Ατομική Διπλωματική Εργασία

ΔΙΕΡΕΥΝΗΣΗ ΜΕΘΟΔΩΝ ΕΚΠΑΙΔΕΥΣΗΣ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ ΑΜΦΙΔΡΟΜΗΣ ΑΝΑΔΡΑΣΗΣ ΓΙΑ ΠΡΟΒΛΕΨΗ ΔΕΥΤΕΡΟΤΑΓΟΥΣ ΔΟΜΗΣ ΠΡΩΤΕΪΝΩΝ

Γεωργία Χριστοδούλου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διερεύνηση Μεθόδων Εκπαίδευσης Νευρωνικών Δικτύων Αμφίδρομης Ανάδρασης Για Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών

Γεωργία Χριστοδούλου

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

Είμαι σήμερα εδώ, με μια ολοκληρωμένη διπλωματική εργασία, που ήταν το αποτέλεσμα της συνεργασίας πέντε σπουδαίων ανθρώπων που είχα την τύχη και την τιμή να βρω στο τέλος της φοιτητικής μου πορείας στο Πανεπιστήμιο Κύπρου. Οι άνθρωποι αυτοί, μου έδειξαν εμπιστοσύνη και πίστεψαν στις δυνάμεις μου. Με στήριξαν και με βοήθησαν ο κάθε ένας με την ιδιότητα και τις γνώσεις του.

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή μου, Δρ. Χρίστο Χριστοδούλου που ακούραστα και επίμονα με καθοδηγούσε και με ενθάρρυνε, όπως επίσης και τον Δρ. Βασίλειο Προμπονά, Λέκτορα του τμήματος Βιολογικών Επιστημών του Πανεπιστημίου Κύπρου, του οποίου οι γνώσεις του στον τομέα της βιολογίας ήταν πολύ σημαντικές για την εργασία μου.

Εν συνεχεία, πολλές ευχαριστίες για τον καλό μου συνεργάτη και διδακτορικό φοιτητή Αντώνη Αντωνίου, που με άντεξε όλο αυτόν τον καιρό και ήταν δίπλα μου στις προσπάθειες μου, στα δύσκολα αλλά και στις επιτυχίες.

Πολλές ευχαριστίες επίσης, στον μεταπτυχιακό φοιτητή Μιχάλη Αγαθοκλέους και στον διδακτορικό φοιτητή Βασίλη Βασιλειάδη, γιατί συνέβαλαν και αυτοί με τις γνώσεις και τις εμπειρίες τους στο να ολοκληρωθεί αυτή η εργασία.

Τέλος ένα μεγάλο ευχαριστώ στην οικογένεια μου, για την στήριξη, την ενθάρρυνση, και κυρίως την αγάπη που μου πρόσφεραν την δύσκολη αυτή περίοδο.

Σας ευχαριστώ,

Γεωργία Χριστοδούλου

Περίληψη

Οι πρωτεΐνες, είναι βιολογικά μακρομόρια, τα οποία είναι πολύ σημαντικά για κάθε ζωντανό οργανισμό. Η σημαντική λειτουργικότητα των πρωτεϊνών μπορεί να εκδηλωθεί μόνο στην τρισδιάστατη τους μορφή, την οποία μόνο για λίγες σχετικά πρωτεΐνες γνωρίζουμε μέχρι σήμερα. Για τις πρωτεΐνες των οποίων γνωρίζουμε μόνο την ακολουθία των αμινοξέων τους, θα ήταν επιθυμητό να γνωρίζαμε την πολύ σημαντική τρισδιάστατη τους μορφή, δηλαδή τον τρόπο με τον οποίο διπλώνεται η πολυπεπτιδική τους αλυσίδα στον χώρο (protein folding problem). Οι πειραματικές μέθοδοι προσδιορισμού της τριτοταγούς δομής των πρωτεϊνών εξακολουθούν να είναι πολύπλοκες, δαπανηρές και χρονοβόρες, γι αυτό έχει αναπτυχθεί μια κατηγορία μεθόδων για την πρόβλεψη των χαρακτηριστικών της δομής των πρωτεϊνών με δεδομένη την αλληλουχία τους. Στόχος αυτής της διπλωματικής εργασίας είναι η διερεύνηση μεθόδων εκπαίδευσης νευρωνικών δικτύων αμφίδρομης ανάδρασης για πρόβλεψη δευτεροταγούς δομής πρωτεϊνών. Το σύστημα υλοποιήθηκε από τον Αγαθοκλέους (Αγαθοκλέους, 2009) και βασίζεται στην αρχιτεκτονική που εισήγαγε πρώτος στον χώρο ο Baldi και οι συνεργάτες του (Baldi et al., 1999, Baldi et al., 2000).

Η μέθοδος που προσεγγίζεται σε αυτήν την εργασία, κρατά σταθερή την αρχιτεκτονική αυτή, αλλά διαμορφώνει τον αλγόριθμο μάθησης και τον τρόπο προσαρμογής των βαρών του δικτύου, ώστε το σύστημα να επεξεργάζεται περισσότερη πληροφορία όσον αφορά μια συγκεκριμένη πρωτεΐνη κάθε χρονική στιγμή. Με την εφαρμογή της μεθόδου της πολλαπλής στοίχισης πρωτεϊνών για την κωδικοποίηση των δεδομένων εισόδου, τα ποσοστό επιτυχίας ήταν μέχρι και 74%, συγκρινόμενο με τα αποτελέσματα του Baldi (73.6%). Επιπρόσθετα, εφαρμόστηκαν πολλές εμπειρικές τεχνικές βελτιστοποίησης και νέες μέθοδοι ανάλυσης αποτελεσμάτων (SOV). Με την αλλαγή της αρχιτεκτονικής και την εφαρμογή της μεθόδου *winner takes all*, το SOV ποσοστό αυξάνεται μέχρι και 65% και το Q3 ποσοστό μέχρι και 76%, συγκρίσιμο με το 75.1% του δικτύου του Baldi (Baldi et al., 1999), ο οποίος χρησιμοποιεί ένα συνδυασμό από 6 τέτοια δίκτυα. Χρησιμοποιώντας *Hidden Markov Models* (HMM) για φιλτράρισμα της εξόδου, το SOV ποσοστό αυξάνεται μέχρι και 70%.

Περιεχόμενα

Κεφάλαιο **1** **Εισαγωγή** **1**

1.1 Βιολογικό Υπόβαθρο 2

1.1.1 Πρωτεΐνες και αμινοξέα 2

1.1.2 Δομή πρωτεϊνών 4

1.1.3 Ρόλος πρωτεϊνών 6

1.1.4 Πρόβλημα *protein folding* 7

1.2 Σχετική έρευνα 8

1.3 Νευρωνικά Δίκτυα 12

1.3.1 Πολυστρωματικά δίκτυα perceptron εμπρόσθιου περάσματος 18

1.3.2 Πολυστρωματικά δίκτυα perceptron με ανάδραση 20

1.3.3 Αλγόριθμος μάθησης με ανάστροφη μετάδοση λάθους 22

1.4 Γενετικοί Αλγόριθμοι 23

Κεφάλαιο **2** **Προηγούμενη Εργασία** **27**

2.1 Αρχιτεκτονική BRNN 28

2.2 Αρχικοποίηση BRNN 30

2.3 Υλοποίηση BRNN 34

Κεφάλαιο **3** **Επεξεργασία Δεδομένων** **41**

3.1 Χρήση MSA profiles 42

3.2 HSSP database 44

3.3 Επεξεργασία δ εδομένων εισόδου για το σύστημα 48
3.3.1 Δημιουργία parsers για διαχείριση αρχείων 48
3.4 Επεξεργασία δεδομένων εξόδου του συστήματος 56
3.4.1 Confusion Matrices 56
3.4.2 Segment OVerlap (SOV) 60

Κεφάλαιο **4** **Σχεδίαση Και Υλοποίηση 69**

4.1 Αλλαγή momentum 70
4.2 Αλλαγή χρονικών μεταβλητών 71
4.3 Υλοποίηση MSA profiles κωδικοποίησης 73
4.4 Υλοποίηση επιλογής συνάρτησης ενεργοποίησης 78
4.4.1 Υπερβολική Εφαπτομένη 80
4.4.2 Γραμμική Συνάρτηση 83
4.5 Τυχαιοποίηση συνόλου δεδομένων 86
4.6 Ενσωμάτωση γενετικών αλγορίθμων 88

Κεφάλαιο **5** **Πειράματα και Ανάλυση Αποτελεσμάτων 91**

5.1 Πειράματα χωρίς την χρήση MSA profiles 92
5.1.1 Πειράματα και αποτελέσματα με αλλαγή τιμών των παραμέτρων του δικτύου 92
5.1.2 Χρήση υπερβολικής εφαπτομένης 100
5.2 Πειράματα με την χρήση MSA profiles 101
5.2.1 Πειράματα χωρίς την χρήση της τυχαιοποίησης συνόλου εκπαίδευσης 102
5.2.2 Πειράματα με την χρήση της τυχαιοποίησης συνόλου εκπαίδευσης 105
5.2.3 Πειράματα με συνδυασμό συναρτήσεων ενεργοποίησης 110
5.2.4 Αποτελέσματα δικτύου με Segment OVerlap score 114

5.2.5 Αποτελέσματα των confusion matrices 117
5.3 Αποτελέσματα γενετικών αλγορίθμων σε συνδυασμό με MSA 121
5.4 Αποτελέσματα τροποποίησης της αρχιτεκτονικής του συστήματος 130
5.4.1 Αποτελέσματα τροποποίησης της αρχιτεκτονικής σε συνδυασμό με WTA
5.5 *Φιλτράρισμα* προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών 137
5.5.1 Αποτελέσματα φιλτραρίσματος προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών με χρήση HMM 138
5.6 Γενικές παρατηρήσεις και συζήτηση 143

Κεφάλαιο **6** **Συμπεράσματα Και Μελλοντική Εργασία 147**

6.1 Συμπεράσματα 148
6.2 Μελλοντική Εργασία 151

Βιβλιογραφία 155

Παράρτημα Α Α-1

Νευρωνικό Δίκτυο Αμφίδρομης Ανάδρασης Α-2

Παράρτημα Β Β-1

Perl Parsers Β-2

Παράρτημα Γ Γ-1

Γενετικοί Αλγόριθμοι Γ-2

Παράρτημα Δ Δ-1

Αρχείο Δεδομένων Εισόδου Δ-2

Παράρτημα Ε Ε-1

Αρχείο Δεδομένων Εξόδου Ε-2

Παράρτημα Στ Στ-1

Αρχείο Κωδικοποίησης Εισόδου με MSA profiles Στ-2

Αρχείο Κωδικοποίησης Εισόδου χωρίς MSA profiles Στ-3

Παράρτημα Ζ Ζ-1

Αρχείο Κωδικοποίησης Εξόδου Ζ-2

Παράρτημα Η Η-1

Αρχείο Παραμέτρων Αρχικοποίησης Η-2

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 Πολυστρωματικά δίκτυα perceptron εμπρόσθιου περάσματος

1.3.2 Πολυστρωματικά δίκτυα perceptron με ανάδραση

1.3.3 Αλγόριθμος μάθησης με ανάστροφη μετάδοση λάθους

1.4 Γενετικοί Αλγόριθμοι

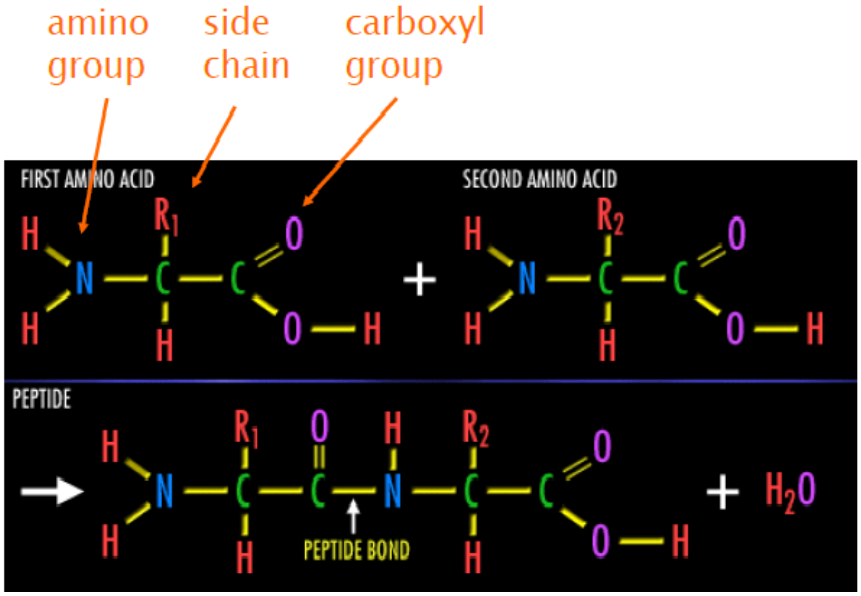
1.1 Βιολογικό υπόβαθρο

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 τριτοταγής 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.

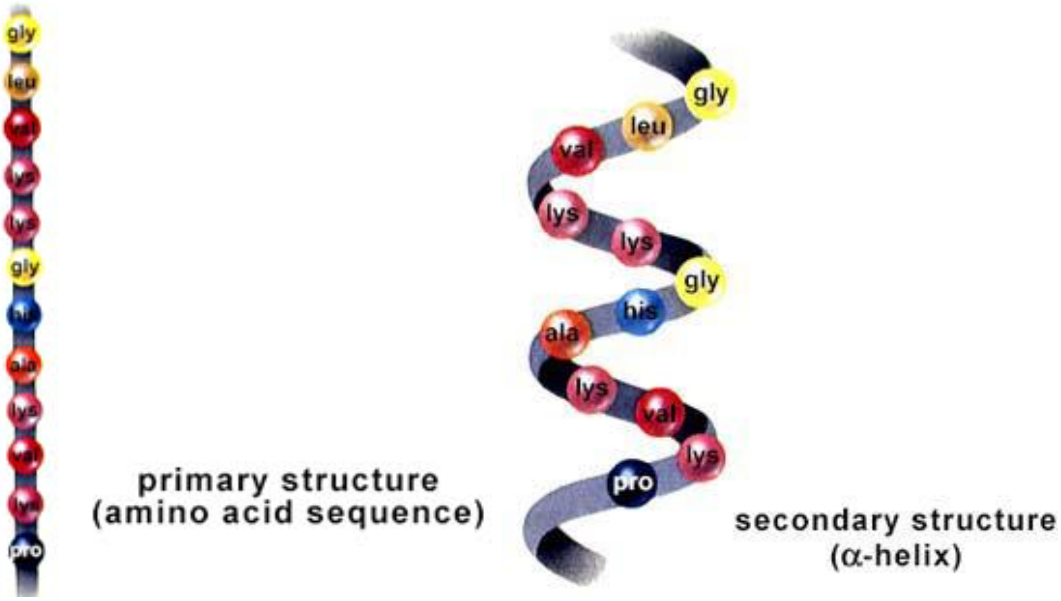
1.1.2 Δομή πρωτεϊνών

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 «διπλώνεται» στον χώρο σχηματίζοντας μια συγκεκριμένη δομή και τι δίνει στις πρωτεΐνες την ευστάθεια τους;

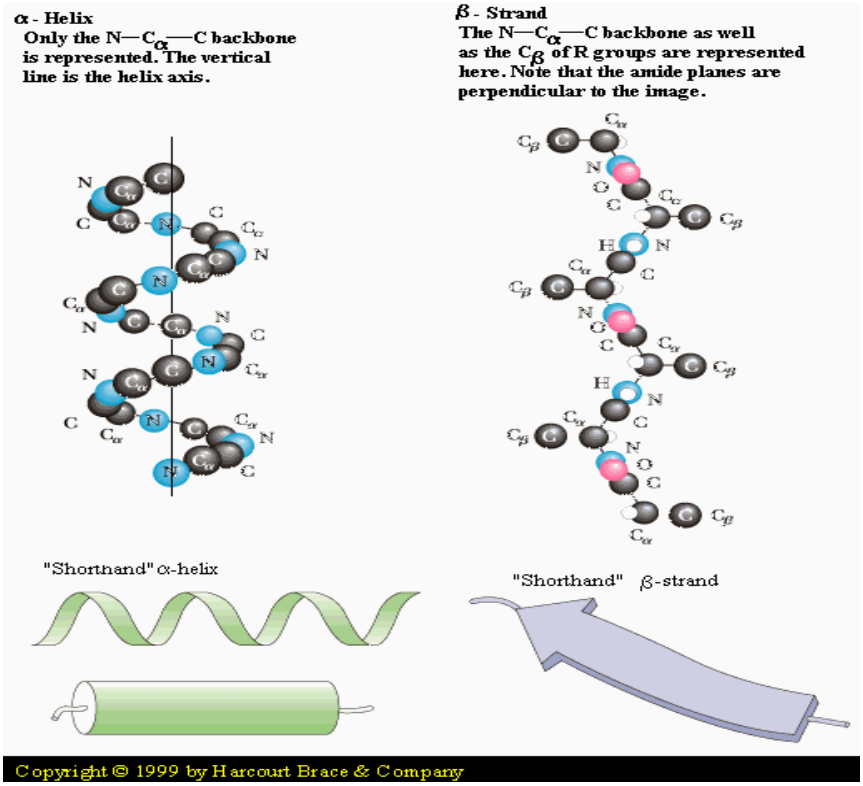
Είναι γενικά αποδεκτό ότι, η ακολουθία των διαφόρων αμινοξέων από τα οποία αποτελείται η πρωτεΐνη, περιέχει όλη εκείνη την πληροφορία που απαιτείται ώστε να αποκτήσει η πρωτεΐνη την τρισδιάστατη δομή της σε συγκεκριμένες συνθήκες. Οι πιθανές αλληλεπιδράσεις ανάμεσα σε αυτά τα αμινοξέα είναι πολλών τύπων, συμπεριλαμβανομένων ηλεκτροστατικών αλληλεπιδράσεων, δυνάμεων Van de 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. 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. 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. Παρά τις μεγάλες ερευνητικές προσπάθειες και την πρόοδο που σημειώνεται εδώ και δεκαετίες, το πρόβλημα αυτό εξακολουθεί να παραμένει άλυτο.

Μια σημαντική 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 επιτυχημένη πρόβλεψη της δευτεροταγούς δομής, μπορεί να αποτελέσει τη βάση για πρόβλεψη της τριτοταγούς δομής, καθώς επιβάλλει τοπικούς δομικούς περιορισμούς.

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

1.2 Σχετική έρευνα

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

Οι στατιστικές μέθοδοι είναι μέθοδοι πρόβλεψης της δευτεροταγούς δομής πρωτεϊνών, οι οποίες είναι βασισμένες σε στατιστικούς κανόνες. 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). Η πιο ευρέως διαδεδομένη στατιστική μέθοδος πρόβλεψης δευτεροταγούς δομής είναι η μέθοδος Chou and Fasman (Chou and Fasman, 1974). Είναι βασισμένη σε αναλύσεις των συχνοτήτων του κάθε αμινοξέως σε δευτεροταγείς δομές (α-helices, β-sheets και turns). Στην συνέχεια κατασκευάζεται ένας πίνακας πιθανοτήτων για την εμφάνιση του κάθε αμινοξέως σε κάθε τύπο δευτεροταγούς δομής και αυτές οι πιθανότητες χρησιμοποιούνται για να προβλεφτεί η μορφή της δευτεροταγούς δομής, δεδομένου μιας άγνωστης ακολουθίας. Η μέθοδος αυτή είναι περίπου 50% – 60% ακριβής με βάση το σκορ Q3 (υποκεφάλαιο 3.4.1, εξίσωση 3.1) στην πρόβλεψη της δευτεροταγούς δομής μιας άγνωστης πρωτεΐνης.

Στην συνέχεια, πιο ακριβείς στατιστικές μέθοδοι παρουσιάστηκαν. Η μέθοδος αυτή είναι η GOR (Garnier – Osguthorpe – Robson) (Garnier et al., 1978), η οποία όπως και η Chou and Fasman, αναλύει την τριτοταγή δομή γνωστών πρωτεϊνών για να εξαγάγει κάποιες πιθανότητες για helix (H), extended (E) και Coil (C). Η GOR μέθοδος, υπολογίζει τις πιθανότητες για κάθε αμινοξύ λαμβάνοντας υπόψη και την πιθανότητα το αμινοξύ να δώσει μια συγκεκριμένη δευτεροταγή δομή, δεδομένου των γειτονικών του αμινοξέων. Χρησιμοποιούν γι’ αυτό 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).

Μέθοδοι όπως ΤΝΔ (υποκεφάλαιο 1.3), Support Vector Machines και HMM (Zvelebil and Baum, 2008), 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. Οι μέθοδοι αυτοί 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.

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%.

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 ποσοστό (υποκεφάλαιο 3.4.2) είναι πιο ακριβές σε σύγκριση με άλλες μεθόδους (66%).

1.3 Νευρωνικά Δίκτυα

Τα Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ), είναι εμπνευσμένα από τον τομέα της Τεχνητής Νοημοσύνης και των Υπολογιστικών Συστημάτων και αποτελούν ένα μαθηματικό μοντέλο, εμπνευσμένο από τον τρόπο λειτουργίας του ανθρώπινου εγκεφάλου. Πρακτικά, το μοντέλο αυτό προσπαθεί να βελτιστοποιήσει (μεγιστοποιήσει / ελαχιστοποιήσει) μια πολύπλοκη μαθηματική συνάρτηση. Τα ΤΝΔ μπορούμε να τα συναντήσουμε σε πάρα πολλές εφαρμογές στις μέρες μας, επειδή αποτελούν ένα συνεχώς αναπτυσσόμενο κλάδο της τεχνητής νοημοσύνης.

Η υπεροχή των ΤΝΔ έναντι άλλων υπολογιστικών μεθόδων οφείλεται σε πολλούς λόγους και κυρίως στο γεγονός ότι τα ΤΝΔ είναι εμπνευσμένα από τον τρόπο λειτουργίας του ανθρώπινου εγκεφάλου, σε αντίθεση με τους υπολογιστικούς κανόνες, όπως είναι τα κύρια χαρακτηριστικά που συνθέτουν τον τρόπο επεξεργασίας δεδομένων με βάση έναν υπολογιστή. Ο ανθρώπινος εγκέφαλος μπορεί να διαχειρίζεται παράλληλα, πολύ γρήγορα και με πολύ εξειδικευμένο τρόπο τα διάφορα προβλήματα που αντιμετωπίζει, τα οποία μπορεί να είναι μεγάλου μεγέθους, ασαφή και πολύπλοκα. Επίσης έχει την ιδιότητα να μαθαίνει, να προσαρμόζεται σε οποιοδήποτε περιβάλλον και μέσα από αυτά να αναδύει *γνώση*. Όπως βλέπουμε, ο ανθρώπινος εγκέφαλος αποτελεί την πιο πολύπλοκη μηχανή επεξεργασίας του

κόσμου μας, και είναι τόσο παράξενης φύσεως που ακόμη δεν έχουμε καταλάβει πλήρως τον τρόπο λειτουργίας της. Τα ΤΝΔ, είναι συστήματα που προσομοιώνουν την συμπεριφορά του εγκεφάλου και χαρακτηρίζονται από ανθεκτικότητα σε διάφορα λάθη, ευρωστία, μπορούν να προσαρμοστούν κατάλληλα σε διάφορα περιβάλλοντα και να μάθουν δεδομένα, τα οποία μπορεί να μην είναι επαρκώς «καθαρά», αλλά συγκεχυμένα. Αφότου μάθουν κάποια δεδομένα, έχουν την δυνατότητα να γενικεύουν την γνώση αυτή σε διάφορα προβλήματα.

Ο ανθρώπινος εγκέφαλος αποτελείται από χιλιάδες νευρώνες, οι οποίοι είναι και τα βασικά κύτταρα του εγκεφάλου. Ένας νευρώνας αποτελείται από τους δενδρίτες, το σώμα και τις διάφορες νευρικές απολήξεις. Ο νευρώνας αυτός επικοινωνεί με άλλους νευρώνες με την χρήση των συναπτικών, νευρικών του απολήξεων: Αφού υπάρξει κάποιο ερέθισμα, πολλά συναπτικά δυναμικά σήματα, (διεγερτικά ή ανασταλτικά), φτάνουν στους δενδρίτες των νευρώνων. Ο 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 πρέπει να μεταδοθεί σε άλλους νευρώνες. Ο τρόπος με τον οποίο γίνεται αυτό προϋποθέτει την έννοια των «συνάψεων», οι οποίες καθορίζουν την σύνδεση των δενδριτών του νευρώνα του οποίος δέχεται τις πληροφορίες με τις νευρικές απολήξεις του νευρώνα που στέλνει τις πληροφορίες. Οι συνάψεις αποτελούν το πιο σημαντικό σημείο της επικοινωνίας των νευρώνων, γιατί καθορίζουν την δύναμη κατά την οποία μια πληροφορία μπορεί ή δεν μπορεί να περάσει στον επόμενο νευρώνα. 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.

Μάθηση: Η ανώτερη βασική λειτουργία του ανθρώπινου εγκεφάλου. Η μάθηση σε συνδυασμό με την μνήμη, μας διαμορφώνει σαν μοναδικά άτομα, μας βοηθά στο να επιβιώνουμε και να δημιουργούμε μια συνεκτική εικόνα της ζωής μας. Πως όμως μαθαίνει ο εγκέφαλος και συνεπώς πως μαθαίνουν τα συστήματα που τον προσομοιώνουν; Κατ αρχήν υπάρχουν τρία είδη μάθησης, η επιβλεπόμενη μάθηση, η μη επιβλεπόμενη και η ενισχυτική μάθηση.

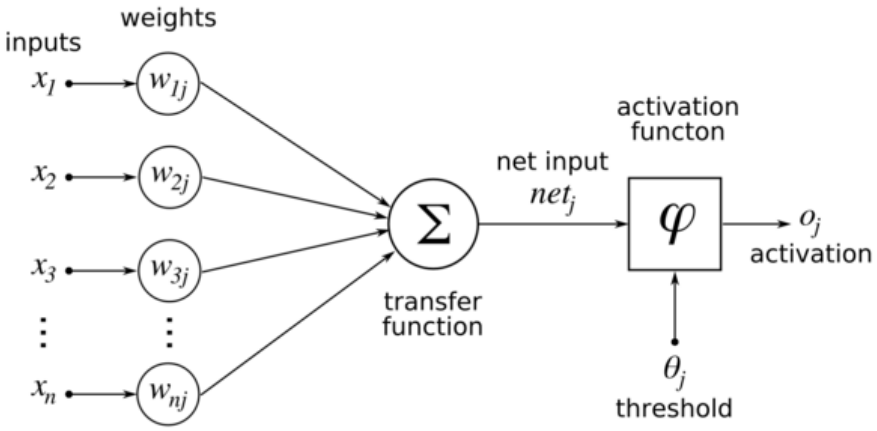
Επιβλεπόμενη μάθηση. Η επιβλεπόμενη μάθηση, στον άνθρωπο, λειτουργεί υπό την επίβλεψη κάποιου *δασκάλου*. Ο μαθητής προσπαθεί να μάθει κάτι από ένα σύνολο από δεδομένα, και ο δάσκαλος είναι εκεί, ώστε σε κάθε λάθος που κάνει, να τον διορθώνει λέγοντας ποια είναι η σωστή. Παράλληλα, ρωτάει τον μαθητή με ερωτήσεις τις οποίες δεν έμαθε άμεσα, αλλά μπορεί να τις προσδιορίσει μέσω αυτών που έχει μάθει ως τώρα. Αν τα αποτελέσματα είναι ικανοποιητικά, τότε σταματά η εκπαίδευση του μαθητή, εάν όχι, τότε ξαναρχινά η ίδια διαδικασία. Αυτό θα γίνεται ώσπου ο μαθητής να μπορεί να δώσει «ικανοποιητικές» απαντήσεις. Αυτήν την διαδικασία ακολουθούν και τα ΤΝΔ. Υπάρχει ένα σύνολο δεδομένων εκπαίδευσης που αποτελείται από δεδομένα επί του θέματος, στα οποία δίνουμε την ερώτηση, αλλά και την απάντηση και από το οποίο το σύστημα θα εκπαιδευτεί. Δίνεται επίσης ένα σύνολο επαλήθευσης, το οποίο αποτελείται από ερωτήσεις 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 υπάρχει δάσκαλος ούτως ώστε σε κάθε λάθος να προσδιορίζει το επιθυμητό αποτέλεσμα, αλλά υπάρχει ένας κριτής ο οποίος δεν προσδιορίζει επακριβώς την σωστή απάντηση αλλά κρίνει τις πράξεις που κάνει ο μαθητής ή το νευρωνικό δίκτυο, δίνει δηλαδή μια επιβράβευση ή τιμωρία για την συγκεκριμένη πράξη. Ο μαθητής, ή το νευρωνικό δίκτυο, προσπαθώντας να αυξήσει την πιθανότητα να παίρνει επιβράβευση και να μειώνει την πιθανότητα να παίρνει τιμωρία, μπορεί να μάθει να κάνει ή να μην κάνει κάτι.

Τέλος, η μη επιβλεπόμενη μάθηση δεν συμπεριλαμβάνει ούτε τα επιθυμητά αποτελέσματα (δάσκαλος), ούτε τον κριτή. Δίνεται στο δίκτυο ένα σύνολο από

δεδομένα, και το δίκτυο αναγνωρίζει με βάση τα κοινά χαρακτηριστικά τους διάφορα υποσύνολα, και τα κατηγοριοποιεί έτσι σε διάφορες ομάδες.

Μια αρχική μοντελοποίηση ενός βιολογικού νευρώνα, η οποία προσομοιώνει τον τρόπο λειτουργίας του με τον τρόπο που εξηγήθηκε στην προηγούμενη παράγραφο, ώστε να δημιουργούν δίκτυα νευρώνων, τα οποία είναι ικανά να μάθουν και να γενικεύσουν πληροφορίες παρουσιάστηκε με την δουλειά των 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, αντιστοιχείται 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 φαίνεται και από το σχήμα 1.4 (transfer function).*

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 κάποιας συνάρτησης όπως φαίνεται από την εξίσωση 1.2. Η συνάρτηση αυτή ελέγχει κατά πόσο η τιμή του νευρώνα ξεπερνά το κατώφλι. Εάν ξεπερνά το κατώφλι πυροβολεί, αλλιώς δεν πυροβολεί.

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

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

Η πιο πάνω συνάρτηση είναι η συνάρτηση κατωφλίου, που ρυθμίζει την έξοδο ενός νευρώνα ανάλογα με ένα κατώφλι θ. Επειδή η συνάρτηση κατωφλίου έχει σοβαρά προβλήματα χρήσης, χρησιμοποιούμε συνήθως την σιγμοειδή συνάρτηση, ή κάποια άλλη παραγωγίσιμη συνάρτηση για ενεργοποίηση του νευρώνα (εξίσωση 1.3).

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

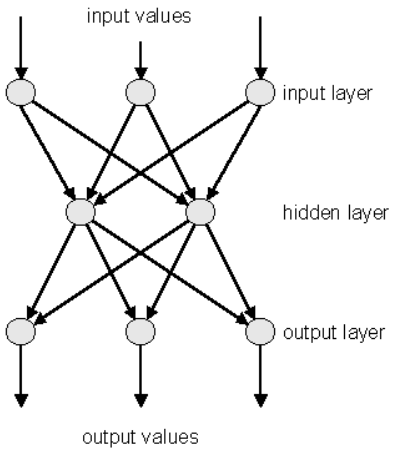
Εξίσωση 1.3: *Το αποτέλεσμα εξόδου ενός νευρώνα, όπου φ η παραγωγίσιμη συνάρτηση ενεργοποίησης.*

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)***.

1.3.1 Πολυστρωματικά δίκτυα perceptron εμπρόσθιου περάσματος

Γενικά, ένα πολυστρωματικό ΤΝΔ, μοιάζει με ένα κατευθυνόμενο γράφο του οποίου οι κόμβοι αποτελούν τις υπολογιστικές μονάδες (νευρώνες), και οι ακμές του συμβολίζουν τα βάρη των συνάψεων μεταξύ των νευρώνων. Λέγεται και δίκτυο εμπρόσθιου περάσματος επειδή σε κάθε επίπεδο δεν υπάρχουν ακμές προς τα προηγούμενα επίπεδα νευρώνων, παρά μόνο προς τα επόμενα (σχήμα 1.5).



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 δυο «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. Φυσικά, για κάθε πρόβλημα στο οποίο χρησιμοποιούνται ΤΝΔ, πρέπει να γίνει προσεκτική επιλογή των εισόδων ώστε να χρησιμοποιούν όλο το φάσμα δεδομένων που προβλήματος, αλλά χωρίς πλεονασμό, ώστε να μην αυξάνεται η πολυπλοκότητα του συστήματος. 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 νευρώνα

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.

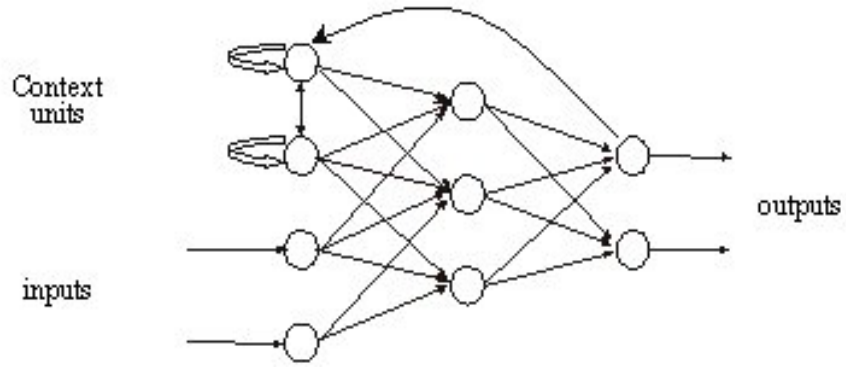
Σχήμα 1.6: *Αλγόριθμος μάθησης perceptron, όπου* x_i *η είσοδος στον νευρώνα,* w_i *το βάρος που αντιστοιχεί στην είσοδο* x_i.

1.3.2 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 προκαλούν κατευθυνόμενους κύκλους. Τα δίκτυα με ανάδραση είναι ένα είδος νευρωνικών δικτύων ειδικά για *δυναμικές* εφαρμογές.

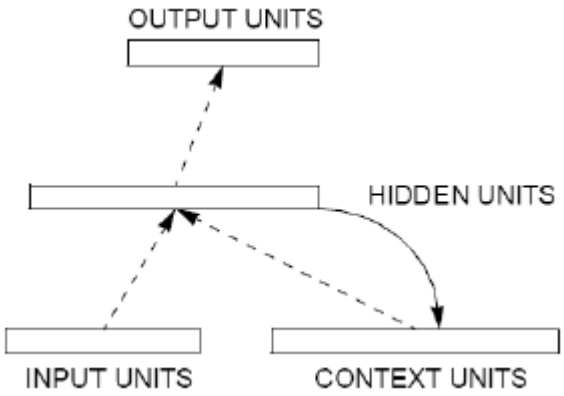
Στα δίκτυα με ανάδραση, γίνεται μοντελοποίηση της αίσθησης του *χρόνου*. Η αρχιτεκτονική τους είναι τέτοια ώστε η είσοδος του δικτύου σε κάποια χρονική στιγμή, να αποτελείται από τα χαρακτηριστικά δεδομένα εισόδου, και από τις εξόδους κάποιων νευρώνων μιας προηγούμενης χρονικής στιγμής. Άρα το δίκτυο 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).*

1.3.3 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 tpj 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.*

1.4 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 το διάστημα αναζήτησης είναι μεγάλο ή σύνθετο, 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 μπορεί να γίνει με πολλούς τρόπους, χρησιμοποιείται όμως ευρέως η 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.

Τώρα είμαστε έτοιμοι να δούμε πως λειτουργούν οι ΓΑ. Μια συνοπτική περιγραφή των λειτουργιών τους φαίνεται στο σχήμα 1.8.

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 1.8: *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 σειρά η διαδικασία της μετάλλαξης, όπου κάποια στοιχεία των χρωμοσωμάτων αλλάζουν ώστε ο αλγόριθμος να αποκλείσει κανένα 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 1.1.

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

Πίνακας 1.1: *Παράμετροι ενός τυπικού Γενετικού Αλγόριθμου.*

Κεφάλαιο 2

Προηγούμενη Εργασία

2.1 Αρχιτεκτονική BRNN

2.2 Αρχικοποίηση BRNN

2.3 Υλοποίηση BRNN

2.1 Αρχιτεκτονική BRNN

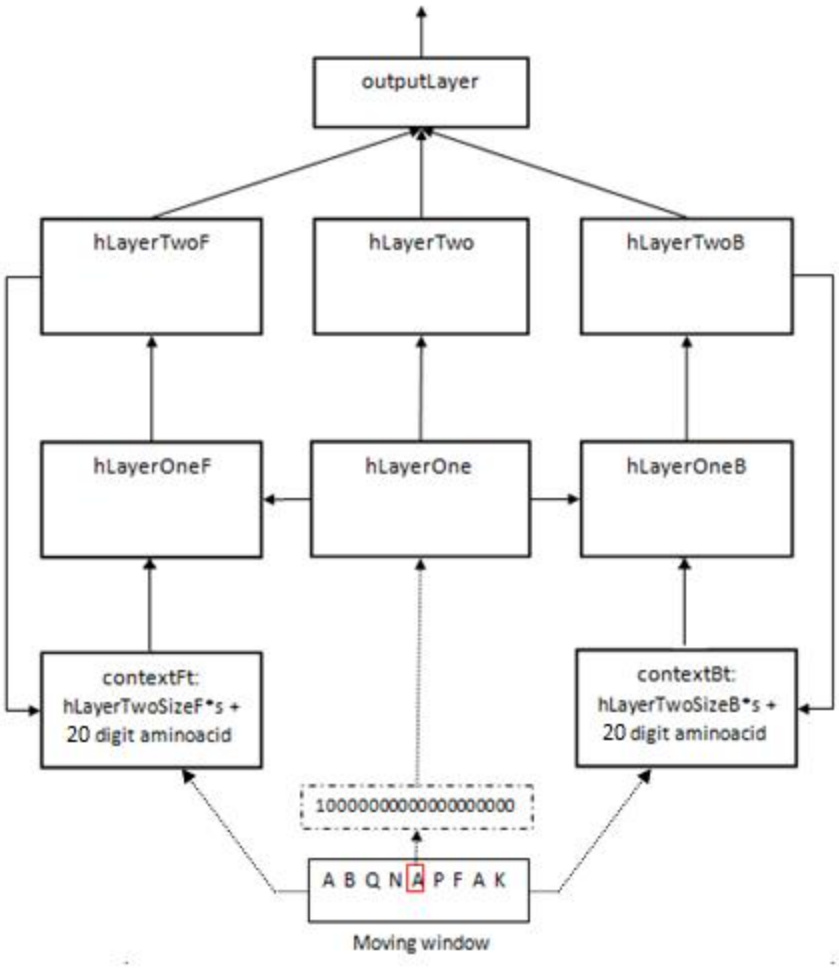
Αρχικά πρέπει να μελετήσουμε την προηγούμενη εργασίας του Μιχάλη Αγαθοκλέους (Αγαθοκλέους 2009) η οποία ήταν η υλοποίηση ενός BRNN, και είναι το σύστημα το οποίο καλούμαστε να επεξεργαστούμε και να επεκτείνουμε στην παρούσα εργασία. Σαν αρχική ιδέα, για το σύστημα χρησιμοποιήθηκε η αρχιτεκτονική που 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.

Αρχικά, θα μελετήσουμε την αρχιτεκτονική του δικτύου που υλοποιήθηκε ως μέρος της προηγούμενης διπλωματικής εργασίας του Αγαθοκλέους (Αγαθοκλέους 2009) (Παράρτημα Α), εντοπίζοντας τα κυριότερα χαρακτηριστικά της, μέσα από την υλοποίηση του.

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 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.



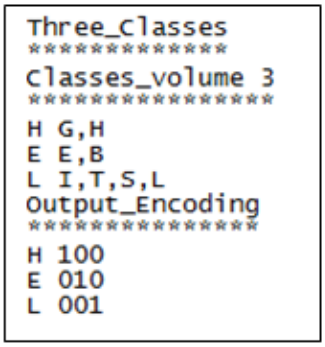
Σχήμα 2.1*: Το τρέχον σύστημα, το οποίο υλοποιήθηκε από τον Αγαθοκλέους (Αγαθοκλέους, 2009).*

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).

2.2 Αρχικοποίηση BRNN

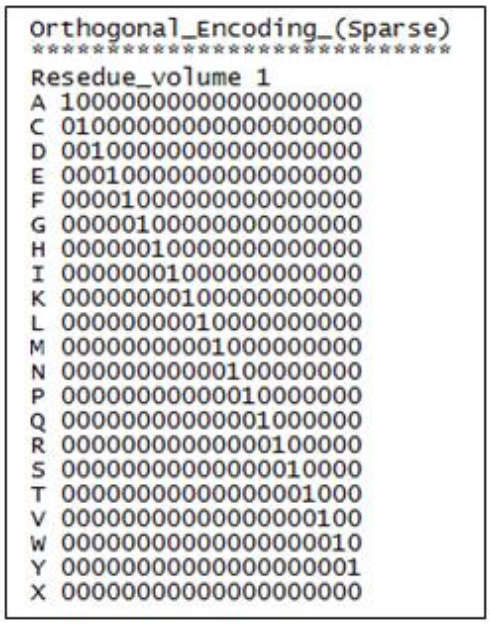
Προγραμματιστικά, το πιο πάνω σύστημα, το οποίο έχει υλοποιήσει ο Αγαθοκλέους, αποτελείται από πολλές διαφορετικές κλάσεις ώστε το πρόγραμμα να έχει μια συνοχή και να είναι ευκολοδιάβαστο. Η παρούσα διπλωματική προϋποθέτει την κατανόηση του υπάρχοντος αυτού προγράμματος, ούτως ώστε αλλαγές αλλά και πειράματα που θα εφαρμοστούν σε αυτό να γίνουν πιο εύκολα και γρήγορα. Μερικές από αυτές τις κλάσεις που συνθέτουν το πρόγραμμα θα αναλυθούν στο υποκεφάλαιο 2.3 ώστε να δώσουμε τα βασικά στοιχεία της κάθε μιας. Θα αρχίσουμε την ανάλυση του τρέχοντος δικτύου από τις γενικές και αφηρημένες κλάσεις για να οδηγηθούμε σιγά σιγά στις βασικότερες λεπτομέρειες που αφορούν το συγκεκριμένο σύστημα. Πριν όμως να γίνει αναφορά στην υλοποίηση του συστήματος, πρέπει να μελετήσουμε πως γίνεται η αρχικοποίηση του, ώστε να διευκολυνθεί αργότερα η μελέτη της υλοποίησης του.

Το σύστημα αρχικοποιείται χρησιμοποιώντας κάποια συγκεκριμένα αρχεία, για τις μετέπειτα λειτουργίες του. Αυτά τα αρχεία έχουν να κάνουν κυρίως με την κωδικοποίηση και αποκωδικοποίηση των δεδομένων, καθώς και με τις παραμέτρους εισόδου. Το αρχείο *threeClasses.txt* (σχήμα 2.2) περιέχει την κωδικοποίηση εξόδου. 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. Αφού το *outputLayer* χρησιμοποιεί τρεις νευρώνες για να κωδικοποιήσει το αποτέλεσμα πρόβλεψης του κάθε αμινοξέως, έχουμε συνεπώς οχτώ συνδυασμούς τιμών, από τις οποίες οι δυο κωδικοποιούν τα H και E αντίστοιχα, και οι υπόλοιπες έξι κωδικοποιούν οτιδήποτε άλλο είδος δευτεροταγούς δομής (Loops). Φυσικά, δίνεται και η κωδικοποίηση εξόδου ενός πραγματικού Loop αμινοξέως. Όπως φαίνεται και από το σχήμα 2.2 η κωδικοποίηση των τριών αποτελεσμάτων γίνεται με το λεγόμενο *orthogonal encoding*, προσδιορίζοντας για κάθε νευρώνα εξόδου, μια συγκεκριμένη δευτεροταγή δομή την οποία να κωδικοποιεί. Παρόλα αυτά, φαίνεται από το αρχείο αυτό, και η ανάλογη συσχέτιση των ομάδων δευτεροταγούς δομής G, H, E, B, I, T, S, L σε μόνο τρεις ομάδες, H, E, L μέσω του προγράμματος τυποποίησης DSSP (Αγαθοκλέους, 2009).



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

Με ανάλογο τρόπο κωδικοποιούνται τα 20 αμινοξέα έτσι ώστε να αντιπροσωπεύουν ένα γράμμα, καθώς και το αμινοξύ Χ το οποίο συμβολίζει τα «αμινοξέα» που υπάρχουν πριν το πρώτο αμινοξύ και μετά το τελευταίο αμινοξύ μιας πρωτεΐνης. Η κωδικοποίηση των *ορίων* της πρωτεΐνης με το υποτιθέμενο αμινοξύ *Χ* το οποίο μας βοηθά να υπολογίσουμε την δευτεροταγή δομή του κεντρικού αμινοξέως του οποίου το κινούμενο παράθυρο αμινοξέων προς επεξεργασία είναι εκτός ορίων. Το αρχείο αυτό λέγεται «sparse.txt» και φαίνεται στο σχήμα 2.3. Το αρχείο αυτό περιέχει εκτός από την κωδικοποίηση των αμινοξέων, και άλλες παραμέτρους εισόδου που αφορούν το σύστημα. Αυτό είναι το residue volume, το οποίο χαρακτηρίζει το πόσα αμινοξέα κωδικοποιούνται, που στην προκειμένη περίπτωση είναι μια κωδικοποίηση για κάθε αμινοξύ.



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

Τέλος, υπάρχει το αρχείο «parameters.dat» το οποίο περιέχει τις παραμέτρους που αφορούν την αρχιτεκτονική του δικτύου και τις οποίες μπορεί να αλλάξει ο χρήστης οποιαδήποτε στιγμή (σχήμα 2.4). Η περιγραφή των παραμέτρων αυτών δίνεται από την προηγούμενη διπλωματική εργασία (Αγαθοκλέους, 2009) και φαίνεται στον πίνακα 2.1.

```
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 (Παράρτημα Η).*

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

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.

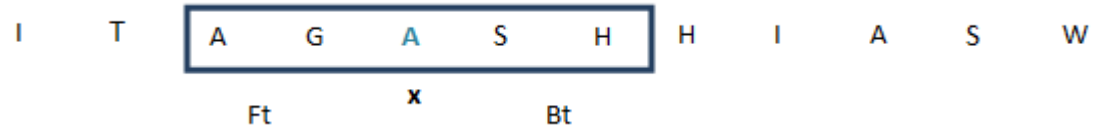
Ο κύριος βρόγχος εκτελείται ανάλογα με τις εποχές που διαλέγει ο χρήστης, και για κάθε εποχή εκτελούνται δυο εσωτερικοί βρόγχοι, ένας ο οποίος εκπαιδεύει το δίκτυο χρησιμοποιώντας το σύνολο πρωτεϊνών εκπαίδευσης, και ο δεύτερος είναι ένας βρόγχος ο οποίος απλώς επαληθεύει το τρέχον δίκτυο χρησιμοποιώντας το σύνολο πρωτεϊνών επαλήθευσης. Η γενική δομή της κλάσης αυτής φαίνεται στο σχήμα 2.5.

```
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*: Η γενική λειτουργία της κλάσης pssp_ucy.cpp κλάσης.*

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

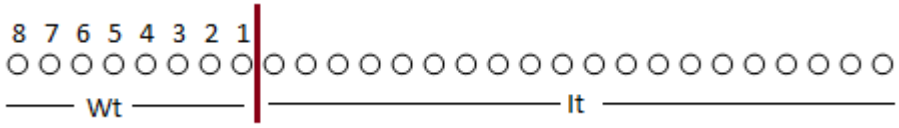
των καταλοίπων που περιέχονται στο κινούμενο παράθυρο, στο δίκτυο για επεξεργασία. Στην συνέχεια υπολογίζεται η έξοδος του δικτύου για το συγκεκριμένο κεντρικό αμινοξύ. Τέλος ακολουθεί η συνάρτηση *doBackPropagation()* για το κεντρικό αμινοξύ. Η συνάρτηση αυτή υπολογίζει το σφάλμα του δικτύου για το 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 διαχωρίζει την επεξεργασία των αμινοξέων ενός κινητού παραθύρου ανάλογα με το ποια έπονται και ποια προηγούνται του κεντρικού αμινοξέως. Για παράδειγμα στο σχήμα 2.6, 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. 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.



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, δεδομένου ότι οι νευρώνες εξόδου είναι τέσσερεις, και s ίσο με δυο.*

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. Άρα και στο σχήμα 2.7, το Wt δεν είναι επακριβώς οι τιμές των νευρώνων εξόδου των προηγούμενων αμινοξέων, αλλά είναι πολλαπλασιασμένες με μια σταθερά. Τέλος, στην επεξεργασία του κεντρικού αμινοξέως, το δίκτυο χρησιμοποιεί μια *step function* (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».*

Κεφάλαιο 3

Επεξεργασία Δεδομένων

3.1 Χρήση MSA profiles

3.2 HSSP database

3.3 Επεξεργασία δεδομένων εισόδου για το σύστημα

3.3.1 Δημιουργία parsers για διαχείριση αρχείων

3.4 Επεξεργασία δεδομένων εξόδου του συστήματος

3.4.1 Confusion Matrices

3.4.2 Segment OVerlap (SOV)

3.1 Χρήση MSA profiles

Θέλουμε να τροποποιήσουμε το δίκτυο ώστε να εκπαιδεύεται από πρωτεΐνες οι οποίες ευθυγραμμίστηκαν 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 η κωδικοποίηση τους. Η μέχρι τώρα κωδικοποίηση των αμινοξέων ήταν σταθερή για κάθε αμινοξύ και λέγεται *orthogonal encoding* (σχήμα 3.1). Θα προστεθεί 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.

```
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
```

Σχήμα 3.1: *Η μέχρι τώρα κωδικοποίηση του κάθε αμινοξέως ήταν μια συμβολοσειρά 20 ψηφίων, τοποθετώντας 1 στην θέση που αναλογεί το αμινοξύ στο αγγλικό αλφάβητο και αφήνοντας τις υπόλοιπες 0.*

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, όπου παρουσιάζεται η πιθανότητα εμφάνισης κάθε αμινοξέως. Ένα πρότυπο της νέας κωδικοποίησης φαίνεται στο σχήμα 3.2.

