);
            BRNN5.initTestInput(&primaryStructureSize);
            BRNN6.initTestInput(&primaryStructureSize);*/

            //
            //ensambleResults.clear();
            while(endOfFile)
            {

                  vectoroutputresult1.clear();
                  /*vectoroutputresult2.clear();
                  vectoroutputresult3.clear();
                  vectoroutputresult4.clear();
                  vectoroutputresult5.clear();
                  vectoroutputresult6.clear();
                  vectoroutputresultensemble.clear();*/

                  //takes the new msa encoding of this protein
                  if(msaEnable)
                  {
                        BRNN1.getMSAInput();
                        /*BRNN2.getMSAInput();
                        BRNN3.getMSAInput();
                        BRNN4.getMSAInput();
                        BRNN5.getMSAInput();
                        BRNN6.getMSAInput();*/
                  }

                  BRNN1.initProtainQuest();
                  /*BRNN2.initProtainQuest();
                  BRNN3.initProtainQuest();
                  BRNN4.initProtainQuest();
                  BRNN5.initProtainQuest();
                  BRNN6.initProtainQuest();*/

                  predictedSecondaryStructure.clear();


                  for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
                  {
```

```
	BRNN1.doFeedForward(sequencet,2,epoch,center_window_size);

	/*BRNN2.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN3.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN4.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN5.doFeedForward(sequencet,2,epoch,center_window_size);

	BRNN6.doFeedForward(sequencet,2,epoch,center_window_size);*/

	BRNN1.retOutput(sequencet,outputresult1,&secSt);

	/*BRNN2.retOutput(sequencet,outputresult2,&secSt);

	BRNN3.retOutput(sequencet,outputresult3,&secSt);

	BRNN4.retOutput(sequencet,outputresult4,&secSt);

	BRNN5.retOutput(sequencet,outputresult5,&secSt);

	BRNN6.retOutput(sequencet,outputresult6,&secSt);*/

			vectoroutputresult1.push_back(outputresult1);

	/*vectoroutputresult2.push_back(outputresult2);
			vectoroutputresult3.push_back(outputresult3);
			vectoroutputresult4.push_back(outputresult4);
			vectoroutputresult5.push_back(outputresult5);

	vectoroutputresult6.push_back(outputresult6);*/


			//calculate the average ensemble output
			/*for(int
ensambleI=0;ensambleI<3;ensambleI++)
			{


	ensambleAverage[ensambleI]=(outputresult1[ensambleI]+outputresu
lt2[ensambleI]+outputresult3[ensambleI]+outputresult4[ensambleI]+outp
utresult5[ensambleI]+outputresult6[ensambleI])/6.0;

			}


	vectoroutputresultensemble.push_back(ensambleAverage);

			maxVal=ensambleAverage[0];
			maxIndex=0;

			for(int position=0;position<3;position++)
			{

				if (maxVal<=ensambleAverage[position])
				{
					maxVal=ensambleAverage[position];
					maxIndex=position;
```

```
                                }
                        }*/

                        /*if((maxIndex==0 && secSt=='H') ||
(maxIndex==1 && secSt=='L') || (maxIndex==2 && secSt=='E'))

      tempadditionprotein=tempadditionprotein+1.0;

                        if(maxIndex==0)

      predictedSecondaryStructure.push_back('H');
                        else if(maxIndex==1)

      predictedSecondaryStructure.push_back('L');
                        else if(maxIndex==2)

      predictedSecondaryStructure.push_back('E');*/



                }


//ensambleResults.push_back(tempadditionprotein*100.0/primaryStructur
eSize);
                //ensambleResults.push_back(primaryStructureSize);

                BRNN1.getSequenceTestingError();
                /*BRNN2.getSequenceTestingError();
                BRNN3.getSequenceTestingError();
                BRNN4.getSequenceTestingError();
                BRNN5.getSequenceTestingError();
                BRNN6.getSequenceTestingError();*/



      BRNN1.printOutputSequence(2,vectoroutputresult1,0);

      /*BRNN2.printOutputSequence(2,vectoroutputresult2,0);

      BRNN3.printOutputSequence(2,vectoroutputresult3,0);

      BRNN4.printOutputSequence(2,vectoroutputresult4,0);

      BRNN5.printOutputSequence(2,vectoroutputresult5,0);

      BRNN6.printOutputSequence(2,vectoroutputresult6,0);*/


      BRNN1.returnData(primaryStructure,secondaryStructure,proteinID)
;

                        //create the ensemble output file
                        //saveOutputData-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecond
aryStructure,proteinID,0,2,vectoroutputresult1,1);
                                //OutputDataForFilter-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecond
aryStructure,proteinID,0,2,vectoroutputresultensemble,0);
```

```
                                    //saveOutputData-
>printEnsembleOutputFile(primaryStructure,secondaryStructure,predicte
dSecondaryStructure,proteinID,vectoroutputresult1,vectoroutputresult2
,vectoroutputresult3,vectoroutputresult4,vectoroutputresult5,vectorou
tputresult6,vectoroutputresultensemble);



      endOfFile=BRNN1.initTestInput(&primaryStructureSize);
                  /*BRNN2.initTestInput(&primaryStructureSize);
                  BRNN3.initTestInput(&primaryStructureSize);
                  BRNN4.initTestInput(&primaryStructureSize);
                  BRNN5.initTestInput(&primaryStructureSize);
                  BRNN6.initTestInput(&primaryStructureSize);*/
            }
                BRNN1.closeFolders();

            }
             epoch++;

      }




      BRNN1.printOutputs();
      /*BRNN2.printOutputs();
      BRNN3.printOutputs();
      BRNN4.printOutputs();
      BRNN5.printOutputs();
      BRNN6.printOutputs();*/
      BRNN1.printNetworkSpecification();
      /*BRNN2.printNetworkSpecification();
      BRNN3.printNetworkSpecification();
      BRNN4.printNetworkSpecification();
      BRNN5.printNetworkSpecification();
      BRNN6.printNetworkSpecification();*/

      //system(scriptcommand);
      //system(viterbicommand);
        //system(scriptcommand2);

      //saveOutputData->closeEnsembleOutputFile();

      return 0;
}
```

ga.h

```
/*************************************************************************
****************************
Ylopoisi twn methodon tis klasis atomo
*************************************************************************
**************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <cmath>
#include <vector>
#include <ctime>

#include "chromosome.h"
using namespace std;
#define POP 2
#ifndef _GA_H
#define     _GA_H

class ga {
public:
   ga();
      ~ga(void);

      void createRoulette();
      void newPopulation();
      void mutation();
      void crossover();
      void chooseBestIndividual(int size);
      void setEvaluation(double eval,int position);
      void printSolution();
      void printCh();
        int findmaximum(int ranking[],int max);
        double uniform(const double min, const double max);
        double gaussian(double mu, double sigma);
        double tA();
        double tB();
        //chromosome prev_population[POP];
        //chromosome next_population[POP];
        chromosome *next_gen;
        chromosome *population;
        double *roulette;
        int *rank;
        ofstream outres;
private:

};

#endif      /* _GA_H */
```

ga.cpp

```
/*****************************************************************************
***************************
Ylopoisi twn methodon tis klasis atomo
*****************************************************************************
**************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <cmath>
#include <set>
#include <vector>
#include <ctime>
#include "chromosome.h"
#include "ga.h"


//statheres
#define POP 200
#define CrossProb 0.20
#define GEN_NUM 2
#define P1 3.14159265
#define CHROMOSOME_SIZE 3089


using namespace std;

ga::ga() {

        outres.open("Report.txt");
        //srand(time(NULL));
        //dimiourgei ena vector apo xromosomata
        rank=new int [POP];
        roulette=new double [POP];
      population=new chromosome [POP];
      next_gen=new chromosome [POP];
      for(int i=0;i<POP;i++){
            //prev_population[i]=chromosome();
            //next_population[i]=chromosome();
            population[i]=chromosome();
            //roulette[i]=0.0;
      }

}



ga::~ga() {
}

void ga:: setEvaluation(double eval,int position)
{

      //next_population[position].setEval(eval);
        population[position].setEval(eval);

}
void ga:: createRoulette()
```

```
{
	double sum_F=0;
	int x;


		/*for(int i=0;i<POP;i++)
		{

	next_population[i].setEval((rand()%1000000)/10000.0);
			//cout<<next_generation[i].returnEval()<<endl;
		}*/


	for(x=0;x<POP;x++)
	{
		sum_F+=population[x].returnEval();

		//cout<<sum_F<<endl;

		roulette[x]=0;
	}
	   //system("pause");

	roulette[0]=(population[0].returnEval()/sum_F)*POP;


	for(int i=1;i<POP;i++){

		roulette[i]=(population[i].returnEval()/sum_F)*POP;
	}
}

void ga :: newPopulation(){

//double temp=0;



    for(int i=0;i<POP;i++){
        rank[i]=0;
    }

    for(int i=0;i<POP;i++){
        for(int j=0;j<POP;j++){
        if(roulette[i]>roulette[j])
        rank[i]=rank[i]+1;

        }
    }
    int ranking[POP];
    for(int i=0;i<POP;i++){
        roulette[i]=round(roulette[i]);
        ranking[i]=rank[i];
    }

    chromosome temp[POP];
    int count=0;
    int flag_rank=1;
```

```
    int flag2=0;
    int max2=0;

    for(int i=0;i<POP;i++){

        max2=findmaximum(ranking,POP);

        int temp2=roulette[max2];
         for(int j=0;j<temp2;j++){
           ranking[max2]=-1;
            temp[count]=population[max2];
             count++;
             if (count>=POP)
                 break;


         }
                  if (count>=POP)
                 break;
              }
    if(count<POP){
         for(int i=0;i<POP;i++){
               max2=findmaximum(ranking,POP);
                ranking[max2]=-1;
           temp[count]=population[max2];
            count++;
            if (count>=POP)
                break;
         }
    }
    for(int i=0;i<POP;i++){
         population[i]=temp[i];
    }


}

void ga:: mutation(){


    srand(time(NULL));
    double mut_pro;
    double g;
    for(int i=0;i<POP;i=i++){
   for(int j=0;j<CHROMOSOME_SIZE;j=j++){
       mut_pro=rand() / ((double) RAND_MAX);
       if(mut_pro>=0.0){
           g=gaussian(0,1);

next_gen[i].individual[j]=population[i].individual[j]+/*population[i]
.mut_r[j]**/g;

          /* while(next_gen[i].individual[j]>5.0 ||
next_gen[i].individual[j]<-5.0){
                g=gaussian(0,1);

next_gen[i].individual[j]=population[i].individual[j]+population[i].m
ut_r[j]*g;
           }*/
```

```
next_gen[i].mut_r[j]=population[i].mut_r[j]*exp(tA()*g)+tB()*gaussian
(0,1);
        }
        else{
           next_gen[i].individual[j]=population[i].individual[j];
        }
    }
    //next_gen[i].convert_evaluate();
    }
     //chooseBestIndividual(POP*2);

}

double ga:: tA(){
    return 1/(sqrt(2*CHROMOSOME_SIZE));
}

double ga:: tB(){
    return 1/(sqrt(2*sqrt(CHROMOSOME_SIZE) ) );
}
double ga:: uniform(const double min, const double max)
{
    return min + rand()/(RAND_MAX/(max - min));
}

double ga:: gaussian(double mu, double sigma)
{
    double p = 1.0, p1, p2;
    double result=0;
    while (p >= 1.0)
    {
        p1 = uniform(-1, 1);
        p2 = uniform(-1, 1);
        p = p1 * p1 + p2 * p2;
    }
    result=mu + sigma * p1 * sqrt(-2.0 * log(p) / p);

    return result;
}
void ga :: crossover(){

    srand(time(NULL));
    double a=0.7;
    int maxrand=20;
    int max=POP-1;
   /* set <int>  taken;
    set<int>::iterator it;

    taken.insert(-1);*/


    vector <chromosome> forcrossover;

    for(int i=0;i<POP;i=i++){
        forcrossover.push_back(population[i]);
    }

    int lucky_one=0;
    int lucky_two=0;
    for(int i=0;i<POP;i=i+2){
```

```
        lucky_one= (rand()%maxrand)+0;

        lucky_two=(rand()%maxrand)+0;
        while(lucky_two==lucky_one ||
forcrossover.at(lucky_one).returnEval()==
forcrossover.at(lucky_two).returnEval()){
             lucky_two=(rand()%maxrand)+0;
        }
        forcrossover.erase(forcrossover.begin()+lucky_one);
        forcrossover.erase(forcrossover.begin()+lucky_two);
        maxrand=maxrand-2;
        for(int j=0;j<CHROMOSOME_SIZE;j++){

next_gen[i].individual[j]=population[lucky_one].individual[j]*a +(1-
a)*population[lucky_two].individual[j];
            /*Ypo kanonikes sinthike prepei mono ena
conver_evaluate() ni ginetai alla e3w apo to 2ro for*/
            next_gen[i].convert_evaluate();

next_gen[i+1].individual[j]=population[lucky_two].individual[j]*a
+(1-a)*population[lucky_one].individual[j];
            next_gen[i+1].convert_evaluate();
        }

    }

    /*for(int i=0;i<POP-1;i=i+2){
       /*  do{
            it=(rand()/(RAND_MAX) + 1.0)* (max - 1) + 1;
            }while(it==i && taken.find(it));
            taken.insert(parent2);
            */
       /* for(int j=0;j<CHROMOSOME_SIZE;j++){

            next_gen[i].individual[j]=population[i].individual[j]*a
+(1-a)*population[i+1].individual[j];*/
            /*Ypo kanonikes sinthike prepei mono ena
conver_evaluate() ni ginetai alla e3w apo to 2ro for*/
            /*next_gen[i].convert_evaluate();

next_gen[i+1].individual[j]=population[i+1].individual[j]*a +(1-
a)*population[i].individual[j];
            next_gen[i+1].convert_evaluate();
        }
    }*/

    chooseBestIndividual(POP*2);



}

void ga :: chooseBestIndividual(int size){

   // int sz=size1;
    chromosome *temp;
    temp=new chromosome [POP*2];

    int rank[POP*2];
    for(int i=0;i<POP;i++){
        temp[i]=population[i];
```

```
    }

    for(int i=0;i<size-POP;i++){
        temp[i+POP]=next_gen[i];
    }

    for(int i=0;i<size;i++){
        rank[i]=0;
    }
    int max =0;
    for(int i=0;i<size;i++){
        for(int j=0;j<size;j++){
        if(temp[i].returnEval()<temp[j].returnEval() && i!=j)
        rank[i]=rank[i]+1;
        if(rank[i]>rank[max])
            max=i;
        }
    }
    int max2=0;
    int high_rank=rank[max];
    int count=0;
    for(int i=0;i<=high_rank;i++){
        max2=findmaximum(rank,size);
        rank[max2]=-1;
        population[i]=temp[max2];
        count++;

        if(count==199)
            break;
        //i=-1;
    }
    delete(temp);
}

int ga :: findmaximum(int ranking[],int max){

    int max2=0;
    int temp=0;
    for(int i=0;i<max;i++){
        temp=ranking[i];
        if(ranking[i]>ranking[max2])
            max2=i;

    }
    return max2;
}
```

chromosome.h

```
/*************************************************************************
****************************
Ylopoisi twn methodon tis klasis atomo
*************************************************************************
**************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <cmath>
#include <vector>
#include <ctime>
#include <iostream>


#define CHROMOSOME_SIZE 3089

#ifndef _CHROMOSOME_H
#define     _CHROMOSOME_H

class chromosome {
public:
    chromosome();
      ~chromosome();

        double individual[CHROMOSOME_SIZE];
        double mut_r[CHROMOSOME_SIZE];
        void setEval(double eval);
        double returnEval();
      void returnTempIndividual(double *tempAtom);
      void crossoveredIndividual(double *tempIndividual);

      int returnPosition();
      void mutation();
      void convert_evaluate();
        void change_mut_r();
private:


    double evaluation;
    //double x1;
    //double x2;



};

#endif      /* _CHROMOSOME_H */
```

chromosome.cpp

```
/*************************************************************************
****************************
Ylopoisi twn methodon tis klasis atomo
*************************************************************************
**************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <cmath>

#include <vector>
#include <ctime>
#include "chromosome.h"
#include "ga.h"

//statheres
#define POPULATION 2
#define CHROMOSOME_SIZE 3089
#define PI 3.14159265
using namespace std;
chromosome::chromosome()
{
        srand(time(NULL));

      evaluation=0;
        //individual=new double [CHROMOSOME_SIZE];
        //mut_r=new double [CHROMOSOME_SIZE];
      //position=position_;

