Ατομική Διπλωματική Εργασία

ΠΡΟΒΛΕΨΗ ΔΕΥΤΕΡΟΤΑΓΟΥΣ ΔΟΜΗΣ ΠΡΩΤΕΙΝΩΝ ΜΕ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΑΜΦΙΔΡΟΜΗΣ ΑΝΑΔΡΑΣΗΣ

Μιχάλης Αγαθοκλέους

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών Με Νευρωνικά Δίκτυα Αμφίδρομης Ανάδρασης

Μιχάλης Αγαθοκλέους

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 2009

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή, Δρ. Χρίστο Χριστοδούλου, που με εμπιστεύθηκε για την εκπόνηση της παρούσας διπλωματικής εργασίας. Θα ήθελα να τον ευχαριστήσω για την εξαιρετική συνεργασία και την πολύτιμη βοήθεια που μου προσέφερε καθοδηγώντας με καθ' όλη την διάρκεια της εργασίας. Οι γνώσεις και οι εμπειρίες που μου πρόσφερε μέσα από αυτή την διπλωματική εργασία θεωρώ ότι είναι πολύ σημαντικές για την μετέπειτα καριέρα μου.

Εν συνεχεία, θα ήθελα να ευχαριστήσω τον Δρ. Βασίλειο Προμπονά, Λέκτορα του Τμήματος Βιολογίας του Πανεπιστημίου Κύπρου, για την άμεση και πλήρης βοήθεια που μου πρόσφερε σχετικά με θέματα που αφορούσαν την επιστήμη της Βιολογίας. Επίσης, θα ήθελα να ευχαριστήσω τον Αντώνη Αντωνίου, μεταπτυχιακό φοιτητή, για την συνεργασία του.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου για την αμέριστη συμπαράσταση και καθοδήγηση που μου προσφέρουν.

Περίληψη

Το έγγραφο αυτό αναφέρεται σε έρευνα που αφορά το πρόβλημα της πρόβλεψης δευτεροταγούς δομή πρωτεϊνών, ένα πρόβλημα που αφορά κατά κύριο λόγο τις επιστήμες της Βιολογίας και της Πληροφορικής.

Οι πρωτεΐνες αποτελούν ένα από τα σημαντικότερα στοιχεία του ανθρωπίνου σώματος, αλλά και κάθε βιολογικά ζωντανού οργανισμού. Η γνώση της ακριβούς δομής των πρωτεϊνών στον τρισδιάστατο χώρο είναι πολύ σημαντική, αφού μόνο έτσι μπορούν να κατασκευαστούν φάρμακα ή και άλλα προϊόντα που αφορούν την λειτουργία αυτών των μακρομορίων. Το πρόβλημα όμως που παρουσιάζεται σε αυτό το σημείο είναι ότι οι διαδικασίες για εξαγωγή της τρισδιάστατης δομής τους είναι χρονοβόρες και πολύπλοκες. Αντίθετα, πολύ εύκολα οι επιστήμονες μπορούν να καταγράψουν την ακολουθία των αμινοξέων, των δομικών στοιχείων των πρωτεϊνών. Το γεγονός αυτό έχει ως αποτέλεσμα, παρόλο που σήμερα βρίσκονται καταγεγραμμένες μερικά εκατομμύρια από αμινοξικές ακολουθίες, μόνο μερικών χιλιάδων ακολουθιών είναι γνωστή η τρισδιάστατη μορφή της αντίστοιχης πρωτεΐνης.

Σκοπός της έρευνας αυτής ήταν αρχικά η μελέτη τεχνικών και αλγορίθμων που χρησιμοποιήθηκαν για πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών, η οποία δίνει πληροφορίες για την τρισδιάστατη μορφή τους, μέσα από την αντίστοιχη πρωτοταγή δομή τους, που αντιπροσωπεύει ακολουθία αμινοξέων. Στη συνέχεια η έρευνα αυτή ασχολείται με το πώς τα Νευρωνικά Δίκτυα μπορούν να δώσουν λύση στο συγκεκριμένο πρόβλημα με την κατασκευή ενός Νευρωνικού Δικτύου αμφίδρομης ανάδρασης, το οποίο θα εκπαιδευτεί με τον αλγόριθμο ανάστροφης μετάδοσης λάθους και θα μπορεί να προβλέψει την δευτεροταγή δομή πρωτεϊνών δεχόμενο στην είσοδό του μόνο την πρωτοταγή δομή τους.

Το Νευρωνικό Δικτύου με αμφίδρομη ανάδραση υλοποιήθηκε, έχοντας αρχικά αποτελέσματα μέχρι και 64% επιτυχία. Τα αποτελέσματα αυτά θεωρούνται ικανοποιητικά σε αρχική φάση, αναλογιζόμενοι το γεγονός ότι η καλύτερη επίδοση σε αντίστοιχες έρευνες ήταν 76%, με την προϋπόθεση ότι το δίκτυο αυτό θα βελτιστοποιηθεί στο μέλλον ώστε να προσφέρει ακόμα καλύτερα αποτελέσματα.

Περιεχόμενα

Λίστα Ακρωνύμιων

BRRN	Bidirectional Recurrent Neural Network
DNA	Deoxyribonucleic acid
DSSP	Dictionary for Secondary Structure of Proteins
HMM	Hidden Markov Model
LAD	Logical Analysis of Data
LMS	Least Mean Square Error
MLP	Multilayer Perceptron
NMR	Nuclear Magnetic Resonance, Πυρηνικός Μαγνητικός Συντονιστής
PIR	Protein Information Resource
PSSP	Protein Secondary Structure Prediction
RNA	Ribonucleic acid
RNN	Recurrent Neural Network

Κεφάλαιο 1

Εισαγωγή

1.1 Ο ρόλος και ο στόχος της έρευνας

Οι πρωτεΐνες αποτελούν ένα από τα σημαντικότερα στοιχεία του ανθρωπίνου σώματος, αλλά και κάθε βιολογικά ζωντανού οργανισμού. Συγκεκριμένα μπορούμε να πούμε ότι τα διαφορετικά είδη πρωτεϊνών, είτε σαν δομικές είτε σαν λειτουργικές μονάδες διασφαλίζουν τις περισσότερες ανάγκες για τη βιολογική λειτουργία του ανθρώπινου οργανισμού. Η λεπτομερής γνώση της δομής των πρωτεϊνών έχει τεράστια σημασία για τις επιστήμες της βιολογίας, της ιατρικής, αλλά και της φαρμακευτικής. Ο λόγος που θεωρείται σημαντική η γνώση της δομής των πρωτεϊνών οφείλεται στο ότι είναι ο μοναδικός τρόπος παραγωγής διαφόρων φαρμάκων που σχετίζονται με αυτές και στόχο έχουν την καλύτερη ποιότητα ζωής του ανθρώπινου γένους, κυρίως όσον αφορά το κεφάλαιο υγεία. Μερικά παραδείγματα είναι ότι μπορούν να κατασκευαστούν διάφορα φάρμακα τα οποία αναστέλλουν τη λειτουργία των πρωτεϊνών ή ενζύμων που εκτελούν συγκεκριμένες λειτουργίες οι οποίες δημιουργούν προβλήματα στον ανθρώπινο οργανισμό. Η γνώση της δομής των πρωτεϊνών εμπλέκεται και με άλλα κεφάλαια που αφορούν την ποιότητα ζωής, όπως την κατασκευή φυτοφαρμάκων για καταπολέμηση ασθενειών στα διάφορα φυτά ή κατασκευή ουσιών οι οποίες βοηθούν τα φυτά να μεγαλώνουν με ταχύτερο ρυθμό και να παράγουν περισσότερους καρπούς. Επίσης, άλλες βιομηχανίες, γνωρίζοντας τη δομή των πρωτεϊνών κατασκευάζουν τεχνητές πρωτεΐνες οι οποίες βοηθούν στην αύξηση της μυϊκής μάζας του ανθρώπου.

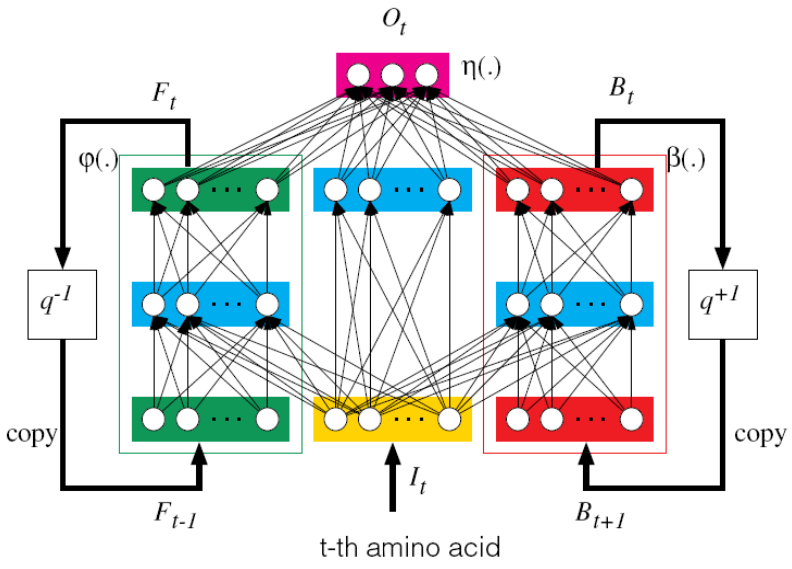
Για τους πιο πάνω λόγους, η μελέτη της δομής των πρωτεϊνών αποτελεί ένα επίκαιρο και κρίσιμο θέμα της σύγχρονης έρευνας. Η έρευνα όμως σχετικά με το θέμα αυτό παρουσιάζει πολλές δυσκολίες. Από την πρωτοταγή δομή μιας πρωτείνης, δηλαδή την αλληλουχία των αμινοξέων τα οποία αποτελούν την δομική μονάδα των πρωτεϊνών, και την οποία εύκολα μπορούμε να καταγράψουμε για κάθε πρωτεΐνη, δεν μπορούμε να εξαγάγουμε τις απαραίτητες πληροφορίες ώστε να δημιουργήσουμε την αντίστοιχη τρισδιάστατη μορφή της, δηλαδή την δευτεροταγή και τριτοταγή δομή της συγκεκριμένης πρωτεΐνης. Συγκεκριμένα στις βάσεις δεδομένων υπάρχουν καταγεγραμμένες περίπου 3,500,000 πρωτεϊνικές ακολουθίες, από τις οποίες μόνο για 53000 γνωρίζουμε τη δευτεροταγή δομή τους (RCSB Protein Data Bank, 20 April 2009). Αυτό οφείλεται στο ότι τα αμινοξέα που

αποτελούν το μακρομόριο της πρωτεΐνης δεν παρέχουν άλλα στοιχεία εκτός από την σειρά και τον αριθμό τους. Τα αμινοξέα όμως αποτελούνται από μικρότερα μόρια τα οποία είναι φορτισμένα αρνητικά ή θετικά, και λόγω των φορτίων αυτών, αλλά και άλλων παραγόντων, αλληλεπιδρούν μεταξύ τους και δημιουργούν την τρισδιάστατη μορφή της πρωτεΐνης. Τα αμινοξέα χρησιμοποιώντας τον πεπτίδιο δεσμό συνδέονται μεταξύ τους σε σειρές, ενώ με τις διάφορες αλληλεπιδράσεις που δημιουργούνται μεταξύ των πλευρικών τους αλυσίδων, αναδιπλώνονται στο χώρο και δημιουργούν τη τρισδιάστατη μορφή της πρωτεΐνης. Οι αλληλεπιδράσεις μεταξύ των πλευρικών αλυσίδων μπορεί να είναι από ένα οποιοδήποτε προς άλλο οποιοδήποτε αμινοξύ της αλληλουχίας. Αυτό δημιουργεί ένα τεράστιο αριθμό πιθανών συνδυασμών αλληλεπιδράσεων. Σημαντικό εδώ είναι το ότι μια συγκεκριμένη σειρά και αριθμός αμινοξέων δημιουργούν πάντοτε τις ίδιες αλληλεπιδράσεις μεταξύ τους, με αποτέλεσμα να έχουμε πάντοτε την ίδια πρωτεΐνη με την ίδια ακρίβεια στην δομή της. Επίσης, η απευθείας μελέτη της τρισδιάστατης μορφής των πρωτεϊνών αποτελεί χρονοβόρα και δαπανηρή διαδικασία.

Το πιο πάνω πρόβλημα ονομάζεται πρόβλεψη δευτεροταγούς δομής προτείνων (PSSP) και για λύση του καταφεύγουμε στη κατηγορία των αλγορίθμων Νευρωνικών Δικτύων. Στα νευρωνικά δίκτυα καταφεύγουμε λόγω του τεραστίου αριθμού των πιθανών συνδυασμών μεταξύ των αμινοξέων και της μη ύπαρξης εφικτής μαθηματικής συνάρτησης η οποία με δεδομένο την πρωτοταγή δομή μιας πρωτεΐνης να δίνει ως αποτέλεσμα την δευτεροταγή δομή της. Συγκεκριμένα καλούμαστε να υλοποιήσουμε ένα νευρωνικό δίκτυο με αμφίδρομη αναδρομή το οποίο θα δέχεται στην είσοδό ένα συγκεκριμένο αμινοξύ μαζί με κάποια σειρά αμινοξέων που προηγούνται και έπονται αυτού του αμινοξέως, δηλαδή μια σειρά από αμινοξέα και θα προβλέπει στην έξοδο του σε ποια από τρεις κατηγορίες δευτεροταγούς δομής ανήκει το συγκεκριμένο σημείο της πρωτεΐνης. Συγκεκριμένα μέσα από την διαδικασία αυτή θα υλοποιήσουμε ένα δίκτυο το οποίο θα προβλέπει την δευτεροταγή δομή μιας πρωτεΐνης για κάποιο αμινοξύ, συσχετίζοντάς το με τα αμινοξέα, άρα και τις δυνάμεις, που βρίσκονται «κοντά» του και μπορεί να επηρεάζουν τη δομή της πρωτεΐνης στο συγκεκριμένο σημείο. Έτσι αφού περάσουμε από την είσοδο όλα τα αμινοξέα, μαζί με τα αντίστοιχα που προηγούνται και έπονται μιας συγκεκριμένης πρωτεΐνης, με τη σειρά που παρουσιάζονται στην πρωτεΐνη, θα έχουμε στην έξοδο του

νευρωνικού δικτύου την πιθανή μορφή δευτεροταγούς δομής της πρωτεΐνης που αντιστοιχεί στο κάθε αμινοξύ της εισόδου. Το δίκτυο θα εκπαιδευτεί με πρωτεΐνες για τις οποίες γνωρίζουμε την πρωτοταγή και δευτεροταγή δομή τους.

Σε μια πιο λεπτομερή ανάλυση της υλοποίησης, θα κατασκευάσουμε ένα Νευρωνικό Δίκτυο με αμφιδρομική ανάδραση (BRNN). Το δίκτυο θα έχει αυτή τη δομή διότι θα προβλέπει τη δευτεροταγή δομή του κάθε αμινοξέως με βάση τα αμινοξέα που προηγούνται και έπονται αυτού. Θα χρησιμοποιήσουμε Νευρωνικό Δίκτυο με ανάδραση διότι πρέπει το ίδιο το δίκτυο να κρατά πληροφορίες σε κάθε χρονική στιγμή για προηγούμενες και επόμενες καταστάσεις της σειράς των δεδομένων, αφού οποιοδήποτε αμινοξύ μπορεί να επηρεάσει τη δευτεροταγή δομή του συγκεκριμένου σημείου που ελέγχουμε. Το δίκτυο μας, πρέπει να υλοποιηθεί με αμφίδρομη ανάδραση διότι η δομή του σημείου της πρωτεΐνης που ελέγχουμε, εξαρτάται από τα αμινοξέα που προηγούνται και έπονται του σημείου αυτού. Το Νευρωνικό Δίκτυο με αμφίδρομη ανάδραση έχει ήδη υλοποιηθεί χρησιμοποιώντας ένα αλγόριθμο μάθησης, βασισμένο στον αλγόριθμο Αναστροφής μετάδοσης λάθους (Backpropagation) (Σχήμα 1.1) από τον Baldi το 1999 (Baldi et al 1999). Η υλοποίηση αυτή είχε την μεγαλύτερη επιτυχία με 76% ποσοστό επιτυχίας στην προσπάθεια επίλυσης του συγκεκριμένου προβλήματος.



Σχήμα 1.1: *Το νευρωνικό δίκτυο αμφίδρομης ανάδρασης του Baldi (1999)*

1.2 Σχετική Έρευνα

Τα τελευταία 30 χρόνια έχουν αναπτυχθεί διάφοροι αλγόριθμοι οι οποίοι προσπαθούν να προβλέψουν τη δευτεροταγή δομή των πρωτεϊνών. Τα αποτελέσματα των αλγόριθμων αυτών κυμαίνονται σε αρκετά υψηλά ποσοστά επιτυχίας Q3 (Εξίσωση 1.1) (Richards and Kundrot 1988), δηλαδή πόσο ποσοστό επιτυχίας υπάρχει σε επίπεδο αμινοξέως προς αμινοξέως για μια άγνωστη προς το δίκτυο πρωτεϊνική ακολουθία. Τα ποσοστά αυτά όμως, τα οποία κυμαίνονται από 63% μέχρι 76%, δεν θεωρούνται ικανοποιητικά όσον αφορά την ολική εξεύρεση λύσης στο πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών. Αυτό οφείλεται κυρίως στον τεράστιο όγκο δεδομένων που εμπερικλείεται έμμεσα στις ακολουθίες των πρωτεϊνικών αμινοξέων. Οι σημαντικότεροι αλγόριθμοι που χρησιμοποιήθηκαν για το σκοπό αυτό και το ποσοστό επιτυχίας Q3 για τον κάθε ένα παρουσιάζονται στον σχετικό πίνακα (Πίνακας 1.1), με το Νευρωνικό Δίκτυο αμφίδρομης ανάδρασης του Baldi (1999) (Baldi et al 1999, Baldi et al 2000) να παρουσιάζει το υψηλότερο ποσοστό επιτυχίας.

$$Q_3 = 100\ \frac{1}{N_{res}} \sum_{i=0}^{3} M_{ii}$$

Εξίσωση 1.1: *Εξίσωση αξιολόγησης ποσοστού επιτυχίας, όπου N_{res} η ποσότητα των αμινοξικών καταλοίπων και M_{ij} να παίρνει την τιμή 0 αν η έξοδος i είναι διαφορετική από την έξοδο j και την τιμή 1 αν η έξοδος i είναι όμοια με την έξοδο j*

Μέσα από μια ιστορική αναδρομή σχετικά με τις προσπάθειες που έγιναν για επίλυση τους του προβλήματος σχετικά με την πρόβλεψη δευτεροταγούς δομής πρωτεϊνών μπορούμε να ξεχωρίσουμε μια σειρά από ερευνητές που με τη δουλειά τους προσπάθησαν να δώσουν κάποια λύση. Οι μεθόδοι που χρησιμοποιήθηκαν αφορούσαν κυρίως υλοποιήσεις νευρωνικών δικτύων και στατιστικές μεθόδους. Εδώ αξίζει να αναφερθεί ότι πιο κάτω παραθέτονται οι σημαντικότερες προσεγγίσεις προς το πρόβλημα, αφού οι μέθοδοι που

χρησιμοποιήθηκαν ανέρχονται σε μεγάλο αριθμό. Ακολουθεί μια σύντομη αναφορά σχετικά με τις προσπάθειες αυτές.

1. Feedforward Fully Connected NN (Qian and Sejnowski 1988): Είναι ένα πλήρως συνδεδεμένο Νευρωνικό Δίκτυο με παράθυρο εισόδου (local input window) συνήθως $13^{ων}$ αμινοξέων ορθογώνιας κωδικοποίησης (orthogonal encoding) και μόνο ένα κρυφό επίπεδο. Η έξοδος του δικτύου ήταν μια εκ των τριών κατηγοριών της δευτεροταγούς δομής των πρωτεϊνών (ελικοειδής, πτυχωτή ή κάτι άλλο) που αναφερόταν στο αμινοξύ που βρισκόταν στο κέντρο του παραθύρου εισόδου. Για την συγκεκριμένη υλοποίηση χρησιμοποιήθηκε και ένα δεύτερο διαδοχικό δίκτυο το οποίο βελτίωνε τα δεδομένα εξόδου του προηγούμενου δικτύου. Η μέθοδος αυτή αντιμετώπιζε προβλήματα υπερεκπαίδευσης (overfitting problem).

2. PHD (Rost and Sander 1993): Η δομή του δικτύου αυτού είναι η ίδια με το «Feedforward Fully Connected NN», με την διαφορά ότι χρησιμοποιήθηκαν κάποιες μεθόδοι για την αντιμετώπιση του προβλήματος υπερεκπαίδευσης. Η πρώτη μέθοδος που χρησιμοποιήθηκε είναι το γρήγορο σταμάτημα (early stopping) και η δεύτερη ο συνολικός μέσος όρος (ensemble average) εκπαιδεύοντας διάφορα δίκτυα ταυτόχρονα με διαφορετικά δεδομένα και μεθόδους μάθησης. Επίσης στα δεδομένα εισόδου στο σύστημα χρησιμοποίησαν τεχνικές, όπως η πολλαπλή στοίχιση (multiple alignment), για εκμετάλλευση της εξελικτικής πληροφορίας.

3. NNSSP (Salamov and Soloveyev 1997): Αυτό το Νευρωνικό Δίκτυο χρησιμοποιεί την μέθοδο του κοντινότερου-γείτωνα (nearest-neighbor method) ούτως ώστε να ομαδοποιήσει τις ακολουθίες ανάλογα με τις ομοιότητές τους και να τις συγκρίνει με άλλες ακολουθίες των οποίων η δευτεροταγής δομή τους είναι γνωστή. Στη συνέχεια με αυτή τη γνώση προσπαθεί να προβλέψει την δευτεροταγή δομή των αγνώστων πρωτεϊνών.

4. DSC (King and Sternberg 1996): Ο αλγόριθμος αυτός ομαδοποιεί σε διάφορες κατηγορίες τα δεδομένα εξόδου του δικτύου και στη συνέχεια με απλές και

γραμμικές στατιστικές μεθόδους προσπαθεί να προσεγγίσει την ακριβή δευτεροταγή δομή των πρωτεϊνών.

5. PREDATOR (Frishman and Argos 1995): Εφαρμόστηκε σε Νευρωνικό Δίκτυο το οποίο δέχεται στην είσοδό του μια ακολουθία από αμινοξέα και προσπαθεί να προβλέψει την αντίστοιχη δευτεροταγή δομή με βάση πιθανούς δεσμούς υδρογόνου που ενδεχομένως να υπάρχουν στην ακολουθία εισόδου.

6. Consensus (Cuff and Barton 1999): Η μέθοδος αυτή αναφέρεται σε Νευρωνικό Δίκτυο το οποίο είχε στην είσοδό του πολλαπλή στοίχιση (multiple alignment) στην οποία έβαζαν επιπλέον στοιχεία για την πρωτεΐνη αντί για μια απλή ακολουθία αμινοξέων. Το δίκτυο αυτό προσπαθούσε να εντοπίσει τις ομοιότητες της εισόδου με άλλες ακολουθίες αμινοξέων, να εντοπίσει τις ομοιότητές τους σε γενετικό κώδικα, εξελικτική ιστορία και κοινές βιολογικές λειτουργίες και ανάλογα να προβλέψει την δευτεροταγή τους δομή.

7. Bidirectional Recurrent Neural Network (BRNN) – Backpropagation (Baldi et al 1999,Baldi et al 2000): Ο αλγόριθμος που είχε την σχετικά την καλύτερη απόδοση στην πρόβλεψη δευτεροταγούς δομής πρωτεϊνών (76% επιτυχές αποτέλεσμα). Ο αλγόριθμος αυτός εφαρμόζεται με Νευρωνικό Δίκτυο το οποίο δέχεται στην είσοδό του ένα παράθυρο με μια ακολουθία από αμινοξέα. Το δίκτυο προσπαθεί να προβλέψει την δευτεροταγή δομή του αμινοξέως που βρίσκεται στο κέντρο του παραθύρου εισόδου με βάση τα αμινοξέα που προηγούνται και έπονται αυτού στην αλυσίδα εισόδου χρησιμοποιώντας αμφίδρομη αναδρομή.

8. LAD (Blazewicz et al 2005): Για εξεύρεση λύσης στο συγκεκριμένο πρόβλημα σημαντική θεωρείται και η έρευνα γύρω από την δομή και τις ιδιότητες των αμινοξέων. Μια τέτοια έρευνα εξάγεται μέσα από την υλοποίηση το αλγόριθμου LAD που αποτελεί ένα αλγόριθμο μηχανικής μάθησης. Ο συγκεκριμένος αλγόριθμος υλοποιήθηκε με σκοπό να αναγνωρίσει ιδιότητες των αμινοξέων που μπορεί να παρέχουν επιπλέων πληροφορίες για την ομοιογένεια των πρωτεϊνών,

κάτι το οποίο μπορεί να βοηθήσει στην πρόβλεψη δευτεροταγούς δομής των πρωτεϊνών. Η συγκεκριμένη μέθοδος είχε σχετικά καλά αποτελέσματα ενώ οδήγησε σε συμπεράσματα που αφορούν τις ιδιότητες των αμινοξέων και τη σχέση που έχουν αυτά στην πρόβλεψη δευτεροταγούς δομής πρωτεϊνών. Συγκεκριμένα η πιο σημαντική ιδιότητα που επηρεάζει την κλάση των ελίκων είναι τα μοριακά βάρη, για την κλάση των πτυχωτών είναι η μέση περιβαλλόμενη υδροφοβία και για τις υπόλοιπες μορφές η πολικότητα .

9. MASSP3 (Armano et al 2006): Αυτή η προσέγγιση προσπαθεί να λύσει το συγκεκριμένο πρόβλημα με μια αρχιτεκτονική δυο επιπέδων, και συγκεκριμένα αποτελείτε από μια ακολουθία προς δομή πρόβλεψη και μια δομή προς δομή πρόβλεψη. Το πρώτο επίπεδο δημιουργήθηκε με βάση μια υβριδική δομή που συνδυάζει γενετικό και νευρωνικό κομμάτι, ενώ το δεύτερο επίπεδο αποτελείται από ένα MLP που δέχεται σαν είσοδο την έξοδο του πρώτου επιπέδου. Τα αποτελέσματα τις προσέγγισης αυτής είχαν αρκετά ψηλά ποσοστά επιτυχίας.

10. Two-Stage method (Yuksektepe et al 2008): Στόχος της συγκεκριμένης μεθοδολογίας ήταν να προβλέψει την δευτεροταγή δομή των πρωτεϊνών σε δυο στάδια. Το πρώτο στάδιο εντόπιζε αστάθειες στο πως αναδιπλώνεται η πρωτεΐνη στο χώρο και προσπαθεί να κατατάξει σε κατηγορίες τα διάφορα σημεία της πρωτεΐνης. Στο δεύτερο στάδιο οι πρωτεΐνες διασπόνται σε ακολουθίες τριών μέχρι επτά κατάλοιπων και ο αλγόριθμος προσπαθεί να εντοπίσει την δευτεροταγή δομή των συγκεκριμένων ακολουθιών.

11. Evolutionary method for learning HMM structure (Won et al 2007): Στη συγκεκριμένη έρευνα, λόγω του ότι είναι περίπλοκη η δημιουργία ενός Hidden Markov Model (HMM), χρησιμοποιήθηκαν γενετική αλγόριθμοι για να αλλάζουν δυναμικά τις παραμέτρους ενός HMM και στην ουσία να το κτίζουν δυναμικά με σκοπό να μπορεί να υπολογίζει την δευτεροταγή δομή πρωτεϊνικών ακολουθιών που δέχεται στην είσοδο.

12. Cascade Bidirectional Recurrent Neural Network (BRNN) (Chen and Chaudhari 2007): Μια από τις πιο πρόσφατες υλοποιήσεις νευρωνικών δικτύων η οποία χρησιμοποιήθηκε για την επίλυση του προβλήματος PSSP αφορά τη δημιουργία ενός δυο επιπέδων νευρωνικού δικτύου. Η υλοποίηση αυτή είχε σκοπό να συμπεριλάβει στα δεδομένα εισόδου την έννοια των μακρινών εξαρτήσεων (long range dependencies) μεταξύ των δεδομένων, κάτι το οποίο πρέπει να ληφθεί υπόψη αφού αυτό παίζει σημαντικό ρόλο στην αναδίπλωση της πρωτεΐνης, και της συσχέτισης που υπάρχει μεταξύ γειτονικών δομών δευτεροταγούς δομής. Συγκεκριμένα, για το τελευταίο, οι συγγραφείς του άρθρου αναφέρονται στη σχέση που έχει η δευτεροταγής δομή ενός αμινοξέως με βάση τη δευτεροταγή δομή των γειτονικών του αμινοξέων. Η μέθοδος αυτή προτείνει την δημιουργία δυο BRNN με την έξοδο του πρώτου να αποτελεί την είσοδο του δεύτερου. Η μέθοδος αυτή είχε αρκετά καλά αποτελέσματα, αλλά όχι καλύτερα από προηγούμενες προσεγγίσεις.

Μέθοδος	***Q3* (%)**
Feedforward Fully Connected NN (Qian και Sejnowski, 1988)	63.300
PHD (Rost, 2001; Rost και Sander, 1993)	71.400
DSC (King και Sternberg, 1996)	71.950
NNSSP (Salamov και Soloveyev, 1997)	68.413
PREDATOR (Frishman και Argos, 1997)	68.602
Consensus (Cuff και Barton, 1999)	72.707
BRNN – Backpropagation (Baldi,et al., 1999)	76.000
LAD (Jacek et al., 2005)	70.600
MASSP3 (Giuliano A. et al., 2005)	76.100
Two-Stage method (Fadime U. Et al., 2007)	74.100
Cascade BRNN (Jinmiao C. και Narendra S.C., 2007)	74.380

Πίνακας1.1: *Οι μέθοδοι που χρησιμοποιήθηκαν για πρόβλεψη δευτεροταγούς δομής πρωτεϊνών και τα ποσοστά επιτυχίας τους*

Οι υλοποιήσεις που εφαρμόστηκαν για λύση του προβλήματος πρόβλεψης δευτεροταγούς δομής πρωτεϊνών ανήκουν σε διάφορες κατηγορίες και χρησιμοποιούν ποικίλες ιδέες χωρίς όμως να πετυχαίνουν πολύ ψηλά ποσοστά σωστής πρόβλεψης. Κάποιες ιδέες θεωρούνται πολύ καλύτερες από κάποιες άλλες αφού προσαρμόζονται στη φύση του προβλήματος, χωρίς όμως κάποια από αυτές να έχει ποσοστό επιτυχίας μεγαλύτερο του 76%. Η πρώτη σημαντική μέθοδος που εφαρμόστηκε από τους Qian και Sejnowski (Qian and Sejnowski 1988) δεν μπορούσε να έχει καλύτερα αποτελέσματα αφού δεν ανταποκρίνεται στη φύση του προβλήματος. Συγκεκριμένα το δίκτυο πρέπει να έχει τους μηχανισμούς να συσχετίζει την συγκεκριμένη σειρά της ακολουθίας των αμινοξέων μεταξύ τους, κάτι το οποίο δεν μπορεί να κάνει ένα απλό νευρωνικό δίκτυο εμπρόσθιου περάσματος. Επίσης, το δίκτυο αυτό με το κινούμενο παράθυρο μπορεί να χρησιμοποιεί τα αμινοξέα που προηγούνται και έπονται του αμινοξέως που εξετάζεται η δευτεροταγής δομή του, όμως η έλλειψη ανάδρασης στο δίκτυο δεν επιτρέπει την συσχέτιση των αμινοξέων αυτών (Elman 1990). Η δεύτερη μεγάλη έρευνα των Rost και Sander (Rost and Sander 1993) ούτε αυτή μπορούσε να έχει καλύτερη τύχη αφού ήταν μια βελτιστοποίηση της προηγούμενης. Στην έρευνα αυτή, σημαντική είναι η ιδέα χρησιμοποίησης αλγορίθμων πολλαπλής στοίχισης (multiple alignment), για ομαδοποίηση των πρωτεϊνικών ακολουθιών αφού έτσι μπορεί η ιδιότητα αυτή να χρησιμοποιηθεί στην δημιουργία των συνόλων δεδομένων, για κάλυψη όλου του εύρους πρωτεϊνικών ακολουθιών.

Οι μέθοδοι των Salamov και Soloveyev (Salamov and Soloveyev 1997) και των King και Sternberg (King and Sternberg 1996) ήταν κυρίως στατιστικές μέθοδοι και στηρίζονταν στις ομοιότητες με πρωτεΐνες που γνώριζαν ήδη την δευτεροταγή δομή τους. Ο λόγος στον οποίο μπορεί να οφείλεται το σχετικά μικρό ποσοστό επιτυχίας τους είναι το γεγονός ότι οι πρωτεϊνικές ακολουθίες των οποίων ήταν γνωστή η δευτεροταγής δομή ήταν μόνο μερικές χιλιάδες, πολύ λιγότερες από τις 53000 πρωτεϊνικές ακολουθίες που είναι σήμερα. Από αυτό το γεγονός μπορούμε να συμπεράνουμε ότι οι πρωτεΐνες αυτές ίσως δεν περιείχαν στη δομή τους όλα τα δεδομένα για την αναγνώριση άγνωστων μοτίβων. Για τον ίδιο λόγο, πολύ πιθανόν να απέτυχε και η μέθοδος

Το νευρωνικό δίκτυο των Frishman και Argos (Frishman and Argos 1995) βασιζόταν στην δομή του πρωτεϊνικού σκελετού και στους δεσμούς υδρογόνου, οι οποίοι παίζουν

σημαντικό ρόλο μόνο στην κατηγορία δευτεροταγούς δομής των α-ελίκων. Έτσι η μέθοδος αυτή στηριζόμενη σε ελλιπής στοιχεία, αν και θεωρείται από τις θεμελιώδεις μεθόδους πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, δεν είχε αποτελέσματα πρόβλεψης μεγαλύτερα του 70%.

Οι πρώτες μεθόδοι με αποτελέσματα πρόβλεψης μεγαλύτερα του 72% εμφανίστηκαν το 1999. Αρχικά το δίκτυο των Cuff και Barton (Cuff and Barton 1999) χρησιμοποιούσε για προεπεξεργασία των δεδομένων αλγόριθμο πολλαπλής στοίχισης (multiple alignment) για συγκεκριμένα χαρακτηριστικά των πρωτεϊνών. Τα χαρακτηριστικά αυτά όντως βοηθούσαν στην πρόβλεψη της δευτεροταγούς δομή αλλά η μέθοδος αυτή δεν βασιζόταν μόνο στην ακολουθία των αμινοξέων, στην οποία βρίσκεται η ουσία στο πως η πρωτεΐνη αναδιπλώνεται στο χώρο. Η μέθοδος αυτή χρησιμοποιούσε εξωγενή στοιχεία των πρωτεϊνών που ίσως ξέφευγαν από την ιδέα της φύσης του προβλήματος, δηλαδή από το γεγονός ότι η δευτεροταγής δομή οφείλεται καθαρά στην ακολουθία των αμινοξέων. Το γεγονός αυτό αξιοποιείται πλήρως από την υλοποίηση του ΤΝΔ αμφίδρομη ανάδραση του Baldi (Baldi et al 1999,Baldi et al 2000). Το δίκτυο αυτό προσπαθούσε να υπολογίσει την δευτεροταγή δομή μιας πρωτεϊνικής ακολουθίας με βάση τα αμινοξέα που προηγούνται και έπονται του κάθε αμινοξέως, μια θεωρία η οποία ανταποκρίνεται πλήρως στη φύση του προβλήματος. Αυτός ήταν και ο λόγος που η υλοποίηση αυτή είχε το μεγαλύτερο ποσοστό επιτυχίας από όλες τις μεθόδους που χρησιμοποιήθηκαν για το συγκεκριμένο πρόβλημα.

Σημαντικές γύρω από το πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών είναι η σύγκριση των Armano (Armano et al 2006) και Chen (Chen and Chaudhari 2007) οι οποίοι υποστηρίζουν ότι η πρόβλεψη δευτεροταγούς δομής πρωτεϊνών αποτελείται από δυο στάδια. Το πρώτο στάδιο προβλέπει την δευτεροταγή δομή με βάση την ακολουθία των αμινοξέων, κάτι το οποία ανταποκρίνεται στην φύση του προβλήματος, και ακολούθως το δεύτερο στάδιο χρησιμοποιεί τα δεδομένα του πρώτου σταδίου για να υπολογίσει την τελική δευτεροταγή δομή με βάση την αρχική πρόβλεψη. Η άποψη αυτή υποστηρίζει ότι η δευτεροταγής δομή ενός αμινοξέως εξαρτάται από τη δευτεροταγή δομή των γειτονικών του αμινοξέων, κάτι το οποίο φαίνεται να ισχύει σε μεγάλο βαθμό παρατηρώντας δευτεροταγείς δομές πρωτεϊνών (RCSB Protein Data Bank, 20 April 2009).

Κεφάλαιο 2

Πρωτεϊνική δομή και λειτουργία

2.1 Ο βιολογικός ρόλος των πρωτεϊνών

2.2 Η δομή των πρωτεϊνών

2.2.1 Τα αμινοξέα

2.2.2 Ο ρόλος των αμινοξέων στην δομή και λειτουργία των πρωτεϊνών

2.3 Επίπεδα οργάνωσης πρωτεϊνών

2.3.1 Πρωτοταγής δομή

2.3.2 Δευτεροταγής δομή

2.1.2.1 Ελικοειδείς δευτεροταγείς δομές - α-έλικες

2.1.2.2 Εκτεταμένες δομές – β-κλώνοι και β-πτυχωτές επιφάνειες

2.3.3 Τριτοταγής και τεταρτοταγής δομή

2.4 Η ανάγκη και τα προβλήματα σχετικά με την μελέτη των πρωτεϊνών

2.5 Βασικοί ορισμοί Βιολογίας – Χημείας και η σχέση τους με τα αμινοξέα

2.5.1 Ομοιοπολικός δεσμός

2.5.2 Μη-ομοιοπολικός δεσμός

2.1 Ο βιολογικός ρόλος των πρωτεϊνών

Τα ζωντανά κύτταρα για να καλύψουν όλες τις ενεργειακές, δομικές και λειτουργικές τους ανάγκες χρησιμοποιούν μεγάλα πολυμερή μόρια, τα λεγόμενα βιολογικά μακρομόρια. Αυτά είναι τα νουκλεϊκά οξέα, οι πολυσακχαρίτες και οι πρωτεΐνες. Τα νουκλεϊκά οξέα, δηλαδή το Deoxyribonucleic acid (DNA) και το Ribonucleic acid (RNA), διατηρούν τις πληροφορίες του κυττάρου τις οποίες και μεταβιβάζουν από γενιά σε γενιά. Οι πολυσακχαρίτες παρέχουν στο ζωντανό κύτταρο όλη την απαιτούμενη ενέργεια που χρειάζεται για να επιβιώνει και να αναπαράγεται. Οι υπόλοιπες εργασίες που εκτελεί το κύτταρο για την επιβίωσή του αναλαμβάνονται από τις πρωτεΐνες.

Οι πρωτεΐνες είναι πολύπλοκες αζωτούχες ενώσεις που υπάρχουν σε όλους τους ζωντανούς οργανισμούς, ζώα και φυτά. Έχουν μεγάλη θρεπτική αξία και λαμβάνουν μέρος άμεσα στις περισσότερες χημικές διαδικασίες που είναι θεμελιώδεις για την ζωή. Η σημασία των πρωτεϊνών αναγνωρίστηκε από τους χημικούς στις αρχές του 19ου αιώνα, οι οποίοι και τους έδωσαν αυτή την ονομασία από την ελληνική λέξη «πρώτειος» ή «πρωτεΐος» που σημαίνει ο «ο κατέχων την πρώτη θέση». Το κύτταρο, ανάλογα με τις ανάγκες που έχει τη δεδομένη στιγμή, κατασκευάζει τις ανάλογες πρωτεΐνες χρησιμοποιώντας απλά μόρια, τα αμινοξέα. Τα φυτικά κύτταρα παράγουν όλων των ειδών τα αμινοξέα που χρειάζονται στη χημική σύνθεση των πρωτεϊνών σε αντίθεση με τα ζωικά κύτταρα που παράγουν μόνο κάποια από τα αμινοξέα. Γι' αυτό το λόγο τα ζώα παίρνουν όλα τα αμινοξέα που υπολείπονται μέσα από φυτικές ή ζωικές τροφές.

Οι πρωτεΐνες, όπως και τα νουκλεϊκά οξέα, είναι γραμμικά πολυμερή και η αλληλουχία των πρωτεϊνών βρίσκεται αποθηκευμένη στο γενετικό υλικό. Ανάλογα με το είδος του κυττάρου εκφράζονται σε αυτό οι κατάλληλες πρωτεΐνες. Σύγχρονες έρευνες αναφέρουν ότι ο αριθμός των διαφορετικών γονιδίων που υπάρχουν στον ανθρώπινο οργανισμό και είναι υπεύθυνα για την κατασκευή πρωτεϊνών είναι περίπου 30000 (RCSB Protein Data Bank, 20 April 2009). Οι πρωτεΐνες χρειάζονται για τη διάσπαση των πολυσακχαριτών ώστε το κύτταρο να παράξει την απαιτούμενη ενέργεια για τη διαβίωσή του, για να διπλασιάσει το γενετικό του υλικό, για την προστασία του κυττάρου ως δομικό υλικό, για

την επικοινωνία με άλλα κύτταρα. Ο συνοπτικός Πίνακας 2.1 παρουσιάζει τις σημαντικότερες λειτουργίες των πρωτεϊνών που αντικατοπτρίζουν τον βιολογικό τους ρόλο. Μέσα από αυτές τις διεργασίες, αλλά και από το γεγονός ότι οι πρωτεΐνες καταλαμβάνουν το 15% της ανθρώπινης μάζας, μπορεί κάποιος να αντιληφθεί την τεράστια σημασία των πρωτεϊνών στην επιβίωση του κάθε κυττάρου και κατά συνέπεια στην επιβίωση του ανθρώπινου είδους.

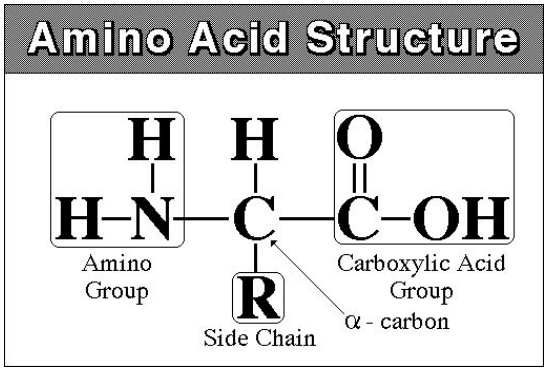
Βιολογικοί Ρόλοι Πρωτεϊνών
Ενζυμική κατάλυση
Μεταφορά και αποθήκευση
Κίνηση
Μηχανική στήριξη
Ανοσοπροστασία
Δημιουργία και μετάδοση νευρικών παλμών
Έλεγχος της ανάπτυξης και διαφοροποίησης

Πίνακας 2.1: *Βιολογικοί ρόλοι πρωτεϊνών.*

2.2 Η δομή των πρωτεϊνών

2.2.1 Τα αμινοξέα και ο ρόλος τους στη δομή των πρωτεϊνών

Η δομή όλων των γνωστών πρωτεϊνών αποτελείται από 20 διαφορετικά αμινοξέα. Συγκεκριμένα, οι πρωτεΐνες στους περισσότερους οργανισμούς συντίθενται με τον πολυμερισμό 20 διαφορετικών τύπων L-a-αμινοξέων (Πίνακας 2.2). Οποιαδήποτε αλλαγή στη σειρά, τον αριθμό και την ποσότητα των αμινοξέων οδηγεί στη δημιουργία μιας διαφορετικής πρωτεΐνης, η οποία έχει διαφορετικές ιδιότητες και χρησιμοποιείται για διαφορετικές λειτουργίες. Η αναπαράσταση μιας πρωτεΐνης μπορεί να γίνει πολύ εύκολα γράφοντας λέξεις με 20 διαφορετικά γράμματα τα οποία θα αντιστοιχούν το κάθε ένα σε ένα αμινοξύ (Πίνακας 2.2).