	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 3.2*: Έστω ότι έχουμε την πρωτεΐνη xxxx. 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. Έχουμε 10% πιθανότητα να εμφανιστεί S στην πρώτη θέση, 16% να εμφανιστεί K, 16% να εμφανιστεί E, 40% να εμφανιστεί N και 10% να εμφανιστεί D. Άρα ολόκληρη η δεύτερη γραμμή κωδικοποιεί την θέση του πρώτου αμινοξέως. Το ίδιο γίνεται και για τις υπόλοιπες θέσεις των αμινοξέων.*

3.2 HSSP database

Στα προηγούμενα πειράματα του δικτύου χωρίς την χρήση MSA profiles το σύνολο πρωτεϊνών που χρησιμοποιήθηκε για εκπαίδευση και επαλήθευση ήταν το σύνολο πρωτεϊνών του Hobohm. Με την χρήση MSA profiles για εκπαίδευση του δικτύου, και πάλι θα χρησιμοποιήσουμε τα MSA profiles των πρωτεϊνών του αρχείου Hobohm. Για να πάρουμε όμως τις πληροφορίες αυτές θα χρησιμοποιήσουμε την βάση HSSP (Homology-derived Secondary Structure of Proteins) η οποία δίνει έτοιμα και την στοίχιση των ομόλογων πρωτεϊνών και το αντίστοιχο MSA profile τους όπως θα δούμε πιο κάτω.

Στην βάση αυτή, για κάθε πρωτεΐνη γνωστής τριτοταγούς δομής που μπορούμε να πάρουμε από την PDB (RCSB Protein Data Bank), η HSSP περιέχει μια στοίχιση της συγκεκριμένης πρωτεΐνης με όλες τις διαθέσιμες προς αυτήν ομόλογες πρωτεΐνες. Ομόλογες πρωτεΐνες, όπως έχουμε προαναφέρει λέγονται οι πρωτεΐνες οι οποίες θεωρούμε ότι έχουν προέλευση από ένα κοινό προγονικό μόριο, έτσι, διατηρούν γενικώς την ίδια τριτοταγή δομή. Η επιλογή των ομόλογων αυτών πρωτεϊνών έγινε μέσω της βάσης Swissprot χρησιμοποιώντας μια μέθοδο δυναμικού προγραμματισμού για στοίχιση ακολουθιών. Ανάλογα με το σκορ της στοίχισης, η ελεγχόμενη πρωτεΐνη ήταν ή όχι ομόλογη με την πρωτεΐνη που εξεταζόταν και ανάλογα επιλεγόταν ή όχι για την επόμενη διαδικασία. Η HSSP εκτός από την στοίχιση των ακολουθιών αυτών με την συγκεκριμένη πρωτεΐνη που εξετάζουμε, παρουσιάζει σαν επόμενο βήμα και ένα προφίλ, το οποίο δείχνει τα χαρακτηριστικά της οικογένειας αυτής των πρωτεϊνών με ένα τρόπο όπως την προηγούμενη εικόνα, που είναι και αυτό το οποίο θα χρειαστούμε για την νέα κωδικοποίηση (Schneider and Sander, 1996).

Κατ’ αρχήν, από την βάση αυτή θέλουμε να πάρουμε μόνο τις πληροφορίες που αφορούν πρωτεΐνες που συμπεριλαμβάνονται στο αρχείο του Hobohm. Γι αυτό μετέπειτα θα κατασκευαστούν ειδικοί parsers οι οποίοι θα παίρνουν μόνο τις συγκεκριμένες πρωτεΐνες που χρειαζόμαστε για το δίκτυο. Ένα δεύτερο σημείο που πρέπει να αναφέρουμε, είναι ότι τα αρχεία της βάσης αυτής έχουν αρκετή

πληροφορία, και επειδή δεν χρειαζόμαστε όλη αυτήν την πληροφορία οι parsers θα επιλέγουν μόνο τα κομμάτια τα οποία χρειάζονται για περαιτέρω επεξεργασία.

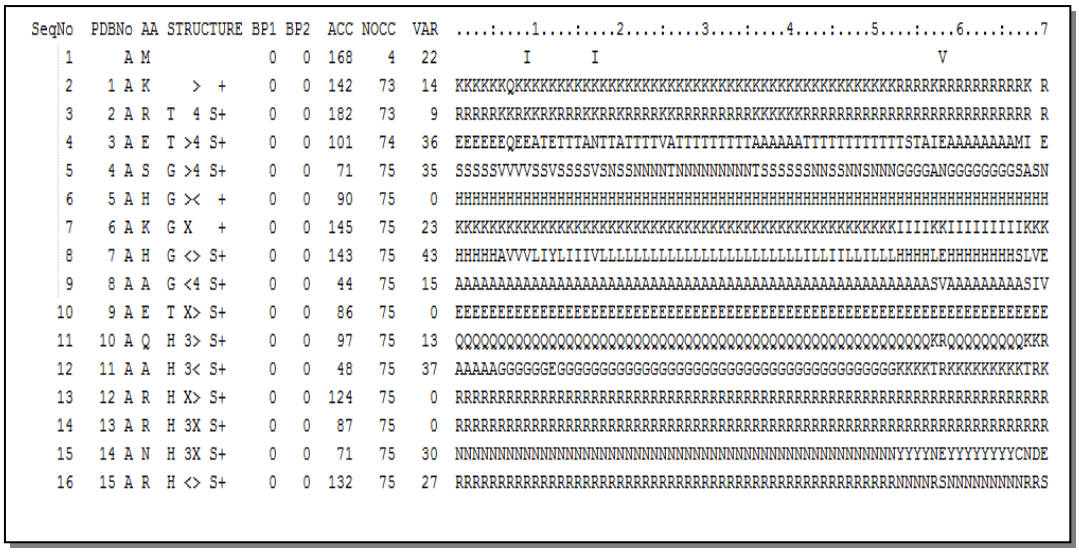
Ένα τυπικό αρχείο μιας πρωτεΐνης στην βάση HSSP, ονομάζεται *xxxx.hssp* όπου xxxx είναι το όνομα της πρωτεΐνης (βάση της PDB). Το τυπικό αρχείο χωρίζεται σε 4 μέρη. Από το πρώτο μέρος του αρχείου, το λεγόμενο HEADERS block μπορούμε να πάρουμε πληροφορίες σχετικά με την πρωτεΐνη, τον αριθμό των αμινοξέων της, τον αριθμό των ακολουθιών που ευθυγραμμίστηκαν με αυτήν, το μήκος της πρωτεΐνης αυτής και κάποιες παραμέτρους σχετικά με την ευθυγράμμιση τους (σχήμα 3.3). Το δεύτερο μέρος του αρχείου λέγεται PROTEINS block και περιέχει πληροφορίες σχετικά με την κατά ζεύγη στοίχιση των πρωτεϊνών που θεωρούνται ομόλογες με την πρωτεΐνη που εξετάζουμε (σχήμα 3.4). Το τρίτο μέρος του αρχείου λέγεται ALINGMENT block και περιέχει πληροφορίες σχετικά με την πολλαπλή ευθυγράμμιση των πρωτεϊνών μεταξύ τους. Είναι ένα σημαντικό μέρος του αρχείου, αφού βάση αυτής της ευθυγράμμισης μπορούμε εύκολα να υπολογίσουμε το προφίλ της οικογένειας αυτής των πρωτεϊνών (σχήμα 3.5). Επίσης υπάρχουν σημαντικές πληροφορίες όπως είναι η πρωτοταγής δομή και η δευτεροταγής δομή της ακολουθίας της πρωτεΐνης που εξετάζουμε. Το τέταρτο και τελευταίο μέρος ενός αρχείου είναι για μας το πιο σημαντικό μέρος του αρχείου αυτού. Ονομάζεται SEQUENCE PROFILE block και είναι το προφίλ της οικογένειας των πρωτεϊνών που θέλουμε να δώσουμε σαν κωδικοποίηση στο πρόγραμμα (σχήμα 3.6) (Schneider and Sander, 1996).

```
HSSP        HOMOLOGY DERIVED SECONDARY STRUCTURE OF PROTEINS , VERSION 1.1 2001
PDBID       1a0a
DATE        file generated on 23-Sep-09
SEQBASE     UniProtKB/Swiss-Prot(22-Sep-2009) UniProtKB/TrEMBL(22-Sep-2009)
PARAMETER   SMIN: -0.5  SMAX:  1.0
PARAMETER   gap-open:  3.0 gap-elongation:  0.1
PARAMETER   conservation weights: YES
PARAMETER   InDels in secondary structure allowed: YES
PARAMETER   alignments sorted according to :DISTANCE
THRESHOLD   according to: t(L)=(290.15 * L ** -0.562) +  5
REFERENCE   Sander C., Schneider R. : Database of homology-derived protein structures.
CONTACT     Maintained at www.CMBI.ru.nl by Elmar.Krieger@cmbi.ru.nl
AVAILABLE   Free academic use. Commercial users must apply for license.
AVAILABLE   No inclusion in other databanks without permission.
HEADER      TRANSCRIPTION/DNA
COMPND       MOLECULE: DNA (5'-
SOURCE       SYNTHETIC: YES;
AUTHOR      T.SHIMIZU,A.TOUMOTO,K.IHARA,M.SHIMIZU,Y.KYOGOKU,N.OGAWA,
SEQLENGTH     63
NCHAIN         2 chain(s) in 1a0a data set
KCHAIN         1 chain(s) used here ; chain(s) : A
NALIGN        76
```

Σχήμα 3.3: *Το πρώτο μέρος ενός αρχείου της HSSP βάσης συμπεριλαμβάνει το όνομα της πρωτεΐνης όπως το παίρνουμε από την PDB (PDBID), το μέγεθος της πρωτεΐνης αυτής (SEQLENGTH), τον αριθμό των ακολουθιών που ευθυγραμμίστηκαν με αυτήν (NALIGN) κ.α.*

NR.	ID	STRID	%IDE	%WSIM	IFIR	ILAS	JFIR	JLAS	LALI	NGAP	LGAP	LSEQ2	ACCNUM	PROTEIN
1 :	Q06859_YEAST		1.00	1.00	2	63	251	312	62	0	0	312	Q06859	SubName: Full=Yeast PHO4 gene mutated by
2 :	Q06860_YEAST		1.00	1.00	2	63	251	312	62	0	0	312	Q06860	SubName: Full=Yeast PHO4 gene mutated by
3 :	PHO4_YEAST	1A0A	0.98	0.99	2	63	251	312	62	0	0	312	P07270	RecName: Full=Phosphate system positive r
4 :	A7A274_YEAS7		0.98	0.99	2	63	251	312	62	0	0	312	A7A274	SubName: Full=Myc-family transcription fa
5 :	B3LUP5_YEAS1		0.98	0.99	2	63	251	312	62	0	0	312	B3LUP5	SubName: Full=Myc-family transcription fa
6 :	C5DHI8_LACTC		0.60	0.66	2	63	404	463	60	1	2	466	C5DHI8	SubName: Full=KLTH0E04664p;
7 :	Q6CQL9_KLULA		0.59	0.65	2	62	629	687	59	1	2	689	Q6CQL9	SubName: Full=KLLA0D16115p;
8 :	C5DWL2_ZYGRC		0.57	0.65	2	63	422	481	60	1	2	483	C5DWL2	SubName: Full=ZYRO0D15730p;
9 :	Q75AP3_ASHGO		0.57	0.65	1	62	337	396	60	1	2	396	Q75AP3	SubName: Full=ADL123Cp;
10 :	A3GFA3_PICST		0.57	0.66	2	63	486	545	60	1	2	550	A3GFA3	SubName: Full=Myc-family transcription fa
11 :	A5DQ49_PICGU		0.56	0.68	2	60	369	425	57	1	2	432	A5DQ49	SubName: Full=Putative uncharacterized pr
12 :	Q6FVW0_CANGA		0.56	0.65	2	60	471	527	57	1	2	533	Q6FVW0	SubName: Full=Similarities with uniprot|P
13 :	Q6C2U3_YARLI		0.57	0.62	2	60	620	672	53	1	6	716	Q6C2U3	SubName: Full=YALI0F05126p;
14 :	Q2HG73_CHAGB		0.54	0.62	2	60	462	534	59	1	14	560	Q2HG73	SubName: Full=Putative uncharacterized pr
15 :	B2AYV8_PODAN		0.53	0.60	2	63	642	726	62	1	23	744	B2AYV8	SubName: Full=Predicted CDS Pa_1_12400;

Σχήμα 3.4: *Το δεύτερο μέρος ενός αρχείου της HSSP βάσης (PROTEINS block) παρουσιάζει τα στοιχεία που αφορούν τις κατά ζεύγη ευθυγραμμίσεις.*



SeqNo	PDBNo	AA	STRUCTURE	BP1	BP2	ACC	NOCC	VAR	....:....1....:....2....:....3....:....4....:....5....:....6....:....7
1	A	M		0	0	168	4	22	I I V
2	1 A	K	> +	0	0	142	73	14	KKKKKKQKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKRRRRKRRRRRRRRRK R
3	2 A	R	T 4 S+	0	0	182	73	9	RRRRRKKRKKRKRRRKKRRKRRRRKRRRRRRRRRKKKKKRRRRRRRRRRRRRRRRRRRRRRRRRRR R
4	3 A	E	T >4 S+	0	0	101	74	36	EEEEEEQEEATETTTANTTATTTTVATTTTTTTTTTAAAAAATTTTTTTTTTTTSTAIEAAAAAAAAMI E
5	4 A	S	G >4 S+	0	0	71	75	35	SSSSSVVVVSSVSSSSVSNSSNNNNTNNNNNNNNNTSSSSSSNNSSNNSNNNGGGGANGGGGGGGSASN
6	5 A	H	G >< +	0	0	90	75	0	HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
7	6 A	K	G X +	0	0	145	75	23	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKIIIIKKIIIIIIIIIKKK
8	7 A	H	G <> S+	0	0	143	75	43	HHHHHAVVVLIYLIIIVLLLLLLLLLLLLLLLLLLLLLLLIIIIILLLLLHHHHLEHHHHHHHHSLVE
9	8 A	A	G <4 S+	0	0	44	75	15	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
10	9 A	E	T X> S+	0	0	86	75	0	EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
11	10 A	Q	H 3> S+	0	0	97	75	13	QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQKRQQQQQQQQQKKR
12	11 A	A	H 3< S+	0	0	48	75	37	AAAAAGGGGGGEGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGKKKKTRKKKKKKKKKTRK
13	12 A	R	H X> S+	0	0	124	75	0	RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
14	13 A	R	H 3X S+	0	0	87	75	0	RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
15	14 A	N	H 3X S+	0	0	71	75	30	NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNYYYYNEYYYYYYYYCNDE
16	15 A	R	H <> S+	0	0	132	75	27	RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRNNNRSNNNNNNNNRRS

Σχήμα 3.5: *Το τρίτο μέρος ενός αρχείου HSSP (ALINGMENT block). Στην στήλη SeqNo, παρουσιάζεται ο αριθμός των αμινοξέων της πρωτεΐνης την οποία ευθυγραμμίζουμε. Στην στήλη PDBNo, παρουσιάζονται δύο στοιχεία, ο αριθμός των αμινοξέων όπως παρουσιάζονται στην βάση PDB καθώς και ένα γράμμα που υποδεικνύει την «αλυσίδα» της συγκεκριμένης πρωτεΐνης που ευθυγραμμίζεται. Η στήλη AA δείχνει την πρωτοταγή δομή της ακολουθίας και αντίστοιχα η στήλη STRUCTURE την δευτεροταγή δομή. Σημαντικό είναι να προσέξουμε ότι μπορεί να υπάρχουν ελλιπή δεδομένα στην δευτεροταγή δομή μιας ακολουθίας.*

SeqNo	PDBNo	V	L	I	M	F	W	Y	G	A	P	S	T	C	H	R	K	Q	E	N	D
1	A	25	0	50	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1 A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	71	1	0	0	0
3	2 A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78	22	0	0	0	0
4	3 A	1	0	3	1	0	0	0	0	26	0	3	43	0	0	0	0	1	20	1	0
5	4 A	8	0	0	0	0	0	0	16	3	0	33	3	0	0	0	0	0	0	37	0
6	5 A	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0
7	6 A	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	83	0	0	0	0
8	7 A	7	47	12	0	0	0	1	0	1	0	1	0	0	21	0	0	0	9	0	0
9	8 A	11	0	1	0	0	0	0	0	85	0	3	0	0	0	0	0	0	0	0	0
10	9 A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0
11	10 A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	4	85	0	0	0
12	11 A	0	0	0	0	0	0	0	61	5	0	0	3	0	0	7	23	0	1	0	0
13	12 A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0
14	13 A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0
15	14 A	0	0	0	0	0	0	16	0	0	0	0	0	1	1	0	0	0	9	71	1
16	15 A	0	0	0	0	0	0	0	0	0	0	9	0	0	0	73	0	0	0	17	0
17	16 A	0	16	68	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	17 A	0	0	0	0	0	0	0	0	5	0	0	3	0	0	0	17	0	0	75	0
19	18 A	7	5	3	8	3	0	0	0	0	0	8	21	0	0	0	0	4	0	41	0
20	19 A	0	0	0	0	0	1	0	25	71	0	0	0	3	0	0	0	0	0	0	0
21	20 A	12	53	17	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21 A	0	1	0	0	0	0	0	0	8	0	4	3	0	7	8	32	19	0	1	17
23	22 A	0	0	0	4	0	0	0	0	0	0	0	15	0	0	1	0	3	72	0	5
24	23 A	0	63	35	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	24 A	0	0	0	0	0	0	0	9	24	0	4	1	0	20	0	1	5	29	4	1
26	25 A	0	3	0	0	0	0	0	3	25	0	28	11	0	0	15	5	3	0	3	5
27	26 A	0	93	1	4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Σχήμα 3.6: *Το τέταρτο μέρος του αρχείου HSSP (SEQUENCE profile). Στην πρώτη γραμμή του προφίλ της οικογένειας, υπάρχουν τα 20 διαφορετικά αμινοξέα. Το SeqNo συμβολίζει τον αύξοντα αριθμό του αμινοξέως στην πρωτεΐνη που εξετάζουμε, το PDBNo χαρακτηρίζεται από δύο τιμές, τον αριθμό του αμινοξέως όπως παρουσιάζεται στην βάση PDB και το είδος της αλυσίδας της πρωτεΐνης. Το μήκος ενός προφίλ είναι ίσο με το μήκος της ακολουθίας που εξετάζουμε. Κάθε γραμμή συμβολίζει το ποσοστό (%) της εμφάνισης του κάθε ενός από τα αμινοξέα στην συγκεκριμένη θέση της πολλαπλής στοίχισης.*

3.3 Επεξεργασία δεδομένων εισόδου για το δίκτυο

3.3.1 Δημιουργία parsers για διαχείριση αρχείων

Όπως είδαμε και στο υποκεφάλαιο 2.2, υπάρχει ένα συγκεκριμένο σύνολο πρωτεϊνών για τις οποίες θέλουμε να πάρουμε τα MSA profiles τους. Αυτό έγινε κατορθωτό με την υλοποίηση κάποιων parsers στην γλώσσα προγραμματισμού PERL. Η PERL είναι μια γλώσσα προγραμματισμού, διάσημη για εφαρμογές στον τομέα της Βιοπληροφορικής, εύκολη στην χρήση και χρήσιμη όταν έχουμε να κάνουμε με αρχεία κειμένου και βάσεις βιολογικών δεδομένων.

Ξεκινώντας, πήραμε το αρχείο του *Heinz-Uwe Hobohm*, (pdb_select25 dataset, 20 April 2009), από την προηγούμενη διπλωματική εργασία (Αγαθοκλέους, 2009), το

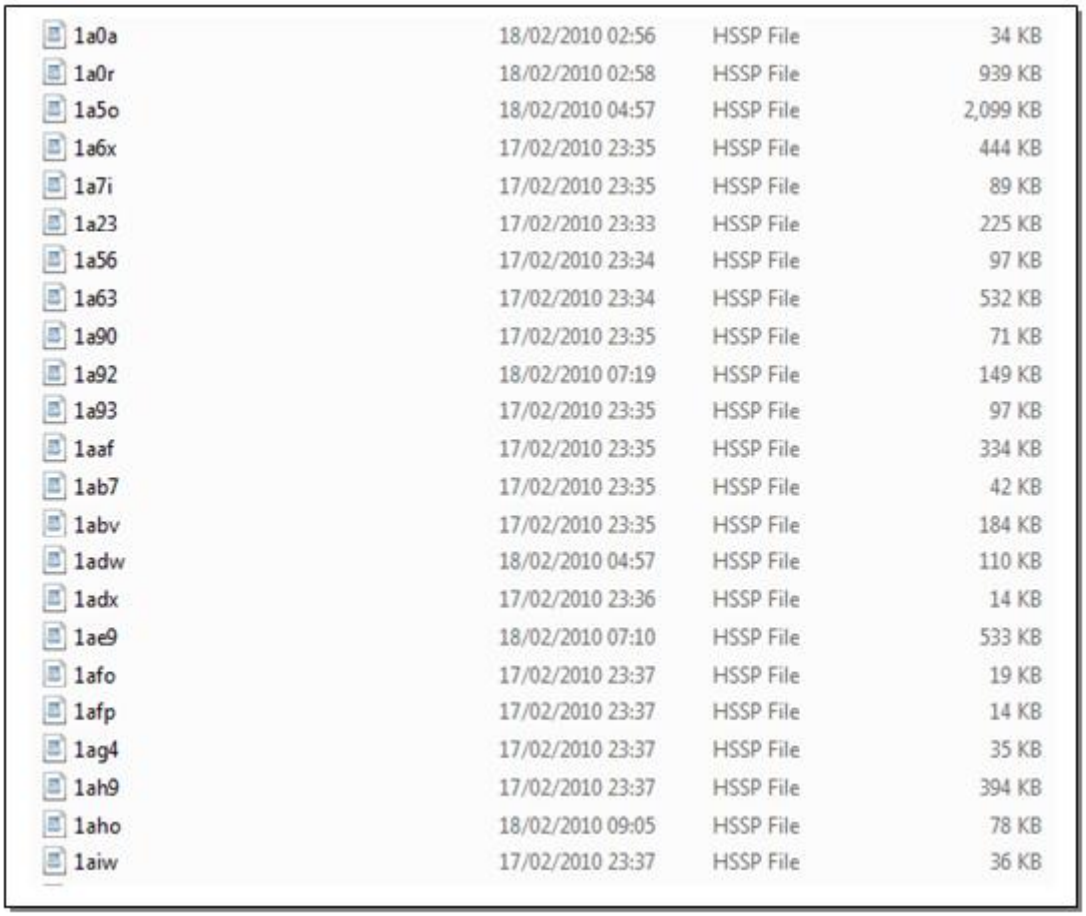
οποίο περιέχει 4018 πρωτεΐνες (σχήμα 3.7). Ο λόγος για τον οποίο επιλέγηκαν οι συγκεκριμένες πρωτεΐνες του συγκεκριμένου αρχείου του Hobohm, περιγράφεται επίσης από την προηγούμενη διπλωματική εργασία (Αγαθοκλέους 2009). Σημαντικό είναι να προσέξουμε την μορφή του ονόματος των πρωτεϊνών αυτών. Όπως περιγράφει και ο Αγαθοκλέους (Αγαθοκλέους, 2009), τα πρώτα τέσσερα γράμματα αφορούν το όνομα της πρωτεΐνης όπως παρουσιάζεται στην PDB και το πέμπτο γράμμα αφορά την συγκεκριμένη πολυπεπτιδική αλυσίδα. Επειδή στο αρχείο αυτό, εκτός από τα ονόματα των πρωτεϊνών, υπάρχουν και άλλες επιπλέον πληροφορίες όπως φαίνεται και στο σχήμα 3.7, πρέπει να κατασκευάσουμε ένα parser ο οποίος να παίρνει από το αρχείο αυτό μόνο τα ονόματα των πρωτεϊνών και να κατασκευάζει ένα νέο αρχείο που αποκλειστικά θα έχει τα ονόματα και μόνο των πρωτεϊνών αυτών. Αυτό είναι πολύ σημαντικό για την συνέχεια επειδή με βάση το νέο αρχείο θα παίρνουμε τα MSA profiles της κάθε πρωτεΐνης. Φυσικά, δεν επιλέγονται όλες οι πρωτεΐνες, αλλά αυτές που πληρούν κάποια κριτήρια (Baldi et al., 1999).

25	1JX5A	98	-1.00	0.00	N	98	98	0	38	10	interleukin enhancer binding factor
25	1X6EA	72	-1.00	0.00	N	72	72	0	24	8	zinc finger protein 24
25	1X6AA	81	-1.00	0.00	N	81	81	0	8	12	lim domain kinase 2
25	1X69A	79	-1.00	0.00	N	79	79	0	0	26	cortactin isoform a
25	1X65A	89	-1.00	0.00	N	89	89	0	4	22	unr protein
25	1X60A	79	-1.00	0.00	N	79	79	0	28	29	sporulation-specific n-acetylmuramoyl-l-alanine amida
25	1X5XA	109	-1.00	0.00	N	109	109	0	0	44	fibronectin type-iii domain containing protein 3a
25	1X5VA	34	-1.00	0.00	N	34	34	1	3	13	pcfk1
25	1X5RA	112	-1.00	0.00	N	112	112	0	16	30	glutamate receptor interacting protein 2
25	1X5PA	97	-1.00	0.00	N	97	97	0	18	27	negative elongation factor e
25	1X5MA	127	-1.00	0.00	N	127	127	0	7	45	calcyclin-binding protein
25	1X5HA	132	-1.00	0.00	N	132	132	0	0	52	neogenin
25	1X58A	62	-1.00	0.00	N	62	62	0	33	0	hypothetical protein 4930532d21rik
25	1X93A	43	-1.00	0.00	N	43	43	0	24	7	hypothetical protein hp0222
25	1X4VA	63	-1.00	0.00	N	63	63	0	4	2	hypothetical protein loc130617
25	1X4SA	59	-1.00	0.00	N	59	59	0	11	10	zinc finger hit domain containing protein 2
25	1X4QA	92	-1.00	0.00	N	92	92	0	56	0	u4/u6 small nuclear ribonucleoprotein prp3
25	1X4MA	94	-1.00	0.00	N	94	94	0	28	18	far upstream element binding protein 1
25	1X4LA	72	-1.00	0.00	N	72	72	0	5	12	skeletal muscle lim-protein 3
25	1X4IA	70	-1.00	0.00	N	70	70	0	15	4	inhibitor of growth protein 3
25	1X4FA	112	-1.00	0.00	N	112	112	0	17	20	matrin 3

Σχήμα 3.7: *Το αρχείο του pdb_select25 του Hobohm. Μας ενδιαφέρει να απομονώσουμε την δεύτερη στήλη η οποία δίνει το όνομα της πρωτεΐνης. Το 5^{ο} γράμμα συμβολίζει την συγκεκριμένη πολυπεπτιδική αλυσίδα της πρωτεΐνης (Αγαθοκλέους 2009).*

Το πρόγραμμα που απομονώνει τα ονόματα των πρωτεϊνών, δηλαδή την δεύτερη στήλη, παίρνει σαν είσοδο ένα αρχείο της μορφής του Hobohm. Στη συνέχεια, το πρόγραμμα διαβάζει γραμμή προς γραμμή τις πρωτεΐνες της δεύτερης στήλης από το αρχείο εισόδου, αποθηκεύει τις πρωτεΐνες αυτές προσωρινά σε ένα πίνακα και στο τέλος τις μεταφέρει στο νέο αρχείο, το αρχείο εξόδου. Το πρόγραμμα αυτό ονομάζεται «perlCreateProteinFromHobohm.pl».

Στην συνέχεια χρησιμοποιήσαμε το «script.sh», πρόγραμμα υλοποιημένο από τον διδακτορικό φοιτητή Αντώνη Αντωνίου, για να πάρουμε τα *.hssp* αρχεία από την βάση HSSP (10 February 2010). Τα αρχεία αυτά αποθηκεύονται σε ένα φάκελο (MSAproduction) για περαιτέρω επεξεργασία. Αυτό που πρέπει να θυμηθούμε, είναι ότι τα αρχεία αυτά, τα οποία έχουν το όνομα της πρωτεΐνης που περιγράφουν, αποτελούνται μόνο από τέσσερα γράμματα και όχι από πέντε όπως είναι οι πρωτεΐνες του Hobohm. Λείπει δηλαδή το πέμπτο γράμμα που συμβολίζει την πολυπεπτιδική αλυσίδα της πρωτεΐνης. Όμως, ένα τυπικό αρχείο μιας πρωτεΐνης από την βάση HSSP, συμπεριλαμβάνει κάποιες πολυπεπτιδικές αλυσίδες που ανήκουν στην πρωτεΐνη αυτή, την πρωτοταγή και την δευτεροταγή δομή τους. Ένα δείγμα από τα αρχεία αυτά, φαίνεται στο σχήμα 3.8.



1a0a	18/02/2010 02:56	HSSP File	34 KB
1a0r	18/02/2010 02:58	HSSP File	939 KB
1a5o	18/02/2010 04:57	HSSP File	2,099 KB
1a6x	17/02/2010 23:35	HSSP File	444 KB
1a7i	17/02/2010 23:35	HSSP File	89 KB
1a23	17/02/2010 23:33	HSSP File	225 KB
1a56	17/02/2010 23:34	HSSP File	97 KB
1a63	17/02/2010 23:34	HSSP File	532 KB
1a90	17/02/2010 23:35	HSSP File	71 KB
1a92	18/02/2010 07:19	HSSP File	149 KB
1a93	17/02/2010 23:35	HSSP File	97 KB
1aaf	17/02/2010 23:35	HSSP File	334 KB
1ab7	17/02/2010 23:35	HSSP File	42 KB
1abv	17/02/2010 23:35	HSSP File	184 KB
1adw	18/02/2010 04:57	HSSP File	110 KB
1adx	17/02/2010 23:36	HSSP File	14 KB
1ae9	18/02/2010 07:10	HSSP File	533 KB
1afo	17/02/2010 23:37	HSSP File	19 KB
1afp	17/02/2010 23:37	HSSP File	14 KB
1ag4	17/02/2010 23:37	HSSP File	35 KB
1ah9	17/02/2010 23:37	HSSP File	394 KB
1aho	18/02/2010 09:05	HSSP File	78 KB
1aiw	17/02/2010 23:37	HSSP File	36 KB

Εικόνα 3.8: *Ο φάκελος MSAprodution περιέχει τα .hssp αρχεία που πήραμε από την HSSP βάση χρησιμοποιώντας το πρόγραμμα «script.sh».*

Έχοντας τα αρχεία όλων των διαθέσιμων πρωτεϊνών από την βάση HSSP το επόμενο βήμα είναι να διαλέξουμε από τα αρχεία αυτά, τα MSA profiles μόνο των πρωτεϊνών που μας ενδιαφέρουν, δηλαδή τις πρωτεΐνες του αρχείου του Hobohm. Άρα η επόμενη διαδικασία είναι να υλοποιήσουμε το κύριο πρόγραμμα το οποίο γνωρίζοντας τις πρωτεΐνες του Hobohm και έχοντας τα αρχεία από την βάση HSSP, θα μας δώσει σαν έξοδο την νέα κωδικοποίηση βασισμένη στα MSA profiles (σχήμα 3.8).

```
Διάβασε hobohmFILE

Αποθήκευσε όλες τις πρωτεΐνες σε ένα πίνακα a

Αποθήκευσε το πλήθος των πρωτεϊνών

Για κάθε πρωτεΐνη x από τον πίνακα a

{

        Πάρε το όνομα της πρωτεΐνης

        Πάρε την κλάση της πρωτεΐνης

        Εάν η πρωτεΐνη με την κλάση της υπάρχει στον φάκελο MSAproductions

        {

                Αποθήκευσε πρωτοταγή δομή

                Αποθήκευσε δευτεροταγή δομή

                Αποθήκευσε msa

        }

        Τύπωσε πρωτοταγή δομή στο αρχείο

        Τύπωσε δευτεροταγή δομή στο αρχείο

        Τύπωσε msa στο αρχείο

}
```

Σχήμα 3.8: *Ψευδοκώδικας για την εξαγωγή των MSA profiles των πρωτεϊνών του Hobohm.*

Αρχικά θα δώσουμε τα τρία αρχεία εισόδου στο πρόγραμμα μέσω της γραμμής εντολών. Το πρώτο αρχείο, είναι το αρχείο εισόδου από το οποίο το πρόγραμμα θα διαβάσει το σύνολο των πρωτεϊνών για τις οποίες θέλουμε να βρει το MSA profile τους. Μπορεί να είναι και ένα προηγούμενο σύνολο εκπαίδευσης ή επαλήθευσης (training/testing dataset) του δικτύου, με την ανάλογη μορφή που δέχεται το πρόγραμμα. Στην περίπτωση μας θέλουμε να πάρουμε τα MSA profiles όλων των πρωτεϊνών του αρχείου του Hobohm άρα το αρχείο του Hobohm θα είναι η είσοδος. Το αρχείο αυτό το πήραμε μέσω του προγράμματος «perlCreateProteinFromHobohm.pl»

Σαν δεύτερο βήμα, το πρόγραμμα διαβάζει και αποθηκεύει σε μια μεταβλητή όλες τις πρωτεΐνες από το αρχείο, καθώς και το πλήθος τους. Να υπενθυμίσουμε ότι κάθε πρωτεΐνη, αφού πήραμε το κωδικό όνομά της από το αρχείο του Hobohm, το πέμπτο γράμμα της συμβολίζει την κλάση της ενώ στον φάκελο με τις πρωτεΐνες που πήραμε από την βάση HSSP το πέμπτο γράμμα των πρωτεϊνών εξαιρείται. Τώρα είμαστε έτοιμοι να κάνουμε την διαδικασία εξαγωγής των MSA profiles. Κατασκευάζουμε ένα νέο φάκελο έστω *MSAfilesTemp* στον οποίο θα αποθηκεύουμε την κωδικοποίηση της κάθε πρωτεΐνης. Το όνομα του φακέλου μπορεί να αλλαχθεί από τον χρήστη μέσω του κώδικα του προγράμματος. Ο φάκελος είναι πολύ σημαντικός γιατί μέσω αυτού θα αποθηκεύονται οι κωδικοποιήσεις για κάθε πρωτεΐνη στο σύστημα. Ο φάκελος αυτός θα έχει μορφή όπως στο σχήμα 3.9.

Σχήμα 3.9: *Τα νέα αρχεία έχουν το όνομα της κάθε πρωτεΐνης συνοδευόμενο από την κλάση της και περιέχουν το SEQUENCE profile block (Υποκεφάλαιο 3.2) .*

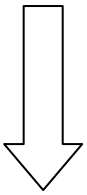
Για κάθε πρωτεΐνη που βρίσκεται στον πίνακα πρωτεϊνών, αποθηκεύουμε το όνομα της χωρίς την κλάση της σε μια μεταβλητή έστω *temp* και το γράμμα που αντιπροσωπεύει την κλάση της σε μια άλλη μεταβλητή έστω *class*. Στην συνέχεια,

ψάχνουμε μέσα στο *MSAproduction* φάκελο για να βρούμε αν υπάρχει το ανάλογο αρχείο της πρωτεΐνης που συμβολίζει το τρέχον temp. Εάν δεν βρεθεί, σημαίνει δεν μπορούμε να δώσουμε κωδικοποίηση για αυτήν την πρωτεΐνη, άρα προχωρούμε στην επόμενη πρωτεΐνη του πίνακα. Επίσης, για ευκολία, στο παρόν στάδιο αποκλείονται όλες οι πρωτεΐνες που έχουν class 0-9 και όχι ένα γράμμα, κάτι το οποίο μπορεί να τροποποιηθεί αργότερα. Εάν βρεθεί λοιπόν το *.hssp* αρχείο της πρωτεΐνης, το ανοίγουμε και προχωρούμε στο ALINGMENT block. Απ εκεί διαβάζουμε κάθε γραμμή του block αυτού, εφόσον η κλάση είναι η ίδια με το class που θέλουμε. Έτσι αποθηκεύουμε την πρωτοταγή δομή και την δευτεροταγή δομή σε δύο μεταβλητές *primary* και *secondary* αντίστοιχα. Το πρόγραμμα, εάν εντοπίσει ελλιπή στοιχεία στην δευτεροταγή δομή βάζει στην θέση τους το γράμμα «L», το οποίο στην γλώσσα της δευτεροταγούς δομής σημαίνει οτιδήποτε εκτός από H και E. Αυτά τα αποθηκεύει στο δεύτερο αρχείο *dataset_file*. Στην συνέχεια διαβάζει από το SEQUENCE PROFILE block το MSA profile της πρωτεΐνης αυτής (ανάλογα πάντα με την πολυπεπτιδική αλυσίδα της πρωτεΐνης). Οι τιμές αυτές πρώτα διαιρούνται με το 100 για να δώσουν ένα ποσοστό από 0 μέχρι 1. Τέλος δημιουργεί ένα αρχείο σύμφωνα με το όνομα και την πολυπεπτιδική αλυσίδα της πρωτεΐνης και αποθηκεύει το MSA profile εκεί. Τα αρχεία αυτά βρίσκονται όπως είπαμε σε ένα φάκελο του οποίου το όνομα και το μονοπάτι μπορεί να καθορίσει ο χρήστης.

Ακόμη ένα χρήσιμο πρόγραμμα που υλοποιήθηκε ήταν το «makeProteinSet.pl», το οποίο είναι ένα πρόγραμμα το οποίο παίρνει σαν είσοδο ένα σύνολο δεδομένων πρωτεϊνών, είτε εκπαίδευσης είτε επαλήθευσης. Στην συνέχεια, διαβάζει από το σύνολο αυτό μόνο τις πρωτεΐνες και αγνοεί τις υπόλοιπες γραμμές. Τις αποθηκεύει όλες (απλά τα ονόματά τους) σε μια γραμμή ενός νέου αρχείου, το οποίο είναι της μορφής των αρχείων που δίνονται σαν είσοδο στο msaProgram.pl. Έτσι εύκολα μπορούμε να πάρουμε τις κωδικοποιήσεις αλλά και το καινούργιο σύνολο δεδομένων ενός συγκεκριμένου συνόλου πρωτεϊνών που χρειαζόμαστε.

Η όλη διαδικασία που ακολουθήθηκε για την εξαγωγή και επεξεργασία των αρχείων, χρησιμοποιώντας τους διάφορους parsers φαίνεται στο σχήμα 3.10.

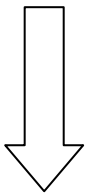
1. Αρχικό αρχείο Hobohm = «recent.pdb_select25»



perlCreateProteinFromHobohm.pl

Έξοδος: αρχείο των πρωτεϊνών.

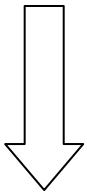
2. Εξαγωγή πρωτεϊνών της μορφής xxxx.hssp από την βάση HSSP και αποθήκευση τους στο MSAproduction φάκελο.



script.sh

Έξοδος: αρχείο των πρωτεϊνών σε μορφή xxxx.hssp.

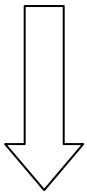
3. Δημιουργία MSA profiles και νέων datasets.



msaProgram.pl

Έξοδος: αρχεία πρωτεϊνών σε μορφή xxxxx.txt.

4. Δημιουργία αρχείου πρωτεϊνών της μορφής HobohmPROTEINS.txt από οποιοδήποτε dataset.



makeProteinSet.pl

Έξοδος: αρχείο της μορφής hobohmPROTEINS.txt.

Σχήμα 3.10: *Η διαδικασία που ακολουθήθηκε για την δημιουργία των msa αρχείων κωδικοποίησης.*

Τέλος δημιουργήθηκε ένα πρόγραμμα για την δημιουργία τυχαίων συνόλων εκπαίδευσης και επαλήθευσης για το σύστημα. Το πρόγραμμα αυτό λέγεται «randomDatasetCreate.pl» και παίρνει σαν είσοδο από την γραμμή εντολών το όνομα του νέου αρχείου εκπαίδευσης που θα δημιουργηθεί, το όνομα του νέου αρχείου επαλήθευσης που θα δημιουργηθεί και τις ποσότητες των πρωτεϊνών που θέλουμε για τα δύο νέα αυτά αρχεία. Το αρχείο από το οποίο δημιουργούνται τα σύνολα δεδομένων εκπαίδευσης και επαλήθευσης, δηλώνεται από τον χρήστη, μέσω της γραμμής εντολών στον κώδικα του προγράμματος και συγκεκριμένα στην μεταβλητή *initialFILE*. Το πρόγραμμα αυτό επιλέγει αρχικά τυχαίες πρωτεΐνες από το initialFILE και δημιουργεί το σύνολο εκπαίδευσης. Έπειτα, οι πρωτεΐνες αυτές που επιλέχθηκαν διαγράφονται από το αρχείο. Έπειτα από τις εναπομείναντες πρωτεΐνες επιλέγεται ένας αριθμός από τυχαίες πρωτεΐνες (παράμετρος του χρήστη), που θα συμπεριλαμβάνονται στο αρχείο επαλήθευσης.

3.4 Επεξεργασία δεδομένων εξόδου του συστήματος

3.4.1 Confusion matrices

Το ποσοστό επιτυχίας του τρέχοντος συστήματος είναι μια τιμή που δείχνει κατά πόσο το δίκτυο πρόβλεψε σωστά τις τιμές από το σύνολο επαλήθευσης. Αποτελείται λοιπόν, από τα διαφορετικά ποσοστά πρόβλεψης της κάθε πρωτεΐνης που ανήκει στο συγκεκριμένο σύνολο επαλήθευσης που εξετάζει και βρίσκει τον μέσο όρο επιτυχίας τους. Αυτός είναι και ο τρόπος που «κρίνουμε» την απόδοση του συστήματος. Για το παρόν σύστημα χρησιμοποιείται η μέθοδος «Q3» (Αγαθοκλέους, 2009), η οποία φαίνεται και στην εξίσωση 3.1. Η μέθοδος Q3 είναι μια μέθοδος βαθμολόγησης της ακρίβειας της πρόβλεψης μιας πρωτεΐνης, η οποία συσχετίζει την κάθε θέση από την επιθυμητή ακολουθία δευτεροταγούς δομής με την προβλεπόμενη ακολουθία δευτεροταγούς δομής και ανάλογα μετρά το ποσοστό ακρίβειας μεταξύ των δύο αυτών θέσεων. Παρόλα αυτά όμως, μια τέτοια τιμή δεν είναι αντιπροσωπευτική της ικανότητας εκμάθησης του δικτύου, εξαιτίας πολλών περιορισμών, όπως θα δούμε πιο κάτω.

$$Qi = \frac{\text{number of residues correctly predicted in state i}}{\text{number of residues observed in state i}} * 100$$

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

Εξίσωση 3.1: *Η μέθοδος βαθμολόγησης της πρόβλεψης του δικτύου, η οποία βασίζεται στον αριθμό των καταλοίπων που προβλέφτηκαν σωστά.*

Επιθυμητή δευτεροταγής δομή:	H H H H H E H H E E
Προβλεπόμενη δευτεροταγής δομή:	H H H H H H H H H H

Σχήμα 3.11: *Μια πρωτεϊνική ακολουθία στην οποία φαίνεται η πραγματική δευτεροταγής δομή της, καθώς και μια προβλεπόμενη αυτής.*

Ας μελετήσουμε λίγο το σχήμα 3.11. Με βάση το Q3 ποσοστό επιτυχίας του δικτύου, έχουμε μια πρόβλεψη με ποσοστό 0.7 γιατί τα 7 στα 10 κατάλοιπα προβλέφτηκαν σωστά. Αυτό, αποτελεί μια πολύ καλή πρόβλεψη για το δίκτυο και θεωρητικά αυτό σημαίνει ότι το δίκτυο μαθαίνει αρκετά καλά. Αυτό όμως δεν συμβαίνει πάντοτε. Μια καλή τιμή με βάση μια μετρική όπως το Q3, δεν σημαίνει απαραίτητα ότι το δίκτυο μπορεί να προβλέψει όλα τα είδη δευτεροταγούς δομής. Υπάρχει πιθανότητα το δίκτυο να συγχύζει δύο ή και περισσότερες τιμές εξόδου. Με βάση την πρόβλεψη της δευτεροταγούς δομής του σχήματος 3.11 για παράδειγμα, μπορεί ναι το δίκτυο να προβλέπει σωστά όλα τα Helixes (H) αλλά, δεν μπορεί να προβλέψει σωστά ούτε ένα από τα 3 Extended (E). Αυτό αποτελεί σημαντικό πρόβλημα για ένα σύστημα. Αυτό

μπορεί να γίνεται και για τα υπόλοιπα είδη δευτεροταγούς δομής, γι αυτό πρέπει να ελεγχθεί κατά πόσο συμβαίνει αυτό στο τρέχον σύστημα και αν όντως συμβαίνει για ποια περίπτωση.

Ένας τρόπος για να μπορέσουμε να εξετάσουμε στατιστικά αυτό το γεγονός και έτσι να μπορέσουμε να «μετρήσουμε» την ικανότητα πρόβλεψης του δικτύου για όλες τις πιθανές τιμές δευτεροταγούς δομής H, E και L ώστε να δούμε εάν παρουσιάζεται ένα τέτοιο πρόβλημα, είναι να κατασκευάσουμε τα confusion matrices κάθε πρωτεΐνης.

Τα confusion matrices είναι ένα είδος πίνακα όπως φαίνεται από το σχήμα 3.12, 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 3.12: *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 3.12 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».

Είναι λοιπόν πάρα πολύ σημαντικό να δημιουργήσουμε τα confusion matrices για τις πρωτεΐνες που εκπαιδεύονται και επαληθεύονται με το τρέχον σύστημα, γιατί είναι ένας τρόπος να το αξιολογήσουμε όσον αφορά τις ξεχωριστές ομάδες της δευτεροταγούς δομής που προβλέπει, και να εντοπίσουμε τυχόν συγχύσεις που μπορεί να γίνονται μεταξύ τους.

Για να σχεδιάσουμε και να υλοποιήσουμε την πιο πάνω διαδικασία για το σύστημα μας, θα κατασκευάσουμε ένα πρόγραμμα, με την βοήθεια της γλώσσας PERL. Αυτό το πρόγραμμα θα είναι επίσης ξεχωριστό και αυτόνομο, ώστε οποιοσδήποτε και οποτεδήποτε να μπορεί να το εκτελέσει. Το πρόγραμμα αυτό θα ονομάζεται «confusionMatrices.pl».

Αρχικά, θα υπάρχει μια μεταβλητή του προγράμματος, η οποία θα πρέπει να αρχικοποιηθεί από τον χρήστη. Η μεταβλητή αυτή αναπαριστά το όνομα του αρχείου εισόδου. Το αρχείο εισόδου θα είναι το αρχείο εξόδου του νευρωνικού δικτύου με ανάδραση, δηλαδή ένα αρχείο που έχει όλες τις πρωτεΐνες επαλήθευσης, την επιθυμητή δευτεροταγή δομή και την προβλεπόμενη δευτεροταγή δομή που δίνει το σύστημα. Το πρόγραμμα, παίρνει από το αρχείο αυτό την επιθυμητή δευτεροταγή ακολουθία καθώς και την προβλεπόμενη ακολουθία που έδωσε το δίκτυο και εξετάζει για κάθε θέση την δευτεροταγή δομή που έχει η επιθυμητή, το συγκρίνει με την δευτεροταγή δομή στην ανάλογη θέση στην προβλεπόμενη ακολουθία και συμπληρώνει έπειτα τον πίνακα με ανάλογο τρόπο όπως φαίνεται και από το σχήμα 3.12. Για κάθε μια από αυτές τις πρωτεΐνες ο ανάλογος πίνακας τους θα αποθηκεύεται σε ένα ξεχωριστό αρχείο .txt το οποίο έχει το όνομα της κάθε πρωτεΐνης.

Ο ψευδοκώδικας για την πιο πάνω διαδικασία φαίνεται στο σχήμα 3.13.

```
Δώσε όνομα αρχείου εισόδου
Για κάθε πρωτεΐνη x του αρχείου εισόδου
{
        Αποθήκευσε επιθυμητή δευτεροταγή δομή
        Αποθήκευσε πραγματική δευτερσταγή 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
        }
        Μετάφερε τον πίνακα στο νέο αρχείο x.txt
}
```

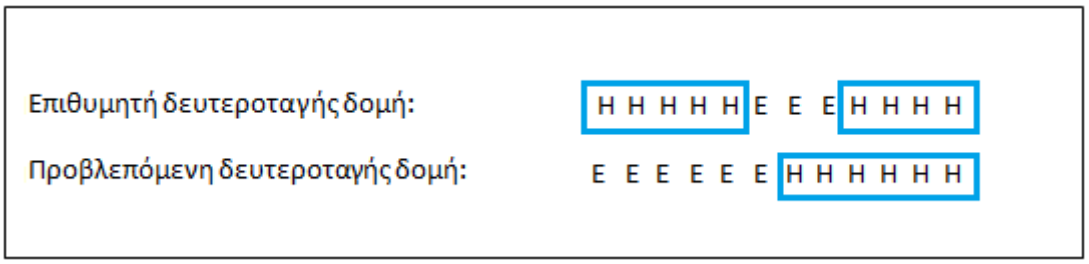
Σχήμα 3.13: *Ψευδοκώδικας για την δημιουργία του προγράμματος «confusionMatrices.pl» το οποίο θα υπολογίζει το confusion matrix κάθε πρωτεΐνης ενός αρχείου εισόδου.*

3.4.2 Segment OVerlap:

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, όπως είδαμε και από την εξίσωση 3.1.

