Ατομική Διπλωματική Εργασία

ECHO STATE NETWORKS ΚΑΙ Η ΠΡΟΒΛΕΨΗ ΤΗΣ ΔΕΥΤΕΡΟΤΑΓΟΥΣ ΔΟΜΗΣ ΤΩΝ ΠΡΩΤΕΪΝΩΝ

Μαρία Ιγκαρίεβνα Μασλιουκόβα

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2017

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Echo State Networks και η Πρόβλεψη της Δευτεροταγούς Δομής των Πρωτεϊνών

Μαρία Ιγκαρίεβνα Μασλιουκόβα

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 2017

Ευχαριστίες

Θα ήθελα να ευχαριστήσω μερικά άτομα που έπαιξαν σημαντικό ρόλο στην ολοκλήρωση αυτής της διπλωματικής εργασίας.

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή Δρ. Χρίστο Χριστοδούλου, ο οποίος πίστεψε σε εμένα και μου έδωσε την ευκαιρία να ολοκληρώσω αυτή την σημαντική εργασία.

Στην συνέχεια, θα ήθελα να ευχαριστήσω πολύ τον διδακτορικό φοιτητή Μιχάλη Αγαθοκλέους, ο οποίος μου προσέφερε τις γνώσεις του και τις συμβουλές τους ώστε να ξεπεράσω δυσκολίες που αντιμετώπισα και να μπορέσω να συνεχίσω με την υλοποίηση της διπλωματικής μου εργασίας.

Τέλος, ένα μεγάλο ευχαριστώ στους γονείς μου οι οποίοι στάθηκαν στο πλευρό μου και μου έδιναν δυνάμεις να συνεχίσω την προσπάθεια μου αυτή την πολύ πιεστική και αγχωτική περίοδο της ζωής μου.

Περίληψη

Αυτή η έρευνα ασχολείται με το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών με την βοήθεια τεχνητών νευρωνικών δικτύων.

Η σωστή λειτουργία ενός οργανισμού εξαρτάται από τις πρωτεΐνες, αφού το πλήθος τους είναι ανάλογο των λειτουργιών τους. Αυτές είναι υπεύθυνες για πολλές λειτουργίες του σώματός μας, όπως την αναπαραγωγή, την άμυνα και την συντήρησή του. Η τριτοταγής δομή μίας πρωτεΐνης είναι αυτή που μας δίνει την πληροφορία για τον ρόλο της, καθώς και τις ιδιότητές της. Άρα, αν υπάρξει μία μέθοδος προσδιορισμού αυτής της δομής, θα δοθεί η ευκαιρία στους επιστήμονες να φτιάξουν πρωτεΐνες με συγκεκριμένες ιδιότητες για την ίαση ασθενειών. Όμως οι μέθοδοι που έχουν αναπτυχθεί μέχρι σήμερα απαιτούν πολύπλοκες, δαπανηρές και χρονοβόρες διαδικασίες. Με αποτέλεσμα ενώ έχουμε αναγνωρίσει μεγάλο αριθμό πρωτεϊνικών ακολουθιών, ένα πολύ μικρό ποσοστό από αυτές να έχει προσδιορισμένη την τριτοταγή της δομή πειραματικά.

Σκοπός αυτής της έρευνας είναι αρχικά η μελέτη της δουλειάς που έχει ήδη γίνει από την φοιτήτρια Ειρήνη Παπακώστα [1] ώστε να καταγραφούν πιο ολοκληρωμένα συμπεράσματα για την εφαρμογή των Echo State Networks (ESN) σε αυτό το πρόβλημα. Έπειτα θα επεκταθεί το δίκτυο, το οποίο θα εκπαιδεύεται με τον αλγόριθμο ανάστροφης μετάδοσης λάθους και δεδομένου της πρωτοταγούς δομής θα προβλέπει την δευτεροταγή. Αυτό το δίκτυο θα αποτελείται από δύο ESN, το ένα ESN θα επεξεργάζεται την πρωτεϊνική ακολουθία από τα αριστερά και το άλλο από τα δεξιά, ώστε να ληφθούν υπόψη όλες οι σχέσεις μεταξύ των αμινοξέων. Τέλος θα χρησιμοποιηθεί και ένα feed forward νευρωνικό δίκτυο το οποίο θα επεξεργάζεται το μεσαίο αμινοξύ κάθε φορά.

Το αρχικό ποσοστό επιτυχίας ελέγχου του δικτύου είναι 72.93% το οποίο θεωρείται ικανοποιητικό για αρχή και αναμένεται να γίνουν βελτιστοποιήσεις, ώστε να πλησιάσει το 76%, το οποίο είναι το καλύτερο καταγεγραμμένο ποσοστό επιτυχίας μέχρι στιγμής που έχει γίνει από παρόμοιες έρευνες.

Περιεχόμενα

Κεφάλαιο 1

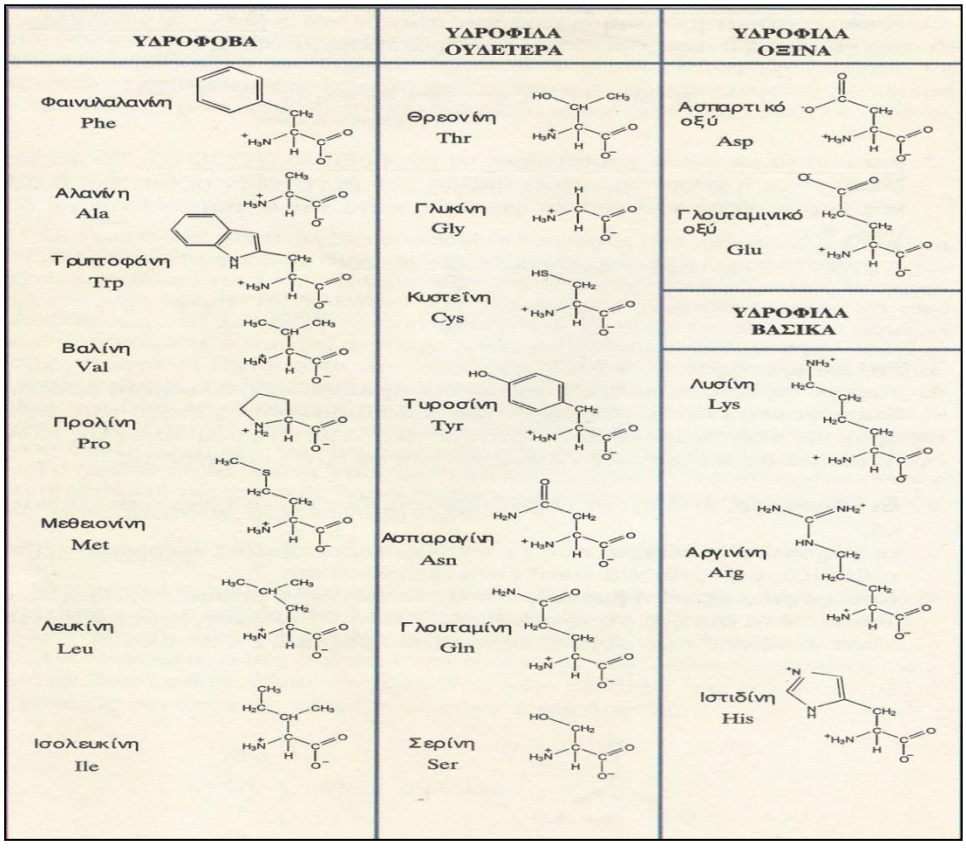
Εισαγωγή

1.1 Βιολογικό Υπόβαθρο

1.1.1 Πρωτεΐνες

Οι πρωτεΐνες αποτελούνται από μία ή περισσότερες (πεπτιδικές) αλυσίδες από απλούστερες χημικές ουσίες, τα αμινοξέα, των οποίων η διάταξη είναι προκαθορισμένη στο γενετικό υλικό (DNA). Στην φύση υπάρχουν πολλά αμινοξέα, όμως στις πρωτεΐνες συναντούμε μόνο 20 από αυτά, γνωστά και ως L-a-αμινοξέα (Πίνακας 1.1).

Επομένως, η κάθε πρωτεΐνη έχει την δική της αλληλουχία αμινοξέων και οποιαδήποτε αλλαγή στην διάταξη ή των αριθμό των αμινοξέων παριστάνει μια άλλη πρωτεΐνη με διαφορετική λειτουργία. Οι βασικότερες λειτουργίες παρουσιάζονται στον Πίνακα 1.2.[2]

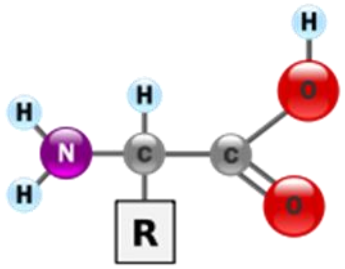


Πίνακας 1.1: Τα 20 L-a-αμινοξέα που απαρτίζουν τις πρωτεΐνες
(https://antonislazaris68.files.wordpress.com/2014/09/ceb1cebcceb9cebdcebfcebeceadceb1.jpg)

Βιολογικός Ρόλος Πρωτεϊνών
Ενζυμική κατάλυση
Μεταφορά και αποθήκευση
Κίνηση
Μηχανική κίνηση
Αντισώματα
Δημιουργία και μετάδοση νευρικών παλμών
Έλεγχος της ανάπτυξης και διαφοροποίησης

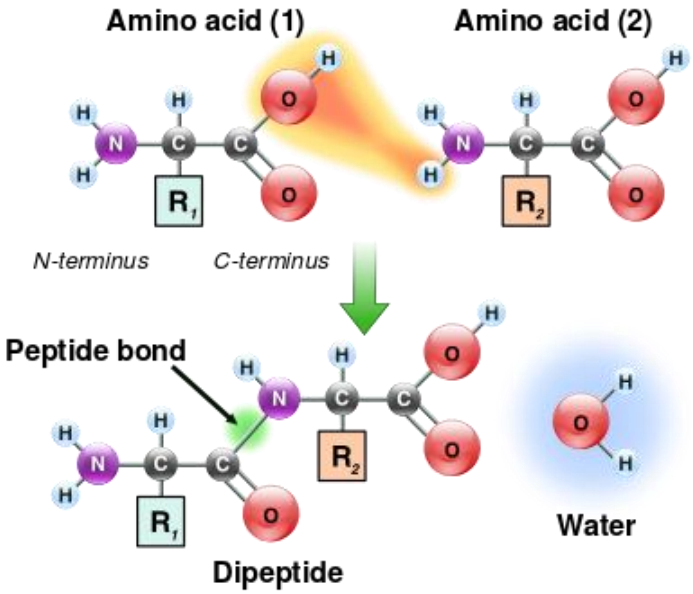
Πίνακας 1.2: Βιολογικός ρόλος πρωτεϊνών

Όπως αναφέραμε και πιο πάνω τα αμινοξέα ενώνονται μεταξύ τους και δημιουργούν πεπτιδικούς δεσμούς. Κάθε αμινοξύ αποτελείται από μία αμινομάδα (-NH2) και μια ομάδα καρβοξυλίου (-COOH) όπως φαίνεται στο Σχήμα 1.1.



Σχήμα 1.1 : Γενικός τύπος των αμινοξέων (https://en.wikipedia.org/wiki/Amino_acid)

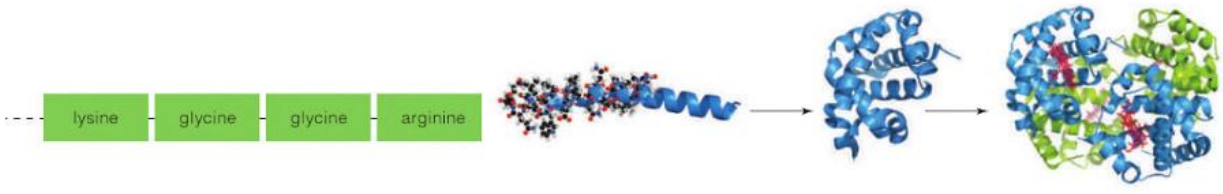
Η πλευρική αλυσίδα R είναι αυτή που διαφοροποιεί τα αμινοξέα και που καθορίζει την ιδιότητά τους. Για να δημιουργηθεί ένας πεπτιδικός δεσμός γίνεται αντίδραση της αμινομάδας ενός αμινοξέος με την καρβοξυλομάδας κάποιου άλλου και την αποβολή ενός μορίου νερού. Έτσι σχηματίζεται ένα διπεπτίδιο (Σχήμα 1.2) το οποίο έχει την δυνατότητα να συνεχίσει την αντίδραση με άλλα αμινοξέα. Καθώς συνεχίζονται οι αντιδράσεις δημιουργούνται πολυπεπτίδια και όταν αποτελούνται από τουλάχιστον 100 αμινοξέα θεωρούνται πρωτεΐνες. [3]



Σχήμα 1.2: Ένωση δύο αμινοξέων
(https://en.wikipedia.org/wiki/Amino_acid)

1.1.2 Επίπεδα οργάνωσης των πρωτεϊνών

Όπως έχουμε ήδη αναφέρει τα αμινοξέα αποτελούν τα δομικά στοιχεία των πρωτεϊνών και αυτά είναι που καθορίζουν την αναδίπλωση της πρωτεΐνης στο χώρο. Δηλαδή στην φύση συναντούμε την πρωτεΐνη στην τριτοταγή της δομής και με αυτή μπορούμε να καθορίσουμε την λειτουργία της. Σημαντικό είναι να αναφέρουμε πως αν πάρουμε μία συγκεκριμένη ακολουθία αμινοξέων πάντοτε προκύπτει η ίδια πρωτεΐνη με την ίδια λειτουργία. Παρόλο που συναντούμε τις πρωτεΐνες στην τελική τους δομή, για την διευκόλυνση της μελέτη τους τις οργανώνουμε σε τέσσερα επίπεδα.



A The primary structure of a protein is its linear sequence of amino acids. This string of amino acids is a polypeptide chain.

B Secondary structure comes about as a polypeptide chain twists or folds. Hydrogen bonds hold the molecule in this shape.

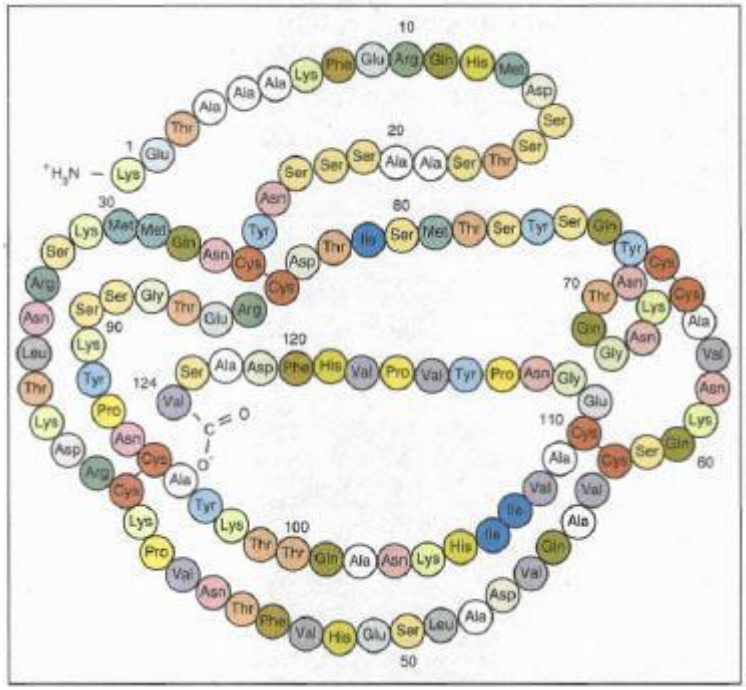
C More folding of the chain produces a protein's tertiary structure—its overall three-dimensional shape. The folding results in pockets or crevices that establish how a protein will function chemically.

D In proteins with quaternary structure, bonds and other forces hold two or more polypeptide chains together in one molecule. This example shows how hemoglobin, which consists of four chains (*here colored green or blue*). A pocket in each chain holds a heme group (*red*) that contains an iron atom.

Σχήμα 1.3: Πρωτοταγής, δευτεροταγής, τριτοταγής και τεταρτοταγής δομή της πρωτεΐνης. [2]

1.1.2.1 Πρωτοταγής δομή

Η πρωτοταγής δομή είναι η σειρά με την οποία συνδέονται τα αμινοξέα της πρωτεΐνης. Εντούτοις η μόνη πληροφορία που αντλούμε από αυτή την δομή είναι η αλληλουχία των αμινοξέων και όχι το πώς αναδιπλώνονται στο χώρο, ώστε να μπορέσουμε να προσδιορίσουμε τον βιολογικό ρόλο της πρωτεΐνης. Σήμερα η αναγνώριση της γίνεται με την μέθοδο της υδρόλυσης ή την αποικοδόμηση Edman.

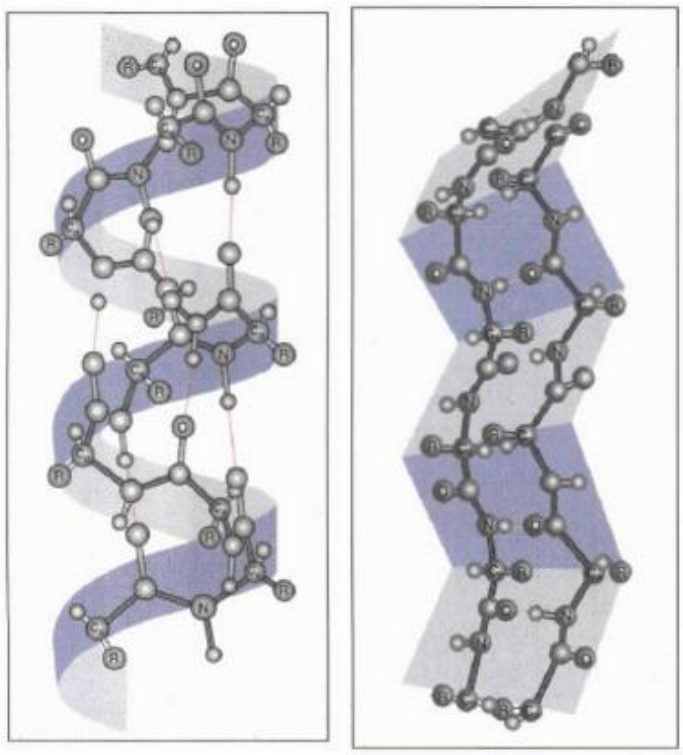


Σχήμα1. 4: Πρωτοταγής δομή πρωτεΐνης. [4]

1.1.2.2 Δευτεροταγής δομή

Η δευτεροταγής δομή μιας πρωτεΐνης αφορά τις τοπικές αναδιπλώσεις της πολυπεπτιδικής αλυσίδας, οι οποίες τελικά θα δώσουν το χαρακτηριστικό σχήμα στην πρωτεΐνη και την λειτουργία της (τριτοταγής δομή). Αυτές οι αναδιπλώσεις παρατηρούνται λόγω της ύπαρξης υδρογονικών δεσμών και ηλεκτροστατικών αλληλεπιδράσεων μεταξύ των τμημάτων της αλυσίδας. Παρατηρήθηκε ότι οι μορφές που μπορούν να έχουν χωρίζονται σε δύο κατηγορίες. Αυτές οι κατηγορίες είναι οι α-έλικας και β-πτυχωτές επιφάνειες, αν και σε μερικές περιπτώσεις η αναδίπλωση γίνεται με τυχαίο τρόπο (coil). Τα τμήματα που ανήκουν στην κατηγορία α-έλικα παίρνουν το σχήμα τους λόγω των δεσμών υδρογόνου μεταξύ αμινοξέων της ίδιας πολυπεπτιδικής

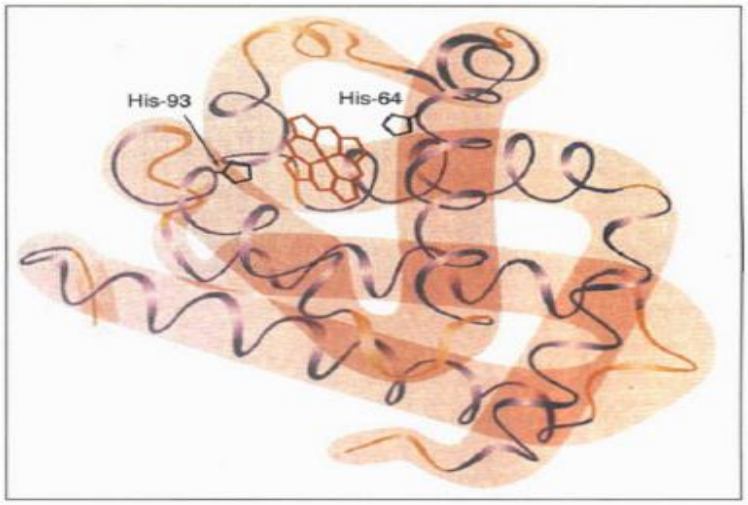
αλυσίδας, τα οποία βρίσκονται σε κοντινή απόσταση μεταξύ τους. Ενώ στην β-πτυχωτή επιφάνεια αυτός ο δεσμός γίνεται μεταξύ διαφορετικών πολυπεπτιδικών αλυσίδων του ίδιου πρωτεϊνικού μορίου. [4]



Σχήμα 1.5: Αριστερά: Δευτεροταγής δομή α-έλικα. Δεξιά: Δευτεροταγής δομή β-πτυχωτής επιφάνειας [4]

1.1.2.3 Τριτοταγής δομή

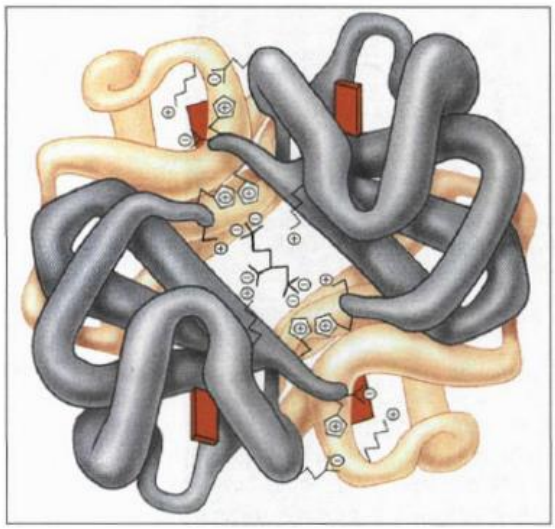
Η τριτοταγής δομή αποτελεί τον συνδυασμό των αναδιπλωμένων τμημάτων της δευτεροταγούς δομής προσδίδοντας το τελικό σχήμα της πρωτεΐνης. Όταν βρούμε την τριτοταγή δομή μπορούμε πλέον να καθορίσουμε και την λειτουργία της πρωτεΐνης που είναι και το ζητούμενο.