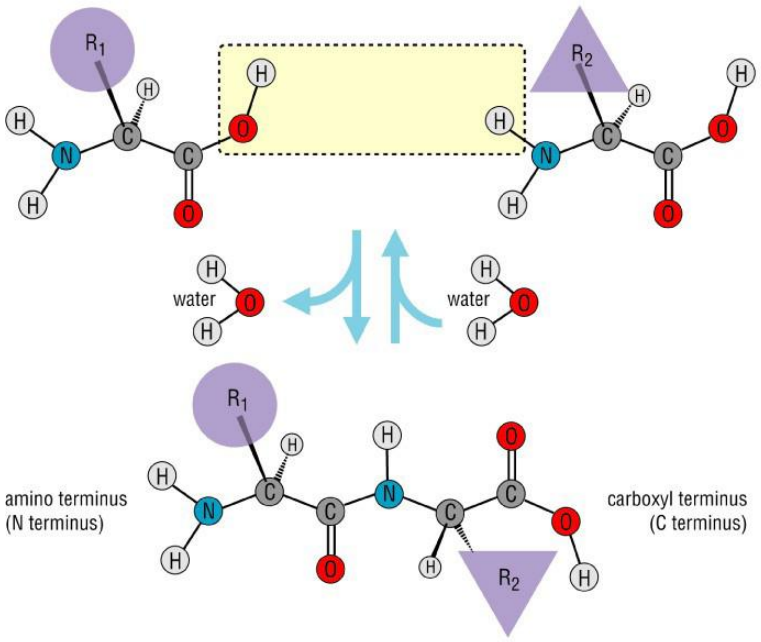


Σχήμα 2.1: *Η δομή των αμινοξέων (Promponas 2004)*

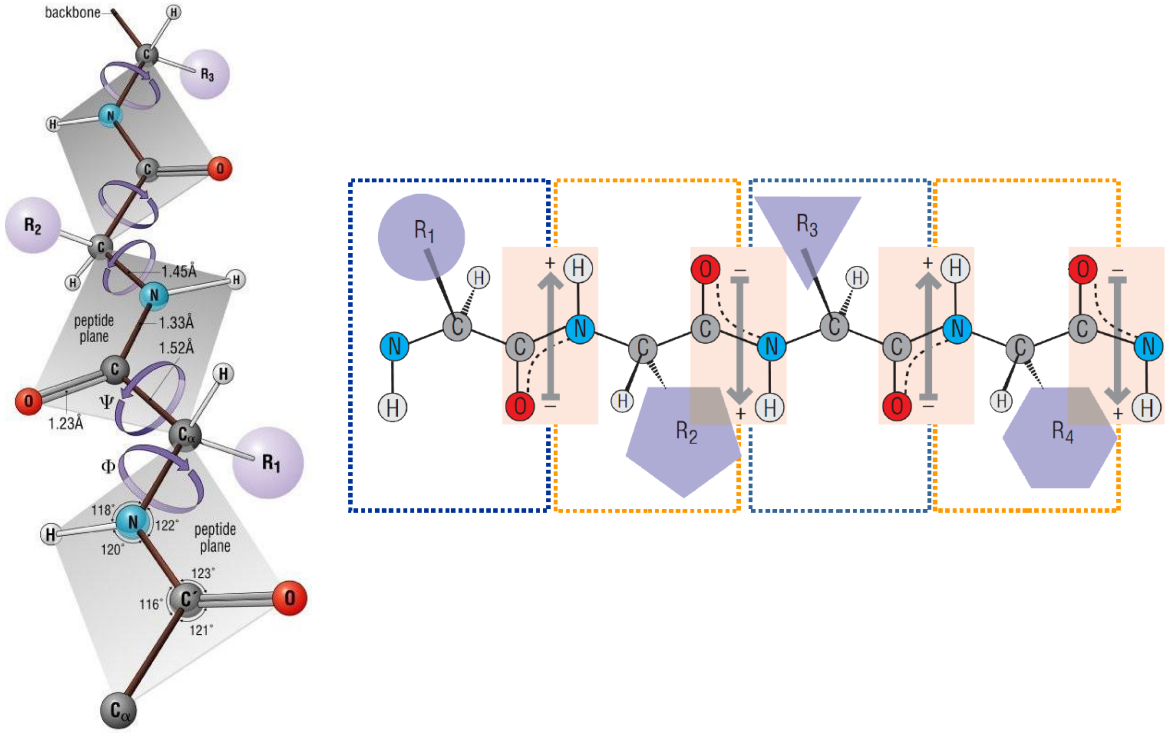
ΑΜΙΝΟΞΥ	ΣΥΜΒΟΛΙΣΜΟΣ		ΣΥΝΤΑΚΤΙΚΟΣ ΤΥΠΟΣ
Γλυκίνη	GLY	G	NH2-CH2-COOH
Αλανίνη	ALA	A	CH3-CH(NH2)-COOH
Βαλίνη	VAL	V	(CH3)2-CH-CH(NH2)-COOH
Ισολευκίνη	ILE	I	CH3-CH2-CH(CH3)-CH(NH2)-COOH
Λευκίνη	LEU	L	(CH3)2-CH-CH2-CH(NH2)-COOH
Φαινυλαλανίνη	PHE	F	Ph-CH2-CH(NH2)-COOH
Προλίνη	PRO	P	NH-(CH2)3-CH-COOH |_____|
Μεθειονίνη	MET	M	CH3-S-(CH2)2-CH(NH2)-COOH
Τρυπτοφάνη	TRP	W	Ph-NH-CH=C-CH2-CH(NH2)-COOH |_____|
Κυστεΐνη	CYS	C	HS-CH2-CH(NH2)-COOH
Σερίνη	SER	S	HO-CH2-CH(NH2)-COOH
Θρεονίνη	THR	T	CH3-CH(OH)-CH(NH2)-COOH
Ασπαραγίνη	ASN	N	H2N-CO-CH2-CH(NH2)-COOH
Γλουταμίνη	GLN	Q	H2N-CO-(CH2)2-CH(NH2)-COOH
Τυροσίνη	TYR	Y	HO-p-Ph-CH2-CH(NH2)-COOH
Ιστιδίνη	HIS	H	NH-CH=N-CH=C-CH2-CH(NH2)-COOH |_____|
Ασπαρτικό οξύ	ASP	D	HOOC-CH2-CH(NH2)-COOH
Γλουταμικό οξύ	GLU	E	HOOC-(CH2)2-CH(NH2)-COOH
Λυσίνη	LYS	K	H2N-(CH2)4-CH(NH2)-COOH
Αργινίνη	ARG	R	HN=C(NH2)-NH-(CH2)3-CH(NH2)-COOH

Πίνακας 2.2: *Τα 20 φυσικά L-α-αμινοξέα που συμμετέχουν στο σχηματισμό των πρωτεϊνών*

Μελετώντας την δομή όλων των αμινοξέων (Σχήμα 2.1) παρατηρούμε ότι αποτελούνται από ένα κεντρικό άτομο άνθρακα το οποίο συνδέεται με ένα υδρογόνο, μια αμινομάδα, μια καρβοξυλομάδα και μια πλευρική αλυσίδα. Η πλευρική αυτή αλυσίδα διαφοροποιεί τις φυσικές και τις χημικές ιδιότητες των 20 διαφορετικών αμινοξέων ενώ ανάλογα με το φορτίο της τα αμινοξέα χωρίζονται σε τέσσερεις κατηγορίες. Η πρώτη κατηγορία είναι τα όξινα αμινοξέα των οποίων η πλευρική αλυσίδα είναι φορτισμένη αρνητικά, η δεύτερη κατηγορία είναι τα βασικά αμινοξέα των οποίων η πλευρική αλυσίδα είναι φορτισμένη θετικά, ακολούθως έχουμε την τρίτη κατηγορία την οποία αποτελούν αμινοξέα χωρίς φορτίο των οποίων όμως η πλευρική αλυσίδα έχει δυο περιοχές αντίθετα φορτισμένες και τέλος στην τέταρτη κατηγορία ανήκουν τα αμινοξέα τα οποία δεν έχουν καθόλου φορτίο. Όλα τα αμινοξέα έχουν φορτισμένη θετικά την αμινομάδα τους και φορτισμένη αρνητικά την καρβοξυλομάδα τους με αποτέλεσμα να συνδέονται εύκολα μεταξύ τους σε σειρές, με ομοιοπολικούς δεσμούς, απελευθερώνοντας ένα μόριο νερού. Οι συνδέσεις αυτές των αμινοξέων σειρά οδηγούν στην δημιουργία των διαφορετικών πρωτεϊνών. Ο δεσμός αυτός που συνδέει δυο αμινοξέα μεταξύ τους ονομάζεται πεπτιδικός (Σχήμα 2.2). Τα αμινοξέα, μετά την συνένωση τους, αναφέρονται ως αμινοξικά κατάλοιπα. Ο πεπτιδικός δεσμός είναι επίπεδος και σχεδόν άκαμπτος περιορίζοντας την περιστροφική ελευθερία μιας πολυπεπτιδικής αλυσίδας (Σχήμα 2.3). Παρατηρώντας το συγκεκριμένο σχήμα μπορούμε να παρατηρήσουμε την επίπεδη διάταξη των πεπτιδικών δεσμών καθώς και την ελευθερία περιστροφής των επιπέδων αυτών γύρω από το μόριο άνθρακα. Η περιστροφές των επιπέδων, δεξιά και αριστερά από το μόριο άνθρακα, περιγράφονται με δυο δίεδρες γωνίες, φ και ψ. Αυτές οι ιδιότητες των πεπτιδικών δεσμών έχουν ως αποτέλεσμα οι συγκεκριμένοι να είναι αρκετά σταθεροί και έτσι τα αμινοξικά κατάλοιπα να μην μπορούν να περιστραφούν γύρω από τον εαυτό τους, ούτε να έχουν μεγάλο εύρος κινήσεων.



Σχήμα 2.2: *Ο πεπτιδικός δεσμός (Promponas 2004)*



Σχήμα 2.3: *Τα επίπεδα και οι γωνίες που σχηματίζουν οι πεπτιδικοί δεσμοί(Promponas 2004)*

Στην πραγματικότητα η πρωτεΐνη δεν είναι μια μεγάλη ευθεία σειρά από αμινοξέα. Οι πρωτεΐνες είναι τρισδιάστατα μακρομόρια η δομή των οποίων καθορίζεται σε συγκεκριμένες συνθήκες από την αμινοξική τους αλληλουχία. Αυτό μπορεί να αποδειχθεί μέσα από ένα συγκεκριμένο πείραμα. Στην ουσία πρέπει να αποδειχθεί ότι η τρισδιάστατη δομή των πρωτεϊνών δεν εξαρτάται από άλλους παράγοντες εκτός από τη δομή της ίδιας της πρωτεΐνης, δηλαδή τη σειρά και τον αριθμό των πρωτεϊνών. Το πείραμα που γίνεται σε αυτές τις περιπτώσεις είναι η τοποθέτηση της πρωτεΐνης σε μη φυσιολογικό περιβάλλον με αποτέλεσμα να μετουσιωθεί και να χάσει το σχήμα της. Στη συνέχεια αν επανατοποθετήσουμε τη πρωτεΐνη σε φυσιολογικές συνθήκες τότε αυτή παίρνει ακριβώς το ίδιο σχήμα που είχε στην αρχή. Μέσα από αυτό το πείραμα καταλήγουμε στο συμπέρασμα ότι η τρισδιάστατη μορφή της πρωτεΐνης, αφού αυτή βρίσκεται σε φυσιολογικές συνθήκες κυττάρου, εξαρτάται αποκλειστικά από τη αλληλουχία των αμινοξέων της. Η αλληλουχία αυτή και κατά συνέπεια το τελικό σχήμα της πρωτείνης καθορίζει και τη λειτουργία που εκτελεί κάθε πρωτεΐνη στο κύτταρο.

Το τελικό σχήμα της πρωτεΐνης οφείλεται στις φυσικές και χημικές ιδιότητες των αμινοξέων και των μεταξύ τους αλληλεπιδράσεων. Η σημαντικότερη αλληλεπίδραση μεταξύ των αμινοξέων είναι ο πεπτιδικός δεσμός, ο οποίος συνδέει δυο αμινοξέα μεταξύ τους. Όπως προαναφέραμε οι πεπτιδικοί δεσμοί είναι αρκετά σταθεροί και έτσι τα αμινοξικά κατάλοιπα δεν μπορούν να περιστραφούν γύρω από τον εαυτό τους, ούτε να έχουν μεγάλο εύρος κινήσεων. Άρα οι πεπτιδικοί δεσμοί αφού περιορίζουν τις πιθανές κινήσεις των αμινοξέων έχουν σημαντικό ρόλο στην τελική διαμόρφωση του σχήματος της πρωτεΐνης. Αφού σχηματιστεί η πρωτεΐνη τα μόνα ελεύθερα φορτία που έχουμε σε αυτή είναι τα φορτία της αμινομάδας και της καρβοξυλομάδας που βρίσκονται στα δυο άκρα της πρωτεΐνης καθώς και τα φορτία των πλευρικών αλυσίδων. Τα φορτία αυτά των πλευρικών αλυσίδων μπορούν να δημιουργήσουν τους μη-ομοιοπολικούς δεσμούς (Υποκεφάλαιο 2.5.2) οι οποίοι επίσης έχουν σημαντικό ρόλο στη διαμόρφωση του τελικού σχήματος της πρωτεΐνης. Οι μη-ομοιοπολικοί δεσμοί είναι έλξεις ή απωθήσεις μεταξύ των πλευρικών αλυσίδων των αμινοξέων. Οι σημαντικότεροι μη-ομοιοπολικοί δεσμοί είναι οι υδρόφοβες αλληλεπιδράσεις, ιοντικοί δεσμοί, δεσμοί υδρογόνου και έλξεις Van der Waals

(Υποκεφάλαιο 2.5.2). Σπανιότερα, μεταξύ αμινοξικών καταλοίπων είναι δυνατό να σχηματιστούν ομοιοπολικοί δεσμοί (Υποκεφάλαιο 2.5.1), όπως οι σιδουλφιδρυλικοί δεσμοί. Οι δεσμοί αυτοί σχηματίζονται μόνο μεταξύ των ατόμων θείου δυο κυστεϊνών και για το λόγο αυτό ονομάζονται και γέφυρες θείου. Εξαιτίας του ομοιοπολικού τους χαρακτήρα είναι πολύ ισχυροί και μπορούν να επηρεάσουν καθοριστικά το τελικό σχήμα μιας πρωτεΐνης. Όλες αυτές οι αλληλεπιδράσεις, με σημαντικότερο παράγοντα το νερό (Υποκεφάλαιο 2.5.2), οδηγούν στην δημιουργία του τελικού σχήματος της πρωτεΐνης.

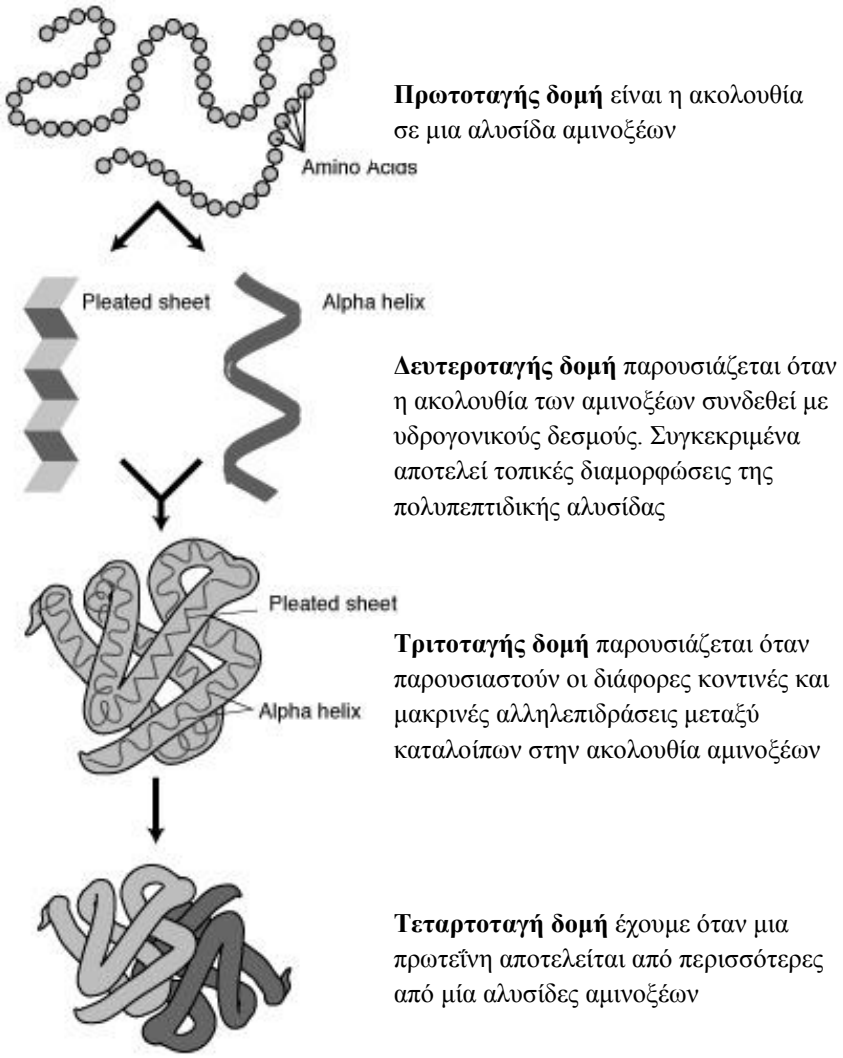
2.2.2 Τα αμινοξέα και ο ρόλος τους στη λειτουργία των πρωτεϊνών

Η αρχή η οποία διέπει την λειτουργικότητα της πρωτεΐνης είναι ότι η τρισδιάστατη δομή που αποκτά στο χώρο η πρωτεΐνη είναι ο καθοριστικός παράγοντας για τη λειτουργία της. Οι διαφορετικοί τρόποι με τους οποίους διευθετούνται οι πολυπεπτιδικές αλυσίδες τοποθετούν τις απαραίτητες χημικές ομάδες σε κατάλληλες θέσεις στον τρισδιάστατο χώρο. Αυτό έχει ως αποτέλεσμα η πρωτεΐνη να λαμβάνει το απαιτούμενο για τη λειτουργία της σχήμα, κατανέμοντας κατάλληλα στο χώρο τις φυσικοχημικές ιδιότητες των αμινοξικών καταλοίπων που τη συνθέτουν. Σημαντικό εδώ είναι το γεγονός ότι αν τοποθετήσουμε αμινοξέα με μια συγκεκριμένη σειρά τότε οι αλληλεπιδράσεις που δημιουργούνται δημιουργούν πάντοτε το ίδιο σχήμα πρωτεΐνης, δηλαδή δημιουργούν μια συγκεκριμένη πρωτεΐνη η οποία έχει συγκεκριμένη λειτουργικότητα.

2.3 Επίπεδα οργάνωση πρωτεϊνών

Για να διευκολυνθεί η μελέτη των πρωτεϊνών, έχει υιοθετηθεί μια ιεραρχία για την οργάνωση της δομής των πρωτεϊνών, παρόλο που στη φύση συναντώνται μόνο στην τελική τρισδιάστατη μορφή τους. Μέσα από αυτή την οργάνωση είναι γίνεται πιο εύκολη η κατανόηση και η παρατήρηση της δομής και της λειτουργίας τους. Τα επίπεδα οργάνωσης

των πρωτεϊνών είναι η πρωτοταγής δομή, η δευτεροταγής δομή, η τριτοταγής δομή και η τεταρτοταγής δομή (Σχήμα 2.4).



Πρωτοταγής δομή είναι η ακολουθία σε μια αλυσίδα αμινοξέων

Δευτεροταγής δομή παρουσιάζεται όταν η ακολουθία των αμινοξέων συνδεθεί με υδρογονικούς δεσμούς. Συγκεκριμένα αποτελεί τοπικές διαμορφώσεις της πολυπεπτιδικής αλυσίδας

Τριτοταγής δομή παρουσιάζεται όταν παρουσιαστούν οι διάφορες κοντινές και μακρινές αλληλεπιδράσεις μεταξύ καταλοίπων στην ακολουθία αμινοξέων

Τεταρτοταγή δομή έχουμε όταν μια πρωτεΐνη αποτελείται από περισσότερες από μία αλυσίδες αμινοξέων

Σχήμα 2.4: *Τα επίπεδα οργάνωσης των πρωτεϊνών(Madison 20April 2009)*

2.3.1 Πρωτοταγής δομή

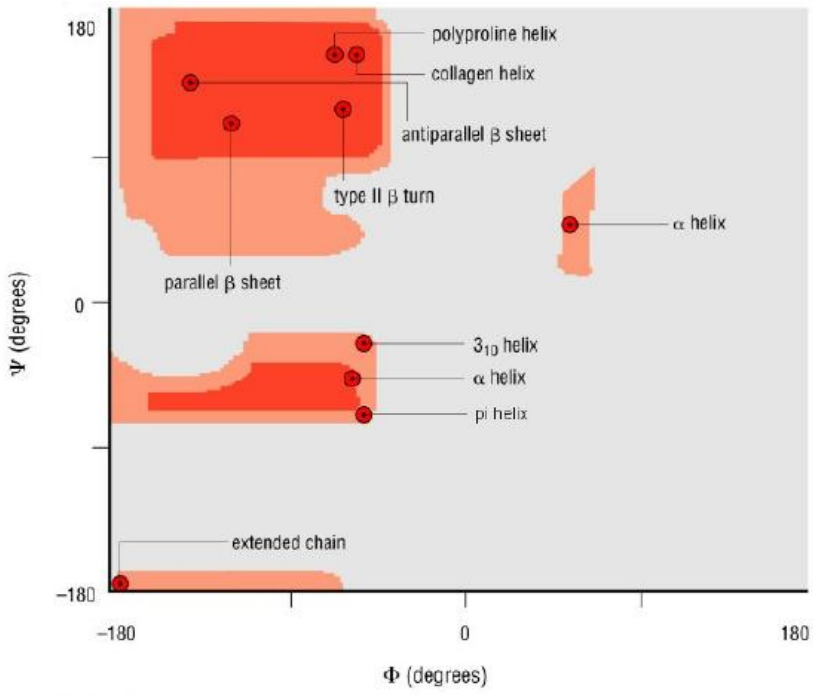
Η πρωτοταγής δομή των πρωτεϊνών αφορά την διακριτή αλληλουχία των αμινοξέων, δηλαδή σε ποια σειρά και ποσότητα βρίσκονται τα αμινοξέα σε μια συγκεκριμένη πρωτεΐνη. Κάθε διακριτή αλληλουχία αμινοξικών καταλοίπων αντιπροσωπεύει διαφορετική πρωτεΐνη και συμβολίζεται με λέξεις οι οποίες δημιουργούνται από το σύνολο γραμμάτων του Πίνακα 2.2. Το μεγαλύτερο ποσοστό δεδομένων που υπάρχει σήμερα στις βάσεις δεδομένων, σχετικά με πρωτεΐνες, αφορά την πρωτοταγή δομή τους, αφού αυτή μπορεί εύκολα να εξαχθεί με την μετάφραση του γενετικού υλικού. Φυσικά, η πρωτοταγής δομή της πρωτεΐνης δεν μας δίνει αρκετά στοιχεία για τη μορφή και τη λειτουργία της, αφού όπως προαναφέρθηκε από αυτή μπορούμε να δούμε μόνο την σειρά των αμινοξικών καταλοίπων που την αποτελούν.

Σχήμα 2.5: *Η πρωτοταγής δομή μιας πρωτεΐνης (Promponas 2004)*

2.3.2 Δευτεροταγής δομή

Η δευτεροταγής δομή των πρωτεϊνών αφορά τα γνωστά μοτίβα, στερεοδιάταξη των καταλοίπων με κανονικές κανονικότητες, που συναντούμε γενικά στις πρωτεΐνες. Οι σημαντικότερες κατηγορίες στοιχείων δευτεροταγούς δομής που συναντούμε σε μια πρωτεΐνη είναι οι α-έλικες και οι β-κλώνοι. Ο δευτεροταγείς αυτές δομές χαρακτηρίζονται

από το συγκεκριμένο εύρος που μπορούν να έχουν οι δίεδρες γωνίες Φ και Ψ (Υποκεφάλαιο 2.2.1). Αυτό σημαίνει ότι συγκεκριμένοι συνδυασμοί των δίεδρων γωνιών Φ και Ψ, που χαρακτηρίζουν τα επίπεδα των δεσμών της «ραχοκοκαλιάς» της δομής γύρω από το κεντρικό μόριο άνθρακα του κάθε αμινοξικού κατάλοιπου, μπορούν να δημιουργήσουν α-έλικες, β-κλώνους ή άλλες μορφές δευτεροταγούς δομής. Οι συγκεκριμένες δίεδρες γωνίες χαρακτηρίζονται από κάποια σταθερότητα, η οποία κατ' επέκταση εξασφαλίζει την σταθερότητα των διάφορων δευτεροταγών μορφών των πρωτεϊνών, λόγω των υδρογονικών δεσμών που σχηματίζονται μεταξύ των αμινοξικών κατάλοιπων. Περισσότερες λεπτομέρειες σχετικά με τις τιμές των γωνιών Χ και Ψ και την συσχέτισή τους με τα μοτίβα δευτεροταγούς δομής μπορούμε να δούμε μέσα από ένα διάγραμμα Ramachandran (Σχήμα 2.6)(Petsko and Ringe 2004).

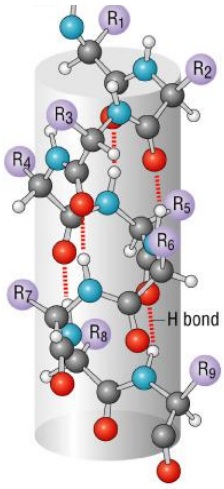


Σχήμα 2.6: *Διάγραμμα Ramachandran επιτρεπτών στερεοδιατάξεων δευτεροταγούς δομής(Promponas 2004)*

Μια ευρέως διαδεδομένη μορφή ορισμού των στοιχείων δευτεροταγούς μορφής είναι η δευτεροταγής δομή που ορίζει το πρόγραμμα DSSP. Σύμφωνα με το DSSP οι κατηγορίες είναι : α–helix (H) , 3–helix (G) , π-helix (I) , β–strand (E) , β–bridge (B) , β-turn (T), bend (S) και τα υπόλοιπα (L). Συνήθως, όμως, οι κατηγορίες που χρησιμοποιούνται στην πράξη είναι τρείς. Η πρώτη κατηγορία είναι οι έλικες και σε αυτή ανήκουν οι ευρύτερες ομάδες H, G και I, η δεύτερη κατηγορία είναι οι εκτεταμένες δομές(κλώνοι) στην οποία ανήκουν οι ομάδες E και B και τέλος στην τρίτη κατηγορία ανήκουν τα υπόλοιπα μοτίβα, τα οποία συχνά χαρακτηρίζονται ως coil (C).

2.1.2.1 Ελικοειδείς δευτεροταγείς δομές - α-έλικες

Οι ελικοειδείς δευτεροταγείς δομές δημιουργούνται όταν κοντινά στην αλληλουχία αμινοξικά κατάλοιπα δημιουργούν υδρογονικούς δεσμούς (Υποκεφάλαιο 2.5.2). Συγκεκριμένα, ο υδρογονικός δεσμός σχηματίζεται μεταξύ αμινομάδας, η οποία δίνει το υδρογόνο, και καρβοξυλομάδας, η οποία δέχεται το υδρογόνο, διαφορετικών αμινοξικών κατάλοιπων (Σχήμα 2.7). Όταν οι δεσμοί αυτοί εμφανίζονται σε κάποιες συγκεκριμένες περιοδικές αποστάσεις οδηγεί στο σχηματισμό ελικοειδών δομών. Οι ελικοειδείς αυτές δομές έχουν διαφορετικά γεωμετρικά και ενεργειακά χαρακτηριστικά.



Σχήμα 2.7: *Ελικοειδής δομή στην οποία με κόκκινο χρώμα παρουσιάζεται ο υδρογονικός δεσμός (Promponas 2004)*

Στη δομή πρωτεϊνών υπάρχουν διαφορετικά είδη ελίκων. Οι έλικες αυτές συμβολίζονται με δυο αριθμούς εκ των οποίων ο πρώτος συμβολίζει τον αριθμό των καταλοίπων που υπάρχουν μεταξύ του δεσμού υδρογόνου και ο δεύτερος συμβολίζει τον αριθμό των ατόμων που σχηματίζουν κλειστό δακτύλιο. Η σημαντικότερη από τις δυνατές ελικοειδείς δομές είναι η δεξιόστροφη α-έλικα και συγκεκριμένα η 4_{13} έλικα. Αν κοιτάξουμε πιο προσεκτικά την έλικα αυτή παρατηρούμε ότι έχει κάποια σταθερά χαρακτηριστικά όπως κάθε βήμα της έλικας 5.4Å, με 3.6 κατάλοιπα ανά στροφή και μέσες γωνίες Φ=-57° και Ψ=-47° (Σχήμα 2.6). Οι μέσες δίεδρες γωνίες για την κάθε διαφορετική μορφή έλικα παρουσιάζονται στον Πίνακας 2.3.

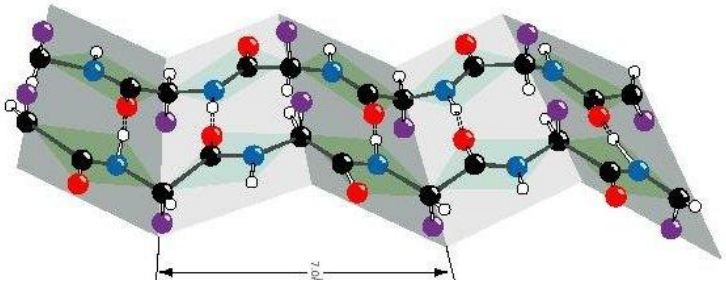
	Φ (°)	***Ψ (°)***
α-έλικα (3.6_{13})	-57	-47
3_{10} έλικα	-49	-26
π-έλικα (5_{16})	-57	-70
έλικα πολυπρολίνης (I)	-83	+158
έλικα πολυπρολίνης (II)	-78	+149
έλικα πολυπρολίνης (III)	-80	+150

Πίνακας 2.3: *Τα γεωμετρικά χαρακτηριστικά ελικοειδών μορφών, όπου Φ και Ψ δίεδρες γωνίες (Υποκεφάλαιο 2.3.2)*

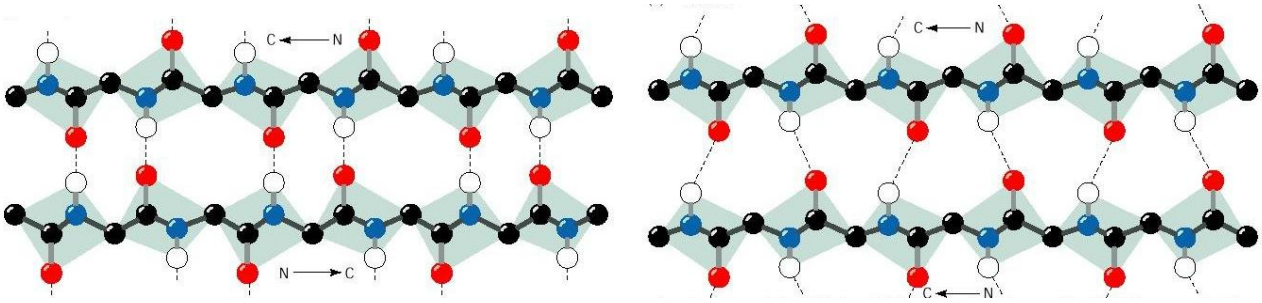
2.1.2.2 Εκτεταμένες δομές – β-κλώνοι και β-πτυχωτές επιφάνειες

Οι β-κλώνοι είναι μια μορφή δευτεροταγούς δομής η οποία παρουσιάζεται πολύ συχνά στις πρωτεΐνες. Οι β-κλώνοι δημιουργούνται όταν η πολυπεπτιδική αλυσίδα βρίσκεται σε εκτεταμένη μορφή με τις αμινομάδες και καρβοξυλομάδες των αμινοξικών κατάλοιπων να διατάσσονται κάθετα προς αυτή. Αυτές οι αμινομάδες και καρβοξυλομάδες μπορούν να δημιουργήσουν δεσμούς υδρογόνου(Υποκεφάλαιο 2.5.2) με άλλα τμήματα πολυπεπτιδικών αλυσίδων που βρίσκονται στην ίδια μορφή με αποτέλεσμα να δημιουργούν β-πτυχωτές επιφάνειες (Σχήμα 2.8). Στις περιπτώσεις αυτές οι πλευρικές ομάδες διατάσσονται κάθετα προς την πολυπεπτιδική αλυσίδα.

Από γεωμετρικής πλευράς, οι β-κλώνοι παρουσιάζονται όταν οι γωνίες Φ και Ψ μεταξύ των πολυπετιδικών αλυσίδων και των κετρικών μορίων άνθρακα των αμινοξικών κατάλοιπων παίρνουν περίπου τις τιμές από -120° εώς -180° και από 120° εώς 180°, αντίστοιχα (Σχήμα 2.6). Πιο συγκεκριμένα οι τιμές που παίρνουν οι γωνίες Φ και Ψ παρουσιάζονται στον Πίνακα 2.4.



Σχήμα 2.8: *β-πτυχωτή επιφάνεια(Promponas 2004)*



Σχήμα 2.9: *Αντιπαράλληλοι και παράλληλοι β-κλώνοι, αντίστοιχα(Promponas 2004)*

Όταν οι β-κλώνοι που αποτελούν μια β-πτυχωτή επιφάνεια έχουν την ίδια κατεύθυνση στο χώρο, τότε η επιφάνεια αυτή ονομάζεται παράλληλη β-πτυχωτή επιφάνεια (Σχήμα 2.9). Με την ίδια λογική αν κάποιοι β-κλώνοι της επιφάνειας έχουν αντίθετη κατεύθυνση μεταξύ τους τότε ονομάζεται αντιπαράλληλη β-πτυχωτή επιφάνεια (Σχήμα 2.9). Τέλος όταν σε μια επιφάνεια παρατηρούνται περιπτώσεις τόσο παράλληλων όσο και αντιπαράλληλων β-κλώνων, τότε έχουμε μικτή β-πτυχωτή επιφάνεια. Ανάλογα με το είδος της πτυχωτής επιφάνειας παρουσιάζονται και τα ανάλογα γεωμετρικά χαρακτηριστικά στις πολυπεπτιδικές αλυσίδες (Πίνακας 2.4).

	Φ (°)	***Ψ (°)***
Αντιπαράλληλες β-πτυχωτές επιφάνειες	-140	+135
Παράλληλες β-πτυχωτές επιφάνειες	-120	+115
Στραμμένες β-πτυχωτές επιφάνειες	-120	+135

Πίνακας 2.4: *Τα γεωμετρικά χαρακτηριστικά πτυχωτών επιφανειών, όπου Φ και Ψ δίεδρες γωνίες (Υποκεφάλαιο 2.3.2)*

2.3.3 Τριτοταγής και τεταρτοταγής δομή

Τα διάφορα στοιχεία της δευτεροταγούς δομής αναδιπλώνονται και συνδυάζονται μεταξύ τους για να δώσουν το τελικό σχήμα της πρωτεΐνης στον τρισδιάστατο χώρο, δηλαδή την τριτοταγή δομή. Η τριτοταγής δομή (Σχήμα 2.10) των πρωτεϊνών, παρά την έντονη ερευνητική δραστηριότητα, εμπερικλείει πολλά ερωτηματικά σχετικά με την διαδικασία του αναδιπλώματος στο χώρο. Σημαντική παρατήρηση στις διάφορες τρισδιάστατες δομές πρωτεϊνών είναι ότι σε αυτές παρατηρούνται τμήματα τα οποία φαίνεται ότι είναι δομικά ανεξάρτητα και παρατηρούνται σε διαφορετικές πρωτεΐνες.

Σε αρκετές περιπτώσεις η δομή μιας πρωτεΐνης αποτελείται από περισσότερες την μιας πολυπεπτιδίων αλυσίδων. Αυτή η δομή ονομάζεται τεταρτοταγής δομή (Σχήμα 2.10) και οι πρωτεΐνες ονομάζονται πρωτεϊνικά σύμπλοκα.

Όπως προαναφέρθηκε, αν σε μια πρωτεΐνη έχουμε μια συγκεκριμένη σειρά αμινοξέων τότε πάντοτε έχουμε και την ίδια τρισδιάστατη μορφή σε συγκεκριμένες συνθήκες. Αυτό συμβαίνει διότι μέσω της εξέλιξης η φύση απέρριψε τις αλληλουχίες αμινοξέων οι οποίες δεν μπορούν να δώσουν πάντοτε την ίδια μοναδική και σταθερή δομή πρωτεΐνης η οποία θα εκτελεί συγκεκριμένη λειτουργία. Έτσι στην πραγματικότητα, ενώ υπάρχει ένας τεράστιος συνδυασμός αμινοξέων, μόνο μερικές χιλιάδες αλληλουχίες παράγουν πρωτεΐνες.

Σχήμα 2.10: *Τριτοταγής και τεταρτοταγής δομή πρωτεϊνών(Promponas 2004)*

2.4 Η ανάγκη και τα προβλήματα σχετικά με την μελέτη των πρωτεϊνών

Η λεπτομερής γνώση της τρισδιάστατης μορφής των πρωτεϊνών έχει τεράστια σημασία για τις επιστήμες της βιολογίας, της ιατρικής, αλλά και της φαρμακευτικής. Γνωρίζοντας τη δομή των πρωτεϊνών μπορούν να κατασκευαστούν διάφορα φάρμακα τα οποία αναστέλλουν τη λειτουργία πρωτεϊνών στόχων που εκτελούν συγκεκριμένες λειτουργίες οι οποίες δημιουργούν προβλήματα στον ανθρώπινο οργανισμό. Η γνώση της δομής των πρωτεϊνών βοηθά τις βιομηχανίες να κατασκευάσουν φάρμακα για καταπολέμηση ασθενειών στα διάφορα φυτά ή να κατασκευάσουν ουσίες οι οποίες βοηθούν τα φυτά να μεγαλώνουν με ταχύτερο ρυθμό και να παράγουν περισσότερους καρπούς. Επίσης, άλλες βιομηχανίες, γνωρίζοντας τη δομή των πρωτεϊνών κατασκευάζουν τεχνητές πρωτεΐνες οι οποίες βοηθούν στην αύξηση της μυϊκής μάζας του ανθρώπου.

Γενικά υπάρχουν πολύ απλές διαδικασίες για την καταγραφή της πρωτοταγούς δομής μιας πρωτεΐνης. Στην πραγματικότητα όμως αυτό που θεωρείται χρήσιμο είναι η γνώση της δευτεροταγούς δομής των πρωτεϊνών η οποία μπορεί να οδηγήσει στην τριτοταγή δομή των πρωτεϊνών. Η απευθείας όμως μελέτη της τριτοταγούς δομής είναι μια πολύπλοκη και δαπανηρή διαδικασία. Οι μέθοδοι που χρησιμοποιούνται σε αυτές τις περιπτώσεις είναι η κρυσταλλογραφία ακτινών Χ και η μέθοδος πυρηνικού μαγνητικού συντονισμού (NMR). Επίσης γνωρίζουμε ότι η δευτεροταγής και ακολούθως η τριτοταγής δομή δεν μπορεί να εξαχθεί εύκολα από την πρωτοταγή δομή διότι αυτή μας παρέχει μόνο πληροφορίες για την σειρά των αμινοξέων. Σήμερα δεν υπάρχει τεχνική που να μπορεί να προβλέψει με ακρίβεια την δευτεροταγή και ακολούθως την τριτοταγή δομή μιας πρωτεΐνης μόνο από την πρωτοταγή δομή τους. Οι τεχνικές αυτές αφορούν κυρίως στατιστικές μεθόδους και υλοποιήσεις νευρωνικών δικτύων.

Οι πληροφορίες στο μικροσκοπικό επίπεδο για τις αλληλεπιδράσεις μεταξύ των αμινοξέων, οι οποίες καθορίζουν και τη δομή της πρωτεΐνης, δεν είναι αρκετές ενώ τα αμινοξικά κατάλοιπα αναπτύσσουν τέτοιες δυνάμεις μεταξύ τους που κατά την αναδίπλωση των πρωτεϊνών στο χώρο επηρεάζουν απρόβλεπτα το ένα τη θέση του άλλου. Αυτό έχει ως αποτέλεσμα η τελική μορφή της πρωτείνης, αν αναλογιστούμε ότι ο αριθμός των πιθανών

αλληλουχιών αλλά και των συνδυασμών που μπορούν να εξαχθούν από τα 20 αμινοξέα είναι τεράστιος, να είναι απρόβλεπτη.

2.5 Βασικοί ορισμοί Βιολογίας – Χημείας και η σχέση τους με τα αμινοξέα

2.5.1 Ομοιοπολικός δεσμός

Ομοιοπολικός ονομάζεται ο χημικός δεσμός που αναπτύσσεται μεταξύ ατόμων που μοιράζονται μεταξύ τους ηλεκτρόνια. Συνήθως οι δεσμοί αυτοί παρατηρούνται σε ενώσεις αμετάλλων. Ένα τέτοιο παράδειγμα είναι το νερό. Συγκεκριμένα το νερό σχηματίζεται από τα αμέταλλα υδρογόνο και οξυγόνο. Τα στοιχεία αυτά μοιράζονται ένα κοινό ζεύγος ηλεκτρονίων με αποτέλεσμα να έλκονται μεταξύ τους και να δημιουργούν το μόριο του νερού.

2.5.2 Μη-ομοιοπολικός δεσμός

Μη-ομοιοπολικός ονομάζεται ο χημικός δεσμός που αναπτύσσεται μεταξύ ατόμων τα οποία λόγω του ότι προσφέρουν ή δέχονται ηλεκτρόνια δημιουργούν έλξεις ή απωθήσεις μεταξύ τους. Οι δεσμοί αυτοί εμφανίζονται κυρίως μεταξύ μετάλλων και αμετάλλων. Οι σημαντικότεροι μη ομοιοπολικοί δεσμοί είναι οι εξής:

1. Υδρόφοβες αλληλεπιδράσεις: Σχηματίζονται μεταξύ μη-πολικών ατόμων και κατά συνέπεια μη πολικών αμινοξέων. Σε αυτή την περίπτωση τα αμινοξέα αναγκάζονται να έρθουν σε επαφή επειδή και τα δυο αποστρέφονται το νερό. Είναι η μοναδική κατηγορία μη-ομοιοπολικών δεσμών η οποία οφείλεται σε απωθητικές δυνάμεις. Αμινοξέα τα οποία ανήκουν στην κατηγορία αυτή βρίσκονται στο εσωτερικό των πρωτεϊνών, όπου δεν υπάρχει νερό. Ο δεσμός αυτός είναι ο ισχυρότερος από τους μη-ομοιοπολικούς δεσμούς στην δημιουργία της πρωτεΐνης λόγο της μεγάλης συγκέντρωση νερού μέσα στο κύτταρο.

2. Ιοντικός δεσμός: Σχηματίζονται μεταξύ ατόμων τα οποία έχουν αντίθετο φορτίο και κατά συνέπεια αμινοξέων τα οποία έχουν αντίθετο φορτίο. Συγκεκριμένα το ένα αμινοξύ προσφέρει ηλεκτρόνια και το άλλο δέχεται ηλεκτρόνια. Στην περίπτωση τον δεσμών αυτών ένα μεταλλικό άτομο χάνει ένα ή περισσότερα ηλεκτρόνια προς όφελος του αμετάλλου. Αυτό έχει ως αποτέλεσμα την δημιουργία δυο ιόντων με θετικό και αρνητικό φορτίο αντίστοιχα, με αποτέλεσμα να έλκονται μεταξύ τους. Ο δεσμός αυτός δημιουργείται συνήθως μεταξύ φορτισμένων πλευρικών αλυσίδων των αμινοξέων και αποτελεί τον ισχυρότερο μη-οιπολικό δεσμό μετά τις υδρόφοβες αλληλεπιδράσεις.