      //dimiourgia toy tixeou xromosomatos me 0 kai 1
        //KAnonika den thelouma max kai min sto ThesisGa
        double max=5.0;
        double min=-5.0;
        for(int i=0;i<CHROMOSOME_SIZE;i++)
      {
            individual[i]=(rand() / (static_cast<double>(RAND_MAX) +
1.0))* (max - min) + min;
                mut_r[i]=0 + rand()/(static_cast<double>(RAND_MAX/(1
- 0)));
      }
        convert_evaluate();


}


void chromosome :: convert_evaluate()
{
     double result=0;

        setEval(result);

    //}
}

void chromosome :: change_mut_r()
{
```

```
}

void chromosome :: setEval(double eval)
{
	evaluation = eval;
}

double chromosome :: returnEval()
{
	return evaluation;
}

chromosome::~chromosome() {
}

void chromosome :: crossoveredIndividual(double *tempIndividual)
{

	for(int i=0;i<CHROMOSOME_SIZE;i++){

		individual[i]=tempIndividual[i];

	}
}

void chromosome :: mutation(){

	double temp;

	for(int i=0;i<CHROMOSOME_SIZE;i++){

		temp=(rand()%1000000)/10000.0;
		if(temp<1){

			if(individual[i]==1)
				individual[i]=0;
			else
				individual[i]=1;

				convert_evaluate();
				//calEval();
		}
	}
}
```

BidirectionalRecurrentNeuralNetwork.h

```
#include "Neuron.h"
#include "DataReader.h"
#include "OutputData.h"
#include "targetver.h"
#include <cstdio>
#include <cstdlib>
#include <cstring>
//#include <fstream>
#include <iostream>
#include <cmath>
#include <cctype>
#include <vector>
#include <time.h>
#include <string>
#include "ga.h"
#include "chromosome.h"

using namespace std;


#ifndef BidirectionalRecurrentNeuralNetwork_h
#define BidirectionalRecurrentNeuralNetwork_h

//*********************************************************************
********************
//class BidirectionalRecurrentNeuralNetwork: This class illustrates
all the functions that
//a Bidirectional Recurent Neural Network needs to be trained and
tested
//*********************************************************************
********************
class BidirectionalRecurrentNeuralNetwork
{

private:

	//members of BidirectionalRecurrentNeuralNetwork class
	vector<string> encodingT;
	vector<string> encodingOutput;
	vector< vector<char> > outputClassification;
	vector<char> currentInput;
	vector<char> primaryStructure;
	vector<char> primaryStructureRead;
	vector<char> secondaryStructure;
	vector<char> secondaryStructureRead;
	vector<char> predictedSecondaryStructure;
	vector<char> predictedSecondaryStructureTrain;
	vector<double> weightsReturn;

	int msa;
	int msaLineLength;
	int inputSizeK;
	int inputSize;
	int halfWindow;
	int halfInputSize;
	int outputSize;
	int sizeOfEachLetter;
```

```
char trainFile[100];
char testFile[100];
char inputProfile[100];
char outputProfile[100];
char proteinName[100];
char proteinID[100];

int hLayerOneSize;
int hLayerTwoSize;
int hLayerOneSizeB;
int hLayerTwoSizeB;
int hLayerOneSizeF;
int hLayerTwoSizeF;
int activationFuncTypeOutput;
int activationFuncTypeHidden;
int windowSize;
int errorFunctionTypeHidden;
int errorFunctionTypeOutput;
int s;
int randomizeTheDataset;
int maxIterations;
int trainingSetAdditionVectorTempSize;
int testSetAdditionVectorTempSize;
int percentageCounter;int percentageCounterTrain;
int prevExcl;

double learningRate;
double momentum;
double qMinus1;
double qPlus1;
double percentage;double percentageTrain;
double trainingSetAddition;
double testSetAddition;
double trainingSetAdditionEpoch;
double testSetAdditionEpoch;
int ss;
char outputLetter;
int tempSizeStart;
int tempSizeEnd;
vector<Neuron> hLayerOne;
vector<Neuron> hLayerTwo;
vector<Neuron> hLayerOneB;
vector<Neuron> hLayerTwoB;
vector<Neuron> hLayerOneF;
vector<Neuron> hLayerTwoF;
vector<Neuron> outputLayer;
vector <int> tempOt;
vector<double> Ot;
vector<double> It;
vector<double> contextFt;
vector<double> contextBt;
vector<double> hLayerOneOutput;
vector<double> hLayerTwoOutput;
vector<double> hLayerOneBOutput;
vector<double> hLayerTwoBOutput;
vector<double> hLayerOneFOutput;
vector<double> hLayerTwoFOutput;
vector<double> inputhLayerOneF;
vector<double> inputhLayerOneB;
vector<double> tempInput;
vector<double> inputOutputLayer;
```

```
        vector<double> targetOutputs;
        vector<double> tempVector;
        vector<double> trainingSetAdditionVector;
        vector<double> trainingSetAdditionVectorTemp;
        vector<double> testSetAdditionVector;
        vector<double> testSetAdditionVectorTemp;
        vector<double> trainingSetAdditionEpochVector;
        vector<double> testSetAdditionEpochVector;
        vector<double> msaInput;

        //objects of input,output to a file
        DataReader* getInputData;
        OutputData* saveOutputData;
         /*OutputData* saveOutputData2;
          OutputData* saveOutputData3;
          OutputData* saveOutputData4;*/

          OutputData* OutputData_Filter;
        DataReader* getInput;
        //private method of BidirectionalRecurrentNeuralNetwork class
        void getInitialInput(void);

public:
        BidirectionalRecurrentNeuralNetwork(char id,int
hLayerOneSize,int hLayerTwoSize,int hLayerOneSizeB,int
hLayerTwoSizeB,int hLayerOneSizeF, int hLayerTwoSizeF,int
activationFuncTypeHidden,int activationFuncTypeOutput,double
learningRate,double momentum,int windowSize, double qMinus1,double
qPlus1,int errorFunctionTypeHidden,int errorFunctionTypeOutput,int
s,int epochN,char trainFile[100],char testFile[100], char
inputProfile[100],char outputProfile[100], int msaEnable,int
randomizeDataset,char outputFolder[100]);
        ~BidirectionalRecurrentNeuralNetwork(void);
        //Methods of BidirectionalRecurrentNeuralNetwork class
        bool initTrainingInput(int *primaryStructureSize);
        void retOutput(int seqt,vector<double> &outputresult, char
*secSt);
        bool initTestInput(int *primaryStructureSize);
        void doFeedForward(int sequencet,int isTrainOrTest,int
epoch,int center_window_size);
        void doBackpropagation(int sequencet,double learningRate,int
enable);
        void openFolders(void);
        void closeFolders(void);
        void getSequenceTrainingError();
        void getSequenceTestingError();
        double getEpochError(int flag);
        void printOutputs();
        void printOutputSequence(int c, vector<vector <double> >
vectoroutputresult,int flag);
        void getTargetOutputs(int sequencet);
        void printNetworkSpecification();
        void getMSAInput();
        void createMSAtempInput (int p, int q, int size);
        void returnData(vector<char> &primaryStructure,vector<char>
&secondaryStructure,char proteinID[100]);
        void initProtainQuest();
        void setWeights_ga(chromosome individual);

};
#endif
```

BidirectionalRecurrentNeuralNetwork.cpp

```
#include "DataReader.h"
#include "BidirectionalRecurrentNeuralNetwork.h"
#include <cstdio>
#include <cstdlib>
#include <cstring>
//#include <fstream>
#include <iostream>
#include <cmath>
#include <cctype>
#include <vector>
#include <time.h>
#include <string>
#include "ga.h"
#include "chromosome.h"


//using namespace std;

//*****************************************************************
*********************
//Constructor
BidirectionalRecurrentNeuralNetwork::BidirectionalRecurrentNeuralNetw
ork
//                (int hLayerOneSizeN,int hLayerTwoSizeN,
//                int hLayerOneSizeBN,int hLayerTwoSizeBN,int
hLayerOneSizeFN,int hLayerTwoSizeFN,
//                int activationFuncTypeN,double learningRateN,double
momentumN,int windowSizeN,
//                double qMinus1N,double qPlus1N,int
errorFunctionTypeN,int sN,int epochN,
//                char trainFileN[100],char testFileN[100],char
inputProfileN[100],char outputProfileN[100]):
//                initiates the BidirectionalRecurrentNeuralNetwork
//Parameters:
//                int hLayerOneSizeN                  :hidden layer
one size
//                int hLayerTwoSizeN                  :hidden layer
two size
//                int hLayerOneSizeBN                 :Backward
hidden layer one size
//                int hLayerTwoSizeBN                 :Backward
hidden layer two size
//                int hLayerOneSizeFN                 :Forward hidden
layer one size
//                int hLayerTwoSizeFN                 :Foeward hidden
layer two size
//                int activationFuncTypeN       :number that
corresponds to an activation function
//                double learningRateN          :learning rate
//                double momentumN              :momentum
//                int windowSizeN                     :the window
size for each specific residue
//                double qMinus1N                     :the q operator
for forward recurrent neural network
//                double qPlus1N                      :the q operator
for backward recurrent neural network
//                int errorFunctionTypeN        :number that
corresponds to an error function
```

```
//                int sN                                        :the
operator s
//                int epochN                              :number of
iterations
//                char trainFileN[100]           :the file name of
training set
//                char testFileN[100]                  :the file name
of testing set
//                char inputProfileN[100]        :the file name of
input profile
//                char outputProfileN[100]       :the file name of
output profile
//*****************************************************************
*********************
BidirectionalRecurrentNeuralNetwork::BidirectionalRecurrentNeuralNetw
ork(char id,int hLayerOneSizeN,int hLayerTwoSizeN,
                  int hLayerOneSizeBN,int hLayerTwoSizeBN,int
hLayerOneSizeFN,int hLayerTwoSizeFN,
                  int activationFuncTypeHiddenN,int
activationFuncTypeOutputN,double learningRateN,double momentumN,int
windowSizeN,
                  double qMinus1N,double qPlus1N,int
errorFunctionTypeHiddenN,int errorFunctionTypeOutputN,int sN,int
epochN,char trainFileN[100],char testFileN[100],char
inputProfileN[100],char outputProfileN[100],int msaEnable,int
randomizeDataset,char outputFolder[100])
{
      //private objects of BidirectionalRecurrentNeuralNetwork to
read from files and write to files
      getInputData=new DataReader();

      saveOutputData=new OutputData(id,outputFolder);
        /*saveOutputData2=new OutputData('2',outputFolder);
        saveOutputData3=new OutputData('3',outputFolder);
        saveOutputData4=new OutputData('4',outputFolder);*/

        //OutputData_Filter=new OutputData(id,outputFolder);
      getInput=new DataReader();

      //variables of class
      inputSizeK=0;
      inputSize=0;
      halfWindow=0;
      halfInputSize=0;
      outputSize=0;
      percentageCounter=0;percentageCounterTrain=0;
      percentage=0.0;percentageTrain=0.0;
      trainingSetAddition=0;
      testSetAddition=0;
      trainingSetAdditionEpoch=0;
      testSetAddition=0;

      //assign parameters of this class
      hLayerOneSize = hLayerOneSizeN;
      hLayerTwoSize = hLayerTwoSizeN;
      hLayerOneSizeB = hLayerOneSizeBN;
      hLayerTwoSizeB = hLayerTwoSizeBN;
      hLayerOneSizeF = hLayerOneSizeFN;
      hLayerTwoSizeF = hLayerTwoSizeFN;
      activationFuncTypeHidden = activationFuncTypeHiddenN;
      activationFuncTypeOutput = activationFuncTypeOutputN;
```

```
	learningRate = learningRateN;
	momentum = momentumN;
	windowSize = windowSizeN;
	msa = msaEnable;

	qMinus1 = (1.0 / qMinus1N);		//change applied by Georgia

	qPlus1 = qPlus1N;
	errorFunctionTypeHidden = errorFunctionTypeHiddenN;
	errorFunctionTypeOutput = errorFunctionTypeOutputN;
	maxIterations=epochN;
	s = sN;
	randomizeTheDataset = randomizeDataset;

	//assign parameters
	for(int i=0;i<100;i++)
	{
		trainFile[i] = trainFileN[i];
		testFile[i] = testFileN[i];
		inputProfile[i] = inputProfileN[i];
		outputProfile[i] = outputProfileN[i];
		proteinID[i] = '\0';
	}

	//printValuesOfOutputNeurons(-1,0);

	//private method that is called to read info about the input
and output encoding
	BidirectionalRecurrentNeuralNetwork::getInitialInput();

	//create variables for RNNs
	halfWindow=floor((float)windowSize/2);
	halfInputSize=floor((float)inputSize/2); //inputSize is the
residue volume -- almost always = 1


	//create vector for the target output of each simulation
	for(int i=0;i<outputSize;i++)
	{
		targetOutputs.push_back(0);
	}

	//create a vector for the current input of the network
	for(int i=0;i<inputSize;i++)
	{
		currentInput.push_back('\0');
	}

	//create a vector for the current input of the network -- is 21
	for(int i=0;i<inputSizeK;i++)
	{
		It.push_back(0);
	}

	//create the hidden layer one of the network
	for(int i=0;i<hLayerOneSize;i++)
	{

	hLayerOne.push_back(Neuron(inputSizeK,momentum,learningRate,act
ivationFuncTypeHidden,activationFuncTypeOutput,errorFunctionTypeHidde
n,errorFunctionTypeOutput,1));
```

```
            hLayerOneOutput.push_back(0);

      }

      //create the hidden layer two of the network
      for(int i=0;i<hLayerTwoSize;i++)
      {

      hLayerTwo.push_back(Neuron(hLayerOneSize,momentum,learningRate,
activationFuncTypeHidden,activationFuncTypeOutput,errorFunctionTypeHi
dden,errorFunctionTypeOutput,1));
            hLayerTwoOutput.push_back(0);
      }

      //create the contex layer of forward recurrent neural network
      if(hLayerTwoSizeF==0)
      {
            for(int i=0;i<(hLayerOneSizeF*s);i++)
            {
                  contextFt.push_back(0);
            }
      }else
      {
            for(int i=0;i<(hLayerTwoSizeF*s);i++)
            {
                  contextFt.push_back(0);
            }
      }

      //create the contex layer of backward recurrent neural network
      if(hLayerTwoSizeB==0)
      {
            for(int i=0;i<(hLayerOneSizeB*s);i++)
            {
                  contextBt.push_back(0);
            }
      }else
      {
            for(int i=0;i<(hLayerTwoSizeB*s);i++)
            {
                  contextBt.push_back(0);
            }
      }

      //create the input vector of hidden layer one
      for(int i=0;i<(inputSizeK+contextFt.size());i++)
      {
            inputhLayerOneF.push_back(0);
      }

      //create the input vector of hidden layer two
      for(int i=0;i<(inputSizeK+contextBt.size());i++)
      {
            inputhLayerOneB.push_back(0);
      }

      //create the hidden layer one of forward recurrent neural
network
      for(int i=0;i<hLayerOneSizeF;i++)
      {
```

```
      hLayerOneF.push_back(Neuron(inputSizeK+contextFt.size(),momentu
m,learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,errorFunctionTypeOutput,1));
            hLayerOneFOutput.push_back(0);
      }

      //create the hidden layer two of forward recurrent neural
network
      for(int i=0;i<hLayerTwoSizeF;i++)
      {

      hLayerTwoF.push_back(Neuron(hLayerOneSizeF,momentum,learningRat
e,activationFuncTypeHidden,activationFuncTypeOutput,errorFunctionType
Hidden,errorFunctionTypeOutput,1));
            hLayerTwoFOutput.push_back(0);
      }

      //create the hidden layer one of backward recurrent neural
network
      for(int i=0;i<hLayerOneSizeB;i++)
      {

      hLayerOneB.push_back(Neuron(inputSizeK+contextBt.size(),momentu
m,learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,errorFunctionTypeOutput,1));
            hLayerOneBOutput.push_back(0);
      }

      //create the hidden layer two of backward recurrent neural
network
      for(int i=0;i<hLayerTwoSizeB;i++)
      {

      hLayerTwoB.push_back(Neuron(hLayerOneSizeB,momentum,learningRat
e,activationFuncTypeHidden,activationFuncTypeOutput,errorFunctionType
Hidden,errorFunctionTypeOutput,1));
            hLayerTwoBOutput.push_back(0);
      }