Σχήμα1. 6: Τριτοταγής δομή με τμήματα α-έλικα και τυχαία αναδίπλωση [4]

1.1.2.4 Τεταρτοταγής δομή

Όπως έχουμε αναφέρει κάποιες πρωτεΐνες αποτελούνται από πολλές πολυπεπτιδικές αλυσίδες, σε αυτή την περίπτωση έχουν και τεταρτοταγή δομή. Σε αυτή την δομή συναντούμε την συνένωση των τριτοταγών δομών των αλυσίδων.



Σχήμα 1.7: Τεταρτοταγής δομή αιμοσφαιρίνης. [4]

1.2 Αναδρομή σε υφιστάμενες υλοποιήσεις

Καθ΄ όλη την διάρκεια που μελετάται αυτό το πρόβλημα, έχουν παρουσιαστεί διάφορες μελέτες με προτεινόμενες προσεγγίσεις για την επίλυση του. Λόγω του μεγάλου αριθμού τους θα αναφερθούν οι σημαντικότερες από αυτές, οι οποίες αφορούν κυρίως υλοποιήσεις με στατιστικές μεθόδους και νευρωνικά δίκτυα. Σε αρκετά σημεία θα αναφέρομαι στο ποσοστό επιτυχίας Q3 (Εξίσωση 1.1) [5] του αλγορίθμου, το οποίο καθορίζει με πόση ακρίβεια το δίκτυο αναγνώρισε την άγνωστη πρωτεϊνική ακολουθία.

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

Εξίσωση 1.1: Ποσοστό επιτυχίας αναγνώρισης πρωτεϊνικής ακολουθίας, όπου N res το πλήθος των αμινοξέων της ακολουθίας εισόδου και Mii παίρνει την τιμή 1 αν η τιμή εξόδου του i είναι ίδια με του j, διαφορετικά παίρνει την τιμή 0.

Μέθοδος Chou - Fasman: Ήταν μία από τις πρώτες και πιο διαδεδομένες στατιστικές μεθόδους που αναπτύχθηκαν σχετικά με αυτό το πρόβλημα στα μέσα του 1970. Αφού μελέτησαν τις ήδη γνωστές δευτεροταγείς δομές έφτιαξαν εμπειρικούς κανόνες για την πρόβλεψη του τύπου της κάθε αναδίπλωσης. Έπειτα όταν μελετηθεί μία άγνωστη πρωτεΐνη υπολογίζουν με μία πιθανότητα βάσει των κανόνων τους την δευτεροταγή της δομή. Αυτή η μέθοδος είχε ποσοστό επιτυχίας 50 - 60%. [6]

Μέθοδος GOR (Garnier – Osguthorpe – Robson): Όπως και η προηγούμενη μέθοδος και αυτή υπολογίζει την πιθανότητα εμφάνισης του κάθε τύπου δευτεροταγούς δομής. Η διαφορά είναι ότι παίρνουν ένα παράθυρο 17 αμινοξέων και βάσει των γειτονικών αμινοξέων προσπαθούν να βρουν τον τύπο του μεσαίου αμινοξέος. Αυτή η μέθοδος έφτασε το 65% επιτυχία.[7]

Μετά την χρήση των στατιστικών μεθόδων στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών εισάχθηκαν τα τεχνητά νευρωνικά δίκτυα (ΤΝΔ). Το πλεονέκτημα των ΤΝΔ είναι ότι μπορούν να βρουν μία συσχέτιση μεταξύ των δεδομένων εισόδου και έπειτα να χρησιμοποιήσουν αυτή την γνώση για να προβλέψουν

την δομή μίας άγνωστης πρωτεΐνης. Στην συνέχεια ακολουθούν προσεγγίσεις που χρησιμοποίησαν ΤΝΔ.

Δίκτυο εμπρόσθιου περάσματος Qian και Sejnowski: Σε αυτή την προσέγγιση χρησιμοποιήθηκε ένα δίκτυο εμπρόσθιου περάσματος με ένα κρυφό επίπεδο και η είσοδος του ήταν ένα παράθυρο 13 αμινοξέων. Σκοπός του ήταν να προβλέψει την δευτεροταγή δομή του μεσαίου αμινοξέος. Στην προσπάθεια τους να αυξήσουν το ποσοστό επιτυχίας πρόσθεσαν ένα δεύτερο δίκτυο, το οποίο έπαιρνε ως είσοδο την έξοδο του πρώτου. Κατάφεραν να αυξήσουν το ποσοστό επιτυχίας από 62.5% σε 64.3%, όμως το δίκτυο τους αντιμετώπιζε προβλήματα υπερεκπαίδευσης. [7,8]

Δίκτυο PHD: Οι Sander και Rost βασίστηκαν στην προηγούμενη μέθοδο και κατάφεραν να ξεπεράσουν το πρόβλημα της υπερεκπαίδευσης. Το δίκτυο τους αποτελείται από τρία διαφορετικά επίπεδα. Στο πρώτο επίπεδο το δίκτυο παίρνει την ακολουθία ως είσοδο και προβλέπει την δευτεροταγή δομή. Το δεύτερο επίπεδο χρησιμοποιεί την έξοδο του πρώτου ως είσοδο και αυτό με την σειρά του προσπαθεί να προβλέψει την δευτεροταγή δομή. Το τελευταίο επίπεδο παίρνει τις εξόδους των δικτύων του δεύτερου επιπέδου, τα οποία εκπαιδεύτηκαν με διαφορετικά δεδομένα, και συνδυάζει όλες τις εξόδους παίρνοντας το μέσο όρο τους. Αυτή η τεχνική ονομάζεται ensemble average και μαζί με την τεχνική του early stopping κατάφεραν να αντιμετωπίσουν το πρόβλημα της υπερεκπαίδευσης. Όμως το σημαντικότερο μέρος της μεθόδου τους ήταν ότι χρησιμοποίησαν την πολλαπλή στοίχιση των πρωτεϊνών (MSA), η οποία εμπλούτιζε τις γνώσεις του δικτύου και πέτυχαν 74% ποσοστό επιτυχίας. [9,10]

Δίκτυο NNSSP: Αυτό το δίκτυο αναπτύχθηκε από τους Salamov και Soloveyev και μοιάζει με αυτό των Sander και Rost. Η μέθοδος που χρησιμοποίησαν ήταν αυτή του κοντινότερου γείτονα, όπου το δίκτυο προσπαθούσε να ομαδοποιήσει τις πρωτεΐνες και να αποφασίσει την δευτεροταγή τους δομή βάσει των γνωστών σε αυτό πρωτεΐνες. Μετέπειτα χρησιμοποίησαν την MSA κωδικοποίηση για τα δεδομένα εισόδου και χρησιμοποίησαν διαφορετικά σύνολα εκπαίδευσης και ελέγχου στο δίκτυο. Αυτό είχε ως αποτέλεσμα την αύξηση του ποσοστού επιτυχίας του στο οποίο έφτασε το 73.5%.[11]

Δίκτυο BRNN: Το Bidirectional Recurrent Neural Network προτάθηκε από τον Baldi και τους συνεργάτες του το 1999. Κατάλαβαν ότι η αδυναμία των προσεγγίσεων που χρησιμοποιούσαν δίκτυα εμπρόσθιου περάσματος ήταν ότι δεν λάμβαναν υπόψη τις δυνάμεις που ασκούνται σε ένα αμινοξύ από τα προηγούμενα και επόμενα του αμινοξέα. Έτσι έφτιαξαν ένα δίκτυο του οποίου η είσοδος είναι ένα κινητό παράθυρο σταθερού μήκους για να αντιμετωπίσουν αυτή την αδυναμία. Όταν δόθηκε στο δίκτυο ως είσοδος η MSA κωδικοποίηση των πρωτεϊνών και χρησιμοποίησαν 6 δίκτυα αυτού του τύπου πέτυχαν μεγαλύτερο ποσοστό επιτυχίας από τις προηγούμενες προσεγγίσεις φτάνοντας το 76%.[12,13]

Δίκτυο Deep Convolutional Neural Fields: Αυτό το δίκτυο προτάθηκε από τους S.Wang, J. Peng, J. Ma και J. Xu το 2016. Το δίκτυο τους μοιάζει με τα Deep Neural Fields (CNF) με την διαφορά ότι χρησιμοποιούν Deep Convolutional Neural Networks αντί του απλού δικτύου που χρησιμοποιούν τα CNF. Με αυτή την αλλαγή κατάφεραν να φτιάξουν ένα δίκτυο το οποίο μπορεί να κωδικοποιήσει την πολύπλοκη συσχέτιση της εισόδου του δικτύου με την επιθυμητή έξοδο. Επίσης το δίκτυο τους λαμβάνει υπόψη του αλληλουχίες μεταξύ αμινοξέων που βρίσκονται σε μεγαλύτερη απόσταση σε σύγκριση με τα CNF. Τέλος η μέθοδος τους λαμβάνει υπόψη τις συσχετίσεις μεταξύ των δευτεροταγών δομών των αμινοξέων, με αποτέλεσμα τα ποσοστά επιτυχίας τους να ξεπερνούν αυτά των προηγούμενων υλοποιήσεων, φτάνοντας το 82.3% χρησιμοποιώντας ως σύνολο δεδομένων το CB513. [14]

1.3 Νευρωνικά Δίκτυα

Τα τεχνητά νευρωνικά δίκτυα (ΤΝΔ) αποτελούν ένα μέρος του κλάδου της Τεχνητής Νοημοσύνης και προσπαθούν να μιμηθούν τον τρόπο λειτουργίας και την αρχιτεκτονική του ανθρώπινου εγκεφάλου. Δηλαδή τα ΤΝΔ είναι μαθηματικά μοντέλα τα οποία, όπως και ο εγκέφαλος, αποτελούνται από νευρώνες που επικοινωνούν και προσπαθούν βάσει των δεδομένων εισόδου τους να βγάλουν κάποια συμπεράσματα.

Ο ανθρώπινος εγκέφαλος, όπως έχουμε αναφέρει, αποτελείται από νευρώνες οι οποίοι επεξεργάζονται τα ερεθίσματα από το περιβάλλον τους στον πυρήνα τους και επικοινωνούν μεταξύ τους μέσω των συνάψεων όταν πυροδοτεί ο πυρήνας. Κύρια

χαρακτηριστικά του εγκεφάλου είναι η γρήγορη και παράλληλη επεξεργασία δεδομένων, η γενίκευση και άντληση πληροφοριών ακόμη και από ασαφή δεδομένα. Τα χαρακτηριστικά αυτά σε συνδυασμό με την ανεκτικότητα στα σφάλματα και τις λιγοστές υπολογιστικές απαιτήσεις δίνουν τη δυνατότητα στα ΤΝΔ να εφαρμοστούν σε προβλήματα όπου οι ικανότητες ενός συμβατικού υπολογιστή είναι περιορισμένες. Τέτοια προβλήματα συνήθως δεν είναι ξεκάθαρα καθορισμένα, τα δεδομένα εισόδου είναι θορυβώδη, χρειάζονται τεράστια υπολογιστική επεξεργασία, γενικεύονται δύσκολα και απαιτούν γρήγορη επεξεργασία.

1.3.1 Εκπαίδευση νευρωνικών δικτύων

Για να μπορέσει όμως ένα ΤΝΔ να λύσει οποιοδήποτε πρόβλημα πρέπει να περάσει πρώτα μία διαδικασία εκπαίδευσης. Ο ανθρώπινος εγκέφαλος αρχικά βλέπει ένα σύνολο δεδομένων και εκπαιδεύεται βάσει αυτού ώστε να βγάλει συμπεράσματα, ”κανόνες”, και να μπορέσει να τα χρησιμοποιήσει σε καινούργια δεδομένα, τα οποία δεν έχει ξανασυναντήσει. Μία παρόμοια διαδικασία ακολουθούν και τα ΤΝΔ χρησιμοποιώντας μία από τις εξής κατηγορίες μάθησης:

Επιβλεπόμενη μάθηση:

Αυτή η διαδικασία προσομοιώνει την σχέση ενός δασκάλου με τον μαθητή του και χρησιμοποιείται σε περιπτώσεις όπου γνωρίζουμε την επιθυμητή έξοδο για τα δεδομένα εισόδου. Δηλαδή όπως ένας μαθητής μελετά ένα σύνολο δεδομένων για να μάθει πώς συσχετίζονται μεταξύ τους. Στην συνέχεια εξετάζεται σε δεδομένα τα οποία δεν έχει ξαναδεί, όμως είναι παρόμοια με αυτά που έχει εκπαιδευτεί, και δίνει την απάντηση που νομίζει ότι είναι σωστή. Ο ρόλος του δασκάλου είναι να διορθώνει τον μαθητή όταν κάνει λάθος ώστε να μειωθούν τα λάθη που κάνει. Η αντίστοιχη διαδικασία για τα ΤΝΔ είναι η εξής: το δίκτυο παίρνει ένα σύνολο δεδομένων εκπαίδευσης και προσπαθεί να τα γενικεύσει αλλάζοντας τις τιμές των μεταβλητών του, προσπαθώντας με αυτό τον τρόπο να μειώσει την διαφορά της επιθυμητής εξόδου με αυτή που υπολόγισε. Όταν ολοκληρώσει την εκπαίδευσή του ελέγχεται με ένα άλλο σύνολο (ελέγχου) με δεδομένα τα οποία δεν έχει ξαναδεί και αυτό γίνεται για να αξιολογήσουμε την εκπαίδευση του

δικτύου μας. Με άλλα λόγια βλέπουμε εάν το δίκτυο έκανα καλή γενίκευση των δεδομένων ή αν τα έμαθε παπαγαλία.

Ενισχυτική μάθηση:

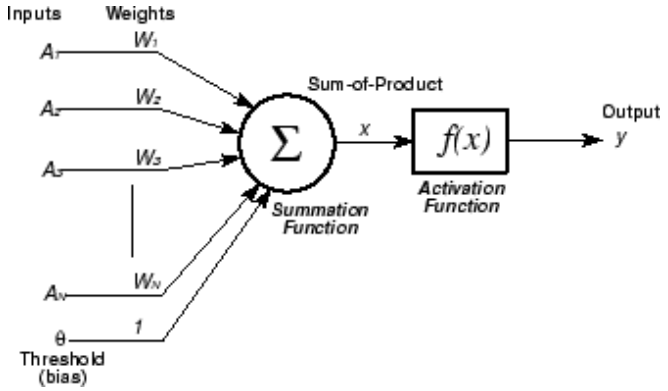
Εδώ ο δάσκαλος αντικαθιστάται με ένα κριτή και συνήθως αυτή η μέθοδος χρησιμοποιείται όταν δεν γνωρίζουμε την επιθυμητή έξοδο. Ο μαθητής, το δίκτυο, αποφασίζει ποια θα είναι η επόμενη του πράξη και ο κριτής τον τιμωρεί ή τον επιβραβεύει ανάλογα με τον αν κινείται προς την σωστή κατεύθυνση, για την επίλυση του προβλήματος. Έτσι ο μαθητής προσπαθεί να μάθει ποιες κινήσεις του επιβραβεύονται για να τις επαναλάβει και ποιες τιμωρούνται για να τις αποφύγει.

Μη επιβλεπόμενη μάθηση:

Σε αυτή την κατηγορία δεν υπάρχει ούτε δάσκαλος ούτε κριτής. Το δίκτυο παίρνει ένα σύνολο από δεδομένα και προσπαθεί να τα χωρίσει σε ομάδες με βάση τα κοινά τους χαρακτηριστικά. Μετά την εκπαίδευση του δικτύου απαιτείται και μία διαδικασία για να μπουν ετικέτες στις ομάδες για να μπορέσουμε να συνδέσουμε την είσοδο του δικτύου με την αντίστοιχη έξοδο.

1.3.2 Μοντελοποίηση νευρωνικών δικτύων

McCulloch και Pitts:



Σχήμα 1.8: Μοντέλο McCulloch και Pitts
(http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node14.html)

Οι McCulloch και Pitts μοντελοποίησαν τον πρώτο τεχνητό νευρώνα (Σχήμα 1.8) χρησιμοποιώντας ως έμπνευση τους νευρώνες του ανθρώπινου εγκεφάλου, με απώτερο σκοπό αυτός ο νευρώνας να μπορεί να γενικεύει πληροφορίες όπως οι ανθρώπινοι νευρώνες. [15]

Ο νευρώνας δέχεται ένα σύνολο από εισόδους στις οποίες αντιστοιχούν βάρη που ενδυναμώνουν ή αποδυναμώνουν την ισχύ της εισόδου. Για να υπολογίσει ο νευρώνας την είσοδό του αθροίζει το γινόμενο των εισόδων με τα βάρη όπως φαίνεται στην Εξίσωση 1.2.

$$u = \sum_{i=1}^{n} wixi$$

Εξίσωση 1.2: Εξίσωση υπολογισμού εισόδου, όπου xi η τιμή εισόδου, όπου wi η τιμή του βάρους προς την είσοδο xi.

Έπειτα οι McCulloch και Pitts χρησιμοποίησαν την βηματική συνάρτηση, Εξίσωση 1.3, ως συνάρτηση κατωφλίου για να υπολογίσει ο νευρώνας την έξοδό του. Αν η τιμή εισόδου του είναι μεγαλύτερη ή ίση από την τιμή του κατωφλίου, τότε η έξοδος του είναι 1 και πυροδοτεί, σε αντίθετη περίπτωση η έξοδός του είναι 0 και δεν κάνει τίποτα.

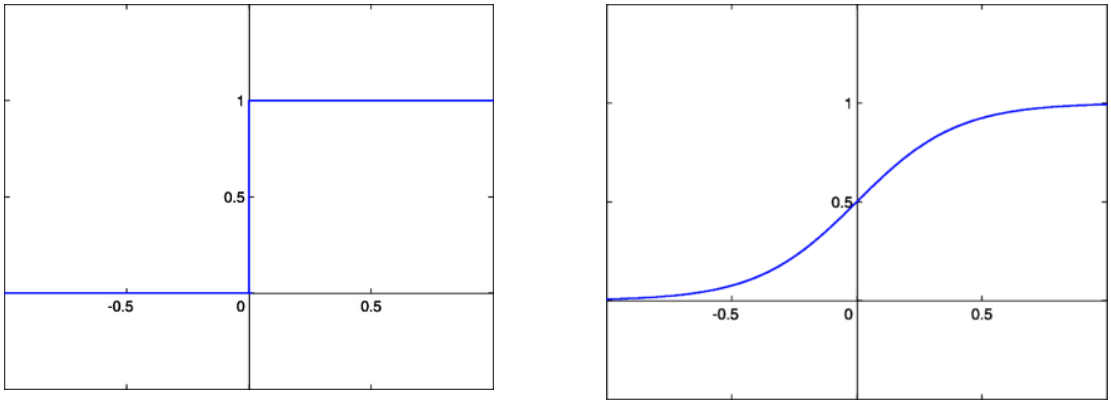
$$y = \begin{cases} 1 & \text{if } u \geq \theta \\ 0 & \text{if } u < \theta \end{cases}$$

Εξίσωση 1.3:Βηματική συνάρτηση κατωφλίου, όπου u το άθροισμα του γινομένου των εισόδων και των βαρών του νευρώνα, όπου θ η τιμή του κατωφλίου, όπου y η έξοδος του δικτύου και παίρνει την τιμή 0 ή 1.