3. Δεσμός υδρογόνου: Με βάση τη χημείοα, ο ορισμός είναι ότι δεσμός υδρογόνου ονομάζεται ένα είδος ελκτικής διαμοριακής δύναμης που αναπτύσσεται μεταξύ δύο μερικών ηλεκτρικών φορτιών αντίθετης πολικότητας, λόγω ανισομερούς κατανομής του ηλεκτρικού φορτίου των μορίων. Ένας τυπικός δεσμός υδρογόνου είναι ασθενέστερος τόσο από τους ομοιοπολικούς δεσμούς όσο και από τους μη-ομοιπολικούς δεσμούς που προαναφέραμε. Ο δεσμός υδρογόνου σχηματίζεται όταν ένα άτομο υδρογόνου παρεμβάλλεται μεταξύ δύο ατόμων που έχουν την ικανότητα να προσελκύουν ηλεκτρόνια, όπως το οξυγόνο και το άζωτο. Ο δεσμός αυτός συναντάται στις πρωτεΐνες μεταξύ ενός ατόμου οξυγόνου και ενός ατόμου υδρογόνου τα οποία βρίσκονται σε δυο πετιδικούς δεσμούς.

4. Έλξεις Van der Waals: Οι έλξεις Van der Waals εμφανίζονται όταν δυο άτομα βρίσκονται πολύ κοντά. Οι έλξεις αυτές είναι πολύ ασθενείς, αλλά αποκτούν μεγάλη σημασία στην δημιουργία των πρωτεϊνών όταν οι επιφάνειες δυο μεγάλων μορίων ή πλευρικών αλυσίδων εμφανίζονται πολύ κοντά η μια στην άλλη.

Κεφάλαιο 3

Νευρωνικά Δίκτυα

3.1 Η ανάπτυξη των νευρωνικών δικτύων και η σημασία τους

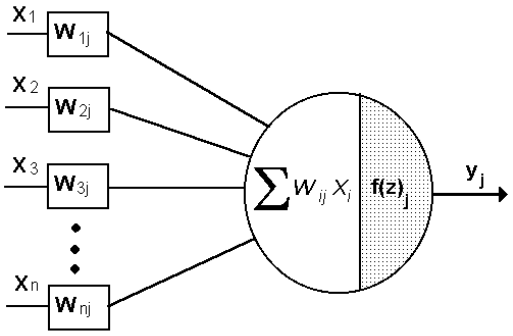
Τα τεχνητά νευρωνικά δίκτυα (ΤΝΔ) αποτελούν ένα σύγχρονο και γοργά αναπτυσσόμενο κλάδο της τεχνητής νοημοσύνης. Τα ΤΝΔ υλοποιούνται με διάφορες τεχνικές οι οποίες προσπαθούν να προσομοιώσουν τον τρόπο με τον οποίο ο ανθρώπινος εγκέφαλος εκπαιδεύεται μέσα από διάφορα δεδομένα που συλλέγει και ανάλογα δίνει εντολές στα διάφορα μέρη του ανθρώπινου σώματος. Ο ανθρώπινος εγκέφαλος αποτελεί το ζωτικότερο και σημαντικότερο όργανο του ανθρώπινου είδους που τον διαφοροποιεί από τους υπόλοιπους ζωντανούς οργανισμούς του πλανήτη. Τα νευρωνικά δίκτυα προσπαθούν να προσομοιώσουν την υπολογιστική ισχύ του εγκεφάλου για επίλυση διαφόρων προβλημάτων, αλλά και για περεταίρω μελέτη του συγκεκριμένου οργάνου. Τα νευρωνικά δίκτυα είναι μαθηματικά μοντέλα τα οποία αποτελούνται από προσομοιωμένους νευρώνες, οι οποίοι αποτελούν το δομικό στοιχείο του ανθρώπινου εγκεφάλου, οι οποίοι ανταλλάσουν μεταξύ τους δεδομένα και προσπαθούν, ανάλογα με την είσοδο που παίρνει το νευρωνικό δίκτυο, να δημιουργήσουν μια χρήσιμη πληροφορία.

Η εξέλιξη των νευρωνικών δικτύων θεωρείται αρκετά σημαντική λόγω του ότι μπορούν να δώσουν λύσεις σε διάφορα υπολογιστικά προβλήματα, τα οποία θα μπορούσαν να λυθούν γρηγορότερα από μια μέθοδο που έχει τα χαρακτηριστικά του ανθρωπίνου εγκεφάλου σε αντίθεση με τα χαρακτηριστικά ενός ηλεκτρονικού υπολογιστή. Συγκεκριμένα ο ανθρώπινος εγκέφαλος μπορεί να επεξεργάζεται πολύπλοκα δεδομένα παράλληλα και με ταχύτητα, με αυτόματη αντιμετώπιση λαθών και ευρωστία. Τα νευρωνικά δίκτυα μπορούν επίσης να προσαρμόζονται ανάλογα με τις απαιτήσεις διαφορετικών προβλημάτων και να μαθαίνουν από τα προβλήματα αυτά ενώ μπορούν να γενικεύουν καταστάσεις ώστε να δίνουν απάντησης σε νέες προς αυτά καταστάσεις. Επίσης δεν έχουν μεγάλες υπολογιστικές απαιτήσεις και μπορούν να λειτουργούν κάτω από την έννοια της μη γραμμικότητας. Όλα αυτά τα χαρακτηριστικά των νευρωνικών δικτύων μπορούν να δώσουν λύσεις σε προβλήματα τα οποία δεν είναι καλά ορισμένα ή δεν έχουν κάποιους κανόνες που οδηγούν σε συγκεκριμένες λύσεις, προβλήματα τα οποία έχουν μεγάλο όγκο δεδομένων και χρειάζονται μεγάλη υπολογιστική ισχύ και ταχύτητα.

Βασικός κανόνας για τη σωστή λειτουργία ενός νευρωνικού δικτύου είναι ότι πρέπει να γίνει σωστή εκπαίδευσή του. Όπως ο ανθρώπινος εγκέφαλος μαθαίνει μέσα από τα δεδομένα που δέχεται στην είσοδό του, έτσι και το νευρωνικό δίκτυο ανάλογα με τα δεδομένα που θα έχει πρέπει μέσα από κάποιους αλγόριθμους και τεχνικές να εκμαιεύει μάθηση από αυτά. Οι αλγόριθμοι μάθησης των νευρωνικών δικτύων χωρίζονται σε τρείς μεγάλες κατηγορίες, επιβλεπόμενη, μη επιβλεπόμενη και ενισχυτική μάθηση. Η επιβλεπόμενη μάθηση έχει την έννοια του δασκάλου, όπου παρουσιάζονται στον ανθρώπινο εγκέφαλο, ανάλογα και στο νευρωνικό δίκτυο, κάποια δεδομένα μαζί με την σωστή λύση. Ανάλογα το ανθρώπινος εγκέφαλος προσπαθεί να γενικεύσει τα δεδομένα και τις λύσεις ώστε να μάθει τι πρέπει μα κάνει σε νέες καταστάσεις τι οποίες δεν ξανασυνάντησε. Η μη επιβλεπόμενη μάθηση ασχολείται με τις περιπτώσεις που ο ανθρώπινος εγκέφαλος δέχεται πολλά δεδομένα και ανάλογα με τα κοινά χαρακτηριστικά τους που εντοπίζει προσπαθεί να τα ομαδοποιήσει. Τέλος η ενισχυτική μάθηση προσομοιώνει τη μέθοδο με την οποία ο ανθρώπινος εγκέφαλος μαθαίνει μέσα από τη διαδικασία της επιβράβευσης και της τιμωρίας, δηλαδή ανάλογα με το τι δέχεται στην είσοδο του και τι αποτέλεσμα έχει στην έξοδό του, αν τιμωρηθεί για αυτό προσπαθεί να μην το επαναλάβει ενώ αν επιβραβευτεί προσπαθεί να το επαναλάβει.

3.2 Τοπολογία νευρωνικών δικτύων

3.2.1 Μοντέλο McCulloch και Pitts και συναρτήσεις ενεργοποίησης



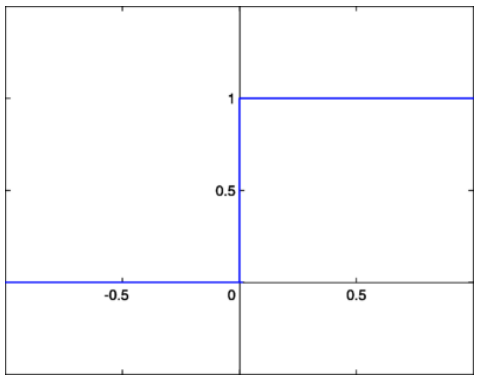
Σχήμα 3.1: *Το μοντέλο McCulloch και Pitts (Branting20 May 2009)*

Κομβικό σημείο για την ανάπτυξη των νευρωνικών δικτύων θεωρείται η δουλειά των McCulloch και Pitts (1943)(McCulloch and Pitts 1943) σχετικά με ένα μοντέλο το οποίο προσομοιώνει τη λειτουργία ενός νευρώνα του ανθρώπινου εγκεφάλου. Το μοντέλο (Σχήμα 3.1) δέχεται ένα σύνολο τιμών εισόδου τις οποίες συσχετίζει με διαφορετικά βάση τα οποία ενδυναμώνουν ή αποδυναμώνουν τις συγκεκριμένες εισόδους. Είσοδος του νευρώνα είναι το άθροισμα των γινομένων μεταξύ των τιμών εισόδων και των συγκεκριμένων βαρών(Εξίσωση 3.1). Τις τιμές εισόδου μπορεί να αποτελούν κάποια δεδομένα εισόδου στο δίκτυο ή τιμές εξόδου άλλων νευρώνων. Το μέρος αυτό του μοντέλου προσομοιώνει τις συνάψεις με άλλους νευρώνες που βρίσκονται στους δεντρίτες του νευρώνα. Στη συνέχεια υπάρχει μια τιμή κατωφλίου. Αν η τιμή εισόδου του νευρώνα είναι πάνω από το κατώφλι τότε ο νευρώνας δίνει στην έξοδο τιμή ένα, διαφορετικά μηδέν. Για την υλοποίηση της διαδικασίας αυτής η είσοδος του νευρώνα περνά από μια μη-γραμμική βηματική συνάρτηση και ανάλογα δίνει στην έξοδο του νευρώνα κάποια τιμή εξόδου. Το μοντέλο αυτό μπορεί να δημιουργήσει και να κρατήσει κάποια γνώση όταν συνενωθεί με άλλους νευρώνες του ιδίου μοντέλου, αφού οι τιμές των βαρών και του κατωφλίου είναι μεταβλητές κατά τη διάρκεια εκπαίδευσης του δικτύου, ώστε δημιουργείται κάποιο είδος γνώσης.

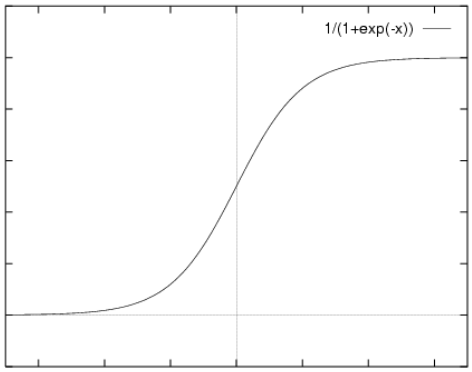
$$netj = \sum_{j=1}^{n} W\,ijXi$$

Εξίσωση 3.1: *Εξίσωση εισόδου στο μοντέλο, όπου Xi είναι η έξοδος νευρώνων που προηγούνται και Wij το βάρος μεταξύ του νευρώνα i και j.*

Η βηματική συνάρτηση που χρησιμοποιήθηκε στο μοντέλο των McCulloch και Pitts ήταν η συνάρτηση κατωφλίου (Σχήμα 3.2). Η μη-γραμμική αυτή συνάρτηση θέτει κάποια τιμή στον άξονα X η οποία αντιπροσωπεύει την τιμή κατωφλίου. Αν η είσοδος του νευρώνα (Εξίσωση 3.1) είναι μεγαλύτερη της τιμή κατωφλίου τότε στην έξοδο ο νευρώνας δίνει ένα διαφορετικά μηδέν.



Σχήμα 3.2: *Η συνάρτηση κατωφλίου*



Σχήμα 3.3: *Η σιγμοειδής συνάρτηση*

Η συνάρτηση κατωφλίου όμως έχει δύο σοβαρά μειονεκτήματα που αφορούν τους αλγόριθμους μάθηση. Το πρώτο μειονέκτημα είναι ότι δεν μπορεί να παραγωγισθεί σε όλο το πεδίο ορισμού ώστε να λειτουργήσει σωστά η μέθοδος κατάβασης κλήσης (Υποκεφάλαιο 3.3.1) ενώ σε περίπτωση που ο νευρώνας δεν δίνει σωστό αποτέλεσμα δεν μας δίνει στοιχεία για να υπολογίσουμε την απόκλιση λάθους. Σε προβλήματα ανάλογα με αυτό που εξετάζεται σε αυτή την έρευνα η συνάρτηση ενεργοποίηση που χρησιμοποιείται είναι η σιγμοειδής. Η σιγμοειδής συνάρτηση (Εξίσωση 3.2) δεν παρουσιάζει τους πιο πάνω περιορισμούς.

$$f(x) = \frac{1}{1 + \mathrm{e}^{-\mathrm{ax}}}$$

Εξίσωση 3.2: *Εξίσωση σιγμοειδούς συνάρτησης, όπου a είναι η κλήση της σιγμοειδούς συνάρτησης.*

3.2.2 Νευρωνικά δίκτυα πολλαπλών στρωμάτων Perceptron

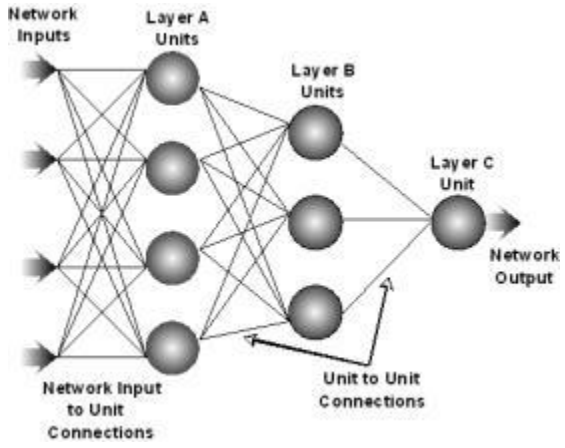
Τα νευρωνικά δίκτυα με βάση το πρόβλημα και την επιθυμητή λύση, μπορούν να οργανωθούν σε διαφορετικές τοπολογίες. Τα πολυστωρματικά δίκτυα Perceptrons είναι πολυστωρματικά δίκτυα εμπρόσθιου περάσματος (feed-forward) τα οποία χτίζονται με μη-γραμμικούς κόμβους του μοντέλου McCulloch και Pitts (Σχήμα 3.4). Κάθε στρώμα αποτελείται από μια σειρά κόμβων των οποίων οι έξοδοι συνδέονται μέσω βαρών με τις εισόδους νευρώνων επόμενων στρωμάτων. Κάθε νευρωνικό δίκτυο αυτού του τύπου αποτελείται από ένα στρώμα εισόδου, το οποίο δεν είναι ενεργό επίπεδο αφού σκοπός του είναι απλά να παρουσιάζει τα δεδομένα εισόδου στο δίκτυο, ένα ή περισσότερα κρυφά στρώματα και ένα στρώμα εξόδου, τα οποία είναι ενεργά στρώματα. Σημαντικό σε αυτό το σημείο είναι να αναφέρουμε το πώς οι κρυφοί κόμβοι αποθηκεύουν γνώση. Συγκεκριμένα οι κόμβοι αυτοί ανιχνεύουν και αποθηκεύουν πρότυπα των πιο χαρακτηριστικών τιμών εισόδων του προβλήματος, με βάση τα βάρη και την τιμή κατωφλίου τους. Έτσι οι κρυφοί νευρώνες προσπαθούν να δημιουργήσουν όρια αποφάσεων, τα οποία ανάλογα με την είσοδο γενικεύουν καθορίζουν την κατηγορία της εξόδου. Ο αριθμός των νευρώνων και των επιπέδων που χρειάζονται για την επίλυση κάποιου προβλήματος δεν μπορεί να καθοριστεί εξ' ορισμού. Ο αριθμός των νευρώνων του επιπέδου εισόδου εξαρτάται από την κωδικοποίηση των δεδομένων εισόδου και ο αριθμός των νευρώνων του επιπέδου εξόδου εξαρτάται από τις επιθυμητές εξόδους του δικτύου. Ο αριθμός των κρυφών επιπέδων και ο αριθμός των νευρώνων του κάθε επιπέδου για την καλύτερη επίλυση ενός προβλήματος καθορίζονται μέσα από διάφορα πειράματα. Ένα στοιχείο που βοηθά στην σωστή δημιουργία ενός δικτύου είναι ότι θεωρητικά, με βάση το θεώρημα Kolmogorov (Kolmogorov 1957), μέχρι δυο κρυφά επίπεδα είναι αρκετά για να δημιουργήσουν τόσα

όρια αποφάσεων ώστε να διαχωριστούν όλες οι κλάσεις του προβλήματος, αλλά και σε αυτή την περίπτωση χρειάζονται πειράματα για τον καθορισμό των κρυφών νευρώνων.

Τα νευρωνικά δίκτυα perceptron μπορούν να εκπαιδευτούν μέσα από ένα απλό αλγόριθμο μάθηση. Σύμφωνα με τον αλγόριθμο αυτό, αφού γίνει στην αρχή η τυχαία αρχικοποίηση των βαρών και των κατωφλίων, παρουσιάζουμε στο δίκτυο τα κωδικοποιημένα δεδομένα εισόδου και την επιθυμητή έξοδο. Στη συνέχεια υπολογίζουμε την έξοδο του δικτύου και ανάλογα τροποποιούμε τα βάρη και τα κατώφλια των κόμβων του δικτύου ώστε αυτό να μάθει τα δεδομένα που του παρουσιάζουμε. Οι αλλαγές στα βάρη και τα κατώφλια γίνονται με βάση τον κανόνα δέλτα (Εξίσωση 3.3) (Widrow and Hoff 1960) ο οποίος παρουσιάζει τις μετατροπές των λαθών ως ανάλογες του λάθους, που υπολογίζεται με βάση την απόκλιση της πραγματικής από την επιθυμητή έξοδο, της εισόδου του νευρώνα και του ρυθμού εκμάθησης. Για την εκπαίδευση του κατωφλίου, αυτό παρουσιάζεται στην είσοδο του κάθε νευρώνα ως μια επιπλέον σύνδεση με είσοδο τιμής -1 το οποίο εκπαιδεύεται επίσης με τον κανόνα Δ.

$$Wi(t+1) = Wi(t) + n\Delta Xi(t)$$

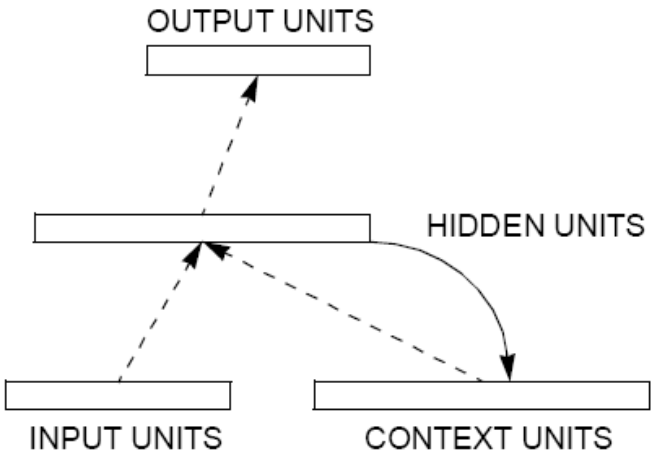
Εξίσωση 3.3: *Ο κανόνα δέλτα, όπου W το βάρος, n ο ρυθμού εκμάθησης, Δ το λάθος και Xi η είσοδος του νευρώνα i.*



Σχήμα 3.4: *Ένα νευρωνικό δίκτυο Perceptron με δύο κρυφά επίπεδα (Branting20 May 2009)*

3.2.3 Νευρωνικό δίκτυο με ανάδραση

Τα νευρωνικά δίκτυα με ανάδραση έχουν την ίδια δομή με τα πολυστρωματικά δίκτυα perceptron αλλά με μια σημαντική διαφορά: η είσοδος του δικτύου ανατροφοδοτείται από μέρος τον δεδομένων που δημιουργεί το δίκτυο. Συγκεκριμένα τα δεδομένα εισόδου του δικτύου αποτελούνται από τα δεδομένα εισόδου της συγκεκριμένης χρονικής στιγμής και τα δεδομένα εξόδου των νευρώνων του επιπέδου εξόδου ή κάποιου κρυφού επιπέδου. Σκοπός της συγκεκριμένης αρχιτεκτονικής νευρωνικών δικτύων είναι η δημιουργία χρονικών καταστάσεων από τις οποίες εξαρτάται κάθε επόμενη είσοδος στο δίκτυο. Για το σκοπό αυτό υπάρχει ένα επιπλέον context επίπεδο το οποίο σκοπό έχει απλά να δέχεται τις εξόδους των κόμβων που χρησιμοποιούνται για ανατροφοδότηση και μετά από μια σταθερή αλλαγή στην τιμή αυτή να παρουσιάζει τα δεδομένα αυτά στην είσοδο του νευρωνικού δικτύου. Με αυτό τον τρόπο δημιουργούμε κάποια μορφή μνήμης στο δίκτυο η οποία δίνει τη δυνατότητα στο δίκτυο η έξοδος του να μην εξαρτάται μόνο από τα δεδομένα εισόδου τις συγκεκριμένης χρονική στιγμής αλλά και από μια σειρά δεδομένων που καθορίζονται από ένα κινούμενο παράθυρο στο χρόνο. Το κινούμενο παράθυρο καθορίζει την ακολουθία των δεδομένων τα οποία πρέπει να περάσουν στο δίκτυο για να έχουμε το επιθυμητό αποτέλεσμα στην έξοδο.



Σχήμα 3.5: *Μια απλή προσέγγιση δικτύου με ανάδραση του Elman (Elman 1990).*

Μια διαδεδομένη αρχιτεκτονική νευρωνικών δικτύων με ανάδραση είναι αυτή που πρότεινε ο Elman (Σχήμα 3.5) (Elman 1990). Σε αυτή την αρχιτεκτονική, η ανάδραση ξεκινά από το κρυφό επίπεδο του δικτύου και καταλήγει σε ένα context επίπεδο το οποίο με μια σταθερά ελαττώνει ρυθμίζει τα δεδομένα που φτάνουν στην είσοδό του. Θεωρητικά η σταθερά αυτή καθορίζει τη σχέση που έχουν τα δεδομένα τη συγκεκριμένη χρονική στιγμή με προηγούμενες χρονικές καταστάσεις. Στη συνέχεια ανατροφοδοτεί την είσοδο του δικτύου με τα δεδομένα αυτά. Αυτή η διαδικασία δημιουργεί μια εσωτερική μνήμη στο δίκτυο η οποία δεν επηρεάζεται από την έξοδο του δικτύου. Μέσα από αυτή τη διαδικασία το δίκτυο μπορεί να δώσει λύση σε προβλήματα που έχουν σχέση με κάποια χρονική εξέλιξη.

3.3 Αλγόριθμοι μάθησης

3.3.1 Μέθοδος κατάβασης κλίσης

Η μέθοδος κατάβασης κλήσης αποτελεί μια ευρέως διαδεδομένη μέθοδο υπολογισμού της αλλαγής που πρέπει να γίνει στο βάρη ενός νευρωνικού δικτύου ώστε αυτό να εκπαιδευτεί με επιβλεπόμενη μάθηση. Σύμφωνα με τη μέθοδο αυτή αρχικά υπολογίζουμε το λάθος στην έξοδο του κάθε νευρώνα του επιπέδου εξόδου με μια στατιστική μέθοδο υπολογισμού του λάθους. Συνήθως η μέθοδος που χρησιμοποιείται είναι το Τετραγωνικό Σφάλμα (LMS) (Εξίσωση 3.4). Η μέθοδος αυτή δίνει για κάθε νευρώνα το λάθος ως τετραγωνική ρίζα της απόκλισης της πραγματικής εξόδου του νευρώνα από την επιθυμητή έξοδο που θα έπρεπε να έχει ο νευρώνας.

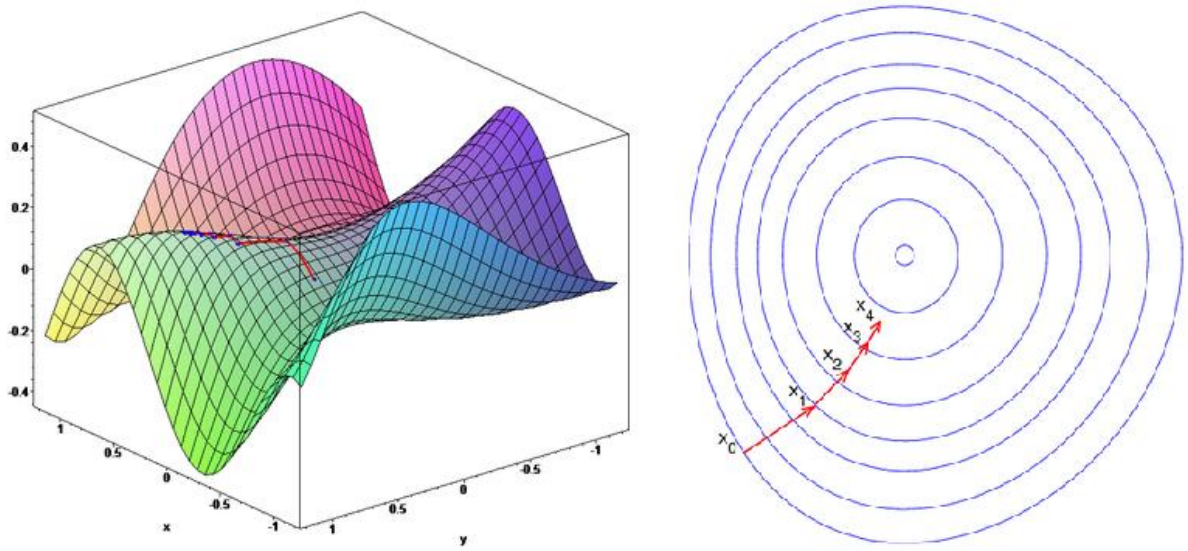
$$E = 1/2 \sum_{j} (t_{pj} - o_{pj})^2$$

Εξίσωση 3.4: *Τετραγωνικό Σφάλμα (LMS) , όπου t_{pj} η επιθυμητή έξοδος του νευρώνα j και o_{pj} η πραγματική έξοδος του νευρώνα j*

Η μέθοδος κατάβασης κλήσης, χρησιμοποιώντας την πιο πάνω πληροφορία και τον κανόνα δέλτα (Εξίσωση 3.3) (Widrow and Hoff 1960) καθορίζει την αλλαγή που πρέπει να γίνει στο βάρη των νευρώνων ώστε το δίκτυο να εκπαιδευτεί. Η κύρια ιδέα της μεθόδου αυτής είναι ότι η αλλαγή των βαρών είναι ανάλογη του αρνητικού της παραγώγου της συνάρτησης του λάθους ως προς τα βάρη (Εξίσωση 3.5). Η μέθοδος κατάβασης κλήσης προσπαθεί να δώσει τέτοιες τιμές στα βάρη ώστε το δίκτυο να δίνει το μικρότερο δυνατό λάθος στην έξοδό του, κάτι το οποίο σημαίνει ότι υπάρχουν περισσότερες πιθανότητες το δίκτυο να είναι κοντά στο επιθυμητό αποτέλεσμα. Παρατηρώντας την γραφική παράσταση του λάθους ως προς τα βάρη (Σχήμα 3.6), τα βάρη τα οποία θα δώσουν το μικρότερο δυνατό λάθος βρίσκονται στο ολικό ελάχιστο της γραφικής αυτής παράστασης. Μερικές φορές, παρατηρείται το φαινόμενο οι τιμές των βαρών να εγκλωβίζονται σε μια άτρακτο με τοπικό ελάχιστο με αποτέλεσμα να μην εκπαιδεύεται το δίκτυο με τα καλύτερα δυνατά βάρη. Σε αυτή την περίπτωση χρησιμοποιούμε το ρυθμό εκμάθησης και την ορμή που καθορίζουν πόσο μεγάλες θα είναι οι αλλαγές στα βάρη ώστε πάντα στόχος να είναι το ολικό και όχι κάποιο τοπικό ελάχιστο.

$$\Delta w_{ij} = -n\frac{\vartheta E_p}{\vartheta w_{ij}}$$

Εξίσωση 3.5: *Η αλλαγή των βαρών στη μέθοδο κατάβασης κλήσης, όπου* n *ο ρυθμός εκμάθησης,* E_p *το λάθος και* w_{ij} *το βάρος από τον νευρώνα* i *στον* j



Σχήμα 3.6: *Η γραφική παράσταση του λάθους E ως προς τα βάρη των νευρώνων και ένα ελάχιστο το οποίο προσεγγίζεται από κάποια βάρη X (Gradient descent 2009)*

3.3.2 Αλγόριθμος εκμάθησης ανάστροφης μετάδοσης του λάθους

Ο αλγόριθμος εκμάθησης ανάστροφής μετάδοσης λάθους (Backpropagation Algorithm) (Werbos 1974) χρησιμοποιείται για την εκπαίδευση νευρωνικών δικτύων Perceptron και συγκεκριμένα για την διόρθωση των βαρών και του κατωφλίου, που αντιμετωπίζεται ως επιπλέων βάρος, του δικτύου. Ο αλγόριθμος αυτό αποτελείται από δυο φάσεις οι οποίες χαρακτηρίζονται από ένα πέρασμα του δικτύου η κάθε μια. Στην πρώτη φάση, Forward, παρουσιάζονται στην είσοδο του δικτύου τα δεδομένα ούτως ώστε να παρουσιαστούν στην έξοδο κάποια αποτελέσματα. Ακολούθως με μια στατιστική μέθοδο, όπως η LMS (Εξίσωση 3.4), υπολογίζεται το λάθος του δικτύου ως προς το επιθυμητό αποτέλεσμα και ακολουθεί η δεύτερη φάση. Στη φάση αυτή έχουμε ένα πέρασμα του δικτύου προς τα πίσω, Backward, με το οποίο μεταφέρουμε το λάθος που παρουσιάστηκε στους νευρώνες εξόδου προς τα βάρη του δικτύου, τα οποία αναπροσαρμόζονται με βάση τον κανόνα δέλτα (Εξίσωση 3.3) (Widrow and Hoff 1960). Οι τύποι που χρησιμοποιούνται για την διόρθωση των βαρών του δικτύου υπολογίζονται με βάση τη μέθοδο κατάβαση κλήσης, τον κανόνα

δέλτα, τη μέθοδο τετραγωνικού σφάλματος (Υποκεφάλαιο 3.3.1) και την σιγμοειδή συνάρτηση (Υποκεφάλαιο 3.2.1).

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

Εξίσωση 3.6: *Ο υπολογισμός του λάθους για νευρώνες του επιπέδου εξόδου, όπου t_{pj} η επιθυμητή έξοδος του νευρώνα j και o_{pj} η πραγματική έξοδος του νευρώνα j.*

$$\delta_{pj} = O_{pj} (1- O_{pj}) \Sigma \delta_{pк} W_{jк}$$

Εξίσωση 3.7: *Ο υπολογισμός του λάθους για νευρώνες κρυφών επιπέδων, όπου o_{pj} η πραγματική έξοδος του νευρώνα j, $\delta_{pк}$ το λάθος από τον νευρώνα κ που είναι συνδεδεμένος με την έξοδο του j και $W_{jк}$ το βάρος που συνδέει τους δυο νευρώνες.*

Ο αλγόριθμος εκμάθησης ανάστροφης μετάδοσης λάθους έχει ως εξής:

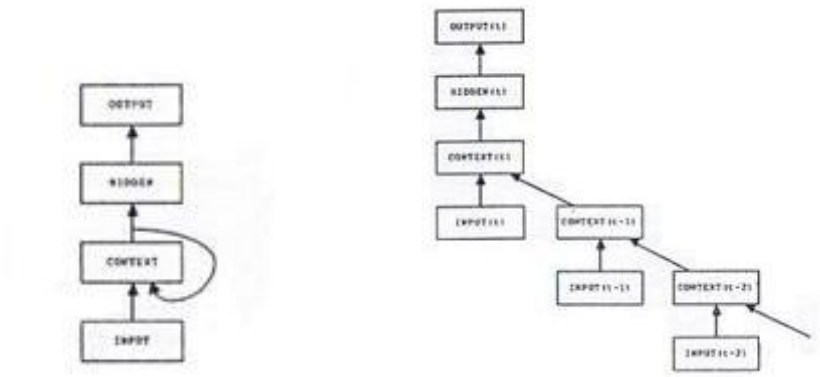
1) Αρχικοποίηση όλων των βαρών του δικτύου με τυχαίες τιμές 0 μέχρι 1.
2) Επανέλαβε μέχρι να μειωθεί το λάθος σε επιθυμητό επίπεδο
   a) Παρουσίασε την είσοδο στο δίκτυο
   b) Αφού υπολογιστεί το πραγματικό αποτέλεσμα του δικτύου υπολόγισε το λάθος
   c) Με βάση τον κανόνα δέλτα μετέτρεψε τα λάθη αντικαθιστώντας στο D ανάλογα το αποτέλεσμα της Εξίσωσης 3.6 ή 3.7

3.3.3 Εκπαίδευση νευρωνικού δικτύου με ανάδραση και ξεδίπλωμα

Τα νευρωνικά δίκτυα με ανάδραση μπορούν να εκπαιδευτούν και αυτά με τον αλγόριθμο εκμάθησης ανάστροφης μετάδοσης λάθους. Για να μπορέσει ο συγκεκριμένος αλγόριθμος να εκπαιδεύσει σωστά τα συγκεκριμένα δίκτυα, τα οποία διατηρούν μνήμη, πρέπει να γίνουν κάποιες τροποποιήσεις στον αλγόριθμο αυτό αλλά και κάποιες υποθέσεις.

Για την εκπαίδευση κάποιου δικτύου με ανάδραση πρέπει πρώτα αυτό να ξεδιπλωθεί στο χρόνο (Σχήμα 3.7) (Mozer 1989). Συγκεκριμένα πρέπει να δημιουργηθεί ένα αντίστοιχο δίκτυο χωρίς ανάδραση, το οποίο θα έχει ένα επιπλέον επίπεδο για κάθε χρονική στιγμή. Ένα τέτοιο δίκτυο γίνεται πολύπλοκο ενώ απαιτεί πολλή μνήμη για κάθε νέα κατάσταση του δικτύου. Έτσι για να δημιουργηθεί αυτό το ξεδιπλωμένο δίκτυο γίνεται η υπόθεση ότι κάθε νέο επίπεδο που δημιουργείται είναι αντίγραφο κάποιου επιπέδου του δικτύου με ανάδραση. Το ίδιο ισχύει και για όλα τα βάρη που συνδέουν τα επίπεδα που δημιουργήθηκαν, είναι αντίγραφα των βαρών του δικτύου με ανάδραση. Τα επίπεδα αυτά έχουν σαν είσοδό τους, αναδρομικά, τα δεδομένα εξόδου της προηγούμενης χρονικής στιγμής και τα νέα δεδομένα εισόδου της συγκεκριμένης χρονικής στιγμής. Με αυτή την υπόθεση όλα τα αντίστοιχα βάρη και κόμβοι τον επιπέδων που δημιουργούνται είναι ίδια μεταξύ του και το δίκτυο ξεδιπλώνεται στον χρόνο. Το δίκτυο αυτό είναι αντίστοιχο ενός πολύστρωματικού δικτύου perceptron και μπορεί να εκπαιδευτεί με τον αλγόριθμο εκμάθησης ανάστροφης μετάδοσης λάθους.

Ένας αλγόριθμος ο οποίος χρησιμοποιείται για την εκπαίδευση των δικτύων αυτών είναι ο αλγόριθμος εκμάθησης ανάστροφης μετάδοσης του λάθους διά μέσω χρόνου (Werbos 1990). Ο αλγόριθμος αυτός αθροίζει το λάθος κάθε χρονικής στιγμής μέχρι να φτάσει στο τέλος του παραθύρου εισόδου. Στη συνέχεια, αναδρομικά κινείται πίσω στο ξεδιπλωμένο δίκτυο και εκτελεί τις ανάλογες διορθώσεις στα βάρη.



Σχήμα 3.7: *Το νευρωνικό δίκτυο με ανάδραση και το αντίστοιχο ξεδιπλωμένο (Mozer 1989)*

3.4 Γιατί χρησιμοποιούνται νευρωνικά δίκτυα στο PSSP

Με βάση όλα όσα περιγράφονται στο συγκεκριμένο κεφάλαιο, τα νευρωνικά δίκτυα φαντάζουν μια ιδανική λύση για την επίλυση του προβλήματος PSSP. Το PSSP είναι ένα πρόβλημα στο οποίο δεν μπορούν να εφαρμοστούν κάποιοι κανόνες οι οποίοι μπορούν από την πρωτοταγή δομή να οδηγήσουν στην δευτεροταγή δομή μιας πρωτεΐνης, λόγω των προβλημάτων που παρουσιάζονται στο Υποκεφάλαιο 2.4. Για αυτό το λόγο πολλές μέθοδοι για την λύση του PSSP καταφεύγουν στα νευρωνικά δίκτυα.

Κάποιο νευρωνικό δίκτυο που θα χρησιμοποιηθεί για το συγκεκριμένο πρόβλημα μπορεί να εκπαιδευτεί με πρωτεΐνες για τις οποίες υπάρχει καταγραμμένη τόσο η πρωτοταγής όσο και η δευτεροταγής δομή τους. Τα δεδομένα αυτά περιέχουν τεράστιο όγκο πληροφοριών καθώς και ελλείπη ή θορυβώδη δεδομένα, κάτι το οποίο μπορούν να διαχειριστούν σωστά και ανθεκτικά τα νευρωνικά δίκτυα. Σε μια τέτοια περίπτωση τα νευρωνικά δίκτυα μπορούν να γενικεύσουν τα δεδομένα εισόδου και να αποθηκεύσουν στα βάρη τους πληροφορίες για συγκεκριμένα μοτίβα των πρωτεϊνών και των σχέσεων μεταξύ τους που δεν μπορεί να εντοπίσει εύκολα ο άνθρωπος. Μπορούν να δημιουργήσουν τα σωστά όρια αποφάσεων στο χώρο που βρίσκονται τα διανύσματα των δεδομένων εισόδου του δικτύου και να προβλέπουν με βάση γνωστά μοτίβα την δευτεροταγή δομή πρωτεϊνών των οποίων δεν είναι γνωστή η δευτεροταγής δομή.

Κεφάλαιο 4

Δεδομένα Εισόδου

4.1 Δεδομένα εισόδου και βάσεις δεδομένων

4.1.1 Γενική περιγραφή δεδομένων εισόδου

Η σωστή επιλογή του συνόλου δεδομένων είναι πάντοτε πολύ σημαντική στην εκπαίδευση ενός νευρωνικού δικτύου. Στο συγκεκριμένο πρόβλημα, τα δεδομένα εισόδου στο δίκτυο είναι πρωτεϊνικές ακολουθίες των οποίων γνωρίζουμε τόσο την πρωτοταγή όσο και την δευτεροταγή δομή. Συγκεκριμένα, είναι πρωτεΐνες με προσδιορισμένη τρισδιάστατη δομή και δευτεροταγή δομή καθορισμένη από το πρόγραμμα DSSP. Τα δεδομένα τα οποία θα επιλεχθούν για την συγκεκριμένη υλοποίηση πρέπει να έχουν κάποια χαρακτηριστικά ώστε να μην υπάρχει οποιαδήποτε υπόνοια ως προς την ορθότητά τους. Ακολούθως αυτά πρέπει να επιλεχθούν και να κωδικοποιηθούν ώστε να μπορεί να τα αναγνώσει το νευρωνικό δίκτυο που θα κατασκευαστεί.

Τα δεδομένα εισόδου για το συγκεκριμένο είδος προβλήματος επιλέγονται από παγκόσμιες βάσεις δεδομένων πρωτεϊνών στις οποίες παρέχεται ελεύθερη πρόσβαση μέσω του διαδικτύου (Υποκεφάλαιο 4.1.2). Στις συγκεκριμένες βάσεις δεδομένων υπάρχουν πληροφορίες για πρωτεϊνικές ακολουθίες, δευτεροταγείς και τριτοταγείς δομές συγκεκριμένων πρωτεϊνών, πληροφορίες για τα πειράματα που έγιναν ώστε να εξαχθεί η τριτοταγής δομή κάποιας πρωτεΐνης, αλλά και πολλές άλλες πληροφορίες που σχετίζονται με τα συγκεκομμένα δεδομένα.

Για την εκπαίδευση και δοκιμή ενός νευρωνικού δικτύου το οποίο θα προβλέπει την δευτεροταγή δομή πρωτεϊνών χρειαζόμαστε, για κάθε πρωτεΐνη η οποία θα επιλεχθεί στο σύνολο δεδομένων, το όνομά της, την πρωτοταγή δομή της και τη δευτεροταγή δομή της. Η πρωτοταγής δομή στις βάσεις δεδομένων περιγράφεται ως μια ακολουθία από το σύνολο γραμμάτων που συμβολίζουν τα αμινοξέα (Πίνακας 2.2). Ανάλογα, η δευτεροταγής δομή περιγράφεται ως μια ακολουθία από το σύνολο γραμμάτων που συμβολίζουν τις δευτεροταγείς μορφές που μπορεί να έχει μια πρωτεΐνη. Σε αντίστοιχες υλοποιήσεις νευρωνικών δικτύων, η δευτεροταγής δομή συμβολίζεται με τρία γράμματα, ένα για κάθε μια από τις ομάδες των α-ελίκων, β-κλόνων και το σύνολο των υπόλοιπων κατηγοριών.

4.1.2 Βάσεις δεδομένων πρωτεϊνών και DSSP

Οι βάσεις δεδομένων στο διαδίκτυο, οι οποίες διατηρούν δεδομένα για πρωτεΐνες και μπορούν να μας δώσουν τις απαραίτητες πληροφορίες που χρειαζόμαστε για να εκπαιδεύσουμε και να ελέγξουμε το νευρωνικό δίκτυο αυτής της έρευνας, διαθέτουν όλων των ειδών πληροφορίες που αφορούν τις πρωτεϊνικές ακολουθίες. Όπως προαναφέραμε (Υποκεφάλαιο 4.1.1) στις συγκεκριμένες βάσεις δεδομένων υπάρχουν πληροφορίες για τις δευτεροταγείς και τριτοταγείς δομές συγκεκριμένων ακολουθιών, πληροφορίες για τα πειράματα που έγιναν ώστε να εξαχθεί η τριτοταγής δομή κάποιας πρωτεΐνης, αλλά και πολλές άλλες πληροφορίες που σχετίζονται με τις πρωτεΐνες. Στις συγκεκριμένες βάσεις δεδομένων μπορεί οποιοσδήποτε να έχει πρόσβαση, ενώ διαθέτουν μηχανισμούς για μεταφορά αρχείων πρωτεϊνών από τις βάσεις δεδομένων σε προσωπικούς υπολογιστές. Επίσης, διαθέτουν προγράμματα τα οποία μπορούν να φιλτράρουν και να επιστρέψουν στο χρήστη συγκριμένες πληροφορίες σχετικά με τις πρωτεΐνες. Οι συγκεκριμένες βάσεις δεδομένων ενημερώνονται και ελέγχονται σε εβδομαδιαία βάση, ούτως ώστε να παρέχουν αξιόπιστες πληροφορίες στους ερευνητές του είδους.