      //creates the output layer with input and output vectors of
this layer
      //the size of the vector depends from the size and existence of
previous layers
      if(hLayerTwoSize==0 && hLayerTwoSizeF==0 && hLayerTwoSizeB==0)
      {
            for(int i=0;i<outputSize;i++)
            {

      outputLayer.push_back(Neuron(hLayerOneSize+hLayerOneSizeF+hLaye
rOneSizeB,momentum,

      learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
                  Ot.push_back(0);
                  tempOt.push_back(0);
            }

            for(int
i=0;i<hLayerOneSize+hLayerOneSizeF+hLayerOneSizeB;i++)
            {
```

```
                  inputOutputLayer.push_back(0);
            }
      }else if(hLayerTwoSize==0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB!=0)
      {
            for(int i=0;i<outputSize;i++)
            {

      outputLayer.push_back(Neuron(hLayerOneSize+hLayerOneSizeF+hLaye
rTwoSizeB,momentum,learningRate,activationFuncTypeHidden,activationFu
ncTypeOutput,errorFunctionTypeHidden,errorFunctionTypeOutput,0));
                  Ot.push_back(0);
                  tempOt.push_back(0);
            }

            for(int
i=0;i<hLayerOneSize+hLayerOneSizeF+hLayerTwoSizeB;i++)
            {
                  inputOutputLayer.push_back(0);
            }

      }else if(hLayerTwoSize==0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB==0)
      {
            for(int i=0;i<outputSize;i++)
            {

      outputLayer.push_back(Neuron(hLayerOneSize+hLayerTwoSizeF+hLaye
rOneSizeB,momentum,

      learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
                  Ot.push_back(0);
                  tempOt.push_back(0);
            }

            for(int
i=0;i<hLayerOneSize+hLayerTwoSizeF+hLayerOneSizeB;i++)
            {
                  inputOutputLayer.push_back(0);
            }
      }else if(hLayerTwoSize==0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB!=0)
      {
            for(int i=0;i<outputSize;i++)
            {

      outputLayer.push_back(Neuron(hLayerOneSize+hLayerTwoSizeF+hLaye
rTwoSizeB,momentum,

      learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
                  Ot.push_back(0);
                  tempOt.push_back(0);
            }

            for(int
i=0;i<hLayerOneSize+hLayerTwoSizeF+hLayerTwoSizeB;i++)
            {
                  inputOutputLayer.push_back(0);
            }
```

```
	}else if(hLayerTwoSize!=0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB==0)
	{
		for(int i=0;i<outputSize;i++)
		{

	outputLayer.push_back(Neuron(hLayerTwoSize+hLayerOneSizeF+hLaye
rOneSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerTwoSize+hLayerOneSizeF+hLayerOneSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
	}else if(hLayerTwoSize!=0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB!=0)
	{
		for(int i=0;i<outputSize;i++)
		{

	outputLayer.push_back(Neuron(hLayerTwoSize+hLayerOneSizeF+hLaye
rTwoSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerTwoSize+hLayerOneSizeF+hLayerTwoSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
	}else if(hLayerTwoSize!=0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB==0)
	{
		for(int i=0;i<outputSize;i++)
		{

	outputLayer.push_back(Neuron(hLayerTwoSize+hLayerTwoSizeF+hLaye
rOneSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerTwoSize+hLayerTwoSizeF+hLayerOneSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
```

```
	}else if(hLayerTwoSize!=0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB!=0)
	{
		for(int i=0;i<outputSize;i++)
		{

	outputLayer.push_back(Neuron(hLayerTwoSize+hLayerTwoSizeF+hLaye
rTwoSizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,
errorFunctionTypeHidden,errorFunctionTypeOutput,0));
			Ot.push_back(0);
			tempOt.push_back(0);
		}

		for(int
i=0;i<hLayerTwoSize+hLayerTwoSizeF+hLayerTwoSizeB;i++)
		{
			inputOutputLayer.push_back(0);
		}
	}
}

//*****************************************************************
*********************
//Deconstructor BidirectionalRecurrentNeuralNetwork(void)
//*****************************************************************
*********************
BidirectionalRecurrentNeuralNetwork::~BidirectionalRecurrentNeuralNet
work(void)
{

}

//executes this once
void BidirectionalRecurrentNeuralNetwork::getInitialInput()
{

	if (msa == 0)
	{	//to msaLineLength den xreiazete otan msa=0, gi afto den
xrisimopiite pouthenaF
		getInput-
>readEncoding(encodingT,&inputSizeK,inputProfile,msa,&msaLineLength);
		inputSize=inputSizeK;
		sizeOfEachLetter=encodingT[0].size();
		inputSizeK=inputSizeK*sizeOfEachLetter;
	}
	else //extra code by Georgia
	{
		//reads the detail lines from the information file
		getInput-
>readEncoding(encodingT,&inputSizeK,inputProfile,msa,&msaLineLength);
		inputSize=inputSizeK;
		sizeOfEachLetter=msaLineLength;
		inputSizeK=inputSizeK*sizeOfEachLetter;
	}


	getInput-
>readOutputEncoding(encodingOutput,outputClassification,&outputSize,o
utputProfile);
```

```
}
//
void BidirectionalRecurrentNeuralNetwork::getMSAInput()
{
	//get the whole msa encoding of the protein
	getInput->readEncodingMSA(encodingT,proteinID);

}

void BidirectionalRecurrentNeuralNetwork::openFolders(void)
{
	getInputData->initiateDataPointers(trainFile,testFile);
}

//
void BidirectionalRecurrentNeuralNetwork::closeFolders(void)
{
	getInputData->closeDataPointers(trainFile,testFile);

	//randomize dataset
	if(randomizeTheDataset==1) {int k = system("perl
randomizeDataset.pl");}
}

bool BidirectionalRecurrentNeuralNetwork::initTrainingInput(int
*primaryStructureSize)
{
	bool endOfFile;

	//the proteinID is the name of the current protein -- reads it
from training set
	endOfFile=getInputData-
>readTrainingInput(primaryStructureRead,secondaryStructureRead,protei
nID);

	primaryStructure.clear();
	secondaryStructure.clear();

	for(int i=0;i<primaryStructureRead.size();i++)
	{
		if(primaryStructureRead[i]!='!')
		{

	primaryStructure.push_back(primaryStructureRead[i]);

	secondaryStructure.push_back(secondaryStructureRead[i]);
		}
	}

	if(endOfFile){

		getInputData-
>translateSecondaryStructure(outputClassification,secondaryStructure)
;
		*primaryStructureSize=primaryStructure.size();

		return endOfFile;
	}
	else { return endOfFile;}
}
```

```
bool BidirectionalRecurrentNeuralNetwork::initTestInput(int
*primaryStructureSize)
{
	bool endOfFile;

	endOfFile=getInputData-
>readTestInput(primaryStructureRead,secondaryStructureRead,proteinID)
;

	primaryStructure.clear();
	secondaryStructure.clear();

	for(int i=0;i<primaryStructureRead.size();i++)
	{
		if(primaryStructureRead[i]!='!')
		{

	primaryStructure.push_back(primaryStructureRead[i]);

	secondaryStructure.push_back(secondaryStructureRead[i]);
		}
	}

	ss=0;
	if(endOfFile){
		getInputData-
>translateSecondaryStructure(outputClassification,secondaryStructure)
;
		*primaryStructureSize=primaryStructure.size();

		return endOfFile;
	}
	else
	{ return endOfFile; }
}
//
void BidirectionalRecurrentNeuralNetwork::createMSAtempInput (int p,
int q, int size)
{
	//msaInput is a double values vector
	msaInput.clear();

	//initialize and split msaInput --encodingT is a vector<string>
	//primaryStructure[p] gives a letter
	msaInput = getInput->splitValues(encodingT[p]);

	for (int i=0;i<size; i++){
	tempInput.push_back(msaInput[i]);

	}
}

void BidirectionalRecurrentNeuralNetwork::initProtainQuest()
{

	prevExcl=0;
	tempSizeStart=0;
	tempSizeEnd=primaryStructure.size();

	for(int i=0;i<primaryStructureRead.size();i++)
	{
```

```
			if(primaryStructureRead[i]=='!')
			{
				tempSizeEnd=i;
				break;
			}
		}

}


void BidirectionalRecurrentNeuralNetwork::doFeedForward(int
sequencet,int isTrainOrTest,int epoch,int center_window_size)
{


		if(sequencet>=tempSizeEnd)
		{

			prevExcl++;

			tempSizeStart=tempSizeEnd;


			tempSizeEnd=primaryStructure.size();
			for(int
i=tempSizeStart+2;i<primaryStructureRead.size();i++)
			{
				if(primaryStructureRead[i]=='!')
				{
					tempSizeEnd=i-prevExcl;
					break;
				}
			}

		}

		//cout<<primaryStructure[sequencet]<<endl;

		if(sequencet==2)
		{
			//cout<<endl;
		}

		//RNNF
		for(int forwardt=sequencet-
halfWindow;forwardt<=sequencet;forwardt++)
		{
			tempInput.clear();

			for(int p=forwardt-halfInputSize;p<(forwardt-
halfInputSize+inputSize);p++)
			{
				if(p<tempSizeStart || p>=tempSizeEnd){
					for(int q=0;q<sizeOfEachLetter;q++)
					{
						tempInput.push_back(0);
					}
				}
				//if to p ine apo 0 mexri to tempsize-1
				else
				{
```

```
                        for(int q=0;q<sizeOfEachLetter;q++)
                        {
                              //if(sequencet==185)
                              //{
                              //    cout<<p<<"
"<<q<<sizeOfEachLetter<<endl;
                              //}
                              if (msa == 0)

      tempInput.push_back((double)encodingT[primaryStructure[p]-
65][q]-48);
                              else{//change by Georgia
                                    //take the input in the sequencet
place -- the p variable express the position
                                    if(sequencet==2)
                                    {
                                          //cout<<p<<" "<<q<<endl;
                                    }

      createMSAtempInput(p,q,sizeOfEachLetter);
                                    break;
                              }
                        }
                  }
            }



            for(int p=0;p<inputhLayerOneF.size();p++)
            {
                  if(p<contextFt.size())
                        inputhLayerOneF[p]=contextFt[p];
                  else{
                        inputhLayerOneF[p]=tempInput[p-
contextFt.size()];

                  }
            }

            for(int p=0;p<hLayerOneF.size();p++)
            {

      hLayerOneFOutput[p]=hLayerOneF[p].getOutput(inputhLayerOneF);
            }

            if(hLayerTwoSizeF==0)
            {

                  int temp=contextFt.size()-hLayerOneFOutput.size();
                  int newPlace=contextFt.size()/s;
                  for(int p=0;p<temp;p++)
                  {
                        contextFt[p]=contextFt[p+newPlace];
                  }
                  for(int p=temp,i=0;p<contextFt.size();p++,i++)
                  {
                        contextFt[p]=hLayerOneFOutput[i]*qMinus1;
                  }
            }else
            {
                  for(int p=0;p<hLayerTwoF.size();p++)
```

```
                {

      hLayerTwoFOutput[p]=hLayerTwoF[p].getOutput(hLayerOneFOutput);
                }

                int temp=contextFt.size()-hLayerTwoFOutput.size();
                int newPlace=contextFt.size()/s;
                for(int p=0;p<temp;p++)
                {
                      contextFt[p]=contextFt[p+newPlace];
                }
                for(int p=temp,i=0;p<contextFt.size();p++,i++)
                {
                      contextFt[p]=hLayerTwoFOutput[i]*qMinus1;
                }
          }
      }

      for(int i=0;i<contextFt.size();i++)
      {
            contextFt[i]=0;
      }

      It.clear();
      for(int center=sequencet-
center_window_size;center<=sequencet+center_window_size;center++){
      for(int p=center-halfInputSize;p<(sequencet-
halfInputSize+inputSize);p++)
      {
            if(p<0)
            {
                  for(int q=0;q<sizeOfEachLetter;q++)
                  {
                        It.push_back(0);
                  }
            }
            else if(p>=primaryStructure.size())
            {
                  for(int q=0;q<sizeOfEachLetter;q++)
                  {
                        It.push_back(0);
                  }
            }
            else
            {
                  for(int q=0;q<sizeOfEachLetter;q++)
                  {
                        if (msa == 0)

      It.push_back((double)encodingT[primaryStructure[p]-65][q]-48);
                        else{
                              msaInput.clear();
                              //initialize and split msaInput --
encodingT is a vector<string>
                              //primaryStructure[p] gives a letter
                              msaInput = getInput-
>splitValues(encodingT[p]);
                              for (int i=0;i<sizeOfEachLetter; i++){
                                    It.push_back(msaInput[i]);
                              }
                        break;
```

```
                        }
                  }
            }

      }
      }

      //BNNF
      for(int
backwardt=sequencet+halfWindow;backwardt>=sequencet;backwardt--)
      {
            tempInput.clear();
            for(int p=backwardt-halfInputSize;p<(backwardt-
halfInputSize+inputSize);p++)
            {
                  if(p<tempSizeStart || p>=tempSizeEnd)
                  {
                        for(int q=0;q<sizeOfEachLetter;q++)
                        {
                              tempInput.push_back(0);
                        }
                  }
                  else
                  {
                        for(int q=0;q<sizeOfEachLetter;q++)
                        {
                              if (msa == 0)

      tempInput.push_back((double)encodingT[primaryStructure[p]-
65][q]-48);
                              else{//change by Georgia
                                    if(sequencet==268)
                                    {
                                          ///cout<<p<<" "<<q<<endl;
                                    }

      createMSAtempInput(p,q,sizeOfEachLetter);
                                    break;
                              }
                        }
                  }
            }

            for(int p=0;p<inputhLayerOneB.size();p++)
            {
                  if(p<tempInput.size())
                  {
                        inputhLayerOneB[p]=tempInput[p];
                  }
                  else
                  {
                        inputhLayerOneB[p]=contextBt[p-
tempInput.size()];
                  }
            }

            for(int p=0;p<hLayerOneB.size();p++)
            {

      hLayerOneBOutput[p]=hLayerOneB[p].getOutput(inputhLayerOneB);
            }
```

```
            if(hLayerTwoSizeB==0)
            {
                  int temp=contextBt.size()-hLayerOneBOutput.size();
                  int newPlace=contextBt.size()/s;
                  for(int
p=contextBt.size();p>=hLayerOneBOutput.size();p--)
                  {
                        contextBt[p]=contextBt[p-newPlace];
                  }
                  for(int p=0;p<hLayerOneBOutput.size();p++)
                  {
                        contextBt[p]=hLayerOneBOutput[p]*qPlus1;
                  }
            }else
            {
                  for(int p=0;p<hLayerTwoB.size();p++)
                  {

      hLayerTwoBOutput[p]=hLayerTwoB[p].getOutput(hLayerOneBOutput);
                  }

                  int temp=contextBt.size()-hLayerTwoBOutput.size();
                  int newPlace=contextBt.size()/s;
                  for(int p=contextBt.size()-
1;p>=hLayerTwoBOutput.size();p--)
                  {
                        contextBt[p]=contextBt[p-newPlace];
                  }
                  for(int p=0;p<hLayerTwoBOutput.size();p++)
                  {
                        contextBt[p]=hLayerTwoBOutput[p]*qPlus1;
                  }
            }
      }
                        //telos BRNN
      for(int i=0;i<contextBt.size();i++)
      {
            contextBt[i]=0;
      }

      for(int p=0;p<hLayerOne.size();p++)
      {
            hLayerOneOutput[p]=hLayerOne[p].getOutput(It);
      }

      if(hLayerTwoSize!=0)
      {
            for(int p=0;p<hLayerTwo.size();p++)
            {

      hLayerTwoOutput[p]=hLayerTwo[p].getOutput(hLayerOneOutput);
            }
      }


      if(hLayerTwoSize==0 && hLayerTwoSizeF==0 && hLayerTwoSizeB==0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerOneFOutput.size())
```