Το SOV είναι μια μέθοδος βαθμολόγησης της προβλεπόμενης ακολουθίας η οποία είναι πολύ πιο αυστηρή από το Q3, 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 Helixes ενώ η προβλεπόμενη ακολουθία προβλέπει μόνο ένα Helix διάστημα (σχήμα 3.14), 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 3.14, 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 3.14: *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 3.2, (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 3.2). Επιπλέον η τιμή δ(s1,s2) καθορίζει μια θετική τιμή ίση με το αποτέλεσμα της εξίσωσης 3.2 (της δεύτερης εξίσωσης), όπου len(s1) το μήκος του s1 τμήματος και len(s2), το μήκος του s2 τμήματος.

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

$$\delta(s_1, s_2) = \min\left|(\text{maxov}(s_1, s_2) - \text{minov}(s_1, s_2)),\right.$$
$$\left.\text{minov}(s_1, s_2), \text{int}(\text{len}(s_1)/2), \text{int}(\text{len}(s_2)/2)\right|$$

Εξίσωση 3.2: *Οι εξισώσεις υπολογισμού της νέας μεθόδου βαθμολόγησης προβλεπόμενων ακολουθιών, SOV (Από Zemla et al., 1999).*

Το εσωτερικό άθροισμα (εξίσωση 3.2) 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 Helixes (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 3.3: *Μέθοδος υπολογισμού του όρου N(i) (Από Zemla et al., 1999).*

Η τιμή N (εξίσωση 3.3), για κάποια δευτεροταγή δομή, 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.

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

Για να δούμε την σημαντικότητα αυτής της νέας μετρικής ας δούμε το πιο πάνω παράδειγμα του σχήματος 3.15, 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 ποσοστό 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.

Το πρόγραμμα το οποίο υπολογίζει αυτήν την νέα μετρική SOV, είναι ήδη υλοποιημένο και διαθέσιμο (SOV, 23 April 2010), έτσι θα το χρησιμοποιήσουμε χωρίς καμία αλλαγή. Το μόνο που θα αλλάξουμε είναι την μορφή των αρχείων που δίνει το τρέχον σύστημα μας, 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 3.2, 3.3), 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 3.16).

```
 .
 .
 .
 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 3.16: *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.*

Θέλουμε λοιπόν να τροποποιήσουμε το πρόγραμμα που υπολογίζει τα confusion matrices για κάθε πρωτεΐνη, ώστε να καλεί το πρόγραμμα αυτό για να υπολογίζει και αυτές τις απαραίτητες πληροφορίες για τις προβλεπόμενες ακολουθίες του συστήματος. Επίσης, λόγω του ότι το πρόγραμμα δημιουργίας των SOV πληροφοριών εμπεριέχει αυτόματα τον υπολογισμό της εξίσωσης Q3, αυτός είναι ένας τρόπος για να επαληθεύσουμε την ορθότητα του δικού μας συστήματος.

Θα προσθέσουμε τρείς διαφορετικές υπορουτίνες στο πρόγραμμα «confusionMatrices.pl» το οποίο μετονομάζεται σε «confusionMatricesAndSOV.pl». 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. Έπειτα το πρόγραμμα SOV, δίνει σαν έξοδο ένα αρχείο στην μορφή του σχήματος 3.16. Από αυτό το αρχείο μας ενδιαφέρει μόνο να πάρουμε 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*. Έπειτα, θα υπάρχει μια τελευταία υπορουτίνα, η οποία 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. Η συνάρτηση αυτή ονομάζεται *printInformationToFile.* Το νέο πρόγραμμα με τις πιο πάνω αλλαγές φαίνεται στον ψευδοκώδικα του σχήματος 3.17.

```
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 3.17: *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 3.18 ενώ η μορφή του γενικού αρχείου για ολόκληρο το σύνολο πρωτεϊνών που σχηματίζεται από το πρόγραμμα, φαίνεται στο σχήμα 3.19. 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           :LHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHI
PredictedSecondaryStructure :LHHHHHHHHHHHHHHHHHHHHLHHHHHHHHHHLLHHHLLLLLLHHHLHHHHLLI


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 3.18: *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
2comA	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 3.19: *Τυπικό γενικό αρχείο εξόδου από το πρόγραμμα «confusionMatricesAndSOV.pl» για όλες τις πρωτεΐνες εισόδου.*

Κεφάλαιο 4

Σχεδιασμός και Υλοποίηση

4.1 Αλλαγή momentum

4.2 Αλλαγή χρονικών μεταβλητών

4.3 Υλοποίηση MSA profiles κωδικοποίησης

4.4 Υλοποίηση επιλογής συνάρτησης ενεργοποίησης

4.4.1 Υπερβολική Εφαπτομένη

4.4.2 Γραμμική Συνάρτηση

4.5 Τυχαιοποίηση συνόλου δεδομένων

4.6 Ενσωμάτωση Γενετικών Αλγορίθμων

4.1 Αλλαγή momentum

Κατ' αρχήν, η πρώτη τροποποίηση που θα γίνει στο υπάρχον πρόγραμμα, θα αφορά τον όρο της ορμής, ο οποίος χρησιμοποιείται στην κλάση *Neuron.cpp* και συγκεκριμένα στην μέθοδο *adjustDw*. Ο όρος της ορμής χρησιμοποιείται σε τεχνητά νευρωνικά δίκτυα σαν ένας επιπρόσθετος όρος στην εξίσωση αλλαγής των βαρών του δικτύου, κατά την δεύτερη φάση του *backpropagation* αλγόριθμου. Η ορμή, έχει την τάση να διαμορφώνει την αλλαγή των βαρών με τον εξής τρόπο: εάν η διαφορά των δύο βαρών είναι μεγάλη, την κάνει ακόμη μεγαλύτερη ούτως ώστε να πλησιάσει περισσότερο στην πρέπουσα τιμή του βάρους, και εάν η διαφορά των δύο βαρών είναι μικρή, την κάνει μικρότερη. Με αυτόν τον τρόπο ελαχιστοποιείται η πιθανότητα το δίκτυο να «κολλήσει» σε ένα τοπικό ελάχιστο, κάτι που δεν θέλουμε να συμβεί κατά την διάρκεια εκπαίδευσης. Έτσι, θα έχουν τα συναπτικά βάρη του δικτύου την δυνατότητα να προσαρμοστούν όσο καλύτερα γίνεται, ώστε να ελαχιστοποιήσουν το συνολικό σφάλμα του δικτύου.

Το τρέχον πρόγραμμα όμως, μηδενίζει τον όρο της ορμής κάθε φορά που υπολογίζονται τα νέα βάρη για το δίκτυο. Άρα η ορμή πάντα είναι μηδέν, δηλαδή ουσιαστικά, οι εξισώσεις των βαρών δεν συμπεριλαμβάνουν τον όρο της ορμής. Γι αυτό, τροποποιήσαμε τις εξισώσεις αυτές ώστε να διορθώσουμε το πρόβλημα. Προγραμματιστικά οι εξισώσεις που χρησιμοποιούνται για την αλλαγή των βαρών και για την ορμή φαίνονται στο σχήμα 4.1 και είναι σύμφωνες με τις εξισώσεις αλλαγής βαρών που χρησιμοποιούνται στα τεχνητά νευρωνικά δίκτυα αφότου υποστούν κάποια περεταίρω μαθηματική επεξεργασία. Δεν χρειάζονται κάποια αλλαγή παρά μόνο να μπουν σε σχόλια οι δύο τελευταίες γραμμές οι οποίες μηδενίζουν τον όρο της ορμής. Παρόλα αυτά έχουμε διαμορφώσει ευκολότερα τις εξισώσεις αυτές, πάντα με το ίδιο σκεπτικό όπως είναι γραμμένες και υλοποιημένες στο παρόν πρόγραμμα ώστε να είναι πιο εύκολα κατανοητές (σχήμα 4.2).

```
Για κάθε βάρος i ενός νευρώνα

{

    Dweight[i] = Dweight[i] + learningRate * inputVector[i] * error + momentum * previousAdjustment[i]

    previousAdjustment[i] = learningRate * inputVector[i] * error + momentum * previousAdjustment[i]

    weightsVector[i] = weightsVector[i] + Dweight[i]

    Dweight[i] = 0

    previousAdjustment[i] = 0

}
```

Σχήμα 4.1: *Τρέχον ψευδοκώδικας της φάσης ενημέρωσης των βαρών του δικτύου με τις ανάλογες εξισώσεις.*

```
Για κάθε βάρος i ενός νευρώνα

{

    Dweight[i] = Dweight[i] + learningRate * inputVector[i] * error + momentum * (weightsVector[i] -
previousAdjustment[i])

    previousAdjustment[i] = weightsVector[i]]

    weightsVector[i] = weightsVector[i] + Dweight[i]

}
```

Σχήμα 4.2: *Νέος ψευδοκώδικας της φάσης ενημέρωσης των βαρών του δικτύου με τις ανάλογες εξισώσεις και ενεργοποίηση του όρου της ορμής.*

4.2 Αλλαγή χρονικών μεταβλητών

Όπως γνωρίζουμε, το νευρωνικό δίκτυο με αμφίδρομη ανάδραση που χρησιμοποιείται στο συγκεκριμένο σύστημα είναι βασισμένο στην αρχιτεκτονική του δικτύου του Baldi (Baldi et al., 1999) και κάνει συνεπώς χρήση των χρονικών μεταβλητών q και q-1. Οι μεταβλητές αυτές χρησιμοποιούνται στα δύο δίκτυα με ανάδραση. Όταν τα νευρωνικά δίκτυα με ανάδραση δώσουν κάποιες τιμές, οι τιμές αυτές πρέπει να επανέλθουν πίσω στο επίπεδο εισόδου. Έτσι προσομοιώνεται η

έννοια της *μνήμης* του δικτύου. Το δίκτυο πρέπει να θυμάται προηγούμενα ή επόμενα αμινοξέα του κεντρικού αμινοξέως, τα οποία επεξεργάστηκαν τα δυο δίκτυα με ανάδραση. Άρα οι τιμές εξόδου των δικτύων αυτών πρέπει να μεταφέρονται στο επίπεδο εισόδου για να επεξεργαστούν ξανά. Όμως αυτές οι τιμές δεν πρέπει να μεταφέρονται για επεξεργασία αναλλοίωτες. Πρέπει να υπάρχει μια μικρή τροποποίηση τους ούτως ώστε να προσομοιώνεται η *αλλαγή των τιμών στον χρόνο,* γι' αυτό οι τιμές αυτές πολλαπλασιάζονται με κάποια σταθερά πριν να εισαχθούν ξανά στο επίπεδο εισόδου.

Με βάση την αρχιτεκτονική του δικτύου του Baldi (Baldi et al., 1999) οι τιμές των Ft και Bt νευρωνικών δικτύων με ανάδραση (υποκεφάλαιο 2.1) αντιγράφονται πίσω στο επίπεδο εισόδου χρησιμοποιώντας τις μεταβλητές q και q^-1 αντίστοιχα, ώστε να ισχύει η εξίσωση 4.1.

$$q * q^{-1} = 1$$

Εξίσωση 4.1: *Η εξίσωση που ικανοποιεί τις μεταβλητές q και q^-1 με βάση τον Baldi (Από Baldi et al., 1999).*

Ακόμη μια τροποποίηση που θα γίνει στο υπάρχον σύστημα λοιπόν, θα αφορά τις χρονικές μεταβλητές q και q^-1 των δύο δικτύων με ανάδραση Ft και Bt αντίστοιχα. Η παρούσα υλοποίηση τους, δεν εφαρμόζει τον κανόνα ότι q * q^-1 = 1, αλλά, εφαρμόζει τον κανόνα q = q^-1. Θα πρέπει λοιπόν να αλλάξει η εξίσωση χρήσης των μεταβλητών αυτών.

Το αρχείο parameters.dat στην παρούσα φάση, είναι σχεδιασμένο τέτοιο ώστε να ζητά τιμές και για το q και για το q^-1. Για να μην αλλάξουμε την δομή των αρχείων αυτών θα το αφήσουμε ακριβώς όπως είναι, και θα προσθέσουμε ένα κομμάτι κώδικα ώστε να επιτυγχάνεται η πιο πάνω εξίσωση. Έτσι λοιπόν, το αρχείο αρχικοποίησης παραμέτρων θα δέχεται και τις δύο αυτές μεταβλητές, με την προϋπόθεση ότι η τιμή τους είναι η ίδια.

Η αλλαγή της q^-1 θα γίνεται εντός του προγράμματος για να μην μπαίνει ο χρήστης σε μαθηματικές διαδικασίες για να δώσει τιμές στο αρχείο εισόδου. Στην κλάση *BiderectionalReccurentNeuralNetwork.cpp* θα προστεθεί ένα κομμάτι κώδικα (σχήμα 4.3) ώστε, με βάση την τιμή που δίνεται στις μεταβλητές q και q^-1 του αρχείου αρχικοποίησης, να υπολογίζεται η δεύτερη μεταβλητή q^-1 και να χρησιμοποιείται η νέα της τιμή στο δίκτυο.

```
q^-1 = ( 1.0 / q )
```

Σχήμα 4.3: *Ψευδοκώδικας τροποποίησης της μεταβλητής q^-1.*

4.3 Υλοποίηση MSA profiles κωδικοποίησης

Αρχικά θα πρέπει να τοποθετήσουμε τον φάκελο με τις νέες κωδικοποιήσεις των πρωτεϊνών (MSA profiles), τις οποίες πήραμε 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, θα χρειαστούμε 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.

Όμως πρέπει να γίνουν και αρκετές αλλαγές σε επίπεδο κώδικα στις διάφορες κλάσεις που απαρτίζουν το σύστημα. Κατ αρχήν προσθέτουμε 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» (υποκεφάλαιο 3.3.1). Έτσι θα υλοποιηθεί 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. Αυτή η μέθοδος θα χρησιμοποιείται μόνο όταν έχουμε κωδικοποίηση βασισμένη σε MSA profiles. Φυσικά θα χρειαστούν επιπρόσθετες μέθοδοι όπως η μέθοδος *splitValues()* η οποία θα παίρνει αυτήν την συμβολοσειρά και θα την σπάζει ώστε να πάρει την κάθε τιμή ξεχωριστά.

```
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 4.4: *Ψευδοκώδικας για την μέθοδο readEncodingMSA.*

Σημαντικές αλλαγές διεκπεραιώθηκαν στην κύρια κλάση του προγράμματος *BidirectionalRecurrentNeuralNetwork.cpp*, ώστε να μπορεί το δίκτυο να επεξεργάζεται την νέα κωδικοποίηση. Κατ' αρχήν πρέπει να αλλάξει η υλοποίηση της μεθόδου *getInitialInput()*, η οποία είναι υπεύθυνη να αρχικοποιεί από κάποια αρχεία κάποιες 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 χρησιμοποιούσαμε 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 4.4).

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 4.5).

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

Σχήμα 4.5: *Ψευδοκώδικας για τη μέθοδο createMSAtempInput.*

Τώρα, όσον αφορά το πότε θα γίνεται αυτό, έχουμε προσθέσει ένα μικρό κομμάτι κώδικα στην 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 δημιουργήθηκε, με τις τιμές των 20 αμινοξέων για την συγκεκριμένη θέση της πρωτεΐνης που εξετάζουμε (σχήμα 4.6).

```
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 4.6: *Ψευδοκώδικας εισαγωγής MSA κωδικοποίησης.*

Οι τελευταίες αλλαγές θα γίνουν 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 αρχίσουμε την επεξεργασία των αμινοξέων χρειάζεται να ξέρουμε για κάθε πρωτεΐνη την κωδικοποίηση της. Έτσι λοιπόν θα προσθέσουμε μια εντολή όπου θα πρέπει να παίρνει την κωδικοποίηση και να την αποθηκεύει στο συγκεκριμένο vector. Μετά θα εκτελούνται οι δύο βασικές συναρτήσεις *doFeedForward()* και *doBackpropagation()* (σχήμα 4.7).

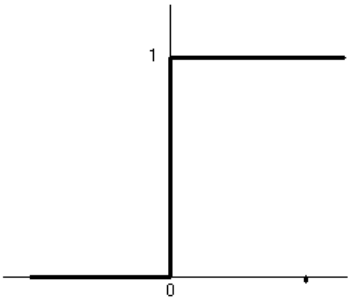
```
While (epoch<maxEpoch)
{
    While (υπάρχουν πρωτεΐνες στο training dataset)
    {
        Πάρε την επόμενη πρωτεΐνη x
        Πάρε την κωδικοποίηση της πρωτεΐνης x
        Για κάθε αμινοξύ t της πρωτεΐνης x
        {
            doFeedForward (t)
            doBackPropagation (t)
        }
      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 4.7: *Ψευδοκώδικας με τον οποίο τροποποιείται η κλάση pssp.ucy, ώστε να επεξεργάζεται την χρήση των MSA profiles.*

4.4 Υλοποίηση επιλογής συνάρτησης ενεργοποίησης

Συνεχίζοντας, θα εφαρμόσουμε διάφορες εμπειρικές τεχνικές για βελτιστοποίηση της απόδοσης του δικτύου. Η επεξεργασία του νευρώνα καθορίζεται από την συνάρτηση ενεργοποίησης η οποία καθορίζει την έξοδο του νευρώνα σε σχέση με τα βάρη και τις τιμές εισόδου του. Οι συναρτήσεις ενεργοποίησης των νευρώνων επηρεάζουν κατά πολύ τα αποτελέσματα εξόδου του δικτύου γι αυτό είναι πολύ κρίσιμη η επιλογή τους.

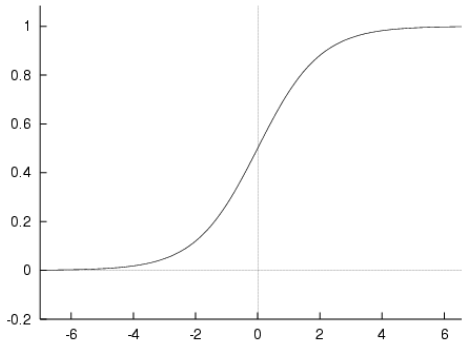
Υπάρχουν πολλές συναρτήσεις ενεργοποίησης η κάθε μια με τα πλεονεκτήματα και τα μειονεκτήματα της. Η συνάρτηση κατωφλίου, ή λεγόμενη *step function* είναι η συνάρτηση ενεργοποίησης που βλέπουμε στο σχήμα 4.8. Η συνάρτηση κατωφλίου χρησιμοποιείται στο σύστημα μας, και ειδικότερα στους νευρώνες εξόδου ώστε να προσομοιώνει τις εξόδους τους σε μια από τις καθορισμένες τιμές εξόδου που ορίσαμε σε αρχείο αρχικοποίησης. Η τιμή των νευρώνων κυμαίνεται μεταξύ 0 και 1 αφού χρησιμοποιούν την σιγμοειδή, έτσι η τιμή του x είναι 0.5 για την περίπτωση αυτή.



$$\phi(\upsilon) = \begin{cases} 1, \upsilon \geq \mathrm{x} \\ 0, \upsilon < \mathrm{x} \end{cases}$$

Σχήμα 4.8: *Συνάρτηση κατωφλίου όπου x δηλώνει την οριακή τιμή μεγαλύτερη της οποίας ο νευρώνας πυροβολεί, και μικρότερη της οποίας ο νευρώνας δεν πυροβολεί.*

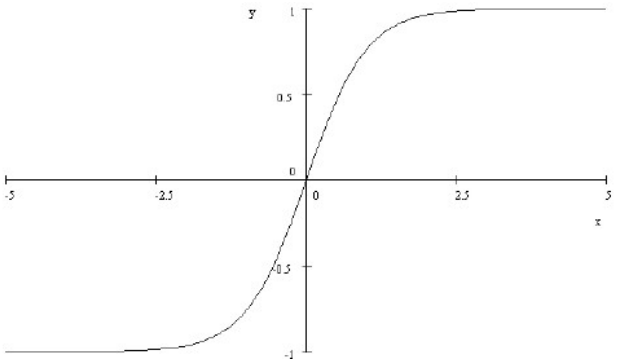
Η σιγμοειδής συνάρτηση ενεργοποίησης είναι η κύρια συνάρτηση όλων των υπόλοιπων επιπέδων του νευρωνικού δικτύου. Είναι μια μη γραμμική συνάρτηση, και το πεδίο τιμών της κυμαίνεται μεταξύ 0 και 1, κάτι που είναι αρκετά σημαντικό εφόσον οι επιθυμητές τιμές και οι τιμές εισόδου είναι 0 και 1 (σχήμα 4.9).



Σχήμα 4.9: *Σιγμοειδής συνάρτηση ενεργοποίησης. Το πεδίο τιμών της κυμαίνεται μεταξύ 0 και 1.*

4.4.1 Υπερβολική Εφαπτομένη

Αρχικά, θα αλλάξουμε την σιγμοειδή συνάρτηση ενεργοποίησης των νευρώνων σε υπερβολική εφαπτομένη.



Σχήμα 4.10: *Η συνάρτηση ενεργοποίησης υπερβολικής εφαπτομένης. Οι τιμές της κυμαίνονται μεταξύ -1 και 1, σε αντίθεση με την σιγμοειδή συνάρτηση ενεργοποίησης.*

Η υπερβολική εφαπτομένη σαν συνάρτηση ενεργοποίησης είναι μια αντισυμμετρική συνάρτηση, σε αντίθεση με την σιγμοειδή η οποία είναι μια μη συμμετρική συνάρτηση. Η υπερβολική εφαπτομένη είναι μια συνάρτηση ενεργοποίησης η οποία προσεγγίζει την βηματική συνάρτηση αλλά εξακολουθεί να είναι παραγωγίσιμη. Μια τέτοια υπερβολική εφαπτομένη συνάρτηση φαίνεται στο σχήμα 4.10, και

χαρακτηρίζεται από την εξίσωση 4.2, όπου α είναι το πλάτος της και b είναι η κλίση της.

$$\phi(\upsilon) = \phi(- \upsilon)$$

$$y = a \tanh(bx)$$

Εξίσωση 4.2: *Εξίσωση της υπερβολικής εφαπτομένης. Από εδώ φαίνεται ότι είναι μια αντισυμμετρική συνάρτηση όπου a το πλάτος της και b η κλίση της συνάρτησης.*

Με βάση τον LeCun (LeCun, 1989), μπορεί να χρησιμοποιηθεί η υπερβολική εφαπτομένη με πλάτος 1.7159 και κλίση 2/3.

Προγραμματιστικά θέλουμε να εντάξουμε στο σύστημα την δυνατότητα χρήσης της συνάρτησης αυτής. Αυτό μπορεί να γίνει πολύ εύκολα, επειδή το αρχείο parameters.dat, δέχεται σαν παράμετρο το είδος της συνάρτησης ενεργοποίησης που θέλει κάποιος να χρησιμοποιήσει για την εκτέλεση του δικτύου, η οποία είναι το «Activation_Function_Type». Όταν η παράμετρος αυτή έχει την τιμή 1 αυτόματα ενεργοποιεί σαν συνάρτηση ενεργοποίησης των νευρώνων όλων των επιπέδων την σιγμοειδή. Όταν όμως θα παίρνει την τιμή 2, θα ενεργοποιείται σαν συνάρτηση ενεργοποίησης η υπερβολική εφαπτομένη. Φυσικά, στην προηγούμενη μελέτη που διεξήχθη από τον Αγαθοκλέους (Αγαθοκλεους, 2009), δεν είχαν μπει παντού στο πρόγραμμα οι έλεγχοι για το ποια από τις δύο συναρτήσεις να χρησιμοποιηθεί, επειδή η κύρια συνάρτηση ενεργοποίησης που χρησιμοποιήθηκε ήταν η σιγμοειδής. Παρόλα αυτά, το δίκτυο προνοεί την ύπαρξη οποιασδήποτε άλλης συνάρτησης, όπως είδαμε και από τις παραμέτρους στο αρχείο εισόδου. Έτσι λοιπόν, προστέθηκαν δύο μεταβλητές amplitude και slope στην μέθοδο «Neuron» οι οποίες παίρνουν κάποιες σταθερές τιμές σύμφωνα με τον LeCun (LeCun, 1989). Έπειτα προσθέσαμε την εξίσωση της υπερβολικής εφαπτομένης με βάση τον LeCun στην μέθοδο *callActivationFunction* μαζί με τους απαραίτητους ελέγχους. Αυτό όμως δεν ήταν αρκετό, αφού για την χρήση κάποιας άλλης συνάρτησης ενεργοποίησης όπως για

παράδειγμα η υπερβολική εφαπτομένη, θα πρέπει να αλλάξουμε και τις εξισώσεις που αφορούν το σφάλμα τόσο στους εξωτερικούς, όσο και στους κρυφούς νευρώνες του δικτύου. Προσθέσαμε λοιπόν, στις μεθόδους *calculateOutputLayerError* και *calculateHiddenLayerError* τις εξισώσεις υπολογισμού του σφάλματος χρησιμοποιώντας την υπερβολική εφαπτομένη. Αυτό, υπάρχει και σαν παράμετρος στο αρχείο parameters.dat σαν *Error_Function_Type* και από εκεί ενεργοποιείται η χρήση του. Πρέπει να χρησιμοποιηθεί παράλληλα με τις τιμές που βάζουμε στο *Activation_Function_Type*.

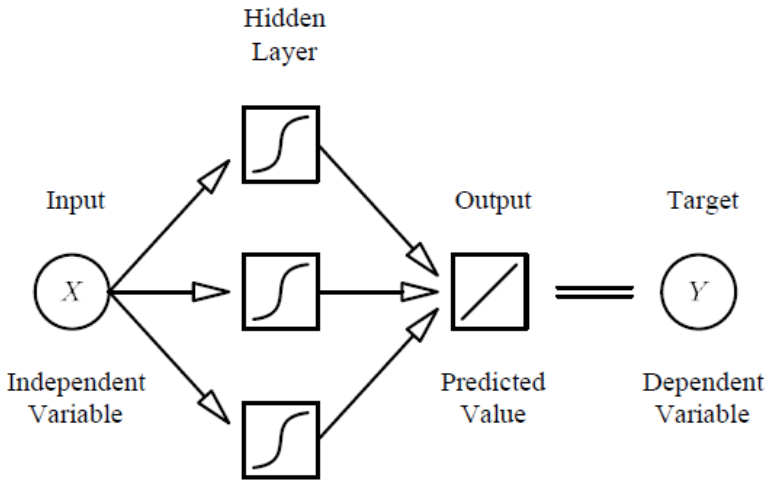
Μια επιπλέον αλλαγή που θα πρέπει να γίνει ώστε να μπορεί το πρόγραμμα να εκτελεστεί χρησιμοποιώντας την υπερβολική εφαπτομένη, είναι να αλλάξει η επιθυμητή έξοδος των τριών εξωτερικών νευρώνων. Αυτό γίνεται γιατί το πεδίο τιμών της υπερβολικής εφαπτομένης με την τροποποίηση του LeCun (LeCun, 1989), κυμαίνεται μεταξύ [-1.7,1.7] σε αντίθεση από της σιγμοειδούς, όπου κυμαίνεται μεταξύ [0,1]. Οι επιθυμητές τιμές των νευρώνων εξόδου, κανονικά, πρέπει να συμπίπτουν με τις ασυμπτωτικές τιμές της συνάρτησης ενεργοποίησης, όπως και γίνεται με την σιγμοειδή. Η υπερβολική εφαπτομένη, αντιθέτως, δεν μπορεί να εφαρμοστεί ακολουθώντας αυτήν την λογική. Είναι μαθηματικά αποδεδειγμένο, ότι λόγω του αρνητικού πεδίου τιμών που έχει, οι επιθυμητές τιμές πρέπει να βρίσκονται σε μικρότερο εύρος τιμών από αυτό που βρίσκονται οι ασυμπτωτικές της (Haykin, 1999). Έτσι λοιπόν, θα χρησιμοποιήσουμε τις τιμές από [-1,1] για να δώσουμε τις επιθυμητές τιμές κάθε αμινοξέως. Τα επιθυμητά αποτελέσματα μπορούν να τροποποιηθούν από το αρχείο *threeClasses.txt* (Παράρτημα Ζ).

Επειδή είναι αρκετά δύσκολο να αλλάξουμε τις επιθυμητές τιμές δημιουργώντας ακόμα ένα αρχείο κωδικοποίησης με νέες τιμές εξόδου (λόγω υλοποίησης προγράμματος), θα βάλουμε κάποιους ελέγχους μέσα στην μέθοδο *BiderectionalRecurrentNeuralNetwork.cpp* όπου να μετατρέπουν τις τρέχουσες τιμές από 0 σε -1 και από 1 να το αφήνουν 1. Αυτό θα γίνει χρησιμοποιώντας το *Activation_Function_Type* που είναι μια παράμετρος του parameters.dat, το οποίο καθορίζει τι είδους συνάρτηση ενεργοποίησης θα χρησιμοποιηθεί. Έτσι, ένας έλεγχος θα ελέγχει εάν η συνάρτηση ενεργοποίησης είναι 2, δηλαδή υπερβολική εφαπτομένη,

τότε θα αλλάζει μόνο το κατώτατο όριο, δηλαδή θα μετατρέπει το 0 σε -1. Φυσικά πρέπει να αλλάξει και η συνάρτηση κατωφλίου, και συγκεκριμένα η τιμή που ορίζει το κατώφλι θα είναι το 0 και όχι το 0.5 όπως είχαμε στην περίπτωση της σιγμοειδούς.

4.4.2 Γραμμική Συνάρτηση

Το συγκεκριμένο πρόβλημα της πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, όπως και όλα τα προβλήματα που λύνονται με νευρωνικά δίκτυα, μπορεί να θεωρηθεί σαν πρόβλημα παλινδρόμησης (υποκεφάλαιο 1.2). Σε τέτοιου είδους προβλήματα, καλό είναι να έχουμε μια μη γραμμική συνάρτηση για τους κρυφούς νευρώνες του δικτύου, ενώ για τους νευρώνες του εξωτερικού επιπέδου τοποθετούμε μια γραμμική συνάρτηση ενεργοποίησης (Warren, 1994). Θέλουμε να προσθέσουμε στο σύστημα την δυνατότητα εκπαίδευσης έχοντας μη γραμμική συνάρτηση στους κρυφούς νευρώνες και μια γραμμική συνάρτηση στους εξωτερικούς νευρώνες (σχήμα 4.11).



Σχήμα 4.11: *Τυπικό νευρωνικό δίκτυο όπου η συνάρτηση ενεργοποίησης των κρυφών νευρώνων είναι μη γραμμική, ενώ η συνάρτηση ενεργοποίησης των νευρώνων εξόδου είναι γραμμική (Από Warren, 1994).*

Κανονικά μια γραμμική συνάρτηση που θα μπορούσε να χρησιμοποιηθεί είναι της μορφής που περιγράφει η εξίσωση 4.3. Όμως επειδή τα όρια των τιμών της

συνάρτησης κλείνουν σε άπειρες τιμές θα κανονικοποιούμε απλώς τις τιμές με την εξίσωση 4.4.

$$y = ax + b$$

Εξίσωση 4.3: *Γραμμική συνάρτηση ενεργοποίησης.*

$$f = o \cdot (f_{max} - f_{min}) + f_{min}$$

Εξίσωση 4.4: *Εξίσωση κανονικοποίησης τιμών εξόδου γραμμικής συνάρτησης, όπου fmax, fmin δηλώνουν την μέγιστη και ελάχιστη τιμή, και ο το αποτέλεσμα του νευρώνα.*

Για να επεκτείνουμε το πρόγραμμα ώστε να μπορεί να χρησιμοποιεί την γραμμική συνάρτηση σαν συνάρτηση ενεργοποίησης οποιουδήποτε επιπέδου, πρέπει να εφαρμόσουμε την ίδια τεχνική με βάση το υποκεφάλαιο 4.4.1.

Προσθέσαμε στην μέθοδο *callActivationFunction* την επιλογή για ενεργοποίηση της γραμμικής συνάρτησης μαζί με τους απαραίτητους ελέγχους. Αυτό όμως προϋποθέτει την αλλαγή και άλλων κλάσεων του προγράμματος οι οποίες έχουν να κάνουν με τον υπολογισμό του σφάλματος στους κρυφούς και εξωτερικούς νευρώνες. Προσθέσαμε λοιπόν ξανά στις μεθόδους *calculateOutputLayerError* και *calculateHiddenLayerError* τις εξισώσεις υπολογισμού του σφάλματος χρησιμοποιώντας την παράγωγο της συνάρτησης σε συνδυασμό με τις συγκεκριμένες εξισώσεις για υπολογισμό του σφάλματος. Η συγκεκριμένη συνάρτηση ενεργοποίησης ενεργοποιείται μέσω του αρχείου parameters.dat βάζοντας «3» στην παράμετρο *Activation_Function_Type* αλλά και στην *Error_Function_Type* η οποία υποδηλώνει την συγκεκριμένη παράγωγο που θα λαμβάνει υπόψη το σύστημα στην διαδικασία υπολογισμού του σφάλματος.

Υπάρχει όμως ένα πρόβλημα. Τι γίνεται σε περίπτωση που θέλουμε να χρησιμοποιήσουμε ένα συνδυασμό των συναρτήσεων αυτών, όπως θέλουμε να

κάνουμε και στην συγκεκριμένη περίπτωση; Θα πρέπει να αλλάξουμε μερικά πράγματα στο αρχείο parameters.dat. Θα προσθέσουμε ακόμα δύο παραμέτρους, *Error_Function_Type_Output* και *Activation_Function_Type_Output* καθώς οι δύο προηγούμενες θα μετονομαστούν σε *Error_Function_Type_Hidden* και *Activation_Function_Type_Hidden*. Με τον τρόπο αυτό, μπορούμε να δηλώσουμε διαφορετική συνάρτηση ενεργοποίησης για τους κρυφούς νευρώνες και διαφορετική συνάρτηση ενεργοποίησης για τους νευρώνες του εξωτερικού επιπέδου.

Οι πιο πάνω αλλαγές φαίνονται αναλυτικά στον ψευδοκώδικα του σχήματος 4.12 και 4.13.

```
If (Activation_Function_Type_Hidden = 1)
{
        Κάλεσε σιγμοειδή συνάρτηση ενεργοποίησης
}
else if (Activation_Function_Type_Hidden = 2)
{
        Κάλεσε υπερβολική εφαπτομένη συνάρτηση ενεργοποίησης
}
else if (Activation_Function_Type_Hidden = 3)
{
        Κάλεσε γραμμική συνάρτηση ενεργοποίησης
}
```

Σχήμα 4.12: *Ψευδοκώδικας επιλογής της ανάλογης συνάρτησης ενεργοποίησης για τους νευρώνες των κρυφών επιπέδων.*

```
 If (Error_Function_Type_Output = 1)
{
        Κάλεσε σιγμοειδή συνάρτηση ενεργοποίησης
}
 else if (Error_Function_Type_Output = 2)
{
        Κάλεσε υπερβολική εφαπτομένη συνάρτηση ενεργοποίησης
}
 else if (Error_Function_Type_Output = 3)
{
        Κάλεσε γραμμική συνάρτηση ενεργοποίησης
}
```

Εικόνα 4.13: *Ψευδοκώδικας επιλογής της ανάλογης συνάρτησης ενεργοποίησης για τους νευρώνες του επιπέδου εξόδου.*

4.5 Τυχαιοποίηση συνόλου δεδομένων

Γνωρίζουμε ότι ένας ευρετικός τρόπος βελτιστοποίησης της εκπαίδευσης των νευρωνικών δικτύων είναι να αλλάζουμε την σειρά με την οποία εισέρχονται τα δεδομένα από το σύνολο εκπαίδευσης στο δίκτυο για επεξεργασία. Φυσικά, αυτό λειτουργεί καλύτερα για προβλήματα ταξινόμησης και όχι παλινδρόμησης όπως είναι το συγκεκριμένο πρόβλημα, αλλά είναι κάτι που θεωρητικά μπορεί να δώσει καλύτερα αποτελέσματα και στο τρέχον δίκτυο, επειδή ισοδυναμεί με την εισαγωγή θορύβου στο δίκτυο.

Θέλουμε λοιπόν, να σχεδιάσουμε και να υλοποιήσουμε ένα ξεχωριστό πρόγραμμα το οποίο θα καλείται εξωτερικά με βάση την επιλογή του χρήστη. Το πρόγραμμα αυτό θα είναι υπεύθυνο να «αλλάζει» την θέση των πρωτεϊνών στο σύνολο δεδομένων το οποίο χρησιμοποιεί το σύστημα την συγκεκριμένη στιγμή για να εκπαιδεύεται. Το νέο

σύνολο δεδομένων θα αποτελείται από τις ίδιες ακριβώς πρωτεΐνες, σε διαφορετική όμως θέση στο σύνολο δεδομένων η οποία επιλέγεται τυχαία για κάθε πρωτεΐνη. Το νέο σύνολο δεδομένων που θα σχηματιστεί θα ξαναγραφτεί πίσω στο αρχείο που αποτελεί το σύνολο εκπαίδευσης.

```
Για κάθε πρωτεΐνη x του αρχείου εισόδου
{
        Αποθήκευσε το κωδικό όνομα της  x
        Αποθήκευσε την πρωτοταγή δομή της  x
        Αποθήκευσε την δευτεροταγή δομή της  x
}
Ενόσω υπάρχουν πρωτεΐνες που δεν μπήκαν στο νέο αρχείο
{
        Επέλεξε μια τυχαία πρωτεΐνη απ αυτές
        Γράψε το κωδικό όνομα της  x
        Γράψε την πρωτοταγή δομή της  x
        Γράψε την δευτεροταγή δομή της  x
}
```

Σχήμα 4.14: *Ψευδοκώδικας για το πρόγραμμα «randomizeDataset.pl».*

Το πρόγραμμα αυτό λέγεται «randomizeDataset.pl» και είναι γραμμένο σε γλώσσα PERL (σχήμα 4.14). Όπως αναφέραμε και πιο πάνω, θα δημιουργηθεί μια επιπλέον μεταβλητή στο αρχείο parameters.dat η οποία θα ενεργοποιεί ή όχι το πρόγραμμα αυτό. Η μεταβλητή αυτή θα λέγεται *randomizeDatasetEnable*, και η τιμή 1 δηλώνει ενεργοποίηση της διαδικασίας, 0 απενεργοποίηση της διαδικασίας.

Το όνομα του αρχείου που θα κάνει τυχαιοποίηση το συγκεκριμένο πρόγραμμα, θα πρέπει να δίνεται δια μέσου του προγράμματος «randomizeDataset.pl». Η τυχαιοποίηση του συνόλου αυτού πρέπει να γίνεται για κάθε εποχή, και γι αυτό το

πρόγραμμα αυτό θα καλείται στο τέλος κάθε εποχής του δικτύου και συγκεκριμένα στην μέθοδο *closeFolders* της κλάσης *BidirectionalRecurrentNeuralNetwork.cpp*.

4.6 Ενσωμάτωση Γενετικών Αλγορίθμων

Μια γενική αναφορά στους Γενετικούς Αλγορίθμους (ΓΑ), έγινε στο υποκεφάλαιο 1.3. Αφού λοιπόν οι ΓΑ είναι τόσο χρήσιμοι και μπορούν να λύνουν πολύπλοκα προβλήματα με πολλές παραμέτρους που απαιτούν πολλούς συνδυασμούς, μπορούν να χρησιμοποιηθούν και στο συγκεκριμένο σύστημα. Πιο συγκεκριμένα, θέλουμε να βρούμε ένα συνδυασμό παραμέτρων για το δίκτυο (πίνακας 4.1) ώστε να μπορούμε να βρούμε ένα βέλτιστο ποσοστό απόδοσης. Η αντικειμενική συνάρτηση την οποία προσπαθούμε να βελτιστοποιήσουμε είναι η παράμετρος απόδοσης Q3.

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 θέλουμε να βρούμε ένα συνδυασμό τιμών ώστε να βελτιστοποιούν το ποσοστό Q3.*

Οι γενετικοί αλγόριθμοι σχεδιάστηκαν και υλοποιήθηκαν από τον Αγαθοκλέους (Αγαθοκλέους, 2009) (Παράρτημα Γ), και θα τους χρησιμοποιήσουμε στο υφιστάμενο πρόγραμμα. Χωρίζονται σε δύο κλάσεις (Chromosome.cpp, GeneticAlgorithm.cpp), οι

οποίες υλοποιούν τον αλγόριθμο των γενετικών όπως είδαμε και στο υποκεφάλαιο 1.3, οι οποίες ενσωματώνονται στο πρόγραμμα.

Το χρωμόσωμα αποτελείται από την δυαδική κωδικοποίηση των παραμέτρων του πίνακα 4.1, ανάλογα φυσικά και με το εύρος των τιμών της κάθε μιας. Σε κάθε γενεά, το κάθε χρωμόσωμο, αποτελεί κάποιες τιμές για τις παραμέτρους του δικτύου. Το δίκτυο αποθηκεύει τις τιμές αυτές σαν τις τρέχουσες τιμές που έχει, και αγνοεί τις τιμές που παίρνει από το αρχείο παραμέτρων. Έτσι λοιπόν, εκπαιδεύεται και επαληθεύεται κανονικά, με αυτές τις συγκεκριμένες τιμές των παραμέτρων. Αφού ολοκληρωθεί μια γενεά (αριθμός χρωμοσωμάτων), επιλέγεται το καλύτερο χρωμόσωμο για την συγκεκριμένη γενεά. Μετά ακολουθούν οι διαδικασίες της επιλογής, της διασταύρωσης και μετάλλαξης, και το δίκτυο προχωρεί στην επόμενη γενεά. Το κριτήριο τερματισμού είναι ένας αριθμός γενεών που πρέπει να εκτελεστούν. Ο γενικός αλγόριθμος του προγράμματος αυτού φαίνεται στο σχήμα 4.15.