Μια από τις μεγαλύτερες βάσεις δεδομένων που βρίσκονται στο διαδίκτυο και περιέχουν πληροφορίες για πρωτεΐνες είναι η iProClass (PIR Abbr. Protein Information Resource, 20 April 2009) που βρίσκεται στο Georgetown University Medical Center. Σύμφωνα με την iProClass βάση δεδομένων του PIR υπάρχουν καταχωρημένες περίπου 3,500,000 αλληλουχίες αμινοξέων από τις οποίες μόνο περίπου 53000 γνωρίζουμε την δευτεροταγή δομή τους. Μια άλλη πολύ μεγάλη βάση δεδομένων η οποία διαθέτει πληροφορίες για την τριτοταγή δομή περίπου 53000 πρωτεϊνικών ακολουθιών, είναι η PDB (RCSB Protein Data Bank, 20 April 2009). Στη συγκεκριμένη βάση δεδομένων καταχωρούνται πρωτεϊνικές ακολουθίες με τις αντίστοιχες τριτοταγείς δομές τους σε συγκεκριμένη μορφή, οι οποίες εξάχθηκαν με την μέθοδο κρυσταλλογραφίας ακτινών Χ και την μέθοδο πυρηνικού μαγνητικού συντονισμού (NMR).

Μια ακόμα σημαντική πηγή δεδομένων στην έρευνα σχετικά την πρόβλεψη δευτεροταγούς δομής πρωτεϊνών είναι το πρόγραμμα DSSP (DSSP 20 April 2009, Kabsch and Sander

1983). Το συγκεκριμένο πρόγραμμα χρησιμοποιείται για τυποποίηση της δευτεροταγούς δομής όλων των πρωτεϊνών που είναι καταγεγραμμένη η τριτοταγής δομή τους στην PDB (RCSB Protein Data Bank, 20 April 2009). Το πρόγραμμα αυτό δεν κάνει πρόβλεψη δευτεροταγούς δομής αλλά με βάση τα στοιχεία που παίρνει από την PDB για τη διεύθυνση/γεωμετρία των αμινοξικών καταλοίπων στο χώρο και τις θέσεις των υδρογονικών δεσμών των πρωτεϊνών και ανιχνεύοντας διάφορα μοτίβα δημιουργεί αρχεία στα οποία υπάρχουν πληροφορίες για την τις πρωτεϊνικές ακολουθίες των πρωτεϊνών, την πρωτοταγή, δευτεροταγή και τριτοταγή δομή τους κτλ. Στα αρχεία αυτά μπορεί να έχει πρόσβαση οποιοσδήποτε μέσο της ιστοσελίδας του DSSP (DSSP, 20 April 2009). Η τυποποίηση της δευτεροταγούς δομής των πρωτεϊνών του DSSP γίνεται με βάση τον Πίνακα 4.1. Δυο άλλα προγράμματα τα οποία χρησιμοποιούνται για τυποποίηση της δευτεροταγούς δομής, χωρίς όμως να είναι τόσο διάσημα όσο το DSSP είναι το DEFINE (Frishman and Argos 1995) και το STRIDE (Richards and Kundrot 1988). Το DEFINE χρησιμοποιεί πίνακες διαφορικών αποστάσεων για να υπολογίσει και να συγκρίνει τις αποστάσεις των πρωτεϊνικών ατόμων με τυποποιημένες δευτεροταγείς δομές ενώ το STRIDE χρησιμοποιεί τις δίεδρες γωνίες των πρωτεϊνών (Υποκεφάλαιο 2.2.1) για να υπολογίσει την δευτεροταγή δομή.

Σύμβολο	**Δευτεροταγής Δομή του DSSP**
H	alpha helix
B	residue in isolated beta-bridge
E	extended strand, participates in beta ladder
G	3-helix (3/10 helix)
I	5 helix (pi helix)
T	hydrogen bonded turn
S	Bend
L	the rest

Πίνακας 4.1: *Η τυποποίηση της δευτεροταγούς δομής των πρωτεϊνών του DSSP*

4.2 Μεθοδολογία επιλογής και προ-επεξεργασία συνόλου δεδομένων

Τα δεδομένα τα οποία χρησιμοποιούνται για την εκπαίδευση των νευρωνικών δικτύων είναι πολύ σημαντικά ώστε το δίκτυο να μπορεί να προβλέπει τα απαιτούμενο αποτέλεσμα σε δεδομένα που δεν χρησιμοποιήθηκαν κατά την εκπαίδευση του. Το ίδιο ισχύει και για το πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών. Τα δεδομένα για την εκπαίδευση του δικτύου αφορούν το σύνολο των πρωτεϊνών που θα επιλεχθούν για την εκπαίδευση και την δοκιμή του δικτύου. Οι πρωτεΐνες που θα επιλεχθούν πρέπει να ανήκουν σε όλο το φάσμα των διαφορετικών πρωτεϊνών ώστε το δίκτυο μετά την εκπαίδευσή του να περιέχει όλα εκείνα τα στοιχεία της σχέσης μεταξύ πρωτοταγούς και δευτεροταγούς δομής που μπορεί να έχει μια πρωτεΐνη ώστε να έχει μεγαλύτερες πιθανότητες γενίκευσης. Συγκεκριμένα οι διαφορετικές αλληλουχίες των αμινοξέων, άρα και οι δυνάμεις που αναπτύσσονται μεταξύ τους (Κεφάλαιο 2), δημιουργούν ένα τεράστιο αριθμό πιθανών αναδιπλώσεων. Για αυτό το λόγο δεν πρέπει να υπάρχει μεγάλο ποσοστό ομοιότητας μεταξύ των αλληλουχιών. Επίσης οι πρωτεΐνες που θα χρησιμοποιηθούν πρέπει να έχουν εξασφαλισμένη την αυθεντικότητά τους.

Με βάση τις πιο πάνω επισημάνσεις χρησιμοποιήθηκαν κάποιοι περιορισμοί ώστε να δημιουργηθεί το κατάλληλο σύνολο δεδομένων για την εκπαίδευση και δοκιμή του δικτύου. Αρχικά, για το λόγο που προαναφέρθηκε, το σύνολο των δεδομένων πρέπει να είναι αντιπροσωπευτικό και γενικά οι πρωτεΐνες πρέπει να έχουν μικρό ποσοστό ομοιότητας διότι σε αντίθετη περίπτωση η δομή των πρωτεϊνών θα είναι όμοια και το δίκτυο θα οδηγείται σε υπερεκπαίδευση. Για την συγκεκριμένη έρευνα, τον περιορισμό αυτό πληρεί το σύνολο δεδομένων του Heinz-Uwe Hobohm (Pdb_Select25 dataset, 20 April 2009), το οποίο διαθέτει 4019 πρωτεϊνικές ακολουθίες των οποίων η ομοιότητα δεν ξεπερνά το 25%. Στο ίδιο αρχείο εκτός από τα ονόματα των πρωτεϊνικών ακολουθιών, υπάρχουν επίσης πληροφορίες για τη μέθοδο που χρησιμοποιήθηκε για την εξαγωγή της τριτοταγούς του δομής, την ποσότητα των αμινοξέων κτλ. Το σύνολο δεδομένων του Heinz-Uwe Hobohm ενημερώνεται σε τακτά χρονικά διαστήματα, με τελευταία ενημέρωση τον Οκτώβριο του 2008. Αντίστοιχο σύνολο δεδομένων χρησιμοποιήθηκε σε ανάλογη έρευνα από τον Baldi (Baldi et al 1999) το 1999 κατά την πρώτη υλοποίηση νευρωνικού δικτύου με αμφίδρομη

ανάδραση, με τη διαφορά όμως ότι το 1999 το σύνολο δεδομένων περιείχε μόνο 824 πρωτεϊνικές ακολουθίες.

Μετά τον εντοπισμό του συνόλου δεδομένων έπρεπε να γίνει κάποια επιπλέον προ-επεξεργασία των δεδομένων ώστε το τελικό σύνολο δεδομένων να έχει όσο το δυνατό μεγαλύτερη ευκρίνεια και μικρότερες πιθανότητες σφάλματος. Για να γίνει επιπλέων σμίκρυνση του συνόλου δεδομένων, ώστε να απορριφθούν πρωτεΐνες η δομή των οποίων δεν είναι προσδιορισμένη σε υψηλή διακριτικότητα, έπρεπε οι τριτοταγείς δομές των πρωτεϊνικών ακολουθιών να έχουν εξαχθεί είτε με τη μέθοδο κρυσταλλογραφίας ακτινών Χ είτε με την μέθοδο πυρηνικού μαγνητικού συντονισμού (NMR), μια πληροφορία η οποία δίνεται μέσα από το σύνολο δεδομένων του Heinz-Uwe Hobohm. Επίσης οι πρωτεϊνικές ακολουθίες πρέπει να βρίσκονται στο σύνολο δεδομένων του προγράμματος DSSP (DSSP, 20 April 2009) αφού με βάση αυτό τυποποιείται η δευτεροταγής δομή τους. Επίσης, οι πρωτεϊνικές ακολουθίες πρέπει να ελεγχθούν ώστε να μην υπάρχουν αμινοξικές ακολουθίες των οποίων η απόσταση των ασύμμετρων ατόμων άνθρακα να είναι μεγαλύτερη των 4.0Å, αφού σε διαφορετική περίπτωση το πιθανότερο είναι να υπάρχει κάποιο σφάλμα και να υπάρχουν αμινοξέα τα οποία να μην εμφανίζονται μέσα από τα πειράματα στον σκελετό της πρωτεΐνης. Τέλος, αλληλένδετο με τον προηγούμενο περιορισμό είναι και το γεγονός ότι η ευκρίνεια που πρέπει να υπάρχει στις δομές από ακτίνες Χ πρέπει να είναι καλύτερη από 1.9Å.

4.3 Δημιουργία του συνόλου δεδομένων

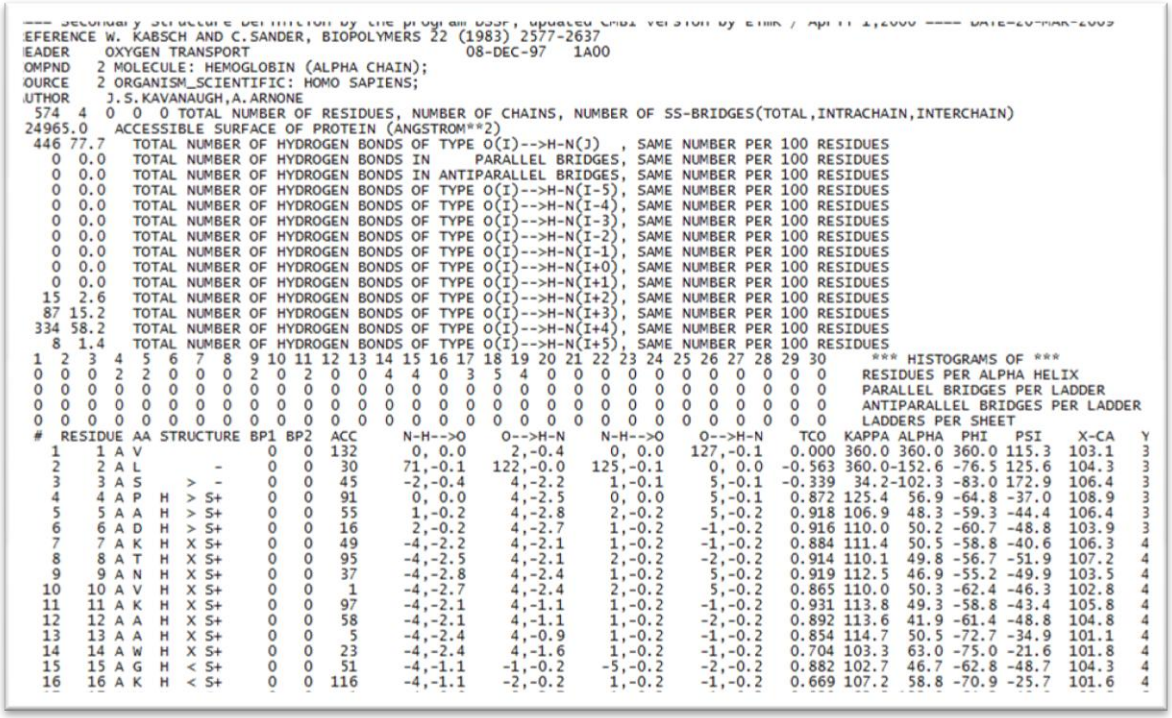
Με βάση τους περιορισμούς και τις πληροφορίες από το Υποκεφάλαιο 4.2 δημιουργήθηκε ένα συγκεκριμένο σύνολο δεδομένων, αρκετά μικρότερο του αρχικού συνόλου δεδομένων του Heinz-Uwe Hobohm. Για τον σκοπό αυτό ακολουθήθηκαν κάποιες διαδικασίες οι οποίες απέρριψαν αρκετές πρωτεϊνικές ακολουθίες ενώ στη συνέχεια δημιουργήθηκαν αρχεία με συγκεκριμένη δομή τα οποία περιέχουν το όνομα, την πρωτοταγή και δευτεροταγή δομή πρωτεϊνικών ακολουθιών ώστε να μπορούν να αναγνωστούν από το νευρωνικό δίκτυο.

Η διαδικασία που ακολουθήθηκε ήταν χρονοβόρα και περιείχε πολλά στάδια μέχρι την τελική δημιουργία του συνόλου δεδομένων. Αρχικά έγινε κάποια έρευνα σχετικά με υπάρχον σύνολα δεδομένων πρωτεϊνικών ακολουθιών τα οποία έπρεπε να έχουν σχετικά μικρό συντελεστή ομοιότητας. Από τα σύνολα δεδομένων που ανεβρέθηκαν χρησιμοποιήθηκε αυτό του Heinz-Uwe Hobohm (Pdb_Select dataset,20 April 2009) από την PDBSELECT (The PDBSELECT database, 20 April 2009) βάση δεδομένων με συντελεστή ομοιότητας 25%. Άλλα σύνολα δεδομένων πρωτεϊνικών ακολουθιών είναι το Evaluation of Automatic Protein Structure Prediction Set (EVA) (Chen and Chaudhari 2007, EVA: PDB sequence unique list 14 May 2009), σύνολα δεδομένων από την βάση δεδομένων ASTRAL (The ASTRAL Compendium for Sequence and Structure Analysis, 20 April 2009) και άλλα σύνολα δεδομένων από τη βάση δεδομένων PDBSELECT (The PDBSELECT database, 20 April 2009).

Ακολούθως, το σύνολο δεδομένων του Heinz-Uwe Hobohm από την PDBSELECT βάση δεδομένων πέρασε μέσα από μια επεξεργασία των εφαρμογών διαδικτύου της PDB βάσης δεδομένων ώστε να διαγραφούν πρωτεΐνες των οποίων των οποίων η απόσταση των διαδοχικών ατόμων άνθρακα να είναι μεγαλύτερη των 4.0Å και η ευκρίνεια είναι από 1.9Å. Αυτή η διαδικασία μπορεί εύκολα να επιτευχθεί δίνοντας όλο το σύνολο δεδομένων στην κατάλληλη εφαρμογή διαδικτύου (An Information Portal to Biological Macromolecular Structures, 20 April 2009) και επιλέγοντας τους ανάλογους περιορισμούς. Στη συνέχεια εξασφαλίστηκαν όλα τα αρχεία πρωτεϊνών του DSSP (DSSP, 20 April 2009) μέσω του ανάλογου διαδικτυακού χώρου. Η ποσότητα των αρχείων αυτών είναι περίπου 56,000 και περιέχουν όλα τα απαραίτητα δεδομένα για την δημιουργία των δεδομένων εισόδου του νευρωνικού δικτύου.

nrsh	ID	naa	Res	Rfac	Methd	n_sid	n_bck	n_naa	n_hlx	n_bta	cmpnd
25	1JXSA	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 amidase
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

Σχήμα 4.1: *Η δομή του αρχείου συνόλου δεδομένων Heinz-Uwe Hobohm.*



```
EFERENCE W. KABSCH AND C.SANDER, BIOPOLYMERS 22 (1983) 2577-2637
EADER    OXYGEN TRANSPORT                        08-DEC-97   1A00
OMPND   2 MOLECULE: HEMOGLOBIN (ALPHA CHAIN);
OURCE   2 ORGANISM_SCIENTIFIC: HOMO SAPIENS;
UTHOR    J.S.KAVANAUGH,A.ARNONE
 574  4  0  0  0 TOTAL NUMBER OF RESIDUES, NUMBER OF CHAINS, NUMBER OF SS-BRIDGES(TOTAL,INTRACHAIN,INTERCHAIN)
24965.0   ACCESSIBLE SURFACE OF PROTEIN (ANGSTROM**2)
 446 77.7    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(J)  , SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS IN     PARALLEL BRIDGES, SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS IN ANTIPARALLEL BRIDGES, SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I-5), SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I-4), SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I-3), SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I-2), SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I-1), SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I+0), SAME NUMBER PER 100 RESIDUES
   0  0.0    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I+1), SAME NUMBER PER 100 RESIDUES
  15  2.6    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I+2), SAME NUMBER PER 100 RESIDUES
  87 15.2    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I+3), SAME NUMBER PER 100 RESIDUES
 334 58.2    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I+4), SAME NUMBER PER 100 RESIDUES
   8  1.4    TOTAL NUMBER OF HYDROGEN BONDS OF TYPE O(I)-->H-N(I+5), SAME NUMBER PER 100 RESIDUES
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30      *** HISTOGRAMS OF ***
 0  0  0  2  2  0  0  0  2  0  2  0  0  4  4  0  3  5  4  0  0  0  0  0  0  0  0  0  0  0     RESIDUES PER ALPHA HELIX
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0     PARALLEL BRIDGES PER LADDER
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0     ANTIPARALLEL BRIDGES PER LADDER
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0     LADDERS PER SHEET
  #  RESIDUE AA STRUCTURE BP1 BP2  ACC     N-H-->O    O-->H-N    N-H-->O    O-->H-N    TCO  KAPPA ALPHA  PHI   PSI    X-CA   Y
    1    1 A V              0   0  132      0, 0.0     2,-0.4     0, 0.0   127,-0.1   0.000 360.0 360.0 360.0 115.3  103.1   3
    2    2 A L              0   0   30     71,-0.1   122,-0.0   125,-0.1     0, 0.0  -0.563 360.0-152.6 -76.5 125.6  104.3   3
    3    3 A S    >  -      0   0   45     -2,-0.4     4,-2.2     1,-0.1     5,-0.1   -0.339  34.2-102.3 -83.0 172.9  106.4   3
    4    4 A P  H  > S+     0   0   91      0, 0.0     4,-2.5     0, 0.0     5,-0.1    0.872 125.4  56.9 -64.8 -37.0  108.9   3
    5    5 A A  H  > S+     0   0   55      1,-0.2     4,-2.8     2,-0.2     5,-0.2    0.918 106.9  48.3 -59.3 -44.4  106.4   3
    6    6 A D  H  > S+     0   0   16      2,-0.2     4,-2.7     1,-0.2    -1,-0.2    0.916 110.0  50.2 -60.7 -48.8  103.9   3
    7    7 A K  H  X S+     0   0   49     -4,-2.2     4,-2.1     1,-0.2    -1,-0.2    0.884 111.4  50.5 -58.8 -40.6  106.3   4
    8    8 A T  H  X S+     0   0   95     -4,-2.5     4,-2.1     2,-0.2    -2,-0.2    0.914 110.1  49.8 -56.7 -51.9  107.2   4
    9    9 A N  H  X S+     0   0   37     -4,-2.8     4,-2.4     1,-0.2     5,-0.2    0.919 112.5  46.9 -55.2 -49.9  103.5   4
   10   10 A V  H  X S+     0   0    1     -4,-2.7     4,-2.4     2,-0.2     5,-0.2    0.865 110.0  50.3 -62.4 -46.3  102.8   4
   11   11 A K  H  X S+     0   0   97     -4,-2.1     4,-1.1     1,-0.2    -1,-0.2    0.931 113.8  49.3 -58.8 -43.4  105.8   4
   12   12 A A  H  X S+     0   0   58     -4,-2.1     4,-1.1     1,-0.2    -2,-0.2    0.892 113.6  41.9 -61.4 -48.8  104.8   4
   13   13 A A  H  X S+     0   0    5     -4,-2.4     4,-0.9     1,-0.2    -1,-0.2    0.854 114.7  50.5 -72.7 -34.9  101.1   4
   14   14 A W  H  X S+     0   0   23     -4,-2.4     4,-1.6     1,-0.2    -1,-0.2    0.704 103.3  63.0 -75.0 -21.6  101.8   4
   15   15 A G  H  < S+     0   0   51     -4,-1.1    -1,-0.2    -5,-0.2    -2,-0.2    0.882 102.7  46.7 -62.8 -48.7  104.3   4
   16   16 A K  H  < S+     0   0  116     -4,-1.1     -2,-0.2     1,-0.2    -1,-0.2    0.669 107.2  58.8 -70.9 -25.7  101.6   4
```

Σχήμα 4.2: *Η έξοδος του προγράμματος DSSP για κάθε πρωτεΐνη.*

Για την δημιουργία των δεδομένων εισόδου του νευρωνικού δικτύου δημιουργήθηκε ένας αναλυτής δεδομένων (Parser) (Παράρτημα Β) ο οποίος δέχεται στην είσοδο ένα αρχείο που περιέχει όλες τις πρωτεϊνικές ακολουθίες που απέμειναν στο σύνολο δεδομένων μετά την προ-επεξεργασία και όλα τα αρχεία του DSSP δίνοντας στην έξοδο τα αρχεία αυτά. Ο αναλυτής δεδομένων αρχικά διαβάζει το σύνολο δεδομένων και με βάση την δομή του αρχείου του Heinz-Uwe Hobohm (Σχήμα 4.1) επιλέγει μόνο τις πρωτεΐνες στις οποίες η έκτη στήλη έχει το γράμμα X ή N τα οποία αντιπροσωπεύουν τις μεθόδους κρυσταλλογραφία ακτινών X και πυρηνικού μαγνητικού συντονισμού (NMR), αντίστοιχα. Εδώ αξίζει να σημειωθεί ότι τα πρώτα τέσσερα γράμματα του ονόματος, κάθε πρωτεϊνικής ακολουθίας, συμβολίζουν το όνομά της πρωτεΐνης. Το πέμπτο γράμμα δηλώνει ποια από τις πολύπεπτιδικές αλυσίδες είναι η συγκεκριμένη, αφού μια πρωτεΐνη μπορεί να αποτελείται από πολλές πρωτεϊνικές ακολουθίες. Ακολούθως, το πρόγραμμα διαβάζει τα ονόματα των πρωτεϊνών ένα προς ένα και αφού εντοπίσει το κάθε ένα παίρνει από αυτό τα απαραίτητα δεδομένα. Το πρόγραμμα, για κάθε πρωτεϊνική ακολουθία, διαβάζει από το

αντίστοιχο αρχείο του DSSP (Σχήμα 4.2) την πρωτοταγή και δευτεροταγή δομή μόνο της πρωτεϊνικής ακολουθίας που αναγράφεται στο αντίστοιχο όνομα, το οποίο και αναπαριστάται από την τρίτη στήλη. Η πρωτοταγής και δευτεροταγής δομή εκλαμβάνονται από την τέταρτη και πέμπτη στήλη, αντίστοιχα. Η δεύτερη στήλη αντιπροσωπεύει την θέση του αμινοξέως στην αμινοξική ακολουθία. Σε αρκετές περιπτώσεις υπάρχουν ελλιπή δεδομένα από την άποψη ότι μπορεί να απουσιάζουν αμινοξέα ή στοιχεία της πρωτοταγούς και δευτεροταγούς δομής. Σε αυτές τις περιπτώσεις οι θέσεις αυτές αναπαριστώνται με το γράμμα Χ. Τέλος, στο αρχείο εξόδου του προγράμματος και αντίστοιχα αρχείο εισόδου του νευρωνικού δικτύου (Παράρτημα Γ), αναγράφονται όλα τα απαραίτητα δεδομένα. Συγκεκριμένα, για κάθε πρωτεΐνη αναγράφεται το όνομά της, στην αμέσως επόμενη γραμμή που τελειώνει με '.' αναγράφεται η πρωτοταγής της δομή και στην αμέσως επόμενη γραμμή που επίσης τελειώνει με '.' αναγράφεται η δευτεροταγής της δομή. Σημαντικό στοιχείο είναι το γεγονός ότι αρχεία τα οποία δεν υπάρχουν στο DSSP απορρίπτονται.

4.4 Ελλιπής δεδομένα και σύνολα δεδομένων εισόδου

Η τελευταία διαδικασία που ακολουθήθηκε για την δημιουργία των δεδομένων εισόδου του νευρωνικού δικτύου ήταν η δημιουργία συγκεκριμένων συνόλων δεδομένων για τα διάφορα πειράματα. Τα σύνολα δεδομένων δημιουργήθηκαν παίρνοντας τυχαία πρωτεϊνικές ακολουθίες από το σύνολο δεδομένων της προ-επεξεργασίας (Υποκεφάλαιο 4.3). Συγκεκριμένα δημιουργήθηκαν τρία διαφορετικά σύνολα δεδομένων των 100 πρωτεϊνικών ακολουθιών για εκπαίδευση του δικτύου με αντίστοιχα τρία διαφορετικά σύνολα δεδομένων των 30 πρωτεϊνικών ακολουθιών για έλεγχο του δικτύου. Τα σύνολα αυτά περιέχουν πρωτεϊνικές ακολουθίες οι οποίες περιέχουν ελλιπής δεδομένα στην πρωτοταγή και δευτεροταγή τους δομή, δηλαδή περιέχουν την τιμή Χ όπου απουσιάζουν δεδομένα από την πρωτοταγή δομή και την τιμή της γενικής κατηγορία για τη δευτεροταγή δομή. Επίσης δημιουργήθηκε ένα σύνολο δεδομένων των 100 πρωτεϊνικών ακολουθιών για εκπαίδευση του δικτύου με ένα αντίστοιχο σύνολα δεδομένων των 30 πρωτεϊνικών

ακολουθιών τα οποία αντικαθιστούν τα ελλιπείς δεδομένα της πρωτοταγής δομής με τυχαίες τιμές του συνόλου των τιμών που μπορεί να έχει η πρωτοταγής και δευτεροταγή δομή, αντίστοιχα. Τέλος, δημιουργήθηκαν σύνολα δεδομένων στα οποία η τυχαία επιλογή των πρωτεϊνών αφορούσε μόνο πρωτεΐνες οι οποίες δεν περιέχουν ελλιπής δεδομένα στην πρωτοταγή δομή τους. Με αυτό τον περιορισμό δημιουργήθηκε ένα σύνολο δεδομένων των 500 πρωτεϊνικών ακολουθιών για εκπαίδευση και 125 πρωτεϊνικών ακολουθιών για δοκιμή του δικτύου, καθώς και ένα σύνολο δεδομένων των 600 πρωτεϊνικών ακολουθιών για εκπαίδευση και 200 πρωτεϊνικών ακολουθιών για δομική του δικτύου. Από τα σύνολα δεδομένων που προαναφέρθηκαν, απουσιάζουν πρωτεϊνικές ακολουθίες οι οποίες στην πρωτοταγή τους δομή περιέχουν στην θέση των αμινοξέων γράμματα τα οποία αναπαριστούν το αμινοξύ κυστεϊνη.

Επιπλέον σύνολα δεδομένων μπορούν να δημιουργηθούν σε μελλοντικό στάδιο. Μπορούν να δημιουργηθούν σύνολα δεδομένων με πολύ μεγαλύτερο αριθμό πρωτεϊνικών ακολουθιών για εκπαίδευση του δικτύου. Μπορούν επίσης να δημιουργηθούν σύνολα δεδομένων στα οποία να υπάρχουν πρωτεϊνικές ακολουθίες στις οποίες τα γράμματα τα οποία αναπαριστούν το αμινοξύ κυστεϊνη να αντικατασταθούν με το σύμβολο C που αναπαριστά την κυστεϊνη. Μια ακόμα πρόταση για επεξεργασία των δεδομένων των συνόλων δεδομένων είναι η δημιουργία συνόλων δεδομένων τα οποία δεν θα έχουν ελλιπή δεδομένα στην πρωτοταγή δομή, ενώ τα ελλιπή δεδομένα της δευτεροταγής δομής θα αντικαθιστώνται με τιμές των γειτονικών ακολουθιών δευτεροταγούς δομής, όχι τυχαίες τιμές, αφού η δευτεροταγής δομή ενός αμινοξέως έχει σχέση με τη δευτεροταγή δομή των γειτονικών του αμινοξέων (Chen and Chaudhari 2007).

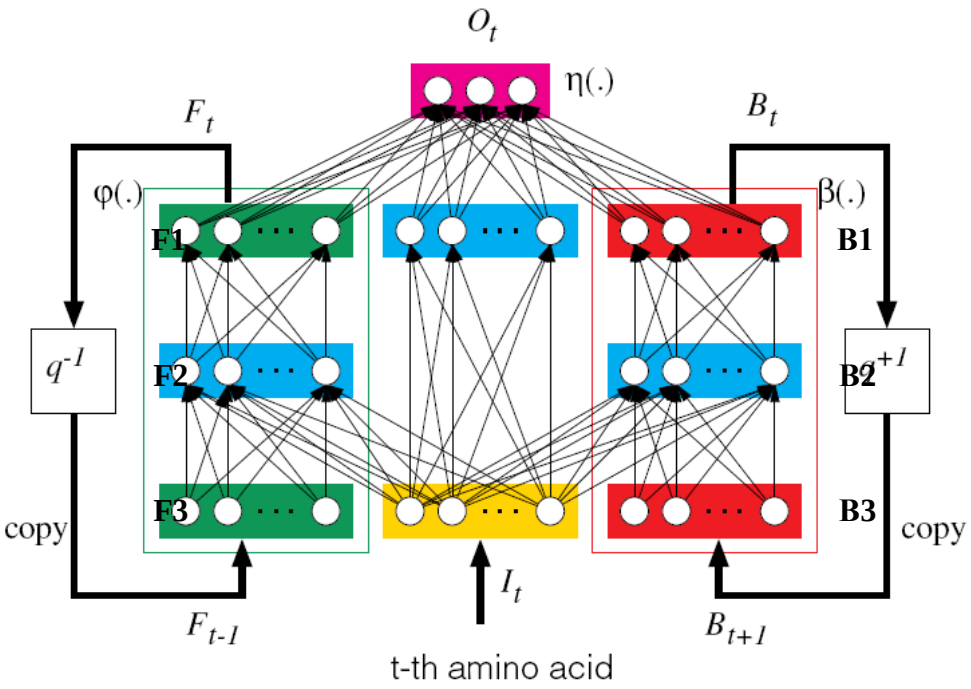
Κεφάλαιο 5

Περιγραφή συστήματος

5.1 Επιλογή του κατάλληλου νευρωνικού δικτύου

Το πρόβλημα της πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, όπως προαναφέρθηκε (Υποκεφάλαιο 1.1), μπορεί να προσεγγιστεί με νευρωνικά δίκτυα. Το πρόβλημα αυτό ανήκει στην κατηγορία των προβλημάτων κατηγοριοποίησης. Τα χαρακτηριστικά τα οποία δίνουν τη δυνατότητα σε ένα νευρωνικό δίκτυο να ομαδοποιεί και να κατηγοριοποιεί τα δεδομένα που δέχεται στην είσοδο σε συγκεκριμένες κατηγορίες επίσης αναφέρονται σε προηγούμενο κεφάλαιο (Κεφάλαιο 3).

Το δίκτυο πρέπει να δέχεται στην είσοδο του ένα αμινοξύ μαζί με όσα στοιχεία χρειάζεται ώστε να δίνει στην έξοδό του το είδος της δευτεροταγούς δομής. Η είσοδος του δικτύου, λόγω της φύσης του προβλήματος, πρέπει να αποτελείται από μια σειρά από αμινοξέα τα οποία θα αντιπροσωπεύουν ένα συγκεκριμένο αμινοξύ, το οποίο θα βρίσκεται στο κέντρο του παραθύρου εισόδου, μαζί με τα αμινοξέα που προηγούνται και έπονται αυτού στην πρωτεϊνική ακολουθία. Αυτά τα δεδομένα είναι απαραίτητα για την ανάθεση του αμινοξέως σε κάποια κατηγορία δευτεροταγούς δομής, διότι η δευτεροταγής δομή σε κάθε σημείο της πρωτεΐνης εξαρτάται κυρίως από τις δυνάμεις και τους δεσμούς που αναπτύσσονται μεταξύ γειτονικών αμινοξέων στον τρισδιάστατο χώρο της πρωτεΐνης. Επομένως η δευτεροταγής δομή που θα έχει ένα σημείο-αμινιξί εξαρτάται αποκλειστικά από την ακολουθία των αμινοξέων, και συγκεκριμένα από τα αμινοξέα που προηγούνται και έπονται αυτού. Με βάση τα δεδομένα αυτά οδηγούμαστε στο συμπέρασμα ότι χρειάζεται να υλοποιηθεί ένα ΤΝΔ με αμφίδρομη ανάδραση (Σχήμα 5.1), το οποίο στην ουσία είναι ένας συνδυασμός δυο κλασικών ΤΝΔ με ανάδραση (Baldi et al 2000). Το πρώτο ΤΝΔ με ανάδραση δέχεται στην είσοδό του την ακολουθία των αμινοξέων που προηγούνται του αμινοξέως που εξετάζουμε την δευτεροταγή δομή του. Αντίστοιχα το δεύτερο ΤΝΔ με ανάδραση δέχεται στην είσοδό του την ακολουθία των αμινοξέων που έπονται του αμινοξέως αυτού. Τα δυο αυτά δίκτυα δημιουργούν ένα είδος μνήμης με το οποίο συσχετίζουν το κάθε ένα ξεχωριστά τις δυο ακολουθίες. Η έξοδος των δυο αυτών δικτύων, μαζί με το εξεταζόμενο αμινοξύ συσχετίζονται και δίνουν στην έξοδο του νευρωνικού δικτύου αμφίδρομης ανάδρασης την κατηγορία δευτεροταγούς δομής του αμινοξέως.



Σχήμα 5.1: *Το νευρωνικό δίκτυο αμφίδρομης ανάδρασης του Baldi (1999)*

5.2 Νευρωνικό δίκτυο αμφίδρομης ανάδρασης με βάση το PSSP

Το νευρωνικό δίκτυο αμφίδρομης ανάδρασης (BRNN), το οποίο υλοποιείται με σκοπό να λύσει το πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, αποτελείται από δυο κλασικά ΤΝΔ με ανάδραση και ένα νευρωνικό δίκτυο εμπρόσθιου περάσματος (feed-forward) (Baldi et al 1999,Baldi et al 2000). Η δομή του δικτύου αυτού παρουσιάζεται στο Σχήμα 5.1 με το F (Forward) και B (Backward) να συμβολίζουν τα δύο ΤΝΔ με ανάδραση και N το νευρωνικό δίκτυο εμπρόσθιου περάσματος. Το κάθε ένα από τα δυο ΤΝΔ με ανάδραση, λειτουργεί αυτόνομα με βάση τα δικά του επίπεδα, δίνοντας την έξοδό του στο επίπεδο εξόδου. Αυτό με τη σειρά του δέχεται επίσης την έξοδο του νευρωνικού δικτύου εμπρόσθιου περάσματος N και ανάλογα αποφασίζει την έξοδο του δικτύου (Εξίσωση 5.1) η οποία και αντιπροσωπεύει τη δευτεροταγή δομή της πρωτεΐνης.

$$O_t = n(F_t, I_t, B_t)$$

Εξίσωση 5.1: *Έξοδος νευρωνικού δικτύου, όπου n() το ΤΝΔ N, F_t η ακολουθία των αμινοξέων jπριν το αμινοξύ στην θέση t, I_t η είσοδος του δικτύου όταν εξετάζεται το αμινοξύ στην θέση t και B_t η ακολουθία των αμινοξέων μετά το αμινοξύ στην θέση t.*

Το δίκτυο αμφίδρομης ανάδρασης έχει σαν είσοδο κάθε στιγμή t, όπως προαναφέραμε (Υποκεφάλαιο 5.1), μια ακολουθία από αμινοξέα τα οποία αντιπροσωπεύουν ένα συγκεκριμένο αμινοξύ που πάντα βρίσκεται στο κέντρο ενός παραθύρου εισόδου μαζί με τα αμινοξέα που προηγούνται και έπονται αυτού στην πρωτεϊνική ακολουθία. Η στιγμή t αντιπροσωπεύει την θέση του αμινοξέως, του οποίου εξετάζουμε την δευτεροταγή δομή του, στην πρωτεϊνική ακολουθία μεγέθους 0 μέχρι T. Το παράθυρο εισόδου έχει σταθερό μέγεθος με κέντρο πάντοτε το t ($0 \leq t \leq T$) και συγκεκριμένο αριθμό αμινοξέων πριν και μετά από αυτό. Βάση της αρχιτεκτονικής του ΤΝΔ αμφίδρομης ανάδρασης για PSSP (Baldi et al 1999) τα δεδομένα που προηγούνται και έπονται του t πρέπει να περάσουν μέσα από δυο μη γραμμικές συναρτήσεις φ() (Εξίσωση 5.2) και β()(Εξίσωση 5.3). Τις συναρτήσεις αυτές αντιπροσωπεύουν τα δυο ΤΝΔ με ανάδραση. Το F ΤΝΔ με ανάδραση δέχεται στην είσοδό του την ακολουθία των αμινοξέων που βρίσκονται στο παράθυρο εισόδου και προηγούνται του αμινοξέως t με σειρά από αριστερά προς δεξιά (Εξίσωση 5.2). Αντίστοιχα, το Β ΤΝΔ με ανάδραση δέχεται στην είσοδό του την ακολουθία των αμινοξέων που βρίσκονται στο παράθυρο εισόδου και έπονται του αμινοξέως t με σειρά από δεξιά προς αριστερά (Εξίσωση 5.3). Τα δυο αυτά δίκτυα δημιουργούν ξεχωριστά ένα είδος μνήμης με το οποίο συσχετίζουν τα συνεχόμενα αμινοξέα της κάθε ακολουθίας (Elman 1990). Ο τρόπος με τον οποίο εισέρχονται σε κάθε ένα από τα δυο δίκτυα τα δεδομένων μπορεί να είναι είτε ένα προς ένα αμινοξύ, είτε ως μια ακολουθία αμινοξέων η οποία έχει σταθερό μέγεθος. Το ξεδιπλωμένο δίκτυο στο χρόνο το οποίο παρουσιάζει τις διαδοχικές εισόδους παρουσιάζεται στο Σχήμα 5.2.

$$F_t = (F_{t-1}, I_t)$$

Εξίσωση 5.2: *Είσοδος Forward νευρωνικού δικτύου με ανάδραση, όπου F_{t-1} η έξοδος του δικτύου την συγκεκριμένη χρονική στιγμή και I_t η νέα είσοδος του δικτύου.*

$$B_t = (B_{t+1}, I_t)$$

Εξίσωση 5.3: *Είσοδος Backward νευρωνικού δικτύου με ανάδραση, όπου B_{t-1} η έξοδος του δικτύου την συγκεκριμένη χρονική στιγμή και I_t η νέα είσοδος του δικτύου.*

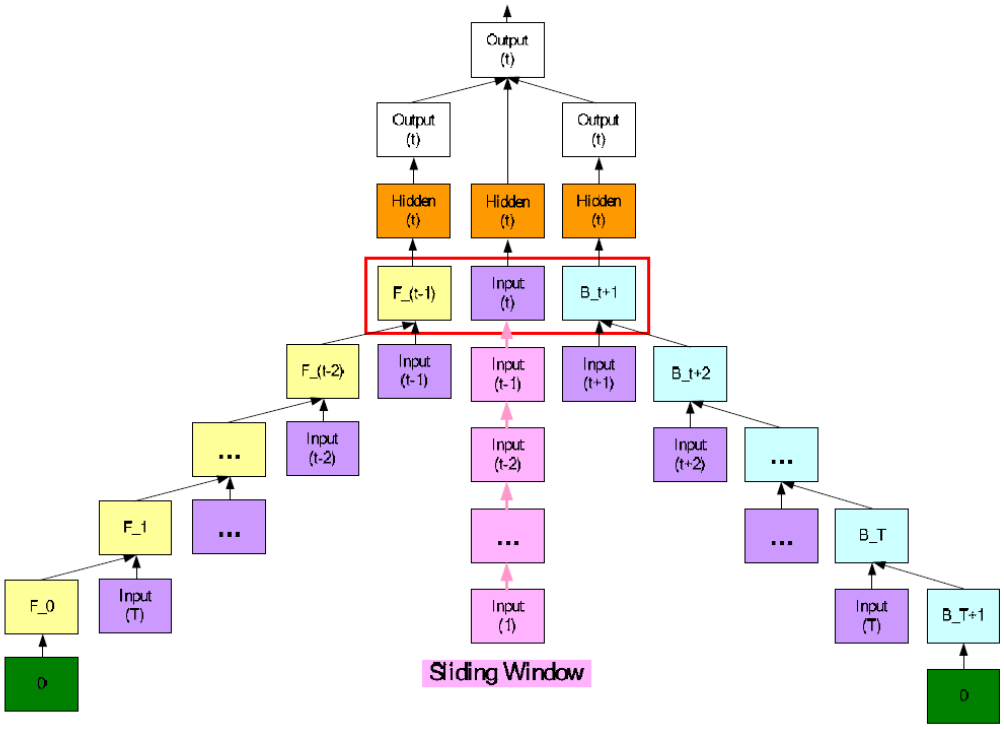
Ο αριθμός των αμινοξέων που εισέρχονται σε κάθε χρονική στιγμή μαζί με το είδος της κωδικοποίησης του κάθε αμινοξέως καθορίζει το μέγεθος k του διανύσματος εισόδου του δικτύου. Αντίστοιχα το μέγεθος του διανύσματος εισόδου το κάθε ΤΝΔ με ανάδραση καθορίζεται από το μέγεθος το context επιπέδου που έχει το κάθε ένα (Υποκεφάλαιο 3.2.3). Αν το μέγεθος του context επιπέδου του δικτύου F είναι n τότε το διάνυσμα εισόδου του δικτύου είναι n+k κάθε χρονική στιγμή μέχρι η είσοδός του να είναι η είσοδος με το αμινοξύ t. Αντίστοιχα, αν το μέγεθος του context επιπέδου του δικτύου B είναι m τότε αυτό θα έχει διάνυσμα εισόδου μεγέθους m+k κάθε χρονική στιγμή μέχρι η είσοδός του να είναι η είσοδος με το αμινοξύ t. Με την ίδια λογική, το διάνυσμα εξόδου ολόκληρου του δικτύου εξαρτάται από την ποσότητα των κατηγοριών δευτεροταγούς δομής που καθορίζουμε και την κωδικοποίηση των κατηγοριών αυτών. Όλες οι πιο πάνω πληροφορίες, καθώς και ο αριθμός των κόμβων του κάθε επιπέδου καθορίζουν τον αριθμό των ελεύθερων παραμέτρων του δικτύου.

$$b_{i,t} = \sigma\left(\sum_{j=1}^{N_\beta} \omega_{i,j}\, h_{j,t}\right) \; i = 1, \dots, n$$

Εξίσωση 5.4: *Εξίσωση εισόδου κάθε νευρώνα του δικτύου (εκτός των νευρώνων κρυφών επιπέδων του κάθε νευρωνικού δικτύου με ανάδραση), όπου $h_{j,t}$ είναι η έξοδος νευρώνων που προηγούνται και $\omega_{i,j}$ το βάρος μεταξύ του νευρώνα i και j.*

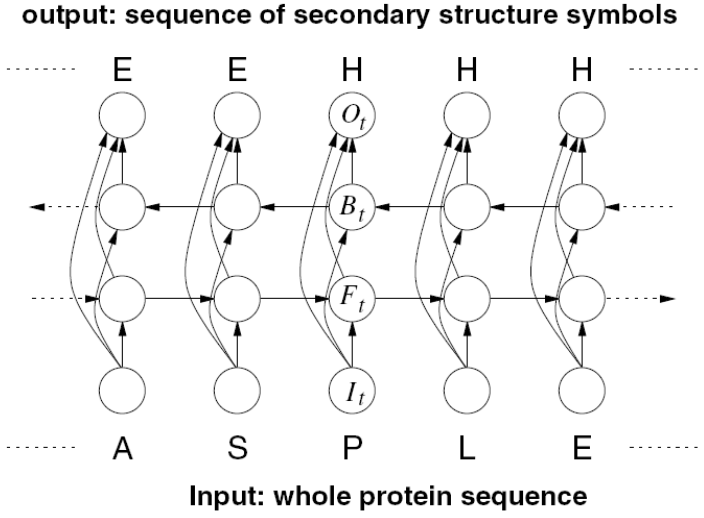
$$h_{j,t} = \sigma\left(\sum_{l=1}^{N_\beta} \omega_{j,l} b_{l,t+1} + \sum_{l=1}^{k} v_{j,l} u_{l,t}\right) \quad j = 1, \ldots, N_\beta$$

Εξίσωση 5.5: *Εξίσωση εισόδου των νευρώνων κρυφών επιπέδων του κάθε νευρωνικού δικτύου με ανάδραση, όπου $b_{l,t+1}$ είναι η έξοδος νευρώνων εξόδου του νευρωνικού δικτύου με ανάδραση, $\omega_{j,l}$ το βάρος μεταξύ του νευρώνα j και $i, u_{l,t}$ η νέα είσοδος του νευρωνικού δικτύου με ανάδραση την χρονική στιγμή t και $v_{j,l}$ το βάρος μεταξύ του νευρώνα j και του νευρώνα εισόδου i.*



Σχήμα 5.2: *Το νευρωνικό δίκτυο αμφίδρομης ανάδρασης ξεδιπλωμένο στο χρόνο.*

Το εμπρόσθιο πέρασμα των δεδομένων από το δίκτυο καθορίζεται από τις εξισώσεις 5.4 και 5.5. Συγκεκριμένα, με βάση το Σχήμα 5.1, η έξοδος κάθε κόμβου ο οποίος ανήκει στα ενεργά επίπεδα F1, B1 και τα επίπεδα του κεντρικού δικτύου εμπρόσθιου περάσματος αντιπροσωπεύεται από την Εξίσωση 5.4, ενώ η έξοδος κάθε κόμβου ο οποίος ανήκει στα ενεργά επίπεδα F2 και B2 αντιπροσωπεύεται από την Εξίσωση 5.5. Οι κόμβοι που βρίσκονται στα ενεργά επίπεδα του δικτύου χρησιμοποιούν την σιγμοειδή συνάρτηση (Εξίσωση 3.2) ως συνάρτηση ενεργοποίησης. Τέλος, τα context επίπεδα F3 και B3 έχουν μια ένα προς ένα και επί σχέση με τα επίπεδα F1 και B1, αντίστοιχα, δίνοντας στην έξοδό τους το αποτέλεσμα εξόδου των επιπέδων F3 και B3 πολλαπλασιαζόμενο με μια τιμή q. Το δίκτυο με βάση τις εξισώσεις αυτές και αφού δεχτεί μια ακολουθία αμινοξέων δίνει στην έξοδο την κατηγορία δευτεροταγούς δομής που αντιπροσωπεύει. Οι αλληλεξαρτήσεις των αμινοξέων τα οποία μέσα από το δίκτυο οδηγούν σε μια συγκεκριμένη δευτεροταγή δομή παρουσιάζονται στο Σχήμα 5.3. Στο συγκεκριμένο σχήμα παρουσιάζεται η εξάρτηση της κάθε μορφής δευτεροταγούς δομής από το κάθε αμινοξύ στη θέση t σε συνάρτηση με τα αμινοξέα που προηγούνται αυτού και βρίσκονται στην μνήμη του F μέχρι και τη θέση t, καθώς και τα αμινοξέα που έπονται αυτού και βρίσκονται αποθηκευμένα στην μνήμη του B από τη θέση t.