Αργότερα ανακαλύφθηκε ότι η βηματική συνάρτηση (Σχήμα 1.9 αριστερά) είχε κάποια μειονεκτήματα τα οποία επηρέαζαν την εκπαίδευση του νευρώνα. Το πρώτο μειονέκτημα είναι ότι η βηματική συνάρτηση δεν είναι παραγωγίσιμη άρα δεν μπορεί μέσω του αλγόριθμου μάθησης να υπολογίσει την σωστή απόκλιση λάθους. Επίσης, η συνάρτηση παίρνει μόνο δύο απόλυτες τιμές 0 ή 1 άρα οι νευρώνες δεν έχουν καμία ένδειξη για το πόσο μεγάλη είναι η απόκλιση τους από την επιθυμητή έξοδο για να προσαρμόσουν ανάλογα τα βάρη τους. Για αυτό το λόγω η συνάρτηση κατωφλίου αντικαταστήθηκε με παραγωγίσιμες συναρτήσεις με την σιγμοειδή συνάρτηση (Εξίσωση 1.4 και Σχήμα 1.9 δεξιά) να είναι η δημοφιλέστερη από αυτές.

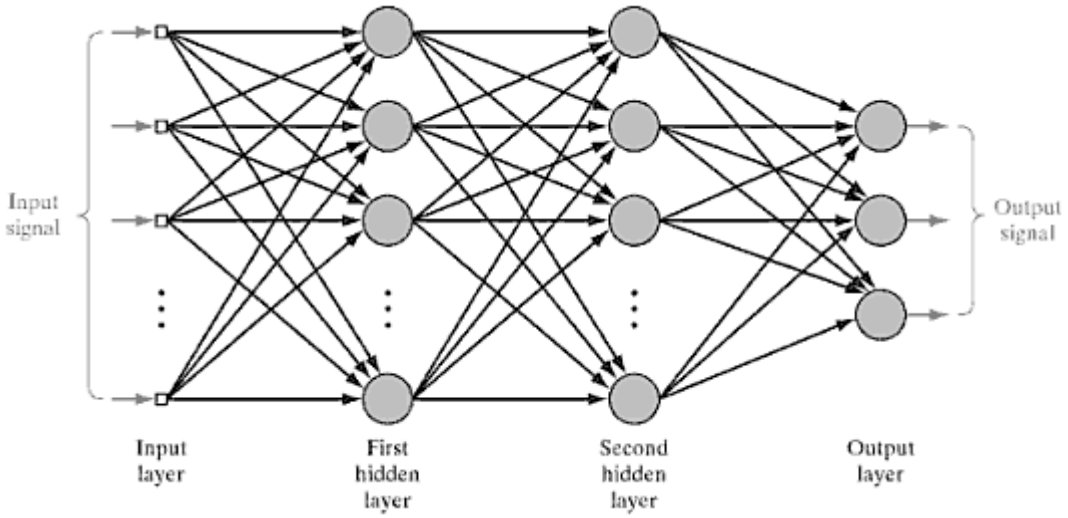
$$f(x) = \frac{1}{1 + e^{-ax}}$$

Εξίσωση 1.4: Σιγμοειδής συνάρτηση κατωφλίου, όπου α η κλίση της συνάρτησης, όπου x το άθροισμα του γινομένου των εισόδων και των βαρών του νευρώνα.



Σχήμα 1.9: Βηματική συνάρτηση (αριστερά) και Σιγμοειδής συνάρτηση (δεξιά)
(https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions)

Πολυστρωματικά δίκτυα perceptron εμπρόσθιου περάσματος:



Σχήμα 1.10: Πολυστρωμαικό δίκτυο perceptron με δύο κρυφά επίπεδα
(https://elogeel.wordpress.com/2010/05/05/multilayer-perceptron/)

Τα πολυστρωματικά δίκτυα Perceptron εμπρόσθιου περάσματος αποτελούνται από νευρώνες McCulloch και Pitts και βάρη που ενώνουν τα επίπεδα μεταξύ τους. Όπως φαίνεται και στο Σχήμα 1.10 το δίκτυο αποτελείται από το ανενεργό επίπεδο εισόδου, γιατί απλά μεταφέρει την είσοδο του δικτύου στο επόμενο επίπεδο, και τα ενεργά κρυφά επίπεδα (βρίσκονται μεταξύ επιπέδου εισόδου και εξόδου) και το επίπεδο εξόδου. Το πλήθος των νευρώνων εισόδου και εξόδου καθορίζονται από το πρόβλημα. Ενώ ο αριθμός των κρυφών επιπέδων καθώς και το πλήθος των νευρώνων τους, καθορίζονται πειραματικά και είναι ανάλογα του προβλήματος που προσπαθούν να λύσουν, μιας και αυτοί είναι οι νευρώνες που αποθηκεύουν την γνώση για την κατηγοριοποίηση των δεδομένων εισόδου.

Η εκπαίδευση του δικτύου γίνεται με επιβλεπόμενη μάθηση με βάσει τον παρακάτω αλγόριθμο:

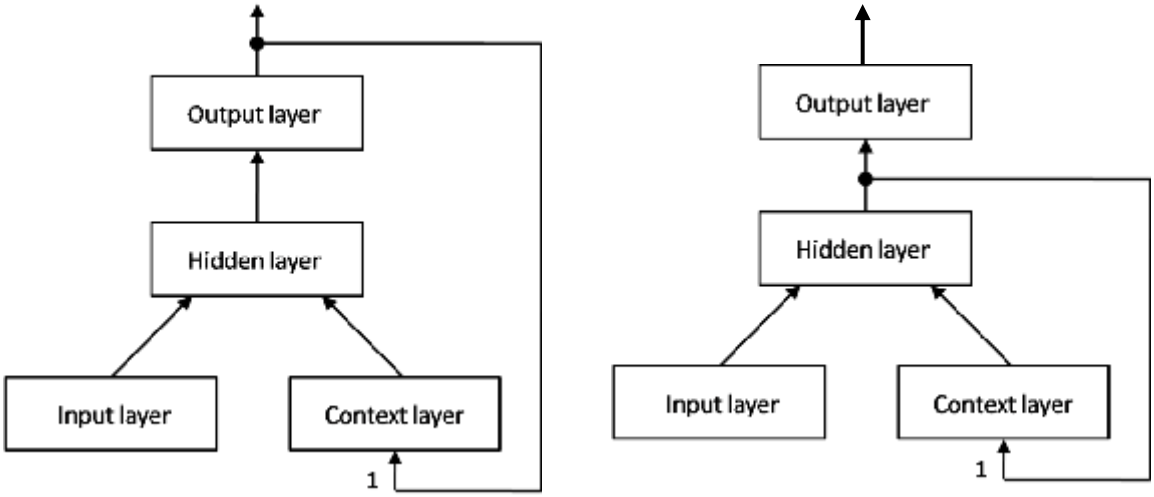
1. Αρχικοποίηση των βαρών και κατωφλίου με μικρές τυχαίες τιμές
2. Παρουσίαση εισόδου και επιθυμητής εξόδου στο δίκτυο
3. Υπολογισμός εξόδου

   $y(t) = f\left(\sum_{i=1}^{n} wixi\right)$, όπου f(.) η συνάρτηση κατωφλίου
4. Αλλαγή βαρών

   $w_i(t+1) = w_i(t) + \eta \Delta x_i(t)$, όπου η ρυθμός μάθησης, $0 \leq \eta \leq 1$

   $\Delta = d(t) - y(t)$, όπου d(t) η επιθυμητή έξοδος
5. Επανάλαβε από το βήμα 2 για όλα τα δεδομένα εκπαίδευσης

Πολυστρωματικά δίκτυα perceptron με ανάδραση:



Σχήμα 1.11: Δίκτυο Jordan (αριστερά) και δίκτυο Elman (δεξιά)
(http://www.ijser.org/paper/Recurrent-Neural-Prediction-Model-for-Digits-Recognition.html)

Τα δίκτυα με ανάδραση (Σχήμα 1.11) μοιάζουν δομικά με τα δίκτυα εμπρόσθιου περάσματος με την διαφορά ότι υπάρχουν βάρη μεταξύ άλλων επιπέδων του δικτύου και της εισόδου. Αυτό σημαίνει ότι η είσοδος του δικτύου δεν εξαρτάται μόνο από τα δεδομένα εισόδου αλλά και από τις εξόδους των νευρώνων την προηγούμενη χρονική στιγμή. Με αυτό τον τρόπο ανατροφοδοτείται πληροφορία στο δίκτυο και εισάγονται οι έννοιες του χρόνου και της βραχυπρόθεσμης μνήμης. Άρα τα δίκτυα με ανάδραση είναι ιδανικά για δυναμικά δεδομένα, δηλαδή τα δεδομένα δεν είναι ανεξάρτητα μεταξύ τους και απαιτούν να διατηρείται κάποιο ιστορικό στο δίκτυο.

Μία τέτοια αρχιτεκτονική πρότεινε και ο Jordan όπου τα βάρη ξεκινούν από το επίπεδο εξόδου και καταλήγουν στο context layer, όπως φαίνεται στο Σχήμα 1.11 αριστερά. Τα βάρη που συνδέουν το επίπεδο εξόδου με το context layer έχουν σταθερές τιμές έτσι ώστε να μην αλλοιώνουν τις εξόδους του δικτύου. Επίσης στο context layer υπάρχει μία σταθερά που χρησιμοποιείται για να εξασθενεί η μνήμη του δικτύου. Όσο μεγαλύτερη είναι η τιμή αυτής της σταθεράς τόσο περισσότερο διατηρείται η πληροφορία στο δίκτυο.

Παρόμοια αρχιτεκτονική πρότεινε και ο Elman (Σχήμα 1.11 δεξιά), ο οποίος όμως παρατήρησε ένα μειονέκτημα στην αρχιτεκτονική του Jordan και αποφάσισε να το διορθώσει. Για αυτό πρότεινε τα βάρη να ξεκινούν από το κρυφό επίπεδο και όχι από επίπεδο εξόδου. Η ιδέα του πίσω από αυτή την αλλαγή ήταν ότι το επίπεδο εξόδου περιορίζεται από την επιθυμητή έξοδο άρα δεν προσφέρει αρκετή γνώση στο δίκτυο και η γνώση του γίνεται ανεξάρτητη της εξόδου του δικτύου. [16]

1.3.3 Ανάστροφη μετάδοση λάθους

Ο αλγόριθμος μάθησης ανάστροφης μετάδοσης λάθους (Backpropagation) είναι ένας από τους πιο διαδεδομένους αλγόριθμους επιβλεπόμενης μάθησης (γνωρίζουμε την επιθυμητή έξοδο) για πολυστρωματικά δίκτυα perceptron. Χαρακτηριστικό αυτού του αλγορίθμου είναι ότι κάνει δύο περάσματα του δικτύου, το πρώτο από αριστερά προς δεξιά και το δεύτερο από δεξιά προς αριστερά.

Στο πρώτο πέρασμα τα δεδομένα εισόδου παρουσιάζονται στο δίκτυο και υπολογίζονται οι έξοδοι του κάθε επιπέδου. Στην συνέχεια υπολογίζεται το σφάλμα του δικτύου με την μέθοδο των ελάχιστων μέσων τετραγώνων (Εξίσωση 1.5) συγκρίνοντας την διαφορά της εξόδου του δικτύου με την επιθυμητή έξοδο. Έπειτα ξεκινά η δεύτερη φάση του αλγορίθμου για ενημέρωση των βαρών του δικτύου ανάλογα με το σφάλμα που υπολογίστηκε και τον κανόνα δέλτα. Ο υπολογισμός του κανόνα δέλτα για το επίπεδο εξόδου φαίνεται στην Εξίσωση 1.6 και για τα κρυφά επίπεδα στην Εξίσωση 1.7. [17]

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

Εξίσωση 1.5: Μέσο τετραγωνικό σφάλμα, όπου tpj η επιθυμητή είσοδος και opj η πραγματική έξοδος του νευρώνα.

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

Εξίσωση 1.6: Υπολογισμός του δέλτα των νευρώνων εξόδου, όπου tpj η επιθυμητή είσοδος και opj η πραγματική έξοδος του νευρώνα.

$$\delta_{pj} = O_{pj} (1- O_{pj}) \Sigma \delta_{p\kappa} W_{j\kappa}$$

Εξίσωση 1.7: Υπολογισμός του δέλτα των νευρώνων των κρυφών επιπέδων, όπου opj η έξοδος του νευρώνα και όπου δ το δέλτα του νευρώνα κ και w_{jk} το αντίστοιχο βάρος.

$$w_{ij} (t+1) = w_{ij} (t)+\eta \delta_{pj} o_{pi} (t)$$

Εξίσωση 1.8: Εξίσωση αλλαγής βαρών, όπου wij(t+1) η νέα τιμή του βάρους wij, wij(t) η προηγούμενη τιμή του βάρους wij, η ο ρυθμός μάθησης, Δpj η τιμή δέλτα του νευρώνα p και opi(t) η έξοδος του νευρώνα.

Ο αλγόριθμος ανάστροφης μετάδοσης λάθους με λεπτομέρεια φαίνεται πιο κάτω:

1. Αρχικοποίηση των βαρών και κατωφλίου με μικρές τυχαίες τιμές
2. Παρουσίαση εισόδου και επιθυμητής εξόδου στο δίκτυο
3. Υπολογισμός εξόδου

   $y_{pj} (t) = f (\sum_{i=1}^{n} wixi)$, όπου f(.) η συνάρτηση κατωφλίου

   Δώσε την έξοδο αυτού του επιπέδου ως είσοδο για το επόμενο
4. Αλλαγή βαρών, ξεκινώντας από το επίπεδο εξόδου προς το πρώτο κρυφό επίπεδο

   $w_{ij} (t+1) = w_{ij} (t) + \eta \Delta_{pj} o_{pi} (t)$, όπου η ρυθμός μάθησης, $0 \leq \eta \leq 1$

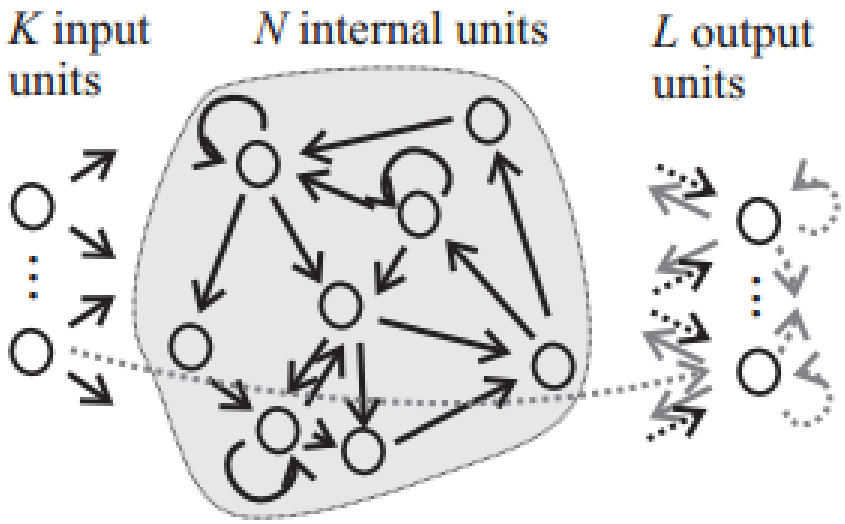
   Αν είσαι στο επίπεδο εξόδου για τον υπολογισμό του Δ_{pj} χρησιμοποίησε την Εξίσωση 1.6 αλλιώς χρησιμοποίησε την Εξίσωση 1.7
5. Επανάλαβε από το βήμα 2 για όλα τα δεδομένα εκπαίδευσης

Κεφάλαιο 2

Γνωσιολογικό υπόβαθρο και προηγούμενη εργασία

2.1 Echo State Network (ESN)

Το Echo State Network (ESN) (Σχήμα 2.1) προτάθηκε για πρώτη φορά από τον Herbert Jaeger το 2001 [18,19] και ανήκει στην κατηγορία του Reservoir Computing. Συγκεκριμένα το ESN είναι ένα τεχνητό νευρωνικό δίκτυο με ανάδραση διακριτού χρόνου, με νευρώνες που χρησιμοποιούν την σιγμοειδή συνάρτηση και εκπαιδεύεται με την τεχνική της επιβλεπόμενης μάθησης. Η χρήση του έγινε ευρέος γνωστή λόγω της απλότητας στην κατανόηση του αλλά και στην πολύ καλή του επίδοση. Η εκπαίδευσή του είναι αρκετά γρήγορη, σε σύγκριση με άλλα νευρωνικά δίκτυα με ανάδραση, εφόσον ανανεώνονται μόνο τα βάρη μεταξύ του reservoir και του επιπέδου εξόδου. Επίσης το κόστος υλοποίησης του είναι χαμηλότερο λόγω της απλής του αρχιτεκτονικής.



Σχήμα 2.1: Βασική αρχιτεκτονική ESN, K είναι το σύνολο των νευρώνων εισόδου, N το σύνολο των νευρώνων στο Dynamic Reservoir και L το σύνολο των νευρώνων εξόδου. Οι συμπαγής γραμμές συμβολίζουν τα βάρη που είναι απαραίτητα για την λειτουργία του δικτύου ενώ οι διακεκομμένες μαύρες γραμμές τα βάρη που δεν είναι απαραίτητα. Με μαύρη διακεκομμένη γραμμή φαίνονται τα βάρη τα οποία εκπαιδεύονται. [19]

2.1.1 Αρχιτεκτονική ESN

Όπως φαίνεται και από το Σχήμα 2.1 το δίκτυο χωρίζεται σε τρία μέρη, το επίπεδο εισόδου, το Dynamic Reservoir (DR) και το επίπεδο εξόδου. Το επίπεδο εισόδου και εξόδου εξαρτώνται από το πρόβλημα. Αφού αποφασίσαμε να κωδικοποιήσουμε τα

δεδομένα με την μέθοδο MSA (Υποκεφάλαιο 2.2) θα χρειαστούμε 20 νευρώνες στο επίπεδο εισόδου. Επίσης έχουμε τρεις πιθανές επιλογές (α-έλικας, β-πτυχωτή επιφάνεια ή τυχαία αναδίπλωση) για την δευτεροταγή δομή της πολυπεπτιδικής αλυσίδας, για αυτό το λόγο θα χρειαστούμε 3 νευρώνες στο επίπεδο εξόδου. Ο αριθμός των νευρώνων στο DR συνήθως καθορίζεται πειραματικά ανάλογα με την πολυπλοκότητα του προβλήματος.

Επιπρόσθετα στο Σχήμα 2.1 φαίνονται και τα βάρη του δικτύου μεταξύ των επιπέδων. Υπάρχουν (K+1)×N βάρη τα οποία συνδέουν το επίπεδο εισόδου με το DR, όπου K+1 είναι ο αριθμός των νευρώνων εισόδου μαζί με το κατώφλι. Οι νευρώνες στο DR είναι πλήρως συνδεδεμένοι, άρα έχουμε ακόμη N×N βάρη. Τέλος οι νευρώνες του DR ενώνονται με κάθε νευρώνα από το επίπεδο εξόδου, δηλαδή επιπρόσθετα N× L βάρη. Όπως αναφέρθηκε και πιο πάνω μόνο τα βάρη από το DR προς το επίπεδο εξόδου αλλάζουν τιμή, για αυτό τα υπόλοιπα βάρη παίρνουν τιμή πριν την εκτέλεση του αλγορίθμου και την διατηρούν μέχρι το τέλος.