```
Για κάθε γενεά
{
  Επιλογή χρωμοσωμάτων από προηγούμενη γενεά
  Διασταύρωση χρωμοσωμάτων
  Μετάλλαξη χρωμοσωμάτων
  Για κάθε χρωμόσωμο του πληθυσμού
  {
        Αποκωδικοποίηση χρωμοσώματος
        Αντικατάσταση παραμέτρων με τις αντίστοιχες τιμές
        Για κάθε εποχή
        {
              Εκπαίδευση δικτύου
              Επαλήθευση δικτύου
        }
        Αποθήκευση Q3 ποσοστού για το χρωμόσωμο
  }
  Επιλογή καλύτερου χρωμοσώματος βάση Q3 ποσοστού
}
Επιλογή καλύτερου χρωμοσώματος όλων των γενεών
```

Σχήμα 4.15: *Ο αλγόριθμος που παρουσιάζει τον τρόπο ενσωμάτωσης και λειτουργίας των γενετικών αλγορίθμων στο υφιστάμενο σύστημα.*

Κεφάλαιο 5

Πειράματα και ανάλυση αποτελεσμάτων

5.1 Πειράματα χωρίς την χρήση MSA profiles

5.1.1 Πειράματα και αποτελέσματα με αλλαγή τιμών των παραμέτρων του δικτύου

5.1.2 Χρήση υπερβολικής εφαπτομένης

5.2 Πειράματα με την χρήση MSA profiles

5.2.1 Πειράματα χωρίς την χρήση της τυχαιοποίησης συνόλου εκπαίδευσης

5.2.2 Πειράματα με την χρήση της τυχαιοποίησης συνόλου εκπαίδευσης

5.2.3 Πειράματα με συνδυασμό συναρτήσεων ενεργοποίησης

5.2.4 Αποτελέσματα δικτύου με Segment OVerlap score

5.2.5 Αποτελέσματα των confusion matrices

5.3 Αποτελέσματα γενετικών αλγορίθμων σε συνδυασμό με MSA

5.4 Αποτελέσματα τροποποίησης της αρχιτεκτονικής του συστήματος

5.4.1 Αποτελέσματα τροποποίησης της αρχιτεκτονικής σε συνδυασμό με WTA

5.5 *Φιλτράρισμα* προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών

5.5.1 Αποτελέσματα φιλτραρίσματος προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών με χρήση HMM

5.6 Γενικές παρατηρήσεις και συζήτηση

5.1 Πειράματα χωρίς την χρήση MSA profiles

Στο τρέχον σύστημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, χρειάστηκε να γίνουν κάποιες αλλαγές όπως αναφέρθηκε και στο κεφάλαιο 4, ούτως ώστε να διασφαλιστεί η σωστή λειτουργία του. Αρχικά, τροποποιήσαμε τον όρο της ορμής στις εξισώσεις ενημέρωσης των βαρών του δικτύου, ώστε να μην μηδενίζεται. Επιπλέον αλλάξαμε την υλοποίηση των χρονικών μεταβλητών, και συγκεκριμένα της χρονικής μεταβλητής q^-1.

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. Επειδή υπάρχουν τα αρχικά πειράματα της εργασίας του Αγαθοκλέους (Αγαθοκλέους, 2009), δεν θα συγκρίνουμε εκείνα τα αποτελέσματα με τα αποτελέσματα των νέων πειραμάτων, απλά θα δημιουργήσουμε εκ νέου πειράματα ώστε να μελετήσουμε την μεταβολή της απόδοσης του δικτύου με τις διάφορες αλλαγές στις παραμέτρους.

Τα μικρά σύνολα πρωτεϊνών για εκπαίδευση και για επαλήθευση του τρέχοντος δικτύου, τα οποία χρησιμοποιήθηκαν από τον Αγαθοκλέους για πειράματα για την εκπόνηση της διπλωματικής του, δεν χρησιμοποιήθηκαν. Αυτό γιατί θέλουμε το σύστημα να εκπαιδευτεί με ένα μεγάλο εύρος δεδομένων και πληροφοριών. Ένα πιο αντιπροσωπευτικό υπάρχον σύνολο για το σύστημα είναι ένα σύνολο το οποίο δεν περιέχει ελλιπή στοιχεία, έχει 513 πρωτεΐνες για εκπαίδευση και 99 πρωτεΐνες για επαλήθευση. Τα σύνολα αυτά πάρθηκαν από τα δεδομένα του Αγαθοκλέους (Αγαθοκλέους, 2009), και ονομάζονται *proteinsCleantrain.txt* και *proteinsCleantest.txt* και με αυτά τα σύνολα θα γίνουν τα περισσότερα πειράματα αυτής της εργασίας.

5.1.1 Πειράματα και αποτελέσματα με αλλαγή τιμών των παραμέτρων του δικτύου

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 μικρότερα σύνολα δεδομένων, μια διαδικασία η οποία

ακολουθήθηκε από τον Αγαθοκλέους και τα οποία δεν συμπεριλαμβάνονται στην παρούσα εργασία. Το καλύτερο ποσοστό ακριβείας με βάση τις τιμές αυτές, είναι 63.8% χρησιμοποιώντας την παράμετρο Q3. Πιστεύουμε ότι οι αρχικές αυτές τιμές των παραμέτρων, είναι μια καλή αρχή για να ξεκινήσουμε τα νέα πειράματα, μελετώντας παράλληλα πως επηρεάζουν οι αλλαγές αυτές την απόδοση του δικτύου.

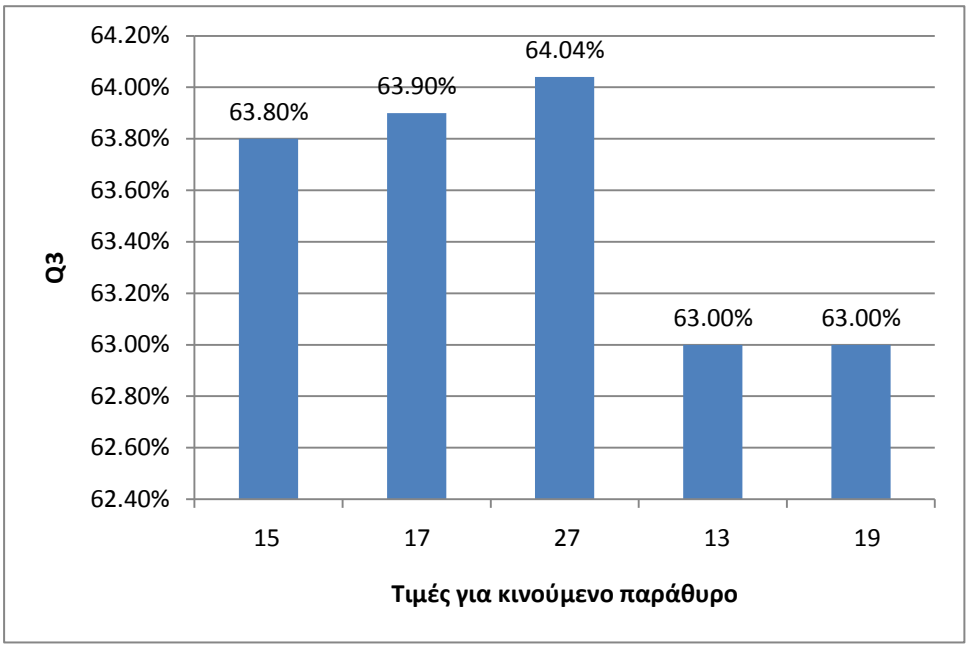
No	ΠΑΡΑΜΕΤΡΟΣ	ΤΙΜΗ
1	Hidden_layer_one_size	12
2	Hidden_layer_two_size	12
3	Hidden_layer_one_of_Backward_size	13
4	Hidden_layer_two_of_Backward_size	12
5	Hidden_layer_one_of_Forward_size	13
6	Hidden_layer_two_of_Forward_size	12
7	Learning_Rate	0.5
8	Momentum	0.1
9	Window_size	15
10	q_minus_1	0.5
11	q_plus_1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

Πίνακας 5.1: *Αρχικές τιμές για τις παραμέτρους του δικτύου (Από Αγαθοκλέους, 2009).*

Για τα διάφορα πειράματα που ακολουθούν, κρατήσαμε τα τρέχοντα βάρη του δικτύου που οδήγησαν τον Αγαθοκλέους (Αγαθοκλέους, 2009) στο αποτέλεσμα 63.83%, σταθερά, και αρχίσαμε μια διαδικασία αλλαγής τιμών στις διάφορες παραμέτρους του πίνακα 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 τα πειράματα τα οποία θα γίνουν, κρατώντας σταθερά τα βάρη τα οποία οδήγησαν το δίκτυο στο καλύτερο του αποτέλεσμα, δεν μπορούν να είναι απόλυτα, επειδή δεν γνωρίζουμε εάν ένας καλύτερος συνδυασμός τιμών στις παραμέτρους μπορεί να δώσει ένα μεγαλύτερο ελάχιστο. Απλά, αρχίζουμε από ένα σημείο αναφοράς, το οποίο είναι

μέχρι τώρα το βέλτιστο, ώστε να δούμε πως επηρεάζουν το δίκτυο οι αλλαγές που εφαρμόσαμε.

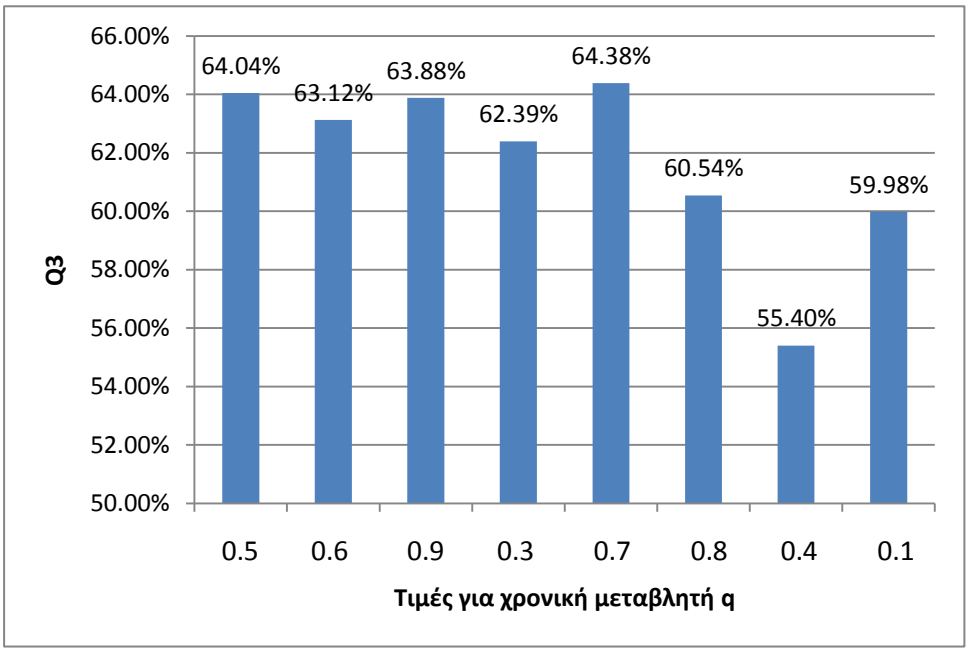
Κατ αρχήν, αλλάξαμε το μέγεθος του κινητού παραθύρου του δικτύου (παράμετρος 9 από πίνακα 5.1), για να μπορέσουμε να δούμε τις διάφορες αλλαγές που συμβαίνουν στο δίκτυο με την επιλογή διάφορων τιμών για την συγκεκριμένη παράμετρο.



Σχήμα 5.1: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας (Q3) σε συνάρτηση με διάφορες τιμές για το κινούμενο παράθυρο.*

Όπως βλέπουμε από την γραφική παράσταση του σχήματος 5.1, εφαρμόζοντας το μέγεθος του κινούμενου παραθύρου σε 27, έχουμε ένα ποσοστό πρόβλεψης του δικτύου σε 64%. Αυτό, δεν συνιστά κάποια σημαντική διαφορά ως προς το προηγούμενο ποσοστό που έδωσαν οι αρχικές παράμετροι (πίνακας 5.1) με το μέγεθος του παραθύρου στο 15, είναι όμως κάποια έστω και ελάχιστη, αλλαγή, γι αυτό στα επόμενα πειράματα θα κρατήσουμε το μέγεθος του παραθύρου στο 27. Αυτό θα γίνεται για κάθε παράμετρο, εφόσον υπάρχει κάποια τιμή η οποία σε συνδυασμό και με τα συγκεκριμένα βάρη τα οποία κρατάμε σταθερά, μπορεί να δώσει καλύτερα ποσοστά απόδοσης από το υφιστάμενο ποσοστό (63.8%).

Στην συνέχεια, αφού αλλάξαμε την υλοποίηση των χρονικών μεταβλητών στο σύστημα (υποκεφάλαιο 4.2), πρέπει να εξετάσουμε τι αποτελέσματα μπορεί το δίκτυο να δώσει εάν εφαρμόσουμε διάφορες τιμές γι' αυτές τις παραμέτρους. Όπως είχαμε δει στο υποκεφάλαιο 4.2, αλλάξαμε την υλοποίηση του q^-1 ώστε να ικανοποιείται η εξίσωση 4.1. Επειδή η αλλαγή του q^-1, εφαρμόζεται αυτόματα από το δίκτυο και δεν αφήνεται στον χρήστη να ορίσει την τιμή της, οι τιμές των παραμέτρων *q_minus_1* και *q_plus_1* (παράμετροι 10 και 11 από πίνακα 5.1) πρέπει να έχουν την ίδια τιμή, με την τιμή που δίνεται από τον χρήστη να αφορά την *q_plus_1* μόνο. Η *q_minus_1*, υπολογίζεται στο σύστημα.

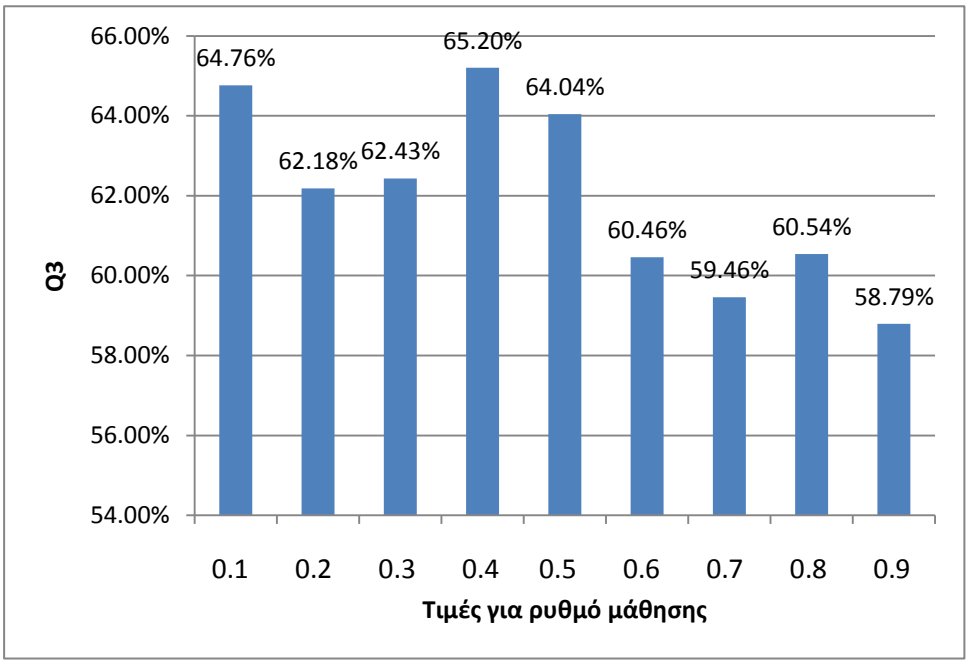


Σχήμα 5.2: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με διάφορες τιμές για την χρονική μεταβλητή q.*

Όπως βλέπουμε από την γραφική παράσταση του σχήματος 5.2, το πιο μεγάλο ποσοστό πρόβλεψης δευτεροταγούς δομής του δικτύου με την παράμετρο Q3, γίνεται όταν η χρονική μεταβλητή q είναι ίση με 0.7, όπου και έχουμε 64.38% ακρίβεια. Συνήθως, όταν η σταθερά αυτή είχε μεγάλες τιμές, παρατηρήσαμε αύξηση του ποσοστού επιτυχίας του δικτύου. Αυτό ίσως να οφείλεται στο ότι μεγαλύτερες τιμές προκαλούν μεγαλύτερες αλλαγές στα δεδομένα της επόμενης χρονικής στιγμής, άρα

τα ίδια δεδομένα δεν διατηρούνται για πολύ με την ίδια τιμή στα δύο δίκτυα αμφίδρομης ανάδρασης.

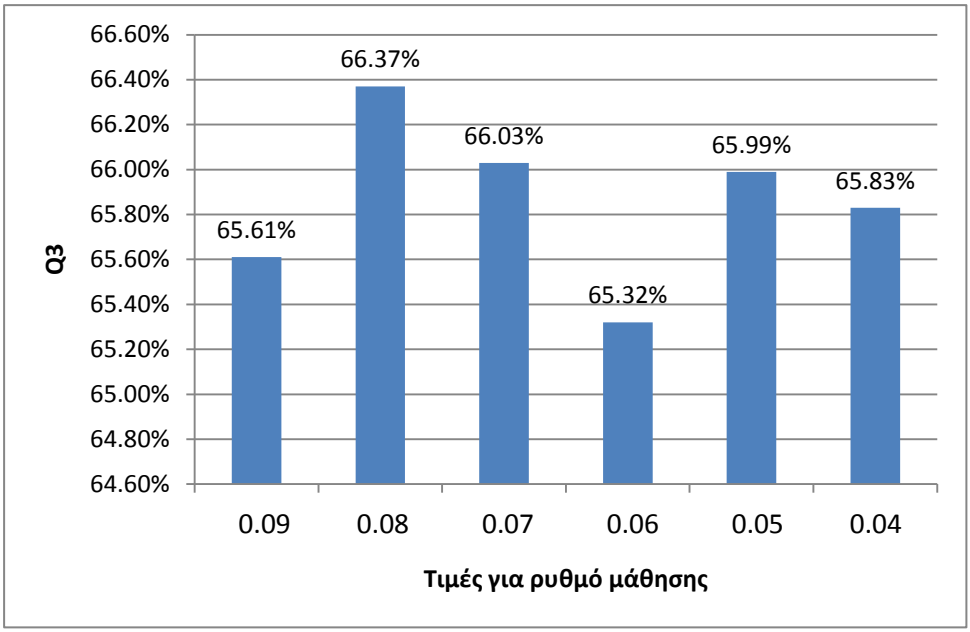
Το επόμενο πείραμα, αφορά την εναλλαγή διάφορων τιμών για την παράμετρο 7 του πίνακα 5.1, που αφορά τον ρυθμό μάθησης του συστήματος. Επειδή ο ρυθμός μάθησης αποτελεί πολύ βασική παράμετρο για το δίκτυο, αφού αφορά την εκπαίδευση του, ξεκινήσαμε εφαρμόζοντας όλες τις πιθανές τιμές για τον ρυθμό μάθησης με ακρίβεια ενός δεκαδικού ψηφίου.



Σχήμα 5.3: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με διάφορες τιμές για τον ρυθμό μάθησης (με ακρίβεια ενός δεκαδικού ψηφίου).*

Απ' ότι βλέπουμε από την γραφική παράσταση του σχήματος 5.3, η απόδοση του συστήματος εφαρμόζοντας στον ρυθμό μάθησης τιμές από 0.6 μέχρι 0.9, είναι χαμηλότερη σε σύγκριση με τα αποτελέσματα της απόδοσης του δικτύου σε τιμές από 0.1 μέχρι 0.5. Το μεγαλύτερο Q3 ποσοστό απόδοσης ήταν 65.20% όταν ο ρυθμός μάθησης ήταν 0.4. Πριν αποφασίσουμε μια τιμή για τον ρυθμό μάθησης για τα υπόλοιπα πειράματα, λόγω της παρατήρησης που έχουμε δει, ότι οι χαμηλότερες τιμές της παραμέτρου αυτής δίνουν καλύτερα αποτελέσματα, θα ελέγξουμε την

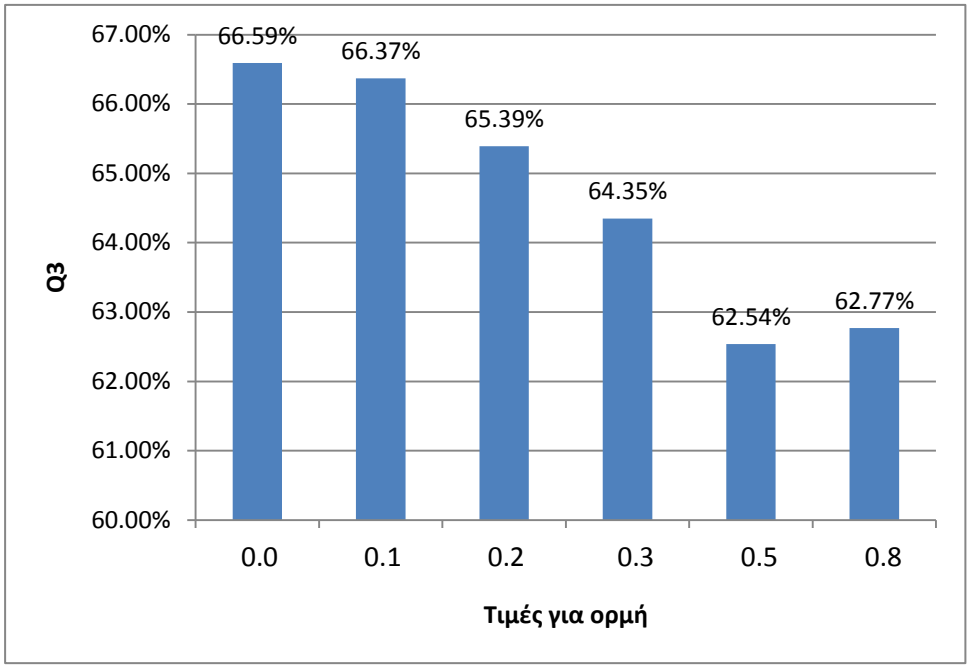
απόδοση του δικτύου για τιμές ρυθμού μάθησης με ακρίβεια δύο δεκαδικών ψηφίων, όπως φαίνεται και στο σχήμα 5.4.



Σχήμα 5.4: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με διάφορες τιμές για τον ρυθμό μάθησης (με ακρίβεια δύο δεκαδικών ψηφίων).*

Από την γραφική παράσταση του σχήματος 5.4 παρατηρούμε ότι το ποσοστό επιτυχίας πρόβλεψης της δευτεροταγούς δομής του δικτύου αυξάνεται ακόμα περισσότερο όταν ο ρυθμός μάθησης ελαττώνεται περισσότερο. Συγκεκριμένα, μπορούμε να δούμε ότι το δίκτυο επιτυγχάνει το μεγαλύτερο μέχρι τώρα ποσοστό απόδοσης (66.37%) χρησιμοποιώντας την παράμετρο Q3, όταν η τιμή του ρυθμού μάθησης είναι 0.08. Θα κρατήσουμε λοιπόν την τιμή αυτή όσον αφορά τον ρυθμό μάθησης, για τα υπόλοιπα πειράματα.

Η επόμενη παράμετρος προς διερεύνηση, είναι η ορμή (παράμετρος 8 του πίνακα 5.1). Να αναφέρουμε ότι σε πειράματα που εκτελέστηκαν με μικρά σύνολα δεδομένων (δεν παρουσιάζονται), παρατηρήθηκε ότι το δίκτυο τις περισσότερες φορές εκπαιδευόταν καλύτερα χωρίς την ορμή. Θέλουμε να δούμε εάν κάτι τέτοιο είναι δυνατό να συμβαίνει και με μεγαλύτερου μεγέθους σύνολα εκπαίδευσης και επαλήθευσης. Έτσι λοιπόν, ορίσαμε διάφορες τιμές για τον όρο της ορμής και κάναμε τα πειράματα βάση των τιμών αυτών. Τα αποτελέσματα φαίνονται στο σχήμα 5.5.



Σχήμα 5.5: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με διάφορες τιμές για όρο της ορμής.*

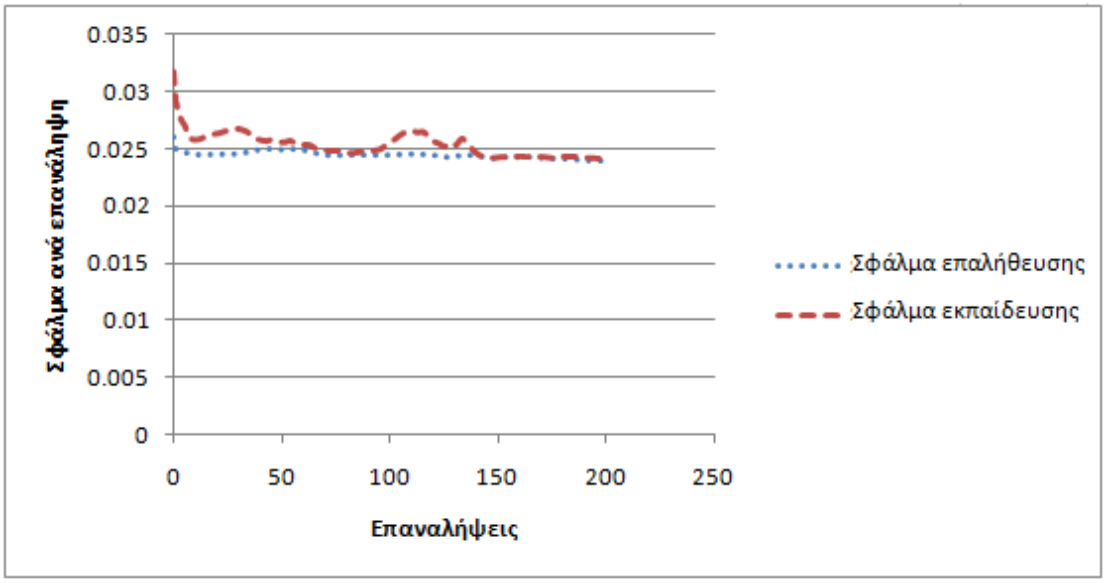
Με βάση τα αποτελέσματα του σχήματος 5.5, παρατηρήσαμε ότι όντως και με μεγαλύτερου μεγέθους σύνολα δεδομένων, χωρίς τον όρο της ορμής, το δίκτυο έχει καλύτερα αποτελέσματα. Το μεγαλύτερο ποσοστό επιτυχίας του δικτύου μέχρι τώρα, ήταν χωρίς την χρήση της ορμής με ένα αποτέλεσμα 66.59%.

Κλείνοντας τον κύκλο των πειραμάτων, μπορούμε να δούμε ότι όντως η μεθοδολογία που ακολουθήθηκε και τα πειράματα που έγιναν για το σύνολο δεδομένων *proteinsCleantrain.txt* και *proteinsCleantest.txt*, έδωσαν κάποιες τοπικά «βέλτιστες» τιμές για τις παραμέτρους του πίνακα 5.1, και ένα ποσοστό πρόβλεψης 66.59%. Οι τιμές αυτές φαίνονται συνοπτικά στον πίνακα 5.2.

ΠΑΡΑΜΕΤΡΟΣ	ΤΙΜΗ
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
Learning_Rate	0.08
Momentum	0.0
Window_size	27
q_minus_1	0.7
q_plus_1	0.7
S	3
Maximum_Iterations	200

Πίνακας 5.2*: Τοπικά βέλτιστες τιμές για τις παραμέτρους του δικτύου, με ποσοστό πρόβλεψης (Q3) 66.59%.*

Στο σχήμα 5.6 φαίνεται η γραφική παράσταση του σφάλματος επαλήθευσης και εκπαίδευσης όταν εφαρμόσουμε τις παραμέτρους του πίνακα 5.2. Όπως βλέπουμε, το σφάλμα επαλήθευσης φτάνει στο 0.0241 την στιγμή που το σφάλμα εκπαίδευσης φτάνει το 0.0239, συνεπώς έχουμε μια πολύ χαμηλή διαφορά στις δύο τιμές.



Σχήμα 5.6: *Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για τον πίνακα 5.2.*

5.1.2 Χρήση Υπερβολικής εφαπτομένης

Γνωρίζοντας την μεγάλη σημασία της υπερβολικής εφαπτομένης για τα προβλήματα παλινδρόμησης (υποκεφάλαιο 4.4.1), εφαρμόσαμε στο δίκτυο την υπερβολική εφαπτομένη σαν συνάρτηση ενεργοποίησης των νευρώνων όλων των επιπέδων. Οι τιμές των διάφορων παραμέτρων που χρησιμοποιήθηκαν για τα πειράματα αυτά είναι οι τιμές των παραμέτρων του πίνακα 5.2.

Δυστυχώς η εφαρμογή της υπερβολικής εφαπτομένης σαν συνάρτηση ενεργοποίησης δεν πέτυχε για το σύστημα κατά την τρέχουσα φάση. Το σύστημα φαίνεται να «κολλά» σε ένα τοπικό ελάχιστο, παρόλο που εφαρμόστηκαν ποικίλα πειράματα με διάφορους συνδυασμούς τιμών για τις παραμέτρους (δεν παρουσιάζονται). Το ποσοστό πρόβλεψης του δικτύου με την εφαρμογή της υπερβολικής εφαπτομένης σαν συνάρτηση ενεργοποίησης, παραμένει σταθερό στο 53.33%.

5.2 Πειράματα με την χρήση MSA profiles

Όπως αναφέραμε και στο υποκεφάλαιο 4.3, εντάξαμε την δυνατότητα στο δίκτυο να εκπαιδεύεται με την χρήση MSA profiles, ώστε να προσθέσουμε κάποιο είδος *εξελικτικής πληροφορίας* των πρωτεϊνών εκπαίδευσης και επαλήθευσης μέσα στο δίκτυο. Με αυτόν τον τρόπο ελπίζουμε ότι η εκπαίδευση θα είναι πιο ακριβής, λόγω της περισσότερης πληροφορίας που θα δίνεται στο δίκτυο. Η θεωρία αυτή εφαρμόστηκε από τους Rost και Sander όπως αναφέρουν και στο άρθρο τους (Rost and Sander, 1994), και συνεχίζει έκτοτε να εφαρμόζεται σε παρόμοια συστήματα, όπως του Baldi και των συνεργατών του (Baldi et al., 1999, 2000).

Αφού εφαρμοστεί η νέα κωδικοποίηση (MSA profiles), θα ελέγξουμε τα σχετικά ποσοστά Q3 απόδοσης του δικτύου χρησιμοποιώντας τα προηγούμενα σύνολα δεδομένων εκπαίδευσης και επαλήθευσης *proteinCleantrain.txt* και *proteinCleantest.txt* αντίστοιχα ώστε να έχουμε ένα μέτρο σύγκρισης της απόδοσης του δικτύου με η χωρίς MSA profiles. Αρχικά χρειάζεται μια μικρή προεργασία των συνόλων αυτών: θα περάσουμε τα σύνολα αυτά από το πρόγραμμα «msaProgram.pl» (υποκεφάλαιο 3.3.1), ώστε να εξακριβώσουμε εάν οι συγκεκριμένες πρωτεΐνες έχουν κάποιο MSA profile και συνεπώς μπορούν να χρησιμοποιηθούν. Μετέπειτα θα δημιουργηθεί και το νέο σύνολο των πρωτεϊνών εκπαίδευσης και επαλήθευσης που μπορούν να χρησιμοποιηθούν με την μέθοδο αυτή.

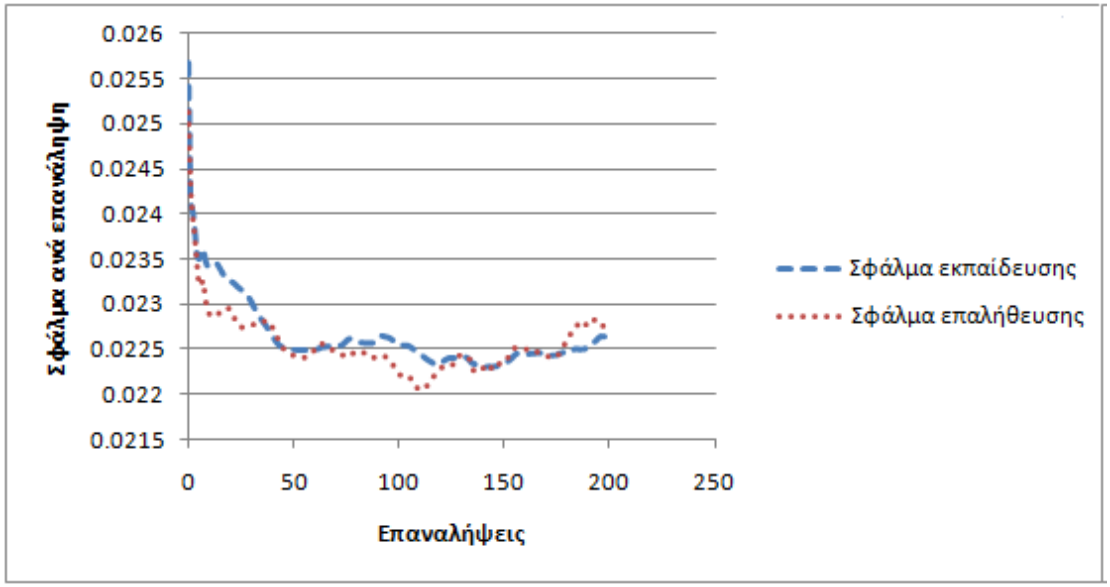
Δυστυχώς, μετά την επεξεργασία τους από το πρόγραμμα «msaProgram.pl» ώστε να δημιουργηθεί το κατάλληλο αρχείο, δεν έχουν βρεθεί όλες οι πρωτεΐνες των συνόλων *proteinCleantrain.txt* και *proteinCleantest.txt* στην βάση HSSP. Σαν αποτέλεσμα εννέα στο σύνολο πρωτεΐνες έχουν αφαιρεθεί από το σύνολο εκπαίδευσης *proteinCleantrain.txt* και δυο πρωτεΐνες έχουν αφαιρεθεί από το σύνολο επαλήθευσης *proteinCleantest.txt*.

Το νέα αυτά αρχεία ονομάστηκαν *msaProteinsTrainBigDataset_afterProcess.txt* με 504 πρωτεΐνες και *msaProteinsTestBigDataset_afterProcess.txt* με 97 πρωτεΐνες για το αρχείο πρωτεϊνών εκπαίδευσης και επαλήθευσης αντίστοιχα, και αυτά τα αρχεία θα χρησιμοποιηθούν όταν το σύστημα θα εκπαιδεύεται με την χρήση MSA profiles.

5.2.1 Πειράματα χωρίς την χρήση της τυχαιοποίησης του συνόλου εκπαίδευσης

Αρχικά, θα δούμε τις διαφορές στα ποσοστά απόδοσης του δικτύου σε σύγκριση με το προηγούμενο καλύτερο ποσοστό που είχαμε για τα συγκεκριμένα σύνολα δεδομένων όταν δεν χρησιμοποιούσαμε MSA profiles (πίνακας 5.2), ώστε να δούμε εάν υπάρχει κάποια σημαντική αλλαγή εκπαιδεύοντας το δίκτυο με τον νέο αυτό τρόπο.

Το σύστημα με την χρήση MSA profiles και χρησιμοποιώντας τις ίδιες τιμές για τις διάφορες παραμέτρους, όμοιες με αυτές που έδωσαν το καλύτερο ποσοστό χωρίς την χρήση MSA profiles (πίνακας 5.2), έδωσε ένα τελικό ποσοστό 70.823%, δηλαδή αύξηση του προηγούμενου ποσοστού κατά 4% χάρη στην χρήση της πολλαπλής στοίχισης αλληλουχιών. Άρα όντως η προσθήκη εξελικτικής πληροφορίας στο δίκτυο, του επιτρέπει να μάθει πιο αποτελεσματικά τα δεδομένα, και μπορεί να γενικεύσει με μεγαλύτερο ποσοστό επιτυχίας.



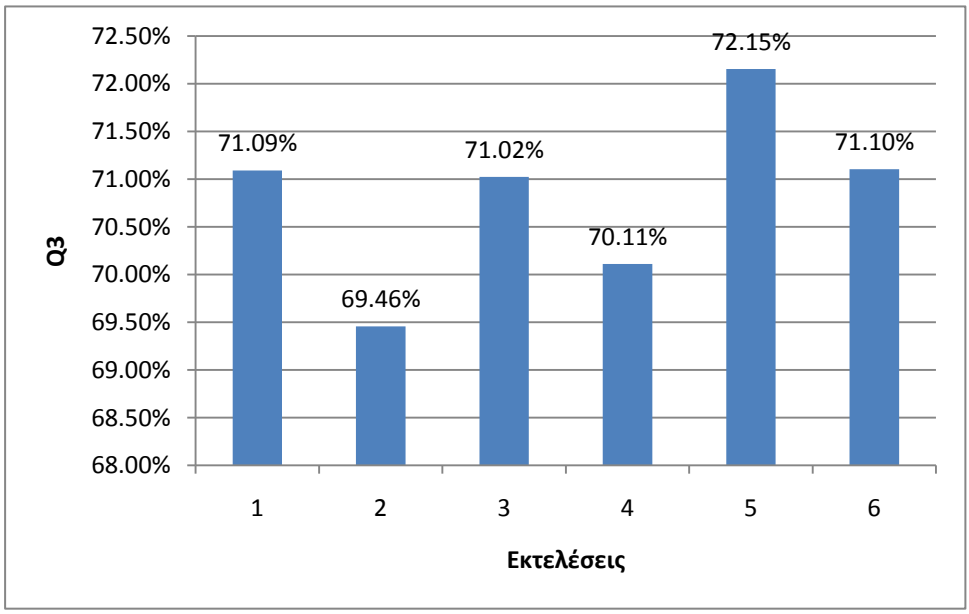
Σχήμα 5.7: *Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για τον πίνακα 5.2 με την χρήση MSA profiles.*

Στο σχήμα 5.7 βλέπουμε την γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για 200 επαναλήψεις. Παρατηρούμε πολλές διακυμάνσεις κατά το σφάλμα εκπαίδευσης, αλλά αυτό ίσως να οφείλεται στο ότι έχουμε πολλά δεδομένα

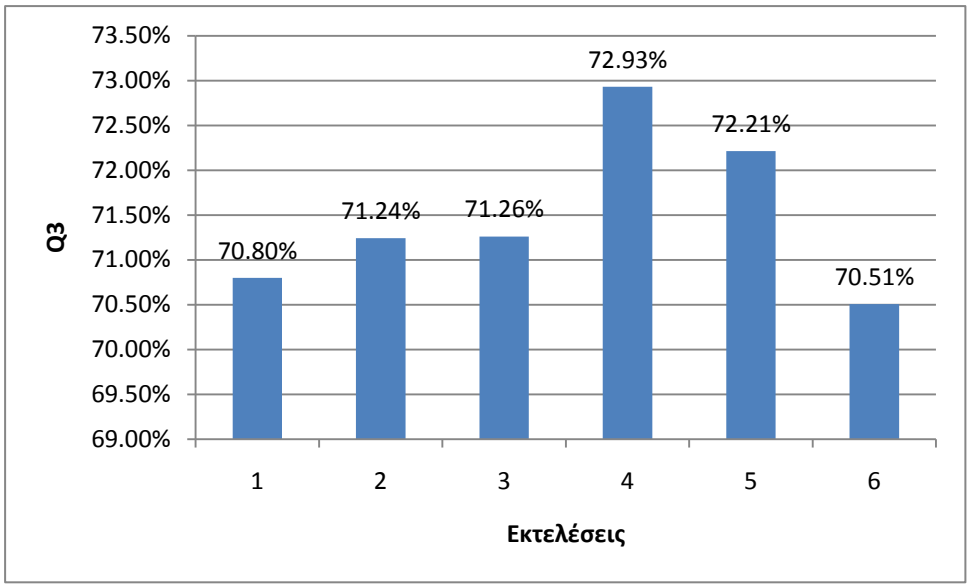
για εκπαίδευση. Παρατηρείται επίσης υπερεκπαίδευση του δικτύου κατά την 140ή επανάληψη, και μια λύση σε αυτό είναι να σταματήσει η εκπαίδευση εκεί (δεν παρουσιάζονται αποτελέσματα).

Μέχρι τώρα, τα αρχικά βάρη του δικτύου ήταν σταθερά. Αυτό όμως, δεν σημαίνει ότι έχουμε το βέλτιστο ελάχιστο, αλλά ένα τοπικά βέλτιστο ελάχιστο. Έτσι, στην συνέχεια, θα εκτελέσουμε κάποια πειράματα, τυχαιοποιώντας τα βάρη του δικτύου ώστε να δούμε μια *γενική* συμπεριφορά του δικτύου, αλλάζοντας παράλληλα κάποιες τιμές των παραμέτρων εφαρμόζοντας την μεθοδολογία που ακολουθήθηκε κατά το υποκεφάλαιο 5.1.1.

Τα πειράματα αυτά χωρίστηκαν σε δύο μέρη, αυτά με την χρήση ορμής, και αυτά χωρίς την χρήση ορμής. Θέλαμε να μελετήσουμε κυρίως τον όρο της ορμής για κάποια τυχαία τιμή (0.5) επειδή ο όρος της ορμής έδωσε παράξενα αποτελέσματα στα προηγούμενα πειράματα χωρίς την χρήση MSA profiles (σχήμα 5.5) και θέλουμε να δούμε κατά πόσο επηρεάζει και αυτήν την περίπτωση.



Σχήμα 5.8: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με έξι εκτελέσεις του δικτύου χωρίς ορμή.*



Σχήμα 5.9: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με έξι εκτελέσεις του δικτύου με ορμή (τιμή ορμής 0.5).*

Τρέξαμε το πρόγραμμα 6 φορές χρησιμοποιώντας ορμή, και 6 φορές χωρίς ορμή. Στο σχήμα 5.8 βλέπουμε τα αποτελέσματα του δικτύου με την χρήση MSA profiles, τυχαιοποίησης βαρών, και χωρίς τον όρο της ορμής. Ο μέσος όρος των πειραμάτων αυτών είναι 70.82%. Αντίθετα, με την χρήση ορμής (σχήμα 5.9), ο μέσος όρος των 6 εκτελέσεων του προγράμματος είναι 71.49%. Με την χρήση της ορμής λοιπόν, βλέπουμε ότι έχουμε καλύτερα αποτελέσματα. Αυτό είναι πολύ παράξενο, γιατί στα πειράματα χωρίς MSA profiles (υποκεφάλαιο 5.1.1), η χρήση της ορμής δεν έδινε υψηλά ποσοστά. Το καλύτερο αποτέλεσμα που έχουμε στην συγκεκριμένη περίπτωση με ορμή είναι 72.93%. Παρόλα αυτά, βλέπουμε ότι και στα δύο πειράματα, έχουμε μεγάλες αποκλίσεις των αποτελεσμάτων της κάθε περίπτωσης και συνεπώς στατιστικά σφάλματα. Γι αυτό, δεν μπορούμε να πούμε με σιγουριά, ότι η χρήση της ορμής βελτιώνει την εκπαίδευση του δικτύου ή ότι η μη χρήση της χειροτερεύει τα αποτελέσματα. Είναι όλα εμπειρικά.

Τα αποτελέσματα που αφορούν πειράματα στις υπόλοιπες παραμέτρους του δικτύου δεν παρουσιάζονται.

5.2.2 Πειράματα με την χρήση της τυχαιοποίησης του συνόλου εκπαίδευσης

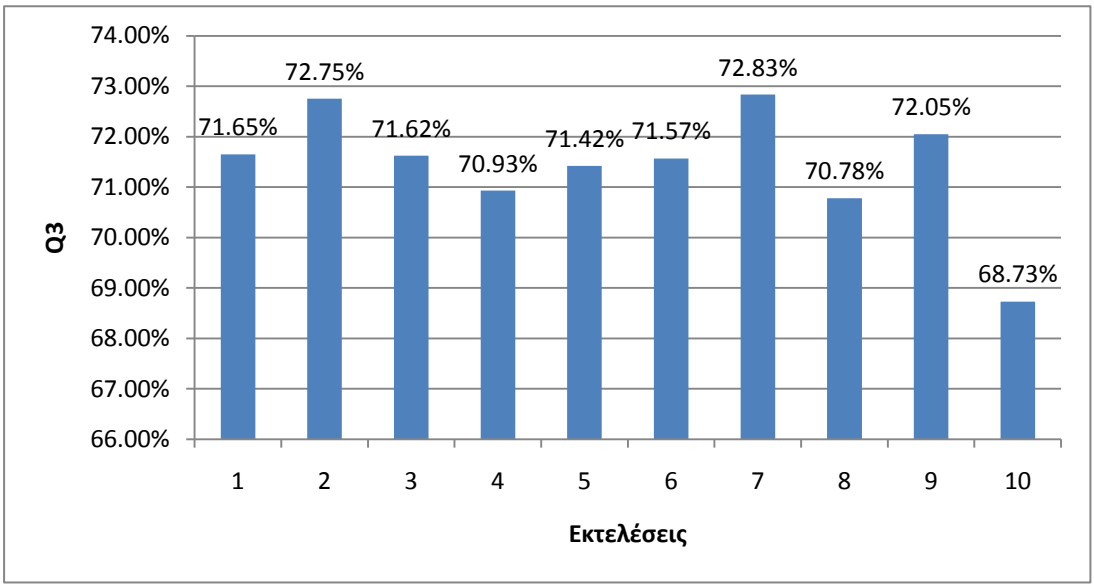
Όπως αναφέρθηκε και στο υποκεφάλαιο 4.5, θέλαμε να δημιουργήσουμε ένα πρόγραμμα που με κάποιο τρόπο θα *ανακατεύει* το σύνολο των πρωτεϊνών εκπαίδευσης του συστήματος σε κάθε επανάληψη. Αυτό γίνεται ώστε να μην υπάρχει μια καθορισμένη σειρά πρωτεϊνών με την οποία να εκπαιδεύεται το σύστημα κάθε εποχή. Η τυχαιοποίηση του συνόλου εκπαίδευσης είναι μια ευρετική μέθοδος για καλυτέρευση της απόδοσης του δικτύου αφού μπορεί να λύσει προβλήματα όπως υπερεκπαίδευση.

Θέλουμε να δούμε πως επηρεάζει αυτή η τυχαιοποίηση του συνόλου εκπαίδευσης των πρωτεϊνών το δίκτυο και γενικότερα την *μάθηση* των δεδομένων. Το πρόγραμμα που το επιτυγχάνει αυτό, ονομάζεται «randomizeDatasets.pl», όπως έχουμε προαναφέρει στο υποκεφάλαιο 4.5 και καλείται μέσω του τρέχοντος συστήματος κάθε εποχή, ώστε να *ανακατεύει* το σύνολο εκπαίδευσης. Οι αρχικές παράμετροι του συστήματος με τις οποίες ξεκινήσαμε τα σχετικά πειράματα φαίνονται στον πίνακα 5.3.

ΠΑΡΑΜΕΤΡΟΣ	ΤΙΜΗ
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
Learning_Rate	0.08
Momentum	0.5
Window_size	15
q_minus_1	0.7
q_plus_1	0.7
S	3
Maximum_Iterations	200

Πίνακας 5.3*: Παράμετροι αρχικοποίησης δικτύου με την χρήση MSA profiles και τυχαιοποίησης συνόλου εκπαίδευσης.*

Σαν πρώτο πείραμα, θα εκτελέσουμε ένα αριθμό από προσομοιώσεις για το τρέχον σύστημα, με την χρήση MSA profiles και τυχαιοποίησης παράλληλα. Επειδή σε κάθε εκτέλεση μπορεί να έχουμε ένα διαφορετικό σύνολο εκπαίδευσης (λόγω της τυχαιοποίησης), θα εκτελέσουμε δέκα πειράματα με βάση τις πιο πάνω παραμέτρους (πίνακας 5.3), ούτως ώστε να δούμε εάν κατά μέσο όρο η προσθήκη της τυχαιοποίησης μπορεί να δώσει καλύτερα ή χειρότερα αποτελέσματα με βάση το πρόβλημα μας.



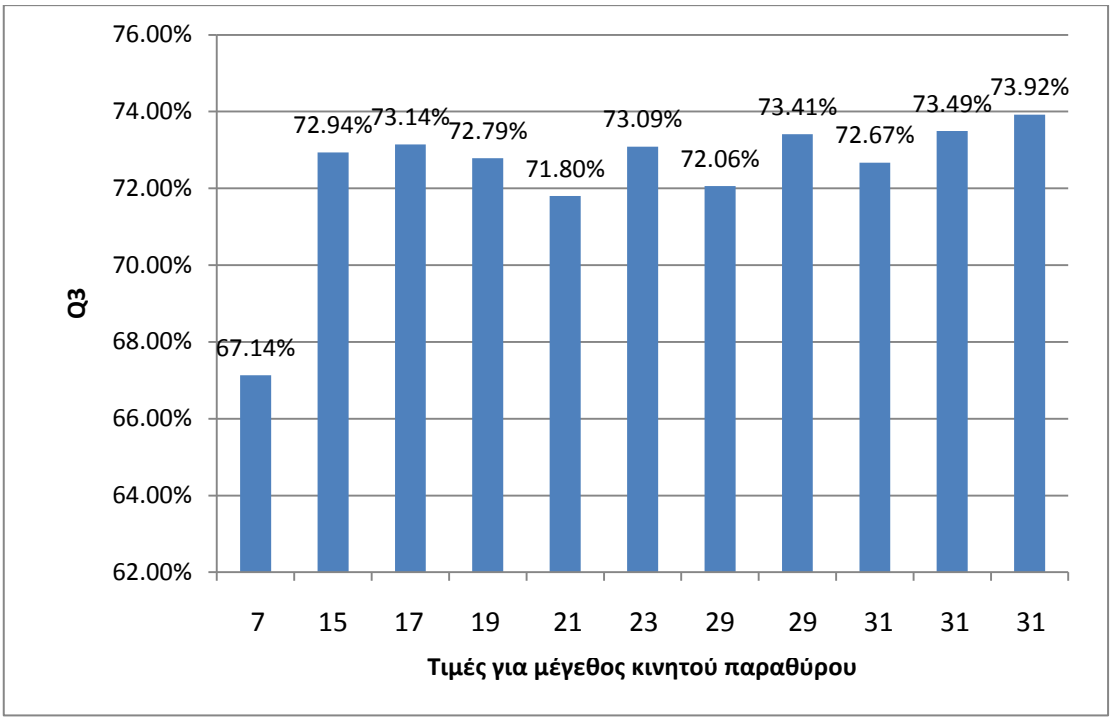
Σχήμα 5.10: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με δέκα εκτελέσεις του δικτύου συμπεριλαμβανομένης της τυχαιοποίησης.*

Με βάση τα αποτελέσματα (σχήμα 5.10), συμπεραίνουμε ότι η τυχαιοποίηση του συνόλου εκπαίδευσης οδηγεί τις πλείστες φορές σε καλύτερα αποτελέσματα σε συνδυασμό και με την ευθυγράμμιση των πρωτεϊνών (MSA profiles). Ο μέσος όρος για δέκα εκτελέσεις είναι 71.43% ένα σαφώς ικανοποιητικό ποσοστό για την χρήση της τυχαιοποίησης. Σημαντικό είναι να παρατηρήσουμε ότι η τυχαιοποίηση του συνόλου, δεν επιφέρει πάντοτε ένα καλύτερο αποτέλεσμα, επειδή η σειρά των πρωτεϊνών που εκπαιδεύονται, παίζει σημαντικότατο ρόλο για το δίκτυο, κάτι το οποίο γίνεται στην περίπτωση που έχουμε τυχαιοποίηση. Και πάλι, εμπειρικά, θεωρούμε ότι η προσθήκη της τυχαιοποίησης επιφέρει καλύτερα αποτελέσματα.

Θέλουμε επίσης να δούμε τα αποτελέσματα του συστήματος ανάλογα με το μέγεθος του κινητού παραθύρου. Δεν χρησιμοποιήσαμε όμως τον πίνακα 5.3 για τις υπόλοιπες τιμές των παραμέτρων αλλά τον πίνακα 5.4, επειδή κατά μέσο όρο έδινε καλύτερα αποτελέσματα στα πιο κάτω πειράματα που θα παρουσιαστούν. Τα αποτελέσματα για τα πειράματα αλλαγής του μεγέθους του κινούμενου παραθύρου με τον πίνακα 5.4, φαίνονται στο σχήμα 5.11.

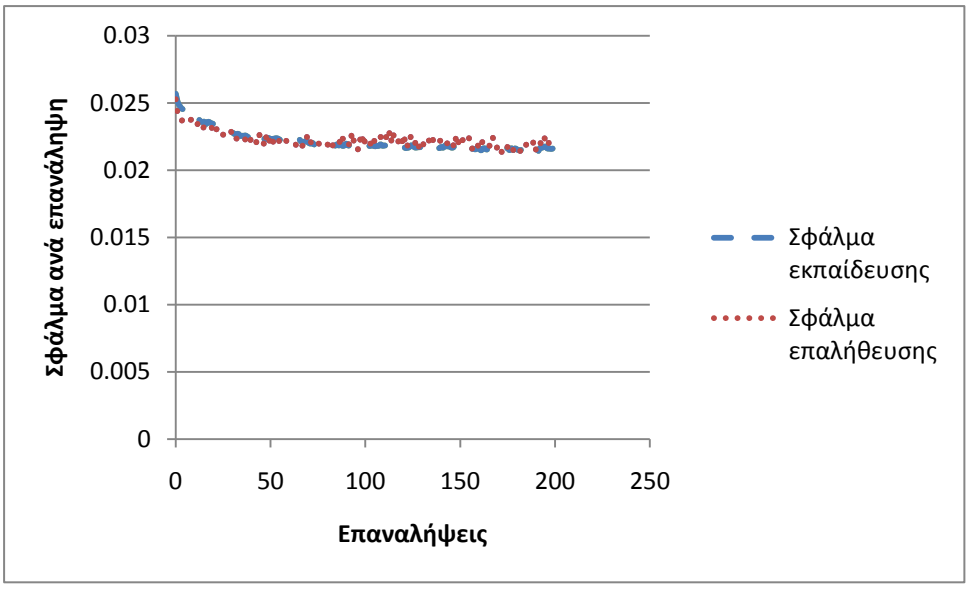
ΠΑΡΑΜΕΤΡΟΣ	ΤΙΜΗ
Hidden_layer_one_size	12
Hidden_layer_two_size	12
Hidden_layer_one_of_Backward_size	13
Hidden_layer_two_of_Backward_size	12
Hidden_layer_one_of_Forward_size	13
Hidden_layer_two_of_Forward_size	12
Learning_Rate	0.09
Momentum	0.5
Window_size	15
q_minus_1	0.6
q_plus_1	0.6
S	3
Maximum_Iterations	200

Πίνακας 5.4*: Εμπειρικές παράμετροι αρχικοποίησης δικτύου με την χρήση MSA profiles και τυχαιοποίησης.*



Σχήμα 5.11: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 σε συνάρτηση με το μέγεθος του κινητού παραθύρου.*

Από το αποτέλεσμα του σχήματος 5.11 παρατηρούμε ότι τα ποσοστά πρόβλεψης με μεγάλο μέγεθος κινητού παραθύρου είναι αρκετά ψηλότερα από τα ποσοστά πρόβλεψης χρησιμοποιώντας μικρότερο μέγεθος κινητού παραθύρου. Ειδικά όταν το μέγεθος του παραθύρου είναι 7, το αποτέλεσμα πέφτει στην χαμηλότερη του τιμή, 67.14%, ενώ όταν το μέγεθος του είναι 31 φτάνει τις τιμές 73.49% και 73.92%. Θέλουμε να ελέγξουμε όμως την γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης, ώστε να δούμε εάν ακόμη τα προβλήματα της υπερεκπαίδευσης που συναντήσαμε στο υποκεφάλαιο 5.2.1 συνεχίζουν να υφίστανται.



Σχήμα 5.12: *Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για το ποσοστό 73.92% με την χρήση της τυχαιοποίησης.*

Απ' ότι φαίνεται και από το σχήμα 5.12, η τυχαιοποίηση του συνόλου εκπαίδευσης, μπορεί να εμποδίσει το δίκτυο να οδηγηθεί σε υπερεκπαίδευση, κάτι που παρατηρήσαμε στο υποκεφάλαιο 5.2.1. Ελέγξαμε το γεγονός αυτό και σε μερικά από τα υπόλοιπα αποτελέσματα του σχήματος 5.11 (δεν παρουσιάζονται) και όντως δεν έχουμε σημεία υπερεκπαίδευσης.

Τέλος, θέλουμε να δούμε πως επηρεάζει η αρχιτεκτονική του δικτύου την εκπαίδευση του καθώς και τα αποτελέσματά του, κάτι το οποίο δεν εξετάστηκε σε προηγούμενα πειράματα. Με τον όρο αρχιτεκτονική του δικτύου, εννοούμε τον αριθμό των νευρώνων σε κάθε επίπεδο.

Q3	Architecture layer by layer	Learning Rate	Momentum	Window Size	q/q^-1
73.05%	18 - 18 - 19 - 18 - 19 - 18	0.09	0.5	29	0.6
73.65%	18 - 18 - 19 - 18 - 19 - 18	0.09	0.5	29	0.6
72.15%	18 - 18 - 19 - 18 - 19 - 18	0.09	0.5	29	0.6
70.81%	16 - 16 - 17 - 16 - 17 - 16	0.09	0.5	29	0.6
73.85%	16 - 16 - 17 - 16 - 17 - 16	0.09	0.5	29	0.9
73.67%	14 - 14 - 15 - 14 - 15 - 14	0.09	0.5	29	0.6
71.35%	10 - 10 - 11 - 10 - 11 - 10	0.09	0.5	29	0.6

Πίνακας 5.5*: Στην δεύτερη στήλη φαίνονται οι τιμές για τον αριθμό των νευρώνων σε κάθε επίπεδο, ενώ οι υπόλοιπες στήλες έχουν τις τιμές των διάφορων παραμέτρων.*

Στον πίνακα 5.5 βλέπουμε τι γίνεται εάν αλλάξουμε την αρχιτεκτονική του δικτύου εφαρμόζοντας διαφορετικό αριθμό νευρώνων για τα διάφορα επίπεδα. Το μέγεθος του κινητού παραθύρου, καθώς και οι χρονικές μεταβλητές, ο ρυθμός μάθησης, και ο όρος της ορμής παραμένουν κατά πλείστον σταθερά για να καταλάβουμε πως επηρεάζει η αρχιτεκτονική του δικτύου την εκπαίδευση του. Όπως βλέπουμε τα αποτελέσματα του συστήματος είναι πάρα πολύ καλά όσο η αρχιτεκτονική του δικτύου αυξάνεται σε μέγεθος, με αποτελέσματα μέχρι και 73.85%.

5.2.3 Πειράματα με συνδυασμό συναρτήσεων ενεργοποίησης

Εφαρμόζοντας την κωδικοποίηση με την χρήση των MSA profiles και της τυχαιοποίησης θέλουμε να δούμε εάν η χρήση της συνάρτησης της υπερβολικής εφαπτομένης μπορεί να «ξεπεράσει» το τοπικό ελάχιστο στο οποίο είχε κολλήσει πριν (υποκεφάλαιο 5.1.2). Η αλλαγή που θα κάνουμε σε αυτήν την περίπτωση, είναι η χρήση κάποιας γραμμικής συνάρτησης στους νευρώνες εξόδου, επειδή είναι γνωστό ότι για τα προβλήματα παλινδρόμησης, ο συνδυασμός μη γραμμικής συνάρτησης στους κρυφούς νευρώνες με μια γραμμική συνάρτηση στους νευρώνες εξόδου οδηγεί

σε πολύ καλά αποτελέσματα (υποκεφάλαιο 4.4.2). Οι τιμές των παραμέτρων με τις οποίες εκπαιδεύσαμε το δίκτυο, ήταν εμπειρικά οι καλύτερες τιμές, που έδωσαν τα καλύτερα ποσοστά πρόβλεψης για τα πιο κάτω πειράματα και φαίνονται στον πίνακα 5.6.

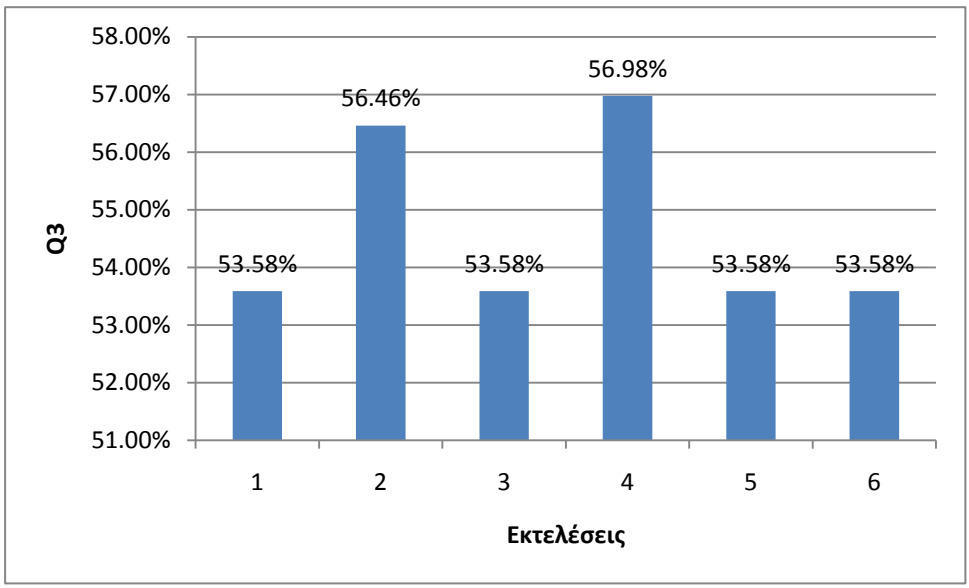
Σημαντικό είναι να αναφέρουμε ότι στα πιο κάτω πειράματα χρησιμοποιήσαμε την αλλαγή της μείωσης του ρυθμού μάθησης που έγινε από τον διδακτορικό φοιτητή Αντώνη Αντωνίου.

ΠΑΡΑΜΕΤΡΟΣ	ΤΙΜΗ
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
Learning_Rate	0.09
Momentum	0.5
Window_size	15
q_minus_1	0.7
q_plus_1	0.7
S	3
Maximum_Iterations	200

Πίνακας 5.6*: Εμπειρικοί παράμετροι αρχικοποίησης δικτύου για τα πειράματα του παρόντος υποκεφαλαίου.*

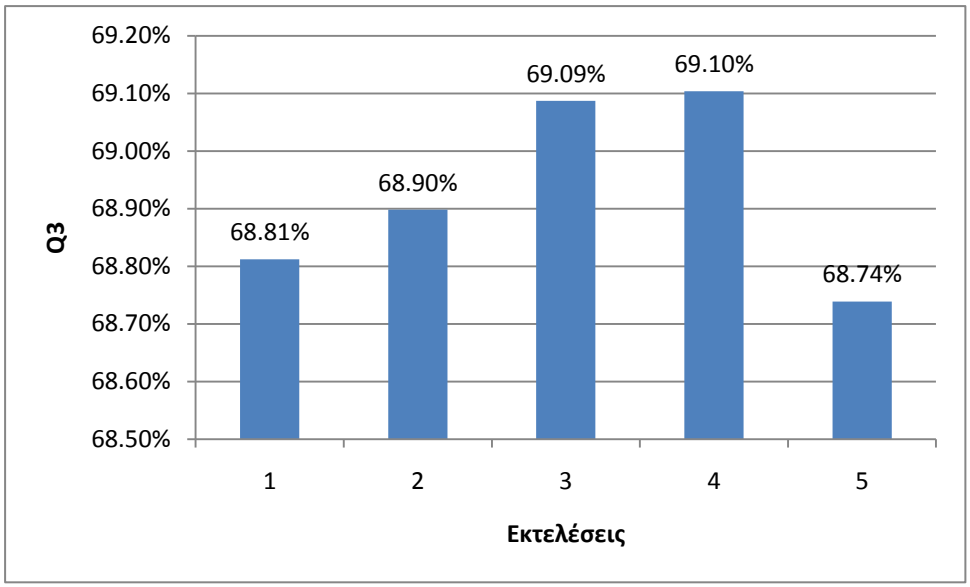
Κατ’ αρχήν δοκιμάσαμε τον συνδυασμό της υπερβολικής εφαπτομένης σαν συνάρτηση ενεργοποίησης των κρυφών νευρώνων σε συνδυασμό με την γραμμική συνάρτηση στο επίπεδο εξόδου (υποκεφάλαιο 4.4.2). Εκτελέσαμε το πρόγραμμα έξι φορές και τα αποτελέσματα παρουσιάζονται στο σχήμα 5.13. Όπως βλέπουμε, δεν υπάρχει σχεδόν καμία βελτίωση του ποσοστού πρόβλεψης όταν χρησιμοποιούμε την υπερβολική εφαπτομένη σε συνδυασμό με την γραμμική συνάρτηση σε σύγκριση με

το καλύτερο ποσοστό πρόβλεψης που αποκτήθηκε μέχρι τώρα (73.92%, σχήμα 5.11). Προφανώς το πρόγραμμα, ακόμη «κολλάει» σε τοπικό ελάχιστο όπως γινόταν και στο υποκεφάλαιο 5.1.2 παρόλο που εκεί δεν εφαρμοζόταν η τυχαιοποίηση του συνόλου εκπαίδευσης αλλά και η κωδικοποίηση με MSA profiles.



Σχήμα 5.13: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας για έξι εκτελέσεις με την χρήση υπερβολικής εφαπτομένης στα κρυφά επίπεδα και γραμμικής συνάρτησης στο επίπεδο εξόδου.*

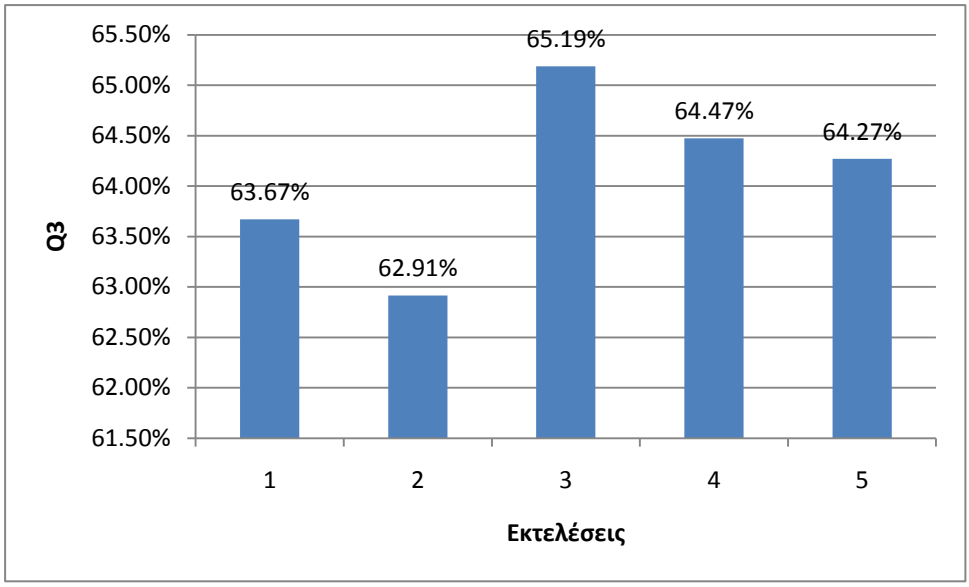
Θέλουμε να κάνουμε περισσότερες εκτελέσεις σε αυτό το στάδιο, αλλά, κρατώντας την γραμμική συνάρτηση στο επίπεδο εξόδου και αλλάζοντας την υπερβολική εφαπτομένη σε σιγμοειδή συνάρτηση ενεργοποίησης. Μπορεί όντως η υπερβολική εφαπτομένη να οδηγεί το δίκτυο σε τοπικό ελάχιστο, γι αυτό δοκιμάσαμε στην θέση της να εφαρμόσουμε την σιγμοειδή συνάρτηση.



Σχήμα 5.14: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας για έξι εκτελέσεις με την χρήση σιγμοειδούς συνάρτησης στα κρυφά επίπεδα και γραμμικής συνάρτησης στο επίπεδο εξόδου.*

Τα αποτελέσματα του πειράματος αυτού φαίνονται στο σχήμα 5.14. Παρατηρούμε προφανώς καλύτερα αποτελέσματα όταν η συνάρτηση ενεργοποίησης των κρυφών νευρώνων είναι η σιγμοειδής παρά η υπερβολική εφαπτομένη. Παρόλα αυτά, δεν έχουμε τα αποτελέσματα που θα περιμέναμε, αφού σε γενικές γραμμές η χρήση της γραμμικής συνάρτησης στο επίπεδο εξόδου, δεν αυξάνει τα ποσοστά περισσότερο από όταν ήταν η σιγμοειδής συνάρτηση στο επίπεδο εξόδου.