Σχήμα 5.3: *Η σχέση που έχει η έξοδος του δικτύου με τα διάφορα αμινοξέα.(Baldi 1999)*

Αφού περάσουν τα δεδομένα εισόδου κάθε παραθύρου στο δίκτυο υπολογίζεται το σφάλμα του δικτύου με βάση την εξίσωση LMS (Εξίσωση 3.4) και ακολούθως το σφάλμα χρησιμοποιείται από τον αλγόριθμο εκμάθησης ανάστροφης μετάδοσης λάθους για να τροποποιηθούν τα βάρη και τα κατώφλια των κόμβων του δικτύου. Ο αλγόριθμος εκμάθηση ανάστροφης μετάδοσης λάθους δια μέσω χρόνου (Υποκεφάλαιο 3.3.3) χρησιμοποιείται για την εκπαίδευση κλασικών ΤΝΔ με ανάδραση. Ο αλγόριθμος αυτός αθροίζει το λάθος κάθε χρονικής στιγμής της ανάδρασης με βάση το επιθυμητό αποτέλεσμα εξόδου σε κάθε χρονική στιγμή. Λόγω της φύσης του προβλήματος PSSP, δεν είναι εφικτή η χρήση του συγκεκριμένου αλγόριθμου για την εκπαίδευση του κάθε ενός ΤΝΔ με ανάδραση στο BRNN. Ο λόγος για αυτό είναι διότι κατά την συνεχόμενη είσοδο μιας πρωτεϊνικής ακολουθίας στο ΤΝΔ με ανάδραση δεν έχουμε κάποιο επιθυμητό αποτέλεσμα. Το μόνο επιθυμητό αποτέλεσμα που υπάρχει στο BRNN είναι η δευτεροταγής δομή κάποιου αμινοξέως η οποία προκύπτει από το συνολικό πέρασμα ενός παραθύρου εισόδου στο δίκτυο. Με βάση αυτή την επισήμανση, το BRNN εκπαιδεύεται με τον αλγόριθμο εκμάθησης ανάστροφης μετάδοσης λάθους που περιγράφεται στο υποκεφάλαιο 3.3.2. Αφού υπολογιστεί το σφάλμα για κάθε βάρος τότε αυτά διορθώνονται με τον κανόνα δέλτα. Αν τα βάρη τα οποία πρέπει να διορθωθούν συνδέουν τα επίπεδα F1 και F2 (Σχήμα 5.1) χρησιμοποιείται η Εξίσωση 5.7, αντίστοιχα αν τα επίπεδα είναι το B1 και B2 (Σχήμα 5.1) χρησιμοποιείται η Εξίσωση 5.8, ενώ για τα υπόλοιπα βάρη χρησιμοποιείται η Εξίσωση 5.6. Απώτερος σκοπό της διαδικασίας αυτής είναι όταν περάσει ένας αριθμός πρωτεϊνικών ακολουθιών από το δίκτυο αυτό να εκπαιδευτεί και να μπορεί να προβλέπει την δευτεροταγή δομή ακολουθιών οι οποίες δεν παρουσιάστηκαν στο νευρωνικό δίκτυο κατά την διαδικασία εκμάθησης.

$$\Delta\omega = \sum_{t=1}^{T} \delta_{i,t} x_{j,t}$$

Εξίσωση 5.6: *Κανόνας Δ για διόρθωση των όλων των βαρών του δικτύου την χρονική στιγμή t(εκτός αυτών που συνδέουν τα context επίπεδα των νευρωνικών δικτύων με ανάδραση με τα αντίστοιχα κρυφά επίπεδα), όπου* $\delta_{i,t}$ *το σφάλμα διόρθωσης από τον νευρώνα ι του επόμενου επιπέδου και* $x_{j,t}$ *η είσοδος του νευρώνα j.*

$$\Delta\omega = \sum_{t=1}^{T} \delta_{i,t} b_{j,t+1}$$

Εξίσωση 5.7: *Κανόνας Δ για διόρθωση των βαρών, την χρονική στιγμή t, που συνδέουν το context επίπεδο του Backward νευρωνικού δικτύου με ανάδραση με το αντίστοιχο κρυφό επίπεδο, όπου* $\delta_{i,t}$ *το σφάλμα διόρθωσης από τον νευρώνα ι του επόμενου επιπέδου και* $b_{j,t+1}$ *η είσοδος του νευρώνα j του κρυφού επιπέδου από το context επίπεδο.*

$$\Delta\omega = \sum_{t=1}^{T} \delta_{i,t} f_{j,t-1}$$

Εξίσωση 5.8: *Κανόνας Δ για διόρθωση των βαρών, την χρονική στιγμή t, που συνδέουν το context επίπεδο του Forward νευρωνικού δικτύου με ανάδραση με το αντίστοιχο κρυφό επίπεδο, όπου* $\delta_{i,t}$ *το σφάλμα διόρθωσης από τον νευρώνα ι του επόμενου επιπέδου και* $f_{j,t-1}$ *η είσοδος του νευρώνα j του κρυφού επιπέδου από το context επίπεδο.*

5.3 Σχεδιασμός και Υλοποίηση νευρωνικού δικτύου με αμφίδρομη ανάδραση

5.3.1 Σχεδιασμός και Υλοποίηση σε επίπεδο κώδικα

Ο σχεδιασμός και η υλοποίηση του νευρωνικού δικτύου αμφίδρομης ανάδρασης λάθους (BRNN) έγινε με βάση το Υποκεφάλαιο 5.2. Αναπτύχθηκαν μια σειρά από κλάσεις οι οποίες με τις ανάλογες μεθόδους μοντελοποιούν το συγκεκριμένο δίκτυο (Σχήμα 5.6). Οι κλάσεις αυτές σχεδιάστηκαν και υλοποιήθηκαν με βάση όλες τις αρχές προγραμματισμού ώστε να ελέγχονται τα δεδομένα, να είναι αποδοτικές και να έχουν χαμηλά ή και ανύπαρκτα ποσοστά λάθους. Συνδέονται μεταξύ τους ώστε να διαβάζουν πρωτεϊνικές

ακολουθίες, να εκπαιδεύουν το δίκτυο και να δημιουργούν μια σειρά από αρχεία με τα ανάλογα αποτελέσματα. Πιο συγκεκριμένα οι κλάσεις αναπτύχθηκαν στην γλώσσα προγραμματισμού C++, η οποία ενδείκνυται για εφαρμογές νευρωνικών δικτύων αφού διαθέτει τα στοιχεία μιας αντικειμενοστραφούς αλλά και εξαιρετικά γρήγορης γλώσσας προγραμματισμού. Οι κλάσεις που αναπτύχθηκαν είναι η BidirectionalRecurrentNeuralNetwork.cpp, η DataReader.cpp, η Neuron.cpp, η OutputData.cpp, η PSSP_UCY.cpp και η Services.cpp.

Η κλάση PSSP_UCY αποτελεί την κυρίως κλάση του προγράμματος. Η κλάση αυτή χρησιμοποιείται για να οργανώνει τις λειτουργίες του δικτύου. Αφού διαβάσει τις παραμέτρους του δικτύου από το κατάλληλο αρχείο (Παράρτημα Δ) μέσω ενός αντικειμένου της κλάσης DataReader, τότε δημιουργεί το δίκτυο με ένα κατασκευαστή της κλάσης BidirectionalRecurrentNeuralNetwork, η οποία δέχεται όλες τις παραμέτρους του δικτύου. Στη συνέχεια ακολουθεί ο βρόγχος με τις επαναλήψεις των δεδομένων στο δίκτυο. Ο βρόγχος αυτός χωρίζεται σε δυο μικρότερους βρόγχους οι οποίοι εκτελούν την εκπαίδευση και δοκιμή του δικτύου, αντίστοιχα. Οι λειτουργίες αυτές εκτελούνται με διάφορες μεθόδους του αντικειμένου της κλάσης BidirectionalRecurrentNeuralNetwork. Μετά την ολοκλήρωση όλων των επαναλήψεων, χρησιμοποιείται ένα αντικείμενο της κλάσης OutputData το οποίο αναλαμβάνει να εκτυπώσει όλα τα αποτελέσματα στα ανάλογα αρχεία.

Ο κάθε κόμβος-Νευρώνας του δικτύου υλοποιείται με την κλάση Neuron. Η κλάση Neuron διαθέτει μεθόδους οι οποίες μπορούν να υλοποιήσουν όλες τις λειτουργίες ενός νευρώνα. Με αυτό τον τρόπο δημιουργείται μια αφαιρετικότητα μεταξύ τον κόμβων του δικτύου η οποία βοηθά στην υλοποίηση του δικτύου, αφού ο κάθε νευρώνας δέχεται απλά εντολές και τις εκτελεί.

Ο κάθε νευρώνας διαθέτει ένα διάνυσμα στο οποίο είναι αποθηκευμένες όλες οι τιμές των βαρών που είναι συνδεδεμένες στη είσοδο του. Οι λειτουργίες που μπορεί να εκτελέσει κάποιος νευρώνας είναι η είσοδος που έχει την κάθε χρονική στιγμή, το αποτέλεσμα της συνάρτησης ενεργοποίησης, ο υπολογισμός του σφάλματος σε περίπτωση που ανήκει σε επίπεδο εξόδου, ο υπολογισμός του σφάλματος σε περίπτωση που ανήκει σε κρυφό επίπεδο,

ο υπολογισμός του λάθους που αντιστοιχεί σε κάθε βάρος που είναι συνδεδεμένο στον νευρώνα ώστε να υπολογιστεί το αποτέλεσμα του κανόνα δέλτα και η τροποποίηση των βαρών που είναι συνδεδεμένα στον νευρώνα με βάση τον κανόνα δέλτα.

Η σημαντικότερη κλάση του συστήματος είναι η BidirectionalRecurrentNeuralNetwork. Αντικείμενο της κλάσης αυτής, ανάλογα με τις παραμέτρους εισόδου του συστήματος δημιουργεί το ΤΝΔ αμφίδρομης ανάδρασης και διαχειρίζεται τις λειτουργίες του με τις ανάλογες μεθόδους. Το δίκτυο είναι χτισμένο με διανύσματα (vectors) τύπου Neuron. Συγκεκριμένα το κάθε επίπεδο αποτελείται από διανύσματα (vectors) τύπου Neuron τα οποία μέσα από διάφορους βρόγχους και μηνύματα μεταξύ των αντικειμένων τύπου Neuron υλοποιούν τις λειτουργίες του δικτύου. Η μέθοδος doFeedForward() της κλάσης αυτής εκτελεί το εμπρόσθιο πέρασμα του δικτύου. Κάθε φορά που καλείται εκτελεί ένα πέρασμα παραθύρου εισόσου στο δίκτυο ενώ υλοποιεί το BRNN με δυο ανεξάρτητα RNN και ένα ΤΝΔ εμπρόσθιου περάσματος. Επίσης, ο αριθμός τον κρυφών επιπέδων καθορίζει το μέγεθος του context επιπέδου. Η λειτουργία της μεθόδου doFeedForward() παρουσιάζεται στο Σχήμα 5.4. Η μέθοδος doBackpropagation() εφαρμόζει τον αλγόριθμο εκμάθησης ανάστροφης μετάδοσης λάθους στο δίκτυο με βάση το Σχήμα 5.5. Το κάθε επίπεδο υπολογίζει το λάθος που πρέπει να προωθήσει προς τα πίσω στο δίκτυο και ανάλογα τροποποιεί τα βάρη που είναι συνδεδεμένα σε αυτό. Τα βάρη τροποποιούνται μετά την είσοδο του κάθε παραθύρου στο δίκτυο εφαρμόζοντας την μέθοδο on-line. Η κλάση αυτή διαθέτει επίσης μια σειρά από μεθόδους οι καλούν μεθόδους της κλάσης OutputData για να εκτυπώσουν τα κατάλληλα αποτελέσματα στα ανάλογα αρχεία ή για να υπολογίσουν το σφάλμα εκπαίδευσης ή δοκιμής κάθε περάσματος δεομένων.

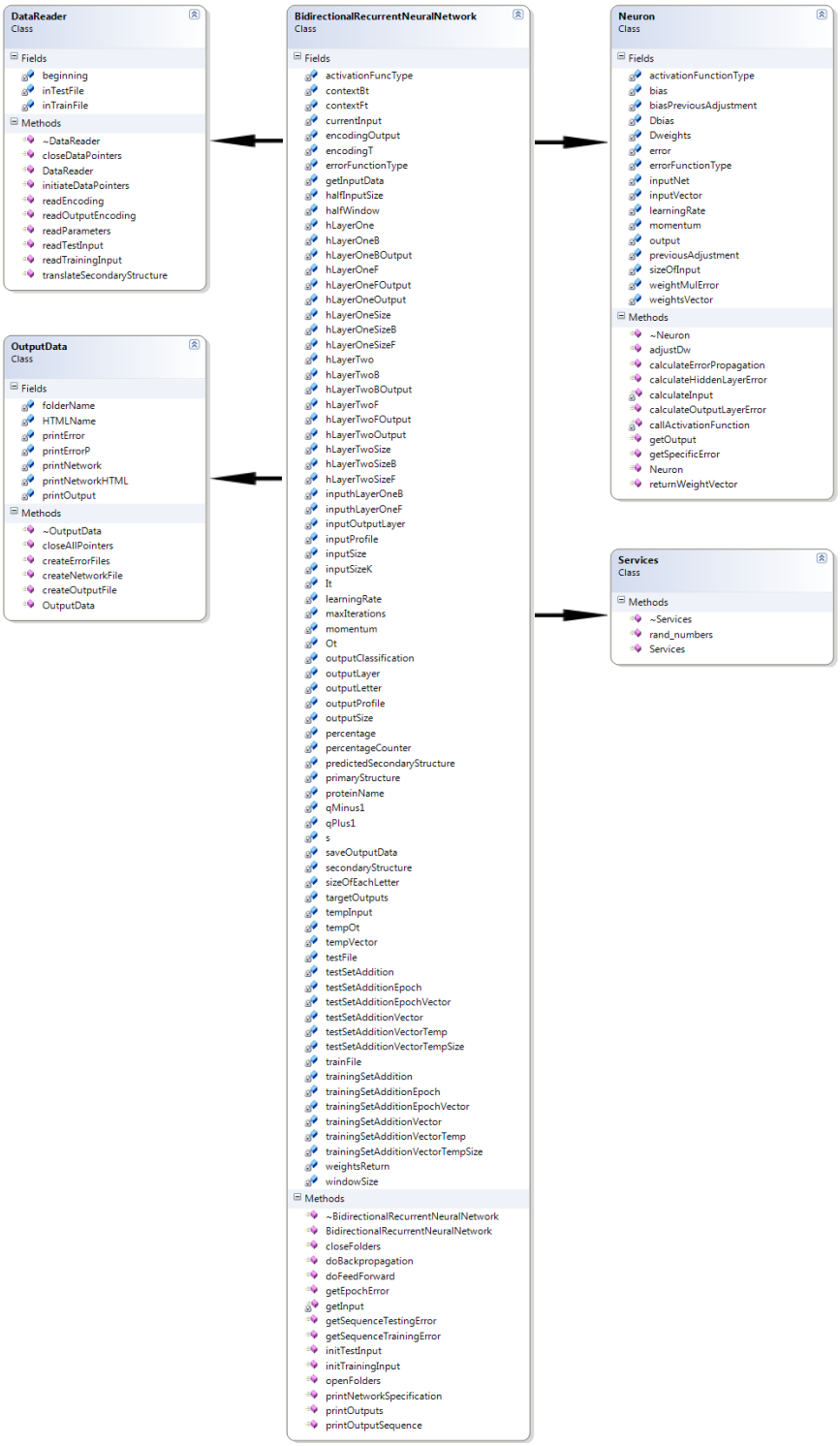
Σημαντικές για το σύστημα είναι η κλάση DataReader η οποία διαβάζει από αρχεία όλες τις εισόδους του συστήματος, η OutputData η οποία εκτυπώνει όλα τα αποτελέσματα του συστήματος σε αρχεία και η Services η οποία παρέχει βοηθητικές μεθόδους στο σύστημα, όπως μια γεννήτρια τυχαίων αριθμών για την αρχικοποίηση των τιμών του συστήματος.

```
doFeedForward(){
        While(Η είσδοσος του F RNN είναι το αμινοξί t){
                Κωδικοποίησε  την είσοδο του δικτύου
                Δώσε την είσοδο του δικτύου μαζί με την έξοδο του context επιπέδου
                Υπολόγισε την έξοδο του δικτύου
        }
        While(Η είσδοσος του B RNN είναι το αμινοξί t){
                Κωδικοποίησε  την είσοδο του δικτύου
                Δώσε την είσοδο του δικτύου μαζί με την έξοδο του context επιπέδου
                Υπολόγισε την έξοδο του δικτύου
        }
        Κωδικοποίησε την είσοδο του δικτύου εμπρόσθιου περάσματος
        Δώσε την έξοδο του δικτύου με βάση την έξοδο των δυο RNN και του εμπρόσθιου περάσματος
        Υπολόγισε ανάλογα το σφάλμα εκπαίδευσης ή δοκιμής
        }
}
```

Σχήμα 5.4: *Ψευδοκώδικας εμπρόσθιου περάσματος δικτύου BRNN*

```
doBackpropagation (){
        Υπολόγισε το σφάλμα που πρέπει να προωθηθεί πίσω στο δίκτυο
        Διόρθωσε τα βάρη που συνδέονται στο επίπεδο εξόδου
        Ανάλογα με τον αριθμό των κρυφών επιπέδων
                Υπολόγισε για κάθε επίπεδο το σφάλμα που πρέπει να προωθηθεί πίσω στο δίκτυο
                Διόρθωσε τα βάρη που συνδέονται στο κάθε επίπεδο
}
```

Σχήμα 5.5: *Ψευδοκώδικας εκμάθησης ανάστροφης μετάδοσης λάθους στο BRNN*



Σχήμα 5.6: *Σχεδιάγραμμα κλάσεων συστήματος, το βέλος συμβολίζει ότι η κλάση στο πίσω άκρο του βέλους χρησιμοποιεί αντικείμενο της κλάσης που βρίσκεται στο μπροστά άκρο του βέλους*

5.3.2 Οι λειτουργίες του BRNN για το PSSP

Το σύστημα BRNN για PSSP δημιουργήθηκε με τέτοιο τρόπο ώστε να παρέχει στον ερευνητή όσο το δυνατό πιο ευέλικτο χειρισμό του με σκοπό την δημιουργία πολλών πιθανών πειραμάτων που αφορούν κωδικοποίηση τιμών, δομή του δικτύου, είσοδο και έξοδο δεδομένων. Αυτές οι δυνατότητες παρέχονται στον χρήστη μέσω διαφόρων αρχείων τα οποία διαβάζει το σύστημα κατά την εκκίνησή του, αλλά και διαφόρων αρχείων τα οποία τα σύστημα παρέχει στον χρήστη κατά την ολοκλήρωση μιας προσομοίωσης.

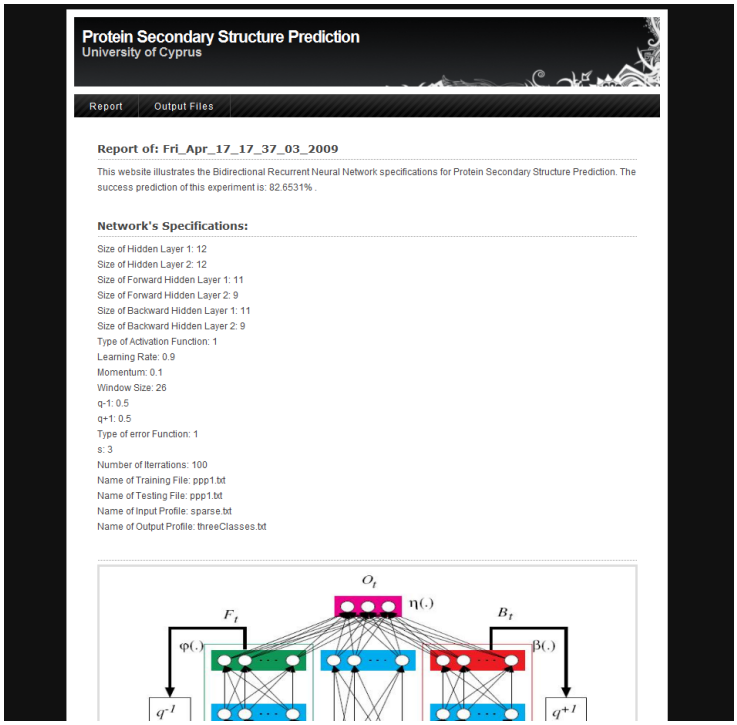
Όνομα Παραμέτρου	**Λειτουργία**
Hidden_layer_one_size	Μέγεθος πρώτου κρυφού επιπέδου δικτύου N
Hidden_layer_two_size	Μέγεθος δεύτερου κρυφού επιπέδου δικτύου N
Hidden_layer_one_of_Backward_size	Μέγεθος πρώτου κρυφού επιπέδου δικτύου B
Hidden_layer_two_of_Backward_size	Μέγεθος δεύτερου κρυφού επιπέδου 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	Συντελεστής εκμάθησης
Momentum	Ορμή
Window_size	Μέγεθος παραθύρου εισόδου
q_minus_one	Σταθερά context επιπέδου δικτύου F
q_plus_one	Σταθερά context επιπέδου δικτύου B
Error_Function_Type	Επιλογή συγκεκριμένης συνάρτησης σφάλματος
s	Εισόδου 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 5.1: *Παράμετροι εισόδου*

Όπως προαναφέραμε, η δομή του δικτύου καθορίζεται από το αρχείο παραμέτρων (Πίνακας 5.1). Το αρχείο παραμέτρων παρέχει τη δυνατότητα στον χρήστη να κτίσει το δίκτυο και να επεξεργαστεί τα δεδομένα με ποικίλους τρόπους. Σημαντικές παράμετροι που αναδεικνύουν το μέγεθος ευελιξίας του συστήματος είναι η Activation_Function_Type και η Error_Function_Type οι οποίες δίνουν τη δυνατότητα στο χρήστη να επηλεξει διαφορετικά είδη συναρτήσεων ενεργοποίησης και σφάλματος για κάθε πείραμα.

Το σύστημα δίνει επίσης την δυνατότητα στον χρήστη να επιλέγει μέσω του input_Profile το είδος της κωδικοποίησης που θα έχει το κάθε αμινοξύ. Το αρχείο αυτό καθορίζει επίσης το μέγεθος του επιπέδου εισόδου του δικτύου με βάση τους κόμβους που χρειάζονται για το κάθε είδος κωδικοποίησης αλλά και τον αριθμό των αμινοξέων που δέχεται κάθε φορά στην είσοδο του το κάθε ένα από τα τρία αλληλένδετα ΤΝΔ (Παράρτημα Ζ). Ανάλογα, το output_Profile καθωρίζει την κωδικοποίση των κατηγοριών εξόδου του δικτύου. Το αρχείο αυτό καθωρίζει την ποσότητα των κατηγοριών εξόδου του δικτύου, το όνομά τους, τα γράμματα δευτεροταγούς δομής που αντιστοιχούν σε αυτές τις κατηγορίες και την κωδικοπήσή τους (Παράρτημα Η).

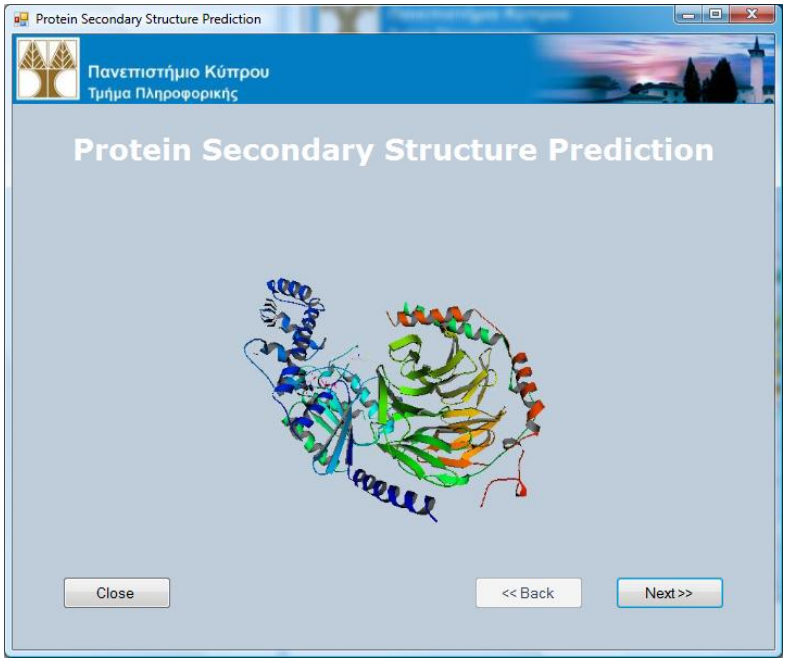
Μετά το πέρας κάθε προσομοίωσης πειράματος το δίκτυο παρουσιάζει στον χρήστη μια σειρά από αρχεία τα οποία δίνουν τα αποτελέσματα του πειράματος. Αρχικά δημιουργείται ένα .html αρχείο (Σχήμα 5.7) το οποίο παρουσιάζει τον μέσο όρο του ποσοστού ομοιότητας της δευτεροταγούς δομής της κάθε πρωτεΐνης με την προβλέψιμη δευτεροταγή δομή. Το ποσοστό ομοιότητας παρουσιάζει την ποσότητα Q3 (EVA measures for secondary structure prediction accuracy,20 April 2009). Επίσης δημιουργούνται αρχεία τα οποία παρουσιάζουν το σφάλμα εκπαίδευσης και δοκιμής ανά εποχή, το σφάλμα εκπαίδευσης και δοκιμής ανά πρωτεΐνη, τις παραμέτρους του δικτύου και την έξοδο του δικτύου για κάθε πρωτεϊνική ακολουθία του συνόλου δεδομένων δοκιμής.



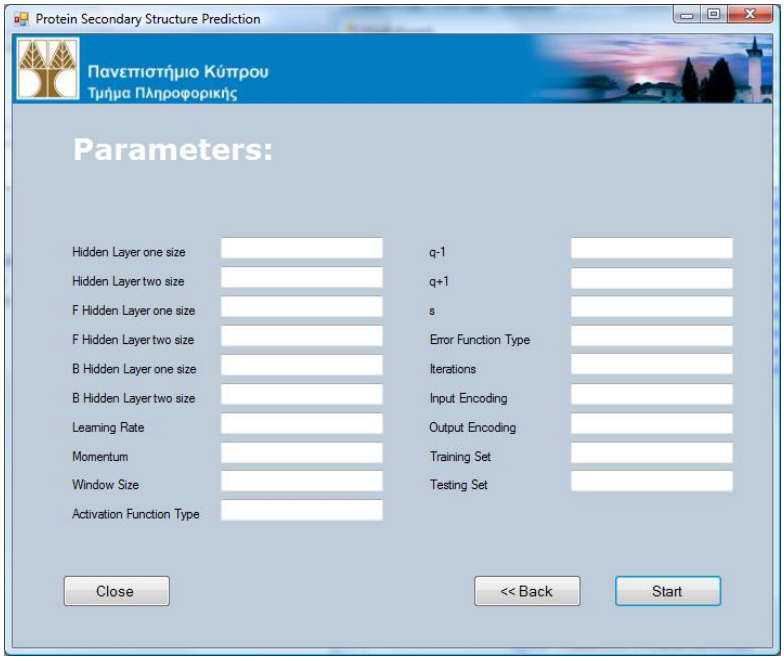
Σχήμα 5.7: *Παρουσίαση αποτελεσμάτων κάθε πειράματος σε αρχείο .html*

5.3.3 Γραφική διαπροσωπεία χρήστη

Το Νευρωνικό Δίκτυο αμφίδρομης ανάδρασης διαθέτει γραφική διαπροσωπεία χρήστη (GUI) ώστε να διευκολύνει χρήστες οι οποίοι δεν μπορούν εύκολα να χρησιμοποιήσουν κάποιο πρόγραμμα από γραμμή εντολών. Η διαπροσωπεία του συγκεκριμένου προγράμματος είναι πολύ απλή. Συγκεκριμένα, αποτελείται από μια φόρμα έναρξης (Σχήμα 5.8) η οποία οδηγεί σε μια φόρμα παραμέτρων (Σχήμα 5.9). Στην φόρμα αυτή αναγράφονται όλοι οι παράμετροι που βρίσκονται καταγραμμένοι στο αρχείο παραμέτρων. Από αυτό το σημείο ο χρήστης έχει την δυνατότητα να τροποποιήσει αυτές τις παραμέτρους. Στη συνέχεια, ο χρήστης πατώντας το κουμπί Start ξεκινά την προσομοίωση. Μια μπάρα προόδου ενημερώνει το χρήστη για το ποσοστό της προσομοίωσης που ολοκληρώθηκε. Τέλος, μετά το πέρας της προσομοίωσης, μια νέα οθόνη παρουσιάζει το ποσοστό επιτυχίας στο χρήστη.



Σχήμα 5.8: *Φόρμα έναρξης*



Σχήμα 5.9: *Φόρμα τροποποίησης παραμέτρων*

5.3.4 Πρόβλημα Vanishing Gradient και αντιμετώπιση του

Το μεγαλύτερο πρόβλημα κατά την εκπαίδευση ενός νευρωνικού δικτύου με ανάδραση είναι η μακρινή σχέση δεδομένων (long-range dependencies), το οποίο στο συγκεκριμένο πρόβλημα αναφέρεται στις σχέσεις των ακολουθιών αμινοξέων που περνούν από το δίκτυο. Στις περιπτώσεις αυτές στο δίκτυο περνά μια μεγάλη ποσότητα δεδομένων μετά από συγκεκριμένο αριθμό επαναλήψεων με αποτέλεσμα κατά την εκπαίδευση του δικτύου το σφάλμα, θεωρητικά, πρέπει να ακολουθήσει αυτό το μονοπάτι προς τα πίσω ώστε να γίνουν οι κατάλληλες διορθώσεις στα βάρη. Αυτό έχει ως αποτέλεσμα την δημιουργία του προβλήματος Vanishing Gradient όπου η κλήση της συνάρτησης του σφάλματος ως προς την αλλαγή των βαρών να αλλάζει εκθετικά, δηλαδή με πολύ μικρό ρυθμό. Αυτό το πρόβλημα καθιστά την εκπαίδευση του δικτύου πολύ δύσκολη αφού δεν παρατηρούνται μεγάλες αλλαγές στα βάρη.

Στο συγκεκριμένο πρόβλημα, ο Baldi (Baldi et al 1999) προτείνει μια λύση η οποία με εξωτερικές διασυνδέσεις στο εμπρόσθιο πέρασμα ή στο ανάστροφο πέρασμα το οποίο δημιουργεί συντόμευση μονοπατιού (shortcut) στο μοντέλο της κλήσης της συνάρτησης του σφάλματος ως προς την αλλαγή των βαρών. Η τεχνική του Baldi P. βασίζεται στα μοντέλα Markov και υλοποιήθηκε στο παρόν σύστημα. Η τεχνική αυτή υλοποιείται με τις Εξισώσεις 5.9, 5.10 και 5.11.

$$F_t = (F_{t-1}, F_{t-2}, \dots, F_{t-s}, I_t)$$

Εξίσωση 5.9: *Είσοδος Forward νευρωνικού δικτύου με ανάδραση, όπου F_{t-s} η έξοδος του δικτύου την συγκεκριμένη χρονική στιγμή και I_t η νέα είσοδος του δικτύου.*

$$B_t = (B_{t-1}, B_{t-2}, \dots, B_{t-s}, I_t)$$

Εξίσωση 5.10: *Είσοδος Backward νευρωνικού δικτύου με ανάδραση, όπου B_{t-s} η έξοδος του δικτύου την συγκεκριμένη χρονική στιγμή και I_t η νέα είσοδος του δικτύου.*

$$O_t = (B_{t-1}, B_{t-2}, \dots, B_{t-s}, F_{t-1}, F_{t-2}, \dots, F_{t-s}, I_t)$$

Εξίσωση 5.11: *Είσοδος επιπέδου εξόδου του νευρωνικού δικτύου με αμφίδρομη ανάδραση, όπου B_{t-s} η έξοδος του Backward δικτύου με ανάδραση την συγκεκριμένη χρονική στιγμή, F_{t-s} η έξοδος του Forward δικτύου με ανάδραση την συγκεκριμένη χρονική στιγμή και I_t η νέα είσοδος του δικτύου.*

Κεφάλαιο 6

Ανάλυση αποτελεσμάτων Νευρωνικού Δικτύου Αμφίδρομης Ανάδρασης

6.1 Σκοπός και πληροφορίες σχετικά με τα πειράματα

Μετά την υλοποίηση του Νευρωνικού δικτύου αμφίδρομης ανάδρασης, πολύ σημαντική κρίνεται η δημιουργία πειραμάτων τα οποία δείχνουν τα ποσοστά επιτυχίας Q3 (Εξίσωση 6.1) (EVA measures for secondary structure prediction accuracy,20 April 2009) του συστήματος που υλοποιήθηκε, δηλαδή πόσο ποσοστό επιτυχίας υπάρχει σε επίπεδο αμινοξέως προς αμινοξέως για μια άγνωστη προς το δίκτυο πρωτεϊνική ακολουθία.

$$Q_3 = 100\ \frac{1}{N_{res}} \sum_{i=0}^{3} M_{ii}$$

Εξίσωση 6.1: *Εξίσωση αξιολόγησης ποσοστού επιτυχίας, όπου N_{res} η ποσότητα των αμινοξικών καταλοίπων και M_{ij} να παίρνει την τιμή 0 αν η έξοδος i είναι διαφορετική από την έξοδο j και την τιμή 1 αν η έξοδος i είναι όμοια με την έξοδο j*

Το σύστημα αυτό με βάση τις παραμέτρους του Πίνακα 5.1 αλλά και τα διαφορετικά αρχεία εισόδου μπορεί να οδηγηθεί σε διαφορετικές δομές, οι οποίες θα οδηγούν με τη σειρά τους σε διαφορετικά αποτελέσματα. Σκοπός των πειραμάτων είναι να εξευρεθούν οι βέλτιστες παράμετροι για το δίκτυο ώστε να επιφέρουν τα καλύτερα δυνατά αποτελέσματα στη διαδικασία πρόβλεψης δευτεροταγούς δομής πρωτεϊνών. Μέσα από τα πειράματα θα εξαχθούν επίσης συμπεράσματα για το αν το συγκεκριμένο δίκτυο μπορεί να δώσει επαρκή λύση για το πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών συγκρινόμενο με προηγούμενες υλοποιήσεις.

Σημαντικό ρόλο στην εκπαίδευση και κατά συνέπεια στα ποσοστά επιτυχίας του δικτύου πολύ πιθανόν να διαδραματίζουν τα δεδομένα εισόδου, όπως σε κάθε νευρωνικό δίκτυο. Για το λόγο αυτό δημιουργήθηκαν μετά από προεπεξεργασία διαφορετικά σύνολα δεδομένων πρωτεϊνών. Όπως προαναφέρθηκε στο Υποκεφάλαιο 4.4 δημιουργήθηκαν τρία διαφορετικά σύνολα δεδομένων των 100 πρωτεϊνικών ακολουθιών για εκπαίδευση του δικτύου με αντίστοιχα τρία διαφορετικά σύνολα δεδομένων των 30 πρωτεϊνικών

ακολουθιών για έλεγχο του δικτύου. Αυτά ονομάζονται Dataset1, Dataset2 και Dataset3. Τα σύνολα αυτά περιέχουν πρωτεΐνες οι οποίες δεν έχουν ελλιπή στοιχεία στην πρωτοταγή δομή τους. Επίσης δημιουργήθηκε ένα σύνολο δεδομένων, με όνομα proteins, δεδομένων των 100 πρωτεϊνικών ακολουθιών για εκπαίδευση του δικτύου με αντίστοιχο σύνολο δεδομένων των 30 πρωτεϊνικών ακολουθιών για έλεγχο του δικτύου. Το σύνολο αυτό περιέχει πρωτεϊνικές ακολουθίες οι οποίες περιέχουν ελλιπής δεδομένα στην πρωτοταγή και δευτεροταγή τους δομή, δηλαδή περιέχουν την τιμή X όπου απουσιάζουν δεδομένα από την πρωτοταγή δομή και την τιμή της γενικής κατηγορία για τη δευτεροταγή δομή. Επίσης δημιουργήθηκε ένα σύνολο δεδομένων των 100 πρωτεϊνικών ακολουθιών για εκπαίδευση του δικτύου με ένα αντίστοιχο σύνολο δεδομένων των 30 πρωτεϊνικών ακολουθιών τα οποία αντικαθιστούν τα ελλιπής δεδομένα της πρωτοταγής δομής με τυχαίες τιμές του συνόλου των τιμών που μπορεί να έχει η πρωτοταγής και δευτεροταγή δομή, αντίστοιχα. Τέλος, δημιουργήθηκαν σύνολα δεδομένων στα οποία η τυχαία επιλογή των πρωτεϊνών αφορούσε μόνο πρωτεΐνες οι οποίες δεν περιέχουν ελλιπής δεδομένα στην πρωτοταγή δομή τους. Με αυτό τον περιορισμό δημιουργήθηκε ένα σύνολο δεδομένων των 500 πρωτεϊνικών ακολουθιών για εκπαίδευση και 125 πρωτεϊνικών ακολουθιών για δομική του δικτύου, καθώς και ένα σύνολο δεδομένων των 600 πρωτεϊνικών ακολουθιών για εκπαίδευση και 200 πρωτεϊνικών ακολουθιών για δομική του δικτύου. Από τα σύνολα δεδομένων που προαναφέρθηκαν, απουσιάζουν πρωτεϊνικές ακολουθίες οι οποίες στην πρωτοταγή τους δομή περιέχουν στην θέση των αμινοξέων γράμματα τα οποία αναπαριστούν το αμινοξύ κυστεΐνη και συμβολίζονται με μικρά γράμματα του αλφαβήτου.

Λόγω μη επαρκούς χρόνου για την εκτέλεση όλων των απαραίτητων πειραμάτων, καθώς ο χρόνος που χρειάζεται ένα Νευρωνικό Δίκτυο αμφίδρομης ανάδρασης είναι πολύ μεγάλος, μετά από μελέτη σχεδιαστήκαν τα πειράματα και εκτελέστηκαν συγκεκριμένα πειράματα. Τα πειράματα αυτά έπρεπε να εκτελεστούν ώστε να παρθούν τα πιο βασικά συμπεράσματα γύρω από τις δυνατότητες του Νευρωνικού δικτύου που υλοποιήθηκε, με σκοπό την περεταίρω βελτίωσή του.