Το πιο σημαντικό κομμάτι της δημιουργίας ενός ESN είναι η επίτευξη της ιδιότητας echo state στο DR. Ένα DR που έχει αυτή την ιδιότητα λειτουργεί ως μνήμη, αλλά μπορεί και να διαχωρίσει τα μη γραμμικά διαχωρίσιμα δεδομένα εισόδου. Αποτελεί μία ξεθωριασμένη μνήμη (fading memory), δηλαδή το DR θυμάται (χαρτογραφεί) τις εισόδους που έχει δει, καθώς και την αντίστοιχη έξοδο, τα οποία επηρεάζουν την εκπαίδευση του δικτύου. Όμως η επιρροή τους εξασθενεί με το πέρασμα του χρόνου, για αυτό και ονομάζεται ξεθωριασμένη μνήμη. Επίσης το DR μπορεί να πάρει τα δεδομένα εισόδου, τα οποία δεν είναι γραμμικά διαχωρίσιμα, να τα επεκτείνει σε μία υψηλότερη διάσταση στην οποία να είναι γραμμικά διαχωρίσιμα και με βάση την κατάστασή του να αποφασίσει την κατάλληλη έξοδο. [18,20]

Τα χαρακτηριστικά που προσφέρει το DR στο δίκτυο είναι απαραίτητα για την επίλυση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Όπως έχουμε αναφέρει μεταξύ των αμινοξέων της πολυπεπτιδικής αλυσίδας ασκούνται δυνάμεις. Αυτό σημαίνει ότι το δίκτυο πρέπει να λαμβάνει υπόψη στην απόφαση του για κάθε αμινοξύ τα γειτονικά του αμινοξέα. Ακόμη μετά την εκπαίδευση του, το

δίκτυο πρέπει να θυμάται τα χαρακτηριστικά των εισόδων και την κατηγορία που τα κατέταξε έτσι ώστε να μπορέσει να κατατάξει τα νέα άγνωστα δεδομένα.
Οι παράμετροι που συμβάλουν στην κατασκευή ενός DR με αυτά τα χαρακτηριστικά είναι[18,20,21]:

i. Μέγεθος DR:

Γενικά υιοθετείται η ιδέα ότι όσο μεγαλύτερο είναι το μέγεθος του, τόσο καλύτερη είναι η απόδοσή του, αφού η εκπαίδευση και εκτέλεση τους είναι γρηγορότερη από άλλα παρόμοια δίκτυα υπάρχει η δυνατότητα να υιοθετηθεί ένα μεγάλο DR. Αυτό σημαίνει ότι είναι ευκολότερο να βρεθεί ένα μονοπάτι που αντιστοιχεί τα χαρακτηριστικά της εισόδου στην επιθυμητή έξοδο. Όμως ένα πολύ μεγάλο DR μπορεί να οδηγήσει στο φαινόμενο της υπερεκπαίδευσης. Για αυτό συνήθως αρχικά επιλέγεται ένα μικρό DR, αρκετά μεγάλο όμως για να θυμάται τις τιμές που χρειάζονται για την κατηγοριοποίηση των δεδομένων, ώστε να βρεθούν οι κατάλληλες τιμές των υπόλοιπων παραμέτρων που επηρεάζουν το DR. Μετά πειραματικά να μπορεί να επιλεχθεί μεγαλύτερο μέγεθος για το DR, εφόσον οι κατάλληλες τιμές των άλλων παραμέτρων θα έχουν ήδη καθοριστεί για αυτό το πρόβλημα και θα λειτουργούν και στο μεγαλύτερο DR.

ii. Πυκνότητα (sparsity) DR:

Αυτή η μεταβλητή δηλώνει πόσο αραιά/πυκνά είναι συνδεδεμένοι οι νευρώνες του DR. Πειραματικά έχει βρεθεί ότι ένα αραιά συνδεδεμένο DR έχει καλύτερα αποτελέσματα, όμως δεν έχει μεγάλη επίδραση στην απόδοση του δικτύου για αυτό δεν είναι πολύ σημαντική η τιμή της. Όσο η τιμή αυτής της μεταβλητής πλησιάζει το 1 τόσο πιο αραιά συνδεδεμένο είναι το DR.

iii. Κατανομή βαρών:

Έχουμε αναφέρει ότι τα βάρη από το επίπεδο εισόδου προς το DR και τα βάρη μεταξύ των νευρώνων του DR αρχικοποιούνται τυχαία και παραμένουν σταθερά. Μία καλή πρακτική είναι να κανονικοποιηθούν οι τιμές τους ώστε το DR να μπορεί να αποδώσει καλύτερα, αφού το πεδίο τιμών που θα επεξεργάζεται είναι περιορισμένο.

iv. Φασματική ακτίνα (spectral radius) DR:

Η επιλογή της σωστής τιμής αυτής της παραμέτρου είναι υψίστης σημασίας, γιατί αυτή είναι που καθορίζει την επιτυχία της echo state ιδιότητας του DR. Δηλαδή η τιμή της καθορίζει πόσο γρήγορα «πεθαίνει» η επίδραση μίας εισόδου στο DR. Μία μικρή τιμή, μικρότερη της μονάδας, σημαίνει γρήγορη εξασθένηση, ενώ μία τιμή κοντά στην μονάδα αργή. Υπάρχουν και περιπτώσεις που παίρνει τιμές πέραν της μονάδας. Σε αυτήν την περίπτωση σημαίνει ότι εξαρτάται από ολόκληρη την ιστορία καταστάσεων, όμως τα βάρη του δικτύου θα αλλάζουν τιμές ανεξέλεγκτα με αποτέλεσμα να μην εκπαιδεύεται.

v. Ποσοστό διαρροής α (leaking rate) DR:

Η μεταβλητή α δηλώνει την ταχύτητα με την οποία ανανεώνεται το DR με την πάροδο του χρόνου. Δηλαδή το χρόνο που χρειάζονται οι νευρώνες του DR να πάρουν την ιδανική τιμή, η οποία θα μετατρέπει την έξοδο του δικτύου στην επιθυμητή έξοδο. Η τιμής της βέβαια εξαρτάται από το κάθε πρόβλημα, για αυτό η ιδανική τιμή καθορίζεται πειραματικά.

Τέλος να αναφέρουμε ότι το DR σε αυτή την εργασία δημιουργείται τυχαία σύμφωνα με τις παραμέτρους του δικτύου, όπως έκανε και ο Jaeger στην αρχική του προσέγγιση. Αυτό έχει τις εξής συνέπειες: δεν γνωρίζουμε σίγουρα ότι το DR που θα δημιουργηθεί θα έχει τις επιθυμητές ιδιότητες. Δηλαδή ότι θα υπάρχουν αναδρομικά βάρη ώστε να υπάρχει μνήμη στο δίκτυο και η echo state ιδιότητα. Τέλος δεν υπάρχει η εγγύηση ότι θα βρεθούν μονοπάτια που να οδηγούν την είσοδο του δικτύου στην επιθυμητή έξοδο. Ως εκ τούτου η δημιουργία ενός DR με κάποιες ιδιότητες οι οποίες εξασφαλίζουν τις παραπάνω προϋποθέσεις για την επιτυχία του θα έχουν αποτέλεσμα την αύξηση της επιτυχίας του δικτύου. Όμως αυτή είναι μία εύκολη υλοποίηση η οποία θα μας δώσει μίαν αρχική ένδειξη για τις ικανότητες του δικτύου για αυτή την εφαρμογή.

2.1.2 Εκπαίδευση ESN

Το ESN εκπαιδεύεται με επιβλεπόμενη μάθηση, δηλαδή το δίκτυο γνωρίζει την επιθυμητή έξοδο. Σκοπός μας είναι το δίκτυο να εκπαιδευτεί και οι τιμές που

υπολογίζει να είναι όσο το δυνατό πιο κοντά στο επιθυμητό αποτέλεσμα. Δηλαδή πρέπει να γενικεύσει τα δεδομένα εισόδου που έχει δει, έτσι ώστε να μπορεί να προβλέπει την δευτεροταγή δομή και άγνωστων προς αυτό δεδομένων. Η επιτυχία της εκπαίδευσης υπολογίζεται με την συνάρτηση σφάλματος (Εξίσωση 2.1), της οποίας θέλουμε να μειώσουμε την τιμή όσο περισσότερο και ως εκ τούτου να αυξήσουμε το ποσοστό επιτυχίας του αλγορίθμου.

Παρακάτω θα παρουσιαστούν οι εξισώσεις που χρησιμοποιεί ο αλγόριθμος κατά την διάρκεια της εκπαίδευσης του.[20]

Στις εξισώσεις χρησιμοποιείται η εξής κωδικοποίηση:

n: Συμβολίζει τον αριθμό της πρωτεΐνης που παίρνει ως είσοδο το δίκτυο. Το n παίρνει τιμές από το 1 μέχρι το T, όπου το T είναι το πλήθος των πρωτεϊνών που περιέχει το αρχείο εκπαίδευσης του δικτύου. Χρησιμοποιείται για αναφορά σε χρονική στιγμή, δηλαδή τι γίνεται στο δίκτυο όταν έχει ως είσοδο την n πρωτεΐνη.

2.1.2.1 Συνάρτηση σφάλματος

Η συνάρτηση υπολογισμού σφάλματος που χρησιμοποιείται είναι η Mean Square Error (MSE). Η Εξίσωση 2.1 υπολογίζει το άθροισμα των τετραγώνων της διαφοράς της επιθυμητής εξόδου από αυτή που υπολόγισε το δίκτυο για όλα τα δεδομένα που έχει δει μέχρι τα ώρα και το διαιρεί με το πλήθος τους.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

Εξίσωση 2.1: Εξίσωση υπολογισμού σφάλματος MSE, n είναι το πλήθος των πλήθος δεδομένων εισόδου, $\hat{y}i$ είναι η επιθυμητή έξοδος και yi είναι η έξοδος που υπολόγισε το δίκτυο.

2.1.2.2 Συναρτήσεις αναβάθμισης

Αναβάθμιση νευρώνων του DR:

Οι νευρώνες του DR ανανεώνουν την τιμή τους βάσει της Εξίσωσης 2.2. Σε αυτή την περίπτωση ως σιγμοειδής συνάρτηση αναβάθμισης επιλέχθηκε η υπερβολική εφαπτομένη.

$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\tilde{\mathbf{x}}(n)$$

Εξίσωση 2.2: Εξίσωση ανανέωσης νευρώνων του DR κατά τα την χρονική στιγμή n, x (n) νέα τιμή του νευρώνα του DR, x(n-1) η τιμή του νευρώνα στην προηγούμενη χρονική στιγμή, α προκαθορισμένη τιμή του leaking rate α ∈ (0, 1] και $\tilde{\mathbf{x}}(n)$ η ποσότητα ανανέωσης της τιμής του νευρώνα. [20]

$$\tilde{\mathbf{x}}(n) = \tanh\left(\mathbf{W}^{\text{in}}[1;\mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)\right)$$

Εξίσωση 2.3: Εξίσωση υπολογισμού ποσότητας ανανέωσης των νευρώνων του DR κατά την χρονική στιγμή n, $\tilde{\mathbf{x}}(n)$ η ποσότητα ανανέωσης της τιμής του νευρώνα ,tahn() η συνάρτηση αναβάθμισης, W^{in} πίνακας με τις τιμές των βαρών εισόδου, $[1;\mathbf{u}(n)]$ πίνακας που περιέχει το κατώφλι και τις τιμές εισόδου του δικτύου $\mathbf{u}(n)$, W οι τιμές των βαρών του DR και x(n-1) η τιμή του νευρώνα στην προηγούμενη χρονική στιγμή. [20]

Δηλαδή η νέα τιμή, x (n), των νευρώνων του DR εξαρτάται από την προηγούμενη τους τιμή, x(n-1), την ποσότητα $\tilde{\mathbf{x}}(n)$ ανανέωσης και το leaking rate a.

Σημαντικό είναι να αναφερθεί ότι έχει χρησιμοποιηθεί η τεχνική του initial transient στην εκπαίδευση του δικτύου. Αυτή η τεχνική προτείνει να μην λαμβάνονται υπόψη στην εκπαίδευση των βαρών εξόδου οι αρχικές αλλαγές στις τιμές των νευρώνων του DR. Αυτό συμβαίνει λόγω της τυχαίας αρχικοποίησης, x(0), των τιμών των νευρώνων ώστε να μπορούν να υπολογιστούν οι Εξισώσεις 2.2 και 2.3. Αυτό σημαίνει ότι η αρχική κατάσταση του δικτύου δεν είναι μία «φυσική» κατάσταση του προβλήματος, για αυτό του δίνουμε μερικούς γύρους ώστε να φτάσει σε μία κατάσταση πιο «φυσιολογική» και να μη επηρεάσει αρνητικά τα βάρη εξόδου.

Αναβάθμιση νευρώνων εξόδου:

Οι νευρώνες εξόδου ανανεώνουν την τιμή τους βάσει της Εξίσωσης 2.4.

$$\mathbf{y}(n) = \mathbf{W}^{\text{out}}[1; \mathbf{u}(n); \mathbf{x}(n)]$$

Εξίσωση 2.4: Εξίσωση αναβάθμισης νευρώνων εξόδου κατά την χρονική στιγμή n, y (n) η νέα τιμή του νευρώνα εξόδου, W^{out} πίνακας με τις τιμές των βαρών μεταξύ του DR και των νευρώνων εξόδου, $[1; u(n); x(n)]$ πίνακας που περιέχει το κατώφλι, τις τιμές εισόδου του δικτύου $u(n)$ και τις τιμές x (n) των νευρώνων του DR. [20]

Αλλαγή βαρών εξόδου:

Τα βάρη εξόδου αλλάζουν την τιμή τους βάσει της Εξίσωσης 2.5.

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}}\mathbf{X}^{\text{T}}\left(\mathbf{X}\mathbf{X}^{\text{T}} + \beta\mathbf{I}\right)^{-1}$$

Εξίσωση 2.5: Εξίσωση αλλαγής βαρών εξόδου W^{out} ,Y^{target} πίνακας με τις τιμές των νευρώνων εξόδου, X πίνακας που περιέχει όλους τους πίνακες με το κατώφλι, τις τιμές εισόδου του δικτύου $u(n)$ και τις τιμές x(n) των νευρώνων του DR που έχουν παραχθεί για την συγκεκριμένη είσοδο του δικτύου, X^T ο ανάστροφος πίνακας του X, β η μεταβλητή regression και I είναι ο μοναδιαίος πίνακας. [20]

Regression β:

Η μεταβλητή Regression β χρησιμεύει στην αποφυγή της υπερεκπαίδευσης του δικτύου.

Πιο πάνω παρουσιάστηκε ο κλασικός αλγόριθμος batch mode, στον οποίο τα βάρη ενημερώνονται στο τέλος αφού το δίκτυο έχει δει όλα τα δεδομένα εκπαίδευσης. Πιο κάτω θα παρουσιαστεί ο online αλγόριθμος, στον οποίο τα βάρη ενημερώνονται μετά από κάθε δεδομένο εισόδου. Μετά από δοκιμές αποδείχθηκε ότι αυτός ο αλγόριθμος αποδίδει καλύτερα από τον κλασικό. Για αυτό το λόγο σχεδιάστηκε ένα δίκτυο το οποίο χρησιμοποιεί και τους δύο αλγόριθμους.

Online αλγόριθμος:

Ο online αλγόριθμος που χρησιμοποιήθηκε είναι ο Recursive Least Square (RLS) , ο οποίος ενσωματώνει το ιστορικό του σφάλματος στον υπολογισμό του παρόντος σφάλματος. Δηλαδή ανανεώνει τα βάρη μετά από κάθε χρονική στιγμή.[22]

Συνάρτηση σφάλματος:

$$E(k) = \sum_{i=1}^{k} \lambda^{k-i} e(k)^2$$

Εξίσωση 2.6: Εξίσωση υπολογισμού σφάλματος αλγόριθμου RLS, λ forgetting factor, e (k) η διαφορά της επιθυμητής εξόδου και της εξόδου που υπολόγισε το δίκτυο την τρέχουσα χρονική στιγμή k. [22]

$$e(k) = y_{target}(k) - y(k)$$

Εξίσωση 2.7: Εξίσωση που υπολογίζει το παρόν σφάλμα e(k), $y^{target}(k)$ η επιθυμητή έξοδος, y(k) η έξοδος του δικτύου, k η τρέχουσα χρονική στιγμή. [22]

Συναρτήσεις αναβάθμισης:

$$\mathbf{w}_{out}(k+1) = \mathbf{w}_{out}(k) + e(k)\mathbf{g}(k)$$

Εξίσωση 2.8: Εξίσωση αλλαγής βαρών εξόδου $W^{out}(k+1)$ για την επόμενη χρονική στιγμή, $W^{out}(k)$ η τιμή του βάρους εξόδου την τρέχουσα χρονική στιγμή, e(k)το παρόν σφάλμα, η τιμή g(k)δείχνει την σημαντικότητα της ιστορίας των σφαλμάτων. [22]

$$\mathbf{g}(k) = \frac{\mathbf{P}(k-1)\mathbf{x}(k)}{\lambda + \mathbf{x}(k)^T \mathbf{P}(k-1)\mathbf{x}(k)}$$

Εξίσωση 2.9: Εξίσωση που υπολογίζει την σημαντικότητα της ιστορίας των σφαλμάτων g(k), P(k - 1) πίνακας με την ιστορία του δικτύου, x(k) οι τιμές των νευρώνων του DR την τρέχουσα χρονική στιγμή k. [22]

$$\mathbf{P}(k) = \lambda^{-1}\mathbf{P}(k-1) - \mathbf{g}(k)\mathbf{x}^{T}(k)\lambda^{-1}\mathbf{P}(k-1)$$

Εξίσωση 2.10: Αναδρομική εξίσωση που υπολογίζει την ιστορία P(k)των σφαλμάτων του δικτύου, λ forgetting factor, P(k-1) η ιστορία του δικτύου την προηγούμενη χρονική στιγμή, η τιμή g(k)δείχνει την σημαντικότητα της ιστορίας των σφαλμάτων, x^T (k) ο ανάστροφος πίνακας των νευρώνων του DR την τρέχουσα χρονική στιγμή k. [22]

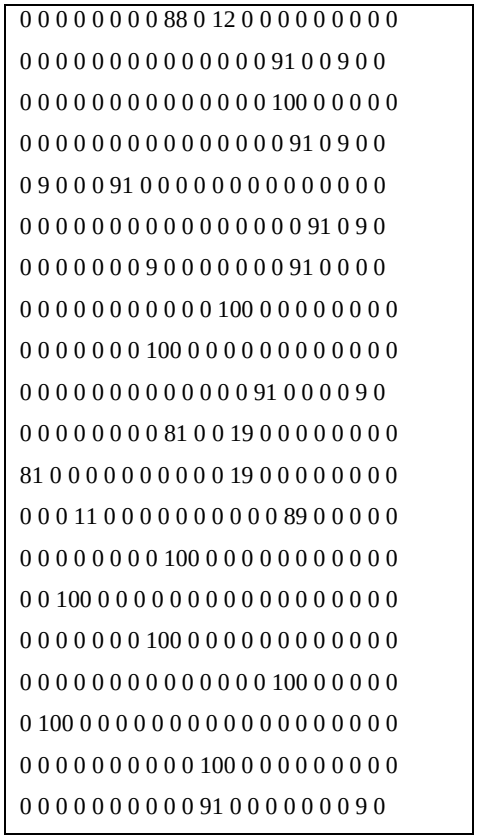
Forgetting factor λ:

Η μεταβλητή λ μειώνει εκθετικά την σημαντικότητα της ιστορίας των σφαλμάτων του δικτύου. Δηλαδή μία τιμή κοντά στην μονάδα δίνει την ίδια βαρύτητα σε όλη την ιστορία, συμπεριλαμβανομένου της τρέχουσας κατάστασης. Μία μικρότερη τιμή μειώνει εκθετικά την σημασία μίας κατάστασης, αυτό σημαίνει ότι κάτι που συνέβηκε στην χρονική στιγμή k έχει μεγαλύτερη σημασία από κάτι που συνέβηκε k-1. Η επιλογή της τιμής της μεταβλητής είναι πολύ σημαντική γιατί συμβάλει στην σταθεροποίηση του δικτύου. Για αυτό το λόγο επιλέγονται τιμές κοντά στην μονάδα.

2.2 Περιγραφή δεδομένων εισόδου

Όσο πιο ρεαλιστική είναι η αναπαράσταση των δεδομένων εισόδου του νευρωνικού δικτύου σε σύγκριση με τα πραγματικά δεδομένα, τόσο πιο αποτελεσματική είναι η εκπαίδευσή του. Για αυτό το λόγο επιλέχθηκε η τεχνική της πολλαπλής ευθυγραμμισμένης ακολουθίας πρωτεϊνών (MSA - Multiple Sequence Alignment) για την εκπαίδευση του ESN.

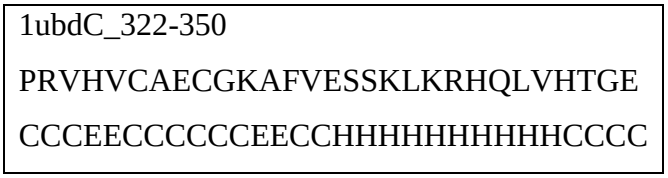
Αυτός ο τρόπος αναπαράστασης των πρωτεϊνών χρησιμοποιείται πολύ συχνά στην επιστήμη της βιοπληροφορικής. Το αποτέλεσμα αυτής της μεθόδου είναι η παράταξη των καταλοίπων για κάθε αμινοξύ σε σύγκριση με τα 20 L-a-αμινοξέα (Πίνακας 1.1) που απαρτίζουν τις πρωτεΐνες. Δηλαδή κάθε αμινοξύ συγκρίνεται με κάθε ένα από τα L-a-αμινοξέα και παίρνουμε το ποσοστό ομοιότητας τους. Ως αποτέλεσμα έχουμε μια λίστα με 20 στήλες όπου στην κάθε μία αναγράφεται το ποσοστό ομοιότητας του κάθε αμινοξέος με τα L-a-αμινοξέα, το άθροισμα των ποσοστών αυτών είναι 100 (Σχήμα 2.2). Η ιδέα πίσω από αυτή την κωδικοποίηση ήταν ότι οι πρωτεΐνες σχετίζονται θα έχουν και την ίδια δευτεροταγή δομή. [23]



Σχήμα 2.2: MSA κωδικοποίηση πρωτεΐνης 1cdlg_796-815

2.2.1 Προεπεξεργασία δεδομένων εισόδου

Κάθε πρωτεΐνη στα αρχεία εκπαίδευσης και ελέγχου του δικτύου αποτελείται από τρεις γραμμές (Σχήμα 2.3). Η πρώτη γραμμή είναι το όνομα τις πρωτεΐνης, η δεύτερη είναι η πρωτοταγής δομή και η τρίτη είναι η δευτεροταγής δομή.