```
                        inputOutputLayer[p]=hLayerOneFOutput[p];
                  else
if(p<(hLayerOneFOutput.size()+hLayerOneOutput.size()))
                        inputOutputLayer[p]=hLayerOneOutput[p-
hLayerOneFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerOneBOutput[p-
hLayerOneFOutput.size()-hLayerOneOutput.size()];
            }
      }else if(hLayerTwoSize==0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB!=0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerOneFOutput.size())
                        inputOutputLayer[p]=hLayerOneFOutput[p];
                  else
if(p<(hLayerOneFOutput.size()+hLayerOneOutput.size()))
                        inputOutputLayer[p]=hLayerOneOutput[p-
hLayerOneFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerTwoBOutput[p-
hLayerOneFOutput.size()-hLayerOneOutput.size()];
            }
      }else if(hLayerTwoSize==0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB==0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerTwoFOutput.size())
                        inputOutputLayer[p]=hLayerTwoFOutput[p];
                  else
if(p<(hLayerTwoFOutput.size()+hLayerOneOutput.size()))
                        inputOutputLayer[p]=hLayerOneOutput[p-
hLayerTwoFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerOneBOutput[p-
hLayerTwoFOutput.size()-hLayerOneOutput.size()];
                  }
      }else if(hLayerTwoSize==0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB!=0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerTwoFOutput.size())
                        inputOutputLayer[p]=hLayerTwoFOutput[p];
                  else
if(p<(hLayerTwoFOutput.size()+hLayerOneOutput.size()))
                        inputOutputLayer[p]=hLayerOneOutput[p-
hLayerTwoFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerTwoBOutput[p-
hLayerTwoFOutput.size()-hLayerOneOutput.size()];
                  }
      }else if(hLayerTwoSize!=0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB==0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerOneFOutput.size())
                        inputOutputLayer[p]=hLayerOneFOutput[p];
```

```
                  else
if(p<(hLayerOneFOutput.size()+hLayerTwoOutput.size()))
                        inputOutputLayer[p]=hLayerTwoOutput[p-
hLayerOneFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerOneBOutput[p-
hLayerOneFOutput.size()-hLayerTwoOutput.size()];
                  }
      }else if(hLayerTwoSize!=0 && hLayerTwoSizeF==0 &&
hLayerTwoSizeB!=0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerOneFOutput.size())
                        inputOutputLayer[p]=hLayerOneFOutput[p];
                  else
if(p<(hLayerOneFOutput.size()+hLayerTwoOutput.size()))
                        inputOutputLayer[p]=hLayerTwoOutput[p-
hLayerOneFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerTwoBOutput[p-
hLayerOneFOutput.size()-hLayerTwoOutput.size()];
                  }
      }else if(hLayerTwoSize!=0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB==0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerTwoFOutput.size())
                        inputOutputLayer[p]=hLayerTwoFOutput[p];
                  else
if(p<(hLayerTwoFOutput.size()+hLayerTwoOutput.size()))
                        inputOutputLayer[p]=hLayerTwoOutput[p-
hLayerTwoFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerOneBOutput[p-
hLayerTwoFOutput.size()-hLayerTwoOutput.size()];
            }
      }else if(hLayerTwoSize!=0 && hLayerTwoSizeF!=0 &&
hLayerTwoSizeB!=0)
      {
            for(int p=0;p<inputOutputLayer.size();p++)
            {
                  if(p<hLayerTwoFOutput.size())
                        inputOutputLayer[p]=hLayerTwoFOutput[p];
                  else
if(p<(hLayerTwoFOutput.size()+hLayerTwoOutput.size()))
                        inputOutputLayer[p]=hLayerTwoOutput[p-
hLayerTwoFOutput.size()];
                  else
                        inputOutputLayer[p]=hLayerTwoBOutput[p-
hLayerTwoFOutput.size()-hLayerTwoOutput.size()];
            }
      }

      //gia kathe output neuron vazw sto Ot tin real timi tou
      for(int p=0;p<outputLayer.size();p++)
      {
            Ot[p]=outputLayer[p].getOutput(inputOutputLayer);
      }
```

```
	//function that takes the target outputs of the output neurons
	getTargetOutputs(sequencet);

	//1 == is train
	if(isTrainOrTest==1)
	{
		for(int p=0;p<Ot.size();p++)
		{
			trainingSetAddition+=(targetOutputs[p]-
Ot[p])*(targetOutputs[p]-Ot[p]);
		}


		//we want to see what were the real outputs of the
training set
		int i=0;
		int counterEnc=0;
		int x;

		//step function
		/*for(int i=0;i<Ot.size();i++)
		{
			//if the activation function is tanh
			if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
			{
				if(Ot[i]>=0.0)
					tempOt[i]= 1;
				else
					tempOt[i]= -1;
			}
			else
			{
				if(Ot[i]>=0.5)		//kanonika edw ine
0.5 an exw sigmoid/linear
					tempOt[i]= 1;
				else
					tempOt[i]= 0;
			}
		}*/

		double maxVal=Ot[0];
		int maxIndex =0;

		for(int i=0;i<Ot.size();i++)
		{

			if (maxVal <= Ot[i])
			{
				maxVal = Ot[i];
				maxIndex = i;
			}
		}


		for (int i=0;i<tempOt.size();i++)
		{
			if (i==maxIndex)
			{
				tempOt[i]=1;
			}
```

```
                  else
                  {
                        if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
                              tempOt[i]=-1;
                        else
                              tempOt[i]=0;
                  }
            }

            for(int i=0;i<encodingOutput.size();i++)
            {
                  counterEnc++;
                  //if the current position has a letter
                  if(encodingOutput[i][0]!='\0')
                  {
                        int counter;
                        int desiredOutput;


      for(counter=0;counter<tempOt.size();counter++)
                        {
                              desiredOutput =
((int)encodingOutput[i][counter]-48);

                              //changes the lower bound if tanh is
used for output neurons
                              if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
                              {
                                    if(desiredOutput==0)
                                          desiredOutput = -1;
                              }

                              if(desiredOutput!=tempOt[counter])
                                    break;

                        }

                        if(counter==tempOt.size())
                        {
                              outputLetter=i+65;
                              break;
                        }
                  }

            }

            x=encodingOutput.size();
            if((int)x == (int)counterEnc)
                  outputLetter='L';

            if(epoch==this->maxIterations-1){

      predictedSecondaryStructureTrain.push_back(outputLetter);



            }

      }
```

```
	//2 == is test -- no backward
	else if(isTrainOrTest==2)

	{
		for(int p=0;p<Ot.size();p++)
		{
			testSetAddition+=(targetOutputs[p]-
Ot[p])*(targetOutputs[p]-Ot[p]);
		}

		int i=0;
		int counterEnc=0;
		int x;

		//step function
		/*for(int i=0;i<Ot.size();i++)
		{
			//if the activation function is tanh
			if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
			{
				if(Ot[i]>=0.0)
					tempOt[i]= 1;
				else
					tempOt[i]= -1;
			}
			else
			{
				if(Ot[i]>=0.5)		//kanonika edw ine
0.5 an exw sigmoid/linear
					tempOt[i]= 1;
				else
					tempOt[i]= 0;
			}
		}*/

		//enable this for "winner takes all" and remove step
function
		double maxVal=Ot[0];
		int maxIndex =0;

		for(int i=0;i<Ot.size();i++)
		{

			if (maxVal <= Ot[i])
			{
				maxVal = Ot[i];
				maxIndex = i;
			}
		}


		for (int i=0;i<tempOt.size();i++)
		{
			if (i==maxIndex)
			{
				tempOt[i]=1;
			}
			else
```

```
                  {
                        if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
                              tempOt[i]=-1;
                        else
                              tempOt[i]=0;
                  }
            }


            for(int i=0;i<encodingOutput.size();i++)
            {
                  counterEnc++;
                  //if the current position has a letter
                  if(encodingOutput[i][0]!='\0')
                  {
                        int counter;
                        int desiredOutput;


      for(counter=0;counter<tempOt.size();counter++)
                        {
                              desiredOutput =
((int)encodingOutput[i][counter]-48);

                              //changes the lower bound if tanh is
used for output neurons
                              if(activationFuncTypeOutput==2 &&
errorFunctionTypeOutput==2)
                              {
                                    if(desiredOutput==0)
                                          desiredOutput = -1;
                              }

                              if(desiredOutput!=tempOt[counter])
                                    break;

                        }

                        if(counter==tempOt.size())
                        {
                              outputLetter=i+65;
                              break;
                        }
                  }

            }

            x=encodingOutput.size();
            if((int)x == (int)counterEnc)
                  //outputLetter='X';
                  outputLetter='L';


            if(epoch==this->maxIterations-1){

      predictedSecondaryStructure.push_back(outputLetter);
            }


      }
```

```
}

void BidirectionalRecurrentNeuralNetwork::getTargetOutputs(int
sequencet)
{
	//always save 1 or 0 in the tagetOutputs vector -- if
activation function is sigmoid/linear it does not need any other
processing
	for(int p=0;p<Ot.size();p++)
	{

	targetOutputs[p]=encodingOutput[secondaryStructure[sequencet]-
65][p]-48;
	}

	//if the output neuron's activation function is tanh changes
those target values from 1 and 0 to 1 and -1 respectively
	if(activationFuncTypeOutput==2 && errorFunctionTypeOutput==2)
	{
		for(int p=0;p<Ot.size();p++)
		{
			//change only the 0 to -1
			if((targetOutputs[p]) == 0 )
			{
				targetOutputs[p] = -1;
			}
		}
	}
}


void BidirectionalRecurrentNeuralNetwork::getSequenceTrainingError(){

	trainingSetAdditionVector.push_back(sqrt(trainingSetAddition)/(
double)Ot.size()/primaryStructure.size());
	trainingSetAdditionVectorTemp.push_back(sqrt(trainingSetAdditio
n)/(double)Ot.size()/primaryStructure.size());
	trainingSetAdditionVectorTempSize=trainingSetAdditionVectorTemp
.size();
	trainingSetAddition=0;
}

void BidirectionalRecurrentNeuralNetwork::getSequenceTestingError(){

	testSetAdditionVector.push_back(sqrt(testSetAddition)/(double)O
t.size()/primaryStructure.size());
	testSetAdditionVectorTemp.push_back(sqrt(testSetAddition)/(doub
le)Ot.size()/primaryStructure.size());
	testSetAdditionVectorTempSize=testSetAdditionVectorTemp.size();
	testSetAddition=0;
}


double BidirectionalRecurrentNeuralNetwork::getEpochError(int flag){

	double tempValue;
	  double result=0; //To Result tha filagei to mean square error
pou einai to evaluation to GA
	tempValue=0;
	  if (flag==0){
	for(int i=0;i<trainingSetAdditionVectorTempSize;i++)
```

```
	{
		tempValue+=trainingSetAdditionVectorTemp[i];
	}

	trainingSetAdditionEpochVector.push_back(tempValue/(double)trai
ningSetAdditionVectorTempSize);
	  result=tempValue/(double)trainingSetAdditionVectorTempSize;
	tempValue=0;
	  }
	  else{
	for(int i=0;i<testSetAdditionVectorTempSize;i++)
	{
		tempValue+=testSetAdditionVectorTemp[i];
	}

	testSetAdditionEpochVector.push_back(tempValue/(double)testSetA
dditionVectorTempSize);
	  }
	trainingSetAdditionVectorTemp.clear();
	testSetAdditionVectorTemp.clear();

	  return result;
}


void BidirectionalRecurrentNeuralNetwork::printOutputSequence(int
trainOrTest,vector<vector <double> > vectoroutputresult,int flag)
{

	double tempPercentage;
	int counter=0;

	if(trainOrTest==2) //testing
	{for(int i=0;i<primaryStructure.size();i++)
	{
		if(secondaryStructure[i]==predictedSecondaryStructure[i])
			counter++;
	}

	tempPercentage=(double)counter*100.0/(double)secondaryStructure
.size();
	saveOutputData-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecond
aryStructure,proteinID,tempPercentage,2,vectoroutputresult,flag);
	percentageCounter++;
	percentage+=tempPercentage;
	predictedSecondaryStructure.clear();}
	else //training
	{

	for(int i=0;i<primaryStructure.size();i++)
	{

	if(secondaryStructure[i]==predictedSecondaryStructureTrain[i])
			counter++;
	}

	tempPercentage=(double)counter*100.0/(double)secondaryStructure
.size();
	saveOutputData-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecond
```

```
aryStructureTrain,proteinID,tempPercentage,1,vectoroutputresult,flag)
;
	percentageCounterTrain++;
	percentageTrain+=tempPercentage;
	predictedSecondaryStructureTrain.clear();
	}
}

void BidirectionalRecurrentNeuralNetwork::printNetworkSpecification()
{

	saveOutputData-
>createNetworkFile(hLayerOneSize,hLayerTwoSize,hLayerOneSizeB,hLayerT
woSizeB,hLayerOneSizeF,

	hLayerTwoSizeF,activationFuncTypeHidden,learningRate,momentum,w
indowSize,

	qMinus1,qPlus1,errorFunctionTypeHidden,s,maxIterations,trainFil
e,testFile,

	inputProfile,outputProfile,percentage/(double)percentageCounter
);

}

void BidirectionalRecurrentNeuralNetwork::printOutputs()
{
	saveOutputData-
>createErrorFiles(trainingSetAdditionEpochVector,testSetAdditionEpoch
Vector,

	trainingSetAdditionVector,testSetAdditionVector); //change by
Georgia - last variable
	saveOutputData->closeAllPointers();
}


void BidirectionalRecurrentNeuralNetwork::doBackpropagation(int
sequencet, double learningRate,int enable)
{

	for(int p=0;p<outputLayer.size();p++)
	{

	outputLayer[p].calculateOutputLayerError(targetOutputs[p]);
		outputLayer[p].calculateErrorPropagation();
		outputLayer[p].adjustDw(learningRate);
	}
//
	if(hLayerTwoSizeF==0 && hLayerTwoSize==0 && hLayerTwoSizeB==0)
	{

		tempVector.clear();
		for(int i=0;i<hLayerOneF.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i));
				}else{
```

```
	tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                  }
            }


            for(int i=0;i<hLayerOneF.size();i++){


	hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF));
                        }else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
);
                        }
                  }
            }



            for(int i=0;i<hLayerOne.size();i++){


	hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF+hLayerOneSize));
                        }else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
+hLayerOneSize);
                        }
                  }
            }


            for(int i=0;i<hLayerOneB.size();i++){
```

```
	hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOneB[i].calculateErrorPropagation();
			hLayerOneB[i].adjustDw(learningRate);
		}


	}else if(hLayerTwoSizeF==0 && hLayerTwoSize==0 &&
hLayerTwoSizeB!=0)
	{

		tempVector.clear();
		for(int i=0;i<hLayerOneF.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i);
				}
			}
		}


		for(int i=0;i<hLayerOneF.size();i++){


	hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOneF[i].calculateErrorPropagation();
			hLayerOneF[i].adjustDw(learningRate);
		}




		tempVector.clear();
		for(int i=0;i<hLayerOne.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
);
				}
			}
		}



		for(int i=0;i<hLayerOne.size();i++){


	hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOne[i].calculateErrorPropagation();
			hLayerOne[i].adjustDw(learningRate);
		}
```

```
            tempVector.clear();
            for(int i=0;i<hLayerTwoB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
+hLayerOneSize);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoB.size();i++){


      hLayerTwoB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoB[i].calculateErrorPropagation();
                  hLayerTwoB[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<hLayerTwoB.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoB[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoB[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }




      }else if(hLayerTwoSizeF==0 && hLayerTwoSize!=0 &&
hLayerTwoSizeB==0)
      {

            tempVector.clear();
            for(int i=0;i<hLayerOneF.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i);
```

```
                        }
                  }
            }


            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerTwo.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
);
                        }
                  }
            }

            for(int i=0;i<hLayerTwo.size();i++){


      hLayerTwo[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwo[i].calculateErrorPropagation();
                  hLayerTwo[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<hLayerTwo.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwo[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwo[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
```

```
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
+hLayerTwoSize);
                        }
                  }
            }


            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }




      }else if(hLayerTwoSizeF==0 && hLayerTwoSize!=0 &&
hLayerTwoSizeB!=0)
      {

            tempVector.clear();
            for(int i=0;i<hLayerOneF.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                  }
            }


            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerTwo.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){
```

```
	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF));
                        }else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
);
                        }
                  }
            }

            for(int i=0;i<hLayerTwo.size();i++){


	hLayerTwo[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwo[i].calculateErrorPropagation();
                  hLayerTwo[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<hLayerTwo.size();j++){
                        if(j==0){

	tempVector.push_back(hLayerTwo[j].getSpecificError(i));
                        }else{

	tempVector[i]+=hLayerTwo[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOne.size();i++){


	hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerTwoB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerOn
eSizeF+hLayerTwoSize));
                        }else{