6.2 Πειράματα και αποτελέσματα

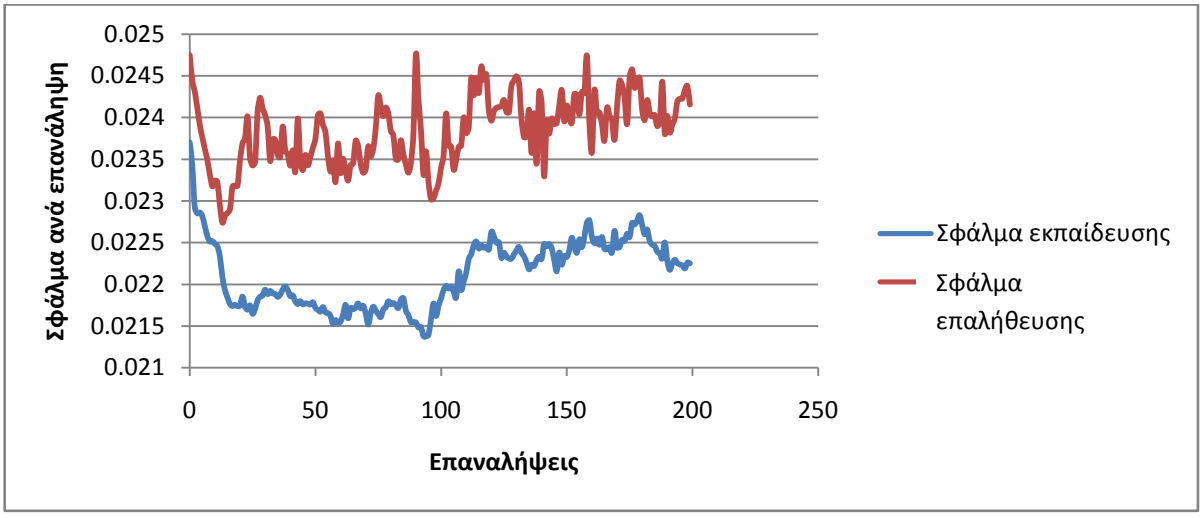
6.2.1 Πειράματα σχετικά με παραμέτρους του δικτύου

Για την καταγραφή και έναρξη των πειραμάτων ακολουθήθηκε μια στρατηγική ώστε τα πειράματα να έχουν κάποια σχέση μεταξύ τους, να μειωθεί ο παράγοντας τύχη και η ακολουθία των αποτελεσμάτων να οδηγήσει στις καλύτερες δυνατές παραμέτρους που επηρεάζουν τα αποτελέσματα του Νευρωνικού Δικτύου αμφίδρομης ανάδρασης. Η στρατηγική αυτή αφορούσε την αρχική καταγραφή κάποιων παραμέτρων και χρήση τους σε ένα συγκεκριμένο σύνολο δεδομένων, ώστε με τροποποίηση των παραμέτρων αυτών να διαφανεί κατά πόσο η κάθε παράμετρος επηρεάζει τα αποτελέσματα. Οι παράμετροι που ορίστηκαν αρχικά ήταν βασισμένες στις ανάλογες παραμέτρους που χρησιμοποίησε ο Baldi (Baldi et al 1999,Baldi et al 2000) σε ανάλογα πειράματα με ανάλογο δίκτυο (Πίνακα 6.1).

Όνομα Παραμέτρου	**Τιμή**
Hidden_layer_one_size	12
Hidden_layer_two_size	12
Hidden_layer_one_of_Backward_size	11
Hidden_layer_two_of_Backward_size	9
Hidden_layer_one_of_Forward_size	11
Hidden_layer_two_of_Forward_size	9
Learning_Rate	0.5
Momentum	0.1
Window_size	27
q_minus_one	0.5
q_plus_one	0.5
s	3
Maximum_Iterations	200

Πίνακα 6.1: *Αρχικές παράμετροι δικτύου*

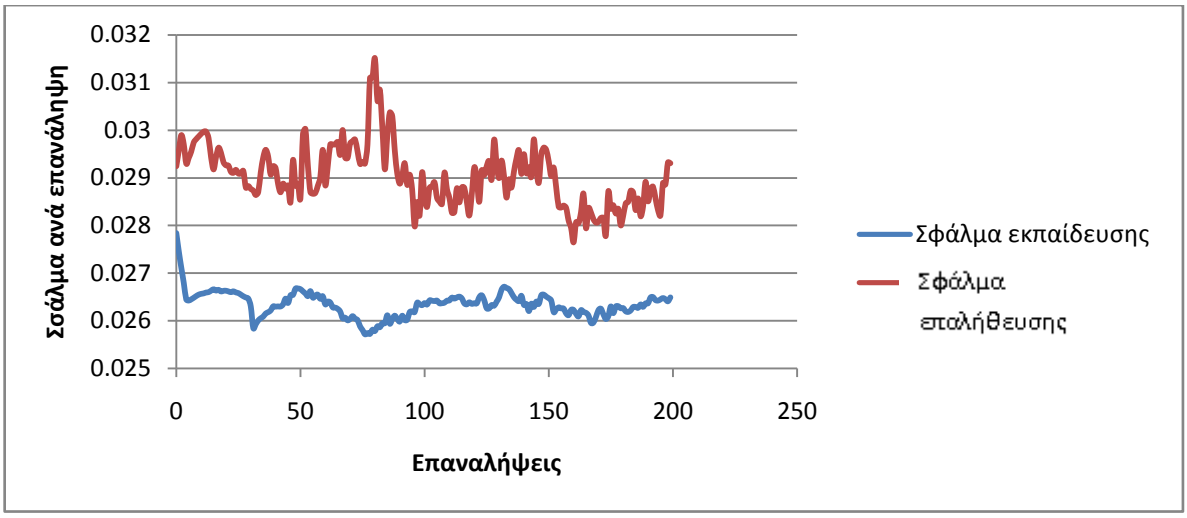
Στα πειράματα που εκτελέστηκαν εκτός από τις παραμέτρους που αναγράφονται στον Πίνακα 6.1, χρησιμοποιήθηκαν επίσης ως βηματική συνάρτηση και συνάρτηση σφάλματος η σιγμοειδής συνάρτηση με κλήση 1 και η LMS, αντίστοιχα. Επίσης, τα σύνολα που χρησιμοποιήθηκαν είναι το Dataset3 και το proteins (Υποκεφάλαιο 6.1) με κωδικοποίηση εισόδου sparse (Παράρτημα Ζ) και τρεις κατηγορίες εξόδου (Παράρτημα Η). Η κωδικοποίηση sparse θεωρείται η πιο απλή για το συγκεκριμένο πείραμα, όμως είναι ιδανική αφού τα διανύσματα εισόδου στον 20διάστατο χώρο έχουν την ίδια απόσταση μεταξύ τους χωρίς να επηρεάζουν το αποτέλεσμα λόγω κωδικοποίησης. Επιπλέον, τα δεδομένα που υπήρχαν σύμφωνα με τον Baldi (Baldi et al 1999, Baldi et al 2000) για τις παραμέτρους συντελεστή μάθησης (learning rate), q-1,q-2 και μέγεθος παραθύρου εισόδου δεν ήταν επαρκές και γι' αυτό το λόγο χρησιμοποιήθηκαν κάποιες αρχικά τυχαίες τιμές. Σκοπός του πειράματος αυτού ήταν η δημιουργία κάποιων αρχικών αποτελεσμάτων ώστε να χρησιμοποιηθούν ως μέτρο σύγκρισης για τα αποτελέσματα από διαφορετικές παραμέτρους.



Σχήμα 6.1: *Γραφική παράσταση του Σφάλματος εκπαίδευσης και δοκιμής για το Dataset3*

Οι γραφικές παραστάσεις στα Σχήματα 6.1 και 6.2 παρουσιάζουν τα αποτελέσματα που αφορούν το σφάλμα εκπαίδευσης και το σφάλμα δοκιμής για τα δυο πειράματα. Στην περίπτωση του Dataset3, το οποίο δεν περιέχει ελλιπής δεδομένα, το ποσοστό επιτυχίας με

βάση το Q3 ήταν 58.46% και στην περίπτωση του proteins, όπου υπάρχουν ελλιπή δεδομένα, το ποσοστό επιτυχίας ήταν 59.38% . Τα ποσοστά αυτά θεωρούνται χαμηλά για τις προβλεπόμενες δευτεροταγείς δομές των πρωτεϊνών των συνόλων δοκιμής. Παρόλα αυτά θεωρούνται ικανοποιητικά για αρχή της προσπάθειας βελτιστοποίησης των αποτελεσμάτων του Νευρωνικού Δικτύου αμφίδρομης ανάδρασης, αφού θεωρητικά το δίκτυο μπορεί να προβλέψει περίπου το 60% των δομών δευτεροταγής δομής που έπρεπε να προβλέπει.



Σχήμα 6.2: *Γραφική παράσταση του Σφάλματος εκπαίδευσης και δοκιμής για το proteins*

Από τις γραφικές παραστάσεις στα Σχήματα 6.1 και 6.2 παρατηρούμε ότι το σφάλμα δοκιμής μειώνεται κατά τη διάρκεια των επαναλήψεων. Στην περίπτωση του Dataset3 αυτό μειώνεται μέχρι το 0.0215 ενώ στην περίπτωση του Proteins αυτό μειώνεται μέχρι 0.026. Στη συνέχεια παρατηρούνται αυξομειώσει στο σφάλμα εκπαίδευσης και στις δυο περιπτώσεις. Αυτό πιθανόν να οφείλεται στο γεγονός ότι το συγκεκριμένο πρόβλημα δημιουργεί πολλά τοπικά ελάχιστα λόγο του τεράστιου όγκου δεδομένων που περιέχεται στα διαφορετικά μοτίβα των ακολουθιών πρωτεϊνών, με αποτέλεσμα δύσκολα να εντοπίζεται το ολικό ελάχιστο. Η μέθοδος κατάβασης κλήσης στην οποία βασίζεται ο αλγόριθμος ανάστροφής μετάδοσης λάθους (Backpropagation Algorithm) (Rumelhart et al 1986) αντιμετωπίζει το πρόβλημα αυτό με το συντελεστή μάθησης (learning rate) και την

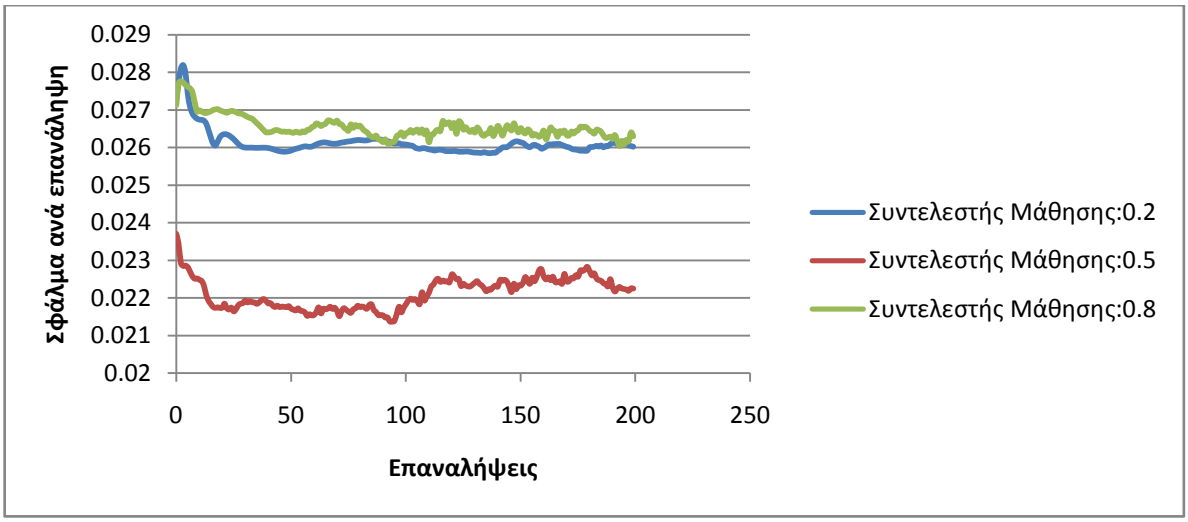
ορμή (momentum). Αυτό όμως δεν δείχνει ικανό να αντιμετωπίσει το πρόβλημα των τοπικών ελαχίστων. Σε αυτές τις περιπτώσεις χρήσιμος μπορεί να είναι ένας αλγόριθμος adaptive learning rate. Ο αλγόριθμος αυτός μπορεί να εντοπίσει αν η μέθοδος κατάβασης κλήσης εγκλωβίστηκε σε τοπικό ελάχιστο ή ολικό ελάχιστο και ανάλογα να μεγαλώνει τον συντελεστή μάθησης αν πρέπει να βγει από αυτό ή να τον ελαττώνει αν πρέπει να μείνει σε αυτό.

Από τις γραφικές παραστάσεις στα Σχήματα 6.1 και 6.2 παρατηρούμε επίσης ότι το σφάλμα δοκιμής είναι πιο ψηλό από το σφάλμα εκπαίδευσης. Αυτό δείχνει ότι τα δεδομένα του συνόλου εκπαίδευσης δεν αναγνωρίζονται σε μεγάλο βαθμό από το δίκτυο μετά την εκπαίδευσή του. Συγκεκριμένα τα σχετικά μικρά σύνολα εκπαίδευσης, τα οποία παρουσιάζουν στο δίκτυο μικρό αριθμό μοτίβων πρωτοταγούς δομής πρωτεϊνών που αντιστοιχούν σε συγκεκριμένη κατηγορία δευτεροταγούς δομής, δεν δημιουργούν στο τρισδιάστατο νοητό χώρο των δεδομένων τα απαραίτητα όρια αποφάσεων. Αποτέλεσμα αυτού είναι να μην μπορεί το δίκτυο να κατηγοριοποιήσει όλα τα διαφορετικά είδη διαφορετικών πρωτοταγών δομών πρωτεϊνών.

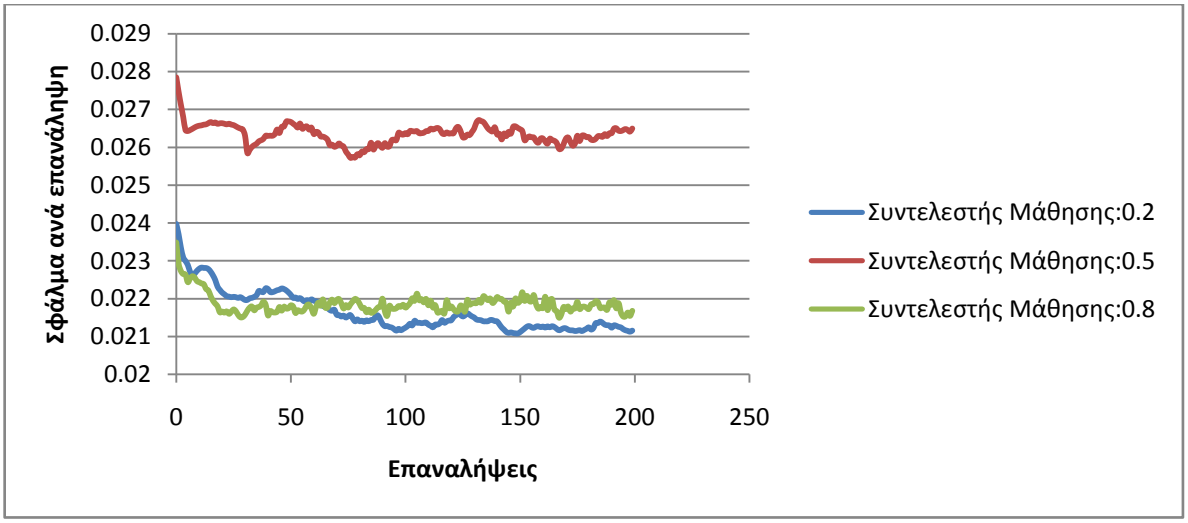
Στη συνέχεια με βάση την ανάλυση που προηγήθηκε εκτελέστηκαν διάφορα πειράματα τα οποία είχαν ως στόχο να δούμε πως επηρεάζουν οι διάφορες παράμετροι τα αποτελέσματα του δικτύου. Οι παράμετροι που ελέχθησαν αρχικά ήταν ο συντελεστής μάθησης (learning rate), το q-1, το q-2 και το μέγεθος παραθύρου εισόδου, αφού για αυτές τις παραμέτρους δεν είχαμε κανένα στοιχείο ως προς το πώς θα έπρεπε να είναι διαμορφωμένες οι τιμές, σε αντίθεση με τις υπόλοιπες παραμέτρους, που υπήρχε κάποια αρχική εκτίμηση με βάση τον Baldi (Baldi et al 1999, Baldi et al 2000). Τα πειράματα αυτά ήταν καταγεγραμμένα και ακολουθήθηκε πιστά ο αρχικός σχεδιασμός πειραμάτων. Στη συνέχεια ανάλογα με τα αποτελέσματα αυτών των πειραμάτων δημιουργήθηκαν και εκτελέστηκαν νέα πειράματα με σκοπό την εξεύρεση παραμέτρων που δίνουν το καλύτερο δυνατό ποσοστό επιτυχίας Q3.

Το επόμενο πείραμα που εκτελέστηκε αφορούσε την διατήρηση όλων των σταθερών παραμέτρων, όπως παρουσιάζονται στον Πίνακα 6.1 με εξαίρεση τον συντελεστή μάθησης. Σκοπός του πειράματος ήταν να εξαχθεί κάποιο συμπέρασμα για το πόσο επηρεάζει το αποτέλεσμα ο συντελεστής μάθησης σε σχέση με το αρχικό πείραμα. Συγκεκριμένα

εκτελέστηκαν τέσσερα πειράματα με τα δυο βασικά σύνολα δεδομένων, Dataset3 και proteins, με συντελεστή μάθησης 0.2 και 0.8.

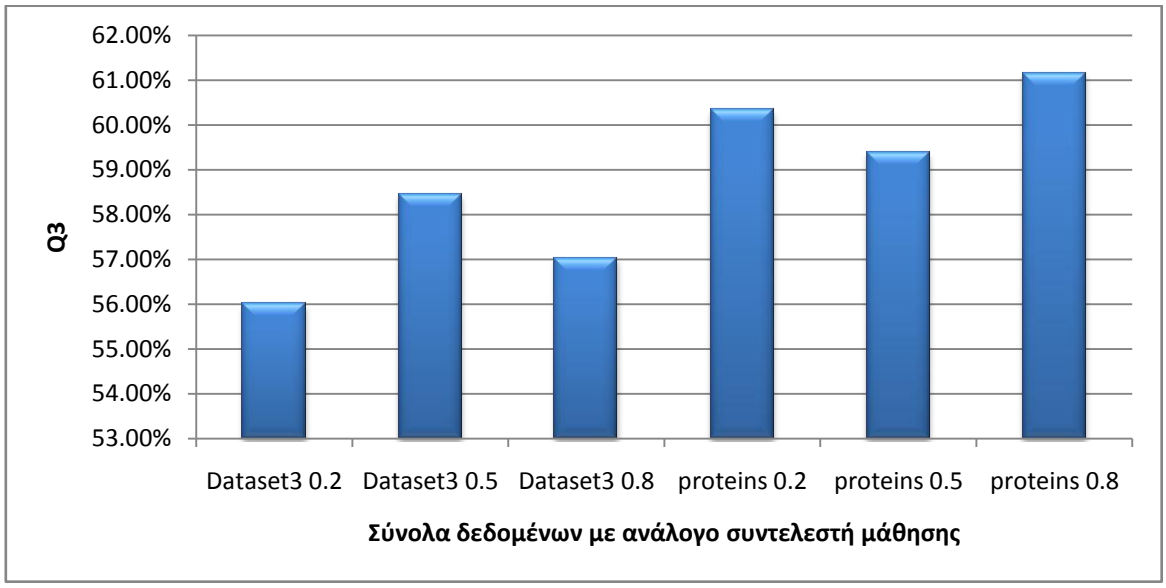


Σχήμα 6.3: *Γραφική παράσταση του Σφάλματος εκπαίδευσης σχετικά με διαφορετικούς συντελεστές μάθησης για το Dataset3*



Σχήμα 6.4: *Γραφική παράσταση του Σφάλματος εκπαίδευσης σχετικά με διαφορετικούς συντελεστές μάθησης για το proteins*

Τα αποτελέσματα των πειραμάτων που εξετάζουν τον συντελεστή μάθησης παρουσιάζονται στα Σχήματα 6.3 και 6.4. Οι γραφικές αυτές παραστάσεις παρουσιάζουν την μείωση του σφάλματος εκπαίδευσης ως προς τις επαναλήψεις της κάθε προσομοίωσης. Επίσης, στο Σχήμα 6.5 παρουσιάζεται το ποσοστό επιτυχίας της κάθε προσομοίωσης. Γενικά, παρατηρείται ότι όταν χρησιμοποιείται το Dataset3, ο συντελεστής μάθησης με τα καλύτερα αποτελέσματα ως προς το Q3 είναι το 0.5. Παρατηρείται ότι στην συγκεκριμένη προσημείωση το σφάλμα εκπαίδευσης είναι μικρότερο από τις περιπτώσεις που ο συντελεστής μάθησης είναι 0.2 και 0.8. Αντίθετα, αν το σύνολο δεδομένων αντικατασταθεί από το proteins παρατηρείται ότι το πιο ψηλό ποσοστό επιτυχίας με ανάλογα το πιο χαμηλό σφάλμα εκπαίδευσης παρατηρείται στους συντελεστές σφάλματος 0.2 και 0.8.

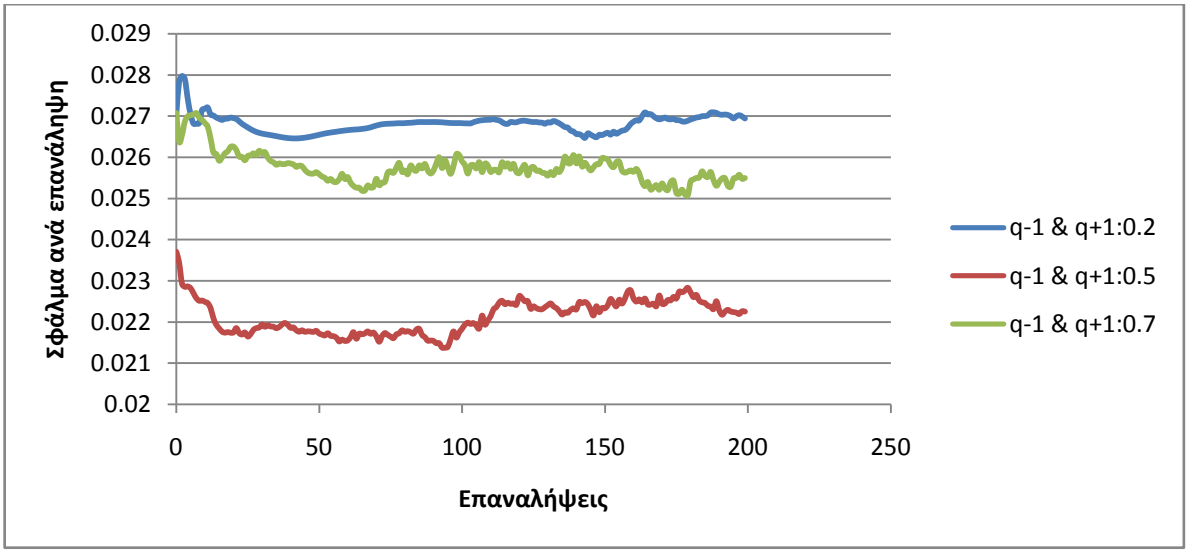


Σχήμα 6.5: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 του συστήματος ανάλογα με το σύνολο δεδομένων και τον συντελεστή μάθησης*

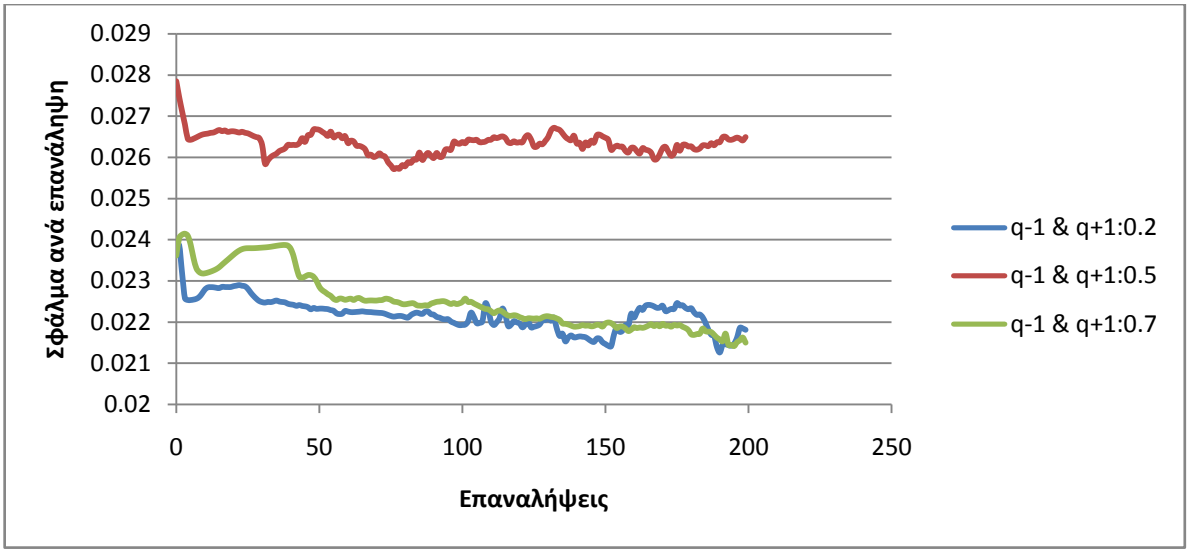
Στην περίπτωση όπου το σύνολο δεδομένων είναι το Dataset3 παρατηρείται μια δυσκολία από το δίκτυο να εντοπίσει το ολικό ελάχιστο αφού υπάρχει μεγαλύτερος όγκος δεδομένων από την περίπτωση του proteins. Εδώ πρέπει να επισημάνουμε ότι το σύνολο δεδομένων proteins περιέχει μεγάλες ακολουθίες δεδομένων οι οποίες περιείχαν ελλιπή δεδομένα και αντικαταστάθηκαν στην πρωτοταγή δομή τους με X και στην δευτεροταγής δομή τους με L,

κάτι το οποίο παρατηρείται ότι το εκμεταλλεύεται πιο εύκολα το δίκτυο. Στην περίπτωση του Dataset3 υπάρχει μεγαλύτερος όγκος δεδομένων με αποτέλεσμα η μέθοδος κατάβασης κλήσης να συναντά πολλά τοπικά ελάχιστα. Μια ενδιάμεση τιμή συντελεστή μάθησης επιτυγχάνει τη συνέχιση της αναζήτησης του ολικού ελάχιστου από την μέθοδο κατάβασης κλήσης χωρίς να εγκλωβίζεται εύκολα στα πολλά τοπικά ελάχιστα κάτι το οποίο δεν κάνει ο συντελεστής μάθησης 0.2. Στην περίπτωση του proteins παρατηρούνται γενικά πιο μεγάλα ποσοστά επιτυχίας λόγο τις μείωσης της πολυπλοκότητας των δεδομένων εισόδου, με τον συντελεστή μάθησης 0.8 να έχει ποσοστό επιτυχίας 61.16%.

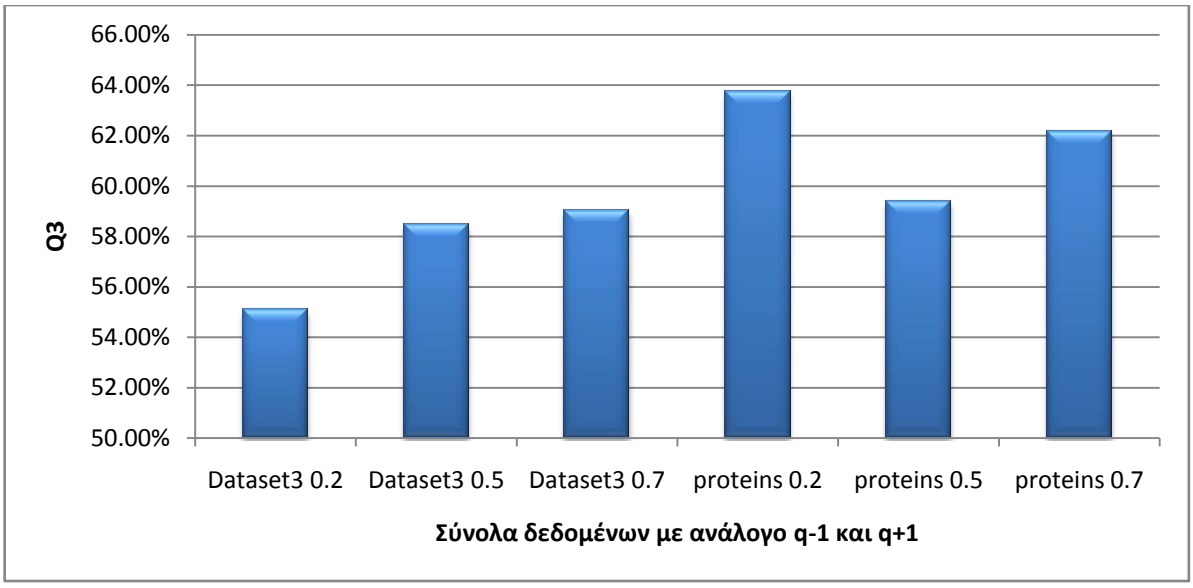
Το επόμενο πείραμα σχεδιάστηκε και εκτελέστηκε με σκοπό να εντοπιστεί η σωστή σταθερά ανάδρασης για το δίκτυο. Η σταθερά αυτή χρησιμοποιείται για την δημιουργία των δεδομένων εισόδου του επιπέδου context του κάθε Νευρωνικού Δικτύου με ανάδραση (Elman 1990). Όπως και στην προηγούμενη περίπτωση εκτελέστηκαν τέσσερα πειράματα με τα δυο βασικά σύνολα δεδομένων, Dataset3 και proteins, με σταθερά ανάδρασης 0.2 και 0.7.



Σχήμα 6.6: *Γραφική παράσταση του Σφάλματος εκπαίδευσης σχετικά με διαφορετικές σταθερές ανάδρασης για το Dataset3*



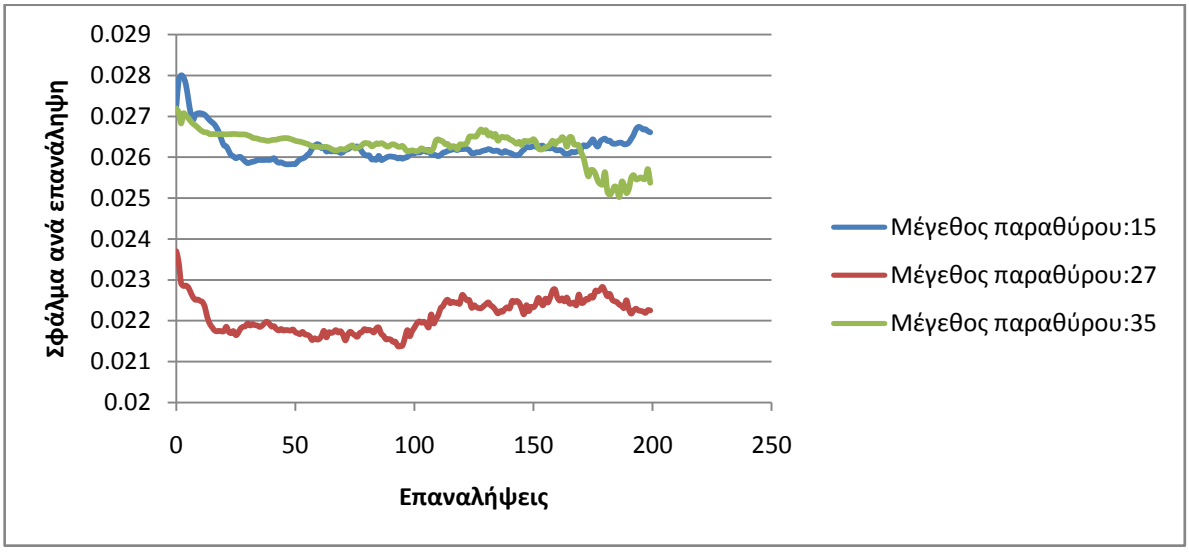
Σχήμα 6.7: *Γραφική παράσταση του Σφάλματος εκπαίδευσης σχετικά με διαφορετικές σταθερές ανάδρασης για το proteins*



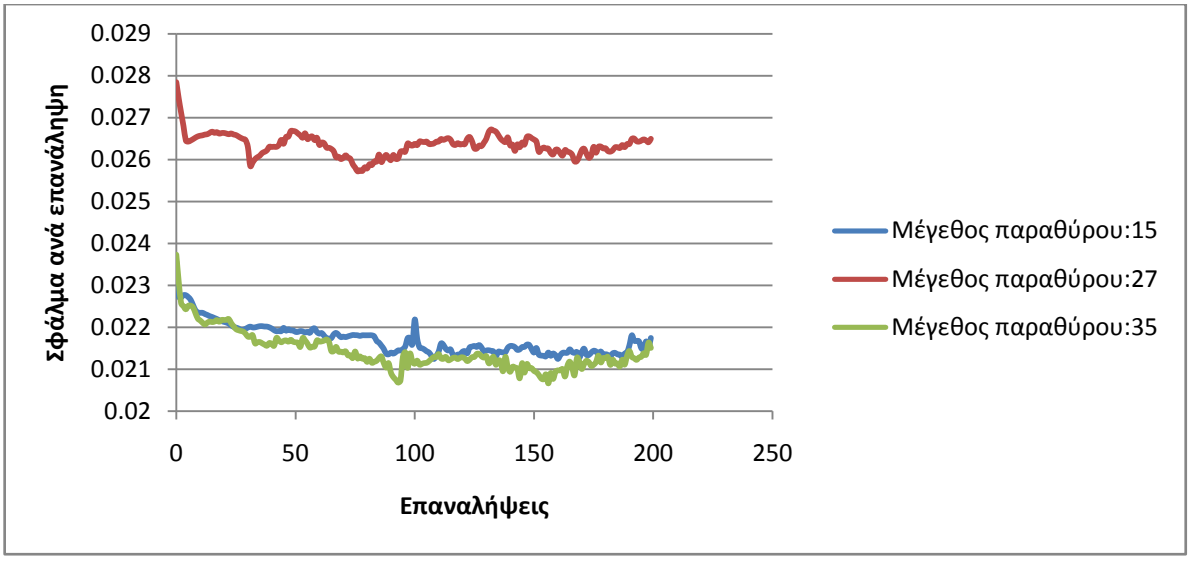
Σχήμα 6.8: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 του συστήματος ανάλογα με το σύνολο δεδομένων και την σταθερά ανάδρασης*

Οι γραφικές παραστάσεις στα Σχήματα 6.7 και 6.8 παρουσιάζουν την μείωση του σφάλματος εκπαίδευσης μετά από 200 επαναλήψεις, ενώ το Σχήμα 6.9 παρουσιάζει τα ποσοστά επιτυχίας του δικτύου με βάση τις διάφορες σταθερές ανάδρασης. Τα αποτελέσματα του δικτύου όταν χρησιμοποιήθηκε το Daraset3 είχαν πιο ψηλό σφάλμα εκπαίδευσης και χαμηλότερα ποσοστά επιτυχίας από τις περιπτώσεις που χρησιμοποιήθηκε το σύνολο δεδομένων proteins, για το λόγο που προαναφέρθηκε στο προηγούμενο πείραμα. Στο πείραμα αυτό εξάχθηκε το ψηλότερο ποσοστό επιτυχίας από όλα τα πειράματα που έγιναν μέχρι τη δεομένη στιγμή με 63.75%.

Από αυτό το πείραμα οδηγούμαστε στο συμπέρασμα ότι η σταθερά ανάδρασης πρέπει να είναι μικρή, αφού το 63.75% ποσοστό επιτυχίας απέχει σε μεγάλο βαθμό από τα υπόλοιπα αποτελέσματα. Σύμφωνα με το Νευρωνικό Δίκτυο με ανάδραση του Elman (Elman 1990), η σταθερά ανάδρασης καθορίζει το πόσο ρόλο έχει κάθε προηγούμενη είσοδος δεδομένων στο νευρωνικό δίκτυο με ανάδραση στο αποτέλεσμα της επόμενης εισόδου δεδομένων. Με βάση τη θεωρία αυτή, όταν η ακολουθία των αμινοξέων περνά σαν είσοδος στο κάθε ένα από τα δυο Νευρωνικά Δίκτυα με ανάδραση, δεν πρέπει να ενσωματώνεται μεγάλο ποσοστό αυτής σε κάθε επόμενη ακολουθία αμινοξέων για να μπορεί το δίκτυο να προβλέπει σωστά τις ανάλογες δευτεροταγείς δομές.



Σχήμα 6.9: *Γραφική παράσταση του Σφάλματος εκπαίδευσης σχετικά με διαφορετικά μεγέθη παραθύρων για το Dataset3*

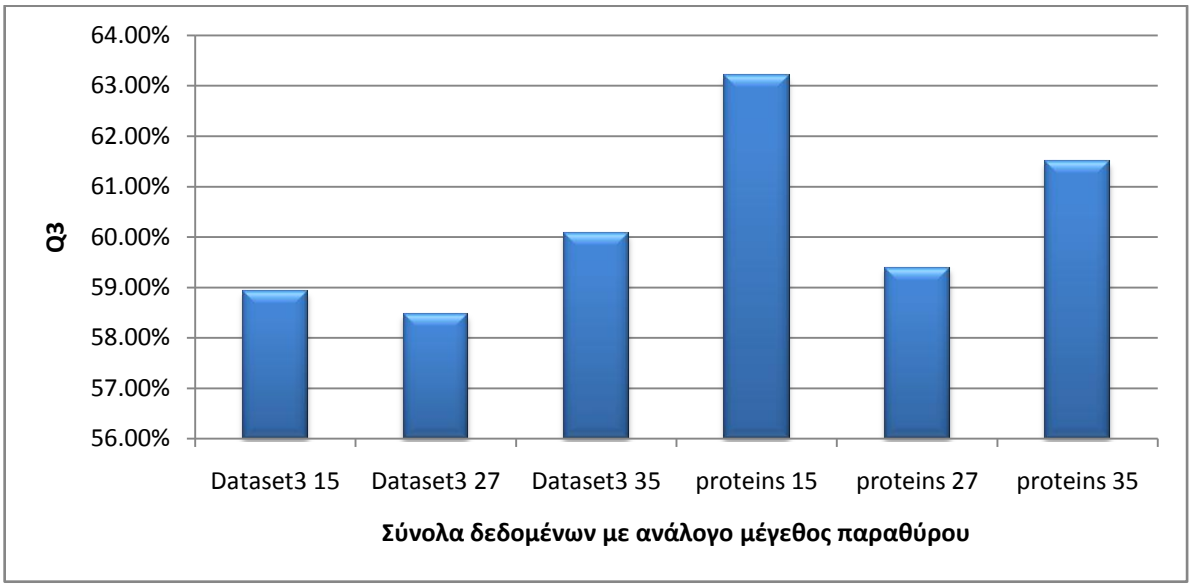


Σχήμα 6.10: *Γραφική παράσταση του Σφάλματος εκπαίδευσης σχετικά με διαφορετικά μεγέθη παραθύρων για το proteins*

Η τρίτη παράμετρος που ελέγχθηκε ήταν το μέγεθος του παραθύρου δεδομένων εισόδου του δικτύου. Η παράμετρος αυτή καθορίζει πόσα αμινοξέα δίνονται κάθε φορά στην είσοδο του δικτύου με αναλογία «Χ αμινοξέα που προηγούνται του αμινοξέως που ελέγχεται η δευτεροταγής δομή του + αμινοξί που ελέγχεται η δευτεροταγής δομή του + Χ αμινοξέα που έπονται του αμινοξέως που ελέγχεται η δευτεροταγής δομή του». Και σε αυτή την περίπτωση εκτελέστηκαν τέσσερα πειράματα με τα δυο βασικά σύνολα δεδομένων, Dataset3 και proteins, με μεγέθη παραθύρων εισόδου 15 και 35.

Τα αποτελέσματα του πιο πάνω πειράματος παρουσιάζονται στις γραφικές παραστάσεις που απεικονίζονται στα Σχήματα 6.9 και 6.10 τα οποία παρουσιάζουν την μείωση του σφάλματος εκπαίδευσης κατά την πάροδο των επαναλήψεων της προσομοίωσης. Επίσης, το Σχήμα 6.11 παρουσιάζει τα ποσοστά επιτυχίας των πειραμάτων αυτών. Παρατηρούμε ότι το μέγεθος παραθύρου 15 με το σύνολο δεδομένων proteins έχει τα πιο ψηλά ποσοστά επιτυχίας με 63.20%. Το αξιοσημείωτο σε αυτό το πείραμα είναι ότι όταν το μέγεθος του παραθύρου είναι 15 αμινοξέα έχουμε μεγάλη διαφορά στα ποσοστά επιτυχίας από τις περιπτώσεις που το παράθυρο επιτυχίας είναι 27 και 35 αμινοξέα. Αυτό πιθανόν να

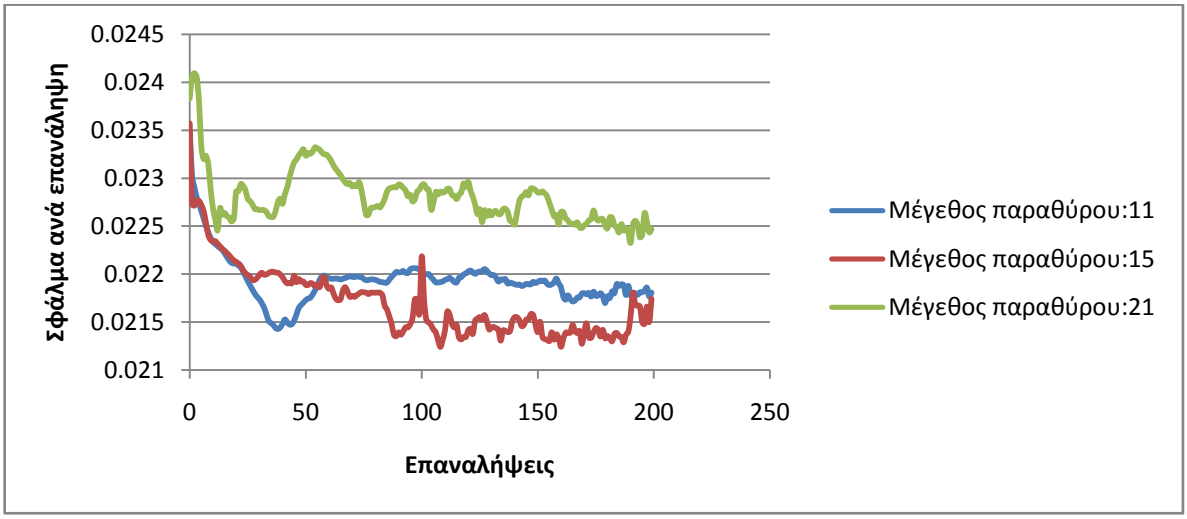
οφείλεται στο γεγονός ότι μεγάλες ακολουθίες αμινοξέων προκαλούν υπερεκπαίδευση στο Νευρωνικό Δίκτυο (Baldi et al 1999). Συγκεκριμένα το δίκτυο με ανάδραση δεν μπορεί να κρατά στη μνήμη του μεγάλες ακολουθίες, κάτι το οποίο έρχεται σε αντίθεση με το πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών. Σύμφωνα με το πρόβλημα που περιγράφεται στο Κεφάλαιο 1, η δευτεροταγής δομή των πρωτεϊνών μπορεί να εξαρτάται από αμινοξέα που βρίσκονται σε μεγάλη απόσταση από το αμινοξύ που εξετάζεται η δευτεροταγής δομή του, κάτι το οποίο μεταφράζεται σε μεγάλο μέγεθος παραθύρου εισόδου στο Νευρωνικό δίκτυο με αμφίδρομη ανάδραση. Μια λύση στο συγκεκριμένο πρόβλημα θα ήταν η χρησιμοποίηση αλγόριθμων συμπίεσης δεδομένων κατά την κωδικοποίηση των δεδομένων εισόδου του δικτύου.