Θέλουμε τέλος να ελέγξουμε τι γίνεται όταν αλλάξουμε το όλο σκεπτικό, και αφαιρέσουμε την γραμμική συνάρτηση από το επίπεδο εξόδου. Στην θέση της θα μπει η σιγμοειδής συνάρτηση, ενώ για τους κρυφούς νευρώνες θα μπει η υπερβολική εφαπτομένη.



Σχήμα 5.15: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας για έξι εκτελέσεις με την χρήση υπερβολικής εφαπτομένης στα κρυφά επίπεδα και της σιγμοειδούς συνάρτησης στο επίπεδο εξόδου.*

Παρατηρούμε από το σχήμα 5.15 ότι και πάλι τα αποτελέσματα δεν ξεπερνούν το 70%. Άρα ούτε και αυτή η περίπτωση μπορεί να μας δώσει τα αποτελέσματα που περιμέναμε. Φυσικά σε σύγκριση με τα προηγούμενα αποτελέσματα, (σχήμα 5.14), στις εκτελέσεις που χρησιμοποιήσαμε γραμμική συνάρτηση στο επίπεδο εξόδου και σιγμοειδή στα κρυφά επίπεδα, τα ποσοστά απόδοσης είναι καλύτερα από τα τρέχοντα, που απλώς «αναμιγνύουν» δυο μη γραμμικές συναρτήσεις ενεργοποίησης για τα κρυφά επίπεδα και το επίπεδο εξόδου.

5.2.4 Αποτελέσματα δικτύου με Segment Overlap score

Αφού υλοποιήσαμε ένα πρόγραμμα το οποίο να παίρνει το αρχείο εξόδου – αρχείο πρόβλεψης που έδωσε το νευρωνικό δίκτυο για τις πρωτεΐνες επαλήθευσης, θέλουμε να δούμε τι γίνεται με το SOV σκορ (Zemla et al., 1999), αφού αποτελεί πολύ χρήσιμη μετρική για την αξιοπιστία των αποτελεσμάτων του δικτύου (υποκεφάλαιο 3.4.2).

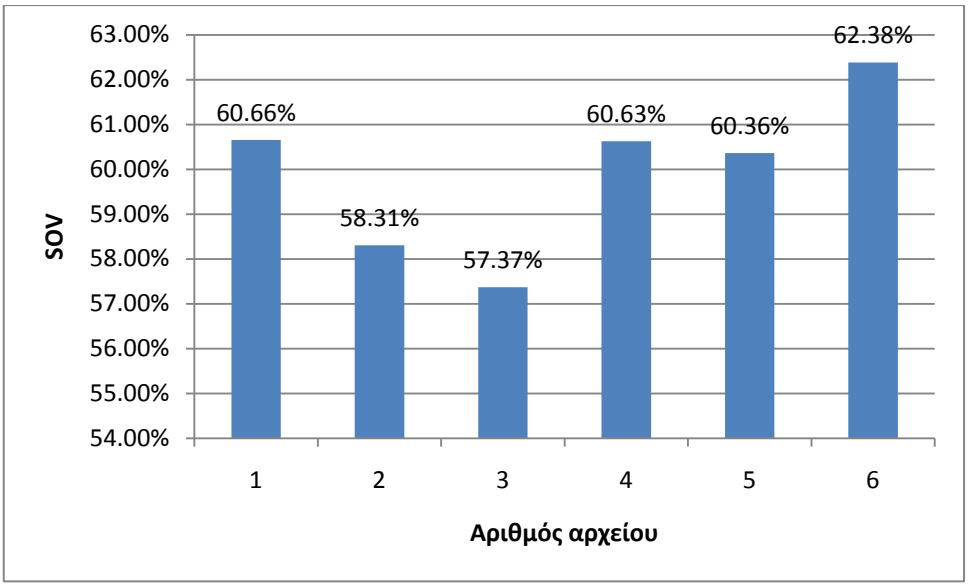
Θα χρησιμοποιήσουμε κάποια ενδεικτικά αρχεία εξόδου που πήραμε από τα μέχρι τώρα πειράματα, για να μελετήσουμε το SOV σκορ του δικτύου, και γενικότερα θα χρησιμοποιήσουμε τα αρχεία που έδωσαν τα καλύτερα αποτελέσματα πρόβλεψης του δικτύου. Τα αρχεία αυτά θα παρθούν από τα αποτελέσματα που έχουμε με την χρήση MSA profiles.

Ο πίνακας με τα αρχεία που θα χρησιμοποιήσουμε φαίνεται στον πίνακα 5.7, όπου φαίνεται το μέγεθος της αρχιτεκτονικής του δικτύου, το μέγεθος του παραθύρου που χρησιμοποιήθηκε και εάν χρησιμοποιήθηκε ή όχι η τυχαιοποίηση του συνόλου εκπαίδευσης στο συγκεκριμένο πείραμα.

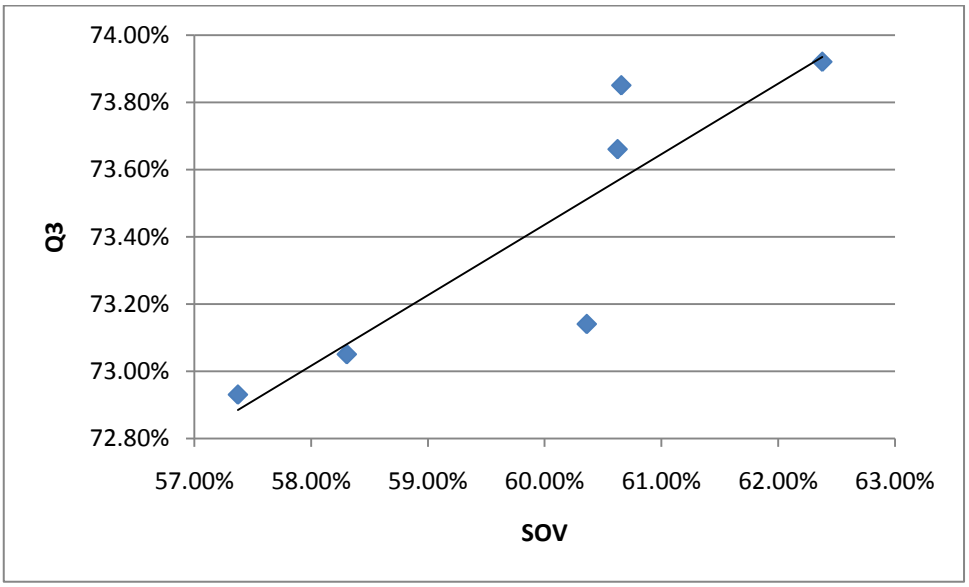
No	Q3 Score	Architecture	Window Size	Randomize
1	73.85%	16 - 16 - 17 - 16 - 17 - 16	29	NAI
2	73.05%	18 - 18 - 19 - 18 - 19 - 18	29	NAI
3	72.93%	12 - 12 - 13 - 12 - 13 - 12	27	OXI
4	73.66%	14 - 14 - 15 - 14 - 15 – 14	29	NAI
5	73.14%	12 - 12 - 13 - 12 - 13 - 12	17	NAI
6	73.92%	12 - 12 - 13 - 12 - 13 - 12	31	NAI

Πίνακας 5.7*: Πίνακας με τα διάφορα αρχεία εξόδου που χρησιμοποιήθηκαν για την μέτρηση του SOV σκορ και τις παραμέτρους που οδήγησαν στο Q3 σκορ (δεύτερη στήλη).*

Τα αποτελέσματα του σχήματος 5.16 δείχνουν για κάθε αρχείο αποτελεσμάτων πρόβλεψης που χρησιμοποιήθηκε (πίνακας 5.7), το SOV σκορ (εξίσωση 3.2). Από ότι παρατηρούμε σε γενικές γραμμές, το SOV σκορ, επί το πλείστον αυξάνεται ανάλογα με την τιμή του Q3 σκορ (σχήμα 5.17).



Σχήμα 5.16: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας SOV για τα έξι αρχεία.*



Σχήμα 5.17: *Γραφική παράσταση που παρουσιάζει την αναλογία των ποσοστών Q3 και SOV για τα έξι αρχεία.*

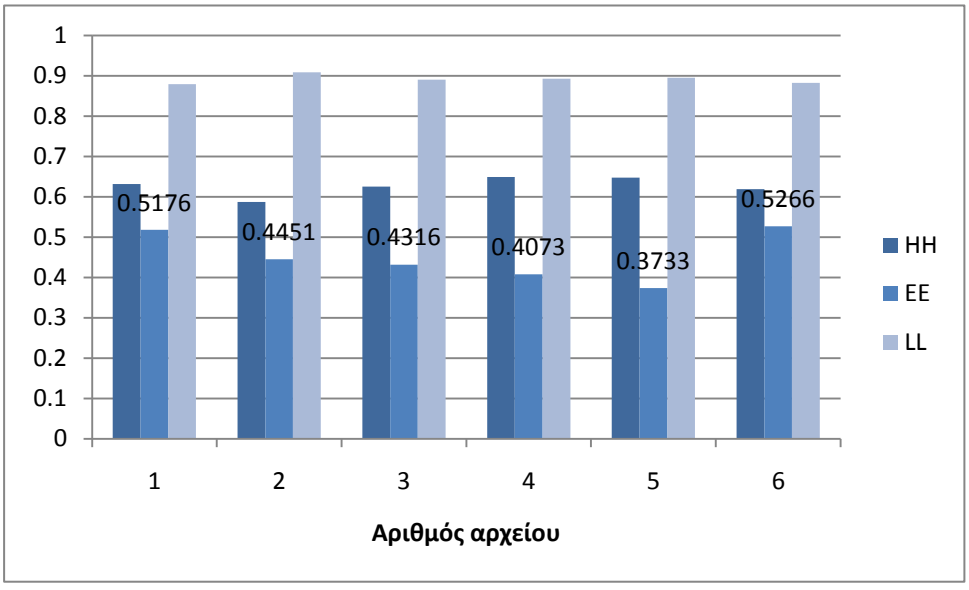
Το μεγαλύτερο SOV σκορ είναι 62.38%, κάτι το οποίο πάρθηκε από το μεγαλύτερο Q3 ποσοστό πρόβλεψης που είχαμε μέχρι τώρα. Βλέπουμε ότι αναλόγως το SOV σκορ δεν

είναι μια καλή τιμή όπως θα αναμέναμε, διότι δεν ισοβαθμεί περίπου με το Q3 σκορ, αφού κυμαίνεται σε τιμές 57% – 62%. Για να εξακριβώσουμε τι γίνεται θα μελετήσουμε τα confusion matrices της κάθε πρόβλεψης (υποκεφάλαιο 3.4.1).

5.2.5 Αποτελέσματα των confusion matrices

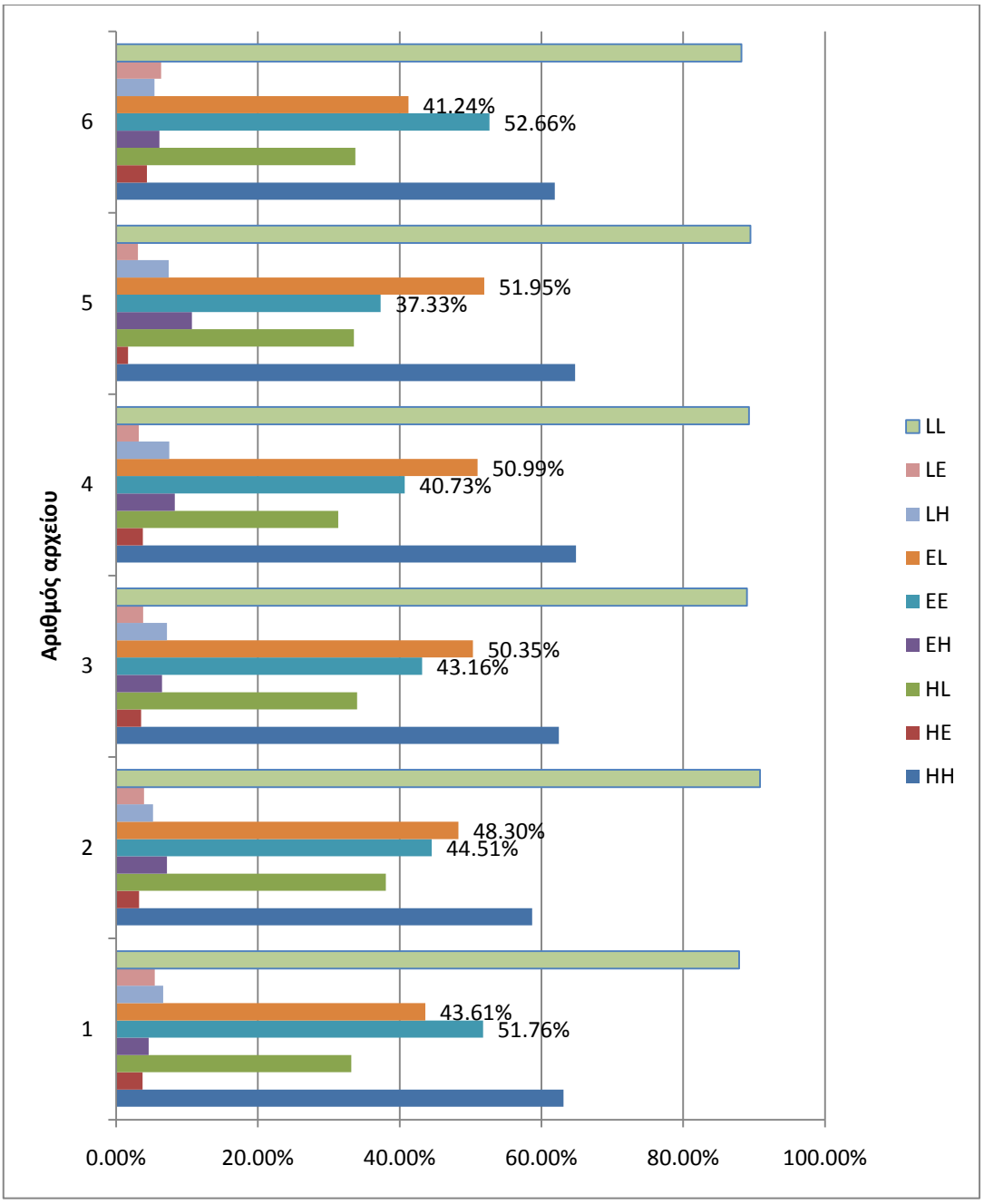
Υπολογίσαμε τα confusion matrices των έξι αρχείων του πίνακα 5.7, όπως το ορίσαμε στο υποκεφάλαιο 3.4.1, ώστε να εξακριβώσουμε για πιο λόγο τα αποτελέσματα του σχήματος 5.16 είναι χαμηλά.

Όπως βλέπουμε από την γραφική παράσταση του σχήματος 5.18, η ομάδα των Strands (E), δεν προβλέπεται σωστά σε μεγάλο βαθμό. Βλέπουμε ότι λιγότερο από 0.5 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 παρουσιάζει πρόβλημα.



Σχήμα 5.18: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα.*

Αφού είδαμε τα αποτελέσματα των confusion matrices 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), είναι πολύ χαμηλός κατά μέσο όρο. Προφανώς, το πρόβλημα αυτό, ίσως να οφείλεται σε μια «σύγχυση» κάποιων ομάδων μεταξύ τους. Για να το δούμε αυτό, πρέπει να δούμε τα αποτελέσματα των confusion matrices για τα υπόλοιπα στοιχεία, δηλαδή EH, EL και άλλα (σχήμα 5.19).



Σχήμα 5.19: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα καθώς και τις λανθασμένες προβλέψεις HE, HL, EH, EL, LE, LH.*

Βλέπουμε στο σχήμα 5.19 πιο καθαρά, τα κατά μέσο όρο confusion matrices του κάθε ενός από τα 6 σύνολα που χρησιμοποιήσαμε ως ενδεικτικά αρχεία. 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).

Παρατηρώντας τα γενικά αποτελέσματα που έχουμε πάρει με την μέθοδο SOV (σχήμα 5.16) καθώς και τα ανάλογα confusion matrices (σχήμα 5.19), βλέπουμε ότι το δίκτυο στην τρέχουσα του μορφή έχει ένα πολύ σημαντικό πρόβλημα, το οποίο αναφέραμε και πιο πάνω. Δεν μπορεί να γίνει σωστά η πρόβλεψη της δευτεροταγούς ομάδας των (E), και από ότι φαίνεται η ομάδα αυτή συγχέεται με την ομάδα των (L).

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.

Για να εξακριβώσουμε κατά πόσο το πρόβλημα της χαμηλής πρόβλεψης ακρίβειας των (E), οφείλεται στην κωδικοποίηση της εξόδου, θα εφαρμόσουμε την μέθοδο *winner takes all* (WTA) στο επίπεδο εξόδου, υλοποιημένη από τον διδακτορικό φοιτητή Αντωνίου. Με αυτήν την μέθοδο, περιορίζεται ο αριθμός των πιθανών συνδυασμών που προκύπτουν από την τρέχουσα σχεδίαση του επιπέδου εξόδου, και συγκεκριμένα από οχτώ έχουμε τρείς συνδυασμούς, ένα να κωδικοποιεί κάθε δευτεροταγή δομή. Η μέθοδος αυτή δεν χρησιμοποιεί την συνάρτηση κατωφλίου για

να δώσει τις τιμές των νευρώνων εξόδου, αλλά, δίνει την τιμή 1 σε όποιον από τους τρεις νευρώνες έχει την μεγαλύτερη τιμή (winner).

Αφού τροποποιήσαμε το σύστημα ώστε να μπορεί να λειτουργήσει χρησιμοποιώντας την μέθοδο αυτή, θα εκτελέσουμε κάποια πειράματα χρησιμοποιώντας τις τιμές του πίνακα 5.7 για να δούμε εάν όντως έχουμε καλύτερα αποτελέσματα της παραμέτρου SOV, και συνεπώς εάν επαληθεύονται οι πιο πάνω θεωρίες.

Με την εκτέλεση των πειραμάτων (δεν δείχνονται αποτελέσματα), τα αποτελέσματα δεν έδειξαν καμία βελτίωση του SOV σκορ ως προς τις προηγούμενες του τιμές χωρίς την εφαρμογή του *WTA* (σχήμα 5.16). Το ίδιο ισχύει και για τα confusion matrices. Τα αποτελέσματα των confusion matrices με *WTA* δεν έχουν καμία διαφορά με τα αποτελέσματα χωρίς την χρήση *WTA* (σχήμα 5.19).

5.3 Αποτελέσματα γενετικών αλγόριθμων σε συνδυασμό με το MSA

Όπως έχουμε προαναφέρει στο υποκεφάλαιο 4.6, οι γενετικοί αλγόριθμοι υλοποιήθηκαν από τον Αγαθοκλέους (Αγαθοκλέους, 2009), ο οποίος τους εφάρμοσε σε διάφορα πειράματα του συστήματος, χωρίς την χρήση όμως των MSA profiles. Τα αποτελέσματα που πήρε με την χρήση των γενετικών αλγορίθμων έδωσαν ένα συνολικό ποσοστό 65.4% για την παράμετρο Q3, την στιγμή που το καλύτερο ποσοστό ήταν 63.83%. Είχε δηλαδή μια αύξηση περίπου 1%, με την χρήση των γενετικών αλγορίθμων. Όντως λοιπόν οι γενετικοί αλγόριθμοι μπορούν να βελτιστοποιήσουν τις τιμές των παραμέτρων του συστήματος, με αποτέλεσμα να δίνουν ένα καλύτερο ποσοστό ακρίβειας πρόβλεψης της δευτεροταγούς δομής.

Στην συγκεκριμένη διπλωματική εργασία, θέλουμε επίσης να δούμε εάν η χρήση των γενετικών αλγορίθμων σε συνδυασμό και με το MSA, μπορεί και πάλι να μας δώσει ένα καλύτερο ποσοστό πρόβλεψης Q3 από το καλύτερο μέχρι τώρα ποσοστό των 73.92%.

Εφαρμόσαμε λοιπόν των κώδικα των γενετικών αλγορίθμων ακριβώς όπως το πήραμε από τον Αγαθοκλέους (Αγαθοκλέους, 2009) κρατώντας τις ίδιες τιμές στις

παραμέτρους του γενετικού αλγορίθμου (πίνακας 5.8), στο σύστημα το οποίο συμπεριλαμβάνει τα MSA profiles. Να σημειωθεί ότι για την μέθοδο κωδικοποίησης των νευρώνων εξόδου, αφέθηκε η υφιστάμενη και όχι η μέθοδος *WTA*.

ΠΑΡΑΜΕΤΡΟΣ ΓΕΝΕΤΙΚΟΥ ΑΛΓΟΡΙΘΜΟΥ	ΤΙΜΗ
Population number	20
Crossover probability	0.25
Mutation probability	0.01
Generation number	10

Πίνακας 5.8*: Πίνακας με τις παραμέτρους αρχικοποίησης του προγράμματος των γενετικών αλγορίθμων*.

Η εφαρμογή των γενετικών αλγορίθμων στο τρέχον πρόγραμμα, αύξησε κατά πολύ την ήδη εκτεταμένη πολυπλοκότητα του, με αποτέλεσμα, λόγω του μεγάλου χρόνου εκτέλεσης που χρειάζεται, να εκτελέσουμε μόνο ένα πείραμα. Το αποτέλεσμα του πειράματος αυτού έδωσε σαν «βέλτιστο» ποσοστό το ποσοστό 72.8074%, το οποίο είναι κατά 1% μικρότερο από το βέλτιστο ποσοστό που παίρνουμε χωρίς την χρήση των γενετικών αλγορίθμων (73.92%, σχήμα 5.11). Δυστυχώς, στην συγκεκριμένη περίπτωση, η εφαρμογή των γενετικών αλγορίθμων δεν έδωσε καλύτερα αποτελέσματα από τα υφιστάμενα. Αυτό, πιθανόν να οφείλεται στο γεγονός ότι ο αλγόριθμος δεν μπορεί να συγκλίνει εξαιτίας της τιμής της παραμέτρου που αφορά τον αριθμό των γενεών που θα εκτελέσει. Επειδή η συγκεκριμένη τιμή είναι αρκετά μικρή, όντως πιθανόν να μην μπορεί να συγκλίνει, άρα πρέπει να αυξήσουμε την τιμή της παραμέτρου αυτής.

Έτσι λοιπόν δοκιμάσαμε ένα νέο πείραμα με τους γενετικούς αλγορίθμους, για *Generation Number* ίσο με 25. Αυτά τα αποτελέσματα όμως έδειξαν κάτι αρκετά παράξενο: Ο γενετικός αλγόριθμος τις πλείστες γενεές έδινε σαν καλύτερο ποσοστό για την παράμετρο Q3 0%, μηδενίζοντας όλες τις τιμές των παραμέτρων του δικτύου

για τις οποίες ψάχνουμε την βέλτιστη τιμή τους. Σίγουρα, αυτό δεν είναι λογικό, άρα κάποιο πρόβλημα πρέπει να υπάρχει με τον αλγόριθμο.

Όπως παρατηρήθηκε αργότερα, ο γενετικός αλγόριθμος όπως είναι σχεδιασμένος και υλοποιημένος, επιτρέπει μηδενικές τιμές για όλες τις παραμέτρους, δηλαδή υπάρχει πρόβλημα στην κωδικοποίηση του πεδίου τιμών των παραμέτρων σε χρωμοσώματα. Το πεδίο των τιμών αυτών, επέτρεπε την περίπτωση να έχουμε χρωμοσώματα με μηδενικές τιμές για όλα τα κρυφά επίπεδα ταυτόχρονα, με αποτέλεσμα το δίκτυο να έχει μόνο επίπεδο εισόδου και εξόδου, κάτι που δεν είναι καθόλου λογικό. Τα χρωμοσώματα αυτά, παρόλο που αναγνωρίζονται σαν μη αποδεκτά από μια εντολή ελέγχου στο πρόγραμμα, εντούτοις τους δίνεται η δυνατότητα να μπουν στην δεξαμενή διασταύρωσης, με σκορ 0%. Δίνοντας τους έτσι την πιθανότητα να επιλεχθούν για την νέα γενεά χρωμοσωμάτων και σε συνδυασμό με τον χαμηλό αριθμό γενεών που έχουμε (25 γενεές), το πρόγραμμα, δεν μπορεί ακόμη να συγκλίνει ώστε να τα απορρίψει από την επιλογή του. Υπολογίζεται ότι λόγω αυτής της σχεδίασης ο αλγόριθμος εμφανίζει το πιο πάνω πρόβλημα.

Για να λύσουμε το πρόβλημα αυτό, τροποποιήσαμε κάποια κομμάτια του αλγορίθμου αυτού. Σαν πρώτο βήμα, διαγράψαμε τον έλεγχο που υπήρχε στην κλάση *pssp_ucy*, ο οποίος έλεγχε εάν τα *hLayerOneSize, hLayerOneSizeB* και *hLayerOneSizeF* είχαν τιμές μηδενικές και εάν ναι, η συνάρτηση αξιολόγησης τους έδινε την τιμή 0. Στην συνέχεια, τροποποιήσαμε τα όρια των τιμών που μπορούν να πάρουν τα χρωμοσώματα που αναπαριστούν τα μεγέθη *hLayerOneSize, hLayerOneSizeB* και *hLayerOneSizeF.* Τα νέα όρια τιμών αλλάχθηκαν από την κλάση *chromosome.cpp* και συγκεκριμένα από την μέθοδο *convert().*

Τα χρωμοσωμάτα αυτά, λέγονται *HiddenLayerOneSize, HiddenLayerOneOfBackwardSize* και *HiddenLayerOneOfForwardSize,* αντιστοιχούν με τα μεγέθη *hLayerOneSize, hLayerOneSizeB* και *hLayerOneSizeF* αντίστοιχα της κλάσης *pssp_ucy.cpp* και συμβολίζουν τον αριθμό των νευρώνων για τα *hLayerOneF, hLayerOneB, hLayerOne* (σχήμα 2.1). Τα νέα όρια τους είναι από 2 μέχρι και 60, δηλαδή το ελάχιστο που μπορούν να πάρουν είναι 2 νευρώνες και το μέγιστο 60 νευρώνες. Το ίδιο έγινε και για τα χρωμοσώματα *WindowSize* και *sVar* που

αναπαριστούν τα μεγέθη *windowSize* και *s* της κλάσης *pssp_ucy.cpp* αντίστοιχα. Αυτά έχουν τις τιμές του κινούμενου παραθύρου και τον αριθμό εισόδου του δικτύου για vanishing gradient (Baldi et al., 1999). Το μέγεθος του παραθύρου αρχικοποιείται μεταξύ 9 και 31, και το μέγεθος της εισόδου του δικτύου για vanishing gradient, από 3 μέχρι και 5. Να σημειωθεί ότι λόγω έλλειψης χρόνου, τα χρωμοσώματα για τον ρυθμό μάθησης, για τον όρο της ορμής και για τις χρονικές μεταβλητές δεν τροποποιήθηκαν, αλλά θα τις θεωρούμε στην υφιστάμενη εργασία σαν σταθερές τιμές (επειδή είναι συνεχείς τιμές). Ο τροποποιημένος κώδικας των Γενετικών Αλγορίθμων φαίνεται στο Παράρτημα Γ.

5.4 Αποτελέσματα τροποποίησης της αρχιτεκτονικής του συστήματος

Μια άλλη μεθοδολογία που θα εφαρμοστεί σε αυτήν την διπλωματική εργασία, θα αφορά πειράματα αποκλειστικά για την αρχιτεκτονική του συστήματος, συμπεριλαμβανομένου των MSA profiles και της τυχαιοποίησης. Αυτό θα γίνει γιατί θέλουμε να δούμε εάν υπάρχει μεγάλη πολυπλοκότητα στο δίκτυο, ακολουθώντας την συγκεκριμένη αρχιτεκτονική του σχήματος 2.1, αλλά και γενικά την συμπεριφορά του δικτύου όταν γίνονται τέτοιες αλλαγές.

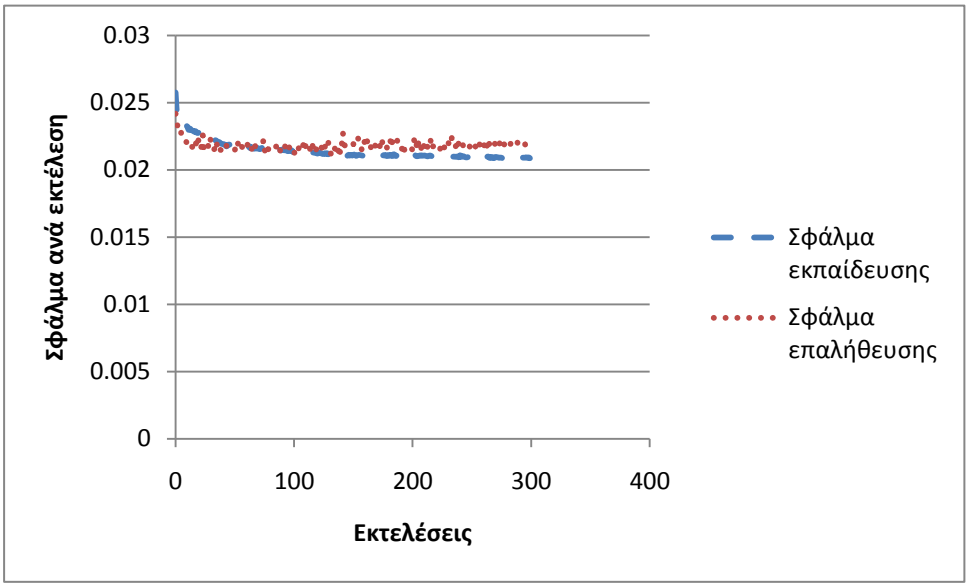
Σε όλα τα πειράματα που έγιναν μέχρι τώρα, η αρχιτεκτονική του δικτύου αφορούσε δυο κρυφά επίπεδα για το κάθε ένα νευρωνικό δίκτυο ανάδρασης αλλά και για το δίκτυο εμπρόσθιου περάσματος. Θέλουμε λοιπόν να δούμε τι θα γίνει όταν η αρχιτεκτονική αυτή αλλάξει: θα κρατήσουμε μόνο ένα κρυφό επίπεδο για κάθε δίκτυο ανάδρασης καθώς και για το δίκτυο εμπρόσθιου περάσματος. Συγκεκριμένα, τα *hLayerTwoF, hLayerTwoB, hLayerTwo* (σχήμα 2.1), θα μηδενιστούν. Η αρχιτεκτονική του προγράμματος προσαρμόζεται αυτόματα ώστε τα context επίπεδα να παίρνουν τις τιμές εξόδου των επιπέδων *hLayerOneF, hLayerOneB, hLayerOne*. Η μέθοδος κωδικοποίησης των νευρώνων εξόδου είναι η υφιστάμενη και όχι η μέθοδος *WTA*. Να σημειωθεί ότι οι υπόλοιπες παράμετροι του δικτύου αρχικοποιούνται με βάση τις τιμές που οδήγησαν στα καλύτερα αποτελέσματα πρόβλεψης με βάση την παράμετρο Q3 και συγκεκριμένα τις τιμές του πίνακα 5.3. Η μόνη διαφορά, είναι η τιμή της

παραμέτρου της χρονικής σταθεράς q, της οποίας η τιμή 0.7 έδωσε κατά μέσο όρο καλύτερα αποτελέσματα στα πιο κάτω πειράματα παρά η τιμή 0.6. Να σημειωθεί ότι για αρχή θα τρέξουμε το σύστημα για 300 επαναλήψεις σε συνδυασμό με διάφορα μεγέθη των επιπέδων, ώστε να δούμε μια γενική συμπεριφορά του δικτύου. Στην συνέχεια θα μπορούμε να γνωρίζουμε εάν υπάρχει υπερεκπαίδευση ή άλλα προβλήματα, ώστε να τα διορθώσουμε.

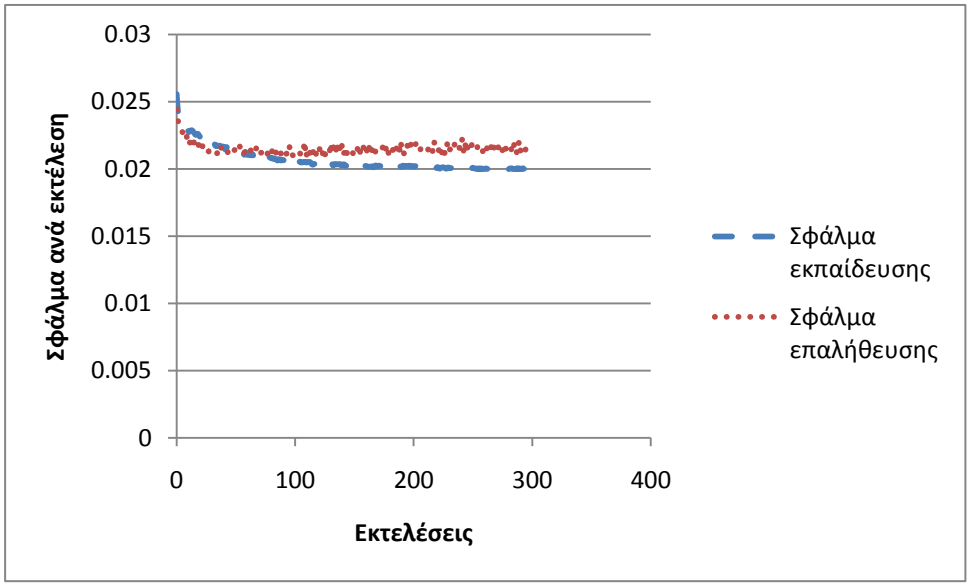
No	Architecture	Number of Iterations	Q3 Score	SOV score
1	31 - 0 - 21 - 0 - 21 - 0	300	74.06%	59.42%
2	61 - 0 - 41 - 0 - 41 - 0	300	74.37%	59.05%
3	51 - 0 - 31 - 0 - 31 - 0	300	74.28%	60.03%
4	41 - 0 - 31 - 0 - 31 - 0	300	73.50%	59.98%

***Πίνακας 5.8**: Πίνακας με τα πρώτα αποτελέσματα που αφορούν την αλλαγή της αρχιτεκτονικής του δικτύου για 300 επαναλήψεις.*

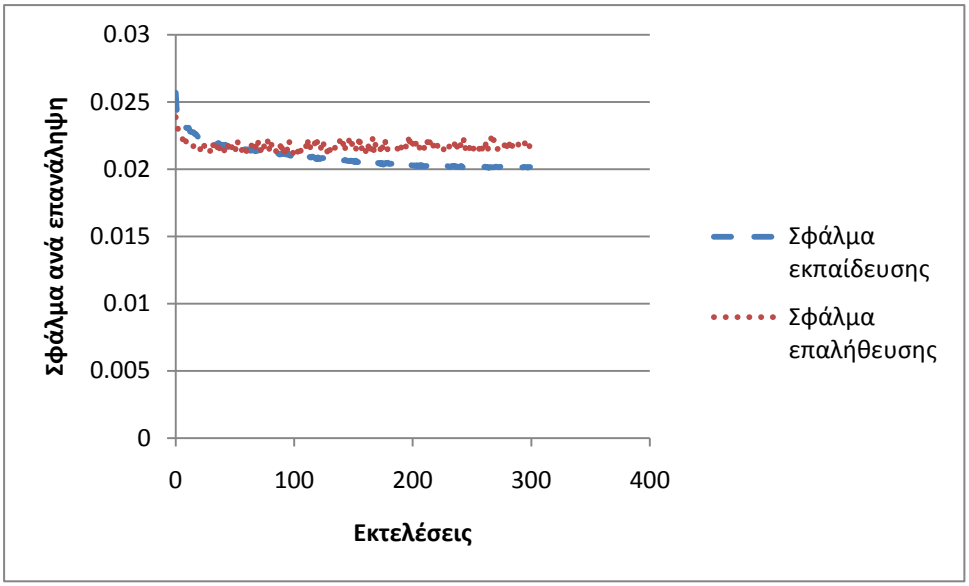
Από τον πίνακα 5.8, παρατηρούμε ότι, η αλλαγή της αρχιτεκτονικής του δικτύου οδηγεί επί το πλείστον σε καλύτερα αποτελέσματα όσον αφορά την παράμετρο Q3. Τα τελευταία καλύτερα αποτελέσματα όσον αφορά την παράμετρο Q3 που είχαμε μέχρι τώρα ήταν 73.92% (σχήμα 5.11), ενώ με αυτήν την περίπτωση τα αποτελέσματα φαίνονται να είναι καλύτερα, αφού δίνουν ακρίβεια πρόβλεψης μέχρι και 74.37%. Παρόλα αυτά, πρέπει να δούμε την γενική συμπεριφορά του δικτύου για τις εκτελέσεις αυτές, ώστε να μπορούμε να βγάλουμε πιο συγκεκριμένα συμπεράσματα. Για να το δούμε αυτό θα μελετήσουμε τα σχήματα 5.20, 5.21, 5.22 και 5.23 όπου δείχνουν την μεταβολή του σφάλματος εκπαίδευσης και επαλήθευσης καθ' όλη την διάρκεια της εκτέλεσης για τις εκτελέσεις του πίνακα 5.8.



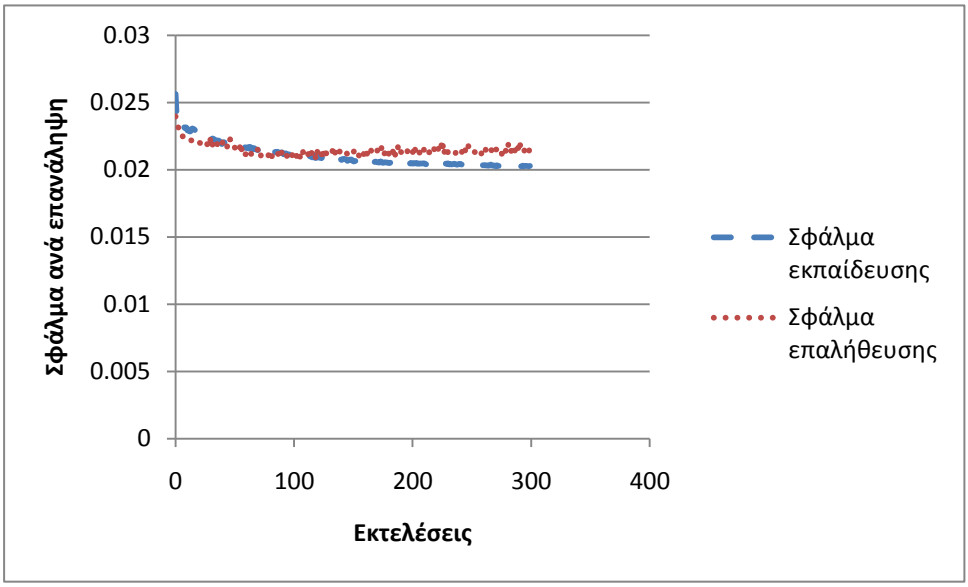
***Σχήμα 5.20**: Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για την εκτέλεση 1 από τον πίνακα 5.8.*



***Σχήμα 5.21**: Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για την εκτέλεση 2 από τον πίνακα 5.8.*



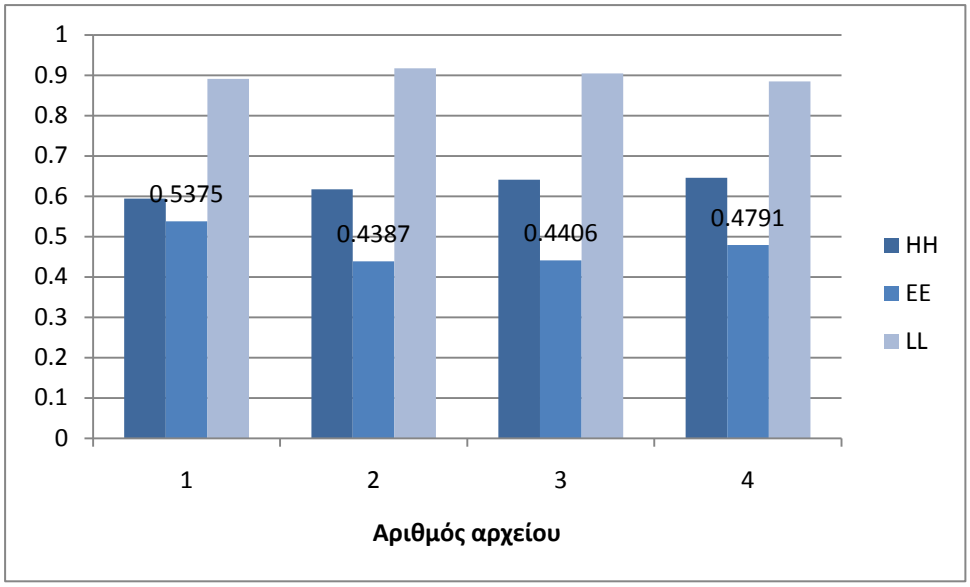
***Σχήμα 5.22**: Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για την εκτέλεση 3 από τον πίνακα 5.8.*



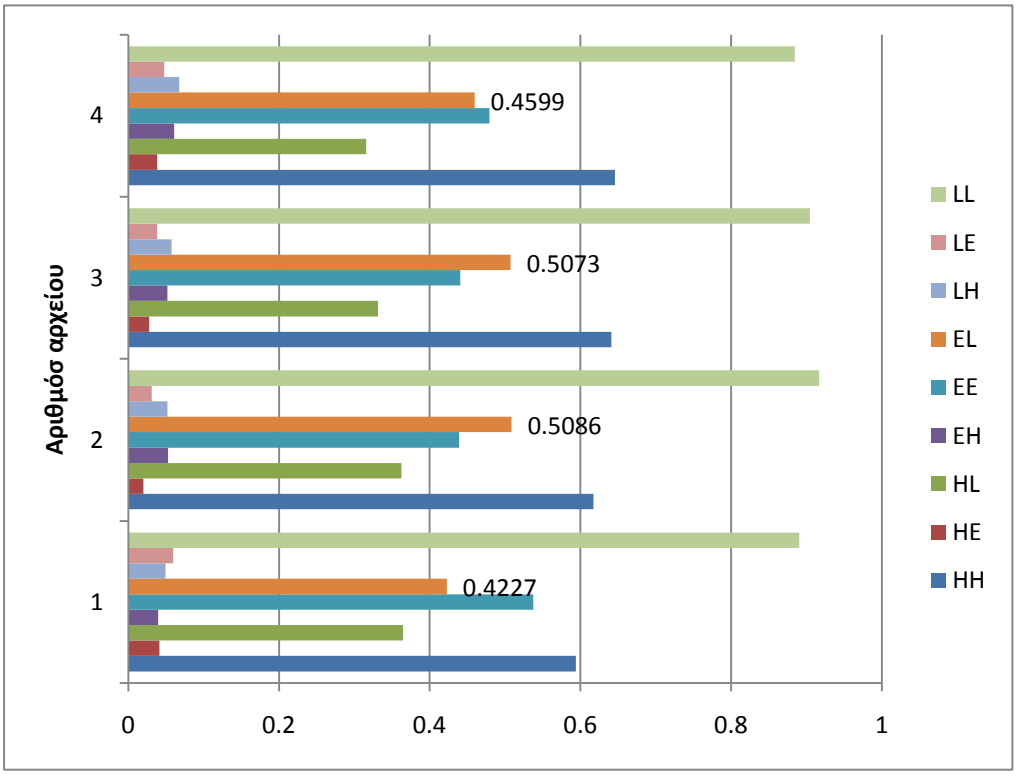
***Σχήμα 5.23**: Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για την εκτέλεση 4 από τον πίνακα 5.8.*

Με βάση τα σχήματα 5.20 – 5.23, βλέπουμε αρκετά παράξενα αποτελέσματα. Το σφάλμα εκπαίδευσης συνεχίζει να ελαττώνεται, ενώ το σφάλμα επαλήθευσης παραμένει κάπως σταθερό. Αυτό, δεν σημαίνει κατ' ανάγκην ότι έχουμε πρόβλημα στην εκπαίδευση του δικτύου, απλά φαίνεται ότι το σύνολο των δεδομένων επαλήθευσης είναι πιο *εύκολο* σύνολο παρά το σύνολο εκπαίδευσης. Αξίζει να σημειώσουμε ότι για τα αποτελέσματα αυτά δεν μπορούμε να πούμε ότι το δίκτυο υπερεκπαιδεύεται, αφού το σφάλμα επαλήθευσης παραμένει σταθερό σε κάποια τιμή και δεν αυξάνεται σε σχέση με τον αριθμό εκτελέσεων.

Συνεχίζοντας, πρέπει να μελετήσουμε τι γίνεται με την παράμετρο SOV επειδή είναι πολύ σημαντική για προβλήματα πρόβλεψης δομής πρωτεϊνών (υποκεφάλαιο 3.4.2). Το καλύτερο μέχρι τώρα αποτέλεσμα για την SOV παράμετρο ήταν το αντίστοιχο του καλυτέρου αποτελέσματος για το Q3 σκορ, και όπως φαίνεται από το σχήμα 5.16 ήταν 62.38%. Τα αντίστοιχα SOV σκορ για τα πειράματα του συγκεκριμένου υποκεφαλαίου φαίνονται στον πίνακα 5.8. Παρατηρούμε ότι σε σχέση με το μέχρι τώρα καλύτερο ποσοστό, τα συγκεκριμένα SOV ποσοστά είναι πάρα πολύ χαμηλά, και δεν συμβαδίζουν με την αύξηση του Q3 σκορ, δεν πλησιάζουν καν το 63% του καλύτερου SOV που έχουμε μέχρι τώρα. Γι αυτό θέλουμε να δούμε τι αποτελέσματα δίνουν τα confusion matrices, ώστε να διαπιστώσουμε που είναι το πρόβλημα.



Σχήμα 5.24: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα για τα αρχεία του πίνακα 5.8.*



Σχήμα 5.25*: Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα καθώς και τις λανθασμένες προβλέψεις HE, HL, EH, EL, LE, LH για τις εκτελέσεις του πίνακα 5.8.*

Απ ότι βλέπουμε από το σχήμα 5.24 και 5.25, έχουμε προβλήματα όσον αφορά την πρόβλεψη της δευτεροταγούς δομής των (E). (Στο σχήμα 5.16, το καλύτερο ποσοστό πρόβλεψης των (E) ήταν 52.66%, για το αρχείο 6). Στα τρέχον πειράματα τα αποτελέσματα της σωστής πρόβλεψης του (E) είναι πάρα πολύ χαμηλά. Η κλάση E συγχέεται με την κλάση L, και συγκεκριμένα, λιγότερο από το 50% των E προβλέπεται σαν L σε κάθε εκτέλεση. Τα αποτελέσματα αυτά, ίσως να οφείλονται στους προηγούμενους λόγους που αναφέρθηκαν στο υποκεφάλαιο 5.2.5. Θέλουμε στο παρόν στάδιο να μελετήσουμε ξανά την κωδικοποίηση εξόδου του συστήματος, επειδή έξι τριαδικοί συνδυασμοί κωδικοποιούν την ομάδα των (L), ένας τριαδικός συνδυασμός την ομάδα των (H) και ένας την ομάδα των (E). Θέλουμε λοιπόν, να δώσουμε *ίσες* ευκαιρίες και στα τρία αποτελέσματα. Γι αυτό θα χρησιμοποιήσουμε ξανά την μέθοδο *WTA*, την οποία αναφέραμε στο κεφάλαιο 5.4, σε συνδυασμό με τις αρχιτεκτονικές τις οποίες μελετούμε σε αυτό το υποκεφάλαιο.

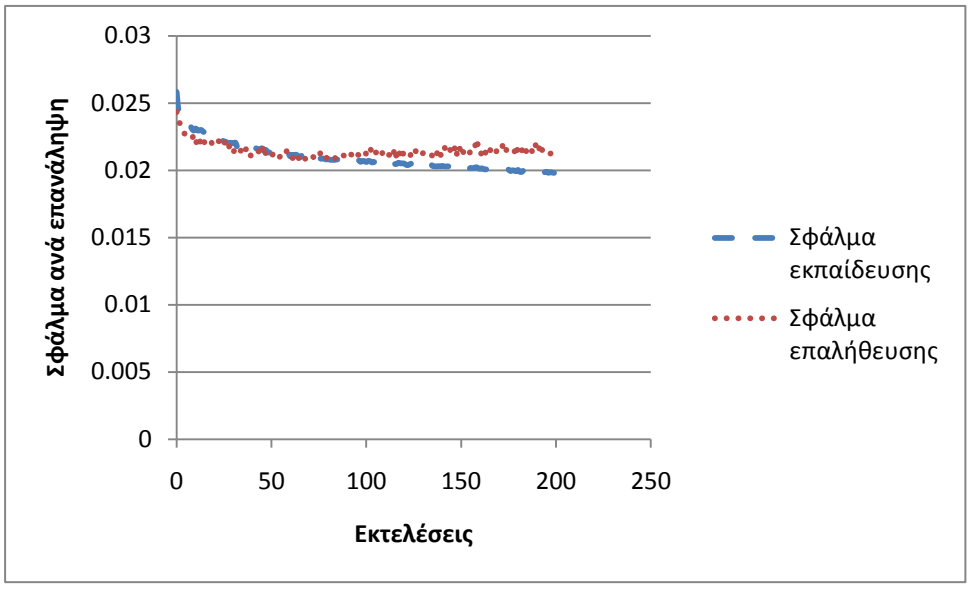
5.4.1 Αποτελέσματα τροποποίησης της αρχιτεκτονικής του δικτύου σε συνδυασμό με την WTA μέθοδο

No	Architecture	Number of Iterations	Q3 Score	SOV Score
1	51 - 0 - 41 - 0 - 41 - 0	200	75.26%	65.21%
2	51 - 0 - 41 - 0 - 41 - 0	200	75.49%	64.45%
3	51 - 0 - 41 - 0 - 41 - 0	200	76.07%	62.43%
4	19 - 0 - 21 - 0 - 21 - 0	200	74.97%	63.08%
5	14 - 0 - 17 - 0 - 17 - 0	200	74.63%	65.18%
6	11 - 0 - 13 - 0 - 13 - 0	300	74.98%	61.30%
7	15 - 0 - 17 - 0 - 17 - 0	200	76.07%	64.32%

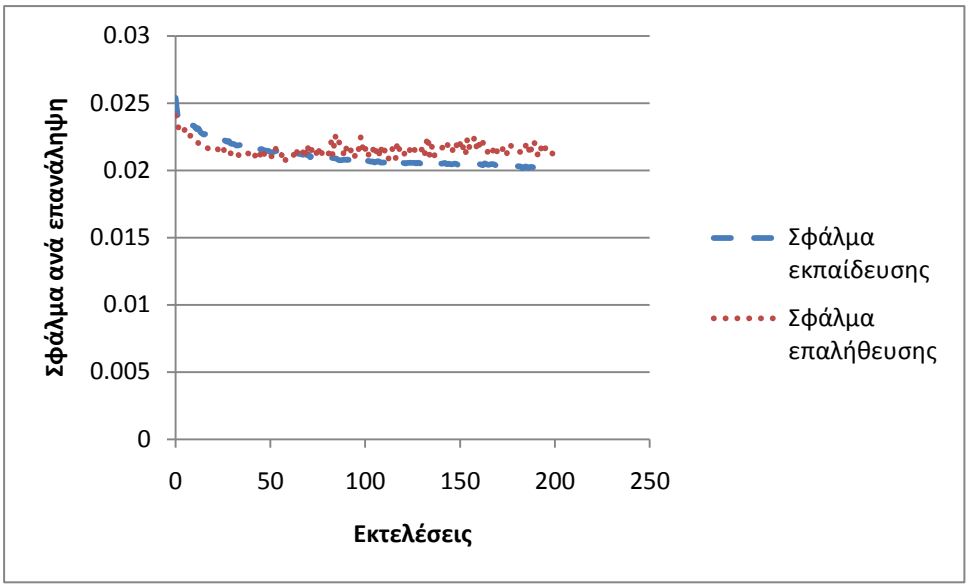
Πίνακας 5.9*: Πίνακας με τα αποτελέσματα που αφορούν την αλλαγή της αρχιτεκτονικής του δικτύου δεδομένου ότι έγινε η εφαρμογή της μεθόδου WTA για την κωδικοποίηση της εξόδου.*

Εκτελέσαμε εφτά νέες προσομοιώσεις αλλάζοντας την αρχιτεκτονική του δικτύου, και εφαρμόζοντας την μέθοδο *WTA* για την κωδικοποίηση της εξόδου (πίνακας 5.9). Με βάση τα αποτελέσματα του πίνακα 5.9 όσον αφορά την Q3 παράμετρο αλλά και την SOV παράμετρο, βλέπουμε ότι έχουμε πολύ σημαντικά αποτελέσματα να παρουσιάσουμε, αφού τα αποτελέσματα αυτά, ξεπερνούν μέχρι τώρα τα βέλτιστα που είχαμε, τόσο σε Q3 ποσοστό όσο και σε SOV. Φυσικά, οι αντιστοιχίες Q3 με SOV δεν είναι γραμμικές, δηλαδή δεν σημαίνει απαραίτητα ότι όσο αυξάνεται το Q3 αυξάνεται και το SOV, όπως φαίνεται και από τα αποτελέσματα του πίνακα 5.9. Αυτό οφείλεται επί το πλείστον σε στατιστικά σφάλματα.

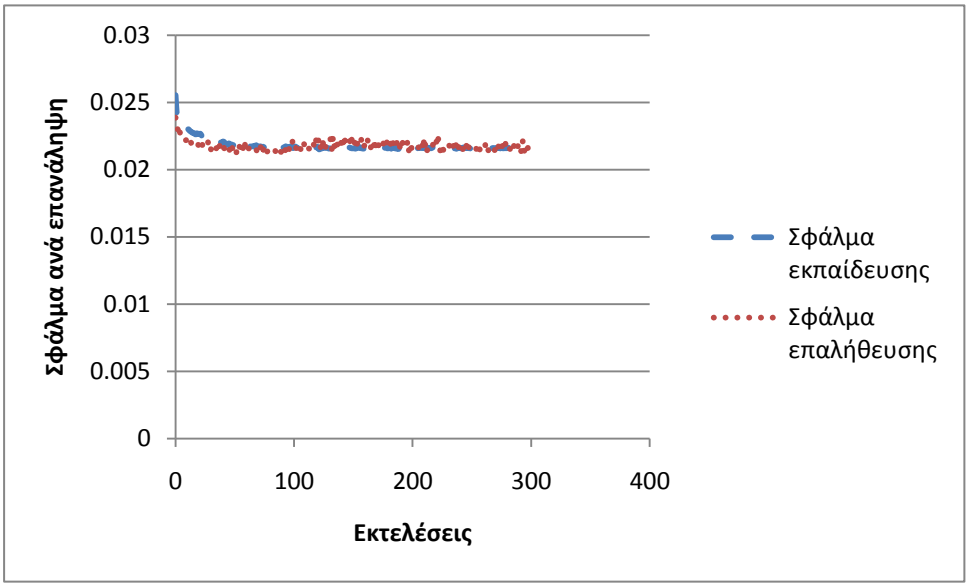
Κατ αρχήν θέλουμε να δούμε τις γραφικές παραστάσεις των σφαλμάτων εκπαίδευσης και επαλήθευσης, για τις εκτελέσεις 2, 3, 6 και 7, ώστε να δούμε πως επηρεάζεται η μορφή τους όταν χρησιμοποιείται η μέθοδος WTA.



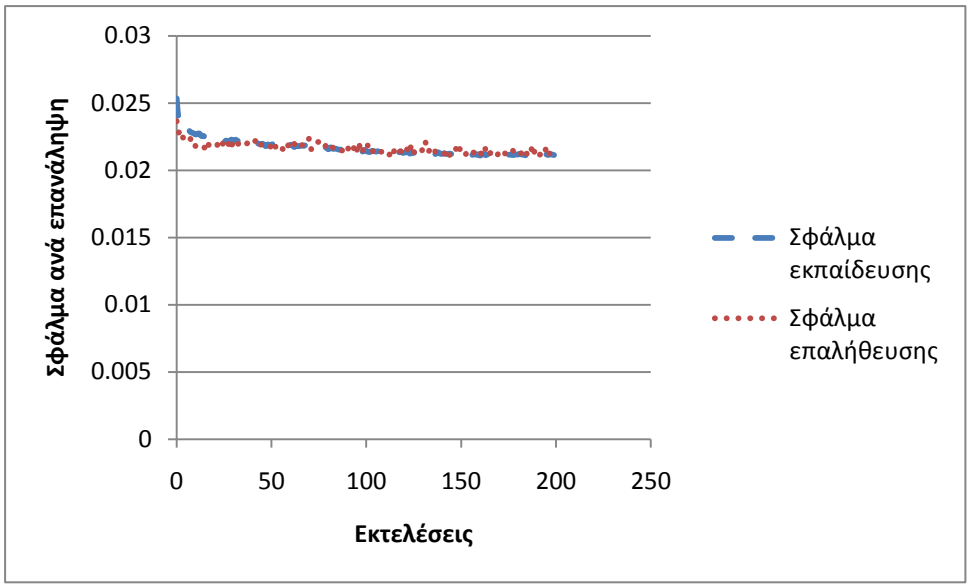
Σχήμα 5.26*: Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για την εκτέλεση 2 από τον πίνακα 5.9.*



***Σχήμα 5.27**: Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για την εκτέλεση 3 από τον πίνακα 5.9.*



***Σχήμα 5.28**: Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για την εκτέλεση 6 από τον πίνακα 5.9.*



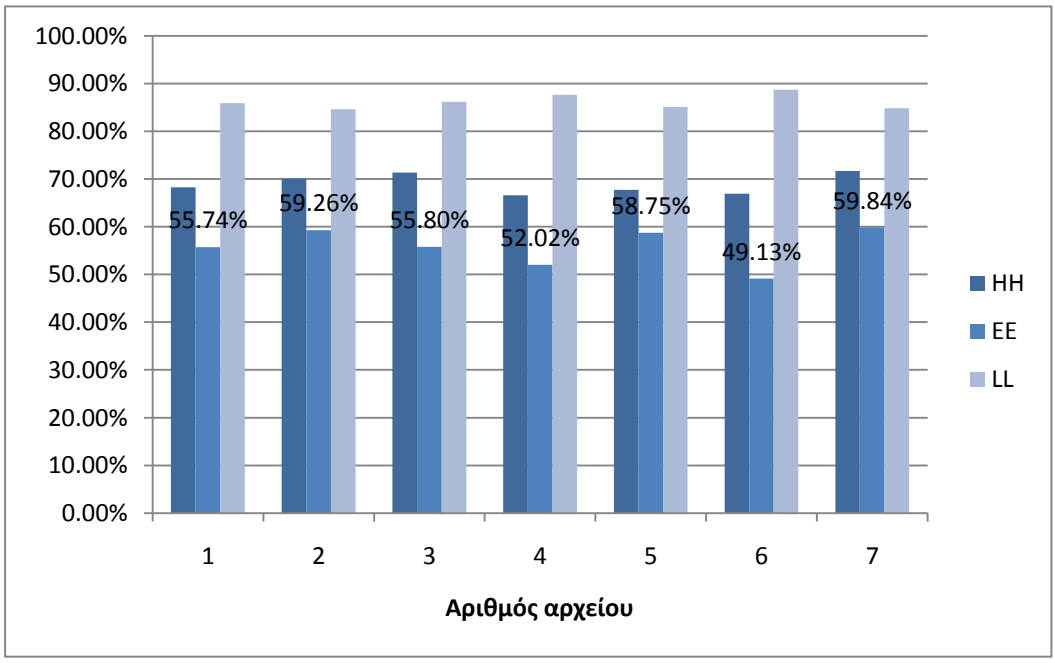
***Σχήμα 5.29**: Γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης για την εκτέλεση 7 από τον πίνακα 5.9.*

Από τα σχήματα 5.26 – 5.29, παρατηρούμε την μεταβολή του σφάλματος εκπαίδευσης και επαλήθευσης των εκτελέσεων 2, 3, 6 και 7 αντίστοιχα (πίνακας 5.9), ανάλογα με τον αριθμό εκτελέσεων της κάθε μιας από τις 4 εκτελέσεις. Παρατηρούμε το ίδιο φαινόμενο που παρατηρήσαμε για τα ανάλογα σχήματα του πίνακα 5.8 (5.20 – 5.23), με την διαφορά ότι σε εκείνη την περίπτωση δεν είχαμε χρησιμοποιήσει την μέθοδο κωδικοποίησης της εξόδου σαν *WTA*. Βλέπουμε γενικά, ότι με την εφαρμογή της νέας αυτής μεθόδου, η μορφή των αποτελεσμάτων που αφορούν το σφάλμα εκπαίδευσης και επαλήθευσης δεν αλλάζουν.

Βλέποντας τις γραφικές αυτές παραστάσεις από τα σχήματα 5.26 – 5.29, αλλά και από τα 5.20 – 5.23, εκτελέσαμε ακόμα ένα τέτοιο πείραμα με βάση την αρχιτεκτονική της προσομοίωσης 3 (πίνακας 5.9), αλλά με περισσότερες επαναλήψεις και συγκεκριμένα 500 επαναλήψεις. Αυτό γιατί θέλουμε να δούμε αν τα σφάλματα συγκλίνουν σε κάποιο στάδιο και αν όντως γίνεται έτσι, τα αποτελέσματα πιθανόν να είναι καλύτερα. Το αποτέλεσμα ήταν 75.03%, με μια παρόμοια γραφική παράσταση του σφάλματος εκπαίδευσης και επαλήθευσης όπως το σχήμα 5.27. Το σφάλμα, είναι χαμηλότερο από το σφάλμα για την εκτέλεση 3 του πίνακα 5.9, έτσι το σφάλμα

φαίνεται ότι έχει φτάσει σε μια συγκεκριμένη τιμή και έχει μικρές ταλαντώσεις γύρω του. Αυξάνοντας έτσι, τον αριθμό των εκτελέσεων μιας τέτοιας προσομοίωσης, δεν παρατηρούμε σημαντική διαφορά στα αποτελέσματα.

Στην συνέχεια θέλουμε να μελετήσουμε τα confusion matrices των εκτελέσεων του πίνακα 5.9, ώστε να δούμε που οφείλεται αυτή η αύξηση του ποσοστού SOV, έστω και ελάχιστη που παρατηρείται σε αυτό το υποκεφάλαιο σε σύγκριση με τα αποτελέσματα του υποκεφαλαίου 5.6 και συγκεκριμένα του πίνακα 5.8. Τα αποτελέσματα που έδωσαν τα confusion matrices φαίνονται στο σχήμα 5.30 και 5.31.

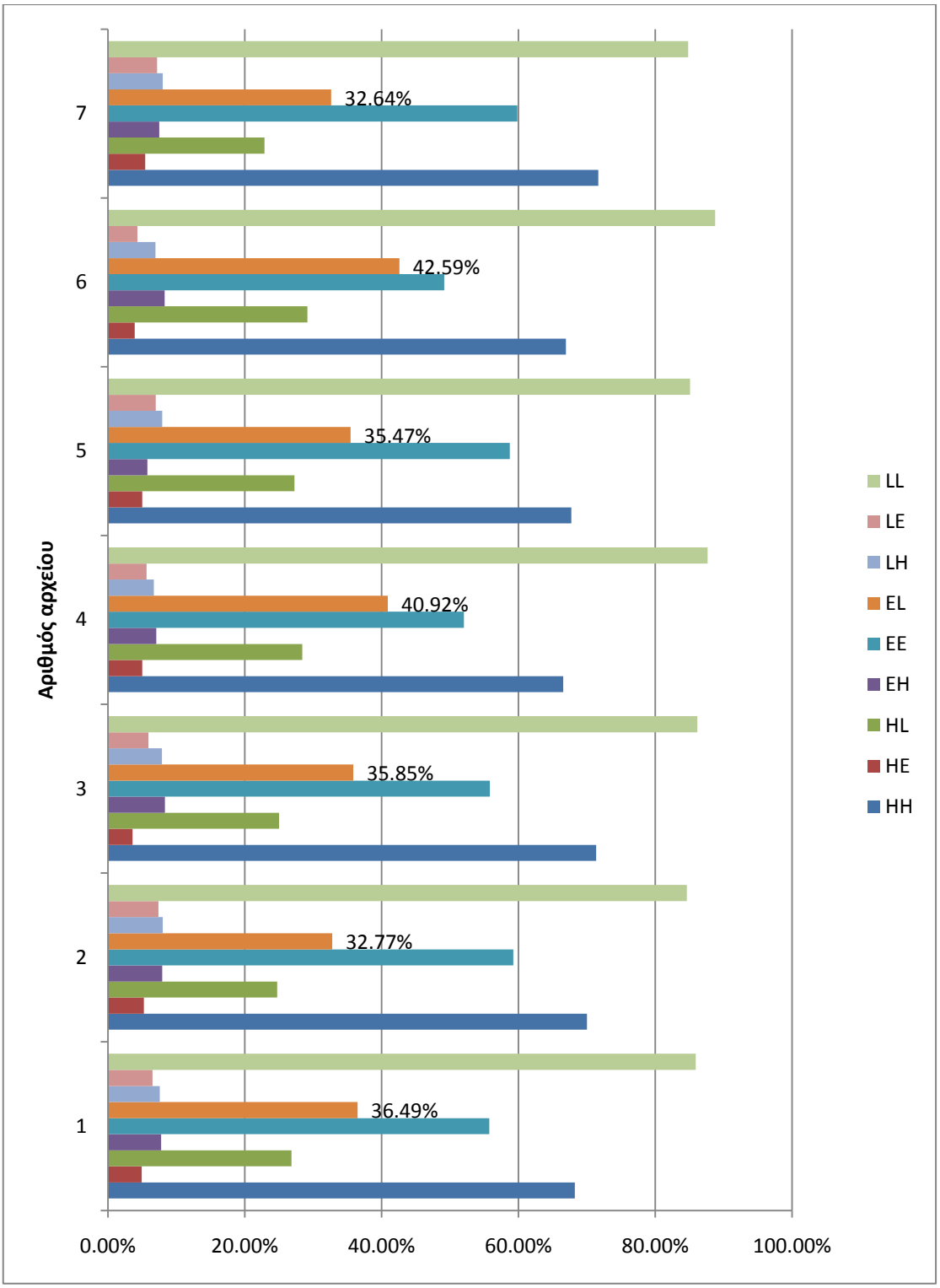


Σχήμα 5.30: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα.*

Από το σχήμα 5.30 παρατηρούμε πολύ καλά αποτελέσματα όσον αφορά την *σωστή* πρόβλεψη που πρέπει να κάνει το δίκτυο, δηλαδή την HH, EE και LL πρόβλεψη, σε σύγκριση με τα αποτελέσματα του προηγούμενου υποκεφαλαίου (υποκεφάλαιο 5.6), όπου δεν χρησιμοποιήθηκε η μέθοδος *WTA*, αλλά η κανονική μέθοδος κωδικοποίησης εξόδου του συστήματος. Τα αποτελέσματα με το σχήμα 5.24, του προηγούμενου υποκεφαλαίου, όσον αφορά την πρόβλεψη EE ήταν λιγότερα από 50% και αυτό ήταν ένα από τα σημαντικά προβλήματα που συναντήσαμε σε όλη την

διπλωματική εργασία. Από το σχήμα 5.30 βλέπουμε ότι τα αποτελέσματα πρόβλεψης ΕΕ είναι μεταξύ 50% – 60%. Δεν παρουσιάζεται μια μεγάλη διαφορά στα αποτελέσματα, όμως παρουσιάζεται έστω και μια μικρή διαφορά, η οποία μας κάνει να συμπεράνουμε ότι η χρήση *WTA* τις περισσότερες φορές μπορεί να αυξήσει την πρόβλεψη ΕΕ, και συνεπώς, μπορεί να *βοηθήσει* στο πρόβλημα της πρόβλεψης της κλάσης των Extended.

Στο σχήμα 5.31 βλέπουμε και τα υπόλοιπα μέρη των confusion matrices. Στο σχήμα 5.25 η πρόβλεψη των E συγχεόταν με την κλάση των L με αποτέλεσμα το 50% σχεδόν των E να προβλέπεται σαν L. Αυτό το γεγονός, βελτιώνεται με την χρήση του WTA, όπως παρατηρούμε και από τις τιμές επάνω στο σχήμα 5.31. Το ποσοστό των E που προβλέπεται σαν L είναι περίπου 30% - 40%, που είναι πολύ σημαντική διαφορά με το σχήμα 5.25.



Σχήμα 5.31: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα καθώς και τις λανθασμένες προβλέψεις HE, HL, EH, EL, LE, LH.*

5.5 *Φιλτράρισμα* προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών

Το πρόβλημα πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, έχουμε προαναφέρει στο υποκεφάλαιο 1.1 ότι δεν είναι ένα απλό πρόβλημα. 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 αμινοξέως. 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).

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).

5.5.1 Αποτελέσματα *φιλτραρίσματος* προβλεπόμενης δευτεροταγούς δομής πρωτεϊνών με χρήση HMM

Παρόμοια τεχνική *φιλτραρίσματος* προβλεπόμενων πρωτεϊνών, θα ακολουθηθεί και στην τρέχουσα διπλωματική εργασία. Λαμβάνοντας υπόψη τις θεωρίες του υποκεφαλαίου 5.7, θα χρησιμοποιήσουμε για το «φιλτράρισμα» των δεδομένων εξόδου του υφιστάμενου δικτύου ένα *Hidden Markov Model* (HMM), που περιγράφεται από τον αλγόριθμο *viterbi* (Durbin et al., 1998). 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.

Για να το εφαρμόσουμε αυτό, πήραμε τα καλύτερα αποτελέσματα που είχαμε μέχρι τώρα, δηλαδή τα αποτελέσματα των εκτελέσεων του πίνακα 5.9 και τα περάσαμε μέσα από το πρόγραμμα «viterbi.pl», το οποίο εφαρμόζει τον ομώνυμο αλγόριθμο, υλοποιημένο από τον Βασίλειο Προμπονά, λέκτορα τμήματος Βιολογικών Επιστημών Πανεπιστήμιου Κύπρου. Το πρόγραμμα αυτό, παίρνει σαν είσοδο τα αποτελέσματα εξόδου που αφορούν την εκπαίδευση του δικτύου, και τα αποτελέσματα εξόδου που αφορούν την επαλήθευση του δικτύου.

Στην συνέχεια, τα αποτελέσματα αυτά θα περάσουν από το πρόγραμμα «confusionMatricesAndSOV.pl». Συγκεκριμένα αυτό που δίνουμε στο πρόγραμμα αυτό είναι την βελτιωμένη ακολουθία της κάθε πρωτεΐνης, όπως την πήραμε από το πρόγραμμα «viterbi.pl» και έπειτα παίρνουμε τα confusion matrices και το ποσοστό της παραμέτρου SOV, δεδομένου της βελτιωμένης ακολουθίας.

Το ποσοστό της παραμέτρου SOV για τις εκτελέσεις του πίνακα 5.9, φαίνεται στον πίνακα 5.10.

No	Architecture	SOV Score
1	51 - 0 - 41 - 0 - 41 - 0	69.09%
2	51 - 0 - 41 - 0 - 41 - 0	69.51%
3	51 - 0 - 41 - 0 - 41 - 0	69.15%
4	19 - 0 - 21 - 0 - 21 - 0	67.91%
5	14 - 0 - 17 - 0 - 17 - 0	69.22%
6	11 - 0 - 13 - 0 - 13 - 0	66.47%
7	15 - 0 - 17 - 0 - 17 - 0	70.32%

Πίνακας 5.10: *Αποτελέσματα SOV παραμέτρου μετά την χρήση HMM για τις εκτελέσεις του πίνακα 5.9.*

Από τον πίνακα 5.10, βλέπουμε πολύ σημαντικές βελτιώσεις σε σύγκριση με τα αποτελέσματα του πίνακα 5.9. Συγκεκριμένα, τα ποσοστά με βάση την SOV παράμετρο κυμαίνονται από 67% - 70%. Αυτό είναι πάρα πολύ σημαντικό, διότι τα HMM, δεν έδιναν σημαντική βελτίωση για προηγούμενα πειράματα εκτός του υποκεφαλαίου 5.6 (δεν παρουσιάστηκαν). Έτσι υποθέταμε ότι δεν μπορούσε να δουλέψει ένα τυπικό HMM για το συγκεκριμένο σύστημα και απορρίφτηκε από τις μεθόδους βελτιστοποίησης των αποτελεσμάτων του. Όμως στο συγκεκριμένο πείραμα φαίνεται ότι λειτουργεί, και μάλιστα αρκετά καλά, αφού αυξάνει τα SOV ποσοστά σε σύγκριση με τον πίνακα 5.9, μέχρι και 5%.

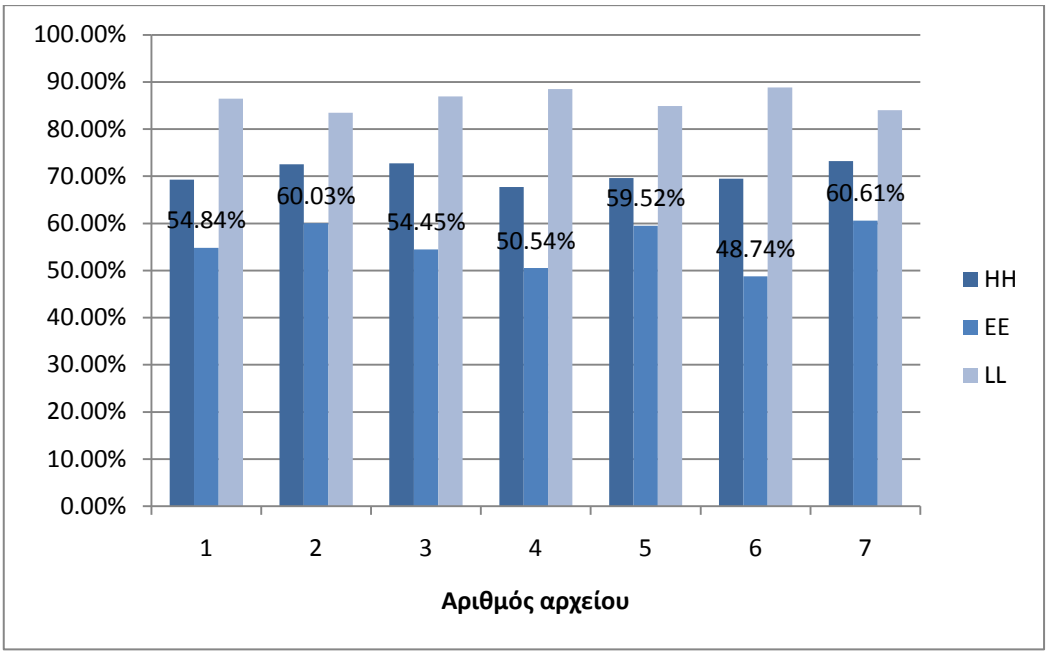
Ένα τυπικό παράδειγμα από τις βελτιωμένες ακολουθίες που έδωσε ο αλγόριθμος *viterbi*, φαίνονται στο σχήμα 5.32, για την πρωτεΐνη 1x22A.