	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF
+hLayerTwoSize);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoB.size();i++){


	hLayerTwoB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoB[i].calculateErrorPropagation();
                  hLayerTwoB[i].adjustDw(learningRate);
```

```
		}


		tempVector.clear();
		for(int i=0;i<hLayerOneB.size();i++){
			for(int j=0;j<hLayerTwoB.size();j++){
				if(j==0){

	tempVector.push_back(hLayerTwoB[j].getSpecificError(i));
				}else{

	tempVector[i]+=hLayerTwoB[j].getSpecificError(i);
				}
		}
		}

		for(int i=0;i<hLayerOneB.size();i++){


	hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOneB[i].calculateErrorPropagation();
			hLayerOneB[i].adjustDw(learningRate);
		}




	}else if(hLayerTwoSizeF!=0 && hLayerTwoSize==0 &&
hLayerTwoSizeB==0)
	{

		tempVector.clear();
		for(int i=0;i<hLayerTwoF.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i);
				}
			}
		}

		for(int i=0;i<hLayerTwoF.size();i++){


	hLayerTwoF[i].calculateHiddenLayerError(tempVector[i]);
			hLayerTwoF[i].calculateErrorPropagation();
			hLayerTwoF[i].adjustDw(learningRate);
		}


		tempVector.clear();
		for(int i=0;i<hLayerOneF.size();i++){
			for(int j=0;j<hLayerTwoF.size();j++){
				if(j==0){

	tempVector.push_back(hLayerTwoF[j].getSpecificError(i));
				}else{
```

```
      tempVector[i]+=hLayerTwoF[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
);
                        }
                  }
            }


            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
+hLayerOneSize);
                        }
                  }
            }


            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
```

```
                hLayerOneB[i].calculateErrorPropagation();
                hLayerOneB[i].adjustDw(learningRate);
          }




      }else if(hLayerTwoSizeF!=0 && hLayerTwoSize==0 &&
hLayerTwoSizeB!=0)
      {

            tempVector.clear();
            for(int i=0;i<hLayerTwoF.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoF.size();i++){


      hLayerTwoF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoF[i].calculateErrorPropagation();
                  hLayerTwoF[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneF.size();i++){
                  for(int j=0;j<hLayerTwoF.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoF[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoF[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
```

```
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
);
                        }
                  }
            }


            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerTwoB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
+hLayerOneSize);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoB.size();i++){


      hLayerTwoB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoB[i].calculateErrorPropagation();
                  hLayerTwoB[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<hLayerTwoB.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoB[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoB[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneB.size();i++){
```

```
	hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOneB[i].calculateErrorPropagation();
			hLayerOneB[i].adjustDw(learningRate);
		}



	}else if(hLayerTwoSizeF!=0 && hLayerTwoSize!=0 &&
hLayerTwoSizeB==0)
	{

		tempVector.clear();
		for(int i=0;i<hLayerTwoF.size();i++){
			for(int j=0;j<outputLayer.size();j++){
				if(j==0){

	tempVector.push_back(outputLayer[j].getSpecificError(i));
				}else{

	tempVector[i]+=outputLayer[j].getSpecificError(i);
				}
			}
		}

		for(int i=0;i<hLayerTwoF.size();i++){


	hLayerTwoF[i].calculateHiddenLayerError(tempVector[i]);
			hLayerTwoF[i].calculateErrorPropagation();
			hLayerTwoF[i].adjustDw(learningRate);
		}


		tempVector.clear();
		for(int i=0;i<hLayerOneF.size();i++){
			for(int j=0;j<hLayerTwoF.size();j++){
				if(j==0){

	tempVector.push_back(hLayerTwoF[j].getSpecificError(i));
				}else{

	tempVector[i]+=hLayerTwoF[j].getSpecificError(i);
				}
		}
		}

		for(int i=0;i<hLayerOneF.size();i++){


	hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
			hLayerOneF[i].calculateErrorPropagation();
			hLayerOneF[i].adjustDw(learningRate);
		}




		tempVector.clear();
		for(int i=0;i<hLayerTwo.size();i++){
			for(int j=0;j<outputLayer.size();j++){
```

```
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
);
                        }
                  }
            }

            for(int i=0;i<hLayerTwo.size();i++){


      hLayerTwo[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwo[i].calculateErrorPropagation();
                  hLayerTwo[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<hLayerTwo.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwo[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwo[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
+hLayerTwoSize);
                        }
                  }
            }

            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
```

```
                  hLayerOneB[i].adjustDw(learningRate);
            }



      }else if(hLayerTwoSizeF!=0 && hLayerTwoSize!=0 &&
hLayerTwoSizeB!=0)
      {

            tempVector.clear();
            for(int i=0;i<hLayerTwoF.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i);
                        }
                  }
            }

            for(int i=0;i<hLayerTwoF.size();i++){


      hLayerTwoF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoF[i].calculateErrorPropagation();
                  hLayerTwoF[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneF.size();i++){
                  for(int j=0;j<hLayerTwoF.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoF[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoF[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneF.size();i++){


      hLayerOneF[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneF[i].calculateErrorPropagation();
                  hLayerOneF[i].adjustDw(learningRate);
            }




            tempVector.clear();
            for(int i=0;i<hLayerTwo.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){
```

```
      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
);
                        }
                  }
            }

            for(int i=0;i<hLayerTwo.size();i++){


      hLayerTwo[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwo[i].calculateErrorPropagation();
                  hLayerTwo[i].adjustDw(learningRate);
            }

            tempVector.clear();
            for(int i=0;i<hLayerOne.size();i++){
                  for(int j=0;j<hLayerTwo.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwo[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwo[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOne.size();i++){


      hLayerOne[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOne[i].calculateErrorPropagation();
                  hLayerOne[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerTwoB.size();i++){
                  for(int j=0;j<outputLayer.size();j++){
                        if(j==0){

      tempVector.push_back(outputLayer[j].getSpecificError(i+hLayerTw
oSizeF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF
+hLayerTwoSize);
                        }
                  }
            }///

            for(int i=0;i<hLayerTwoB.size();i++){


      hLayerTwoB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerTwoB[i].calculateErrorPropagation();
```

```
                  hLayerTwoB[i].adjustDw(learningRate);
            }


            tempVector.clear();
            for(int i=0;i<hLayerOneB.size();i++){
                  for(int j=0;j<hLayerTwoB.size();j++){
                        if(j==0){

      tempVector.push_back(hLayerTwoB[j].getSpecificError(i));
                        }else{

      tempVector[i]+=hLayerTwoB[j].getSpecificError(i);
                        }
            }
            }

            for(int i=0;i<hLayerOneB.size();i++){


      hLayerOneB[i].calculateHiddenLayerError(tempVector[i]);
                  hLayerOneB[i].calculateErrorPropagation();
                  hLayerOneB[i].adjustDw(learningRate);
            }







      }


}

void BidirectionalRecurrentNeuralNetwork::retOutput(int
seqt,vector<double> &outputresult,char *secSt)
{

      outputresult=Ot;
      *secSt=secondaryStructure[seqt];

}

void BidirectionalRecurrentNeuralNetwork::returnData(vector<char>
&primaryStructureReturn,vector<char> &secondaryStructureReturn,char
proteinIDReturn[100])
{

      primaryStructureReturn.clear();
      secondaryStructureReturn.clear();

      for(int i=0;i<100;i++)
      {
            proteinIDReturn[i]=proteinID[i];
      }


      for(int i=0;i<primaryStructure.size();i++)
      {
            primaryStructureReturn.push_back(primaryStructure[i]);
```

```
      secondaryStructureReturn.push_back(secondaryStructure[i]);
      }


}

void BidirectionalRecurrentNeuralNetwork::setWeights_ga(chromosome
chromo){

    int chromo_pos=0;
    for(int i=0;i<hLayerOne.size();i++)
      {
                    hLayerOne[i].setWeights_ga(chromo,chromo_pos);

chromo_pos=chromo_pos+hLayerOne[i].returnWeightsVectSize()+1;   //---
-------------------->to 21
      }
     chromo_pos=0;
      for(int i=0;i<hLayerTwo.size();i++)
      {
                    hLayerOne[i].setWeights_ga(chromo,chromo_pos);

chromo_pos=chromo_pos+hLayerOne[i].returnWeightsVectSize()+1;   //---
-------------------->to 21
      }
     chromo_pos=0;
    for(int i=0;i<hLayerOneB.size();i++)
      {
                    hLayerOneB[i].setWeights_ga(chromo,chromo_pos);

chromo_pos=chromo_pos+hLayerOneB[i].returnWeightsVectSize()+1;   //--
--------------------->to 21
      }
     chromo_pos=0;
    for(int i=0;i<hLayerOneF.size();i++)
      {
                    hLayerOneF[i].setWeights_ga(chromo,chromo_pos);

chromo_pos=chromo_pos+hLayerOneF[i].returnWeightsVectSize()+1;;   //-
---------------------->to 21
      }
     chromo_pos=0;
    for(int i=0;i<outputLayer.size();i++)
      {
                    outputLayer[i].setWeights_ga(chromo,chromo_pos);

chromo_pos=chromo_pos+outputLayer[i].returnWeightsVectSize()+1;;
//---------------------->to 21
      }
}
```

Neuron.h

```
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>
#include "ga.h"
#include "chromosome.h"
using namespace std;

#ifndef Neuron_h
#define Neuron_h

//*****************************************************************************
*********************
//class Neuron: This class illustrates all the functions of a neuron
in a Bidirectional
//Recurent Neural Network that is trained by a variation of
Backpropagation Algorithm
//through time
//*****************************************************************************
*********************
class Neuron
{

private:

	//members of Neuron class
	vector<double> inputVector;
	vector<double> weightsVector;
	vector<double> weightMulError;
	vector<double> previousAdjustment;
	vector<double> Dweights;

	int sizeOfInput;
	int activationFunctionTypeOutput;
	int activationFunctionTypeHidden;
	int errorFunctionTypeHidden;
	int errorFunctionTypeOutput;
	int neuronType;

	double output;
	double error;
	double bias;
	double Dbias;
	double inputNet;
	double momentum;
	double learningRate;
	double biasPreviousAdjustment;
	double slope;
	double amplitude;
	double fmin;
	double fmax;
```

```
        void callActivationFunction(void);
        void calculateInput(void);

public:

        Neuron(int size,double momentumN,double learningRateN,int
functionTypeHiddens,int functionTypeOutput,int errorTypeHidden,int
errorTypeOutput,int neuronType);
        ~Neuron(void);



        //methods of Neuron class
        double getOutput(vector<double> getInput);
        void calculateOutputLayerError(double targetOutput);
        void calculateErrorPropagation();
        void calculateHiddenLayerError(double prevWeightSum);
        void adjustDw(double learningRateNew);
        double getSpecificError(int x);
        void returnWeightVector(vector<double> *weihtsReturn);
        void printTest();

          double returnWeightsVectSize();
          void setWeights_ga(chromosome chromo,int chromo_pos);

};
#endif
```

Neuron.cpp

```
#include "Neuron.h"
#include "Services.h"
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>
#include "ga.h"
#include "chromosome.h"
using namespace std;


//*********************************************************************
*********************
//Constructor Neuron(int size,double momentumN,double
learningRateN,int functionType):
//initiates the Neuron
//Parameters:
//          int size                        :Specifies the input size
//          double momentumN        :Specifies the initial momentum
of the neuron
//          double learningRateN    :Specifies the initial learning
rate of the neuron
//          int functionType        Specifies the activation function
of the neuron
//*********************************************************************
*********************
Neuron::Neuron(int size,double momentumN,double learningRateN,int
functionTypeHiddens,int functionTypeOutput,int errorTypeHidden,int
errorTypeOutput,int neuronTypeN)
{

      Services newServices=Services();

      for(int i=0;i<size;i++){
            inputVector.push_back(0);
            weightsVector.push_back(newServices.rand_numbers());
            weightMulError.push_back(0);
            Dweights.push_back(0.0);
            previousAdjustment.push_back(0.0);  //the previous
weights are saved in this variables
      }


      output=0;
      error=0;
      Dbias=0;
      inputNet=0;
      biasPreviousAdjustment=0;

      bias=newServices.rand_numbers();
```

```
	sizeOfInput=size;
	momentum=momentumN;
	learningRate=learningRateN;
	errorFunctionTypeOutput=errorTypeOutput;
	errorFunctionTypeHidden=errorTypeHidden;
	activationFunctionTypeHidden=functionTypeHiddens;
	activationFunctionTypeOutput=functionTypeOutput;

	neuronType=neuronTypeN;

	//parameters for tanh() activation function
	slope = (2.0/3.0);
	amplitude = 1.7159;

	//sigmoid bounds
	fmin = 0.0;
	fmax = 1.0;

}

//*****************************************************************
*********************
//Deconstructor Neuron(void)
//*****************************************************************
*********************
Neuron::~Neuron(void)
{
}

//*****************************************************************
*********************
//void calculateInput(void): private method that is called to
calculate the
//neuron's input. Specifically, it multiplies each input(output of a
neuron
//from the previous layer) with the appropriate  weight and then sum
all the
//results together to calculate the new neuron□s input
//*****************************************************************
*********************
void Neuron :: calculateInput(void)
{

	inputNet=0;

	for(int i=0;i<sizeOfInput;i++){
		inputNet+=(inputVector[i]*weightsVector[i]);
	}

	inputNet=(inputNet+(-bias));

}

//*****************************************************************
*********************
//void activationFunction(void): private method that is called to
calculate the
//neuron's output through an activation function
//*****************************************************************
*********************
void Neuron :: callActivationFunction(void)
```

```
{

	//hidden neurons
	if(neuronType==1)
	{
		if(activationFunctionTypeHidden==1) //using sigmoid
		{
			output=(1/(1+pow(2.718281828,-inputNet)));
		}
		else if (activationFunctionTypeHidden==2) //using tanh
		{
			output = amplitude * tanh(inputNet*slope);
		}
		else if (activationFunctionTypeHidden==3) //using linear
		{
			output = inputNet * ((fmax - fmin) + fmin);
		}
	}
	//output neurons (neuronType==0)
	else
	{
		if(activationFunctionTypeOutput==1) //using sigmoid
		{
			output=(1/(1+pow(2.718281828,-inputNet)));
		}
		else if (activationFunctionTypeOutput==2) //using tanh
		{
			output = amplitude * tanh(inputNet*slope);
		}
		else if (activationFunctionTypeOutput==3) //using linear
		{
			output = inputNet * ((fmax - fmin) + fmin);
		}
	}

}

//*************************************************************************
*********************
//double getOutput(vector <double> getInputs): public method that is
called to calculate
//the output of the neuron
//Parameters:
//		vector <double> getInputs	:the outputs of the
previous layer that are given as
//							inputs to this
neuron and are used to calculate the
//					net input
//Return:
//		double output			:the output of the
neuron
//*************************************************************************
*********************
double Neuron :: getOutput(vector<double> getInputs)
{

	inputVector=getInputs;

	calculateInput();

	callActivationFunction();
```

```
	return output;

}

//*********************************************************************
*********************
//void calculateOutputLayerError(double targetOutput): public method
that is called to
//calculate the output error of the neurons of output Layer
//Parameters:
//		double targetOutput		:the specific,target output
of the neuron for each
//                                                        time step
//*********************************************************************
*********************
void Neuron :: calculateOutputLayerError(double targetOutput)
{
	//sigmoid function
	if (errorFunctionTypeOutput==1)
	{
		error = output*(1-output)*(targetOutput-output);
	}
	//tanh function
	else if (errorFunctionTypeOutput==2)
	{
		error = (slope/amplitude) * (targetOutput-output) *
(amplitude-output) * (amplitude+output);
	}
	//linear function
	else if (errorFunctionTypeOutput==3)
	{
		error = 1.0 * (targetOutput-output);
	}
}

//*********************************************************************
*********************
//void calculateErrorPropagation(): public method that is called to
calculate the error
//that must be propagated back to the weights that are connected to
the specific neuron
//by multiply all the weights of the neuron with the epecific error
//*********************************************************************
*********************
void Neuron :: calculateErrorPropagation(){

	for(int i=0;i<weightMulError.size();i++){
		weightMulError[i]= error * (weightsVector[i]);
	}
}

//*********************************************************************
*********************
//void calculateHiddenLayerError(double prevWeightSum): public method
that is called to
//calculate the output error of the neurons of Hidden Layers
//Parameters:
//		double prevWeightSum		:the sum of the neurons'
error that are connectet
```