Σχήμα 2.3: Αναπαράσταση μίας πρωτεΐνης στο αρχείο εκπαίδευσης και ελέγχου.

Τα αρχεία εκπαίδευσης και ελέγχου δίνονται στο αρχείο storeProteins.py (Παράρτημα Γ) για να αντιστοιχήσει το όνομα της πρωτεΐνης με την MSA κωδικοποίηση της, η οποία είναι και η είσοδος του δικτύου. Συγκεκριμένα για κάθε πρωτεΐνη διαβάζει το όνομά της και βάσει αυτού ψάχνει στο φάκελο msaFilesCorrect για την MSA

κωδικοποίηση της και την αποθηκεύει στον πίνακα data. Έπειτα παίρνει την δευτεροταγή δομή της πρωτεΐνης, δηλαδή την επιθυμητή έξοδο του δικτύου, και την αποθηκεύει στον πίνακα Yt. Όμως οι πιθανοί τύποι δευτεροταγούς πρώτα πρέπει να κωδικοποιηθούν ώστε να ταυτίζονται με τους νευρώνες εξόδου του δικτύου, η κωδικοποίηση φαίνεται στον Πίνακα 2.1.

Δευτεροταγής δομή	Κωδικοποίηση δομής	Αριθμός νευρώνα που ενεργοποιείται
α-έλικας (H)	[1,0,0]	1
β-πτυχωτή επιφάνεια (E)	[0,1,0]	2
Τυχαία αναδίπλωση (C)	[0,0,1]	3

Πίνακας 2.1: Κωδικοποίηση δευτεροταγής δομής

2.2.2 CB513

Οι Rost και Sander εργάστηκαν πάνω στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών [9,10] και για την εκπαίδευση και τον έλεγχο των δικτύων τους χρησιμοποίησαν ένα αρχείο (RS126) με συνολικά 126 πρωτεΐνες.[24] Οι πρωτεΐνες που επιλέχτηκαν να μπουν στο αρχείο έπρεπε να διαφέρουν ανά ζεύγη τουλάχιστον κατά 25% ανά 80 κατάλοιπα αμινοξέων, αυτή η μέθοδος ονομάζεται ποσοστιαία ταυτότητα (percentage identity). Όμως αυτή η τεχνική αποδείχτηκε ότι δεν ήταν αρκετά εξελιγμένη ώστε να βρίσκει όλες τις συσχετιζόμενες πρωτεΐνες. Ακόμη υπήρχαν περιπτώσεις όπου άσχετες μεταξύ τους πρωτεΐνες, οι οποίες μοιράζονταν παρόμοια σύνθεση αμινοξέων παρουσίαζαν υψηλή ποσοστιαία ταυτότητα ενώ στην πραγματικότητα δεν συσχετίζονται. Αυτό συνέβαινε επειδή αυτή η μέθοδος λαμβάνει υπόψη της μόνο το μήκος και την σύνθεση των αμινοξέων.

Στην συνέχεια όμως προτάθηκαν νέες μέθοδοι, οι οποίες αντιμετώπισαν τα ελαττώματα της ποσοστιαίας ταυτότητας. Μία εκ των οποίων είναι και η τυπική απόκλιση (standard deviation). Σε αυτή την μέθοδο οι δύο πρωτεΐνες, που θα βρεθεί η ομοιότητα τους, στοιχίζονται βάσει ενός τυπικού δυναμικού αλγορίθμου. Έπειτα η ακολουθία των αμινοξέων κάθε πρωτεΐνης αλλάζει με τυχαίο τρόπο και οι ακολουθίες

ξαναστοιχίζονται βάσει του δυναμικού αλγόριθμου. Αυτή η διαδικασία επαναλαμβάνεται αρκετές φορές και παίρνουμε τον μέσο όρο τους καθώς και την τυπική απόκλιση των συγκρίσεων των δύο ακολουθιών.

Πολλοί χρησιμοποίησαν το RS126 στους αλγόριθμούς τους, όμως υπήρχε η ανάγκη για την ανάπτυξη αρχείων με περισσότερες ανεξάρτητες πρωτεΐνες. Για αυτό επέλεξαν πρωτεϊνικές ακολουθίες από μία βάση δεδομένων με την χρήση ενός ποιο ευαίσθητου αλγορίθμου ως προς την ομοιότητα των ακολουθιών και αναλύοντας τις αμινο-ομάδες. Από αυτή την διαδικασία προέκυψε ένα σύνολο ακολουθιών, από το οποίο στην συνέχεια αφαιρέθηκαν οι πολυεπίπεδες πρωτεΐνες αφήνοντας 988 από τις 1233. Στην συνέχεια κράτησαν μόνο τις ακολουθίες των οποίων είχαν αναγνωρίσει την δευτεροταγή τους δομή με την μέθοδο της κρυσταλλογραφίας με ανάλυση Angstrom ≤ 2.5. αυτό είχε ως αποτέλεσμα την δημιουργία του CB554 με τις 554 πρωτεΐνες που έμειναν μετά το φιλτράρισμα του προηγούμενου αρχείου. Έπειτα στην προσπάθεια τους να εξασφαλίσουν ότι το CB554 δεν είχε καθόλου ομοιότητες με το RS126 ένωσαν τα αρχεία και σύγκριναν όλα τα ζεύγη των πρωτεϊνών με μία άλλη μέθοδο. Αυτή η μέθοδος ήταν πιο αυστηρή για αυτό βρέθηκαν περισσότερες πανομοιότυπες πρωτεΐνες, οι οποίες και αφαιρέθηκαν δημιουργώντας το αρχείο CB396. Στα δύο αρχεία (CB396 και RS126) υπήρξαν 11 πανομοιότυπα ζεύγη πρωτεϊνικών ακολουθιών, για αυτό τα σύγκριναν και αφαίρεσαν τα 9 ζεύγη που είχαν τυπική απόκλιση ≥ 5. Το αποτέλεσμα του συνδυασμού των αρχείων CB396 και RS126 με εξαίρεση τις 9 ακολουθίες ήταν το αρχείο CB513.

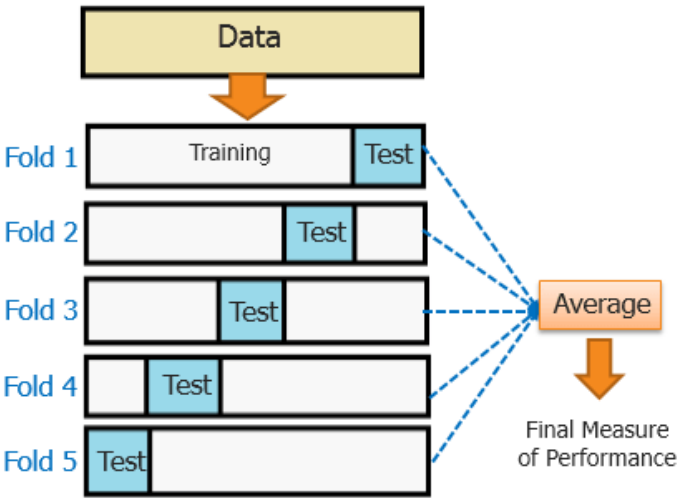
Το σύνολο δεδομένων CB513 είναι κατάλληλο για την χρήση της διασταυρωμένης επικύρωσης (cross validation) για μεθόδους πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Αυτό είναι και το αρχείο δεδομένων που χρησιμοποιήθηκε σε αυτή την προσέγγιση του προβλήματος. [25]

2.2.3 Cross validation

Το cross validation είναι ένας εναλλακτικός τρόπος να εισάγεις τα δεδομένα εκπαίδευσης και ελέγχου στο δίκτυο. Ο πιο συνήθης τρόπος είναι τα δεδομένα εκπαίδευσης να είναι ξένα ως προς τα δεδομένα ελέγου. Στην περίπτωση του cross

validation όμως όλα τα δεδομένα βρίσκονται σε ένα αρχείο το οποίο διαχωρίζεται σε k σχεδόν ίσου μεγέθους κομμάτια. Σε κάθε γύρο τα k -1 κομμάτια χρησιμοποιούνται για εκπαίδευση του δικτύου και το υπολειπόμενο μέρος για τον έλεγχο, αλλάζοντας κάθε φορά το σύνολο των δεδομένων ελέγχου. Δηλαδή ο αλγόριθμος εκτελείται k φορές , όσες και τα κομμάτια που χωρίσαμε τα δεδομένα μας, ώστε όλα τα κομμάτια των δεδομένων να περάσουν από την διαδικασία ελέγχου του δικτύου. Έπειτα για να υπολογίσουμε το τελικό σφάλμα του δικτύου παίρνουμε τον μέσο όρο των σφαλμάτων όλων των εκτελέσεων.

Αυτή η μέθοδος έχει το πλεονέκτημα ότι όλα τα δεδομένα θα περάσουν από την διαδικασία ελέγχου για αυτό δεν πολύ σημαντικός ο τρόπος που θα διαχωριστούν τα δεδομένα. Άρα μειώνονται και οι πιθανότητες να πέσουν περίεργα δεδομένα στο αρχείο ελέγχου με αποτέλεσμα να μην έχουμε αντικειμενική άποψη για τις ικανότητες του δικτύου μας. Ωστόσο το αρνητικό αυτής της μεθόδου είναι ότι χρειάζεται περίπου k φορές περισσότερο χρόνο να ολοκληρώσει, αφού τρέχει ξανά από την αρχή τον αλγόριθμο για κάθε νέο σύνολο δεδομένων ελέγχου. [26]



Σχήμα 2.4: Cross validation για k=5 (http://tomaszkacmajor.pl/index.php/2016/05/01/svm-model-selection2/)

2.3 Υλοποίηση ESN

Πριν ξεκινήσουμε την μελέτη της υλοποίησης του ESN που έγινε από την Ειρήνη Παπακώστα θα εξηγήσουμε τις παραμέτρους (Πίνακας 2.2) που χρησιμοποιούνται για την αρχικοποίηση του δικτύου.

Παράμετρος	Επεξήγηση
maxIterations	Αριθμός επαναλήψεων εκπαίδευσης του δικτύου
initLen	Initial Transient. Αριθμός των δεδομένων που θα αγνοηθούν στην αρχή κάθε επανάληψης
inSize	Αριθμός νευρώνων εισόδου του δικτύου
outSize	Αριθμός νευρώνων εξόδου του δικτύου
resSize	Αριθμός νευρώνων του DR
a	Leaking rate. Ταχύτητα με την οποία ανανεώνονται οι τιμές των νευρώνων του DR
window	Μέγεθος κινητού παραθύρου
sparsity	Πυκνότητα νευρώνων του DR
spectral_radius	Φασματική ακτίνα
forgettingFactor	Forgetting factor. Σημαντικότητα της ιστορίας των σφαλμάτων του δικτύου.
regression	Μεταβλητή που βοηθά στην αποφυγή της υπερεκπαίδευσης του δικτύου
num_folds	Number of folds. Σε πόσα κομμάτια να χωριστούν τα δεδομένα εισόδου για το cross validation.

Πίνακας 2.2: Επεξήγηση παραμέτρων αρχικοποίησης του δικτύου.

Αφού αρχικοποιηθούν οι παράμετροι δημιουργείται το ESN, το οποίο όπως έχουμε αναφέρει έχει 20 νευρώνες στο επίπεδο εισόδου, το DR 500 νευρώνες και 3 στο επίπεδο εξόδου, αφού ως είσοδο του δικτύου χρησιμοποιούμε την MSA κωδικοποίηση των αμινοξέων και κωδικοποιήσαμε τις δευτεροταγείς δομές σε τρεις κατηγορίες (H, E, C). Έπειτα αρχικοποιούνται τα βάρη του επιπέδου εισόδου και του DR με τιμές [-1,1]. Τα βάρη του DR διαιρούνται και με την μεταβλητή regression. Αυτά τα βάρη δεν θα

αλλάξουν τιμή καθ' όλη την διάρκεια εκτέλεσης του αλγορίθμου. Τέλος αρχικοποιούνται οι τιμές των νευρώνων του DR και οι υπόλοιπες μεταβλητές του αλγορίθμου όπως το σφάλμα του δικτύου και το ποσοστό επιτυχίας. Αφού γίνει η αρχικοποίηση ξεκινά η διαδικασία του cross validation και η εκτέλεση του αλγορίθμου. Δηλαδή χωρίζονται τα δεδομένα του αρχείου εισόδου σε όσα κομμάτια όσα και η τιμή της μεταβλητής num_folds. Από αυτά το 1 κομμάτι θα χρησιμοποιηθεί ως δεδομένα ελέγχου σε αυτόν το γύρο και τα υπόλοιπα ως δεδομένα εκπαίδευσης. Όπως έχουμε αναφέρει και στο υποκεφάλαιο 2.2.1 Προεπεξεργασία δεδομένων εισόδου θα γίνει κατάλληλη επεξεργασία των δεδομένων πριν δοθούν στο δίκτυο. Μετά ξεκινά η διαδικασία εκπαίδευσης και ελέγχου του δικτύου, η οποία επαναλαμβάνεται τόσες εποχές όσες και το maxIterations. Επίσης ολόκληρος ο αλγόριθμος επαναλαμβάνεται num_folds φορές αλλάζοντας κάθε φορά το σύνολο δεδομένων ελέγχου. Σημαντικό είναι να σημειωθεί ότι στην αρχή κάθε νέας εκτέλεσης με νέο σύνολο δεδομένων ελέγχου γίνεται ανανέωση των βαρών εξόδου σύμφωνα με την Εξίσωση 2.5, την οποία χρησιμοποιεί και ο Jaeger.

Στην συνέχεια ξεκινά η εκπαίδευση. Στο δίκτυο χρησιμοποιείται η τεχνική του κινητού παραθύρου, δηλαδή επεξεργάζεται ένα αριθμό από αμινοξέα για τα οποία αναβαθμίζει τους νευρώνες (Εξισώσεις 2.2 και 2.3) και υπολογίζει την έξοδο του δικτύου (Εξίσωση 2.4) αλλά μόνο για το τελευταίο αμινοξύ του παραθύρου θα ανανεώσει τα βάρη εξόδου (Εξισώσεις 2.7 – 2.10) και θα υπολογίσει το σφάλμα του δικτύου (Εξίσωση 2.1). Επίσης χρησιμοποιείται και το initial transient, αυτό σημαίνει ότι στις αρχικές επαναλήψεις αγνοούνται οι αλλαγές στους νευρώνες του DR και δεν αλλάζουν τιμές τα βάρη εξόδου. Αυτό συμβαίνει επειδή οι τιμές των νευρώνων του DR αρχικοποιούνται με 0 το οποίο είναι μία αφύσικη κατάσταση για το δίκτυο. Άρα το δίκτυο παίρνει ακολουθίες αμινοξέων όσο και το μέγεθος το κινητού παραθύρου και αλλάζει τις τιμές των νευρώνων του DR. Αν ο αριθμός των επαναλήψεων που έχουν γίνει μέχρι στιγμής έχουν ξεπεράσει την τιμή του initLen (initial transient) το δίκτυο ξεκινά να υπολογίζει την έξοδό του δικτύου και μόνο για το τελευταίο αμινοξύ του παραθύρου αλλάζει και τα βάρη εξόδου του δικτύου. Όταν το δίκτυο δει όλα τα δεδομένα εκπαίδευσης υπολογίζει το συνολικό ποσοστό επιτυχίας και συνεχίζει με την διαδικασία ελέγχου του δικτύου. Στον έλεγχο του δικτύου ακολουθείται η ίδια διαδικασία με αυτή της

εκπαίδευσης με την διαφορά ότι δεν αλλάζουν τα βάρη εξόδου του δικτύου και δεν υπάρχει το initial transient.

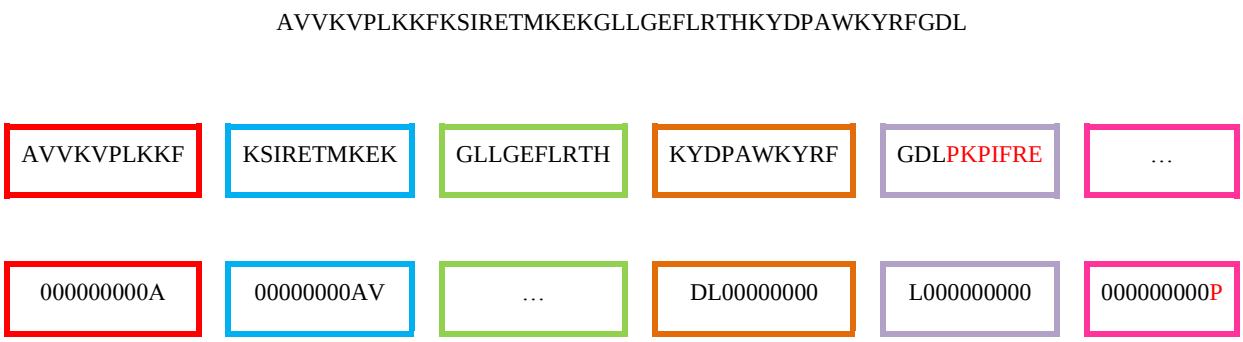
2.4 Διορθώσεις και αλλαγές

Αφού μελέτησα την προηγούμενη υλοποίηση του ESN παρατήρησα ότι υπήρχαν κάποια λάθη τα οποία έπρεπε να διορθωθούν για να είναι πιο αντιπροσωπευτικά τα αποτελέσματα του δικτύου.

Για παράδειγμα παρατήρησα ότι κατά τον έλεγχο χρησιμοποιούνται άλλοι νευρώνες για το DR από ότι στην εκπαίδευση. Για αυτό το άλλαξα ώστε να χρησιμοποιείται το ίδιο DR, γιατί με την προηγούμενη υλοποίηση το δίκτυο δεν χρησιμοποιούσε την γνώση που απόκτησε κατά την εκπαίδευση του αλλά αντίθετα ξαναπροσαρμόζονταν οι τιμές του DR από την αρχή ενώ παράλληλα ελέγχαμε τις γνώσεις του. Ακόμη παρατήρησα ότι στον επαναληπτικό βρόγχο που χρησιμοποιεί για να διατρέξει τα αμινοξέα εκπαίδευσης και ελέγχου διαιρούσε τον πλήθος των αμινοξέων του αρχείου με τον πλήθος των νευρώνων εισόδου. Άρα η εκπαίδευση και έλεγχος του δικτύου γινόταν με το 1/20 των δεδομένων του αρχείου και όχι ολόκληρο.

Επίσης υπήρχε πρόβλημα με τον τρόπο υλοποίησης του κινητού παραθύρου. Το παράθυρο κάθε φορά που άλλαζε θέση μετακινιόταν 10 θέσεις αντί μία, με αποτέλεσμα οι αλλαγές στα βάρη του δικτύου να γίνονται μόνο για κάθε 10^{o} αμινοξύ κάθε ακολουθίας. Άρα το δίκτυο δεν λάμβανε υπόψη τις σχέσεις μεταξύ όλων αμινοξέων της πρωτεϊνικής ακολουθίας άρα δεν έκανε τις αντίστοιχες αλλαγές στα βάρη. Επίσης φτάνοντας στο τέλος μίας πρωτεΐνης επειδή κατά την επεξεργασία τους μπήκαν όλες σε ένα πίνακα η μία μετά την άλλη υπήρχε περίπτωση να πάρει αμινοξέα δύο πρωτεϊνών και να τα δώσει ως είσοδο στο δίκτυο. Δηλαδή αν τύγχανε το μήκος μίας πρωτεΐνης να μην είναι πολλαπλάσιο του μεγέθους του παραθύρου θα έπαιρνε αμινοξέα και από την επόμενη πρωτεΐνη, αφού είναι συνεχόμενες. Το οποίο ήταν λάθος γιατί τα αμινοξέα των δύο πρωτεϊνών δεν συνδέονται μεταξύ τους, άρα το δίκτυο μαθαίνει λάθος πληροφορίες. Ένα τέτοιο παράδειγμα φαίνεται στο Σχήμα 2.5 στην δεύτερη γραμμή, όπου τα κόκκινα γράμματα υποδηλώνονται τα αμινοξέα της επόμενης πρωτεΐνης. Η λύση αυτού του προβλήματος ήταν να προστεθούν μηδενικά, όσα και το μέγεθος του

κινητού παραθύρου, πριν και μετά από μία πρωτεΐνη ώστε η πρόβλεψη των αμινοξέων να ξεκινά από το πρώτο αμινοξύ. Άρα αφού έγινε αυτή αλλαγή στα δεδομένα εισόδου του δικτύου έπρεπε να ενημερωθεί και ο πίνακας με τις επιθυμητές εξόδους (πίνακας Yt στην κλάση storeProteins.py). Δηλαδή για κάθε 0 που προσθέτουμε στην ακολουθία των αμινοξέων προσθέτουμε στον πίνακα Yt την τιμή [0,0,0] (Σχήμα 2.6). Επίσης το παράθυρο δεν θα μετακινείται 10 θέσεις σε κάθε επανάληψη αλλά μόνο μία για να μελετηθούν όλα τα αμινοξέα από το δίκτυο και να προβλεφθεί η δευτεροταγής τους δομή. Στο Σχήμα 2.5 στην 3η γραμμή φαίνεται ο σωστός τρόπος λειτουργίας του παραθύρου για την ίδια πρωτεΐνη με πριν.