Σχήμα 6.11: *Γραφική παράσταση που παρουσιάζει τα ποσοστά επιτυχίας Q3 του συστήματος ανάλογα με το σύνολο δεδομένων και το παράθυρο εισόδου*

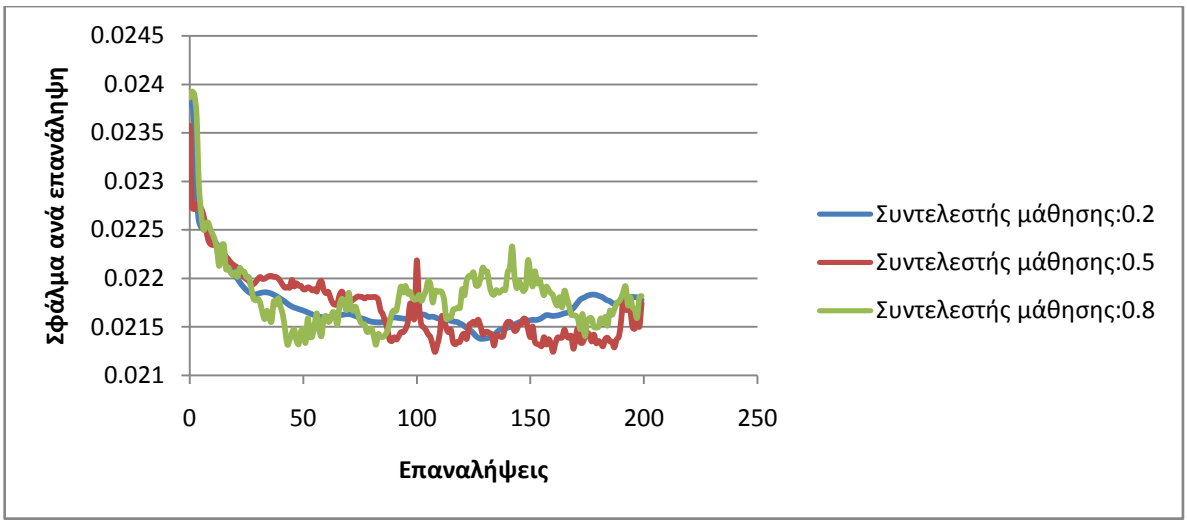
Η πιο πάνω ανάλυση οδηγεί στο συμπέρασμα ότι το μέγεθος του παραθύρου εισόδου δεδομένων έχει καταλυτικό ρόλο στη διαδικασία εξεύρεσης λύσης στο πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών. Για αυτό το λόγο εκτελέστηκαν κάποια επιπλέον πειράματα σχετικά με το μέγεθος του παραθύρου εισόδου χρησιμοποιώντας το σύνολο δεδομένων proteins. Τα επιπλέον μεγέθη παραθύρων που χρησιμοποιήθηκαν είναι

11 και 21, τα οποία είναι σχετικά μικρά και κοντά στο μέγεθος του παραθύρου μεγέθους 15. Τα αποτελέσματα των πειραμάτων αυτών παρουσιάζονται στο Σχήμα 6.12. Από αυτή την γραφική παράσταση παρατηρούμε ότι την μεγαλύτερη μείωση στο σφάλμα εκπαίδευσης έχει το μέγεθος παραθύρου 15, ενώ τα ανάλογα μεγέθη των 11 και 21 αμινοξέων δεν μπορούν να εκπαιδεύσουν σε καλύτερο βαθμό το δίκτυο. Αυτό παρατηρείται και από τα ποσοστά επιτυχίας των συγκειμένων παραθύρων με 60.61% το παράθυρο μεγέθους 11 60.07% το αντίστοιχο μεγέθους 21.



Σχήμα 6.12: *Γραφική παράσταση του Σφάλματος εκπαίδευσης σχετικά με διαφορετικά μεγέθη παραθύρων για το proteins*

Από τα πιο πάνω πειράματα παρατηρήθηκε ότι το δίκτυο εκπαιδεύτηκε με τέτοιο τρόπο ώστε να έχει τα καλύτερα αποτελέσματα όταν είχε μέγεθος παραθύρου εισόδου 15. Αυτή η παρατήρηση οδήγησε στα επόμενα πειράματα στα οποία γίνονται κάποιες τροποποιήσεις στις παραμέτρους του δικτύου διατηρώντας σταθερό το παραθύρου εισόδου στα 15 αμινοξέα.

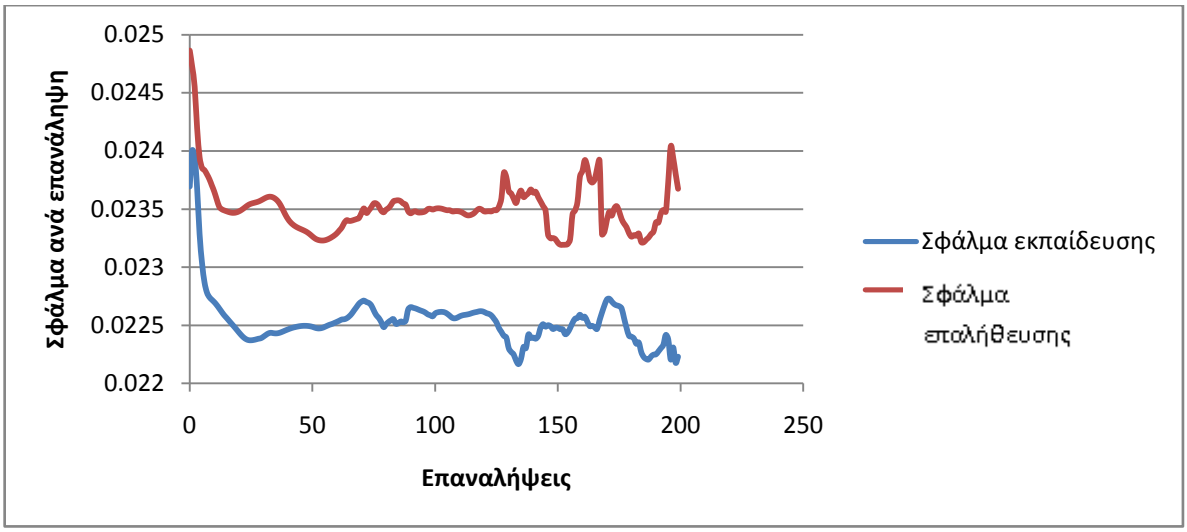


Σχήμα 6.13: *Γραφική παράσταση του Σφάλματος εκπαίδευσης σχετικά με διαφορετικούς συντελεστές μάθησης για το proteins, με σταθερό μέγεθος παραθύρου εισόδου 15*

Αφού υλοποιήθηκε το πείραμα με σταθερό παράθυρο εισόδου 15 και διαφορετικούς συντελεστές μάθησης, παρατηρήθηκε ότι η εκπαίδευση του δικτύου δεν ήταν καλύτερη. Συγκεκριμένα όταν ο συντελεστή μάθησης ήταν στο 0.2 το ποσοστό επιτυχίας ήταν 60.49%. Όπως παρατηρείται και από την γραφική παράσταση στο Σχήμα 6.13, στην περίπτωση αυτή το ποσοστό επιτυχίας είναι χειρότερο αφού αν η μέθοδος κατάβασης κλήσης εγκλωβιστεί σε κάποιο τοπικό ελάχιστο δεν μπορεί να βγει από αυτό. Αντίθετα, στην περίπτωση που ο συντελεστή μάθησης είναι 0.8, έχει αποτελέσματα 63.12%, ανάλογο με την περίπτωση όπου είναι 0.5, αφού σε αυτές τις περιπτώσεις η μέθοδος κατάβασης κλήσης μπορεί να ξεφύγει από τα τοπικά ελάχιστα. Αυτό από την γραφική παράσταση στο Σχήμα 6.13, αφού στις περιπτώσεις αυτές η καμπύλη μείωσης του σφάλματος δεν είναι ομαλή.

Βασισμένο στα πειράματα που έγιναν ήταν και το επόμενο πείραμα. Στο πείραμα αυτό διατηρήθηκε το παράθυρο εισόδου μεγέθους 15 και επιλέχτηκε η σταθερά ανάδρασης στο 0.2, αφού σε προηγούμενο πείραμα η παράμετρος αυτή με την συγκεκριμένη τιμή είχε αρκετά καλά αποτελέσματα. Το αποτέλεσμα της εκπαίδευσης του δικτύου παρατηρείται

στο Σχήμα 6.14. Το ποσοστό επιτυχίας του πειράματος αυτού ήταν 61.61%, κάτι το οποίο ήταν κατώτερο των προσδοκιών.



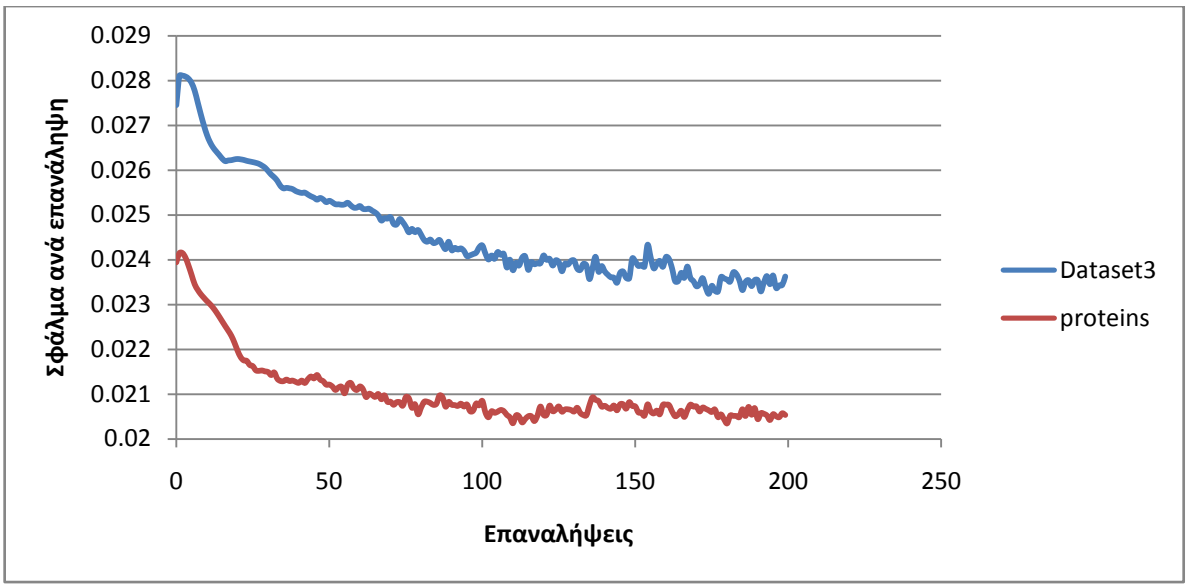
Σχήμα 6.14: *Γραφική παράσταση του πειράματος με σταθερά ανάδρασης 0.2 και σταθερό μέγεθος παραθύρου εισόδου 15*

Η παράμετροι για τις οποίες έπρεπε να γίνουν επιπλέον πειράματα ήταν αυτοί που αφορούσαν τν δομή του δικτύου. Τα κυριότερα πειράματα με τις παραμέτρους τους, που εκτελέστηκαν με σκοπό να εντοπιστεί η καλύτερη δυνατή δομή του δικτύου, παρουσιάζονται στον Πίνακα 6.2. Ο πίνακας αυτός παρουσιάζει πειράματα που εκτελέστηκαν με διαφορετικούς αριθμούν νευρώνων στα διάφορα κρυφά επίπεδα. Τα σημαντικότερα από αυτά τα πειράματα είναι η περίπτωση όπου υπάρχει μόνο ένα κρυφό επίπεδο στο κεντρικό νευρωνικό δίκτυο εμπρόσθιου περάσματος του Νευρωνικού Δικτύου αμφίδρομης ανάδρασης και η περίπτωση που υπάρχει μόνο ένα κρυφό επίπεδο στα δυο Νευρωνικά Επίπεδα με ανάδραση. Τα πειράματα αυτά είχαν ποσοστά επιτυχίας 60.22% και 61.43%, αντίστοιχα. Τα αποτελέσματα αυτά δεν θεωρούνται ικανοποιητικά. Ένα ακόμα πείραμα που θεωρείται πολύ σημαντικό είναι αυτό που βρίσκεται στην τρίτη στήλη πειραμάτων του Πίνακα 6.2 με το σχετικά ψηλό ποσοστό επιτυχίας 63.83%.

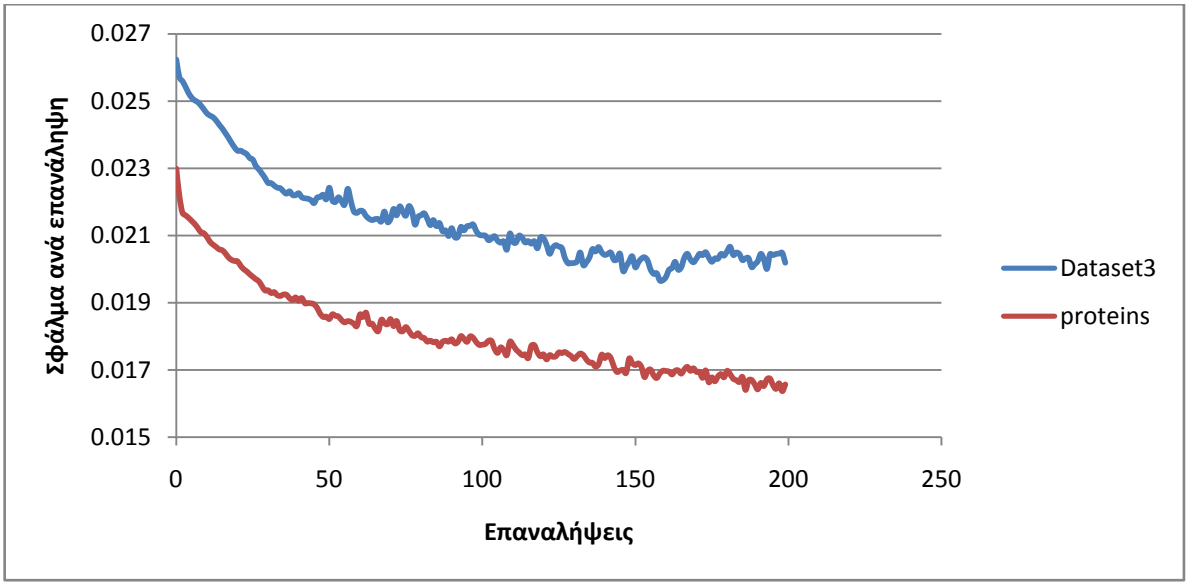
Όνομα Παραμέτρου	**Τιμές**					
Hidden_layer_one_size	14	14	12	11	16	20
Hidden_layer_two_size	14	14	12	11	0	20
Hidden_layer_one_of_Backward_size	8	11	13	10	11	12
Hidden_layer_two_of_Backward_size	8	0	12	8	9	11
Hidden_layer_one_of_Forward_size	8	11	13	10	11	12
Hidden_layer_two_of_Forward_size	8	0	12	8	9	11
Learning_Rate	0.5	0.5	0.5	0.5	0.5	0.5
Momentum	0.1	0.1	0.1	0.1	0.1	0.1
Window_size	15	15	15	15	15	15
q_minus_one	0.5	0.5	0.5	0.5	0.5	0.5
q_plus_one	0.5	0.5	0.5	0.5	0.5	0.5
s	3	3	3	3	3	3
Maximum_Iterations	200	200	200	200	200	200
Ποσοστό επιτυχίας Q3 (%)	**54.91**	**61.43**	**63.83**	**60.22**	**57.40**	**60.85**

Πίνακας 6.2: *Αποτελέσματα πειραμάτων που αφορούσαν την δομή του δικτύου*

Τέλος, η σειρά πειραμάτων περιλάμβανε πειράματα σχετικά με την κωδικοποίηση των δεδομένων, χρησιμοποιώντας κωδικοποίηση διανύσματος μεγέθους 5, και διαφορετική ποσότητα αμινοξέων ανά είσοδο, συγκεκριμένα τρία αμινοξέα. Τα πειράματα αυτά έγιναν για λόγους διευκρίνησης σχετικά με τον ρόλο τους στην εκπαίδευση του δικτύου. Οι παράμετροι του δικτύου που χρησιμοποιήθηκαν στο δίκτυο αυτό είναι αυτοί που αναγράφονται στον Πίνακα 6.1. Τα αποτελέσματα των πειραμάτων αυτών παρατηρούνται στα Σχήματα 6.15 και 6.16, χωρίς να έχουν να επιδείξουν κάτι αξιόλογο αφού κινούνται σε ποσοστό επιτυχίας από 53%-58%.



Σχήμα 6.15: *Γραφική παράσταση του Σφάλματος ανά επανάληψη για έλεγχο της κωδικοποίησης τύπου binary*



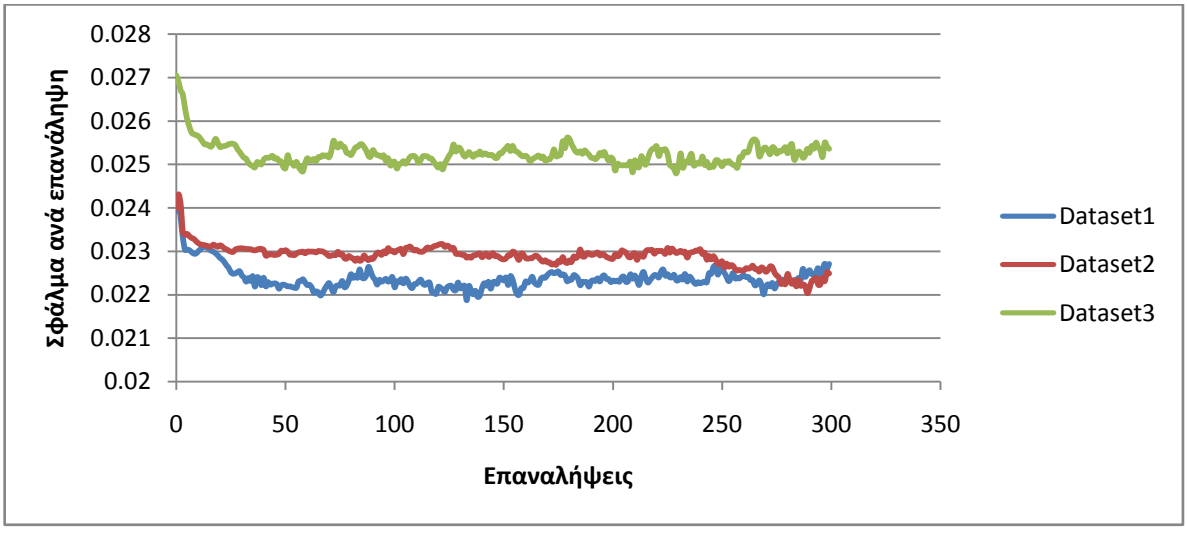
Σχήμα 6.16: *Γραφική παράσταση του Σφάλματος ανά επανάληψη για έλεγχο εισόδου ανά τρία αμινοξέα*

Σημαντικό θεωρείται επίσης το γεγονός ότι έγιναν πειράματα με περισσότερες επαναλήψεις, χωρίς όμως να παρατηρείται κάποια σημαντική αλλαγή στην εκπαίδευση του δικτύου μετά τις 200 επαναλήψεις.

6.2.2 Πειράματα σχετικά με σύνολα δεδομένων εισόδου

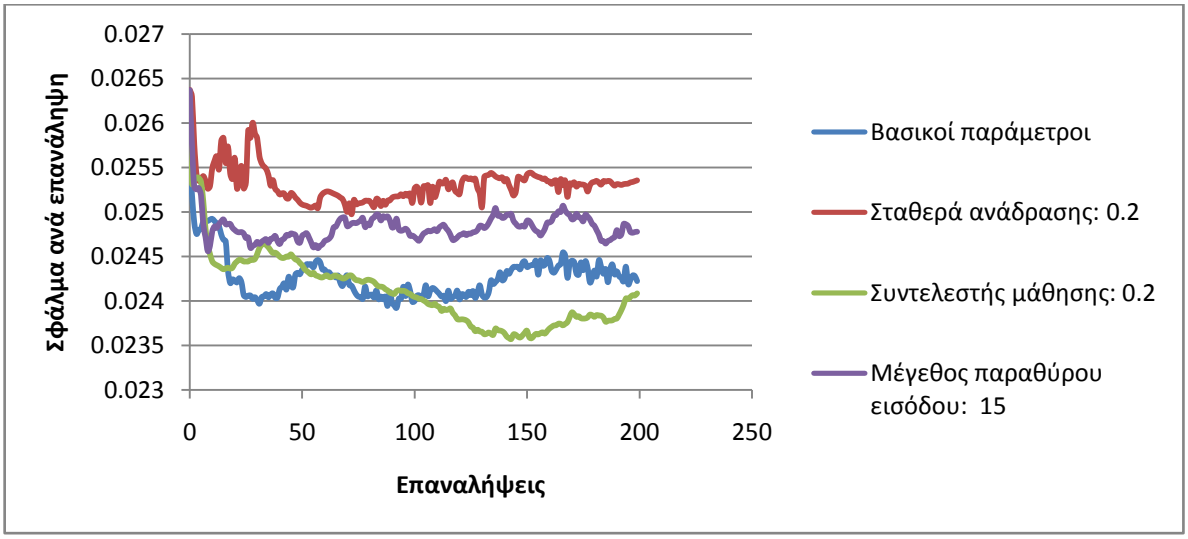
Τα σύνολα δεδομένων που χρησιμοποιούνται για την εκπαίδευση του κάθε νευρωνικού δικτύου έχουν σημαντικό ρόλο στην εκπαίδευσή του. Η σωστή επιλογή και προεπεξεργασία των συνόλων δεδομένων μπορεί να οδηγήσει στην σωστή εκπαίδευση ενός νευρωνικού δικτύου, όπως περιγράφεται στο Κεφάλαιο 4. Έτσι για την εκπαίδευση του Νευρωνικού Δικτύου αμφίδρομης ανάδρασης πρέπει να γίνει σωστή επιλογή των πρωτεϊνών για εκπαίδευση του δικτύου. Στη συγκεκριμένη έρευνα, για το λόγο ότι δεν υπήρχε αρκετός χρόνος για εκπαίδευση του δικτύου με μεγάλα σύνολα δεδομένων, δημιουργήθηκαν κάποια σχετικά μικρά σύνολα δεδομένων ούτως ώστε να εξαχθούν κάποια αρχικά συμπεράσματα για το δίκτυο.

Το πρώτο πείραμα, που αφορούσε τα σύνολα δεδομένων, εκτελέστηκε με σκοπό να διαφανεί κατά πόσο ιδίου μεγέθους σύνολα δεδομένων επιδρούν διαφορετικά στην εκπαίδευση του δικτύου. Σε αυτό το πείραμα χρησιμοποιήθηκαν τα σύνολα δεδομένων Dataset1, Dataset2 και Dataset3 (Υποκεφάλαιο 6.1) σε δίκτυο που έτρεχε με τις παραμέτρους που καθορίζονται στον Πίνακα 6.1. Τα αποτελέσματα ήταν αναμενόμενα και παρουσιάζονται στο Σχήμα 6.17. Τα σύνολα δεδομένων διαδραματίζουν διαφορετικό ρόλο ως προς την εκπαίδευση του Νευρωνικού Δικτύου αμφίδρομης ανάδρασης.



Σχήμα 6.17: *Γραφική παράσταση με σταθερές παραμέτρους και διαφορετικά σύνολα δεδομένων*

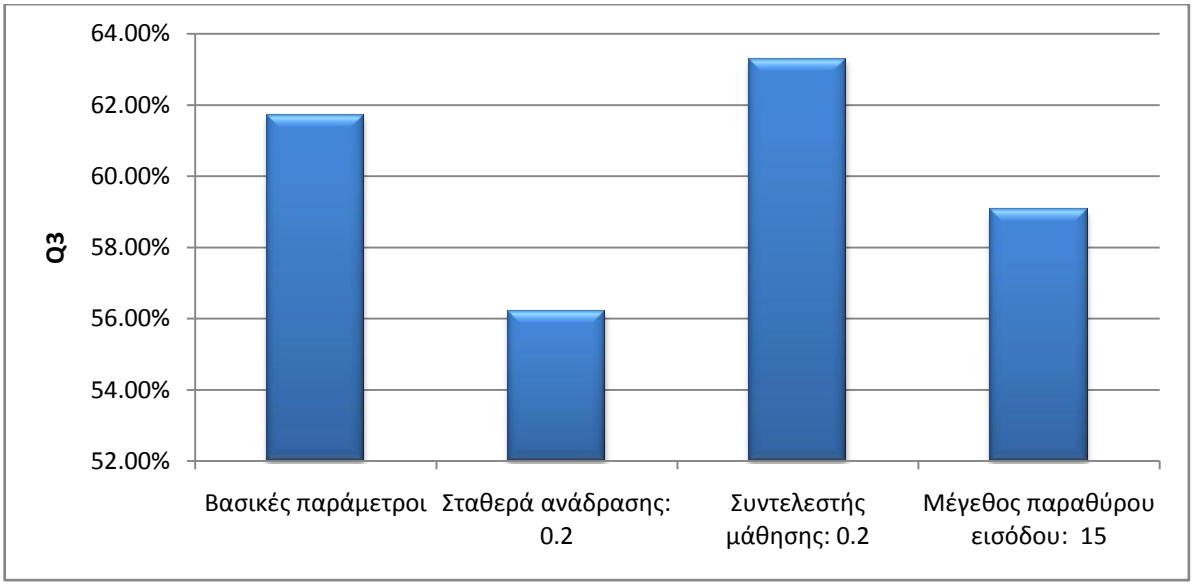
Μετά την εξαγωγή των αποτελεσμάτων του Υποκεφαλαίου 6.2.1 θεωρήθηκε απαραίτητο να εκτελεστούν κάποια πειράματα με σύνολα δεδομένων τα οποία θα είχαν μεγαλύτερα σύνολα εκπαίδευσης και δοκιμής. Με αυτό τον τρόπο θα μπορούσαν να εξαχθούν συμπεράσματα κατά πόσο το Νευρωνικό Δίκτυο αμφίδρομης ανάδρασης μπορεί να συλλέξει περισσότερες πληροφορίες και να εκπαιδευτεί καλύτερα μέσα από μεγαλύτερο εύρος διαφορετικών πρωτεϊνικών ακολουθιών. Για το πείραμα αυτό χρησιμοποιήθηκε το σύνολο δεδομένων που κατασκευάστηκε με 600 πρωτεΐνες για εκπαίδευση του δικτύου και 200 πρωτεΐνες για δοκιμή του δικτύου, οι οποίες δεν περιείχαν ελλιπή δεδομένα. Συγκεκριμένα εκτελέστηκαν τέσσερα πειράματα με παραμέτρους οι οποίες βασίζονταν στα στοιχεία του Πίνακα 6.1. Το πρώτο πείραμα ακολουθούσε πιστά τα στοιχεία του Πίνακα 6.1, ενώ τα υπόλοιπα τροποποιήθηκαν ανάλογα με τις περιπτώσεις που υπήρχαν σχετικά ψηλά ποσοστά επιτυχίας, στα πειράματα του Υποκεφαλαίου 6.1. Αυτές οι περιπτώσεις ήταν με σταθερά ανάδρασης 0.2, συντελεστή μάθησης 0.2 και μέγεθος παραθύρου εισόδου 15, αντίστοιχα.



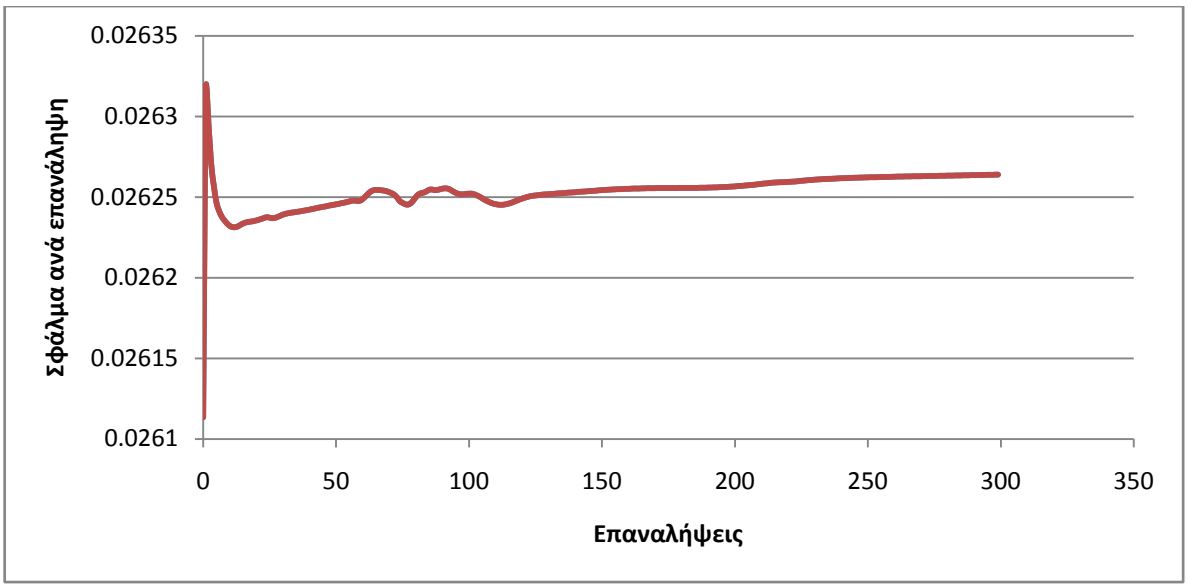
Σχήμα 6.18: *Γραφική παράσταση του Σφάλματος ανά επανάληψη με σύνολο δεδομένων πρωτεϊνών το οποίο έχει 600 πρωτεΐνες για εκπαίδευση και 200 πρωτεΐνες για δοκιμή*

Η γραφική παράσταση στο Σχήμα 6.18 παρουσιάζει τα αποτελέσματα των πειραμάτων αυτών. Πιο συγκεκριμένα παρατηρούμε τα ποσοστά επιτυχίας των πειραμάτων αυτών στο Σχήμα 6.19. Τα αποτελέσματα αυτά συγκρινόμενα με τα αντίστοιχα των αποτελεσμάτων του Dataset3, Υποκεφάλαιο 6.1, παρατηρείται ότι έχουν ψηλότερα ποσοστά επιτυχίας. Τα αποτελέσματα συγκρίνονται με αυτά του Dataset3 διότι ούτε αυτό περιέχει ελλιπή δεδομένα.

Το πείραμα αυτό οδηγεί στο συμπέρασμα ότι μεγαλύτερα σύνολα δεδομένων κατά το στάδιο της εκπαίδευσης του δικτύου, οδηγούν σε ψηλότερα ποσοστά επιτυχίας. Αυτό συμβαίνει διότι η πρωτοταγής δομή των πρωτεϊνών έχει εκατομμύρια συνδυασμούς στην πρωτοταγή δομή, που αντιστοιχούν σε συγκεκριμένα μοτίβα της δευτεροταγούς δομής. Έτσι αν χρησιμοποιηθεί στην εκπαίδευσή του δικτύου μεγάλο σύνολο δεδομένων από το οποίο θα μπορέσει το δίκτυο να δει και να γενικεύσει μεγαλύτερη ποσότητα πληροφοριών, θα μπορέσει να έχει μεγαλύτερες πιθανότητες να προβλέψει την δευτεροταγή δομή άγνωστων προς αυτό πρωτεϊνικών ακολουθιών.



Σχήμα 6.19: *Γραφική παράσταση των ποσοστών επιτυχίας με σύνολο δεδομένων πρωτεϊνών το οποίο έχει 600 πρωτεΐνες για εκπαίδευση και 200 πρωτεΐνες για επαλήθευση*



Σχήμα 6.20: *Γραφική παράσταση του Σφάλματος ανά επανάληψη με σύνολο δεδομένων πρωτεϊνών στο οποίο τα ελλιπής δεδομένα αντικαταστάθηκαν με τυχαίες τιμές*

Σημαντικό ρόλο στην εκπαίδευση ενός Νευρωνικού δικτύου αποτελεί ο τρόπος με τον οποίο γίνεται χειρισμός των ελλιπών δεδομένων. Στα πειράματα που εκτελέστηκαν στο Υποκεφάλαιο 6.1 με το σύνολο δεομένων proteins, τα ελλιπή δεδομένα παρουσιάζονταν στο δίκτυο ως ελλιπή δεδομένα και έχουν καλύτερα αποτελέσματα από τα σύνολα δεδομένων όπου δεν υπάρχουν άγνωστες τιμές. Ο λόγος που συμβαίνει αυτό εξηγείται στο άνωθεν υποκεφάλαιο. Με βάση τις παρατηρήσεις αυτές εκτελέστηκε ένα πείραμα όπου το σύνολο δεδομένων που χρησιμοποιήθηκε αντικαθιστούσε όλες τις ελλιπείς τιμές με τυχαία δεδομένα. Το δίκτυο αυτό χρησιμοποίησε παραμέτρους αυτές που καθορίζει ο Πίνακας 6.1.

Τα αποτελέσματα του πειράματος αυτού ήταν άκρως απογοητευτικά. Η γραφική παράσταση, στο Σχήμα 6.20, δείχνει ότι το δίκτυο δεν μπορεί μειώσει το σφάλμα του για να εκπαιδευτεί. Αυτό πιθανόν να οφείλεται στο γεγονός ότι τα τυχαία δεδομένα δημιουργούν ακόμα περισσότερα μοτίβα πρωτοταγής δομής στο δίκτυο, και ανάλογα τοπικά ελάχιστα με αποτέλεσμα το δίκτυο να μην έχει την δυνατότητα να εκπαιδευτεί.

6.3 Γενικές παρατηρήσεις

Μετά την εκτέλεση διαφόρων πειραμάτων διαφαίνονται μέσα από τα αποτελέσματα οι δυνατότητες του Νευρωνικού Δικτύου αμφίδρομης ανάδρασης που κατασκευάστηκε. Μπορεί να διαφανεί το ποσοστό επιτυχίας που μπορεί να έχει αλλά και ο εντοπισμός διαφόρων προβλημάτων που μπορούν να βελτιστοποιηθούν. Το Νευρωνικό Δίκτυο αμφίδρομης ανάδρασης που κατασκευάστηκε μπορεί να έχει ποσοστό επιτυχίας περίπου 64%. Το ποσοστό αυτό θεωρείται σχετικά μικρό, παρόλα αυτά έχει προοπτικές βελτίωσης αφού παρατηρείται μέσα από τα αποτελέσματα και την ανάλυσή τους ότι το δίκτυο μαθαίνει.

Το δίκτυο μέσα από τα διάφορα πειράματα που έγιναν μειώνει το σφάλμα εκπαίδευσης και αυξάνει το ποσοστό επιτυχίας όταν το παράθυρο εισαγωγής δεδομένων έχει μέγεθος 15 ή όταν η σταθερά ανάδρασης είναι 0.2. Το πρώτο δεδομένο οδηγεί στο συμπέρασμα ότι το δίκτυο με ανάδραση δεν μπορεί να κρατά στη μνήμη του μεγάλες ακολουθίες, κάτι το οποίο έρχεται σε αντίθεση με το πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, όπως εξηγείται στο Υποκεφάλαιο 6.2. Το δεύτερο δεδομένο καταδεικνύει ότι το δίκτυο για να έχει σχετικά ψηλό ποσοστό επιτυχίας πρέπει τα δεδομένα που επιστρέφονται στην είσοδο του δικτύου για να υπολογιστεί το νέο αποτέλεσμα δεν πρέπει να έχουν μεγάλη διάρκεια στον χρόνο. Σημαντική θεωρείται επίσης η επισήμανση ότι για να δίνει ψηλό ποσοστό επιτυχίας το δίκτυο πρέπει να υπάρχουν στα δεδομένα εκπαίδευσης του δικτύου πρωτεΐνες από μεγάλο αριθμό διαφορετικών πρωτεϊνικών ακολουθιών, ώστε να μπορεί το δίκτυο να διαμορφώσει ανάλογα τα όρια αποφάσεων του στον τρισδιάστατο χώρο.

Το σημαντικότερο πρόβλημα που εντοπίζεται μέσα από τα αποτελέσματα του δικτύου και την ανάλυσή τους είναι ότι ο μεγάλος όγκος δεομένων που περιέχεται στις πρωτεϊνικές ακολουθίες, τα διαφορετικά μοτίβα που οδηγούν σε συγκεκριμένες μορφές δευτεροταγούς δομής, δυσκολεύει την εκπαίδευση του δικτύου ενώ δημιουργεί πολλά τοπικά ελάχιστα, στα οποία εύκολα καταλήγει η μέθοδος κατάβασης κλήσης. Αποτέλεσμα του γεγονότος αυτού είναι ότι δύσκολα εντοπίζεται το ολικό ελάχιστο το οποία θεωρητικά θα δώσει τα υψηλότερα ποσοστά επιτυχίας.

Στα προβλήματα που εντοπίστηκαν δίνονται λύσεις μέσα από το Υποκεφάλαιο 6.2, κατά την ανάλυση των αποτελεσμάτων και το Υποκεφάλαιο 7.2, που παρουσιάζει την μελλοντική εργασία. Ο Πίνακας 6.3 παρουσιάζει των συνδυασμό των παραμέτρων που οδήγησαν στα καλύτερα αποτελέσματα του δικτύου.

Όνομα Παραμέτρου	**Τιμές**		
Hidden_layer_one_size	12	12	12
Hidden_layer_two_size	12	12	12
Hidden_layer_one_of_Backward_size	11	11	13
Hidden_layer_two_of_Backward_size	9	9	12
Hidden_layer_one_of_Forward_size	11	11	13
Hidden_layer_two_of_Forward_size	9	9	12
Learning_Rate	0.5	0.5	0.5
Momentum	0.1	0.1	0.1
Window_size	15	27	15
q_minus_one	0.5	0.2	0.5
q_plus_one	0.5	0.2	0.5
s	3	3	3
Maximum_Iterations	200	200	200
Ποσοστό επιτυχίας Q3 (%)	**63.75**	**63.20**	**63.83**

Πίνακας 6.3: *Αποτελέσματα πειραμάτων που αφορούσαν την δομή του δικτύου*

Κεφάλαιο 7

Συμπεράσματα και μελλοντική εργασία

7. 1 Γενικά Συμπεράσματα

Η έρευνα που εκπονήθηκε αφορά το πρόβλημα της πρόβλεψης δευτεροταγούς δομή πρωτεϊνών, ένα πρόβλημα που αφορά κατά κύριο λόγο τις επιστήμες της Βιολογίας και της Πληροφορικής. Για την εξεύρεση λύσης στο πρόβλημα αυτό ερευνήθηκαν διάφορες μεθόδοι και αλγόριθμοι από την επιστήμη της Πληροφορικής και συγκεκριμένα από τα Νευρωνικά Δίκτυα. Υλοποιήθηκε ένα Νευρωνικού Δικτύου με αμφίδρομη ανάδραση το οποίο μπορεί να προβλέψει την δευτεροταγή δομή πρωτεϊνών, δεχόμενο στην είσοδό του μόνο την πρωτοταγή δομή τους. Το νευρωνικό δίκτυο που υλοποιήθηκε έχει σαν ψηλότερο ποσοστό επιτυχής πρόβλεψης δευτεροταγούς δομής πρωτεϊνών το 64%.

Το Νευρωνικό δίκτυο αντιμετωπίζει δυσκολίες ως προς την εκπαίδευση του λόγω κυρίως της πολυπλοκότητας των δεδομένων που δέχεται στην είσοδό του. Η σωστή προεπεξεργασία και επιλογή των δεδομένων εισόδου του δικτύου καθώς και ο περεταίρω εμπλουτισμός του δικτύου με τεχνικές που αναφέρονται κατά την ανάλυση των αποτελεσμάτων (Υποκεφάλαιο 6.2) και την μελλοντική εργασία (Υποκεφάλαιο 7.2). Αναλογιζόμενοι την φύση του προβλήματος και τον τρόπο που λειτουργεί το Νευρωνικό Δίκτυο αμφίδρομης ανάδρασης η λύση αυτή φαντάζει ιδανική, ως επακόλουθο οι τεχνικές αυτές μπορούν να βελτιώσουν τα ποσοστά επιτυχίας στην πρόβλεψη δευτεροταγούς δομής.

Το Νευρωνικό Δίκτυο με αμφίδρομη ανάδραση υλοποιήθηκε, έχοντας αρχικά αποτελέσματα μέχρι και 64% επιτυχία. Τα αποτελέσματα αυτά θεωρούνται ικανοποιητικά σε αρχική φάση, αναλογιζόμενοι το γεγονός ότι η καλύτερη επίδοση σε αντίστοιχες έρευνες ήταν 76% σε επίσης Νευρωνικό Δίκτυο με αμφίδρομη ανάδραση. Τα αρχικά αποτελέσματα που εξάχθηκαν από το δίκτυο που υλοποιήθηκε απέχουν 4-10% από τις καλύτερες σε επιδώσεις μεθόδους πρόβλεψης δευτεροταγούς δομής πρωτεϊνών που αναφέρονται στο Υποκεφάλαιο 1.2. Με βάση αυτό το γεγονός πρέπει να προχωρήσει η έρευνα γύρω από το Νευρωνικού Δικτύου με αμφίδρομη ανάδραση για την βελτίωση των αποτελεσμάτων του.

Ένας προβληματισμός που δημιουργείται γύρω από το πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών είναι το αν όντως μπορεί να λυθεί το συγκεκριμένο πρόβλημα. Η δευτεροταγής δομή των πρωτεϊνών στην πραγματικότητα δεν είναι τίποτε

άλλο από μια δημιουργία του ανθρώπου, ως ενδιάμεσο στάδιο από την πρωτεϊνική ακολουθία στην τρισδιάστατη δομή των πρωτεϊνών. Το ερώτημα που τίθεται είναι αν υπάρχει μια λογική αντιστοιχία μεταξύ της πρωτοταγούς και της δευτεροταγούς δομής των πρωτεϊνών την οποία να μπορεί να εντοπίσει κάποιο νευρωνικό δίκτυο, ώστε να δημιουργήσει τα σωστά όρια αποφάσεων και να λύσει το συγκεκριμένο πρόβλημα.

7.2 Μελλοντική εργασία

Η συγκεκριμένη έρευνα αφορά ένα σημαντικό πρόβλημα γύρω από τους κλάδους της Πληροφορικής και της Βιολογίας. Με βάση τη μελέτη και τα αποτελέσματα από την συγκεκριμένη εργασία μπορούν να ερευνηθούν πολλές διεργασίες οι οποίες μπορεί να αποφέρουν καλύτερα αποτελέσματα.

Βασικό μειονέκτημα της συγκεκριμένη εργασίας ήταν η έλλειψη χρόνου για περεταίρω πειράματα στο υφιστάμενο Νευρωνικό Δίκτυο. Κρίνεται απαραίτητη η καταγραφή και εκτέλεση διάφορων πειραμάτων τα οποία θα βασίζονται στις παραμέτρους, οι οποίες θα καθορίσουν το μέγιστο της απόδοσης του Νευρωνικού Δικτύου αμφίδρομης ανάδρασης που υλοποιήθηκε. Επίσης στο σημείο αυτό πρέπει να αναφερθεί ότι τα πειράματα αυτά πρέπει να εκτελεστούν αρκετές φορές ώστε να δημιουργείται ένας μέσος όρος αποτελεσμάτων για το κάθε ένα. Αυτή η διαδικασία θεωρείται απαραίτητη αφού η εκπαίδευση Νευρωνικών Δικτύων μπορεί σε κάποια πειράματα να οδηγείται σε τοπικά ελάχιστα με αποτέλεσμα να μην γίνεται σωστά. Σημαντικό θεωρείται επίσης να εκτελεστούν πειράματα με μεγαλύτερα σύνολα δεδομένων πρωτεϊνών, κάτι το οποίο είναι εξαιρετικά χρονοβόρο αλλά απαραίτητο. Το Νευρωνικό Δίκτυο μέσα από μεγαλύτερα σύνολα δεδομένων μπορεί να συλλέξει πληροφορίες οι οποίες να οδηγούν σε πολύ καλύτερα αποτελέσματα. Στην διαδικασία αυτή μπορεί να είναι αρκετά χρήσιμο το Grid (Foster et al 2001), το οποίο είναι ένα σύστημα Ερευνητικού Επιπέδου. Το σύστημα αυτό επιτρέπει την αποστολή του προγράμματος μέσο διαδικτύου σε συστήματα πιο ισχυρά από συνηθισμένους ηλεκτρονικούς υπολογιστές, τα χαρακτηριστικά των οποίων καθορίζει ο χρήστης. Με αυτό τον τρόπο μπορεί να εκτελεστεί αρκετά μεγάλος αριθμός πειραμάτων ταυτόχρονα.