```
//                                                        with the output
of the specific neuron
//*************************************************************************
*********************
void Neuron :: calculateHiddenLayerError(double prevWeightSum)
{
	//sigmoid function
	if (errorFunctionTypeHidden==1)
	{
		error = output*(1-output)*prevWeightSum;
	}
	//tanh function
	else if (errorFunctionTypeHidden==2)
	{
		error = (slope/amplitude) * (amplitude-output) *
(amplitude+output) * prevWeightSum;
	}
	//linear function
	else if (errorFunctionTypeHidden==3)
	{
		error = 1.0 * prevWeightSum;
	}

}

//*************************************************************************
*********************
//void getSpecificError(int position): public method that is called
to return the
//result of the multiplication of a specific weight with the neurons
error
//Parameters:
//		double position			:the position of the weight
in the weight's vector
//*************************************************************************
*********************
double Neuron :: getSpecificError(int position){

	return weightMulError[position];

}

//*************************************************************************
*********************
//void adjustDw(): public method that is called to calculate the
adjustment of each
//weight and bias of the specific neuron. The bias is adjusted like a
neuron's weight
//*************************************************************************
*********************
void Neuron :: adjustDw(double learningRateNew){

	for(int i=0;i<Dweights.size();i++)
	{

	/*Dweights[i]+=learningRate*inputVector[i]*error+momentum*previ
ousAdjustment[i];

	previousAdjustment[i]=learningRate*inputVector[i]*error+momentu
m*previousAdjustment[i];
		weightsVector[i] = weightsVector[i] + Dweights[i];*/
```

```
            Dweights[i] =
learningRateNew*inputVector[i]*error+momentum*(weightsVector[i]-
previousAdjustment[i]);
            previousAdjustment[i] = weightsVector[i];
            weightsVector[i] = weightsVector[i] + Dweights[i];

            //Dweights[i]=0;
            //previousAdjustment[i]=0;
      }

      Dbias = learningRateNew*1*error+momentum*(bias-
biasPreviousAdjustment);
      biasPreviousAdjustment = bias;
      bias = - bias + Dbias;
      bias = -bias;

      /*Dbias+=learningRate*1*error+momentum*biasPreviousAdjustment;
      biasPreviousAdjustment=learningRate*1*error+momentum*biasPrevio
usAdjustment;
      bias=-bias+Dbias;
      bias=-bias;*/

      //Dbias=0;
      //biasPreviousAdjustment=0;

}
void Neuron :: printTest(){

      for (int i=0;i<inputVector.size();i++)
      {
            cout<<weightsVector[i]<<" ";
      }cout<<endl;



}

double Neuron::returnWeightsVectSize()
{
    return weightsVector.size();
}

void Neuron::setWeights_ga(chromosome chromo,int chromo_pos){
    int i=0;
    for (i;i<weightsVector.size();i++)
      {
            weightsVector.at(i)=chromo.individual[i+chromo_pos];
        }
    bias=chromo.individual[i+chromo_pos];
}
```

DataReader.h

```
#include <fstream>
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>

using namespace std;

#ifndef DataReader_h
#define DataReader_h

class DataReader
{

private:
	//members of DataReader class
	ifstream inTrainFile;
	ifstream inTestFile;
	ifstream encodingFile;
	int beginning;
	int numberOfResidues;
	ifstream inMsaFile;
public:
	DataReader(void);
	~DataReader(void);
	//methods of DataReader class
	void readParameters(float parameters[17],char
inputProfile[100],char outputProfile[100],
			char trainFile[100],char testFile[100], int*
msaEnable,int* randomizeDataset,int *center_w_size,char
outputFolder[100]);
	void readEncoding(vector<string> &encodingT,int
*inputSizeK,const char inputProfile[100], int msa, int* length);
	void initiateDataPointers(char trainFile[100],char
testFile[100]);
	bool readTrainingInput(vector<char>
&primaryStructure,vector<char> &secondaryStructure,char name[100]);
	bool readTestInput(vector<char> &primaryStructure,vector<char>
&secondaryStructure,char name[100]);
	void readOutputEncoding(vector<string> &encodingT,vector<
vector<char> > &outputClassification,int *outputSize,
			const char outputProfile[100]);
	void translateSecondaryStructure(vector< vector<char> >
&outputClassification,vector<char> &secondaryStructure);
	void closeDataPointers(char trainFile[100],char testFile[100]);
	void readEncodingMSA (vector<string> &encodingT,char
proteinID[100]);
	vector<double> splitValues (string stringEncoding);

};
#endif
```

DataReader.cpp

```
#include "DataReader.h"
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>
#include <sstream>
#include <algorithm>
#include <iterator>

using namespace std;


//*********************************************************************
*********************
//Constructor DataReader(void):initiates the DataReader
//*********************************************************************
*********************
DataReader::DataReader(void)
{

}

//*********************************************************************
*********************
//Deconstructor DataReader(void)
//*********************************************************************
*********************
DataReader::~DataReader(void)
{

}

//*********************************************************************
*********************
//void readParameters(float parameters[15],char
inputProfile[100],char outputProfile[100],
//                 char trainFile[100],char testFile[100]): public
method that is called to
//get the parameters of the neural network from the parameters.dat
file
//Parameters:
//          float parameters[17]          :this table returns alla
the numeric parameters for
//                                                BRNN. Each
position illustrates a parameter and which are
//                                                depended from
their order in the parameters.dat
//                                                file
//          char inputProfile[100]        :this table returns the
input profile file name
```

```
//          char outputProfile[100]       :this table returns the
output profile file name
//          char trainFile[100]                 :this table returns
the file name which contains the
//                                                    training set
//          char testFile[100]                  :this table returns
the file name which contains the
//                                                    testing set
//*****************************************************************
*******************
//*****************************************************************
*******************
void DataReader::readParameters(float parameters[17],char
inputProfile[100],char outputProfile[100],
                  char trainFile[100],char testFile[100], int*
msaEnable,int* randomizeDataset,int *center_w_size,char
outputFolder[100])
{
      //variables
      char temp[100];
      char state[100];
      int counter=0;

      //opens the DataIn\\parameters.dat file to read the parameters
      ifstream inFile;
      inFile.open("DataIn//parameters.dat");

      if (!inFile) {
            cerr << "Unable to open file parameters.dat";
            int x=0;
            cin>>x;
            exit(1);
      }

      //takes all the network parameters with specific order
      for(int i=0;i<27;i++){
            //reads the general parameters
            if(i<17)
            {
                  inFile >> temp;
                  inFile >> parameters[i];
            }else if(i==18)
            {
                  inFile >> temp;
                  inFile >> inputProfile;
            }else if(i==19)
            {
                  inFile >> temp;
                  inFile >> outputProfile;
            }else if(i==20)
            {
                  inFile >> temp;
                  inFile >> trainFile;
            }else if(i==21)
            {
                  inFile >> temp;
                  inFile >> testFile;
            }else if(i==23)
            {
                  inFile >> temp;
                  inFile >> *msaEnable;
```

```
		}
		else if(i==24)
		{
			inFile >> temp;
			inFile >> *randomizeDataset;
		}else if(i==25)
		{
			inFile >> temp;
			inFile >> *center_w_size;
		}else if(i==26)
		{
			inFile >> temp;
			inFile >> outputFolder;
		}
		else if((i==17)||(i==22))
		{
			inFile >> temp;
		}

	}

	inFile.close();
}


//*****************************************************************
*********************
//void readEncoding(vector<string> &encodingT,int *inputSizeK,const
char inputProfile[100]):
//public method that is called to read the file with input encoding
of a protein's primary
//structure. The file must have a specific layout. The encoding of
each letter is saved in
//a vector to the position that the letter has in the English
alphabet.
//Parameters:
//		vector<string> &encodingT	:returns a vector of
strings that illustrates the encoding
//							of each letter.
The vector has size of 26 strings which
//							portrays the
place of each letter
//		int *inputSizeK				:returns the size of
each input(how many residues)
//		const char inputProfile[100]:the name of input profile
file name
//*****************************************************************
*********************
void DataReader::readEncoding(vector<string> &encodingT,int
*inputSizeK,const char inputProfile[100], int msa, int* length)
{

	//variables
	char temp[100];
	char tempChar;
	int tempK;
	string tempString="\0";

	for(int i=0;i<100;i++)
		temp[i]='\0';
```

```
	//create the path string
	strcat(temp,"EncodingProfiles/");
	strcat(temp,inputProfile);

	//points to the file
	encodingFile.open(temp);

	if (!encodingFile)
	{
		cerr << "Unable to open file "<<inputProfile;
		int x=0;
		cin>>x;
		exit(1);
	}

	if(msa == 0)
	{
	//create the size of those vectors
	for(int i=0;i<26;i++)
	encodingT.push_back(tempString);


		for(int i=0;i<24;i++)
		{
			if(i<2)
			{
				encodingFile >> temp;
			}
			else if(i==2)
			{
				encodingFile >> temp;
				encodingFile >> tempK;
				*inputSizeK = tempK; //residue volume

			}
			else if(i<24)
			{
				encodingFile >> tempChar;
				encodingFile >> tempString;
				//The encoding of each letter is saved in a
vector to the position that
				//the letter has in the English alphabet
				encodingT[tempChar-65]+=tempString;

			}
		}
	}

	//add by Georgia
	/* extra code for the msa.txt file*/
	else
	{
		for(int i=0;i<6;i++)
		{
			if(i<2)
			{
				encodingFile >> temp;
			}
			else if(i==2)
			{
				encodingFile >> temp;
```

```
                        encodingFile >> tempK;
                        *inputSizeK = tempK; //residue volume
                  }
                  else if(i==3)
                  {
                        //the **************
                        encodingFile >> temp;
                  }
                  else if(i==4)
                  {
                        encodingFile >> temp;
                        encodingFile >> numberOfResidues;
                        *length = numberOfResidues;
                  }
                  else if(i==5)
                  {
                        //the **************
                        encodingFile >> temp;
                  }

            }

      }

      //close the pointer
      encodingFile.close();
}

void DataReader::readOutputEncoding(vector<string> &encodingT,vector<
vector<char> > &outputClassification,
                  int *outputSize,const char outputProfile[100])
{

      //pointer to a file
      ifstream inFile;

      //variables
      int tempK;
      int counter;
      char temp[100];
      char state[100];
      char tempChar;
      vector<char> tempCharV;
      string tempString="\0";

      //create the size of those vectors
      for(int i=0;i<26;i++)
            encodingT.push_back(tempString);

      for(int i=0;i<100;i++)
            temp[i]='\0';

      //create the path string
      strcat(temp,"OutputProfiles/");
      strcat(temp,outputProfile);

      //point to the file
      inFile.open(temp);

      if (!inFile) {
            cerr << "Unable to open file "<<outputProfile;
```

```
		int x=0;
		cin>>x;
		exit(1);
	}

	//read alla data from the file with a specific order
	for(int i=0;i<100;i++){
		if(i<2)
		{
			inFile >> temp;
		}else if(i==2)
		{
			inFile >> temp;
			inFile >> tempK;
			*outputSize = tempK;
		}else if(i<4)
		{
			inFile >> temp;
		}else if(i<(4+tempK))
		{
			//reads from the file all the secondary structure's
letters and their class
			tempCharV.clear();
			inFile >> state;
			tempCharV.push_back(state[0]);
			inFile >> state;
			counter=0;

			//remove alla the commas
			while(state[counter]!='\0')
			{
				if(state[counter]!=',')
				{
					tempCharV.push_back(state[counter]);
				}
				counter++;
			}
			//Each position of the vector of vector holds
			//the characteristic letter of the class which is
			//followed by the secondary structure letters of
			//that class.
			outputClassification.push_back(tempCharV);

		}else if(i<(4+tempK+2))
		{
			inFile >> temp;
		}else if(i<(6+tempK+tempK))
		{
			inFile >> tempChar;
			inFile >> tempString;
			//The encoding of each letter is saved in a vector
to the position that
			//the letter has in the English alphabet
			encodingT[tempChar-65]+=tempString;
		}
	}
	inFile.close();
}

//*****************************************************************
*********************
```

```
//void initiateDataPointers(char trainFile[100],char testFile[100]):
//public method that is called to initiate tha pointers to the files
which contain the training
//and testing sets.
//Parameters:
//          char trainFile[100]                         :the file name
of training sets
//          char testFile[100]                          :the file name
of testing sets
//*****************************************************************
*********************
void DataReader::initiateDataPointers(char trainFile[100],char
testFile[100])
{
      //variables
      char temp[100];

      //create the size of those vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      //create the path string
      strcat(temp,"TrainingSets/");
      strcat(temp,trainFile);

      //points to the file
      inTrainFile.open(temp);

      if (!inTrainFile) {
            cerr << "Unable to open file "<<temp;
            int x=0;
            cin>>x;
            exit(1);
      }

      //create the size of those vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      //create the path string
      strcat(temp,"TestSets/");
      strcat(temp,testFile);

      //points to the file
      inTestFile.open(temp);

      if (!inTestFile) {
            cerr << "Unable to open file "<<temp;
            int x=0;
            cin>>x;
            exit(1);
      }
}

//*****************************************************************
*********************
//void closeDataPointers(char trainFile[100],char testFile[100]):
//public method that is called to close tha pointers to the files
which contain the training
//and testing sets.
//Parameters:
```

```
//          char trainFile[100]                         :the file name
of training sets
//          char testFile[100]                          :the file name
of testing sets
//*************************************************************************
*******************
void DataReader::closeDataPointers(char trainFile[100],char
testFile[100])
{

      //variables
      char temp[100];

      //creates the size of those vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      //create the path string
      strcat(temp,"TrainingSets/");
      strcat(temp,trainFile);

      //close the pointer
      inTrainFile.clear();
      inTrainFile.close();

      //create the size of those vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      //creates the path string
      strcat(temp,"TestSets/");
      strcat(temp,testFile);
      //close the pointer
      inTestFile.clear();
      inTestFile.close();
}

//*************************************************************************
*******************
//bool readTrainingInput(vector<char> &primaryStructure,vector<char>
&secondaryStructure):
//public method that is called to read from a file the primary and
secondary structure of
//proteins that included in training sets. the pointers to the files
are initiated with
//initiateDataPointers method and closed with closeDataPointers.The
file must have a
//specific layout.
//Parameters:
//          vector<char> &primaryStructure      :returns the primary
structure of the protein
//          vector<char> &secondaryStructure:returns the secondary
structure of the protein
//Return:
//          true                                        :if there are
more sequences
//          false                                       :if the pointer
points to eof
//*************************************************************************
*******************
```

```
bool DataReader::readTrainingInput(vector<char>
&primaryStructure,vector<char> &secondaryStructure,char
proteinID[100])
{
	//variables
	char name[100];
	char temp[1000];

	primaryStructure.clear();

	inTrainFile >> name;

	for(int i=0;i<100;i++)
	{
		proteinID[i]='\0';
	}

	for(int i=0;i<1000;i++)
	{
		temp[i]='\0';
	}

	strcat(proteinID,name);

	//if end of file returns false
	if(inTrainFile.eof()) { return false;}

	inTrainFile.getline(temp,1000);
	inTrainFile.getline(temp,1000);
	//inTrainFile>> temp;

	//reads all the letters of primary structure until the "."
	//while(temp!='.')
	//{
		//primaryStructure.push_back(temp);
		//inTrainFile >> temp;
	//}

	for(int i=0;(i<1000) && (temp[i]!='\0');i++)
	{
		primaryStructure.push_back(temp[i]);
	}


	for(int i=0;i<1000;i++)
	{
		temp[i]='\0';
	}

	inTrainFile.getline(temp,1000);

	secondaryStructure.clear();

	//inTrainFile >> temp;

	//reads all the letters of secondary structure until the "."
	//while(temp!='.')
	//{
	//	secondaryStructure.push_back(temp);
	//	inTrainFile >> temp;
	//}
```