```
1x22A Correctness Percentage:66.6667% Q3: 76.1904761904762
primaryStructure           :GKIPVKAIKKAGAAIGKGLRAINIASTAHDVYSFFKPKHKKK
secondaryStructure         :LLLLHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHLLLLLLL
predictedSecondaryStructure:LLELHHHHHHHHHHHHLLHHHHELHLLHLLLLLLLLLLLLLL
optimizedSecondaryStructure:LLLLHHHHHHHHHHHHHHHHHHHHLLLLLLLLLLLLLLLLL
```

Σχήμα 5.32: *Τυπικό αρχείο εξόδου του προγράμματος viterbi για την πρωτείνη 1x22A, που χαρακτηρίζεται από το optimizedSecondaryStructure. Η προβλεπόμενη ακολουθία από το δίκτυο φαίνεται από το predictedSecondaryStructure.*

Από το σχήμα 5.32 βλέπουμε ότι η προβλεπόμενη ακολουθία εξόδου του δικτύου, έχει προβλέψει την δευτεροταγή δομή του κάθε αμινοξέως χωρίς να λαμβάνει υπόψη της τις συσχετίσεις μεταξύ τους. Για παράδειγμα, δεν μπορούμε εκ φύσεως να έχουμε L H L, δηλαδή να προβλέπεται τμήμα από Helices με ένα H μόνο. Ως γνωστόν, ο μέσος όρος του μήκους ενός τμήματος από Helices είναι 10 (H) (Mount, 2001), συνεπώς, πολλά στοιχεία της πιο πάνω προβλεπόμενης δευτεροταγούς δομής που έδωσε το δίκτυο είναι λανθασμένα. Εντούτοις, ο αλγόριθμος *viterbi* εντοπίζει και διορθώνει τα αποτελέσματα αυτά, όπως φαίνεται και από το σχήμα 5.32. Όσον αφορά τα β – sheets, το μήκος ενός τμήματος είναι μεταξύ 5 – 10 συνεχόμενα (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).

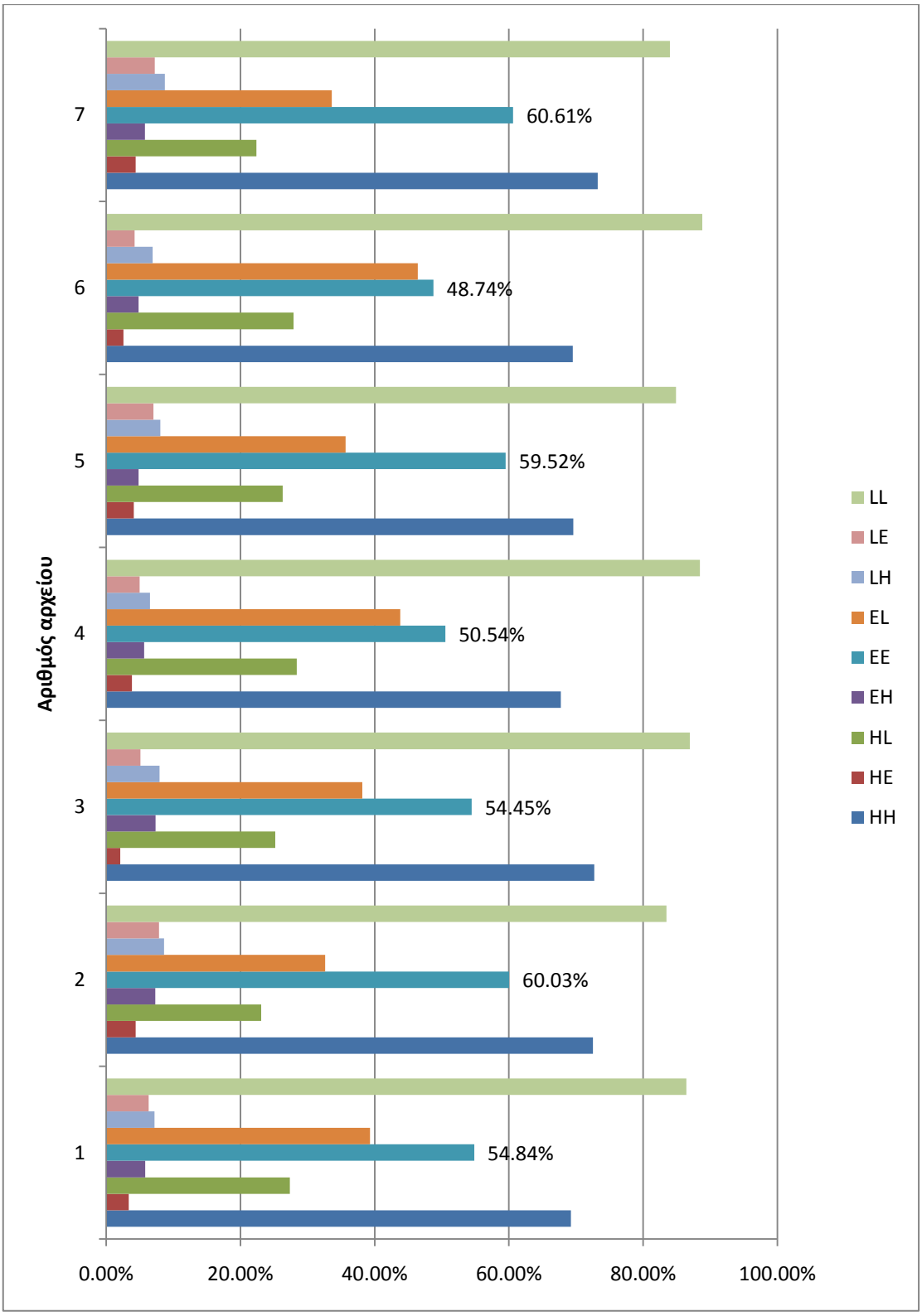
Στην συνέχεια θα δούμε πως βελτιώθηκαν οι προβλέψεις του δικτύου για τα συγκεκριμένα αποτελέσματα του πίνακα 5.10, χρησιμοποιώντας τα confusion matrices. Τα αποτελέσματα φαίνονται στα σχήματα 5.33 και 5.34.



Σχήμα 5.33: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα.*

Από τα σχήματα 5.33 και 5.34 βλέπουμε ότι τα Extended (E), τα οποία δεν μπορούσαν να προβλεφθούν σωστά για πάνω από 50%, όντως μπορούν και προβλέπονται σωστά μέχρι και 60%, χρησιμοποιώντας φιλτράρισμα της ακολουθίας εξόδου του δικτύου με την βοήθεια του αλγορίθμου *viterbi*. Αυτό είναι πολύ σημαντικό γιατί έχουμε λύσει ένα μικρό μέρος του προβλήματος πρόβλεψης των Extended (λόγω της σύγχυσης τους με την ομάδα των Loops), με την βοήθεια του διορθωτικού αλγορίθμου *viterbi*.

Παρότι βλέπουμε ότι το *φιλτράρισμα* που γίνεται από το HMM βελτιώνει τα αποτελέσματα στο SOV σκορ, χρειάζεται περαιτέρω ανάλυση των αποτελεσμάτων πριν και μετά το φιλτράρισμα, ώστε να δούμε ποιοτικά και ποσοτικά χαρακτηριστικά της βελτίωσης. Επιπλέον, δεν έχουμε με κάποιο τρόπο ενσωματώσει την δυνατότητα μοντελοποίησης της κατανομής των μηκών των διάφορων στοιχείων δευτεροταγούς κατανομής (duration modelling), πράγμα που αναμένουμε ότι θα βελτίωνε ακόμη περισσότερο την ποιότητα των αποτελεσμάτων.



Σχήμα 5.34: *Γραφική παράσταση που παρουσιάζει τον μέσο όρο πρόβλεψης H, E και L, όταν το επιθυμητό αποτέλεσμα είναι H, E και L αντίστοιχα καθώς και τις λανθασμένες προβλέψεις HE, HL, EH, EL, LE, LH, για τα αποτελέσματα του πίνακα 5.10, αφού φιλτραριστούν με την βοήθεια του αλγορίθμου viterbi.*

5.6 Γενικές παρατηρήσεις και συζήτηση

Αρχικά, πολύ σημαντική για την βελτίωση των ποσοστών ακρίβειας του συστήματος είναι η προσθήκη της πολλαπλής στοίχισης των πρωτεϊνών του συνόλου εκπαίδευσης και επαλήθευσης. Η πολλαπλή στοίχιση των ακολουθιών προσθέτει στο δίκτυο μια επιπρόσθετη εξελικτική πληροφορία και συνεπώς δίνει την δυνατότητα στο σύστημα να προβλέπει με μεγαλύτερη ακρίβεια την δευτεροταγή δομή της πρωτεΐνης (Rost and Sander, 1993).

Σημαντική βελτίωση του ποσοστού πρόβλεψης των πρωτεϊνών του συστήματος, είχε και η πρόσθεση της τυχαιοποίησης του συνόλου των δεδομένων εκπαίδευσης. Η τυχαιοποίηση όπως αναφέραμε στο υποκεφάλαιο 4.5, είναι μια ευρετική μέθοδος βελτίωσης της απόδοσης του δικτύου. Επιτελώντας σε κάθε εποχή ένα *ανακάτεμα* του συνόλου εκπαίδευσης, προσθέτει ένα είδος «θορύβου» στο δίκτυο το οποίο στην συγκεκριμένη περίπτωση βελτιώνει τα αποτελέσματα του.

Μια επίσης σημαντική παρατήρηση που πρέπει να γίνει αφορά την φύση του προβλήματος της πρόβλεψης δευτεροταγούς δομής των πρωτεϊνών με την χρήση νευρωνικών δικτύων αμφίδρομης ανάδρασης, για το οποίο υλοποιήθηκε το συγκεκριμένο σύστημα. Όπως έχουμε αναφέρει στο υποκεφάλαιο 1.2, κάθε πρόβλημα το οποίο προσπαθεί να αντιμετωπιστεί με την χρήση νευρωνικών δικτύων τα οποία προβλέπουν την έξοδο, μπορεί να θεωρηθεί σαν πρόβλημα παλινδρόμησης. Ως γνωστόν, σε προβλήματα παλινδρόμησης σημαντικό ρόλο στην επίδοση τους, παίζει η παρουσία γραμμικής συνάρτησης ενεργοποίησης στο επίπεδο εξόδου και μη γραμμικής στα υπόλοιπα επίπεδα. Θεωρώντας λοιπόν, ότι θα είχαμε καλύτερα αποτελέσματα, επεκτείναμε το σύστημα ώστε να δέχεται τον συνδυασμό αυτό. Δυστυχώς, τα αποτελέσματα δεν επαλήθευσαν την θεωρία αυτή (Σχήμα 5.12). Έτσι λοιπόν, μπορούμε να καταλήξουμε στο συμπέρασμα ότι το συγκεκριμένο πρόβλημα, δεν θεωρείται πρόβλημα παλινδρόμησης, αλλά είναι ένα καθαρό πρόβλημα κατηγοριοποίησης.

Η μεθοδολογία που ακολουθήθηκε στο υποκεφάλαιο 5.2.2 έδωσε σημαντικές παρατηρήσεις που αφορούν την εκπαίδευση και κατά συνέπεια τα αποτελέσματα του

δικτύου. Κατ' αρχήν, παρατηρήθηκε ότι κατά μέσο όρο τα μεγαλύτερου μεγέθους κινούμενα παράθυρα, έδιναν καλύτερα αποτελέσματα (Σχήμα 5.11). Αυτό, γιατί τα μεγαλύτερα παράθυρα μπορούν να συσχετίσουν μια μεγαλύτερη περιοχή από κατάλοιπα μιας πρωτεΐνης μεταξύ τους, κάτι που όπως γνωρίζουμε ισχύει και στην φύση, αφού οι πρωτεΐνες χαρακτηρίζονται από αλληλεπιδράσεις μεταξύ των καταλοίπων που την αποτελούν, οι οποίες δεν είναι απαραίτητα κοντινές. Έτσι, με το μεγαλύτερου μεγέθους κινητό παράθυρο παίρνουμε πιο μακρινές πληροφορίες που αφορούν τα αμινοξέα και τις συνδέσεις τους και συνεπώς έχουμε καλύτερα αποτελέσματα.

Βάσει των αποτελεσμάτων του υποκεφαλαίου 5.4, παρατηρούμε ένα βασικό πρόβλημα του δικτύου, που αφορά την πρόβλεψη των Strands (E). Το δίκτυο δεν μπορεί να προβλέψει σωστά την ομάδα των (E), όπως φαίνεται και από τα αποτελέσματα των confusion matrices (Σχήμα 5.19). Η ομάδα αυτή συγχέεται με την ομάδα των (L).

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. Πιθανές λύσεις είναι να αλλάξουμε το σύνολο δεδομένων με το οποίο εκπαιδεύουμε το δίκτυο, ώστε να βάλουμε πρωτεΐνες που να αντιπροσωπεύουν σε μεγαλύτερο βαθμό την ομάδα των Strands (E), ή τουλάχιστον να δημιουργήσουμε ένα σύνολο στο οποίο να αντιπροσωπεύονται όλες οι ομάδες δευτεροταγούς δομής που μελετούμε στον ίδιο βαθμό. 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 (σχήμα 5.19).

Εφαρμόζοντας την μέθοδο *winner takes all* (WTA), ώστε να λύσουμε κατά κάποιο τρόπο το πρόβλημα της κωδικοποίησης του δικτύου, δεν παρατηρείται καμία αλλαγή στο ποσοστό πρόβλεψης.

Στην συνέχεια αλλάξαμε την αρχιτεκτονική του δικτύου αυτού, με τέτοιο τρόπο ώστε να κρατήσουμε μόνο ένα κρυφό επίπεδο σε κάθε ένα από τα τρία ΤΝΔ από τα οποία αποτελείται το σύστημα αυτό (υποκεφάλαιο 5.6). Με την αλλαγή της αρχιτεκτονικής, τα αποτελέσματα πρόβλεψης με βάση την Q3 παράμετρο αυξήθηκαν μέχρι και 74.36% και με βάσει την SOV παράμετρο γύρω στο 60%. Το πρόβλημα της χαμηλής πρόβλεψης των (E) συνέχισε να υφίστανται παρόλο που το Q3 ποσοστό είχε αυξηθεί. Όταν τροποποιήσαμε το σύστημα ώστε η κωδικοποίηση της εξόδου να γίνεται με την μέθοδο *winner takes all*, τα αποτελέσματα πρόβλεψης έφτασαν το 76% με βάσει την Q3 παράμετρο και 65% βάσει της SOV παραμέτρου (υποκεφάλαιο 5.6.1).

Η εφαρμογή του φιλτραρίσματος της προβλεπόμενης ακολουθίας από το δίκτυο έγινε με την χρήση του αλγορίθμου *viterbi* (υποκεφάλαιο 5.7.1). 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), αλλά και τις μακρινές αλληλεπιδράσεις των αμινοξέων. Αυτό εφαρμόστηκε πάνω στις καλύτερες προβλέψεις που είχαμε μέχρι τώρα, με το καλύτερο SOV αποτέλεσμα να είναι 70%.

Η χρήση των γενετικών αλγορίθμων (ΓΑ), η οποία υλοποιήθηκε και χρησιμοποιήθηκε αρχικά από τον Αγαθοκλέους (Αγαθοκλέους, 2009), δεν έδωσε κάποια καλύτερα

αποτελέσματα για το σύστημα, συμπεριλαμβανομένου του MSA, όπως αναμέναμε, αλλά έδωσε κάποια παράξενα αποτελέσματα. Αυτό πιστεύεται ότι οφείλεται σε μια ασάφεια όσον αφορά την υλοποίηση τους. Παρόλα αυτά, οι γενετικοί αλγόριθμοι τροποποιήθηκαν κατάλληλα για επίλυση της ασάφειας αυτής όπως φαίνεται και στο παράρτημα Γ.

Νέες προσομοιώσεις με την χρήση των ΓΑ ήδη εκτελούνται, αλλά λόγω του χρόνου εκτέλεσης, δεν θα παρουσιαστούν τα τελικά αποτελέσματα στην παρούσα εργασία. Οι εκτελέσεις που εκτελούνται την συγκεκριμένη στιγμή δίνουν μέχρι τώρα Q3 ποσοστά α) 76% για αρχιτεκτονική δικτύου με ένα κρυφό επίπεδο για κάθε ΤΝΔ και χρήση *WTA* για την κωδικοποίηση εξόδου, β) 75% για αρχιτεκτονική με τα δυο κρυφά επίπεδα για κάθε ΤΝΔ και χρήση *WTA*, γ) 73% για αρχιτεκτονική δικτύου με ένα κρυφό επίπεδο για κάθε ΤΝΔ και χωρίς την χρήση *WTA* για την κωδικοποίηση εξόδου και δ) 75% για μια αρχιτεκτονική με δυο κρυφά επίπεδα για κάθε ΤΝΔ, και χωρίς *WTA* για το επίπεδο εξόδου. Οι προσομοιώσεις στο παρόν στάδιο, βρίσκονται και οι τέσσερεις στην δεύτερη γενεά του γενετικού αλγορίθμου. Τα πιο πάνω αποτελέσματα είναι τα καλύτερα αποτελέσματα του δικτύου κατά την πρώτη γενεά των ΓΑ.

Κεφάλαιο 6

Συμπεράσματα και μελλοντική εργασία

6.1 Συμπεράσματα

6.2 Μελλοντική εργασία

6.1 Συμπεράσματα

Η τρέχουσα διπλωματική εργασία αφορά την πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών με χρήση νευρωνικών δικτύων αμφίδρομης ανάδρασης, μιας αρχιτεκτονικής νευρωνικών δικτύων που εισήγαγε πρώτος στον χώρο της έρευνας αυτής ο Baldi (Baldi et al., 1999,Baldi et al., 2000), και για την οποία έχει σημαντικά αποτελέσματα να παρουσιάσει. Αυτό, γιατί η αρχιτεκτονική αυτή «αντιπροσωπεύει» το πρόβλημα σε όλο του το φάσμα. Παρόλα αυτά, το τρέχον σύστημα, κρατώντας την ιδέα της αρχιτεκτονικής του Baldi, παρουσιάζει αρκετές διαφορές ως προς την υλοποίηση του, τον τρόπο επεξεργασίας των δεδομένων του, και γενικά τον αλγόριθμο εκπαίδευσης που εκτελείται (υποκεφάλαιο 2.3). Οι διαφορές αυτές, κάνουν το τρέχον σύστημα ένα ξεχωριστό σύστημα αφού *εσωτερικά* επεξεργάζεται τις πληροφορίες με ένα εντελώς διαφορετικό τρόπο. Ο τρόπος αυτός, εκτός φυσικά της εκτεταμένης πολυπλοκότητας του, πιστεύουμε ότι δίνει περισσότερη πληροφορία στο δίκτυο, και αναγκάζει το δίκτυο να μαθαίνει την τοπική πληροφορία πιο αποτελεσματικά.

Τα αποτελέσματα από την εκπαίδευση του συγκεκριμένου συστήματος μπορούμε να πούμε ότι είναι ικανοποιητικά. Απαραίτητη θεωρείται η χρήση των MSA profiles, αφού εισάγουν ένα είδος εξελικτικής πληροφορίας στο δίκτυο, με αποτέλεσμα το δίκτυο να μπορεί να γενικεύει καλύτερα. Το Q3 σκορ για τα διάφορα πειράματα που διεξήχθησαν με MSA profiles αλλά και με την τεχνική της τυχαιοποίησης είναι αρκετά υψηλό, γεγονός που δείχνει την αποτελεσματικότητα του συγκεκριμένου δικτύου, με μεγαλύτερο σκορ 73.9%. Σε σύγκριση με τα αποτελέσματα του Baldi, τα οποία κυμαίνονται από 73.2% με 73.6%, τα αποτελέσματα του δικτύου είναι ελαφρώς καλύτερα. Ένα σημαντικό γεγονός που παρατηρείται για την απόδοση του τρέχοντος δικτύου, είναι ότι εκπαιδεύεται καλύτερα όταν το μέγεθος του κινούμενου παραθύρου είναι μεγάλο. Αυτό, οφείλεται στον τρόπο επεξεργασίας των δεδομένων του δικτύου, και μπορούμε να πούμε, ότι λόγω αυτής της ιδιότητας το δίκτυο μπορεί να παίρνει και να επεξεργάζεται περισσότερη πληροφορία σε κάθε εκπαίδευση του.

Επίσης οι γενικές παράμετροι της ορμής, των χρονικών μεταβλητών, του ρυθμού μάθησης, ακόμα και η τεχνική της τυχαιοποίησης του συνόλου εκπαίδευσης δεν εγγυώνται κάποια βέλτιστη τιμή για το δίκτυο, αφού με βάση τα αποτελέσματα που πάρθηκαν κυμαίνονται μεταξύ μεγάλου φάσματος τιμών, και αυτό θα προσδιοριστεί και θα ξεκαθαριστεί μόνο με την χρήση στατιστικών σφαλμάτων.

Κάθε πρόβλημα το οποίο προσπαθεί να αντιμετωπιστεί με την χρήση νευρωνικών δικτύων τα οποία προβλέπουν την έξοδο, μπορεί να θεωρηθεί σαν πρόβλημα παλινδρόμησης. Ως γνωστόν, σε προβλήματα παλινδρόμησης σημαντικό ρόλο στην επίδοση τους, παίζει η παρουσία γραμμικής συνάρτησης ενεργοποίησης στο επίπεδο εξόδου και μη γραμμικής στα υπόλοιπα επίπεδα. Ωστόσο, εφαρμογές των συναρτήσεων υπερβολικής εφαπτομένης σε συνδυασμό με την γραμμική συνάρτηση ενεργοποίησης στο τρέχον δίκτυο απέτυχαν, ενώ περιμέναμε να μας δώσουν ένα καλό αποτέλεσμα. Έτσι μπορούμε να πούμε ότι το πρόβλημα αυτό δεν ανήκει στα προβλήματα παλινδρόμησης αλλά, στα προβλήματα κατηγοριοποίησης.

Φυσικά, παρουσιάζεται ένα σημαντικό πρόβλημα, το πρόβλημα του ποσοστού πρόβλεψης των επιθυμητών Strands (E) στο σύστημα. Πιο συγκεκριμένα το δίκτυο στην συγχύζει την ομάδα αυτή με τις υπόλοιπες ομάδες δευτεροταγούς δομής. Αυτό είναι ένα πολύ *λογικό*, αν όχι και αναμενόμενο πρόβλημα για τέτοια δίκτυα, επειδή 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). Είναι πιθανόν λοιπόν τα Strands (E) 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 υπόλοιπες ομάδες.

Σημαντική παρατήρηση για το δίκτυο είναι ότι με την αλλαγή της αρχιτεκτονικής (υποκεφάλαιο 5.6) τα αποτελέσματα πρόβλεψης με βάση την Q3 παράμετρο αυξήθηκαν μέχρι και 74.36% και με βάσει την SOV παράμετρο γύρω στο 60%. Η αλλαγή της αρχιτεκτονικής αυτής αφήνει μόνο ένα κρυφό επίπεδο στο κάθε ΤΝΔ από τα τρία τα οποία αποτελείται το τρέχον σύστημα. Παρόλα αυτά, το πρόβλημα της χαμηλής πρόβλεψης των (Ε) συνέχισε να υφίστανται.

Εφαρμόζοντας στο σύστημα την μέθοδο *winner takes all* (WTA), για το επίπεδο εξόδου, τα αποτελέσματα πρόβλεψης έφτασαν το 76% με βάσει της Q3 παραμέτρου και 65% βάσει της SOV παραμέτρου (υποκεφάλαιο 5.6.1).

Επιπρόσθετη επεξεργασία των αποτελεσμάτων εξόδου του δικτύου, αποτέλεσε η εφαρμογή του φιλτραρίσματος της προβλεπόμενης ακολουθίας που δίνει το δίκτυο. Αυτό έγινε με την χρήση του αλγορίθμου *viterbi* (υποκεφάλαιο 5.7.1). Το *φιλτράρισμα* αποσκοπεί να διορθώσει την προβλεπόμενη ακολουθία 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), αλλά και τις μακρινές αλληλεπιδράσεις των αμινοξέων. Αυτό εφαρμόστηκε πάνω στις καλύτερες προβλέψεις που είχαμε μέχρι τώρα, με το καλύτερο SOV αποτέλεσμα να είναι 70%.

Η χρήση των γενετικών αλγορίθμων (ΓΑ) σε συνδυασμό και με την εφαρμογή του MSA, η οποία υλοποιήθηκε και χρησιμοποιήθηκε αρχικά από τον Αγαθοκλέους (Αγαθοκλέους, 2009), δεν έδωσε καλύτερες τιμές για το σύστημα όπως αναμέναμε. Επειδή όμως παρουσιάστηκε ένα κομμάτι της υλοποίησης που παραβίαζε τους βασικούς περιορισμούς του συστήματος, έγινε η διόρθωση των κατάλληλων κλάσεων (Παράρτημα Γ), ώστε να λειτουργεί σύμφωνα με τους περιορισμούς.

Νέες προσομοιώσεις με την χρήση των ΓΑ ήδη εκτελούνται, αλλά λόγω του χρόνου εκτέλεσης, δεν θα παρουσιαστούν τα τελικά αποτελέσματα στην παρούσα εργασία. Τα βέλτιστα μέχρι τώρα αποτελέσματα για Q3 ποσοστά είναι α) 76% για αρχιτεκτονική δικτύου με ένα κρυφό επίπεδο για κάθε ΤΝΔ και χρήση *WTA* για την κωδικοποίηση εξόδου, β) 75% για αρχιτεκτονική με τα δυο κρυφά επίπεδα για κάθε ΤΝΔ και χρήση *WTA*, γ) 73% για αρχιτεκτονική δικτύου με ένα κρυφό επίπεδο για

κάθε ΤΝΔ και χωρίς την χρήση *WTA* για την κωδικοποίηση εξόδου και δ) 75% για μια αρχιτεκτονική με δυο κρυφά επίπεδα για κάθε ΤΝΔ, και χωρίς *WTA* για το επίπεδο εξόδου. Οι προσομοιώσεις στο παρόν στάδιο, βρίσκονται και οι τέσσερεις στην δεύτερη γενεά του γενετικού αλγορίθμου. Τα πιο πάνω αποτελέσματα είναι τα καλύτερα αποτελέσματα του δικτύου κατά την πρώτη γενεά των ΓΑ.

6.2 Μελλοντική εργασία

Πολύ σημαντική είναι η εισαγωγή ενός δεύτερου νευρωνικού δικτύου, το οποίο σε γενικές γραμμές, θα παίρνει τα αποτελέσματα του τρέχοντος συστήματος και κατά κάποιο τρόπο, θα τα *βελτιώνει* (Qian and Sejnowski, 1988), (Rost and Sander, 1993). Παρόμοια μέθοδος έχει ήδη περιγραφεί και εφαρμοστεί στο υποκεφάλαιο 5.7.1, όμως γίνεται με την χρήση *Hidden Markov Models* (HMM). Θέλουμε όμως να δοκιμάσουμε να ενσωματώσουμε το δεύτερο νευρωνικό δίκτυο όπως περιγράφεται από τους Chen and Chaudhari (Chen and Chaudhari, 2007). Με την εισαγωγή του δεύτερου αυτού δικτύου θα σχηματίσουμε ένα σύστημα το οποίο θα αποτελείται από δυο επιμέρους υποσυστήματα νευρωνικών δικτύων. Το πρώτο δίκτυο θα είναι το υποσύστημα το οποίο θα είναι υπεύθυνο να παίρνει την πρωτοταγή δομή μιας πρωτεΐνης και να προβλέπει την δευτεροταγή δομή της. Αυτό αποτελεί το τυπικό σύστημα που έχει υλοποιηθεί μέχρι στιγμής. Το δεύτερο δίκτυο, θα αντιπροσωπεύει το δεύτερο υποσύστημα, το οποίο θα είναι υπεύθυνο να παίρνει την προβλεπόμενη έξοδο του πρώτου και να την *διορθώνει*. Φυσικά, δεν είναι απαραίτητο ότι το δεύτερο νευρωνικό δίκτυο θα πρέπει κατ' ανάγκη να ακολουθεί την αρχιτεκτονική του πρώτου δικτύου. Αντιθέτως, μπορεί να υλοποιηθεί αρκετά απλά, με ένα απλό δίκτυο εμπρόσθιου περάσματος, χρησιμοποιώντας ένα απλό αλγόριθμο εκπαίδευσης *BackPropagation* (υποκεφάλαιο 1.3.3). Το υποσύστημα αυτό θα εκπαιδεύεται κατάλληλα χρησιμοποιώντας τα δεδομένα εξόδου του πρώτου υποσυστήματος.

Σημαντικό είναι να υπολογιστούν και να παρουσιαστούν στατιστικά σφάλματα των διάφορων σχετιζόμενων πειραμάτων που εκτελέστηκαν στο κεφάλαιο 5.

Όσον αφορά τους γενετικούς αλγορίθμους (ΓΑ) οι οποίοι έχουν αναφερθεί στο υποκεφάλαιο 4.6, θα μπορούσαμε να χρησιμοποιήσουμε πολλές τεχνικές βελτιστοποίησης τους. Αρχικά, πρέπει να χρησιμοποιηθούν νέες μέθοδοι διασταύρωσης. Στο τρέχον πρόγραμμα υλοποιούνται απλά οι αρχικές ιδέες διασταύρωσης και αναπαραγωγής, αλλά αυτό δεν σημαίνει ότι δίνουν την βέλτιστη λύση για τον αλγόριθμο αυτό, ή ότι είναι οι πιο αποτελεσματικές. Υπάρχουν πολλές νέες μέθοδοι αναπαραγωγής και διασταύρωσης, πιο συγκεκριμένες, ώστε να επιτυγχάνεται μια μεγαλύτερη πιθανότητα να πάρουμε «καλύτερα» άτομα, τα οποία με την σειρά τους θα κωδικοποιήσουν λύσεις πιο κοντά στην βέλτιστη λύση του προβλήματος. Επίσης, θα πρέπει να εκτελεστούν επιπρόσθετα πειράματα με την χρήση ΓΑ, επειδή δεν υπάρχουν συγκεκριμένα συμπεράσματα για την χρήση τους.

Το σύστημα στην τρέχουσα κατάστασή του αντιμετωπίζει κάποια προβλήματα, όπως έχουμε προαναφέρει στο υποκεφάλαιο 5.6. Το πιο κρίσιμο πρόβλημα και ίσως το πιο σημαντικό αφορά την πρόβλεψη της δευτεροταγούς ομάδας των (E). Φυσικά, το πρόβλημα όπως είπαμε, μπορεί να οφείλεται σε πολλούς λόγους όπως για παράδειγμα ότι ο αριθμός των πρωτεϊνών με περιοχές (E) δεν είναι τόσο αντιπροσωπευτικός σε σχέση με τις άλλες ομάδες δευτεροταγούς δομής όπως είναι τα (L) και (H). Επειδή αυτό το πρόβλημα ίσως να οφείλεται στην αρχιτεκτονική του δικτύου, προτείνεται να γίνουν κάποιες αλλαγές για να διαπιστώσουμε κατά πόσο αυτό ισχύει λόγω της αρχιτεκτονικής του επιπέδου εξόδου σε συνδυασμό με την τρέχουσα κωδικοποίηση. Σαν μια πρώτη εφαρμογή, το επίπεδο εξόδου θα μπορούσε να το αποτελεί ένας μόνο νευρώνας αντί τρεις όπως είναι την συγκεκριμένη στιγμή. Με τον τρόπο αυτό, αντιμετωπίζουμε το πρόβλημα των «επιπλέον» συνδυασμών που δίνει η κωδικοποίηση με τρεις νευρώνες εξόδου. Φυσικά, εφαρμόζοντας ένα νευρώνα στο επίπεδο εξόδου, θα πρέπει να γίνουν αλλαγές όσον αφορά το πεδίο τιμών που αντιπροσωπεύουν την κάθε ομάδα δευτεροταγούς δομής. Στην δεύτερη εφαρμογή, ο σχεδιασμός του επιπέδου εξόδου αλλάζει και πάλι συμπεριλαμβάνοντας δύο νευρώνες μόνο, έναντι τριών που προϋπάρχουν. Με δύο νευρώνες, έχουμε μια παρόμοια κωδικοποίηση όπως την τρέχουσα, απλά αντί για οχτώ καταστάσεις, έχουμε μόνο τέσσερεις. Οι ομάδες

δευτεροταγούς δομής όμως, είναι μόνο τρείς. Άρα οι βασικές αλλαγές που θα συμπεριλαμβάνει η υλοποίηση αυτής της μεθόδου, θα είναι μόνο η επιλογή της ομάδας που κωδικοποιεί ο τέταρτος συνδυασμός. Τα υπόλοιπα βασίζονται στην τρέχουσα κωδικοποίηση. Πιθανόν ο τέταρτος συνδυασμός να ανήκει και πάλι στην ομάδα των (L). Φυσικά, ένα μειονέκτημα της αλλαγής αυτής είναι ότι μειώνεται η ικανότητα κωδικοποίησης της εξόδου του δικτύου.

Εξαιτίας του προβλήματος που παρουσιάστηκε στην τρέχουσα διπλωματική εργασία όσον αφορά την χαμηλή πρόβλεψη των Extended, θέλουμε να εφαρμοστεί μια μέθοδος ανάλυσης και επεξεργασίας δεδομένων, ώστε να μπορεί να ισοζυγίζει κατά κάποιο τρόπο τα σύνολα των πρωτεϊνών που χρησιμοποιούνται στο πρόγραμμα. Αυτό σίγουρα θα δίνει το ίδιο *βάρος* σε όλες τις ομάδες δευτεροταγούς δομής και πιθανόν να αυξηθεί τόσο το ποσοστό Q3, αλλά και το ποσοστό SOV.

Μια τακτική που ακολουθήθηκε από τον Baldi (Baldi et al., 1999), για να πάρει τα καλύτερα ποσοστά του, ήταν να πάρει ένα σύνολο από δίκτυα και να δημιουργήσει ένα ποσοστό πρόβλεψης βασισμένο στα αποτελέσματα τους. Έλαβε υπόψη του έξι δίκτυα τα οποία χρησιμοποιούσαν MSA profiles για την κωδικοποίηση της εισόδου τους. Τα δίκτυα αυτά δεν είχαν απαραίτητα την ίδια αρχιτεκτονική δομή, δηλαδή οι παράμετροι τους ήταν διαφορετικές. Τα αποτελέσματα τους κυμαίνονταν από 73.2% μέχρι 73.6%. Το καλύτερο ποσοστό πρόβλεψης δευτεροταγούς δομής με την συνένωση αποτελεσμάτων έξι δικτύων και με την χρήση 7-fold cross validation ήταν 75.1%. Στην παρούσα διπλωματική εργασία το σύστημα ενός δικτύου, δίνει Q3 αποτελέσματα μέχρι και 76%. Προτείνουμε λοιπόν, να εφαρμοστεί μια παρόμοια μέθοδος και στο τρέχον σύστημα, όπου θα παίρνουμε ένα σύνολο από δίκτυα που έχουν ένα καλό ποσοστό, τόσο σε βαθμό Q3 όσο και σε SOV, και θα υπολογίζουμε συνολικό αντιπροσωπευτικό αποτέλεσμα των δικτύων αυτών χρησιμοποιώντας *negative correlation learning* (Liu and Yao, 1999). Πιστεύεται ότι η εφαρμογή αυτή θα δώσει πολύ καλά αποτελέσματα, συγκρίσιμα με τα αποτελέσματα του Baldi (Baldi et al., 1999), ίσως ακόμη και περισσότερο ακριβή.

Τέλος, χρησιμοποιώντας την πολλαπλή στοίχιση των πρωτεϊνών από τα οποία παίρνουμε τα MSA profiles, θα εφαρμοστεί μια μέθοδος η οποία: α) θα εντοπίζει τις

βασικές συσχετίσεις ανάμεσα στα αμινοξέα κάθε μιας ακολουθίας που έλαβε μέρος στην πολλαπλή στοίχιση, β) διαχωρίζει τις ακολουθίες της πολλαπλής στοίχισης με βάση τις συσχετίσεις αυτές, γ) υπολογίζει ένα βάρος για την κάθε ακολουθία, δ) διαχωρίζει τις ακολουθίες αυτές για δημιουργία νέου συνόλου δεδομένων.

Αναφορές

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

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, pp.937-946, 1999.

Baldi P., Brunak S., Frasconi P., Soda G. and Pollastri G. "Bidirectional Dynamics for Protein Secondary Structure Prediction", Sequence Learning, R. Sun and L. Giles Editors, Springer Verlag, LNAI 1828, pp. 80-104, 2000.

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.

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.

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

HSSP, February 10, 2010, http://www.sander.embl-heidelberg.de/

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.

Mount D. “Bioinformatics: Sequence and Genome Analysis”, Cold Spring Harbor, Laboratory Press, 2001.

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

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.

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., Sander C. and Schneider R. “Redefining the goals of protein secondary structure prediction”, J Mol Biol, Vol. 235, pp. 13–26, 1994.

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.

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.

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: Structure, Function, and Genetics Vol. 34, pp. 220–223, 1999.

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.

Gradient Descent, May 20, 2009, http://en.wikipedia.org/wiki/Gradient_descent

ΠΑΡΑΡΤΗΜΑ Α

Νευρωνικό Δίκτυο Αμφίδρομης Ανάδρασης

pssp_ucy.cpp