Σχήμα 2.5: Υλοποίηση κινητού παραθύρου. Στην 1η γραμμή φαίνεται η πρωτοταγής δομή της πρωτεΐνης που θα προβλέψουμε την δευτεροταγή δομή, στην 2η γραμμή φαίνεται το πως υλοποιήθηκε και στην 3η πως θα έπρεπε να υλοποιηθεί.

Δευτεροταγής δομή της πρωτεΐνης πριν την επεξεργασία
CCEEEEEEECCCHHHHHHHCCCHHHHHCCCCCCHHHHHCCCCC

Δευτεροταγής δομή της πρωτεΐνης μετά την επεξεργασία
[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],[1,0,0],[1,0,0] … [1,0,0],[1,0,0],[0,0,0],[0,0,0],[0,0,0].[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0],[0,0,0]

Σχήμα 2.6: Δευτεροταγή δομή πρωτεΐνης 1htrP_1-43 πριν και μετά την επεξεργασία της από την κλάση storeProteins.py. Με κόκκινο χρώμα φαίνονται τα μηδενικά που πρέπει να προστεθουν, αν το μέγεθος του παραθύρου είναι 10.

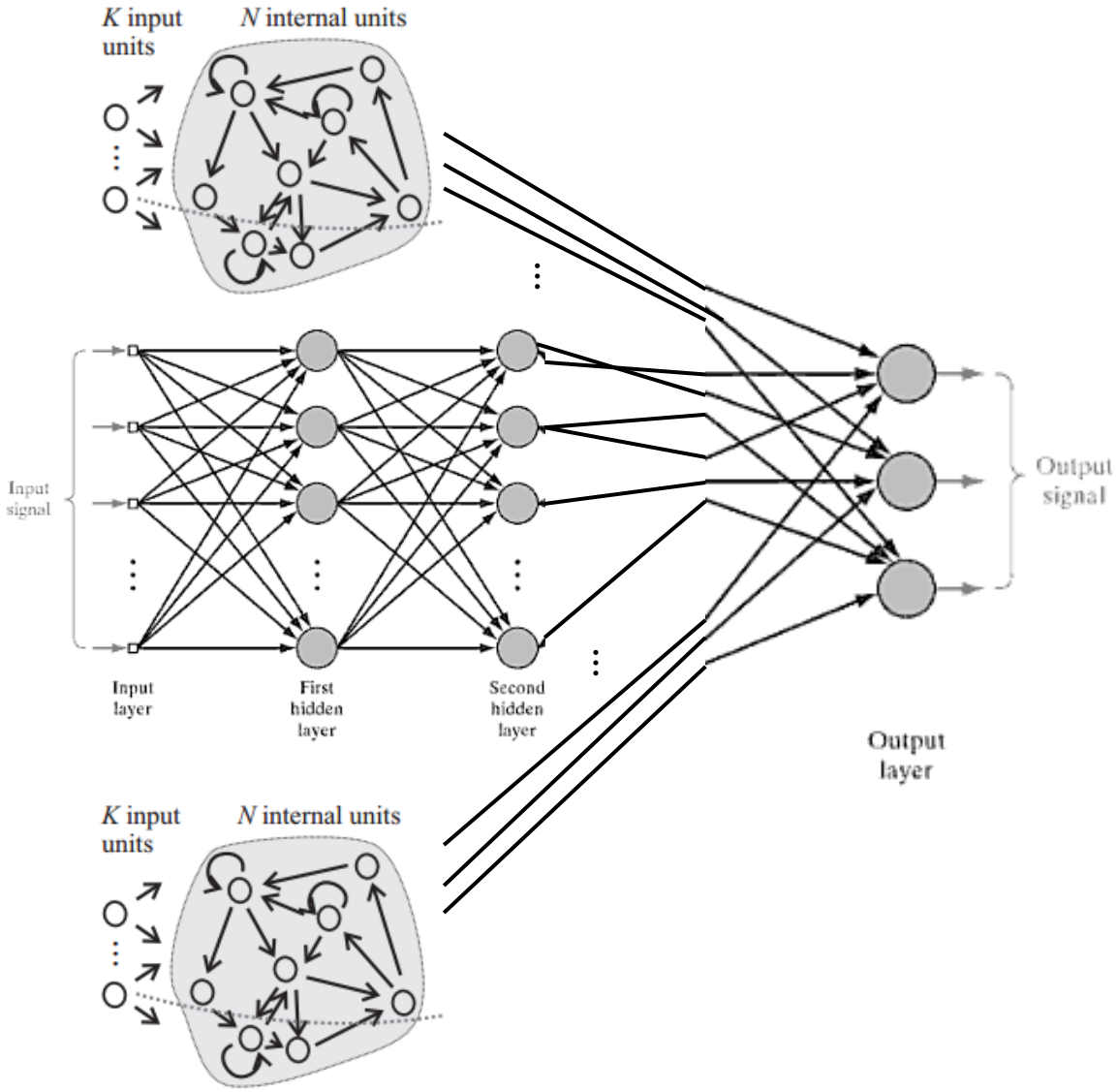
Τέλος αφαιρέθηκε η ενημέρωση των βαρών που γινόταν στην αρχή πριν από κάθε εκτέλεση του αλγορίθμου χρησιμοποιώντας την Εξίσωση 2.5, έτσι ώστε να μην αναιρεί τις αλλαγές που έγιναν κατά την εκπαίδευση του δικτύου με τις εξισώσεις του online αλγόριθμου (Εξισώσεις 2.7 – 2.10).

Κεφάλαιο 3

Υλοποίηση

3.1 Νέο νευρωνικό δίκτυο

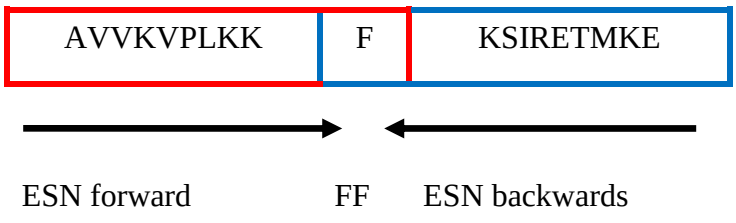
Αφού μελετήθηκε η προηγούμενη εργασία και έγιναν οι διορθώσεις που αναφέρθηκαν στο προηγούμενο κεφάλαιο το δίκτυο επεκτάθηκε προσπαθώντας να αυξήσουμε το ποσοστό επιτυχίας του δικτύου. Δηλαδή δημιουργήθηκε ένα νέο δίκτυο το οποίο αποτελείται από δύο ESNs και ένα δίκτυο εμπρόσθιου περάσματος με κοινό επίπεδο εξόδου (Σχήμα 3.1). Αφήνοντας όμως την κωδικοποίηση των δεδομένων εισόδου (MSA) και των εξόδων την ίδια με πριν.



Σχήμα 3.1: Νέο νευρωνικό δίκτυο με δύο ESNs και ένα δίκτυο εμπρόσθιου περάσματος.

Η ιδέα πίσω από αυτή την υλοποίηση ήταν να δημιουργηθεί ένα νέο recurrent δίκτυο, όπως αυτό του Baldi χρησιμοποιώντας όμως την ταχύτητα και τις ιδιότητες του ESN. Το ένα ESN (forward) θα μελετά τις δυνάμεις που ασκούνται στο κεντρικό αμινοξύ από τα αριστερά προς τα δεξιά και το άλλο (ESN backwards) από δεξιά προς αριστερά, έτσι θα λαμβάνονται υπόψη όλες οι δυνάμεις που ασκούνται στο κοινό τους αμινοξύ, οι οποίες του προσδίδουν την δευτεροταγή του δομή. Το δίκτυο εμπρόσθιου περάσματος (FF) μπορεί να αποτελείται από ένα ή δύο κρυφά επίπεδα, ανάλογα με τις παραμέτρους αρχικοποίησης του, και επεξεργάζεται το κεντρικό αμινοξύ (Σχήμα 3.2). Με αυτή την υλοποίηση έχουμε το εξής πλεονέκτημα: τα ESNs μελετούν τις δυνάμεις που ασκούν τα γειτονικά αμινοξέα ενδιαφερόμενο αμινοξύ ενώ το FF μελετά το ενδιαφερόμενο αμινοξύ.

Τα κάθε ESN έχει το δικό του κινητό παράθυρο, δηλαδή παίρνουν ένα ένα τα αμινοξέα και ενημερώνουν τις τιμές των νευρώνων του DR μέχρι το τελευταίο αμινοξύ χωρίς όμως να υπολογίσουν την έξοδο του δικτύου. Μετά το FF δίκτυο παίρνει το κεντρικό αμινοξύ και υπολογίζει τις εξόδους των επιπέδων του μέχρι και το δεύτερο κρυφό επίπεδο (αν δεν υπάρχει, τότε μέχρι το πρώτο). Έπειτα συνδυάζονται οι τιμές που υπολόγισαν τα τρία δίκτυα και υπολογίζεται η έξοδος του δικτύου στο κοινό επίπεδο εξόδου (Εξίσωση 3.1), χρησιμοποιώντας ως συνάρτηση κατωφλίου την σιγμοειδή συνάρτηση (Εξίσωση 1.4). Επειδή στο δίκτυο προστέθηκε το δίκτυο εμπρόσθιου περάσματος η ενημέρωση των βαρών του ESN θα γίνεται όπως και στο δίκτυο εμπρόσθιου περάσματος, για να υπάρχει μία ομοιομορφία στις αλλαγές των βαρών. Για αυτό ως εξίσωση ενημέρωσης των βαρών όλων των δικτύων χρησιμοποιείται η Εξίσωση 3.2.



Σχήμα 3.2: Αναπαράσταση κινητού παραθύρου του νέου νευρωνικού δικτύου, όπου με κόκκινο χρώμα φαίνονται τα στοιχεία που επεξεργάζεται το ESN forward, με μπλε του ESN backwards και το κεντρικό στοιχείο επεξεργάζεται από το δίκτυο εμπρόσθιου περάσματος (FF).

$$y = \text{dot}(W^{out}, x) + \text{dot}(W^{out}\_B, x\_B) + \text{dot}(bp.weight\ bp.layer)$$

Εξίσωση 3.1: Υπολογισμός εξόδου δικτύου, όπου dot (Wout, x) το γινόμενο των βαρών με τις τιμές των νευρώνων του DR του ESN forward, dot (Wout_B, x_B) το γινόμενο των βαρών με τους νευρώνες του DR του ESN backward, dot (bp.weight bp.layer) το γινόμενο των βαρών με τους νευρώνες του τελευταίου επιπέδου του FF.

$$newWeight = W^{out} - learningRate * delta * x + momentum * (W^{out} - previousW^{out})$$

Εξίσωση 3.2: Αλλαγή βαρών δικτύου, όπου newWeight η νέα τιμή των βαρών, W^{out} τα βάρη του επιπέδου εξόδου, learningRate ρυθμός μάθησης, delta η τιμή δέλτα, x οι τιμές των νευρώνων του DR, momentum η ορμή, $previousW^{out}$ η προηγούμενη τιμή των βαρών.

Το δέλτα για τα ESNs υπολογίζεται με την Εξίσωση 3.3, ενώ για το δίκτυο εμπρόσθιου περάσματος χρησιμοποιούνται οι Εξισώσεις 1.6 και 1.7 ανάλογα με το κρυφό επίπεδο στο οποίο βρισκόμαστε.

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

Εξίσωση 3.3: Υπολογισμός του δέλτα των νευρώνων εξόδου, όπου tpj η επιθυμητή είσοδος και opj η πραγματική έξοδος του νευρώνα.

Στην δική μου υλοποίηση επέλεξα να τροποποιήσω την εξίσωση για την αλλαγή των βαρών εφαρμόζοντας τον επιπλέον όρο της ορμής (momentum), όπως φαίνεται στην Εξίσωση 3.2. Η ορμή κάνει ακόμη πιο έντονες τις αλλαγές στις τιμές των βαρών, δηλαδή αν η αλλαγή που θα γίνει στο βάρος είναι μεγάλη θα γίνει ακόμη μεγαλύτερη αντίθετα αν η αλλαγή είναι μικρή θα γίνει μικρότερη. Με αυτό τον τρόπο τα βάρη θα πλησιάζουν γρηγορότερα την επιθυμητή τιμή, η οποία θα μειώσει το σφάλμα του δικτύου Με αυτό τον τρόπο μειώνονται και οι πιθανότητες το δίκτυο να μείνει σε ένα τοπικό ελάχιστο αντί στο ολικό.

Τέλος, επειδή θα άλλαζε το δίκτυο αποφάσισα να μην χρησιμοποιήσω τις τεχνικές του cross validation και initial transient, ώστε να δω τις πραγματικές ικανότητες του δικτύου στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών.

3.2 Παράμετροι και αλγόριθμος

Αφού άλλαξε το δίκτυο άλλαξαν και κάποιες από τις παραμέτρους του.

Παράμετροι ESNs	Επεξήγηση
maxIterations	Αριθμός επαναλήψεων εκπαίδευσης του δικτύου
initLen	Initial Transient. Αριθμός των δεδομένων που θα αγνοηθούν στην αρχή κάθε επανάληψης
inSize	Αριθμός νευρώνων εισόδου του δικτύου
outsize	Αριθμός νευρώνων εξόδου του δικτύου
resize	Αριθμός νευρώνων του DR
A	Leaking rate. Ταχύτητα με την οποία ανανεώνονται οι τιμές των νευρώνων του DR
Window	Μέγεθος κινητού παραθύρου
Sparsity	Πυκνότητα νευρώνων του DR
spectral_radius	Φασματική ακτίνα
learningRate	Ρυθμός μάθησης
Momentum	Ορμή

Πίνακας 3.1: Παράμετροι ESN νευρωνικών δικτύων.

Παράμετροι FF	Επεξήγηση
hidden1	Αριθμός νευρώνων πρώτου κρυφού επιπέδου
hidden2	Αριθμός νευρώνων δεύτερου κρυφού επιπέδου, αν δεν υπάρχει hidden2=0
num_input	Αριθμός νευρώνων εισόδου του δικτύου
num_output	Αριθμός νευρώνων εξόδου του δικτύου
learningRate	Ρυθμός μάθησης
Momentum	Ορμή

Πίνακας 3.2: Παράμετροι δικτύου εμπρόσθιου περάσματος.

Περιγραφή αλγορίθμου:

1. Δημιουργία δικτύων ESN forward, ESN backwards και εμπρόσθιου περάσματος (FF)
2. Αρχικοποίηση βαρών, νευρώνων και παραμέτρων των δικτύων
3. Επεξεργασία συνόλου δεδομένων εκπαίδευσης και ελέγχου (storeProteins.py)
4. Ενόσω δεν έφτασες στο μέγιστο αριθμό επαναλήψεων
    1. Μηδενισμός σφάλματος και επιτυχίας επανάληψης
    2. Εκπαίδευση δικτύου
    3. Έλεγχος δικτύου
    4. Υπολογισμός σφάλματος και ποσοστού επιτυχίας της εκπαίδευσης και ελέγχου του δικτύου
5. Υπολογισμός συνολικού ποσοστού επιτυχίας του δικτύου

Εκπαίδευση δικτύου:

[1]Ενόσω υπάρχουν ακόμη πρωτεΐνες
1. Αν έφτασες στο τέλος της πρωτεΐνης
    1. Θέσε σφάλμα επανάληψης = σφάλμα επανάληψης + 0.5* σφάλμα πρωτεΐνης
    2. Θέσε σφάλμα πρωτεΐνης = 0
2. Πάρε τα επόμενα x αμινοξέα, όπου x το μέγεθος του κινητού παραθύρου
    1. Για κάθε αμινοξύ (το παράθυρο του ESN forward προχωρά από τα αριστερά προς δεξια και του ESN backwards αντίθετα)
        1. Υπολόγισε τις τιμές των νευρώνων των δύο ESNs (Εξίσωση 2.2)
        2. Αν είσαι στο αμινοξύ x
            1. Υπολόγισε τις εξόδους των κρυφών επιπέδων του FF
            2. Συνδύασε τις εξόδους όλων των δικτύων (Εξίσωση 3.1)
            3. Υπολόγισε την έξοδο του δικτύου (Εξίσωση 1.4)
            4. Ενημέρωσε τις τιμές όλων των βαρών του FF και των W^{out} των ESNs (Εξίσωση 3.2)
            5. Υπολόγισε το σφάλμα της πρόβλεψης (Εξίσωση 2.1)
2. Υπολόγισε το ποσοστό επιτυχίας (Εξίσωση 1.1)

Ο έλεγχος του δικτύου ακολουθεί την ίδια διαδικασία με την διαφορά ότι δεν γίνεται αλλαγή στα βάρη.

Κεφάλαιο 4

Πειράματα, Συμπεράσματα και μελλοντική εργασία

4.1 Πειράματα

Όπως έχουμε δει και αναφέρει, κυρίως στο υποκεφάλαιο 2.1.1 Αρχιτεκτονική ESN, οι σωστές τιμές στις παραμέτρους του δικτύου και ειδικά του DR μπορούν να επηρεάσουν σοβαρά την απόδοση του δικτύου. Οι σωστές τιμές πολλές φορές εξαρτώνται από την φύση του προβλήματος για αυτό και πρέπει να καθοριστούν πειραματικά, αν και υπάρχουν κάποιες κατευθυντήριες γραμμές για το ποιες τιμές των μεταβλητών προσφέρουν κάποια ιδιότητα στο δίκτυο. Ένα τέτοιο παράδειγμα είναι η φασματική ακτίνα η οποία είναι υπεύθυνη για την δημιουργία της echo state ιδιότητας στο ESN.

Για αυτό έγιναν αρκετά πειράματα χρησιμοποιώντας μόνο μία πρωτεΐνη (την ίδια) για την εκπαίδευση και τον έλεγχο του δικτύου. Τα κριτήρια επιλογής της πρωτεΐνης που χρησιμοποιήθηκε για αυτά τα πειράματα ήταν να είναι αρκετά μεγάλη η ακολουθία των αμινοξέων της και στις δευτεροταγείς τους δομές να υπάρχουν και οι τρείς τύποι (H, E, C). Με αυτό τον τρόπο κάνουμε γρήγορα και εύκολα δοκιμές στο δίκτυο και βλέπουμε αν εκπαιδεύεται και βρίσκουμε περίπου σε ποιο πεδίο τιμών θα κυμαίνονται οι κατάλληλες τιμές των μεταβλητών και για μεγαλύτερα αρχεία.

Σημαντικό είναι να αναφερθεί ότι το δίκτυο εμπρόσθιου περάσματος δεν είναι απαραίτητο να δίνεται ως είσοδος μόνο ένα αμινοξύ κάθε φορά. Δηλαδή μπορεί να δοθεί μία ακολουθία από συγκολλημένα γειτονικά αμινοξέα, ώστε το δίκτυο να λάβει υπόψη του περισσότερα δεδομένα. Αυτό καθορίζεται από την τιμή της μεταβλητής overlap. Αν για παράδειγμα το overlap = 0 αυτό σημαίνει ότι το δίκτυο θα παίρνει ως είσοδο μόνο το κεντρικό αμινοξύ των κινητών παραθύρων των δυο ESNs. Για οποιαδήποτε άλλη τιμή του overlap, έστω x, το δίκτυο θα πάρει ως είσοδο την συγκόλληση των x αμινοξέων που προηγούνται του κεντρικού αμινοξέος, το κεντρικό αμινοξύ και τα x αμινοξέα που έπονται του κεντρικού αμινοξέος.

Στον Πίνακα 4.1 φαίνονται τα πειράματα που έγιναν για την μία πρωτεΐνη. Με κόκκινο χρώμα φαίνονται οι μεταβλητές που αλλάζουν τιμή σε σύγκριση με το προηγούμενο πείραμα. Η πράσινη γραμμή δηλώνει τα καλύτερα αποτελέσματα που πέτυχα και οι τιμές των μεταβλητών των πειραμάτων που θα ακολουθήσουν για το αρχείο CB513 (Υποκεφάλαιο 2.2.2) θα είναι παρόμοιες αυτές της πράσινης γραμμής.