```
	for(int i=0;(i<1000) && (temp[i]!='\0');i++)
	{
		secondaryStructure.push_back(temp[i]);
	}

	return true;
}

//*****************************************************************
*********************
//bool readTestInput(vector<char> &primaryStructure,vector<char>
&secondaryStructure,
//                 char name[100]):
//public method that is called to read from a file the primary and
secondary structure of
//proteins that included in testing sets. The pointers to the files
are initiated with
//initiateDataPointers method and closed with closeDataPointers.The
file must have a
//specific layout.
//Parameters:
//          vector<char> &primaryStructure      :returns the primary
structure of the protein
//          vector<char> &secondaryStructure:returns the secondary
structure of the protein
//          char name[100]                              :the name of
the protein
//Return:
//          true                                        :if there are
more sequences
//          false                                       :if the pointer
points to eof
//*****************************************************************
*********************
bool DataReader::readTestInput(vector<char>
&primaryStructure,vector<char> &secondaryStructure,char
proteinID[100])
{
	/*
	//variables
	char temp;
	char name[100];

	primaryStructure.clear();

	inTestFile >> name;

	for(int i=0;i<100;i++) {proteinID[i]='\0';}

	strcat(proteinID,name);


	//if end of file returns false
	if(inTestFile.eof()){ return false;}

	inTestFile >> temp;

	//reads all the letters of primary structure until the "."
	while(temp!='.')
	{
```

```
	primaryStructure.push_back(temp);
	inTestFile >> temp;
}

//if there in low case letter replace it with capital case
for(int i=0;i<primaryStructure.size();i++)
{
	if(primaryStructure[i]>='a' && primaryStructure[i]<='z')
		primaryStructure[i]=primaryStructure[i]-32;
}


secondaryStructure.clear();

inTestFile >> temp;

//reads all the letters of secondary structure until the "."
while(temp!='.')
{
	secondaryStructure.push_back(temp);
	inTestFile >> temp;
}

return true;
*/
//variables
char name[100];
char temp[1000];

primaryStructure.clear();

inTestFile >> name;

for(int i=0;i<100;i++)
{
	proteinID[i]='\0';
}

for(int i=0;i<1000;i++)
{
	temp[i]='\0';
}

strcat(proteinID,name);

//if end of file returns false
if(inTestFile.eof()) { return false;}

inTestFile.getline(temp,1000);
inTestFile.getline(temp,1000);
//inTrainFile>> temp;

//reads all the letters of primary structure until the "."
//while(temp!='.')
//{
	//primaryStructure.push_back(temp);
	//inTrainFile >> temp;
//}

for(int i=0;(i<1000) && (temp[i]!='\0');i++)
{
```

```
			primaryStructure.push_back(temp[i]);
		}

		//if there in low case letter replace it with capital case
		for(int i=0;i<primaryStructure.size();i++)
		{
			if(primaryStructure[i]>='a' && primaryStructure[i]<='z')
				primaryStructure[i]=primaryStructure[i]-32;
		}

		for(int i=0;i<1000;i++)
		{
			temp[i]='\0';
		}

		inTestFile.getline(temp,1000);

		secondaryStructure.clear();

		//inTrainFile >> temp;

		//reads all the letters of secondary structure until the "."
		//while(temp!='.')
		//{
		//	secondaryStructure.push_back(temp);
		//	inTrainFile >> temp;
		//}

		for(int i=0;(i<1000) && (temp[i]!='\0');i++)
		{
			secondaryStructure.push_back(temp[i]);
		}

		return true;
}

//*****************************************************************
*********************
//void translateSecondaryStructure(vector<vector<char>>
&outputClassification,
//				vector<char> &secondaryStructure):
//public method that is called to replace all the letters of
secondary structure with
//the specific letter of their class
//Parameters:
//		vector<vector<char>> &outputClassification	:contains
a vector with the output classes of the
//		network. Each position of the vector of vector holds
//		the characteristic letter of the class which is
//		followed by the secondary structure letters of
//		that class
//		vector<char> &secondaryStructure	:contains the
secondary structure of the protein
//*****************************************************************
*********************
void DataReader::translateSecondaryStructure(vector< vector<char> >
&outputClassification,vector<char> &secondaryStructure)
{
		for(int i=0;i<secondaryStructure.size();i++)
		{
			for(int j=0;j<outputClassification.size();j++)
```

```
		{
			for(int k=0;k<outputClassification[j].size();k++)
			{

	if(secondaryStructure[i]==outputClassification[j][k])
				{

	secondaryStructure[i]=outputClassification[j][0];
					break;
				}
			}
		}
	}
}

//Created by Georgia - MSA
//*****************************************************************
*******************
//This function reads the msa dataset for each protein -- use only
with msa
//*****************************************************************
*******************
//encodingT contains the same vector of strings, each string
represents an aminoacid
//*****************************************************************
*******************
void DataReader::readEncodingMSA(vector<string> &encodingT,char
proteinID[100])
{
	//pointer to protein's file
	ifstream inFile;

	//variables
	string tempString="\0";
	string tempLine="\0";
	int aminoacid = 0;
	int digit = 0;
	char temp[100];

	//create the size of those vectors
	for(int i=0;i<100;i++) {temp[i]='\0';}


	//create the path string
	strcat(temp,"EncodingProfiles/msaFiles/");



	char c;
	for(int i=0;i<100;i++)
	{

		proteinID[i]=tolower(proteinID[i]);

	}


	strcat(temp,proteinID);
	//strcat(temp,".txt");
```

```
	strcat(temp,".hssp");

	//tolower(temp);



	//cout<<proteinID<<endl;

	//points to the file
	inFile.open(temp);

	if (!inFile) {
		cerr << "Unable to open file of protein "<<temp;
		int x=0;
		cin>>x;
		exit(1);
	}

	//clear this vector for the next protein
	encodingT.clear();

	//read msa encoding for each protein:

	while ( !inFile.eof() )
	{
		//read the next encoding digit
		inFile >> tempString;

		if(digit != numberOfResidues )
		{
			//store the previous digit
			tempLine=tempLine+tempString+" ";
			digit++;
			if(digit == numberOfResidues)
			{
				digit = 0;
				encodingT.push_back(tempLine);
				tempLine = "\0";
			}

		}
	}

}

vector<double> DataReader::splitValues (string stringEncoding)
{
	vector<double> doubleEncoding;
	istringstream iss(stringEncoding);

	copy(istream_iterator<double>(iss),
		istream_iterator<double>(),
		 back_inserter<vector<double> >(doubleEncoding));
	for(int i=0;i<doubleEncoding.size();i++)
		doubleEncoding[i]=doubleEncoding[i]/100.0;


	return doubleEncoding;
}
```

OutputData.h

```
#include "targetver.h"

#include <stdio.h>

#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>



using namespace std;
#ifndef OutputData_h
#define OutputData_h

//*****************************************************************
*********************
//class OutputData: This class illustrates all the functions that the
Bidirectional
//Recurent Neural Network needs to write to the output files
//*****************************************************************
*********************
class OutputData
{

private:

	//members of OutputData class
	ofstream printError;
	ofstream printErrorP;
	ofstream printOutput;
	ofstream printOutputTrain;
	ofstream printNetwork;
	ofstream printNetworkHTML;
	ofstream printEnsembleResults;
	char folderName[100];
	char HTMLName[100];
	char networkId[100];
	char ensembleResultsName[100];
	int ensembleOutputCounter;

public:

	OutputData(char id,char outputFolder[100]);
	  OutputData(char *id,char outputFolder[100]);
	~OutputData(void);
	char nameOfSimulation[200];

	//methods of OutputData class
	void createErrorFiles(vector<double>
trainingError,vector<double> testingError,
                        vector<double>
proteinTrainingError,vector<double> proteinTestingError);
```

```
      void createOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
                        vector<char> predictedSecondaryStructure,char
proteinName[100],double percentage,int c,vector<vector <double> >
vectoroutputresult,int flag);
      void createZOutputFileforFilter(vector<char>
primaryStructure,vector<char> secondaryStructure,
                        vector<char> predictedSecondaryStructure,char
proteinName[100],double percentage,int c,vector<vector <double> >
vectoroutputresult);
      void createNetworkFile(int hLayerOneSizeN,int
hLayerTwoSizeN,int hLayerOneSizeBN,
                        int hLayerTwoSizeBN,int hLayerOneSizeFN,int
hLayerTwoSizeFN,int activationFuncTypeN,
                        double learningRateN,double momentumN,int
windowSizeN,double qMinus1N,
                        double qPlus1N,int errorFunctionTypeN,int
temporaryN,int epochN,char trainFileN[100],
                        char testFileN[100],char
inputProfileN[100],char outputProfileN[100],double percentage);
      void closeOutputPointers(void);
        void closeAllPointers(void);
      void createEnsembleOutputFile(void);
      void closeEnsembleOutputFile(void);
      void printEnsembleOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
                  vector<char> predictedSecondaryStructure,char
proteinName[100], vector<vector <double> > vectoroutputresult1,
                  vector<vector <double> > vectoroutputresult2,
vector<vector <double> > vectoroutputresult3,
                  vector<vector <double> > vectoroutputresult4,
vector<vector <double> > vectoroutputresult5,
                  vector<vector <double> > vectoroutputresult6,
vector<vector <double> > vectoroutputresultensemble);
};
#endif
```

OutputData.cpp

```
#include "OutputData.h"
#include "targetver.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <iostream>
#include <math.h>
#include <ctype.h>
#include <vector>
#include <time.h>
#include <string>



using namespace std;

//
//**********************************************************************
*********************
//Constructor DataReader(void):initiates the DataReader
//**********************************************************************
*********************
OutputData::OutputData(char id,char outputFolder[100])
{
	//it takes the time from the system and creates a folder in the
DataOut folder
	//this folder will contain all tha output files of each
simulation
	//Also initiate the pointer to the output file and HTML output
file
	time_t rawtime;
	struct tm * timeinfo;


	networkId[0]=id;

	networkId[1]='\0';

	//cout<<networkId[0]<<endl;

    time ( &rawtime );
	timeinfo = localtime ( &rawtime );



	char temp[200];char temp3[200];

	//cout<<networkId<<endl;

	for(int i=0;i<200;i++)
	{
		temp[i]='\0';
		temp3[i]='\0';

	}

	for(int i=0;i<100;i++)
```

```
{
	folderName[i]='\0';
	HTMLName[i]='\0';
	nameOfSimulation[i]='\0';
}

//cout<<networkId<<endl;

strcat(temp,outputFolder);
//strcat(temp,asctime (timeinfo));
strcat(temp3,outputFolder);
//strcat(temp3,asctime (timeinfo));
//strcat(HTMLName,asctime(timeinfo));
strcat(nameOfSimulation,outputFolder);


//cout<<networkId<<endl;


//replace some characters
/*for(int i=0;i<100;i++){
	if(temp[i]=='\n')
	{
		temp[i]='\0';
	}
	if(temp[i]==' ')
	{
		temp[i]='-';
	}
	if(temp[i]==':')
	{
		temp[i]='_';
	}
}

for(int i=0;i<100;i++){
	if(temp3[i]=='\n')
	{
		temp3[i]='\0';
	}
	if(temp3[i]==' ')
	{
		temp3[i]='-';
	}
	if(temp3[i]==':')
	{
		temp3[i]='_';
	}
}*/

/*for(int i=0;i<100;i++){
	if(HTMLName[i]=='\n')
	{
		HTMLName[i]='\0';
	}
	if(HTMLName[i]==' ')
	{
		HTMLName[i]='_';;
	}
	if(HTMLName[i]==':')
	{
```

```
            HTMLName[i]='_';
      }
}*/

//create folder
//system("mkdir temp");

//create the string path
//strcat(temp,"\\");
//strcat(temp,asctime (timeinfo));
//strcat(temp3,"\\");
//strcat(temp3,asctime (timeinfo));

/*for(int i=0;i<100;i++)
{
      if(temp[i]=='\n')
      {
            temp[i]='\0';
      }
      if(temp[i]==' ')
      {
            temp[i]='-';
      }
      if(temp[i]==':')
      {
            temp[i]='_';
      }
}

for(int i=0;i<100;i++)
{
      if(temp3[i]=='\n')
      {
            temp3[i]='\0';
      }
      if(temp3[i]==' ')
      {
            temp3[i]='-';
      }
      if(temp3[i]==':')
      {
            temp3[i]='_';
      }
}*/

for(int i=0;i<100;i++)
{
      folderName[i]=temp[i];
}



strcat(temp,networkId);
strcat(temp3,networkId);
strcat(temp,"_Output.txt");
strcat(temp3,"_OutputForTrain.txt");
```

```
	//initiate pointer of the output file for test


	printOutput.open(temp);
	printOutputTrain.open(temp3);

	ensembleOutputCounter=0;



}
OutputData::OutputData(char *id,char outputFolder[100])
{

	  strcpy(networkId,id);

	char temp[200];

	//cout<<networkId<<endl;

	for(int i=0;i<200;i++)
	{
		temp[i]='\0';


	}

	for(int i=0;i<100;i++)
	{
		folderName[i]='\0';

	}

	strcat(temp,outputFolder);

	  for(int i=0;i<100;i++)
		{
		folderName[i]=temp[i];
		}

	strcat(temp,networkId);
	strcat(temp,"_Output.txt");
	  printOutput.open(temp);


}

//*********************************************************************
*********************
//Deconstructor DataReader(void)
//*********************************************************************
*********************
OutputData::~OutputData(void)
{

}

//*********************************************************************
*********************
//void createErrorFiles(vector<double> trainingError,vector<double>
testingError,
```

```
//                vector<double> proteinTrainingError,vector<double>
proteinTestingError):
//public method that is called to create the files that contains the
training and testing
//error per protein and per epoch
//Parameters:
//                vector<double> trainingError        :vector with
training errors per epoch
//                vector<double> testingError               :vector
with testing errors per epoch
//                vector<double> proteinTrainingError :vector with
training errors per protein
//                vector<double> proteinTestingError  :vector with
testing errors per protein
//*****************************************************************
*********************
void OutputData::createErrorFiles(vector<double>
trainingError,vector<double> testingError,
                  vector<double> proteinTrainingError,vector<double>
proteinTestingError)
{
      //variables
      char temp[100];
      char temp1[100];

      //create vectors
      for(int i=0;i<100;i++){
            temp[i]='\0';
            temp1[i]='\0';
      }

      //create the names of output files
      for(int i=0;i<100;i++)
      {
            temp[i]=folderName[i];
            temp1[i]=folderName[i];
      }

      //creates the string path and opens the output file with errors
per epoch
      strcat(temp,networkId);


      strcat(temp,"_error_per_epoch.txt");
      printError.open(temp);

      //write all the data to the file
      printError<<"No          Training Error          Test
Error\n";

printError<<"***********************************************\n\n";

      for(int i=0;i<trainingError.size();i++)
      {

      printError<<i<<".\t\t"<<trainingError[i]<<"\t\t"<<testingError[
i]<<endl;
      }

      //creates the string path and opens the output file with errors
per protein
```

```
	strcat(temp1,networkId);
	strcat(temp1,"_error_per_protein.txt");
	printErrorP.open(temp1);

	//write all the data to the file
	printErrorP<<"No		Training Error		Test
Error\n";

	printErrorP<<"*********************************************
\n\n";

	if(proteinTrainingError.size()<=20000)
	 for(int i=0;i<proteinTrainingError.size();i++)
	 {

	printErrorP<<i<<".\t\t"<<proteinTrainingError[i]<<"\t\t"<<prote
inTestingError[i]<<endl;

	 }
}

//*****************************************************************
*******************
//void OutputData::createOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
//		vector<char> predictedSecondaryStructure,char
proteinName[100],double percentage)):
//public method that is called  at the end of each epoch to write in
the output file the name of
//a protein, its primary structure, its secondary structure, its
predicted secondary structure and
//the percentage of succes for the specific simulation
//Parameters:
//		vector<char> primaryStructure
	:protein's primary structure
//		vector<char> secondaryStructure
	:protein's secondary structure
//		vector<char> predictedSecondaryStructure
	:protein's predicted secondary structure
//		char proteinName[100]
	:protein's name
//		double percentage
	:persentage of succes for a simulation
//*****************************************************************
*******************
void OutputData::createOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
		vector<char> predictedSecondaryStructure,char
proteinName[100],double percentage,int trainOrTest,
			vector<vector <double> >
vectoroutputresult,int flag)
{
	if(trainOrTest==2)
	{
	//writes the name of a protein and the percentage of success
	printOutput<<proteinName<<" Correctness
Percentage:"<<percentage<<"%"<<endl;
	//cout<<proteinName<<endl;
	//writes the primary structure
	printOutput<<"primaryStructure		:";
	for(int i=0;i<primaryStructure.size();i++)
```

```
	{
		printOutput<<primaryStructure[i];
	}
	printOutput<<endl;