```
//included libraries

#include "DataReader.h"
#include "Neuron.h"
#include "BidirectionalRecurrentNeuralNetwork.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;

	//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,testFile,
&msaEnable,&randomizeDataset);

	//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
BRNN=BidirectionalRecurrentNeuralNetwork(hLayerOneSize,

	hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,

	hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutput,le
arningRate,momentum,windowSize,

	qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,s,ma
xIterations,trainFile,testFile,
			inputProfile,outputProfile,msaEnable,randomizeDataset);


	//the main loop of the program.
	while(epoch<maxIterations)
	{
		//open pointers to the output files
		BRNN.openFolders();

		//takes the first protein of the training set
		endOfFile=BRNN.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)
				{BRNN.getMSAInput();}

			for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
			{
				//processing
				BRNN.doFeedForward(sequencet,1,epoch);
```

```
                        BRNN.doBackpropagation(sequencet,learningRate,1);
                  }


                  //pernoume to error gia tin protein pou isoute me to
sinoliko lathos/ton arithmo twn aa pou exei i proteini
                  BRNN.getSequenceTrainingError();

                  if(epoch==maxIterations-1)
                        BRNN.printOutputSequence(1);

                  //takes the next protein from the training file

      endOfFile=BRNN.initTrainingInput(&primaryStructureSize);

            }


            //
            endOfFile=BRNN.initTestInput(&primaryStructureSize);

            //
            while(endOfFile)
            {

                  //takes the new msa encoding of this protein
                  if(msaEnable)
                        {BRNN.getMSAInput();}


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

                        BRNN.doFeedForward(sequencet,2,epoch);

                  }

                  BRNN.getSequenceTestingError();

                  if(epoch==maxIterations-1)
                        BRNN.printOutputSequence(2);

                  endOfFile=BRNN.initTestInput(&primaryStructureSize);
            }

            //
            BRNN.getEpochError();

            //
            BRNN.closeFolders();
            cout<<epoch<<endl;
            epoch++;
      }
```

```
        BRNN.printOutputs();
        BRNN.printNetworkSpecification();

        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> secondaryStructure;
	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;

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

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;
        DataReader* getInput;

        //ofstream printOutputValues;

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

public:

        BidirectionalRecurrentNeuralNetwork(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);

        ~BidirectionalRecurrentNeuralNetwork(void);

        //Methods of BidirectionalRecurrentNeuralNetwork class
        bool initTrainingInput(int *primaryStructureSize);
        bool initTestInput(int *primaryStructureSize);
        void doFeedForward(int sequencet,int isTrainOrTest,int epoch);
        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);
        void getTargetOutputs(int sequencet);
        void printNetworkSpecification();
        void getMSAInput();
        void createMSAtempInput (int p, int q, int size);
};
#endif
```

BidirectionalRecurrentNeuralNetwork.cpp

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

using namespace std;

//*********************************************************************
*****************
//Constructor
BidirectionalRecurrentNeuralNetwork::BidirectionalRecurrentNeuralNetwork
//                (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::BidirectionalRecurrentNeuralNetwork(
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)
{
      //private objects of BidirectionalRecurrentNeuralNetwork to read
from files and write to files
      getInputData=new DataReader();
      saveOutputData=new OutputData();
      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';
	}



	//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,activat
ionFuncTypeHidden,activationFuncTypeOutput,errorFunctionTypeHidden,errorF
unctionTypeOutput,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,acti
vationFuncTypeHidden,activationFuncTypeOutput,errorFunctionTypeHidden,err
orFunctionTypeOutput,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(),momentum,le
arningRate,activationFuncTypeHidden,activationFuncTypeOutput,errorFunctio
nTypeHidden,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,learningRate,ac
tivationFuncTypeHidden,activationFuncTypeOutput,errorFunctionTypeHidden,e
rrorFunctionTypeOutput,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(),momentum,le
arningRate,activationFuncTypeHidden,activationFuncTypeOutput,errorFunctio
nTypeHidden,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,learningRate,ac
tivationFuncTypeHidden,activationFuncTypeOutput,errorFunctionTypeHidden,e
rrorFunctionTypeOutput,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+hLayerOne
SizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,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+hLayerTwo
SizeB,momentum,learningRate,activationFuncTypeHidden,activationFuncTypeOu
tput,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+hLayerOne
SizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,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+hLayerTwo
SizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,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+hLayerOne
SizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,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+hLayerTwo
SizeB,momentum,

	learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,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+hLayerOne
SizeB,momentum,

      learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,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+hLayerTwo
SizeB,momentum,

      learningRate,activationFuncTypeHidden,activationFuncTypeOutput,erro
rFunctionTypeHidden,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::~BidirectionalRecurrentNeuralNetwork
(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,outpu
tProfile);
}
//
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(primaryStructure,secondaryStructure,proteinID);
	if(endOfFile){

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

		return endOfFile;
	}
	else { return endOfFile;}
}

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

	endOfFile=getInputData-
>readTestInput(primaryStructure,secondaryStructure,proteinID);
	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::doFeedForward(int sequencet,int
isTrainOrTest,int epoch)
{

	int tempsize=(primaryStructure.size());
```

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

		for(int p=forwardt-halfInputSize;p<(forwardt-
halfInputSize+inputSize);p++)
		{
			if(p<0 || p>= tempsize){
				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 (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

	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 p=sequencet-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++)
                  {

      It.push_back((double)encodingT[primaryStructure[p]-65][q]-48);
                  }
            }

      }

      //BNNF
      for(int
backwardt=sequencet+halfWindow;backwardt>=sequencet;backwardt--)
      {
            tempInput.clear();
            for(int p=backwardt-halfInputSize;p<(backwardt-
halfInputSize+inputSize);p++)
            {
                  if((p) >= tempsize || p<0)
                  {
                        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

      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]);
            }


            int i=0;
            int counterEnc=0;
            int x;

            //step function - remove the /* .. */ to enable 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;
                  }
            }*/

            //WTA
            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
                  {
                        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 - remove the /* .. */ to enable 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;
                  }
            }*/

            //WTA
            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
                  {
                        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)/(doub
le)Ot.size()/primaryStructure.size());
      trainingSetAdditionVectorTemp.push_back(sqrt(trainingSetAddition)/(
double)Ot.size()/primaryStructure.size());
      trainingSetAdditionVectorTempSize=trainingSetAdditionVectorTemp.siz
e();
      trainingSetAddition=0;
}

void BidirectionalRecurrentNeuralNetwork::getSequenceTestingError(){

      testSetAdditionVector.push_back(sqrt(testSetAddition)/(double)Ot.si
ze()/primaryStructure.size());
      testSetAdditionVectorTemp.push_back(sqrt(testSetAddition)/(double)O
t.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)training
SetAdditionVectorTempSize);

      tempValue=0;

      for(int i=0;i<testSetAdditionVectorTempSize;i++)
      {
            tempValue+=testSetAdditionVectorTemp[i];
      }

      testSetAdditionEpochVector.push_back(tempValue/(double)testSetAddit
ionVectorTempSize);

      trainingSetAdditionVectorTemp.clear();
      testSetAdditionVectorTemp.clear();
```

```
}


void BidirectionalRecurrentNeuralNetwork::printOutputSequence(int
trainOrTest)
{

      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.siz
e();
      saveOutputData-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecondaryS
tructure,proteinID,tempPercentage,2);
      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.siz
e();
      saveOutputData-
>createOutputFile(primaryStructure,secondaryStructure,predictedSecondaryS
tructureTrain,proteinID,tempPercentage,1);
      percentageCounterTrain++;
      percentageTrain+=tempPercentage;
      predictedSecondaryStructureTrain.clear();
      }
}

void BidirectionalRecurrentNeuralNetwork::printNetworkSpecification()
{

      saveOutputData-
>createNetworkFile(hLayerOneSize,hLayerTwoSize,hLayerOneSizeB,hLayerTwoSi
zeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,learningRate,momentum,windo
wSize,
```

```
	qMinus1,qPlus1,errorFunctionTypeHidden,s,maxIterations,trainFile,te
stFile,

	inputProfile,outputProfile,percentage/(double)percentageCounter);

}

void BidirectionalRecurrentNeuralNetwork::printOutputs()
{
	saveOutputData-
>createErrorFiles(trainingSetAdditionEpochVector,testSetAdditionEpochVect
or,

	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+hLayerOneSiz
eF));
                        }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+hLayerOneSiz
eF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF+hLa
yerOneSize);
                        }
                  }
            }


            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+hLayerOneSiz
eF));
                        }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+hLayerOneSiz
eF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF+hLa
yerOneSize);
```

```
                        }
                  }
            }

            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+hLayerOneSiz
eF));
                        }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+hLayerOneSiz
eF+hLayerTwoSize));
                        }else{
```

```
      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF+hLa
yerTwoSize);
                        }
                  }
            }


            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+hLayerOneSiz
eF));
                        }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+hLayerOneSiz
eF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerOneSizeF+hLa
yerTwoSize);
                        }
                  }
            }

            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+hLayerTwoSiz
eF));
                                }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+hLayerTwoSiz
eF+hLayerOneSize));
                                }else{

        tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF+hLa
yerOneSize);
                                }
                        }
                }


                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+hLayerTwoSiz
eF));
                        }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+hLayerTwoSiz
eF+hLayerOneSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF+hLa
yerOneSize);
                        }
                  }
            }

            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+hLayerTwoSiz
eF));
                        }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+hLayerTwoSiz
eF+hLayerTwoSize));
                        }else{

      tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF+hLa
yerTwoSize);
                        }
```

```
                  }
            }

            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+hLayerTwoSiz
eF));
                        }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+hLayerTwoSiz
eF+hLayerTwoSize));
                        }else{
```

```
	tempVector[i]+=outputLayer[j].getSpecificError(i+hLayerTwoSizeF+hLa
yerTwoSize);
                        }
                  }
            }///

            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);
            }
      }
}
```

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*previousA
djustment[i];

      previousAdjustment[i]=learningRate*inputVector[i]*error+momentum*pr
eviousAdjustment[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*biasPreviousAd
justment;
      bias=-bias+Dbias;
      bias=-bias;*/

      //Dbias=0;
      //biasPreviousAdjustment=0;

}
```

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);
```

```
      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)
{
      //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<25;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==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 readOutputEncoding(vector<string> &encodingT,vector<vector<char>>
&outputClassification,
//		int *outputSize,const char outputProfile[100]):
//public method that is called to read the file with output encoding of a
protein's secondary
//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.
Alse a second vector
//is returned with the classification of the secondary structure letters.
//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
//          vector<vector<char>> &outputClassification      :returns 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
//          int *outputSize
      :returns how many classes
//          const char outputProfile[100]                   :the name of
output profile file name
//*************************************************************************
*****************
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;

      primaryStructure.clear();

      inTrainFile >> name;

      for(int i=0;i<100;i++) {proteinID[i]='\0';}

      strcat(proteinID,name);

      //if end of file returns false
      if(inTrainFile.eof()) { return false;}

      inTrainFile >> temp;

      //reads all the letters of primary structure until the "."
      while(temp!='.')
      {
            primaryStructure.push_back(temp);
            inTrainFile >> temp;
      }

      secondaryStructure.clear();

      inTrainFile >> temp;

      //reads all the letters of secondary structure until the "."
      while(temp!='.')
      {
            secondaryStructure.push_back(temp);
            inTrainFile >> temp;
      }

      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;
}

//*********************************************************************
*****************
//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/");
	strcat(temp,proteinID);
	strcat(temp,".txt");


	//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));

      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;
	char folderName[100];
	char HTMLName[100];

public:

	OutputData(void);
	~OutputData(void);

	//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);
      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 closeAllPointers(void);
};
#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(void)
{
	//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;

    time ( &rawtime );
	timeinfo = localtime ( &rawtime );

	char temp[100];char temp3[100];

	for(int i=0;i<100;i++)
	{
		temp[i]='\0';
		temp3[i]='\0';
		folderName[i]='\0';
		HTMLName[i]='\0';
	}

	strcat(temp,"DataOut//");
	strcat(temp,asctime (timeinfo));
	strcat(temp3,"DataOut//");
	strcat(temp3,asctime (timeinfo));
	strcat(HTMLName,asctime(timeinfo));

	//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,"_Output.txt");
	strcat(temp3,"_OutputForTrain.txt");

	//initiate pointer of the output file for test
	printOutput.open(temp);

	printOutputTrain.open(temp3);
}

//**********************************************************************
*****************
//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,"_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,"_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"<<proteinTe
stingError[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)
{
      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;
	}
	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;
	}
}
```

```
//*********************************************************************
****************
//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];
      }

      //creates the string path and opens the output file with parameters
      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,"_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();}
```

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;

#ifndef Services_h
#define Services_h

//************************************************************************
****************
//class Services: This class illustrates functions that the Bidirectional
//Recurent Neural Network needs to work properly
//************************************************************************
****************
class Services
{

public:

	Services(void);
	~Services(void);

	//method of the Services class
	double rand_numbers();
};
#endif
```

Services.cpp

```
#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;


Services::Services(void)
{
}

Services::~Services(void)
{
}

//generate random values 0..1
double Services::rand_numbers(){

	double temp_rand,temp_prosimo;

	temp_rand=0;temp_prosimo=0;

	//basic rand() function: %1000 means numbers from 0 to 1000
	temp_rand=((rand()%1000));

	//temp_rand/1000 because we want the range 0 to 1
	temp_rand/=10000;

	//we randomly choose if it will be a possitive or a negative number
	temp_prosimo=rand()%2;
	if(temp_prosimo==0)
		temp_rand*=-1;

	return temp_rand;
}
```

ΠΑΡΑΡΤΗΜΑ Β

PERL parsers

perlCreateProteinFromHobohm.pl

```
#!perl
#This program will read all the proteins of Hobohm's file and store them into a new .txt file in a specific format

#Rule: must declare every variable
use strict;

#Declaration of every value with "my"
my $hobohmFILE = "recent.pdb_select25"; ##Give the name of the input file
my $newFile = "hobohmPROTEINS.txt";  ##Give the name of the output file
my @array;
my $number;
my $line;
my $counter = 0;
my $num = 0;

#Read the "recent.pdb_select25" file which contains the name of all the proteins
if(!open(OLD, $hobohmFILE)) {die "Cannot open this file: $! \n";}

#The first line of $hobohmFILE contains the number of chains ex: 25% threshold list: 4018 chains with 568384 residues
chomp($line = <OLD>);
$line =~ /\d+.\s*\w+\s*\w+.\s*(\d+)/;

#Store the number of chains
$number = $1;

print $number."\n";

#Ignore the next two lines
chomp($line = <OLD>); chomp($line = <OLD>);

#Read and store the chains into a table
#A typical line is:    25   1JXSA    98 -1.00  0.00     N    98    98     0    38    10  interleukin enhancer binding factor
while ($counter != $number)
{
        chomp($line = <OLD>);
        $line =~ /\s*\d+\s*([0-9A-Za-z]+)/;
        $array[$counter] = $1;
        $counter++;
}

#opens the new file and trasfers them there
```

```
if(!open(NEW,">", $newFile)) {die "Cannot create this file: $! \n";}

while ($num <= $#array)
{
        print NEW $array[$num];
        print NEW " ";
        $num++;
}
```

msaProgram.pl

```
#!perl
#This program creates a folder of proteins' msa based on the input file
#Also creates the new dataset based on the input file

#Rule: must declare every variable
use strict;

#Rule: use paths
use Cwd;

#Declaration of every value with "my"
my $hobohmFILE ;                    ##tha name of the file must be in a format like
this
my $dataset_file;                   ##the name of the dataset file
my $msa_file;                       ##the name of the msa file
my @proteins;
my $line;
my $counter1 = 0;
my $temp;
my $hssp;
my $class;
my $numNotInHSSP = 0;
my $numYes = 0;
my $numFalseClass = 0;
my $secondaryList;
my $primaryList;
my $signal = 0;
my $dir;                            ##You must initialize the path in line 42 and 44
my @missingProteins;


#Gets the files
if($#ARGV!=2) {print "\nWrong number of arguments! (Needs: initialFile.txt
nameOfNewDataset.txt nameOfMSAfile.txt)\nExiting..\n\n"; exit 0;}
else{$hobohmFILE = $ARGV[0]; $dataset_file = $ARGV[1]; $msa_file = $ARGV[2];}

#Read the "hssp.txt" file which contains the name of all proteins and store them into "@proteins"
{
        if(!open(FROM, $hobohmFILE)) {die "Cannot open this file: $! \n";}
        #while (chomp($line = <FROM>)) {push(@proteins,$line); {last if eof;}}
        chomp($line = <FROM>);
        @proteins = split(/ /, $line);
        close FROM;
}
```

```
#Create the new dataset file, the msa file and the folder for the msa
{
	#create folder and create path
	mkdir("MSAfilesTemp");
	$dir = getcwd;
	$dir = $dir."/MSAfilesTemp/";

	#open them
	open(DATASET, ">", $dataset_file);
	open(MSA, ">", $msa_file);
}

#Main loop
while ($counter1 != $#proteins+1 )
{

	print $proteins[$counter1]."\n";

	$temp = $proteins[$counter1];
	$temp =~ s/(....)(.)/$1/;		#$temp: the name of the protein without the chain
	$temp = lc $temp;			#lower case
	$class = $2;				#$class: the class of the chain

	#create the file: concatenation of $temp and ".hssp" string
	$hssp = $temp.".hssp";

	#Search to find if an hssp file of the $hssp exists in the current folder
	if(!open(HSSP, $hssp)) { push(@missingProteins,$proteins[$counter1]);
$numNotInHSSP++; $counter1++; next;}

	#If exists but the class is between 0 and 9 ignores it
	if($class =~ /[0-9]/) { $numFalseClass++; $counter1++; next;}

	#If everything its ok, it must store the primary and secondary structure of the protein with
chain CLASS:
	#1. ignore the first rows
	chomp($line = <HSSP>);

	while (!($line =~ m/## ALIGNMENTS/))
	{
		#end of file
		if ($line =~ /\/\//) { $signal=1; last;}
		chomp($line = <HSSP>);
	}
```

```
	if ($signal == 1) {push(@missingProteins,$proteins[$counter1]); $numNotInHSSP++;
$signal = 0; $counter1++; next;}

	chomp($line = <HSSP>);			#the first row after the "## ALIGNMENTS"

	#2. Store primary and secondary structure
	while(!($line =~ m/## SEQUENCE PROFILE AND ENTROPY/))
	{
		#Read next line
		chomp($line = <HSSP>);

		#a typical line:   3    5 A G  S OR  241       ! !
		#another typical line:     1   -2 A G            0   0  111   14    0                G
G
		$_ = $line;
		if(!(/^\s*\d+\s*\-?(\d+|\s)\s*\w\s*\w\s\s./)) { if (/## ALIGNMENTS/) {last;} else
{next;} }
		else
		{
			$line =~ /\s*(\d+)\s*\-?(\d+|\s)\s*(\w)\s*(\w)\s\s(.)/;

			#check for the appropriate chain
			if ($3 eq $class)
			{
				my $a = $5;
				if ($a eq " ") {$a = "L";}
				$secondaryList = $secondaryList.$a;
				$primaryList = $primaryList.$4;
			}
			else
			{
				next;
			}
		}

	}

	#if the $secondaryList & $primaryList are both empty goes to the next protein
	if($primaryList eq "") { push(@missingProteins,$proteins[$counter1]); $numNotInHSSP++;
$counter1++; next;}

	#finds the msa
	if(!($line =~ m/## SEQUENCE PROFILE AND ENTROPY/))
	{
		while (!($line =~ m/## SEQUENCE PROFILE AND ENTROPY/))
		{ chomp($line = <HSSP>); }
	}
```

```
	chomp($line = <HSSP>);

	#creates the protein file
	open(FILE_PROTEIN, ">", $dir.$temp.$class.".txt");

	#print MSA $temp.$class."\n";

	#read every msa line and store it
	while (!(($line =~ m/^\/\//) || ($line =~ m/## INSERTION LIST/)))
	{
		$_ = $line;
		if(!(/^\s*\d+\s*\-?(\d+|\s)\s*\w/)) { chomp($line = <HSSP>); next;}
		else
		{
			#finds the motifs
			$line =~ /\s*(\d+)\s*\-
?(\d+|\s)\s*(\w)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\
s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*(\d+)\s*/;

			#if the class is the same as the protein's
			if($class eq $3)
			{
				#stores the motifs into the array
				my @temps =
($4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,$15,$16,$17,$18,$19,$20,$21,$22,$23,$24);

				#bounds: between 0 and 1
				my $x = 0;
				while ($x != $#temps)
				{
					$temps[$x] = $temps[$x] / 100;
					$x++;
				}

				my $theLine = join (" ", @temps);

				print MSA "$theLine \n";
				print FILE_PROTEIN "$theLine \n";
			}
		}
		chomp($line = <HSSP>);
	}


	#prints the primary and secondary structures
	print DATASET $temp.$class."\n";
```

```
            print DATASET $primaryList.".\n";
            print DATASET $secondaryList.".\n";
            $primaryList = ""; #empty
            $secondaryList = ""; #empty

            $numYes++;
            $counter1++;

            close HSSP;
            close FILE_PROTEIN;

}

print "\nStatistics: \n";
print "---------- \n";
print "There are $counter1 proteins: \n";
print "Found $numYes proteins \n";
print "$numNotInHSSP proteins are missing from HSSP \n";
print "$numFalseClass proteins with false chain exist \n";

#If we want to print the missing proteins enable the 3 comments below
print "\nMissing proteins: \n";
print "----------------- \n";
foreach $_ (@missingProteins) {print $_."\n";}

#close filehandles
close DATASET;
close MSA;
```

makeProteinSet.pl

```
#!perl
#Creates a file of proteins included in "$old_dataset_file" text file

#Rule: declare every variable with "my"
use strict;

#Variable declaration
my $old_dataset_file; ##The name of the input file
my $new_dataset_file; ##The name of the output file
my $counter=0;
my $line;

#Gets the files
if($#ARGV!=1) {print "\nWrong number of arguments! (Needs: oldDatasetFile.txt
newDatasetFile.txt)\nExiting..\n\n"; exit 0;}
else{$old_dataset_file = $ARGV[0]; $new_dataset_file = $ARGV[1]; }

#Open files
if(!open(OLD_DATASET, $old_dataset_file)) {die "Cannot open the $old_dataset_file: $! \n";}
if(!open(NEW_DATASET, ">", $new_dataset_file)) {die "Cannot open the $new_dataset_file: $!
\n";};


#Read every protein and print it in "$new_dataset_file"
while (1)
{
        chomp($line = <OLD_DATASET>);

        #print them all in one line -- the special format for msaProgram.pl
        if ($line =~ m/^(\s*)\d/)
                { print NEW_DATASET $line; print NEW_DATASET " "; $counter++; }

        last if eof;
}

#Print information about the proteins
print "There are $counter proteins in $old_dataset_file. \n";
```

randomDatasetCreate.pl

```
!perl
#This program will randomly choose x proteins for a training set, it will remove them from the list
#Then, it will randomly choose about

#Rule: must declare every variable
use strict;

#Declaration of every value with "my"
my $initialFILE = "moreThan60length.txt";          ##Give the name of the whole file
my $trainingFile;
my $testFile;
my $numberOfTraining;
my $numberOfTest;
my @proteins;
my @primary;
my @lengthProteins;
my @secondary;
my $line;
my $counter = 0;
my $random_number;
my $tempCounter;

#Gets files
if($#ARGV!=3) {print "\nWrong number of arguments! (Needs: nameOfTrainFile.txt
numProteinsForTrain testFile.txt numProteinsForTest)\nExiting..\n\n"; exit 0;}
else{$trainingFile = $ARGV[0]; $numberOfTraining = $ARGV[1]; $testFile = $ARGV[2];
$numberOfTest = $ARGV[3];}

#Read the initial file which contains hobohm's all proteins
if(!open(FROM, $initialFILE)) {die "Cannot open the initial file: $! \n";}

#Read every protein and store its name and length into an array
$counter = 0;
$tempCounter = 0;

while(1)
{
        $counter++;

        chomp($line = <FROM>);

        if($counter==1){push(@proteins,$line); }
        elsif ($counter==2){  push(@primary,$line); }
        else{push(@secondary,$line); $counter=0; }
```

```
	last if eof;

}
close FROM;

#open them
open(TRAIN, ">", $trainingFile);
open(TEST, ">", $testFile);

while($numberOfTraining != 0)
{

	$random_number = int(rand($#proteins));

	print TRAIN
"$proteins[$random_number]\n$primary[$random_number]\n$secondary[$random_number]\n"
;

	#diagrafi kai miwsi pinakwn wste i kathe proteini na epilegete mia kai mono fora
	splice(@proteins,$random_number,1);
	splice(@primary,$random_number,1);
	splice(@secondary,$random_number,1);

	$numberOfTraining--;

}

while($numberOfTest != 0)
{

	$random_number = int(rand($#proteins));

	print TEST
"$proteins[$random_number]\n$primary[$random_number]\n$secondary[$random_number]\n"
;

	#diagrafi kai miwsi pinakwn
	splice(@proteins,$random_number,1);
	splice(@primary,$random_number,1);
	splice(@secondary,$random_number,1);

	$numberOfTest--;
}

close TRAIN;
close TEST;
```

randomizeDataset.pl

```
#!perl

#Rule: must declare every variable
use strict;

#Rule: use paths
use Cwd;

#Declaration of every value with "my"
my $dir;
my $range = 100;
my $random_number;
my @proteins;
my $line;
my $k;
my @secondary;
my @primary;



#create folder and create path
$dir = getcwd;
$dir = $dir."/TrainingSets/";
#$dir = $dir."msaProteinsTrainBigDataset_afterProcess.txt";
$dir = $dir."msaProteinsTrainBigDataset_afterProcess.txt";

#create tables;
if(!open(FROM, $dir)) {die "Cannot open this file: $! \n";}

$k = 0;

while(1)
{
	$k++;

	chomp($line = <FROM>);
	#print $line."\n";

	if($k==1){push(@proteins,$line); }
	elsif ($k==2){push(@primary,$line);}
	else{push(@secondary,$line); $k=0;}

	last if eof;
```

```
}

my $total = $#proteins;

if(!open(ELA, ">", $dir)) {die "Cannot open this file: $! \n";}

while ($total != -1)
{

	$random_number = int(rand($total));

	print ELA
"$proteins[$random_number]\n$primary[$random_number]\n$secondary[$random_number]\n"
;

	#diagrafi kai miwsi pinakwn
	$total--;

	splice(@proteins,$random_number,1);
	splice(@primary,$random_number,1);
	splice(@secondary,$random_number,1);

}

close ELA;
close FROM;
```