Iterations	Reservoir Size	Leaking Rate a	Window Size	Sparsity	Spectral Radius	Overlap	Learning Rate FF	Momentum	Learning Rate ESN	Train Accuracy	Test accuracy
100	200	0.4	10	0.9	1.0	0	0.4	0.7	0.4	70.48	67.86
100	300	0.4	10	0.9	1.0	0	0.4	0.7	0.4	71.31	69.14
100	300	0.4	15	0.9	1.0	0	0.4	0.7	0.4	71.72	68.79
100	300	0.4	10	0.9	1.0	2	0.4	0.7	0.6	86.33	83.03
100	300	0.4	15	0.9	1.0	2	0.4	0.7	0.4	85.58	82.09
100	300	0.4	15	0.9	1.0	2	0.6	0.7	0.4	87.23	83.23
100	300	0.4	15	0.8	1.0	2	0.6	0.7	0.4	86.28	82.42
100	300	0.4	15	0.9	1.0	2	0.6	0.5	0.4	87.01	82.15
100	300	0.4	15	0.9	1.0	2	0.6	0.7	0.6	84.85	81.49
100	300	0.4	15	0.5	1.0	2	0.6	0.7	0.4	86.13	81.56
100	300	0.6	15	0.9	1.0	2	0.6	0.7	0.4	85.89	81.72
150	300	0.4	15	0.9	1.0	2	0.6	0.7	0.4	89.34	85.05
100	300	0.4	15	0.9	1.0	3	0.6	0.7	0.4	88.86	84.71
150	300	0.4	15	0.9	1.0	3	0.6	0.7	0.4	91.89	88.92

Πίνακας 4.1: Τιμές μεταβλητών για τα πειράματα με την μία πρωτεΐνη.

Σε αυτό το σημείο να αναφέρουμε ότι η μικρή διαφορά που υπάρχει μεταξύ του ποσοστού επιτυχίας και ελέγχου που παρατηρούμε στον Πίνακα 4.1 είναι αναμενόμενη και αποδεκτή, άσχετα ότι χρησιμοποιείται μόνο μία πρωτεΐνη για την εκπαίδευση και τον έλεγχο. Αυτή η διαφορά παρατηρείται για τον εξής λόγο: το δίκτυο μελετά ολόκληρη την ακολουθία των αμινοξέων κάνοντας τις απαραίτητες αλλαγές στα βάρη και μετά ελέγχει τις γνώσεις του δικτύου με την ίδια πρωτεΐνη. Άρα επειδή συνεχίζουν να γίνονται αλλαγές στο δίκτυο οι οποίες μπορεί να αναιρούν πρόοδο που έγινε για κάποια αμινοξέα παρατηρούμε αυτή τη διαφορά στις τιμές. Αν όμως παίρναμε ένα αμινοξύ κάναμε τις αλλαγές στο δίκτυο και αμέσως κάναμε τον έλεγχο με το ίδιο αμινοξύ το ποσοστό επιτυχίας του ελέγχου και της εκπαίδευσης θα ήταν τα ίδια.

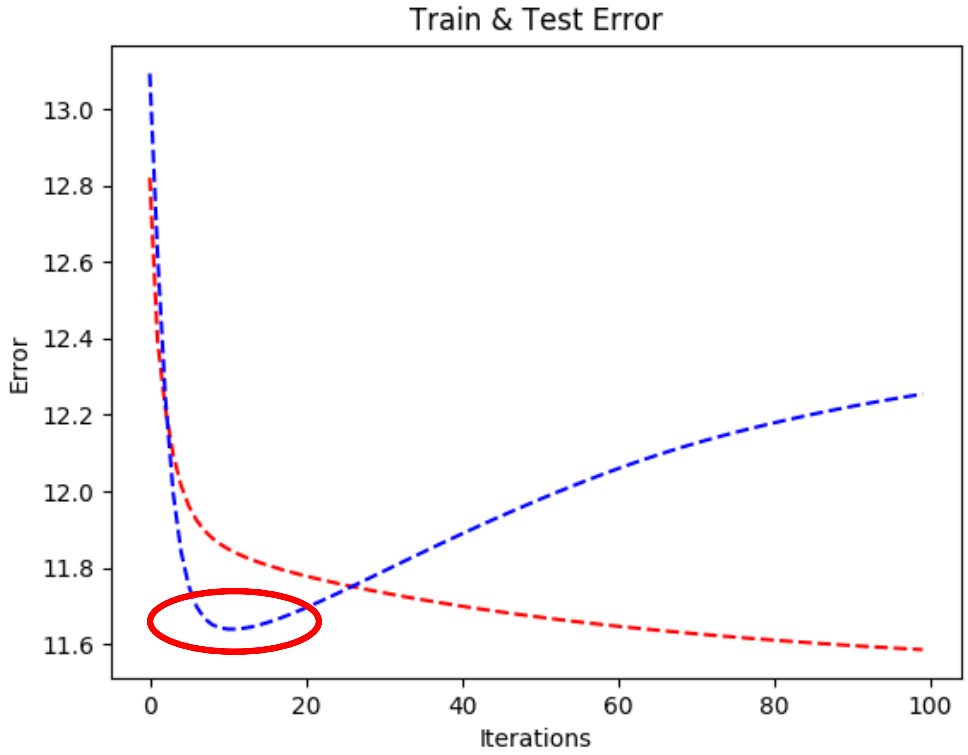
Πείραμα 1

Αρχικά αποφάσισα να χρησιμοποιήσω μόνο τα δύο ESNs για να δω πόσο είναι το ποσοστό επιτυχίας τους, καθώς και αν θα ωφελούσε αργότερα η προσθήκη του δικτύου εμπρόσθιου περάσματος. Η αρχικοποίηση των παραμέτρων του δικτύου έγιναν βάσει

του Πίνακα 4.2 και το ποσοστό επιτυχίας εκπαίδευσης αυτού του πειράματος ήταν 71.65% και του ελέγχου 68.79%, τα οποία για αρχή είναι πολύ ικανοποιητικά.

Παράμετρος	Τιμή
Iterations	100
Reservoir Size	300
Leaking Rate a	0.4
Window Size	15
Sparsity	0.9
Spectral Radius	1.0
Learning Rate ESN	0.4
Momentum	0.7

Πίνακας 4.2: Παράμετροι δικτύου πρώτου πειράματος.



Σχήμα 4.1: Γραφική σφάλματος εκπαίδευσης και ελέγχου του δικτύου για το πρώτο πείραμα. Με κόκκινο χρώμα φαίνεται το σφάλμα εκπαίδευσης και με μπλε του ελέγχου.

Όπως παρατηρούμε και στην γραφική του σφάλματος εκπαίδευσης και ελέγχου το δίκτυο υπερεκπαιδεύεται. Αυτό το καταλαβαίνουμε από το σημείο καμπής (φαίνεται

στον κόκκινο κύκλο) στην γραφική του σφάλματος ελέγχου. Αυτό δηλώνει ότι το δίκτυο μαθαίνει τα δεδομένα εκπαίδευσης και δεν τα γενικεύει ώστε να μπορέσει να προβλέψει τα άγνωστα προς αυτό δεδομένα ελέγχου. Αυτό μπορεί να διορθωθεί με την τεχνική του early stopping, δηλαδή να σταματούμε την εκπαίδευση του δικτύου μόλις αρχίσει να υπερεκπαιδεύεται. Ή να βρεθούν καλύτερες τιμές για τις άλλες παραμέτρους του δικτύου, όπως για παράδειγμα η πυκνότητα και το μέγεθος του δικτύου που είναι υπεύθυνες για την δημιουργία της μνήμης.

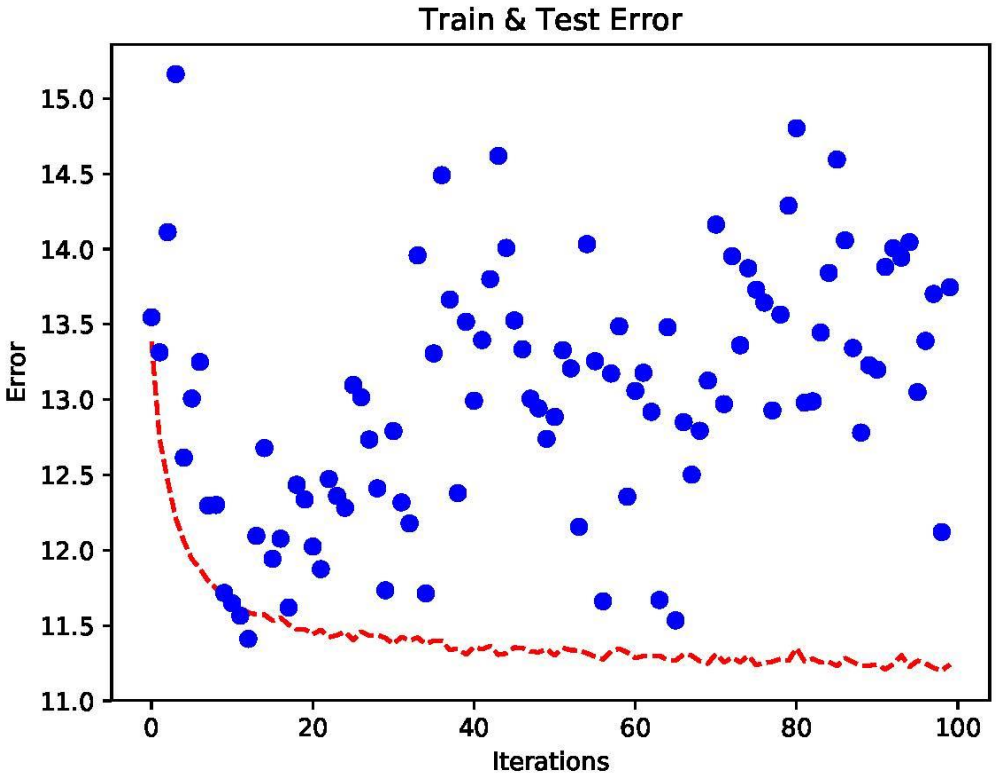
Πείραμα 2

Σε αυτό το πείραμα προστέθηκε και το δίκτυο εμπρόσθιου περάσματος για να δούμε ότι όντως προσφέρει γνώση στο δίκτυό μας. Η αρχικοποίηση των παραμέτρων του έγινε με βάση των Πίνακα 4.3 και το ποσοστό επιτυχίας της εκπαίδευσης ήταν 74.12% και του ελέγχου 69.34%. Να σημειωθεί ότι χρησιμοποιήθηκε μόνο ένα κρυφό επίπεδο.

Παράμετρος	Τιμή
Overlap	3
Learning Rate FF	0.6
Momentum	0.7
Input Size FF	10

Πίνακας 4.3: Παράμετροι δικτύου εμπρόσθιου περάσματος δεύτερου πειράματος.

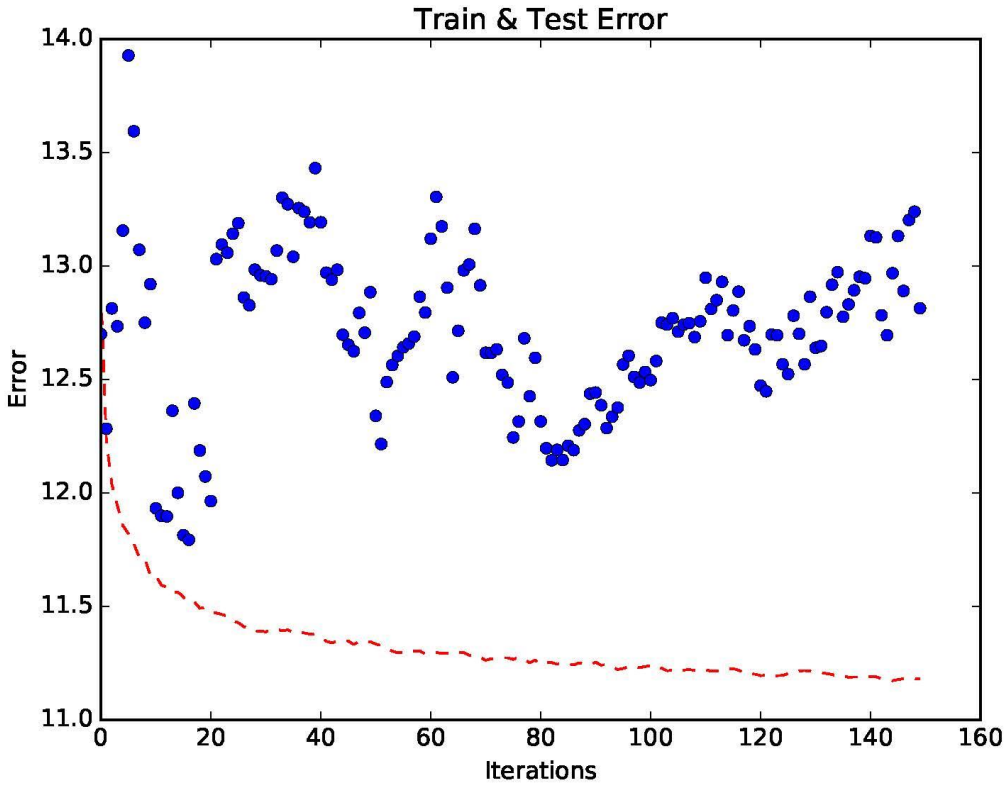
Όπως φαίνεται και από το Σχήμα 4.2 το σφάλμα της εκπαίδευσης σωστά είναι χαμηλότερο από του ελέγχου, αφού όσο προχωρά η διαδικασία το δίκτυο μαθαίνει ακόμη περισσότερο. Το ανησυχητικό είναι ότι υπάρχουν πολλά σκαμπανεβάσματα στο σφάλμα ελέγχου το οποίο μπορεί να οφείλεται στην υπερεκπαίδευση του δικτύου ή στο ότι οι πρωτεΐνες που βρίσκονται στο αρχείο ελέγχου είναι περίεργες και το δίκτυο δεν μπορεί να κάνει καλή γενίκευση. Όμως όντως βλέπουμε ότι η προσθήκη του δικτύου εμπρόσθιου περάσματος αύξησε τα ποσοστά επιτυχίας του δικτύου μας.



Σχήμα 4.2: Γραφική σφάλματος εκπαίδευσης και ελέγχου του δικτύου για το πρώτο πείραμα. Με κόκκινο χρώμα φαίνεται το σφάλμα εκπαίδευσης και με μπλε του ελέγχου.

Πείραμα 3

Σε αυτό το πείραμα η μόνη μεταβλητή που άλλαξε σε σχέση με το προηγούμενο πείραμα είναι ο αριθμός των επαναλήψεων, από 100 έγινε 150. Σκοπός αυτού του πειράματος ήταν να δω πόσο επηρεάζεται η απόδοση του δικτύου από τον αριθμό των επαναλήψεων. Το ποσοστό επιτυχίας εκπαίδευσης για αυτό το πείραμα ήταν 74.89%και του ελέγχου 70.63%. Από ότι παρατηρούμε το ποσοστό επιτυχίας αυξήθηκε και στις δύο περιπτώσεις, όμως η αύξηση τους δεν ήταν πολύ σημαντική. Άρα και να αυξηθεί ακόμη περισσότερο ο αριθμός των επαναλήψεων δεν θα βελτιωθούν πολύ τα ποσοστά επιτυχίας. Μάλιστα μπορεί να έχουμε αντίθετα αποτελέσματα και το δίκτυο να υπερεκπαιδευτεί.

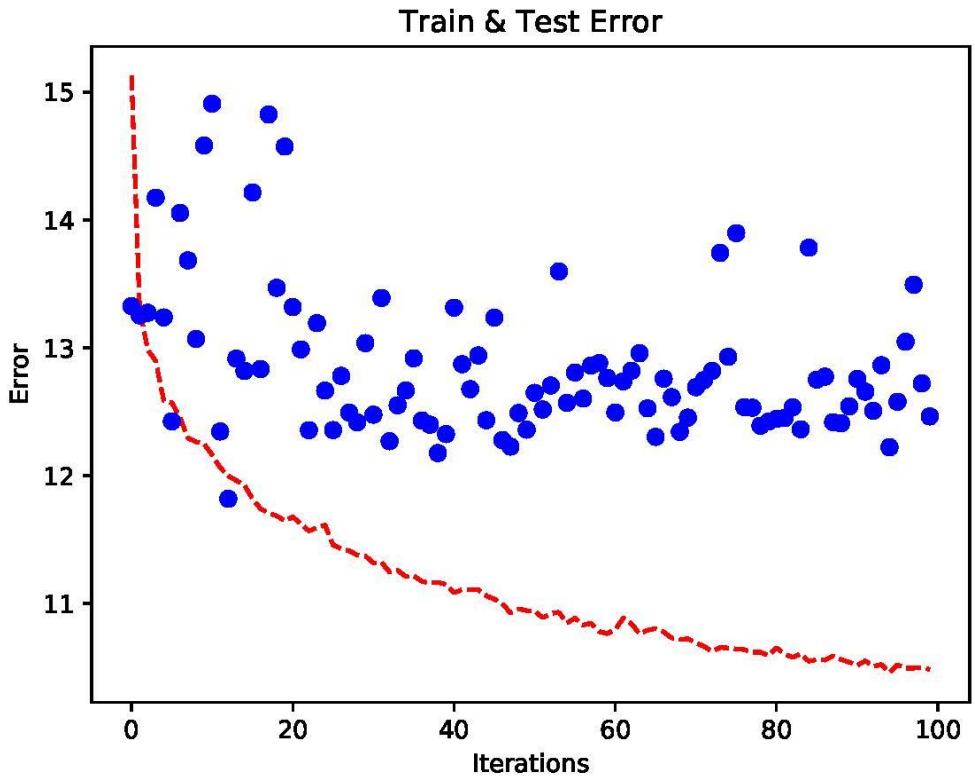


Σχήμα 4.3: Γραφική σφάλματος εκπαίδευσης και ελέγχου του δικτύου για το τρίτο πείραμα.

Από ότι βλέπουμε και σε αυτή την γραφική (Σχήμα 4.3) υπάρχει το πρόβλημα ότι στις τιμές του σφάλματος ελέγχου, εξακολουθούν να υπάρχουν πολλά σκαμπανεβάσματα.

Πείραμα 4

Σε αυτό το πείραμα οι παράμετροι είναι οι ίδιες με του πρώτου με την διαφορά ότι αύξησα τον αριθμό των νευρώνων του πρώτου κρυφού επιπέδου του δικτύου εμπρόσθιου περάσματος, από 10 σε 40. Η σκέψη πίσω από αυτή την αλλαγή ήταν ότι αφού το δίκτυο δέχεται ως είσοδο 7 αμινοξέα να αυξήσω τους νευρώνες του δικτύου, ώστε να έχει την δυνατότητα να αποκτήσει και να συγκρατήσει περισσότερη γνώση. Δηλαδή δίνοντας του περισσότερους νευρώνες έχει την δυνατότητα να θέσει περισσότερες γραμμές κατηγοριοποίησης και να χωρίσει τα δεδομένα που βλέπει σε περισσότερες ομάδες.



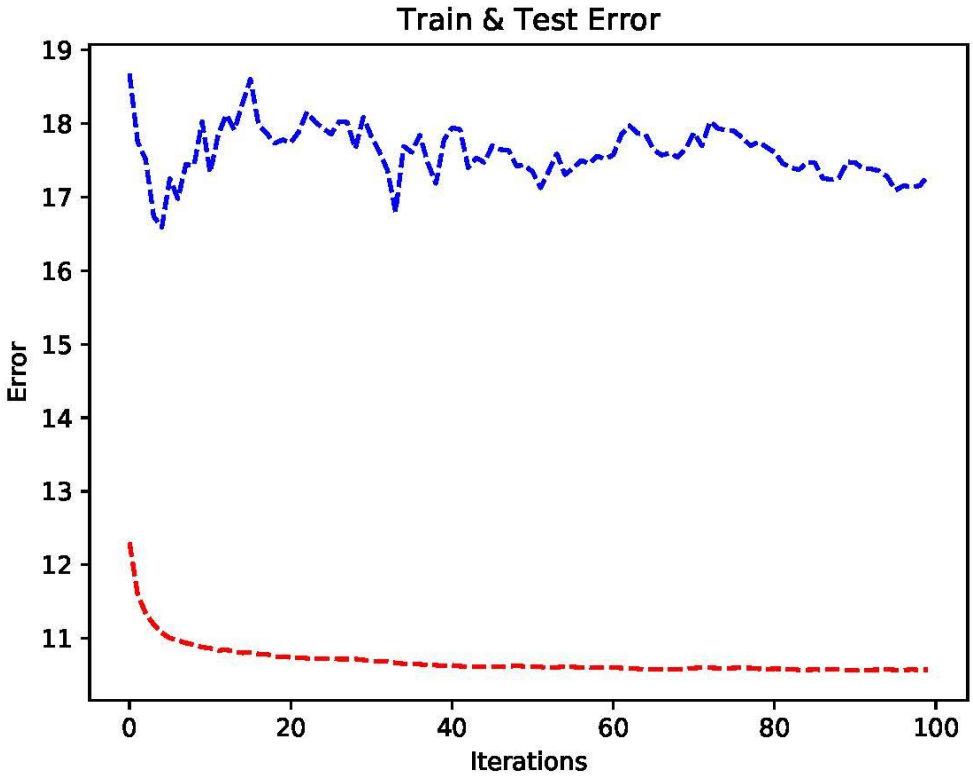
Σχήμα 4.4: Γραφική σφάλματος εκπαίδευσης και ελέγχου του δικτύου για το τέταρτο πείραμα.