	//writes the secondary structure
	printOutput<<"secondaryStructure         :";
	for(int i=0;i<secondaryStructure.size();i++)
	{
		printOutput<<secondaryStructure[i];
	}
	printOutput<<endl;

	//writes the predicted secondary structure
	printOutput<<"predictedSecondaryStructure:";
	for(int i=0;i<predictedSecondaryStructure.size();i++)
	{
		printOutput<<predictedSecondaryStructure[i];
	}
	printOutput<<endl;

	  //writes the
	//printOutput<<"predictedSecondaryStructure:";
	 /* if(flag==0){
	   printOutput<<"Output Value of:"<<endl;
	  for(int i=0;i<vectoroutputresult.at(1).size();i++){

		if(i==0)
		    printOutput<<"H\t";
		else if(i==1)
		    printOutput<<"L\t";
		else if(i==2)
		    printOutput<<"E\t";*/
	/*for(int k=0;k<vectoroutputresult.size();k++)
	{
		    printOutput.precision(4);
		printOutput<<vectoroutputresult.at(k).at(i);
		    if(k!=vectoroutputresult.size()-1)
		        printOutput<<",";
	}*/
	  printOutput<<endl;
	  /*}
	  }*/
	  /*else
	printOutput<<endl;*/
	}
	else
	{
	//writes the name of a protein and the percentage of success
	printOutputTrain<<proteinName<<" Correctness
Percentage:"<<percentage<<"%"<<endl;
	//cout<<proteinName<<endl;
	//writes the primary structure
	printOutputTrain<<"primaryStructure           :";
	for(int i=0;i<primaryStructure.size();i++)
	{
		printOutputTrain<<primaryStructure[i];
	}
	printOutputTrain<<endl;

	//writes the secondary structure
```

```
	printOutputTrain<<"secondaryStructure         :";
	for(int i=0;i<secondaryStructure.size();i++)
	{
		printOutputTrain<<secondaryStructure[i];
	}
	printOutputTrain<<endl;

	//writes the predicted secondary structure
	printOutputTrain<<"predictedSecondaryStructure:";
	for(int i=0;i<predictedSecondaryStructure.size();i++)
	{
		printOutputTrain<<predictedSecondaryStructure[i];
	}
	printOutputTrain<<endl;
	  /*for(int i=1;i<OtH.size();i++)
	{
		printOutput<<OtH.at(i)<<",";
	}
	printOutput<<endl;*/
	}
}




//*****************************************************************
*********************
//void createNetworkFile(int hLayerOneSizeN,int hLayerTwoSizeN,int
hLayerOneSizeBN,
//                int hLayerTwoSizeBN,int hLayerOneSizeFN, int
hLayerTwoSizeFN,int activationFuncTypeN,
//                double learningRateN,double momentumN,int
windowSizeN,double qMinus1N,double qPlus1N,
//                int errorFunctionTypeN,int temporaryN,int
epochN,char trainFileN[100],char testFileN[100],
//                char inputProfileN[100],char
outputProfileN[100],double percentage):
//public method that is called to create a file with the parameters
of the network and an HTML file
//with information about the simulation
//Parameters:
//                int hLayerOneSizeN                  :hidden layer
one size
//                int hLayerTwoSizeN                  :hidden layer
two size
//                int hLayerOneSizeBN                 :Backward
hidden layer one size
//                int hLayerTwoSizeBN                 :Backward
hidden layer two size
//                int hLayerOneSizeFN                 :Forward hidden
layer one size
//                int hLayerTwoSizeFN                 :Foeward hidden
layer two size
//                int activationFuncTypeN       :number that
corresponds to an activation function
//                double learningRateN          :learning rate
//                double momentumN              :momentum
//                int windowSizeN                     :the window
size for each specific residue
```

```
//                double qMinus1N                     :the q operator
for forward recurrent neural network
//                double qPlus1N                      :the q operator
for backward recurrent neural network
//                int errorFunctionTypeN        :number that
corresponds to an error function
//                int sN                                    :the
operator s
//                int epochN                          :number of
iterations
//                char trainFileN[100]          :the file name of
training set
//                char testFileN[100]                 :the file name
of testing set
//                char inputProfileN[100]       :the file name of
input profile
//                char outputProfileN[100]      :the file name of
output profile
//                double percentage             :the percentage of
succes
//*****************************************************************
*********************
void OutputData::createNetworkFile(int hLayerOneSizeN,int
hLayerTwoSizeN,int hLayerOneSizeBN,
                  int hLayerTwoSizeBN,int hLayerOneSizeFN, int
hLayerTwoSizeFN,int activationFuncTypeN,
                  double learningRateN,double momentumN,int
windowSizeN,double qMinus1N,double qPlus1N,
                  int errorFunctionTypeN,int sN,int epochN,char
trainFileN[100],char testFileN[100],
                  char inputProfileN[100],char
outputProfileN[100],double percentage)
{

      //variables
      char temp[100];
      char temp1[100];

      //create the vectors
      for(int i=0;i<100;i++)
      {
            temp[i]='\0';
      }

      for(int i=0;i<100;i++)
      {
            temp1[i]='\0';
      }

      for(int i=0;i<100;i++)
      {
            temp[i]=folderName[i];
            temp1[i]=folderName[i];

      }

      //creates the string path and opens the output file with
parameters
      strcat(temp,networkId);
      strcat(temp,"_Network_Specifications.txt");
      printNetwork.open(temp);
```

```
	//creates the string path and opens the HTML file with results
	//strcat(temp1,"DataOut//");
	strcat(temp1,HTMLName);
	strcat(temp1,networkId);
	strcat(temp1,"_1.html");
	printNetworkHTML.open(temp1);

	//writes the information to the output file with parameters
	printNetwork<<"Network Specification"<<endl;
	printNetwork<<"**********************"<<endl;
	printNetwork<<"Size of Hidden Layer 1:"<<hLayerOneSizeN<<endl;
	printNetwork<<"Size of Hidden Layer 2:"<<hLayerTwoSizeN<<endl;
	printNetwork<<"Size of Forward Hidden Layer
1:"<<hLayerOneSizeFN<<endl;
	printNetwork<<"Size of Forward Hidden Layer
2:"<<hLayerTwoSizeFN<<endl;
	printNetwork<<"Size of Backward Hidden Layer
1:"<<hLayerOneSizeBN<<endl;
	printNetwork<<"Size of Backward Hidden Layer
2:"<<hLayerTwoSizeBN<<endl;
	printNetwork<<"Type of Activation Function (Hidden
Neurons):"<<activationFuncTypeN<<endl;
	printNetwork<<"Learning Rate:"<<learningRateN<<endl;
	printNetwork<<"Momentum:"<<momentumN<<endl;
	printNetwork<<"Window Size:"<<windowSizeN<<endl;
	printNetwork<<"q-1:"<<qMinus1N<<endl;
	printNetwork<<"q+1:"<<qPlus1N<<endl;
	printNetwork<<"Type of error Function (Hidden
Neurons):"<<errorFunctionTypeN<<endl;
	printNetwork<<"s:"<<sN<<endl;
	printNetwork<<"Number of Iterrations: "<<epochN<<endl;
	printNetwork<<"Name of Training File: "<<trainFileN<<endl;
	printNetwork<<"Name of Testing File: "<<testFileN<<endl;
	printNetwork<<"Name of Input Profile: "<<inputProfileN<<endl;
	printNetwork<<"Name of Output Profile: "<<outputProfileN<<endl;

	//writes the information to the HTML file with the simulation
results
	printNetworkHTML<<"<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML
1.0 Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd\">"<<endl;
	printNetworkHTML<<"<html
xmlns=\"http://www.w3.org/1999/xhtml\">"<<endl;
	printNetworkHTML<<"<head>"<<endl;
	printNetworkHTML<<"<title>Website Title</title>"<<endl;
	printNetworkHTML<<"<meta http-equiv=\"Content-Type\"
content=\"text/html; charset=UTF-8\" />"<<endl;
	printNetworkHTML<<"<link rel=\"stylesheet\" type=\"text/css\"
href=\"style.css\" media=\"screen\" />"<<endl;
	printNetworkHTML<<"</head>"<<endl;
	printNetworkHTML<<"<body>"<<endl;
	printNetworkHTML<<"<div id=\"content\">"<<endl;
	printNetworkHTML<<"<div id=\"header\">"<<endl;
	printNetworkHTML<<"<h1><a href=\"#\">Protein Secondary
Structure Prediction </a></h1>"<<endl;
	printNetworkHTML<<"<h2>University of Cyprus </h2>"<<endl;
	printNetworkHTML<<"</div>"<<endl;
	printNetworkHTML<<"<div id=\"navigation\">"<<endl;
	printNetworkHTML<<"<ul>"<<endl;
	printNetworkHTML<<"<li><a href=\"#\">Report</a></li>"<<endl;
```

```
      printNetworkHTML<<"<li><a href=\""<< HTMLName
<<"_2.html\">Output Files</a></li>"<<endl;
      printNetworkHTML<<"</ul>"<<endl;
      printNetworkHTML<<"</div>"<<endl;
      printNetworkHTML<<"<div class=\"right\">"<<endl;
      printNetworkHTML<<"<h2>Report of: "<< HTMLName <<"</h2>"<<endl;
      //printNetworkHTML<<"<p>This website illustrates the
Bidirectional Recurrent Neural Network specifications for Protein
Secondary Structure Prediction. The success prediction of this
experiment is: "<<percentage<<"% </a>.</p>"<<endl;
      //printNetworkHTML<<"</div>"<<endl;
      printNetworkHTML<<"<div class=\"right\">"<<endl;
      printNetworkHTML<<"<h2>Network's Specifications: </h2>"<<endl;
      printNetworkHTML<<"<p>Size of Hidden Layer 1: "<<
hLayerOneSizeN<<"</p>"<<endl;
      printNetworkHTML<<"<p>Size of Hidden Layer 2: "<<hLayerTwoSizeN
<<" </p>"<<endl;
      printNetworkHTML<<"<p>Size of Forward Hidden Layer 1:
"<<hLayerOneSizeFN <<" </p>"<<endl;
      printNetworkHTML<<"<p>Size of Forward Hidden Layer 2:
"<<hLayerTwoSizeFN <<" </p>"<<endl;
      printNetworkHTML<<"<p>Size of Backward Hidden Layer 1:
"<<hLayerOneSizeBN <<" </p>"<<endl;
      printNetworkHTML<<"<p>Size of Backward Hidden Layer 2: "<<
hLayerTwoSizeBN<<" </p>"<<endl;
      printNetworkHTML<<"<p>Type of Activation Function (Hidden
Neurons): "<<activationFuncTypeN <<" </p>"<<endl;
      printNetworkHTML<<"<p>Learning Rate: "<<learningRateN <<"
</p>"<<endl;
      printNetworkHTML<<"<p>Momentum: "<<momentumN <<" </p>"<<endl;
      printNetworkHTML<<"<p>Window Size: "<<windowSizeN <<"
</p>"<<endl;
      printNetworkHTML<<"<p>q-1: "<< qMinus1N<<" </p>"<<endl;
      printNetworkHTML<<"<p>q+1: "<<qPlus1N <<" </p>"<<endl;
      printNetworkHTML<<"<p>Type of error Function (Hidden Neurons):
"<<errorFunctionTypeN <<" </p>"<<endl;
      printNetworkHTML<<"<p>s: "<<sN <<" </p>"<<endl;
      printNetworkHTML<<"<p>Number of Iterrations: "<<epochN <<"
</p>"<<endl;
      printNetworkHTML<<"<p>Name of Training File: "<< trainFileN<<"
</p>"<<endl;
      printNetworkHTML<<"<p>Name of Testing File: "<< testFileN<<"
</p>"<<endl;
      printNetworkHTML<<"<p>Name of Input Profile: "<<inputProfileN
<<" </p>"<<endl;
      printNetworkHTML<<"<p>Name of Output Profile: "<<outputProfileN
<<" </p>"<<endl;
      printNetworkHTML<<"<h2> </h2>"<<endl;
      printNetworkHTML<<"<a href=\"#\" title=\"Link Title\"><img
src=\"pic1.jpg\" alt=\"Something\" width=\"695\" height=\"367\"
style=\"border: 3px solid #ddd;\" /></a>"<<endl;
      printNetworkHTML<<"</div>"<<endl;
      printNetworkHTML<<"<div style=\"clear: both;\"> </div>"<<endl;
      printNetworkHTML<<"<div id=\"footer\">"<<endl;
      printNetworkHTML<<"&copy; Copyright by <a href=\"#\">University
of Cyprus</a> | Designed by <a href=\"#\">Michalis
Agathocleous</a></a>"<<endl;
      printNetworkHTML<<"</div>"<<endl;
      printNetworkHTML<<"</div>"<<endl;
      printNetworkHTML<<"</body>"<<endl;
      printNetworkHTML<<"</html>"<<endl;
```

```
	//close the pointers
	printNetwork.close();
	printNetworkHTML.close();

}

//*********************************************************************
*********************
//void closeAllPointers(): public method that is called to close all
the pointers to
//specific files
//*********************************************************************
*********************
void OutputData::closeAllPointers(){

	printError.close();
	printErrorP.close();
	printOutput.close();
}


void OutputData::closeOutputPointers(){
    printOutput.close();
}


void OutputData::createEnsembleOutputFile(){

	printEnsembleResults.open("ensembleResults.txt");

	printEnsembleResults<<"ID\t\tpdbCode\t\tchain\t\tAA\t\tObsSS\t\
tPredSS\t\tH\t\tE\t\tL\t\tBRNN1_H\t\tBRNN1_E\t\tBRNN1_L\t\tBRNN2_H\t\
tBRNN2_E\t\tBRNN2_L\t\tBRNN3_H\t\tBRNN3_E\t\tBRNN3_L\t\tBRNN4_H\t\tBR
NN4_E\t\tBRNN4_L\t\tBRNN5_H\t\tBRNN5_E\t\tBRNN5_L\t\tBRNN6_H\t\tBRNN6
_E\t\tBRNN6_L"<<endl;
	printEnsembleResults<<"*************************************
*********************************************************************
*********************************************************************
*********************************************************************
************"<<endl;

}



void OutputData::closeEnsembleOutputFile(){

	printEnsembleResults.close();

}


void OutputData::printEnsembleOutputFile(vector<char>
primaryStructure,vector<char> secondaryStructure,
		vector<char> predictedSecondaryStructure,char
proteinName[100], vector<vector <double> > vectoroutputresult1,
		vector<vector <double> > vectoroutputresult2,
vector<vector <double> > vectoroutputresult3,
		vector<vector <double> > vectoroutputresult4,
vector<vector <double> > vectoroutputresult5,
```

```
            vector<vector <double> > vectoroutputresult6,
vector<vector <double> > vectoroutputresultensemble){



      //cout<<primaryStructure.size()<<"
"<<predictedSecondaryStructure.size()<<endl;
      for(int i=0;i<primaryStructure.size();i++)
      {
            ensembleOutputCounter++;

      printEnsembleResults<<ensembleOutputCounter<<"\t\t"<<proteinNam
e[0]<<proteinName[1]<<proteinName[2]<<proteinName[3]<<"\t\t"<<protein
Name[4]<<"\t\t"<<primaryStructure[i]<<"\t\t"<<secondaryStructure[i]<<
"\t\t"<<predictedSecondaryStructure[i]<<"\t\t"<<vectoroutputresultens
emble[i][0]<<"\t\t"<<vectoroutputresultensemble[i][2]<<"\t\t"<<vector
outputresultensemble[i][1]<<"\t\t"<<vectoroutputresult1[i][0]<<"\t\t"
<<vectoroutputresult1[i][2]<<"\t\t"<<vectoroutputresult1[i][1]<<vecto
routputresult2[i][0]<<"\t\t"<<vectoroutputresult2[i][2]<<"\t\t"<<vect
oroutputresult2[i][1]<<vectoroutputresult3[i][0]<<"\t\t"<<vectoroutpu
tresult3[i][2]<<"\t\t"<<vectoroutputresult3[i][1]<<vectoroutputresult
4[i][0]<<"\t\t"<<vectoroutputresult4[i][2]<<"\t\t"<<vectoroutputresul
t4[i][1]<<vectoroutputresult5[i][0]<<"\t\t"<<vectoroutputresult5[i][2
]<<"\t\t"<<vectoroutputresult5[i][1]<<vectoroutputresult6[i][0]<<"\t\
t"<<vectoroutputresult6[i][2]<<"\t\t"<<vectoroutputresult6[i][1]<<end
l;
      }

}
```