confusionMatricesAndSOV.pl

```
!perl
#This program will takes as input the output file from the PSSP system
#It calculates SOV and confusion matrices of it
#Needs SOV.c

#Rule: must declare every variable
use strict;

#Rule: use paths
use Cwd;

#Declaration of every value with "my"
my $initialFILE;
my $outFile;
my $nameOfGeneralFile;
my $sovFile = "tempSOV.txt";
my $dir;
my $temp;
my $protein;
my $primary;
my $secondaryReal;
my $secondaryPredicted;
my $aminoacidReal;
my $aminoacidPredicted;
my $line;
my $tempLength;
my $temp2=0;
my $tempLine;
my $counter = 0;
my @counterEall;
my @counterHall;
my @counterLall;
my @proteinArray;
my @secondaryRealArray;
my @secondaryPredictedArray;
my @primaryArray;
my @firstLine;
my @secondLine;
my @thirdLine;
my @arrayH;
my @arrayE;
my @arrayL;
my @primaryLine;
my $lineSOV;
```

```
my $Q3;
my $sov;
my @sovi;
my $sum;

#Gets the files
if($#ARGV!=1) {print "\nWrong number of arguments! (Needs: initialFile.txt
outputFile.txt)\nExiting..\n\n"; exit 0;}
else{$initialFILE = $ARGV[0]; $nameOfGeneralFile = $ARGV[1];}


#Read the initial file which contains the dataset
if(!open(FROM, $initialFILE)) {die "Cannot open the initial file: $! \n";}

#Open the general file and write the first lines
open(GENERAL_FILE, ">", $nameOfGeneralFile);
print GENERAL_FILE "Protein    Q3    SOV(all)  SOV(H)   SOV(E)   SOV(L)   HH     HE      HL      EH
EE      EL      LH      LE      LL\n";
print GENERAL_FILE "-----------------------------------------------------------------------------------------------------------
-----------------------------\n";

#create folder and create path
mkdir("Temp");
$dir = getcwd;
$dir = $dir."/Temp/";

#Read every protein's information and store them into arrays
$counter = 0;

while(1)
{
        $counter++;

        chomp($line = <FROM>);

        if($counter==1){push(@proteinArray, $line); }                                   #first line
has the name of the protein
        elsif($counter==2) {push(@primaryArray, $line); }                               #second
line has the primary structure
        elsif ($counter==3){push(@secondaryRealArray, $line); }                         #third
line has the observed secondary structure
        elsif ($counter==4) {push(@secondaryPredictedArray, $line); $counter=0; }       #fourth
line has the predicted secondary structure
        else {;}

        last if eof;
```

```
}

$counterEall[0]=0;$counterEall[1]=0;$counterEall[2]=0;$counterEall[3]=0;
$counterHall[0]=0;$counterHall[1]=0;$counterHall[2]=0;$counterHall[3]=0;
$counterLall[0]=0;$counterLall[1]=0;$counterLall[2]=0;$counterLall[3]=0;

$sum = 0;
$counter=0;

#for every protein calculate the SOV and confusion matrix
while($#proteinArray>=$counter)
{

		@firstLine = split(/ /, $proteinArray[$counter]); $protein = @firstLine[0];
	#takes the name of protein
		@primaryLine = split(/:/, $primaryArray[$counter]); $primary = @primaryLine[1];
		@secondLine = split(/:/, $secondaryRealArray[$counter]); $secondaryReal =
@secondLine[1];		#takes the observed secondary structure
		@thirdLine = split(/:/, $secondaryPredictedArray[$counter]); $secondaryPredicted
= @thirdLine[1]; #takes the predicted secondary structure

		$temp=0;

		print $protein."\n";

		#initialize the matrix
		while($temp<3)
		{
			$arrayH[$temp] = 0;
			$arrayE[$temp] = 0;
			$arrayL[$temp] = 0;

			$temp++;
		}


		$temp=0;

		#create the confusion matrix
		while($temp<length($secondaryReal))
		{

			$aminoacidReal = substr($secondaryReal,$temp,1);
			$aminoacidPredicted = substr($secondaryPredicted,$temp,1);

			if($aminoacidReal eq 'H')
			{
```

```
                                $counterHall[0]++;

                                if($aminoacidPredicted eq 'H') {$arrayH[0] = $arrayH[0] + 1;
$counterHall[1]++;}
                                elsif($aminoacidPredicted eq 'E') {$arrayH[1] = $arrayH[1] + 1;
$counterHall[2]++;}
                                else {$arrayH[2] = $arrayH[2] + 1; $counterHall[3]++;}
                        }
                        elsif($aminoacidReal eq 'E')
                        {
                                $counterEall[0]++;

                                if($aminoacidPredicted eq 'H') {$arrayE[0] = $arrayE[0] + 1;
$counterEall[1]++;}
                                elsif($aminoacidPredicted eq 'E') {$arrayE[1] = $arrayE[1] + 1;
$counterEall[2]++;}
                                else {$arrayE[2] = $arrayE[2] + 1; $counterEall[3]++;}
                        }
                        else
                        {
                                $counterLall[0]++;

                                if($aminoacidPredicted eq 'H') {$arrayL[0] = $arrayL[0] + 1;
$counterLall[1]++;}
                                elsif($aminoacidPredicted eq 'E') {$arrayL[1] = $arrayL[1] + 1;
$counterLall[2]++;}
                                else {$arrayL[2] = $arrayL[2] + 1; $counterLall[3]++;}
                        }


                        $temp++;
                }

                #create a temporary file needed for SOV.c program
                createTempFile();

                #read from sov.c output file
                readFromSOV();

                #print the confusion matrix
                printInformationToFile();

                $counter++;

}

my $x = $sum/$#proteinArray;
```

```
print "\nRESULTS:\n\n";
printf ("sum = %5.5s %\n\n",$x);
printf ("Average prediction of this dataset for Helixes: %6.6s
%\n",$counterHall[1]/$counterHall[0]);
printf ("Average prediction of this dataset for Extended: %6.6s
%\n",$counterEall[2]/$counterEall[0]);
printf ("Average prediction of this dataset for Loops: %6.6s
%\n\n",$counterLall[3]/$counterLall[0]);

printf ("HE = %6.6s, HL = %6.6s, EH = %6.6s, EL = %6.6s, LH = %6.6s, LE =
%6.6s\n\n",($counterHall[2]/$counterHall[0]),($counterHall[3]/$counterHall[0]),($counterEall[1]/
$counterEall[0]),($counterEall[3]/$counterEall[0]),($counterLall[1]/$counterLall[0]),($counterLall[
2]/$counterLall[0]));

#########################################################
###       S U B R O U T I N E S         ###
#########################################################

#Read the temporary file created by SOV.c program and store the needed information#
sub readFromSOV
{
        open(FILE1, "sovOutputFile.txt");

        chomp($lineSOV = <FILE1>);

        while(!($lineSOV =~ /SECONDARY STRUCTURE PREDICTION ACCURACY EVALUATION/))
        {chomp($lineSOV = <FILE1>);}

        {chomp($lineSOV = <FILE1>);chomp($lineSOV = <FILE1>);chomp($lineSOV =
<FILE1>);chomp($lineSOV = <FILE1>);}

        $lineSOV =~ /(\s*)Q3(\s*):(\s*)([0-9]+.[0-9]*|[0-9]+)/;
        $Q3 = $4;

        {chomp($lineSOV = <FILE1>);chomp($lineSOV = <FILE1>);}

        $lineSOV =~ /(\s*)SOV(\s*):(\s*)([0-9]+.[0-9]*|[0-9]+)(\s*)([0-9]+.[0-9]*|[0-9]+)(\s*)([0-
9]+.[0-9]*|[0-9]+)(\s*)([0-9]+.[0-9]*|[0-9]+)/;

        $sov = $4;$sovi[0]=$6;$sovi[1]=$8;$sovi[2]=$10; $sum=$sum+$sov;

        close FILE1;

}

#Print the information to a general file and to the proteins file#
sub printInformationToFile
```

```
{

		$outFile = $dir.$protein.".txt";
		open(OUT, ">", $outFile);

		print OUT "ProteinName                          :$protein \n\n";
		print OUT "PrimaryStructure          :$primary \n";
		print OUT "SecondaryStructure        :$secondaryReal \n";
		print OUT "PredictedSecondaryStructure :$secondaryPredicted \n\n\n";


		print OUT "Q3     SOV(all)   SOV(H)    SOV(E)    SOV(L)\n";
		print OUT "-------------------------------------------------\n";
		printf (OUT "%-8s %-10s %-10s %-10s %-10s
\n\n\n\n",$Q3,$sov,$sovi[0],$sovi[1],$sovi[2]);

		print OUT "Confusion Matrix:   HH    HE    HL     EH    EE    EL     LH    LE    LL\n";
		print OUT "                 ---------------------------------------------------\n";
		printf (OUT "                 %-5s %-5s %-5s    %-5s %-5s %-5s    %-5s %-5s %-5s
\n\n",$arrayH[0],$arrayH[1],$arrayH[2],$arrayE[0],$arrayE[1],$arrayE[2],$arrayL[0],$arrayL[1],$ar
rayL[2]);

		close OUT;

		printf (GENERAL_FILE "%-10s %-8s %-8s %-8s %-8s %-8s %-8s %-8s %-8s %-8s %-8s
%-8s %-8s %-8s %-8s
\n",$protein,$Q3,$sov,$sovi[0],$sovi[1],$sovi[2],$arrayH[0],$arrayH[1],$arrayH[2],$arrayE[0],$arr
ayE[1],$arrayE[2],$arrayL[0],$arrayL[1],$arrayL[2]);

}

#create the temporary input file for the sov.c program based to its format#
sub createTempFile
{
		$temp2 = 0;

		open(SOV, ">", $sovFile);

		print SOV ">OSEC"."\n";

		$tempLength = length($secondaryReal);
		while($temp2<=$tempLength)
		{
			$tempLine = substr($secondaryReal,$temp2,100);

			print SOV $tempLine."\n";
```

```
		$temp2=$temp2+100;
	}

	print SOV ">PSEC"."\n";

	$temp2 = 0;
	while($temp2<=$tempLength)
	{
		$tempLine = substr($secondaryPredicted,$temp2,100);

		print SOV $tempLine."\n";

		$temp2=$temp2+100;
	}

	#executes the sov program and save the results to sovOutputFile.txt
	my $dirProgram = $dir;
	system ("./sov tempSOV.txt > sovOutputFile.txt");
}
```

viterbi.pl

```
#!/usr/bin/perl

# Usage
# viterbi.pl inputfile
# Implementation of a HMM for postprocessing BRNN secondary structure prediction
# Prints results in STDOUT ...

use Algorithm::Viterbi;
use strict;
use warnings;

my @headers=(); # Header lines
my @seq=(); # Sequence
my @inSS=(); # This is the BRNN output
my @outSS=();# This is the observed Secondary Structure
##############
my @t_headers=(); # Header lines for test file
my @t_seq=(); # Sequence for test file
my @t_inSS=(); # This is the BRNN output for test file
my @t_outSS=();# This is the observed Secondary Structure for test file

readTrainingData($ARGV[0]);
readValidationData($ARGV[1]);

my $training_data = prepareTrainingData();
my $validation_data = prepareValidationData();

my $v = Algorithm::Viterbi->new();
$v->train($training_data);

#use Data::Dumper;
#print Dumper($v);
#exit;

my $count = 0;

#checks validation proteins
foreach my $in (@t_inSS)
{
        my $observations=[];
        @$observations = split '', $in;
        my ($prob, $v_path, $v_prob) = $v->forward_viterbi($observations);
        my $q3 = calculateScores($t_outSS[$count],$v_path);
```

```
	print $t_headers[$count], " Q3: $q3\n";# Forward probability: $prob Viterbi probability:
$v_prob\n";
	print "primaryStructure          :", $t_seq[$count],"\n";
	print "secondaryStructure        :", $t_outSS[$count],"\n";
	#print "predictedSecondaryStructure:", $t_inSS[$count],"\n";
	print "optimizedSecondaryStructure:", @$v_path,"\n";

	$count++;
}

sub calculateScores
{
	my $obs= shift;
	my $pred_ref= shift;

	my @o = split '', $obs;
	my $N = scalar @o;
	my @p = @$pred_ref;
	my $Q3 = 0;
	for (my $i=0; $i<$N; $i++)
	{
		$Q3 += 1 if $o[$i] eq $p[$i];
	}
	$Q3/=$N;
	return (100 * $Q3);
}

sub readTrainingData
{
my $file = shift;
open(IN, $file) or die "Input train file error";
	while(<IN>) # Read header
	{
		chomp $_; push @headers, $_;
		my $seq = <IN>;
		chomp $seq;
		$seq = fix($seq, "primaryStructure");
		push @seq, $seq;
		my $obs = <IN>;
		chomp $obs;
		$obs = fix($obs, "secondaryStructure");
		push @outSS, $obs;
		my $pred = <IN>;
		chomp $pred;
		$pred = fix($pred, "predictedSecondaryStructure");
		push @inSS, $pred;
```

```
	}
}

sub readValidationData
{
my $file = shift;
open(INN, $file) or die "Input test file error";
	while(<INN>) # Read header
	{
		chomp $_; push @t_headers, $_;
		my $seq = <INN>;
		chomp $seq;
		$seq = fix($seq, "primaryStructure");
		push @t_seq, $seq;
		my $obs = <INN>;
		chomp $obs;
		$obs = fix($obs, "secondaryStructure");
		push @t_outSS, $obs;
		my $pred = <INN>;
		chomp $pred;
		$pred = fix($pred, "predictedSecondaryStructure");
		push @t_inSS, $pred;
	}
}

sub fix
{
	my $seq = shift;
	my $str = shift;
	chomp $str;
	$seq =~ s/^$str\s*\://;
	return $seq;
}

sub prepareValidationData
{
	my $retT= [];
	for(my $i=0; $i<=$#t_inSS; $i++) # $i iterates over sequences
	{
		my $in = $t_inSS[$i];
		my $out = $t_outSS[$i];
		for(my $j=0;$j<length($in);$j++) # $j iterates over sequence positions
		{
			push @$retT, [ substr($in,$j,1), substr($out,$j,1) ];
		}
	}
return $retT;
```

```
}

sub prepareTrainingData
{
	my $ret= [];
	for(my $i=0; $i<=$#inSS; $i++) # $i iterates over sequences
	{
		my $in = $inSS[$i];
		my $out = $outSS[$i];
		for(my $j=0;$j<length($in);$j++) # $j iterates over sequence positions
		{
			push @$ret, [ substr($in,$j,1), substr($out,$j,1) ];
		}
	}
return $ret;
}
```

ΠΑΡΑΡΤΗΜΑ Γ

Γενετικοί Αλγόριθμοι

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>

using namespace std;
#ifndef chromosome_h
#define chromosome_h
class chromosome
{
public:
	chromosome(int position);
	~chromosome();

	int returnHiddenLayerOneSize();
	int returnHiddenLayerTwoSize();
	int returnHiddenLayerOneOfBackwardSize();
	int returnHiddenLayerTwoOfBackwardSize();
	int returnHiddenLayerOneOfForwardSize();
	int returnHiddenLayerTwoOfForwardSize();
	double returnEval();
	double returnLearningRate();
	double returnmomentum();
	int returnWindowSize();
	double returnqVar();
	int returnsVar();
	void print();

	void setEval(double eval);

	void returnTempAtom(vector<int> tempAtom);
	void crossoveredAtom(vector<int> tempAtom);

	int returnPosition();
	void mutation();
	void convert();

	//void calEval();

private:
	double evaluation;

	int HiddenLayerOneSize;
```

```
        int HiddenLayerTwoSize;
        int HiddenLayerOneOfBackwardSize;
        int HiddenLayerTwoOfBackwardSize;
        int HiddenLayerOneOfForwardSize;
        int HiddenLayerTwoOfForwardSize;
        double LearningRate;
        double momentum;
        int WindowSize;
        double qVar;
        int sVar;

        vector<int> chromosomeAtom;
        int position;

        //void createX(int temp[]);
};

#endif
```

chromosome.cpp

```
/*************************************************************************
************************
Ylopoisi twn methodon tis klasis atomo
*************************************************************************
***********************/
#include "chromosome.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>

//statheres
#define POPULATION 20
#define PC 25
//#define PM 01
#define GEN_NUM 200
#define P1 3.14159265
#define CHROMOSOME_LENGTH 25



using namespace std;
/*************************************************************************
************************
 * Constractor gia to kathe atomo : Kata ti dimiourgia tou kathe atomou
tis protis geneas
 * arxikopoiountai tixaia to kathe atomo. Ta prota 4 bits tou atomou
simvolizoun to x1 kai ta
 * epomena 4 bits simvolizoun to x2. Afou gini i tixea arxikopoiisi tote
ipologizetai to x1, x2
 * kai Evaluation tou kathe atomou

*************************************************************************
***********************/
chromosome::chromosome(int position_)
{
	evaluation=0;
	position=position_;

	//dimiourgia toy tixeou xromosomatos me 0 kai 1
	for(int i=0;i<CHROMOSOME_LENGTH;i++)
	{
		chromosomeAtom.push_back(rand()%2);
		//cout<<chromosomeAtom[i];
	}
	//cout<<endl;
	convert();
```

```
}

chromosome :: ~chromosome(void)
{


}

//oi parametroi tou diktiou
void chromosome :: convert(){

	HiddenLayerOneSize=0;
	HiddenLayerTwoSize=0;
	HiddenLayerOneOfBackwardSize=0;
	HiddenLayerTwoOfBackwardSize=0;
	HiddenLayerOneOfForwardSize=0;
	HiddenLayerTwoOfForwardSize=0;
	LearningRate=0.9;
	momentum=0.5;
	WindowSize=0;	qVar=0.6;
	sVar=0;

	//ta prwta 5 aforoun to HiddenLayerOneSize etsi vgazei ton arithmo
pou analogei sta 5 prwta bits
	for(int i=0;i<CHROMOSOME_LENGTH;i++){
		switch(i){
			case 0:
				if(chromosomeAtom[0]==1)
					HiddenLayerOneSize=HiddenLayerOneSize+32;
				break;
			case 1:
				if(chromosomeAtom[1]==1)
					HiddenLayerOneSize=HiddenLayerOneSize+16;
				break;
			case 2:
				if(chromosomeAtom[2]==1)
					HiddenLayerOneSize=HiddenLayerOneSize+8;
				break;
			case 3:
				if(chromosomeAtom[3]==1)
					HiddenLayerOneSize=HiddenLayerOneSize+4;
				break;
			case 4:
				if(chromosomeAtom[4]==1)
					HiddenLayerOneSize=HiddenLayerOneSize+2;
				break;
			case 5:
				if(chromosomeAtom[5]==1)
					HiddenLayerOneSize=HiddenLayerOneSize+1;
				break;
			case 6:
				if(chromosomeAtom[6]==1)

	HiddenLayerOneOfBackwardSize=HiddenLayerOneOfBackwardSize+32;
				break;
			case 7:
```

```
                    if(chromosomeAtom[7]==1)

HiddenLayerOneOfBackwardSize=HiddenLayerOneOfBackwardSize+16;
                    break;
              case 8:
                    if(chromosomeAtom[8]==1)

HiddenLayerOneOfBackwardSize=HiddenLayerOneOfBackwardSize+8;
                    break;
              case 9:
                    if(chromosomeAtom[9]==1)

HiddenLayerOneOfBackwardSize=HiddenLayerOneOfBackwardSize+4;
                    break;
              case 10:
                    if(chromosomeAtom[10]==1)

HiddenLayerOneOfBackwardSize=HiddenLayerOneOfBackwardSize+2;
                    break;
              case 11:
                    if(chromosomeAtom[11]==1)

HiddenLayerOneOfBackwardSize=HiddenLayerOneOfBackwardSize+1;
                    break;
              case 12:
                    if(chromosomeAtom[12]==1)

HiddenLayerOneOfForwardSize=HiddenLayerOneOfForwardSize+32;
                    break;
              case 13:
                    if(chromosomeAtom[13]==1)

HiddenLayerOneOfForwardSize=HiddenLayerOneOfForwardSize+16;
                    break;
              case 14:
                    if(chromosomeAtom[14]==1)

HiddenLayerOneOfForwardSize=HiddenLayerOneOfForwardSize+8;
                    break;
              case 15:
                    if(chromosomeAtom[15]==1)

HiddenLayerOneOfForwardSize=HiddenLayerOneOfForwardSize+4;
                    break;
              case 16:
                    if(chromosomeAtom[16]==1)

HiddenLayerOneOfForwardSize=HiddenLayerOneOfForwardSize+2;
                    break;
              case 17:
                    if(chromosomeAtom[17]==1)

HiddenLayerOneOfForwardSize=HiddenLayerOneOfForwardSize+1;
                    break;
              case 18:
                    if(chromosomeAtom[18]==1)
```

```
                              WindowSize=WindowSize+16;
                        break;
                  case 19:
                        if(chromosomeAtom[19]==1)
                              WindowSize=WindowSize+8;
                        break;
                  case 20:
                        if(chromosomeAtom[20]==1)
                              WindowSize=WindowSize+4;
                        break;
                  case 21:
                        if(chromosomeAtom[21]==1)
                              WindowSize=WindowSize+2;
                        break;
                  case 22:
                        if(chromosomeAtom[22]==1)
                              WindowSize=WindowSize+1;
                        break;
                  case 23:
                        if(chromosomeAtom[23]==1)
                              sVar=sVar+2;
                        break;
                  case 24:
                        if(chromosomeAtom[24]==1)
                              sVar=sVar+1;
                        break;
            }
      }

      //tipos metatropis anamesa sta bounds: lowBound + numberStoDekadiko
*(upperBound - lowBound)/2^m - 1
      HiddenLayerOneSize = 2.0 + HiddenLayerOneSize * ((60.0 - 2.0) /
63.0) ;     //[2..60]
      //HiddenLayerTwoSize = 0.0 + HiddenLayerTwoSize * 60.0 / 63.0 ;
//[0..60]
      HiddenLayerOneOfBackwardSize = 2.0 + HiddenLayerOneOfBackwardSize *
((60.0 - 2.0) / 63.0) ;//[2..60]
      //HiddenLayerTwoOfBackwardSize = 0.0 + HiddenLayerTwoOfBackwardSize
* 60.0 / 63.0 ;
      HiddenLayerOneOfForwardSize = 2.0 + HiddenLayerOneOfForwardSize *
((60.0 - 2.0) / 63.0) ;//[2..60]
      //HiddenLayerTwoOfForwardSize = 0.0 + HiddenLayerTwoOfForwardSize *
60.0 / 63.0 ;
      //LearningRate = 0 + LearningRate * (1.0 / 15.0) ; //[0..1] + 1
dekadiko psifio akrivias : 4 digits to learning rate
      //momentum = 0 + momentum * (1.0 / 15.0) ; //[0..1]
      sVar = 3.0 + (( sVar * (5.0 - 3.0)) / 3.0);
      WindowSize = 9.0 + ((WindowSize * (31.0 - 9.0)) / 31.0);
      //qVar = 0.0 + qVar * (1.0 / 15.0) ;
}


int chromosome :: returnHiddenLayerOneSize()
{

      return HiddenLayerOneSize;
```

```
}

int chromosome :: returnHiddenLayerTwoSize()
{
	return HiddenLayerTwoSize;
}

int chromosome :: returnHiddenLayerOneOfBackwardSize()
{
	return HiddenLayerOneOfBackwardSize;
}

int chromosome :: returnHiddenLayerTwoOfBackwardSize()
{
	return HiddenLayerTwoOfBackwardSize;
}

int chromosome :: returnHiddenLayerOneOfForwardSize()
{
	return HiddenLayerOneOfForwardSize;
}

int chromosome :: returnHiddenLayerTwoOfForwardSize()
{
	return HiddenLayerTwoOfForwardSize;
}

double chromosome :: returnEval()
{
	return evaluation;
}

double chromosome :: returnLearningRate()
{
	return LearningRate;
}

double chromosome :: returnmomentum()
{
	return momentum;
}

int chromosome :: returnWindowSize()
{
	return WindowSize;
}

double chromosome :: returnqVar()
{
	return qVar;
}

int chromosome :: returnsVar()
{
	return sVar;
```

```
}

void chromosome :: setEval(double eval)
{
	evaluation = eval;
}

void chromosome :: returnTempAtom(vector<int> tempAtom)
{
	for(int i=0;i<tempAtom.size();i++){
		tempAtom[i]=chromosomeAtom[i];
	}
}


void chromosome :: crossoveredAtom(vector<int> tempAtom)
{
	for(int i=0;i<tempAtom.size();i++){
		chromosomeAtom[i]=tempAtom[i];
	}
}


int chromosome :: returnPosition()
{
	return position;
}



/**************************************************************************
***********************
 * Methodos pou efarmozi tin metallaxi sto siggekrimeno atomo. Elegxei
ena pros ena ta bits ke
 * an i pithanotita pou simiourgitai einia mikroteri apo 0.01 tote an to
bit einai 0 ginete 1
 * kai an einai 1 ginetai 0. STi sinexeia ipologizetai to neo x1 kai x2
kai to neo evaluation

**************************************************************************
**********************/
void chromosome :: mutation(){

	double temp;

	for(int i=0;i<CHROMOSOME_LENGTH;i++){

		temp=(rand()%1000000)/10000.0;
		if(temp<1){

			if(chromosomeAtom[i]==1)
				chromosomeAtom[i]=0;
			else
				chromosomeAtom[i]=1;

				convert();
```

```
                        //calEval();
                }
        }
}


void chromosome :: print()
{
        for(int i=0;i<CHROMOSOME_LENGTH;i++)
                {cout<<chromosomeAtom[i]<<" "; }
}
```

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>



//statheres
#define POPULATION 20
#define PC 0.25
#define PM 0.01
#define GEN_NUM 200
#define P1 3.14159265

/*************************************************************************
************************
 * Constractor gia to kathe atomo : Kata ti dimiourgia tou kathe atomou
tis protis geneas
 * arxikopoiountai tixaia to kathe atomo. Ta prota 4 bits tou atomou
simvolizoun to x1 kai ta
 * epomena 4 bits simvolizoun to x2. Afou gini i tixea rxikopoiisi tote
ipologizetai to x1, x2
 * kai Evaluation tou kathe atomou

*************************************************************************
************************/

//entoli ston compiler gia anagnorisi ton bibliothikwn tis C++
using namespace std;

#ifndef ga_h
#define ga_h

class ga
{
public:
      ga();
      ~ga(void);

      void createRoulette();
      void newPopulation();
      void mutation();
      void crossover();
      void chooseBestAtom(int gen);
```

```
        void setEvaluation(double eval,int position);
        void printSolution();
        void printCh();



        vector <chromosome> prev_generation;
        vector <chromosome> next_generation;
        vector <double> roulette;
        //int solutionX1,solutionX2;
        int solutionHiddenLayerOneSize;
        int solutionHiddenLayerTwoSize;
        int solutionHiddenLayerOneOfBackwardSize;
        int solutionHiddenLayerTwoOfBackwardSize;
        int solutionHiddenLayerOneOfForwardSize;
        int solutionHiddenLayerTwoOfForwardSize;
        double solutionLearningRate;
        double solutionmomentum;
        int solutionWindowSize;
        double solutionqVar;
        int solutionsVar;
        double solutionEval;
        ofstream outf;



private:

};

#endif
```

ga.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 20
#define GEN_NUM 200
#define P1 3.14159265
#define CHROMOSOME_LENGTH 25


using namespace std;

ga::ga()
{

	//dimiourgei ena vector apo xromosomata
	for(int i=0;i<POPULATION;i++){
		prev_generation.push_back(chromosome(i));
		next_generation.push_back(chromosome(i));
		roulette.push_back(0.0);
	}

	 solutionHiddenLayerOneSize=0;
	 solutionHiddenLayerTwoSize=0;
	 solutionHiddenLayerOneOfBackwardSize=0;
	 solutionHiddenLayerTwoOfBackwardSize=0;
	 solutionHiddenLayerOneOfForwardSize=0;
	 solutionHiddenLayerTwoOfForwardSize=0;
	 //solutionLearningRate=0.0;
	 //solutionmomentum=0.0;
	 solutionWindowSize=0;
	 //solutionqVar=0.0;
	 solutionsVar=0;

	//open file
	outf.open("GA_Report.txt");

}

ga :: ~ga(void)
{
```

```
}

void ga:: setEvaluation(double eval,int position)
{
      next_generation[position].setEval(eval);
}

void ga:: createRoulette()
{

      double sum_F=0;
      int x;

            for(int i=0;i<20;i++)
            {
                  next_generation[i].setEval((rand()%1000000)/10000.0);
                  //cout<<next_generation[i].returnEval()<<endl;
            }



      for(x=0;x<POPULATION;x++)
      {
            sum_F+=next_generation[x].returnEval();
            //cout<<sum_F<<endl;

            roulette[x]=0;
      }
         //system("pause");

      roulette[0]=next_generation[0].returnEval()/sum_F;


      for(int i=1;i<POPULATION;i++){

            roulette[i]=roulette[i-
1]+next_generation[i].returnEval()/sum_F;
      }
}



void ga :: newPopulation(){

double temp=0;



for(int i=0;i<POPULATION;i++){
       prev_generation[i]=next_generation[i];
}



for(int i=0;i<POPULATION;i++){
```

```
      temp=(rand()%1000000)/1000000.0;

      int x=0;


    while(1)
      {

      if(temp<=roulette[0]){
            break;
      }else if(temp>=roulette[x-1] && temp<=roulette[x]){
            break;
      }

    x++;
    }

      //cout<<roulette[0]<<endl;

    next_generation[i]=prev_generation[x];
    }
}


void ga :: mutation(){

for(int i=0;i<POPULATION;i++){

      next_generation[i].mutation();
}
}




/*************************************************************************
***********************
 * Sinartisi pou ektelei tin diadikasia tis diastaturwsis gia tin
siggekrimeni genea. Epilegi
 * me pithanotita 0,25 tixea atoma kai ta stellei pros diasstautwsi.
Epilegei ana dio atoma se pio
 * simeio tha diastaurwthoun tixea kai ektelei tin diastaurwsi. Sti
sinexeia ta nea atoma topothetounte
 * stis thesis twn prokatoxwn tous. Se periptwsi pou i epilogi twn atomon
einai monos arithmos
 * tote to teleiteo den diastaurwnetai.

*************************************************************************
***********************/
void ga :: crossover(){

double temp;
int x,temp2,correct,maxIter,flag;
vector <chromosome> tempAtoms;
vector <int> tempAtom1;
```

```
vector <int> tempAtom2;
vector <int> tempPlace;

	for(int i=0;i<POPULATION;i++){

		temp=(rand()%1000000)/10000.0;


		if(temp<25){
			tempAtoms.push_back(next_generation[i]);

		}
	}



	for(int p=0;p<CHROMOSOME_LENGTH;p++){

		tempAtom1.push_back(0);
		tempAtom2.push_back(0);
		tempPlace.push_back(0);

	}

	x=0;
	while((x+2)<tempAtoms.size()){

		correct=0;
		maxIter=0;
		flag=0;

			tempAtoms[x].returnTempAtom(tempAtom1);

			tempAtoms[x+1].returnTempAtom(tempAtom2);

			temp2=rand()%CHROMOSOME_LENGTH;

			for(int m=0;m<temp2;m++){

				tempPlace[m]=tempAtom1[m];

			}
			for(int m=0;m<temp2;m++){

				tempAtom1[m]=tempAtom2[m];

			}

			for(int m=0;m<temp2;m++){

				tempAtom2[m]=tempPlace[m];

			}
```

```
	next_generation[tempAtoms[x].returnPosition()].crossoveredAtom(temp
Atom1);

	next_generation[tempAtoms[x+1].returnPosition()].crossoveredAtom(te
mpAtom2);


	next_generation[tempAtoms[x].returnPosition()].convert();

	//next_generation[tempAtoms[x].returnPlace()].calEval();


next_generation[tempAtoms[x+1].returnPosition()].convert();

//next_generation[tempAtoms[x+1].returnPlace()].calEval();

		x+=2;
		}

	while(!tempAtom1.empty()){

		tempAtom1.pop_back();
		tempAtom2.pop_back();
		tempPlace.pop_back();

	}

		while(!tempAtoms.empty()){

			tempAtoms.pop_back();

		}
}



/**************************************************************************
***********************
 * Sinartisi i opoia elegxei poio atomo apo tin genea exei to kalitero
evaluation kai epistrefei
 * os apantisi tis sinartisis to siggekrimeno x1 kai x2. Sti sinexeia
elegxei an i apantisi einai
 * kaliteri apo tin oliki apantisi pou exei mexri tin siggekrimeni stigmi
kai an ne tis orizei
 * san nea lisi gia tin sinartisi.

**************************************************************************
**********************/
void ga :: chooseBestAtom(int gen){

	int bestHiddenLayerOneSize=0;
	int bestHiddenLayerTwoSize=0;
	int bestHiddenLayerOneOfBackwardSize=0;
	int bestHiddenLayerTwoOfBackwardSize=0;
	int bestHiddenLayerOneOfForwardSize=0;
```

```
	int bestHiddenLayerTwoOfForwardSize=0;
	//double bestLearningRate=0.0;
	//double bestmomentum=0.0;
	int bestWindowSize=0;
	//double bestqVar=0.0;
	int bestsVar=0;



	double bestEval=0.0;
	double sum_F=0.0;

	for(int i=0;i<POPULATION;i++){

		if(next_generation[i].returnEval()>bestEval){


	bestHiddenLayerOneSize=next_generation[i].returnHiddenLayerOneSize(
);

	bestHiddenLayerTwoSize=next_generation[i].returnHiddenLayerTwoSize(
);

	bestHiddenLayerOneOfBackwardSize=next_generation[i].returnHiddenLay
erOneOfBackwardSize();

	bestHiddenLayerTwoOfBackwardSize=next_generation[i].returnHiddenLay
erTwoOfBackwardSize();

	bestHiddenLayerOneOfForwardSize=next_generation[i].returnHiddenLaye
rOneOfForwardSize();

	bestHiddenLayerTwoOfForwardSize=next_generation[i].returnHiddenLaye
rTwoOfForwardSize();

	//bestLearningRate=next_generation[i].returnLearningRate();
			//bestmomentum=next_generation[i].returnmomentum();
			bestWindowSize=next_generation[i].returnWindowSize();
			//bestqVar=next_generation[i].returnqVar();
			bestsVar=next_generation[i].returnsVar();

			bestEval=next_generation[i].returnEval();
		}
		sum_F+=next_generation[i].returnEval();
	}

	if(bestEval>solutionEval)
	{

		solutionHiddenLayerOneSize=bestHiddenLayerOneSize;
		solutionHiddenLayerTwoSize=bestHiddenLayerTwoSize;

	solutionHiddenLayerOneOfBackwardSize=bestHiddenLayerOneOfBackwardSi
ze;
```

```
	solutionHiddenLayerTwoOfBackwardSize=bestHiddenLayerTwoOfBackwardSi
ze;

	solutionHiddenLayerOneOfForwardSize=bestHiddenLayerOneOfForwardSize
;

	solutionHiddenLayerTwoOfForwardSize=bestHiddenLayerTwoOfForwardSize
;
		//solutionLearningRate=bestLearningRate;
		//solutionmomentum=bestmomentum;
		solutionWindowSize=bestWindowSize;
		//solutionqVar=bestqVar;
		solutionsVar=bestsVar;

		solutionEval=bestEval;
	}


//printf("Generation %d\t: X1=%d\t X2=%d\t Evaluation=%lf\t Generation
Evaluation=%f \t\n",gen,bestx1,bestx2,bestEval,sum_F);
	outf<<"Generation "<<gen<<"\t\t: bestHiddenLayerOneSize =
"<<bestHiddenLayerOneSize<<"\t\t: bestHiddenLayerTwoSize =
"<<bestHiddenLayerTwoSize<<"\t\t: bestHiddenLayerOneOfBackwardSize =
"<<bestHiddenLayerOneOfBackwardSize<<"\t\t:
bestHiddenLayerTwoOfBackwardSize =
"<<bestHiddenLayerTwoOfBackwardSize<<"\t\t:
bestHiddenLayerOneOfForwardSize =
"<<bestHiddenLayerOneOfForwardSize<<"\t\t:
bestHiddenLayerTwoOfForwardSize =
"<<bestHiddenLayerTwoOfForwardSize<<"\t\t: bestWindowSize =
"<<bestWindowSize<<"\t\t: bestsVar = "<<bestsVar<<"\t\t Evaluation =
"<<bestEval<<"\n\n";

//cout<<"Best atom's score for "<<gen<<" generation :"<<bestEval<<endl;

}

void ga :: printSolution()
{
	outf<<"**********************************************************
******************"<<endl;
	outf<<"solutionHiddenLayerOneSize =
"<<solutionHiddenLayerOneSize<<"\t\t solutionHiddenLayerTwoSize =
"<<solutionHiddenLayerTwoSize<<"\t\t:
solutionHiddenLayerOneOfBackwardSize =
"<<solutionHiddenLayerOneOfBackwardSize<<"\t\t:
solutionHiddenLayerTwoOfBackwardSize =
"<<solutionHiddenLayerTwoOfBackwardSize<<"\t\t:
solutionHiddenLayerOneOfForwardSize =
"<<solutionHiddenLayerOneOfBackwardSize<<"\t\t:
solutionHiddenLayerTwoOfForwardSize =
"<<solutionHiddenLayerTwoOfBackwardSize<<"\t\t: solutionWindowSize =
"<<solutionWindowSize<<"\t\t: solutionsVar = "<<solutionsVar<<"\t\t
Evaluation = "<<solutionEval<<"\n\n";
```

```
	outf.close();
}

void ga :: printCh()
{
	for(int i=0;i<20;i++)
	{
		next_generation[i].print();

	}
}
```

pssp_ucy.cpp (only for GAs)

```
//included libraries
#include "DataReader.h"
#include "Neuron.h"
#include "BidirectionalRecurrentNeuralNetwork.h"
#include "chromosome.h"
#include "ga.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;

	ga *gaAlgorithm = new ga();

	//object of the DataReader class that reads the program's
parameters from the parameter file
	DataReader *getInput=new DataReader();

	BidirectionalRecurrentNeuralNetwork *BRNN;

	//Parameters' entry
	getInput-
>readParameters(parameters,inputProfile,outputProfile,trainFile,testFile,
&msaEnable,&randomizeDataset);
```

```
	//about GA
	int generations=0;
	int hLayerOneSize;
	int hLayerTwoSize;
	int hLayerOneSizeB;
	int hLayerTwoSizeB;
	int hLayerOneSizeF;
	int hLayerTwoSizeF;
	int activationFuncTypeOutput;
	int errorFunctionTypeOutput;
	int activationFuncTypeHidden;
	int errorFunctionTypeHidden;
	double learningRate;
	double momentum;
	int windowSize;
	double qMinus1;
	double qPlus1;
	int errorFunctionType;
	int s;
	int maxIterations;


	//katagrafi protwn minimatwn sto arxeio apotelesmatwn
	gaAlgorithm->outf<<"Results:"<<endl;
	gaAlgorithm->outf<<"********"<<endl;

	while(generations<15)
	{
		//gaAlgorithm->printCh();

		gaAlgorithm->createRoulette();
		gaAlgorithm->newPopulation();
		gaAlgorithm->crossover();
		gaAlgorithm->mutation();

		//in the end, for every atom of the population it calculates
the value of each parameter and for this atom, executes the program!!
		for(int i=0;i<POPULATION;i++)
		{

			//initiate the program's parameter
			gaAlgorithm->next_generation[i].convert();

			hLayerOneSize = gaAlgorithm-
>next_generation[i].returnHiddenLayerOneSize();
			hLayerTwoSize = 0;//gaAlgorithm-
>next_generation[i].returnHiddenLayerTwoSize();
			hLayerOneSizeB = gaAlgorithm-
>next_generation[i].returnHiddenLayerOneOfBackwardSize();
			hLayerTwoSizeB = 0;//gaAlgorithm-
>next_generation[i].returnHiddenLayerTwoOfBackwardSize();
			hLayerOneSizeF = gaAlgorithm-
>next_generation[i].returnHiddenLayerOneOfBackwardSize();
			hLayerTwoSizeF = 0;//gaAlgorithm-
>next_generation[i].returnHiddenLayerTwoOfBackwardSize();
```

```
                  activationFuncTypeOutput = 1;
                  errorFunctionTypeOutput = 1;
                  activationFuncTypeHidden = 1;
                  errorFunctionTypeHidden = 1;
                  learningRate = 0.09;//gaAlgorithm-
>next_generation[i].returnLearningRate();
                  momentum = 0.5;//gaAlgorithm-
>next_generation[i].returnmomentum();
                  windowSize = gaAlgorithm-
>next_generation[i].returnWindowSize();
                  qMinus1 = 1.0/ 0.6;//(gaAlgorithm-
>next_generation[i].returnqVar());
                  qPlus1 = 0.6; //gaAlgorithm-
>next_generation[i].returnqVar();
                  errorFunctionType = 1;
                  s = gaAlgorithm->next_generation[i].returnsVar();
                  maxIterations = 200;


                  /*if(hLayerOneSize==0 || hLayerOneSizeB==0 ||
hLayerOneSizeF==0 || learningRate<0.0000001 || momentum<0.000001 ||
windowSize==0 || qMinus1<0.00001 || s==0)
                  {
                        gaAlgorithm->setEvaluation(0.0,i);
                        //free(BRNN);
                        continue;
                  }*/

                  BRNN=new
BidirectionalRecurrentNeuralNetwork(hLayerOneSize,hLayerTwoSize,hLayerOne
SizeB,hLayerTwoSizeB,hLayerOneSizeF,

      hLayerTwoSizeF,activationFuncTypeHidden,activationFuncTypeOutput,le
arningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionTypeHidden,errorFunctionTypeOutput,s,ma
xIterations,trainFile,testFile,
                  inputProfile,outputProfile,msaEnable,randomizeDataset);

            //the main loop of the program.
            epoch=0;

            while(epoch<maxIterations)
            {
                  //open pointers to the output files
                  BRNN->openFolders();

                  //
                  endOfFile=BRNN-
>initTrainingInput(&primaryStructureSize);

                  //
                  while(endOfFile)
                  {
                        if(msaEnable==1) BRNN->getMSAInput();
```

```
                        for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
                        {
                              //
                              BRNN->doFeedForward(sequencet,1,epoch);
                              BRNN-
>doBackpropagation(sequencet,learningRate,1);
                        }

                        //
                        BRNN->getSequenceTrainingError();

                        if(epoch==maxIterations-1)
                              BRNN->printOutputSequence(1);

                        //
                        endOfFile=BRNN-
>initTrainingInput(&primaryStructureSize);
                  }

                  //
                  endOfFile=BRNN->initTestInput(&primaryStructureSize);

                  //
                  while(endOfFile)
                  {
                        if(msaEnable==1) BRNN->getMSAInput();


                        for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
                        {
                              BRNN->doFeedForward(sequencet,2,epoch);
                        }

                        BRNN->getSequenceTestingError();

                        if(epoch==maxIterations-1)
                              BRNN->printOutputSequence(2);

                        endOfFile=BRNN-
>initTestInput(&primaryStructureSize);
                  }

                  //
                  BRNN->getEpochError();

                  //
                  BRNN->closeFolders();


                  cout<<"Generation:"<<generations<<"
Population:"<<i<<"   Epoch:"<<epoch<<endl;

                  epoch++;
```

```
            }//teliwse to programma (to big while)

            BRNN->printOutputs();

            BRNN->printNetworkSpecification();

            gaAlgorithm->setEvaluation(BRNN-
>returnSuccessPercentageAverage(),i);

            free(BRNN);

            }//teliwse gia kapio atom, vrike to evaluation tou
sigkekrimenou atomou kai paei na piasei to next atom tou plithismou

      //afou teliwsei gia olokliro ton plithismo, pernei to kalytero
atomo me vasi to evaluation pou edwsan
      gaAlgorithm->chooseBestAtom(generations);

      generations++;

      }

      gaAlgorithm->printSolution();

      return 0;
}
```

ΠΑΡΑΡΤΗΜΑ Δ

Αρχείο Δεδομένων Εισόδου

1bujA

AVINTFDGVADYLIRYKRLPDNYITKSQASALGWVASKGNLAEVAPGKSIGGDVFSNREGRLPSASGRTWREADI
NYVSGFRNADRLVYSSDWLIYKTTDHYATFTRIR.

LLLLSHHHHHHHHHHHSSLLTTEELHHHHHHHTLLSSSSLHHHHSTTLEEEEEELLLLSSSSLLSSSLLEEEEESSLSSS
SLLLSEEEEETTLLEEEESSSSSSLEELL.

2el8A

GSSGSSGEVLAKEEARRALETPSCFLKVSRLEAQLLLERYPECGNLLLRPSGDGADGVSVTTRQMHNGTHVVRHY
KVKREGPKYVIDVEQPFSCTSLDAVVNYFVSHTKKALVPFLLD.

LLLLLLLLLLLSLLLLLLSSSLTTLLLLLHHHHHHHHHHSSTTLSBEEEELLSSSSSEEEEELLLBTTBLLLEEELBLLBTTBL
LBSSSSLLLLSSHHHHHHHHHHHHSSSLLLBLLLL.

1co4A

MVVINGVKYACDSCIKSHKAAQCEHNDRPLKILKPRGRPPTT.

LEEETTEEEEETTTTTTSGGGGLLLLSSLEEEELSLLLLSLL.

1edsA

TTLYTSLHGYFVFGPTGCNLEGFFATLGGEI.

LLLSSTTSSLGGGLSSTTLIIIIIIIIIILL.

2byeA

GEERKCLQTHRVTVHGVPGPEPFTVFTINGGTKAKQLLQQILTNEQDIKPVTTDYFLMEEKYFISKEKNECRKQPF
QRAIGPEEEIMQILSSWFPEEGYMGRIVLKTQQE.

LLLSSLLLLLEEEESSLSSSSSSEEEELLTTLLHHHHHHHHHHLLSSSSLLLLLSEEEEEELLLLSSSSLSLTTSLLEEEELSSSL
HHHHHHHLLTTTTTLLLEEEEESLL.

1hf9A

ALKKHHENEISHHAKEIERLQKEIERHKQSIKKLKQSEDDD.

LLSSHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHLL.

ΠΑΡΑΡΤΗΜΑ Ε

Αρχείο Δεδομένων Εξόδου

2e5tA Correctness Percentage:78.2609%

primaryStructure :IDVLRAKAAKERAERRLQSQQDDIDFKRAELALKRAMNRLSVAEMK

secondaryStructure :LLHHHHHHHHHHHHHHHLLLLLLLLHHHHHHHHHHHHHHHHHHHLL

predictedSecondaryStructure:LLLHHHHHHHHHHLHLLLLLLLLLHHLHHHHHHHHHHHHHLLLLLL

2e60A Correctness Percentage:84.1584%

primaryStructure
:GSSGSSGVAPLGLSVPSDVELPPTAKMHAIIERTASFVCRQGAQFEIMLKAKQARNSQFDFLRFDHYLNPYYKFI
QKAMKEGRYTVLAENKSDEKKKSGVS

secondaryStructure
:LLLLLLLLLLLLLLLLLLLLLLLLHHHHHHHHHHHHHHHHHHLLHHHHHHHHHHLLLLLLLHHHLLLLLHHHHHHHHHH
HHHHHLLLLLLLLLLLLLLLLLLL

predictedSecondaryStructure:LLLLLLLLLLLLLLLLLLLLLLLHHHHHHHHHLLLLEHLLLLLHHHHHHHLLLLLLLL
LLELLLLLLLLLHHHHHHHHHHHLLLLLLLLLLLLLLLLLL

2e61A Correctness Percentage:82.6087%

primaryStructure
:GSSGSSGEISGFGQCLVWVQCSFPNCGKWRRLCGNIDPSVLPDNWSCDQNTDVQYNRCDIPEETWTGLE

secondaryStructure
:LLLLLLLLLLLLLLLLEEELLLLLLLEEELLLLLLLLLLLLLLHHHLLLHHHLLLLLLLLLLLL

predictedSecondaryStructure:LLLLLLLLLLLLLLLLELLLLLLLLLELLLLLLLLLLLLLLELEELLLLLLLLLLLLLLLEL
LLLL

2e62A Correctness Percentage:72.1311%

primaryStructure
:GSSGSSGNGMDEEQRQKRRRIEVALIEYRETLEEQGMKNPEEIERKVEINRKRLEVDYGLS

secondaryStructure
:LLLLLLLLLLHHHHHHHHHHHHHHHHHHHHHHHHHLLLLHHHHHHHHHHHHHHHHHLLLL

predictedSecondaryStructure:LLLLLLLLLHHHLLHHHHHHHHELLLHHHHLHLLLLLLLLHHHLHHHHHHLLL
HHHLLLLL

ΠΑΡΑΡΤΗΜΑ Στ

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 0.27 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 1 0 0 0 0 0 0
0 0 0.17 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 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 1 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 Ζ

Αρχείο Κωδικοποίησης Εξόδου

Three_Classes

****

Classes_volume 3

****

H G,H

E E,B

L I,T,S,L

Output_Encoding

****

H 100

E 001

L 010

ΠΑΡΑΡΤΗΜΑ Η

Αρχείο Παραμέτρων Αρχικοποίησης

Hidden_layer_one_size 14

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 0.7

q_plus_one 0.7

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 msaProteinsTrainBigDataset_afterProcess.txt

test_File msaProteinsTestBigDataset_afterProcess.txt

/*********OtherParams*********/

msa_Enable 1

randomizeDataset 1