Όσον αφορά το υφιστάμενο Νευρωνικό Δίκτυο, μπορούν επίσης να γίνουν κάποιες αλλαγές σε αυτό ώστε να ελεγκτεί κατά πόσο αυτές βοηθούν στην εκπαίδευσή του. Σε μερικά πειράματα ήταν φανερό ότι η μέθοδος κατάβασης κλίσης οδηγούσε σε τοπικά ελάχιστα, κάτι το οποίο μπορεί να βελτιωθεί χρησιμοποιώντας αλγόριθμους adaptive learning rate καθώς και η μέθοδος του early stopping (Rost and Sander 1993) ούτως ώστε όταν η μέθοδος κατάβασης κλήσης εντοπίσει το ολικό ελάχιστο να μην φεύγει από αυτό. Επίσης μπορεί να ερευνηθεί κατά πόσο το δίκτυο μπορεί να επιφέρει διαφορετικά αποτελέσματα μέσω διαφορετικών συναρτήσεων σφάλματος, όπως η relative entropy συνάρτηση (Baldi et al 1999), αφού αυτό προσφέρεται μέσα από τις παραμέτρους του Νευρωνικού Δικτύου που υλοποιήθηκε. Μπορούν ακόμα να γίνουν δοκιμές κατά πόσο επηρεάζουν τα αποτελέσματα διαφορετικές κωδικοποιήσεις εισόδου και εξόδου, όπως κωδικοποίησης δεκαδικού αριθμού. Σε αυτή την περίπτωση μπορεί να εφαρμοστεί επίσης αλγόριθμος συμπίεσης δεδομένων, αφού η δευτεροταγής δομή του κάθε αμινοξέως εξαρτάται από μεγάλο αριθμό αμινοξέων που προηγούνται και έπονται αυτού. Πειράματα μπορούν επίσης να γίνουν για δοκιμή διαφορετικών κατηγοριών στην έξοδο, όπως οι οκτώ κατηγορίες του DSSP (DSSP, 20 April 2009).

Μια διαδικασία η οποία μπορεί να επιφέρει καλύτερα αποτελέσματα είναι η επιπλέον προεπεξεργασία των συνόλων δεδομένων. Όπως παρατηρείται από τα αποτελέσματα είναι σημαντικό να υπάρχουν στα δεδομένα εκπαίδευσης του δικτύου διαφορετικής ομοιότητας πρωτεΐνες. Καλύτερα αποτελέσματα ίσως υπήρχαν αν δημιουργείται κάποιο πρόγραμμα το οποίο μπορούσε να ομαδοποιήσει τις πρωτεΐνες με βάση τα χαρακτηριστικά τους, όπως κάποια υλοποίηση μη επιβλεπόμενης μάθησης Kohonen, και στη συνέχεια να οδηγούσε τις πρωτεΐνες σε νευρωνικά δίκτυα αμφίδρομης ανάδρασης τα οποία είναι ανάλογα εκπαιδευμένα. Κάτω από την ίδια λογική, θα μπορούσε να δημιουργηθεί κάποιο πρόγραμμα το οποίο χρησιμοποιώντας πολλαπλή στοίχιση (multiple alignment) (Rost and Sander 1993) θα εντόπιζε ομοιότητες στις ακολουθίες των πρωτεϊνών και θα τις κατηγοριοποιούσε. Κάτι το οποίο θα μπορούσε να χρησιμοποιηθεί ώστε η κάθε κατηγορία να χρησιμοποιείται ξεχωριστά για την εκπαίδευση διαφορετικών Νευρωνικών Δικτύων αμφίδρομης ανάδρασης. Στη συνέχεια, κατά την χρήση του όλου συστήματος, η κάθε νέα πρωτεΐνη περνώντας από το πρόγραμμα που χρησιμοποιεί πολλαπλή στοίχιση (multiple

alignment) θα ξέραμε σε πια κατηγορία ανήκει και ανάλογα θα περνά από το κατάλληλο Νευρωνικών Δικτύων αμφίδρομης ανάδρασης ώστε να προβλεφτεί η δευτεροταγής δομή της.

Όσον αφορά τα δεδομένα εισόδου του συστήματος και την πρωτοταγή δομή των πρωτεϊνών, σημαντικό ρόλο στην αναδίπλωσή της στο χώρο έχουν η πολικότητα των πλευρικών αλυσίδων του κάθε αμινοξέως. Ίσως αυτή η πληροφορία να μπορούσε να χρησιμοποιηθεί σε μελλοντικό στάδιο και να παρέχεται στο δίκτυο, μέσω των δεδομένων εισόδου του συστήματος, μαζί με την πρωτοταγής δομή των πρωτεϊνών.

Μια άλλη βελτιστοποίηση είναι η δημιουργία δεύτερου επιπέδου επεξεργασίας της εξόδου του νευρωνικού δικτύου. Αυτό το νευρωνικό δίκτυο θα προσπαθεί να βελτιστοποιήσει τα δεδομένα εξόδου του πρώτου Νευρωνικών Δικτύων αμφίδρομης ανάδρασης με βάση τα τις δευτεροταγείς δομές που προηγούνται και έπονται της δευτεροταγής δομής του αμινοξέως η οποία θα έχει προβλεφθεί. Αυτή η υλοποίηση υπάρχει ήδη με τον αλγόριθμο ανάστροφής μετάδοσης λάθους (Chen and Chaudhari 2007), αλλά ένας πιο αποδοτικός αλγόριθμος στο δεύτερο επίπεδο ίσως να αποκόμιζε καλύτερα αποτελέσματα. Αυτός ο αλγόριθμος θα μπορούσε να είναι ο Conjucate Gradient (Chang and Mark 1999) ή ο Scaled Conjucate Gradient (Moller 1993), οι οποίοι είναι πολύ πιο αποδοτικοί σε περιπτώσεις που υπάρχει μεγάλος όγκος δεδομένων, όπως στο πρόβλημα πρόβλεψης δευτεροταγούς δομής πρωτεϊνών, ενώ ταυτόχρονα είναι πολύ πιο ταχείς. Ο αλγόριθμος αυτός θα χρησιμοποιηθεί σε μελλοντική έρευνα και για το πρώτο επίπεδο Νευρωνικού Δικτύου αμφίδρομης ανάδρασης. Ένας άλλος αλγόριθμος εκπαίδευσης που μπορεί να χρησιμοποιηθεί και είναι ταχύτερος από την μέθοδο ανάστροφης μετάδοσης του λάθους είναι ο Quickpropt TT. Στο ίδιο σύστημα, μπορούν να χρησιμοποιηθούν αλγόριθμοι βελτιστοποίησης, όπως οι γενετικοί αλγόριθμοι ή η έννοια τις ασάφειας, και μέσα από τα ανάλογα πειράματα να εξαχθούν συμπεράσματα κατά πόσο μπορούν να προσεγγίσουν καλύτερα αποτελέσματα. Στην περίπτωση των γενετικών αλγορίθμων μπορούν να ορισθούν σε κάθε χρωμόσωμα όλες οι παράμετροι του δικτύου και σε κάθε γενεά να επιλέγονται οι παράμετροι που δημιουργούν το δίκτυο με το υψηλότερο ποσοστό επιτυχίας. Μέσα από τη διαδικασία αυτή μπορούν να εντοπιστούν οι καλύτερες παράμετροι για το δίκτυο.

Αναφορές

An Information Portal to Biological Macromolecular Structures, April 20, 2009, http://www.rcsb.org/pdb/search/advSearch.do?st=SequenceQuery

Armano G., Orro A. and Vargiu E. *''MASSP3: A Systemfor Predicting Protein Secondary Structure''*, EURASIP Journal on Applied Signal Processing, Vol. 2006, Article ID 17195, (2006) : 1–9.

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

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

Bengio Y., Simard P. and Frasconi P. *"Learning long-term dependencies with gradient descent is difficult"*, IEEE Trans, Neural Networks 5, (2004) : 157-166.

Blazewicz J., Hammer L.P. and Lukasiak P. *''Predicting Secondary Structures of Proteins Recognizing Properties of mino Acids with the Logical Analysis of Data Algorithm''*, IEEE Engineering in medicine and biology magazine, May/June (2005) : 88-94.

Branting K., Lead Artificial Intelligence Engineer, May 20, 2009, http://www.karlbranting.net/papers/plummer/Paper_7_12_00_files/image016.jpg

Chang W. and Mark M. *"A conjugate gradient learning algorithm for recurrent neural networks"*, Neurocomputing 24 (1999) : 173-189.

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 (2007) : 572-582.

Cuff J.A. and Barton G.J. *''Evaluation and Improvement of Multiple Sequence for Protein Secondary Structure Prediction"*, Proteins, 34, (1999) : 508-519.

DSSP, April 20, 2009, http://swift.cmbi.kun.nl/gv/dssp/

Elman L.J. ‘‘Finding Structure in Time’’ Cognitive Science, 14, (1990) : 179-211.

EVA: PDB sequence unique list, May 14, 2009 http://cubic.bioc.columbia.edu/eva/res/unique_list.html

EVA measures for secondary structure prediction accuracy, April 20, 2009, http://cubic.bioc.columbia.edu/eva/doc/measure_sec.html

Foster I.,Kesselman C. and Tuecke S. “The Anatomy if the Grid”, Intl J. Supercomputer Applications (2001).

Frishman D. and Argos P. *‘‘Knowledge-based Secondary Structure Assignment’’*, Proteins, 23, (1995) : 566-579.

Frishman D. and Argos P. *“Knowledge-based Secondary Structure Assignment”*, Proteins, 23 (1995) : 566-579.

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

King R.D. and Sternberg M.J.E. *‘‘Identification and Application of the Concepts Important for Accurate and Reliable Protein Secondary Structure Prediction’’*, Protein Science, 5, (1996) : 2298–2310.

Kabsch D. and Sander C. *‘‘Dictionary of Protein Secondary Structure: Pattern Recognition of Hydrogen-Bonded and Geometrical Features’’*, Biopolymers, Vol.22, (1983) : 2577-2637.

Kolmogorov A.N. *‘‘On the Representations of Continuous Functions of Many Variables by Superpositions of Continuous Functions of one Variable and Addition’’* Docklady Akademii Nauk, USSR, 114(5), (1957) : 953-956.

Madison Area Technical Collage, April 20, 2009, http://matcmadison.edu/biotech/resources/proteins/labManual/images/220_04_114.png

McCulloch W.S. and Pitts W. *''A Logical Calculus of the Ideas Immanent in Nervous Activity''*, Bulletin of Mathematical Biophysics, 5, (1943) : 115-133.

Moller M. *"A Scaled conjugate gradient algorithm for fast supervised learning"*, Neural Networks, Vol 6 (1993) : 525 – 533.

Mozer C.M *''A Focused Backpropagation Algorithm for Temporal Pattern Recognition''*, Complex Systems, 3, (1989) : 349-381.

Pdb_Select dataset, April 20, 2009, http://bioinfo.tg.fh-giessen.de/pdbselect/recent.pdb_select25

Petsko J.J. and Ringe D. *"Proteins Structure and Function"*, New Science Press, London, (2004).

PIR, Abbr. Protein Information Resource, April 20, 2009, http://pir.georgetown.edu/

Promponas B. *"Bioinformatics Notes"*, Athens 2004-2005.

Qian N. and Sejnowski T.J. *"Predicting the Secondary Structure of Globular Proteins Using Neural Network Models"*, J.Mol.Biol, 202, (1988) : 71-84.

RCSB Protein Data Bank, April 20, 2009, http://www.rcsb.org/pdb/home/home.do

Richards F.M. and Kundrot C.E *''Identification of Structural Motifs from Proteins Coordinate Data: Secondary Structure and First-level Supersecondary Structure''*, Proteins, 3, (1988) : 71-84.

Rost B. and Sander C. *"Improved Prediction of Protein Secondary Structure by Use of Sequence Profiles and Neural Networks"*, Proc.Natl.Acad.Sci, 90, (1993) : 7558-7562.

Rumelhart D.E., Hinton G.E. and Williams R.J. *''Learning Internal Representation by Error Propagation. In Rumelhart D. E. and McClelland J. L. (Eds), Parallel Distributed Processing: Explorations in the Microstructure of Cognition''*, MIT Press, Cambridge, MA, Vol. 1 (1986) : 318-362.

Salamov A.A. and Soloveyev V.V. *''Protein Secondary Structure Prediction Using Local Alignments''*, Journal of Molecular Biology, 268, (1997) : 31–36.

The ASTRAL Compendium for Sequence and Structure Analysis, April 20, 2009, http://astral.berkeley.edu/

The PDBSELECT database, April 20, 2009, http://swift.cmbi.kun.nl/swift/pdbsel/

Werbos J.P. ''*Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences''* PhD thesis, Harvard University, Cambridge, MA (1974).

Werbos J.P. ''*Backpropagation Through Time: What It Does and How to Do It''* Proceeding of the IEEE, Vol. 28, No. 10, (1990) : 1550-1560.

Widrow B. and Hoff J.M.E. *''Adaptive switching circuits''*, IRE WESCON Convention Record, (1960) : 961-1104.

Won K.J., Hamelryck T., Prügel-Bennett A. and Krogh S *''An Evolutionary Method for Learning HMM Structure: Prediction of Protein Secondary Structure''*, BMC Bioinformatics, Vol. 8, (2007) : 357-381.

Yuksektepe U.F., Yılma O. and Turkay M. *''Prediction of secondary structures of proteins using a two-stage method''*, Computers and Chemical Engineering, 32 (2008) : 78–88.

Γενική Βιβλιογραφία

Bishop C. M. "Neural Networks for Pattern Recognition", Oxford, UK (2005).

Brooks, C. L., Kurplus, M. and Pettitt, B. M. "PROTEINS: A Theoretical Perspective of Dynamics, Structure, and Thermodynamics", John Wiley and Sons, Canada (1990).

Creighton, T. E. "PROTEINS: Structures and Molecular Properties", Second Edition, W. H. Freeman and Company, New York (1996).

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

Taylor J.G and Mannion C.L.T. “Theory and Applications of Neural Networks”, Springer-Verlag, UK (1990)

Παράρτημα Α

Νευρωνικό Δίκτυο Αμφίδρομης Ανάδρασης

PSSP_UCY.cpp

```
#pragma once
//included libraries
#include "stdafx.h"
#include "DataReader.h"
#include "Neuron.h"
#include "BidirectionalRecurrentNeuralNetwork.h"

int _tmain(int argc, _TCHAR* argv[])
{
	//initiate random function
	srand(time(NULL));

	//vaariables of the main program
	char trainFile[100];
	char testFile[100];
	char inputProfile[100];
	char outputProfile[100];
	int primaryStructureSize;
	float parameters[15];
	int epoch=0;
	bool endOfFile=true;

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

	//Parameters' entry
	getInput.readParameters(parameters,inputProfile,outputProfile,train
File,testFile);

	//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 activationFuncType = parameters[6];
	double learningRate = parameters[7];
	double momentum = parameters[8];
	int windowSize = parameters[9];
	double qMinus1 = parameters[10];
	double qPlus1 = parameters[11];
	int errorFunctionType = parameters[12];
	int s = parameters[13];
	int maxIterations = parameters[14];

	//creation of the Bidirectional Recurrent Neural Network with the
specific parameters
	BidirectionalRecurrentNeuralNetwork
BRNN=BidirectionalRecurrentNeuralNetwork(hLayerOneSize,

	hLayerTwoSize,hLayerOneSizeB,hLayerTwoSizeB,hLayerOneSizeF,
```

```
      hLayerTwoSizeF,activationFuncType,learningRate,momentum,windowSize,

      qMinus1,qPlus1,errorFunctionType,s,maxIterations,trainFile,testFile,
                  inputProfile,outputProfile);

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

            //get training sequence
            endOfFile=BRNN.initTrainingInput(&primaryStructureSize);

            //training loop
            while(endOfFile)
            {
                  //the input of a sequence
                  for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
                  {
                        //feedforward and backpropagation for a moving
window
                        BRNN.doFeedForward(sequencet,1,epoch);
                        BRNN.doBackpropagation(sequencet);
                  }

                  //get training error
                  BRNN.getSequenceTrainingError();

                  //get training sequence
                  endOfFile=BRNN.initTrainingInput(&primaryStructureSize);
            }

            //get testing sequence
            endOfFile=BRNN.initTestInput(&primaryStructureSize);

            //testing loop
            while(endOfFile)
            {
                  //the input of a sequence
                  for(int
sequencet=0;sequencet<primaryStructureSize;sequencet++)
                  {
                        //feedforward for a moving window
                        BRNN.doFeedForward(sequencet,2,epoch);
                  }

                  //get testing error
                  BRNN.getSequenceTestingError();

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

                  //get testing sequence
```

```
                endOfFile=BRNN.initTestInput(&primaryStructureSize);
            }

            //get the error of an epoch
            BRNN.getEpochError();

            //close the pointers of the files
            BRNN.closeFolders();
            cout<<epoch<<endl;
            epoch++;
        }

        //create the output files
        BRNN.printOutputs();
        BRNN.printNetworkSpecification();

        return 0;
}
```

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 activationFunctionType;
	int errorFunctionType;

	double output;
	double error;
	double bias;
	double Dbias;
	double inputNet;
	double momentum;
	double learningRate;
	double biasPreviousAdjustment;

	void callActivationFunction(void);
	void calculateInput(void);

public:

	Neuron(int size,double momentumN,double learningRateN,int
functionType,int errorType);
	~Neuron(void);

	//methods of Neuron class
	double getOutput(vector<double> getInput);
	void calculateOutputLayerError(double targetOutput);
	void calculateErrorPropagation();
	void calculateHiddenLayerError(double prevWeightSum);
	void adjustDw();
	double getSpecificError(int x);
	void returnWeightVector(vector<double> *weihtsReturn);

};
```

Neuron.cpp

```
#include "StdAfx.h"
#include "Neuron.h"
#include "Services.h"

//*********************************************************************
*****************
//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
functionType,int errorType)
{

      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);

      }

      errorFunctionType=errorType;
      output=0;
      error=0;
      Dbias=0;
      inputNet=0;
      bias=newServices.rand_numbers();
      sizeOfInput=size;
      momentum=momentumN;
      learningRate=learningRateN;
      activationFunctionType=functionType;
      biasPreviousAdjustment=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)
{

      if(activationFunctionType==1)
      {
            output=(1/(1+pow(2.718281828,-inputNet)));
      }

}

//*************************************************************************
*****************
//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){

	error=output*(1-output)*(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){

      error=output*(1-output)*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(){

      for(int i=0;i<Dweights.size();i++)
      {

      Dweights[i]+=learningRate*inputVector[i]*error+momentum*previousAdj
ustment[i];

      previousAdjustment[i]=learningRate*inputVector[i]*error+momentum*pr
eviousAdjustment[i];
            weightsVector[i]=weightsVector[i]+Dweights[i];
            Dweights[i]=0;
            previousAdjustment[i]=0;
```

```
        }

        Dbias+=learningRate*1*error+momentum*biasPreviousAdjustment;
        biasPreviousAdjustment=learningRate*1*error+momentum*biasPreviousAd
justment;

        bias=-bias+Dbias;
        bias=-bias;

        Dbias=0;
        biasPreviousAdjustment=0;

}
```

DataReader.h

```
#pragma once
#include <fstream>

//*************************************************************************
*****************
//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;
	int beginning;

public:

	DataReader::DataReader();
	~DataReader(void);

	//methods of DataReader class
	void readParameters(float parameters[13],char
inputProfile[100],char outputProfile[100],
			char trainFile[100],char testFile[100]);
	void readEncoding(vector<string> &encodingT,int *inputSizeK,const
char inputProfile[100]);
	void initiateDataPointers(char trainFile[100],char testFile[100]);
	bool readTrainingInput(vector<char> &primaryStructure,vector<char>
&secondaryStructure);
	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]);

};
```

DataReader.cpp

```
#include "StdAfx.h"
#include "DataReader.h"

//*********************************************************************
*****************
//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[15]          :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[15],char
inputProfile[100],char outputProfile[100],
                char trainFile[100],char testFile[100])
```

```
{

	//variables
	char temp[100];
	char state[100];
	int counter=0;

	//opens the DataIn\\parameters.dat file to read the parameters
	ifstream inFile;
	inFile.open("DataIn\\parameters.dat");

	if (!inFile) {
		cerr << "Unable to open file parameters.dat";
		int x=0;
		cin>>x;
		exit(1);
	}

	//takes all the network parameters with specific order
	for(int i=0;i<19;i++){
		if(i<15)
		{
			inFile >> temp;
			inFile >> parameters[i];
		}else if(i==15)
		{
			inFile >> temp;
			inFile >> inputProfile;
		}else if(i==16)
		{
			inFile >> temp;
			inFile >> outputProfile;
		}else if(i==17)
		{
			inFile >> temp;
			inFile >> trainFile;
		}else if(i==18)
		{
			inFile >> temp;
			inFile >> testFile;
		}
	}
	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])
{

      //pointer to a file
      ifstream inFile;

      //variables
      char temp[100];
      char tempChar;
      int tempK;
      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,"EncodingProfiles\\\\");
      strcat(temp,inputProfile);

      //points to the file
      inFile.open(temp);

      if (!inFile) {
            cerr << "Unable to open file "<<inputProfile;
            int x=0;
            cin>>x;
            exit(1);
      }

      //read alla data from the file with a specific order
      for(int i=0;i<24;i++)
      {
            if(i<2)
            {
                  inFile >> temp;
            }else if(i==2)
            {
                  inFile >> temp;
                  inFile >> tempK;
```

```
                  *inputSizeK = tempK;
            }else if(i<24)
            {
                  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;
            }
      }

      //close the pointer
      inFile.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)
{
	//variables
	char name[100];
	char temp;

	primaryStructure.clear();

	inTrainFile >> name;

	//if end of file returns false
```

```
	if(inTrainFile.eof()){
		return false;
	}

	inTrainFile >> temp;

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

	secondaryStructure.clear();

	inTrainFile >> temp;

	//reads all the letters of secondary structur 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 name[100])
```

```
{
	//variables
	char temp;

	primaryStructure.clear();

	inTestFile >> name;

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

	inTestFile >> temp;

	//reads all the letters of primary structur 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 structur 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;
                        }
                  }
            }
      }
}
```

OutputData.h

```
#pragma once

//*************************************************************************
****************
//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 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);
	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);
};
```

OutputData.cpp

```
#include "StdAfx.h"
#include "OutputData.h"

//**********************************************************************
*****************
//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];

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

	strcat(temp,"DataOut\\\\");
	strcat(temp,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(HTMLName[i]=='\n')
		{
			HTMLName[i]='\0';
```

```
		}
		if(HTMLName[i]==' ')
		{
			HTMLName[i]='_';;
		}
		if(HTMLName[i]==':')
		{
			HTMLName[i]='_';
		}
	}

	//create folder
	mkdir(temp);

	//create the string path
	strcat(temp,"\\\\");
	strcat(temp,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++)
	{
		folderName[i]=temp[i];
	}

	strcat(temp,"_Output.txt");

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

//************************************************************************
*****************
//Deconstructor DataReader(void)
//************************************************************************
*****************
OutputData::~OutputData(void)
{

}
```

```
//*************************************************************************
****************
//void createErrorFiles(vector<double> trainingError,vector<double>
testingError,
//                vector<double> proteinTrainingError,vector<double>
proteinTestingError):
//public method that is called to create the files that contains the
training and testing
//error per protein and per epoch
//Parameters:
//                vector<double> trainingError        :vector with
training errors per epoch
//                vector<double> testingError               :vector with
testing errors per epoch
//                vector<double> proteinTrainingError :vector with
training errors per protein
//                vector<double> proteinTestingError  :vector with
testing errors per protein
//*************************************************************************
****************
void OutputData::createErrorFiles(vector<double>
trainingError,vector<double> testingError,
                vector<double> proteinTrainingError,vector<double>
proteinTestingError)
{
      //variables
      char temp[100];
      char temp1[100];

      //create vectors
      for(int i=0;i<100;i++)
            temp[i]='\0';

      for(int i=0;i<100;i++)
            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";
      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)
{

      //writes the name of a protein and the persentage of succes
      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;
}

//**********************************************************************
*****************
//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,"_Nwtwork_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:"<<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:"<<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:
"<<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: "<<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

```
#pragma once

//*********************************************************************
****************
//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();
};
```

Services.cpp

```
#include "StdAfx.h"
#include "Services.h"

Services::Services(void)
{
}

Services::~Services(void)
{
}

double Services::rand_numbers(){

	double temp_rand,temp_prosimo;

	temp_rand=0;temp_prosimo=0;

	temp_rand=((rand()%1000));
	temp_rand/=1000;
	temp_prosimo=rand()%2;

	if(temp_prosimo==0)
		temp_rand*=-1;

	return temp_rand;
}
```

BidirectionalRecurrentNeuralNetwork.h

```
#pragma once
#include "Neuron.h"
#include "DataReader.h"
#include "OutputData.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<double> weightsReturn;

	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];

	int hLayerOneSize;
	int hLayerTwoSize;
	int hLayerOneSizeB;
	int hLayerTwoSizeB;
	int hLayerOneSizeF;
	int hLayerTwoSizeF;
	int activationFuncType;
	int windowSize;
	int errorFunctionType;
	int s;
	int maxIterations;
	int trainingSetAdditionVectorTempSize;
```

```
        int testSetAdditionVectorTempSize;
        int percentageCounter;

        double learningRate;
        double momentum;
        double qMinus1;
        double qPlus1;
        double percentage;
        double trainingSetAddition;
        double testSetAddition;
        double trainingSetAdditionEpoch;
        double testSetAdditionEpoch;

        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;

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

        //provate method of BidirectionalRecurrentNeuralNetwork class
        void getInput(void);

public:
```

```
      BidirectionalRecurrentNeuralNetwork(int hLayerOneSize,int
hLayerTwoSize,int hLayerOneSizeB,int hLayerTwoSizeB,int hLayerOneSizeF,
                                                                  int
hLayerTwoSizeF,int activationFuncType,double learningRate,double
momentum,int windowSize,
                                                                  double
qMinus1,double qPlus1,int errorFunctionType,int s,int epochN,char
trainFile[100],char testFile[100],
                                                                  char
inputProfile[100],char outputProfile[100]);
      ~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);
      void openFolders(void);
      void closeFolders(void);
      void getSequenceTrainingError();
      void getSequenceTestingError();
      void getEpochError();
      void printOutputs();
      void printOutputSequence();
      void printNetworkSpecification();

};
```

BidirectionalRecurrentNeuralNetwork.cpp

```
#include "StdAfx.h"
#include "BidirectionalRecurrentNeuralNetwork.h"
#include "DataReader.h"

//*********************************************************************
*****************
//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 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])
{
      //objects of BidirectionalRecurrentNeuralNetwork to read from files
and write to files
      getInputData=new DataReader();
      saveOutputData=new OutputData();

      //variables
      inputSizeK=0;
      inputSize=0;
      halfWindow=0;
      halfInputSize=0;
      outputSize=0;
      percentageCounter=0;
      percentage=0.0;

      //assign parameters
      hLayerOneSize = hLayerOneSizeN;
      hLayerTwoSize = hLayerTwoSizeN;
      hLayerOneSizeB = hLayerOneSizeBN;
      hLayerTwoSizeB = hLayerTwoSizeBN;
      hLayerOneSizeF = hLayerOneSizeFN;
      hLayerTwoSizeF = hLayerTwoSizeFN;
      activationFuncType = activationFuncTypeN;
      learningRate = learningRateN;
      momentum = momentumN;
      windowSize = windowSizeN;
      qMinus1 = qMinus1N;
      qPlus1 = qPlus1N;
      errorFunctionType = errorFunctionTypeN;
      maxIterations=epochN;
      s = sN;

      //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];
      }

      //variables
```

```
	trainingSetAddition=0;
	testSetAddition=0;
	trainingSetAdditionEpoch=0;
	testSetAddition=0;

	//private method that is called to read the input and output
parameters
	BidirectionalRecurrentNeuralNetwork::getInput();

	//create variables for RNNs
	halfWindow=floor((float)windowSize/2);
	halfInputSize=floor((float)inputSize/2);

	//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
	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
ionFuncType,errorFunctionType));
		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
vationFuncType,errorFunctionType));
		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,
				activationFuncType,errorFunctionType));
		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
tivationFuncType,errorFunctionType));
		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,activationFuncType,
                        errorFunctionType));
            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
tivationFuncType,errorFunctionType));
            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,activationFuncType,errorFunctionType));
                  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,activationFuncType,errorFunctionType));
                  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,activationFuncType,errorFunctionType));
                  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,activationFuncType,errorFunctionType));
                  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,activationFuncType,errorFunctionType));
                  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,activationFuncType,errorFunctionType));
                  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,activationFuncType,errorFunctionType));
                  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,activationFuncType,errorFunctionType));
                  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)
{

}

void BidirectionalRecurrentNeuralNetwork::getInput(void)
{
      DataReader  getInput=DataReader();
      getInput.readEncoding(encodingT,&inputSizeK,inputProfile);
      inputSize=inputSizeK;
      sizeOfEachLetter=encodingT[0].size();
      inputSizeK=inputSizeK*sizeOfEachLetter;
      getInput.readOutputEncoding(encodingOutput,outputClassification,&ou
tputSize,outputProfile);
}

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


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

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

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

      return endOfFile;
}

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

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

      return endOfFile;
}
```

```
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);
				}
			}
			else
			{
				for(int q=0;q<sizeOfEachLetter;q++)
				{

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

		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++)
				{

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

		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()];
            }
      }
```

```
	for(int p=0;p<outputLayer.size();p++)
	{
		Ot[p]=outputLayer[p].getOutput(inputOutputLayer);
	}

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

	targetOutputs[p]=encodingOutput[secondaryStructure[sequencet]-
65][p]-48;
	}

	if(isTrainOrTest==1)
	{
		for(int p=0;p<Ot.size();p++)
		{
			trainingSetAddition+=(targetOutputs[p]-
Ot[p])*(targetOutputs[p]-Ot[p]);
		}
	}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;
		for(int i=0;i<Ot.size();i++)
		{
			if(Ot[i]>=0.5)
				tempOt[i]=1;
			else
				tempOt[i]=0;
		}

		for(int i=0;i<encodingOutput.size();i++)
		{
			counterEnc++;
			if(encodingOutput[i][0]!='\0')
			{
				int counter;
				for(counter=0;counter<tempOt.size();counter++)
				{
					if((int)encodingOutput[i][counter]-
48!=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::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()
{

	double tempPercentage;
	int counter=0;

	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,proteinName,tempPercentage);
	percentageCounter++;
	percentage+=tempPercentage;
	predictedSecondaryStructure.clear();
}

void BidirectionalRecurrentNeuralNetwork::printNetworkSpecification()
{

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

	hLayerTwoSizeF,activationFuncType,learningRate,momentum,windowSize,

	qMinus1,qPlus1,errorFunctionType,s,maxIterations,trainFile,testFile,

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

}

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

```
	trainingSetAdditionVector,trainingSetAdditionVector);
	saveOutputData->closeAllPointers();
}


void BidirectionalRecurrentNeuralNetwork::doBackpropagation(int
sequencet){

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

		outputLayer[p].calculateOutputLayerError(targetOutputs[p]);
		outputLayer[p].calculateErrorPropagation();
		outputLayer[p].adjustDw();

	}

	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();
		}




		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();
            }

            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();
            }


     }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();
            }



            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();
            }

            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();
            }


            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();
            }



      }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();
            }



            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();
            }

            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();
            }

            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();
            }
```

```
      }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();
             }




             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();
             }

             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();
                }

                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();
                }


                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();
                }
```

```
}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();
	}


	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();
	}




	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();
            }

            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();
            }




     }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();
            }


            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();
            }




            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();
            }

            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();
            }


            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();
            }



      }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();
      }


      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();
      }




      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();
            }

            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();
            }

            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();
            }



     }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();
            }


            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();
            }




            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();
            }

            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();
            }

            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();
            }


            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();
                }




        }


}
```

Παράρτημα Β

Parser

Readfile.cpp

```
//read file and input characters

#include <iostream>
using std::cerr;
using std::cout;
using std::endl;
using std::fixed;
using std::ios;
using std::left;
using std::right;
using std::showpoint;

#include <fstream> // file stream
using std::ifstream; // input file stream
using std::ofstream; // output file stream

#include <iomanip>
using std::setw;

#include <string>
using std::string;

#include <vector>
using std::vector;

#include <sstream>
using namespace std;

int main()
{
   char str[1000];
   char str1[1000];
   char
residues[20]={'A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R'
,'S','T','V','W','Y'};
      char ss[3]={'H','E','L'};

      ofstream print;
      print.open("proteins.txt");

   int dummyLines;
   bool isReadable=true;
   bool isFound=true;

   vector<char> primaryStructure;
   vector<char> secondaryStructure;

   vector<string> pdb_select; //create vector pdb_select to store string
name of Hobohm's protein name
   vector<string> pdb_select_dssp; //create vector pdb_select to store
string name of Hobohm's protein name lowercase
```

```
   //vector<string> dssp;//create vector dssp to store ****.dssp files

   string temp,temp1,string1,chainNumber; //string variables
   temp="     ";
   temp1="    .dssp";

   ifstream inClientFile;//if stream constructor creates object
   inClientFile.open("recent.pdb_select25");// inClientFile object opens
the file
    ifstream inClientFile2;

   istringstream inputStringPdb( temp1 );

   int dsspLines;
   int dsspResidue;
   int previousDsspResidue;
   int equalizarion;

   //create new file ofstream outClientFile( "protein.dat" , ios::out );
   // exit program if unable to create file
   if ( !inClientFile ) //overloaded ! operator
   {
      cerr << "File could not be opened" << endl;
      exit( 1 );
   } // end if

   while (!inClientFile.eof())          //read data untill end-of-file
   {
          inClientFile.getline(str,1000);

            if(str[36]=='N' || str[36]=='X') //select only proteins with
N (NMR) and X (X-Ray)
            {

            for(int i=0;i<5;i++) //for loop selecting
            {

                  temp[i]=str[i+8]; //store proteins name in temp
variable
                  if(i<4 && temp[i]>='A' && temp[i]<='Z')//turn uppercase
to lowercase letters
                        temp[i]+=32;

            }

            for(int i=0;i<4;i++)//store from pdb_select to
pdb_vector_dssp vector
            {
                  temp1[i]=temp[i];
            }

            pdb_select.push_back(temp); //function push_back to store
variables in vector pdb_select
            pdb_select_dssp.push_back(temp1);//function push_back to
store variables in vector pdb_select_dssp
    }
```

```
   }

   inClientFile.clear();//clear file
   inClientFile.close();//close file

int counter1=1;
int count=0;
char tempString[1000];



while(count<pdb_select_dssp.size())

{
      dummyLines=0;
      isFound=true;
      isReadable=true;
      equalizarion=0;
   string1="";
   string1.append("dsspdata1\\\\");  //open folder and read files
   string1.append(pdb_select_dssp[count]);
      cout<<string1<<endl;

      for(int i=0;i<100;i++)
         tempString[i]='\0';

          int counter=0;
            counter1=1;
        while(counter<string1.size())
          {
               tempString[counter]=string1[counter];
               counter++;
        }

        inClientFile2.open(tempString);
        //cout<<tempString<<endl;

        if ( !inClientFile2 ) //overloaded ! operator
        {
           cout << "File could not be opened" << endl;
                  isFound=false;
            }else{
                  isFound=true;
            }
            // end if loop


             while (!inClientFile2.eof() && isFound==true)          //read
data until end-of-file
        {

                  if (counter1 < 29)
                  {
              inClientFile2.getline(str,1000);
                  //cout<<"counter: "<<counter1<<endl;
                  }
```

```
                  else {

                        str1[11]='\0';

                        if(isReadable==true){
                              inClientFile2.getline(str1,1000);
                        }

                              istringstream inputStringDssp(str1);


                              inputStringDssp>>dsspLines;
                              //previousDsspResidue=dsspResidue;
                              inputStringDssp>>dsspResidue;

                        //cout<<str1<<endl;

                        //if (pdb_select[count][4] == str1[11])
                        if(pdb_select[count][4] != str1[11] &&
str1[11]<pdb_select[count][4]){
                              dummyLines++;


                        }
                        else if(pdb_select[count][4] == str1[11])
                        {

                              //cout<<dsspLines<<" "<<dsspResidue<<endl;

                              if((dsspLines-
dummyLines)+equalizarion==dsspResidue)
                              {
                                    if(str1[13]==' '){
                                    //    primaryStructure.push_back('X');

      primaryStructure.push_back(residues[rand()%20]);

                                    }
                                    else

      primaryStructure.push_back(str1[13]);

                                    if(str1[16]==' ')

      secondaryStructure.push_back(ss[rand()%3]);

                                    else

      secondaryStructure.push_back(str1[16]);

                                    isReadable=true;

                              }
                              else
                              {
```

```
                                    for(int i=0;i<(dsspResidue-
dsspLines);i++)
                                    {
      primaryStructure.push_back(residues[rand()%20]);
      secondaryStructure.push_back(ss[rand()%3]);


                                    }
                                   equalizarion+=(dsspResidue-dsspLines);
                                    if(str1[13]==' ')
      primaryStructure.push_back(residues[rand()%20]);
                                    else
      primaryStructure.push_back(str1[13]);
                                    if(str1[16]==' ')
      secondaryStructure.push_back(ss[rand()%3]);
                                    else
      secondaryStructure.push_back(str1[16]);
                              }


                        }


                  }
                                        counter1++;

            }

            print<<pdb_select[count]<<endl;
            for(int p=0;p<primaryStructure.size();p++)
            {
                  print<<primaryStructure[p];
            }

            print<<".";
            print<<endl;
            for(int p=0;p<secondaryStructure.size();p++)
            {
                  print<<secondaryStructure[p];
            }
            print<<".";
            print<<endl;

            primaryStructure.clear();
            secondaryStructure.clear();
            //cout<<endl;
```

```
        inClientFile2.clear();//clear file
            inClientFile2.close();
        count++;


}



    print.close();
    system("pause");
    return 0;

                                        }
```

Παράρτημα Γ

Αρχείο δεδομένων εισόδου

1jxsA

DSKPPYSYAQLIVQAITMAPDKQLTLNGIYTHITKNYPYYRTADKGWQNSIRHNLSLNRYFIKVPRSQEEPGKGSF
WRIDPASESKLIEQAFRKRRPR.

LLLLLLLTTHHHHHHHHHTSTTSEELHHHHHHHHHHHHSSLSSSSLTTTHHHHHHHHHHHLLLEEEESLLLSSSSSSLEE
EELTTTHHHHHHHHHSLSLLL.

1x4qA

GSSGSSGMALSKRELDELKPWIEKTVKRVLGFSEPTVVTAALNCVGKGMDKKKAADHLKPFLDDSTLRFVDKLFE
AVEEGRSSRHSSGPSSG.

LLLLSSLLLLLHHHHHHHHHHHHHHHHHHHHSSLLHHHHHHHHHHHHHTTLLHHHHHHHHHTTTTGGGTHHHHH
HHHHHHHHHSLSLLLSSLLLL.

1x4mA

GSSGSSGHGDGPGNAVQEIMIPASKAGLVIGKGGETIKQLQERAGVKMVMIQDGPQNTGADKPLRITGDPYKV
QQAKEMVLELIRDQGSGPSSG.

LLLLLLLLLLLLLLEEEEEEELHHHHHHHHSLSSSSHHHHHHHHHHTSEEEELLSLLLSSLSLEEEEEEELTTTHHHHHHH
HHHHHLLLSSLSSLL.

1x4lA

GSSGSSGCAGCTNPISGLGGTKYISFEERQWHNDCFNCKKCSLSLVGRGFLTERDDILCPDCGKDISGPSSG.

LLSLSLSBTTTTBLLLLSSSLSLEELSSLEELTTTLBLSSSLLBLTTSLLEELSSSEELHHHHHTLLLSSLL.

1x4iA

GSSGSSGYCICNQVSYGEMVGCDNQDCPIEWFHYGCVGLTEAPKGKWYCPQCTAAMKRRGSRHKSGPSSG.

LLLSLLLLSTTSLLLLSSEELLSLTTLSLLLEEHHHHTLSSLLSSLLLLHHHHHHHHHHHHTTLLLLLLLL.

1x4fA

GSSGSSGKKPEGKPDQKFDQKQELGRVIHLSNLPHSGYSDSAVLKLAEPYGKIKNYILMRMKSQAFIEMETREDA
MAMVDHCLKKALWFQGRCVKVDLSEKYKKLVSGPSSG.

LLLSSSLLLSLLLLLLLLLSSLLLLEEEEESLLLSSLLSHHHHTTTTTTSLLSEEEEETTTTEEEEELSSHHHHHHHHHHH
HHSLLLSSSSLLEEEEELSLSSSSLLLLLL.

Παράρτημα Δ

Παράμετροι

```
Hidden_layer_one_size 12
Hidden_layer_two_size 12
Hidden_layer_one_of_Backward_size 11
Hidden_layer_two_of_Backward_size 9
Hidden_layer_one_of_Forward_size 11
Hidden_layer_two_of_Forward_size 9
Activation_Function_Type 1
Learning_Rate 0.2
Momentum 0.1
Window_size 15
q_minus_one 0.5
q_plus_one 0.5
Error_Function_Type 1
s 3
Maximum_Iterations 200
input_Profile sparse.txt
output_Profile threeClasses.txt
train_File proteins.txt
test_File proteins.txt
```

Παράρτημα Ε

Αρχείο δεδομένων εξόδου

2prfA Correctness Percentage:52.8%

primaryStructure :SWQTYVDTNLVGTGAVTQAAILGLDGNTWATSAGFAVTPAQGQTLASAFNNADP
IRASGFDLAGVHYVTLRADDRSIYGKKGSAGVITVKTSKSILVGVYNEKIQPGTAANVVEKLADYLIGQGF

secondaryStructure :LHHHHHLLLLLLLLLLLLEEEELLLLLLLEEELLLLLLLHHHHHHHHHLLLLLLLLLLLEELL
LLEELLLLLLLLLLEEELLLLEEEEEELLLEEEEEELLLLLLLHHHHHHHHHHHHHHHLLLL

predictedSecondaryStructure:HHHHHHLLHHLLLHHHHHHLHLLLLLHHHHLHHHHHLLLHHLHHHHLHLHL
LLLHLLLLLHHLLHHHEELLLLLLLLLLHLLLLLLLHHHLHHHHHLHHHHHHLLLLHHHHHHHHHHHHLLHLH

1lv3A Correctness Percentage:47.6923%

primaryStructure :MSETITVNCPTCGKTVVWGEISPFRPFCSKRCQLIDLGEWAAEEKRIPSSGDLSESD
DWSEEPKQ

secondaryStructure :LLLLLEEELLLLLEEELLLLLLLLLLLHHHHHHHHHLLLLLLLLLLLLLLLLLLLLLLL

predictedSecondaryStructure:LLLEELLLLLLLHEEHHHHHLLHHLHHHHHHHHHLHLHHHHHHHLLLLLLLH
LHLLHHHHHHHH

2pq4B Correctness Percentage:62.8571%

primaryStructure :MKLSRRSFMKANAVAAAAAAAGLSVPGVARAVVGQ

secondaryStructure :LLLLLHHHHHHHHHHHHHHHLLLLLLLLLLLLL

predictedSecondaryStructure:HHHHHHHHHHHHHHHHHHHHLLLLLLHHHHHHH

2pq4A Correctness Percentage:47.7778%

primaryStructure :GSHMHTNWQVCSLVVQAKSERISDISTQLNAFPGCEVAVSDAPSGQLIVVVEAEDS
ETLIQTIESVRNVEGVLAVSLVYHQQEEQGEETP

secondaryStructure :LLLLLLLLLEEEEEEELHHHHHHHHHHHHHLLLLEEEELLLLLLEEEEEELLLHHHH
HHHHHHHHHLLLEEEEELLEELLLLLLLL

predictedSecondaryStructure:LLLLLLHHHHHHHHHHHHHHHHHHLLHHLLLLLELELLLLLLLLHHHHHHH
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHLLLHLLH

1lpvA Correctness Percentage:53.8462%

primaryStructure :SISPRTPPNCARCRNHGLKITLKGHKRYCKFRYCTCEKCRLTADRQRVMALQ

secondaryStructure :LLLLLLLLLLHHHHLLLLLLLLLLLHHHLLLLLLLLHHHHHHHHHHLLLLLL

predictedSecondaryStructure:LLLLLLLLHHHHLLLLLHHHHHHLHHHHHHHHHHLHHHHHHHHHHHHH
HHH

Παράρτημα Στ

Αρχείο κωδικοποίησης εισόδου

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

Παράρτημα 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 010

L 001