Σε αυτό το πείραμα το ποσοστό επιτυχίας εκπαίδευσης ήταν 77.20% και του ελέγχου 71.94%. Παρατηρούμε ότι αυξήθηκε αρκετά το ποσοστό επιτυχίας σε σύγκριση με τα προηγούμενα πειράματα, άρα είναι σημαντικό να καθοριστεί ο κατάλληλος αριθμός των νευρώνων του δικτύου εμπρόσθιου περάσματος ανάλογα με των αριθμό των αμινοξέων που δέχεται ως είσοδο. Στο Σχήμα 4.4 βλέπουμε ότι εξακολουθεί το σφάλμα ελέγχου να έχει μία σταθερή μείωση αλλά συνεχίζει να έχει σκαμπανεβάσματα. Το θετικό είναι ότι δεν είναι τόσο έντονες οι αλλαγές σε αντίθεση με πριν.

Πείραμα 5

Σε αυτό το πείραμα αποφάσισα να ελέγξω πόσο επηρεάζουν τα δεδομένα ελέγχου τα ποσοστά επιτυχίας και το σφάλμα του δικτύου. Για αυτό πήρα ένα μέρος των δεδομένων εκπαίδευσης και τα χρησιμοποίησα ως δεδομένα ελέγχου και τα δεδομένα ελέγχου τα πρόσθεσα στο αρχείο εκπαίδευσης. Με αυτό τον τρόπο θα έβρισκα και αν

όντως τα δεδομένα που χρησιμοποιούσα μέχρι τώρα για τον έλεγχο ήταν περίεργα και ευθύνονταν για τα σκαμπανεβάσματα του σφάλματος ελέγχου. Το ποσοστό επιτυχίας που επιτεύχθηκε σε αυτό το πείραμα για την εκπαίδευση είναι 75.22% και του ελέγχου 70.62%. Δηλαδή υπάρχει μία μικρή βελτίωση περίπου 1% σε σύγκριση με το πρώτο πείραμα.

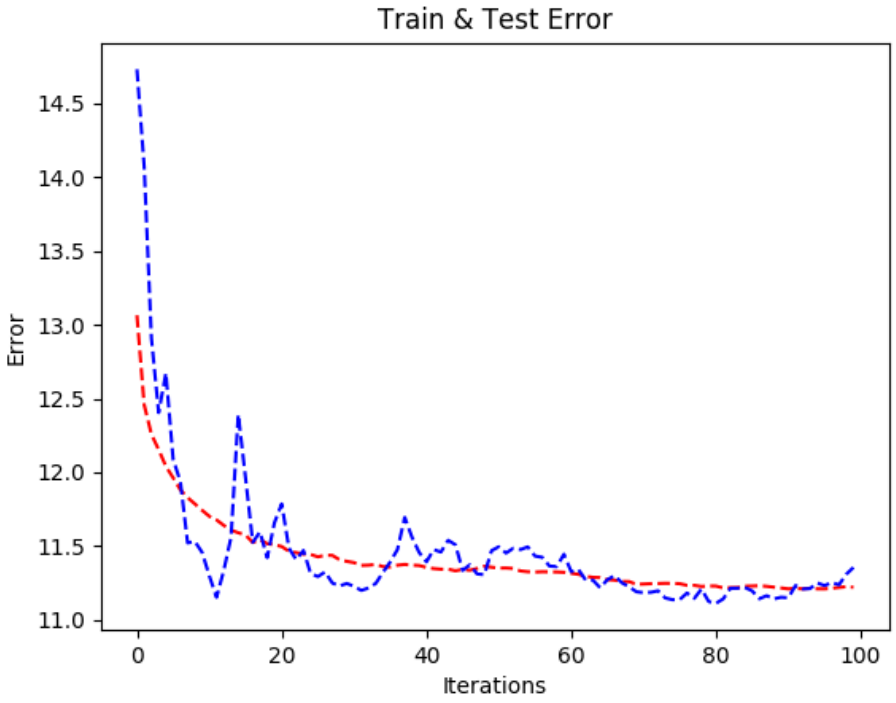


Σχήμα 4.5: Γραφική σφάλματος εκπαίδευσης και ελέγχου του δικτύου για το πέμπτο πείραμα.

Βλέπουμε στο Σχήμα 4.5 ότι δεν είναι τόσο έντονα τα σκαμπανεβάσματα στο σφάλμα εκπαίδευσης όμως έχει αυξηθεί κατά 4%. Άρα ο τρόπος που θα διαχωριστούν τα δεδομένα στα δύο αρχεία (εκπαίδευσης και ελέγχου) είναι σημαντικός. Αυτό συμβαίνει κυρίως λόγω των δεδομένων που βρίσκονται στο CB513, γιατί φρόντισαν οι πρωτεΐνες του αρχείου να μην έχουν μεγάλο ποσοστό ομοιότητας.

Πείραμα 6

Στο τελευταίο πείραμα επέλεξα να αλλάξω την τιμή της φασματικής ακτίνας (spectral radius) από 1.0 σε 0.8. Αυτή είναι μία από τις σημαντικότερες παραμέτρους του δικτύου και είναι σημαντικό να βρεθεί η βέλτιστη τιμή της, αφού ευθύνεται για την επιτυχία της echo state ιδιότητας στο δίκτυο.



Σχήμα 4.6: Γραφική σφάλματος εκπαίδευσης και ελέγχου του δικτύου για το έκτο πείραμα.

Όπως φαίνεται στο Σχήμα 4.6 μειώθηκε αρκετά το σφάλμα ελέγχου και έγινε σχεδόν ίδιο με το σφάλμα εκπαίδευσης το οποίο σημαίνει και αύξηση στα ποσοστά επιτυχίας με αυτό του ελέγχου να φτάνει το 72.93% και της εκπαίδευσης το 74.80%. Επίσης βλέπουμε ότι εξομαλυνθήκαν τα σκαμπανεβάσματα αν και εξακολουθούν να υπάρχουν. Αυτό το πείραμα αποδεικνύει ότι είναι απαραίτητο να καθοριστούν οι ιδανικές τιμές των παραμέτρων γιατί επηρεάζουν την απόδοση του δικτύου άρα και επιτυχία των προβλέψεων.

4.2 Συμπεράσματα και μελλοντική εργασία

Είναι θετικό και αναμενόμενο το γεγονός ότι το νέο νευρωνικό δίκτυο έχει καλύτερες αποδώσεις από το απλό ESN και ότι τα αποτελέσματα του πλησιάζουν αυτά του Baldi. Το ενθαρρυντικό είναι ότι έχει επιτευχθεί ένα ποσοστό περίπου 73% χωρίς να χρησιμοποιηθούν οι μέθοδοι Filtering και Ensembles που χρησιμοποίησε ο Baldi.[12,13]

Αρχικά χρειάζεται να γίνουν και άλλα πειράματα ώστε να δούμε αν μπορεί να αυξηθεί και άλλο το ποσοστό επιτυχίας και κυρίως να δούμε γιατί υπάρχουν τόσα σκαμπανεβάσματα στο σφάλμα ελέγχου. Δηλαδή να μάθουμε αν το δίκτυο όντως υπερεκπαιδεύεται ή αν αυτή του η συμπεριφορά οφείλεται στα δεδομένα. Σε αυτή την περίπτωση θα χρειαστεί να μπει κάποιου είδους «θόρυβος» στα δεδομένα εισόδου ώστε να μην μαθαίνει τα δεδομένα παπαγαλία και να κάνει καλύτερη γενίκευση.

Ακόμη καλό θα ήταν να χρησιμοποιηθεί η μέθοδος του Cross Validation για να περάσουν όλα τα δεδομένα από το σύνολο ελέγχου και να πάρουμε τον μέσο όρο επιτυχίας του δικτύου για ολόκληρο το CB513. Καλό θα ήταν να χρησιμοποιηθεί και η τεχνική τουInitial Transient γιατί το δίκτυο δεν ξεκινά από μία φυσική κατάσταση, άρα καλό είναι να αγνοηθούν οι αρχικές αλλαγές στου νευρώνες του DR ώστε να έρθει το δίκτυο σε μία πιο φυσιολογική κατάσταση, χωρίς να επηρεάσει αρνητικά τις τιμές των βαρών.

Επίσης καλό θα ήταν να χρησιμοποιηθεί η μετρική Segment Overlap (SOV) μαζί με την Q3 (Εξίσωση 1.1) για την βαθμολόγηση της επιτυχίας των προβλέψεων του δικτύου. Αυτή η μέθοδος είναι πιο αυστηρή από την Q3 γιατί συγκρίνει διαστήματα αμινοξέων αντί απλά την προβλεπόμενη έξοδο με την επιθυμητή. Αν για παράδειγμα η δευτεροταγής δομή της πρωτεΐνης αποτελείται από ένα διάστημα α-ελίκων ακολουθούμενο από ένα διάστημα β-πτυχωτών επιφανειών και μετά ακόμη ένα διάστημα α-ελίκων (π.χ.HHHHHEEEEHHHHH) και το δίκτυο πρόβλεψε ένα διάστημα β-πτυχωτών επιφανειών και μία α-ελίκων (EEEEEEHHHHHHHH) το ποσοστό επιτυχίας Q3 θα είναι πολύ διαφορετικό από το SOV. Αυτό συμβαίνει γιατί για τα περισσότερα αμινοξέα πρόβλεψε σωστά την δευτεροταγή τους δομή (ποσοστό Q3)

όμως δεν πρόβλεψε σωστά τα διαστήματα της δευτεροταγούς δομής (ποσοστό SOV). Δηλαδή χρησιμοποιώντας και τους δύο τρόπους αξιολόγησης των προβλέψεων θα ξέρουμε το ποσοστό των αμινοξέων που προβλέπει σωστά το δίκτυο καθώς και το αν προβλέπει σωστά την δομή της πρωτεΐνης. [27] Καλό θα ήταν να φαίνονται και τα ποσοστά επιτυχίας των προβλέψεων για κάθε πρωτεΐνη ξεχωριστά εκτός από το συνολικό ποσοστό επιτυχίας της εκτέλεσης. Έτσι θα γνωρίζουμε για κάθε πρωτεΐνη την γνώση που αποκτά το δίκτυο και πόσο αυτή βελτιώνεται με το πέρασμα του χρόνου.

Όπως έχουμε αναφέρει στην περιγραφή του ESN το DR δημιουργείται τυχαία βάσει των παραμέτρων του δικτύου από το οποίο πήραμε την αρχική ιδέα για τις ικανότητες του στην εφαρμογή αυτού του προβλήματος, όμως όχι τα βέλτιστα αποτελέσματα. Άρα χρειάζεται να υπάρχει μία στρατηγική στον τρόπο δημιουργίας του DR και των βαρών ώστε να επιτευχθεί η echo state ιδιότητα αλλά και να έχουμε καλύτερα ποσοστά πρόβλεψης. Για αυτό καλό θα ήταν μελλοντικά το απλό ESN να αντικατασταθεί με το growing ESN (GESN). Το GESN, όπως φαίνεται και από το όνομα τους, μεγαλώνουν το DR μελετώντας τις γραμμικές αλγεβρικές ιδιότητες του πίνακα των βαρών του. Δηλαδή προσθέτουν μικρότερα reservoir, ένα κάθε φορά, μέχρι να ικανοποιηθεί το κριτήριο που θέσαμε. Έτσι αυξάνεται η πυκνότητα του δικτύου και η επίδοση του χωρίς να χρειάζεται αναπροσαρμογή των βαρών και διατηρείται η echo state ιδιότητα.[28]

Τέλος η εφαρμογή των μεθόδων Filtering και Ensembles, οι οποίες δίνουν στο δίκτυο περισσότερες πληροφορίες σχετικά με τις πρωτεΐνες ίσως αυξήσουν το ποσοστό επιτυχίας του δικτύου. Επίσης αν εφαρμοστούν, θα μπορούμε να κάνουμε πιο αντικειμενική σύγκριση του δικτύου μας με αυτό του Baldi, αφού θα χρησιμοποιηθούν οι ίδιες μέθοδοι για τα δεδομένα εισόδου και τα αποτελέσματα θα βασίζονται στις ικανότητες του δικτύου.

Βιβλιογραφία

Μ. Αγαθοκλέους, "Πρόβλεψη δευτεροταγούς δομής πρωτεϊνών με Νευρωνικά δίκτυα Αμφίδρομης Ανάδρασης", Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου,2009.

Christodoulou C. EPL442 notes., Nicosia 2015 – 2016.

Αναφορές

[1] Ε. Παπακώστα, "Πρόβλεψη της Δευτεροταγούς Δομής των Πρωτεϊνών με την βοήθεια Echo State Network και δημιουργία διαδικτυακής εφαρμογής για Αναπαράσταση του προβλήματος", Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου,2016.

[2] C. Starr and B. McMillan, "Human Biology", Cengage Learning: Boston, MA (2016).

[3] Γ. Π. Γιαλούρης, Κ. Μποσινάκου και Δ. Σίδερης, "Κεφάλαιο 2-Αμινοξέα – Πεπτίδια", Βιοχημεία, Τεχνολογικής Κατεύθυνσης, Γ' Τάξης Γενικού Λυκείου, Ψηφιακό Σχολείο, Οργανισμός εκδόσεως διδακτικών βιβλίων, Αθήνα.

Μεταφορτώθηκε στις 23/12/2016:

“ http://ebooks.edu.gr/modules/ebook/show.php/DSGL-C120/480/3166,12747/ ”

[4] Γ. Π. Γιαλούρης, Κ. Μποσινάκου και Δ. Σίδερης, " Κεφάλαιο 3- Πρωτεΐνες ", Βιοχημεία ,Τεχνολογικής Κατεύθυνσης, Γ' Τάξης Γενικού Λυκείου, Ψηφιακό Σχολείο, Οργανισμός εκδόσεως διδακτικών βιβλίων, Αθήνα.

Μεταφορτώθηκε στις 23/12/2016:

“ http://ebooks.edu.gr/modules/ebook/show.php/DSGL-C120/480/3166,12748/ ”

[5] F. Richards and C. Kundrot "Identification of Structural Motifs from Proteins Coordinate Data: Secondary Structure and First-level Supersecondary Structure", Proteins, vol. 3, Issue 2, (1988): 71-84.

[6] PY Chou and GD Fasman, "Prediction of protein conformation", Biochemistry, vol. 13, Issue 2, pp. 222-245, (1974).

[7] M. Zvelebil and J. Baum, "Understanding Bioinformatics", Garland Science: New York, NY (2008).

[8] N. Qian and TJ. Sejnowski, "Predicting the secondary structure of globular proteins using neural network models", Journal of Molecular Biology, vol. 202, pp. 865-884, (1988).

[9] B. Rost and C. Sander, "Improved prediction of protein secondary structure by use of sequence profiles and neural networks", Proc. Natl. Acad. Sci. USA 90(16), pp. 7558 – 7562, (1993).

[10] B. Rost and C. Sander, "Combining evolutionary information and neural networks to predict protein secondary structure", Proteins 19, pp. 55–72, (1994).

[11] A. Salamov and V. Soloveyev, "Protein secondary structure prediction using local alignments", Journal of Molecular Biology 268, pp. 31–36, (1997).

[12] P. Baldi, S. Brunak, P. Frasconi, G. Soda and G. Pollastri, "Exploiting the Past and the Future in Protein Secondary Structure Prediction", Bioinformatics, vol. 15, Issue 11, pp. 937-946, (1999).

[13] P. Baldi, S. Brunak, P. Frasconi, G. Soda and G. Pollastri, "Bidirectional Dynamics for Protein Secondary Structure Prediction", Sequence Learning, Paradigms, Algorithms and Applications, Lecture Notes in Computer Science, ed. by R. Son and C. L. Giles, Springer: Heidelberg, Vol 1828, pp.80-104, (2001).

[14] S. Wang, J. Peng, J. Ma and J. Xu, "Protein secondary structure prediction using deep convolutional neural fields", Scientific Rep. 6:18962, (2016).

[15] W.S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", Bulletin of Mathematical Biophysics, 5, pp.115-133, (1943).

[16] Elman L.J., "Finding Structure in Time" Cognitive Science, 14, pp.179-211, (1990).

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

[18] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the “echo state network” approach", Technical Report GMD Report 159, German National Research Center for Information Technology, (2002).

[19] H. Jaeger, "The "echo state" approach to analyzing and training recurrent neural networks – with an Erratum note1", Corrected version of the technical report H. Jaeger (2001): The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, German National Research Center for Information Technology, (2010).

[20] M. Lukoševičius, "A Practical Guide to Applying Echo State Networks." In: G. Montavon, G. B. Orr, and K.-R. Müller (eds.) Neural Networks: Tricks of the Trade, 2nd ed. Springer: Heidelberg, Lecture Notes in Computer Science, Vol. 7700, pp.659-686, (2012).

[21] M. Hermans and B. Schrauwen, "Recurrent kernel machines: Computing with infinite echo state networks", pp. 104-133, Neural Computation, Volume 24(1), (2012).

[22] D. Dean, S. Nasuto and K. Warwick, "Recursive Least Squares for Echo State Network Damage Compensation" Proceedings of the 50th Annual Convention of the Society for the Study of Artificial Intelligence and the Simulation of Behaviour, AISB 2014, London, UK (2014).

[23] C. B. Do and K. Katoh, "Protein Multiple Sequence Alignment", In Functional Proteomics: Methods and Protocols, Series: Methods in Molecular Biology, ed. by J. D. Thomson, C. Schaeffer-Reiss, and M. Ueffing, vol. 484, pp. 379-413, Humana Press, Totowa, NJ (2008).

[24] B. Rost and C. Sander, "Prediction of protein secondary structure at better than 70% accuracy.", J. Mol. Biol., 232, pp.584-599,(1993).

[25] J.A. Cuff and G.J. Barton, "Evaluation and Improvement of Multiple Sequence for Protein Secondary Structure Prediction", Proteins, 34, pp.508-519, (1999).

[26] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection", In Proceedings of the 14th International Joint Conference on Artificial Intelligence, San Francisco, CA: Morgan Kaufmann, pp. 1137-1143, (1995).

[27] Γ. Χριστοδούλου, "Διερεύνηση Μεθόδων Εκπαίδευσης Νευρωνικών Δικτύων Αμφίδρομης Ανάδρασης Για Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών", Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, (2010).

[28] J. Qiao, F. Li, H. Han and W. Li, "Growing echo-state network with multiple subreservoirs", IEEE Transactions on Neural Networks and Learning Systems, vol.28, pp. 391-404, (2017).

Παράρτημα Α

Main.py

```
##
# Initialize all parameters and run network  the training and testing data.
#
import ESN



network = ESN.TrainESN(
    maxIterations=150,
    initLen=0,
    inSize=20,
    outSize=3,
    resSize=300,
    a=0.4,
    window=15,
    sparsity=0.9,
    spectral_radius=1.0,
    regression=1e-8,
    learningRate=0.4,
    momentum=0.7
)

network.run("../input/TrainingSets/trainSet4.bin","../input/TestSets/testSet4.bin")
```

Παράρτημα Β

ESN.py

```
##
# Run network.
##

from numpy import *
from matplotlib.pyplot import *
import storeProteins
import BackPropagation



class TrainESN:
    def __init__(self, maxIterations, initLen, inSize, outSize, resSize, a, window, sparsity, spectral_radius,
                 regression, learningRate, momentum):
        self.maxIteration = maxIterations
        self.initLen = initLen
        self.inSize = inSize
        self.outSize = outSize
        self.resSize = resSize
        self.a = a
        self.window = window
        self.sparsity = sparsity
        self.spectral_radius = spectral_radius
        self.regression = regression
        self.outTrain = open("output/OutputTrain.txt", "w")
        self.outTest = open("output/OutputTest.txt", "w")
        self.lr = learningRate
        self.momentum = momentum
```

```
def init_weights(self):
    self.delta = zeros((self.outSize, 1))

    self.W_in = random.uniform(-0.1, 0.1, (self.resSize, 1 + self.inSize))
    self.W = random.uniform(-0.1, 0.1, (self.resSize, self.resSize))
    self.W_in_B = random.uniform(-0.1, 0.1, (self.resSize, 1 + self.inSize))
    self.W_B = random.uniform(-0.1, 0.1, (self.resSize, self.resSize))

    self.W_out = random.uniform(-0.1, 0.1, (self.resSize, self.outSize))
    self.W_out_B = random.uniform(-0.1, 0.1, (self.resSize, self.outSize))
    self.prev_W_out = zeros((self.resSize, self.outSize))
    self.prev_W_out_B = zeros((self.resSize, self.outSize))

    print ('Computing spectral radius...')
    rhoW = max(abs(linalg.eig(self.W)[0]))  # regression
    rhoW_B = max(abs(linalg.eig(self.W_B)[0]))
    print ('done.')
    print (rhoW)
    print (rhoW_B)

    self.W *= self.spectral_radius / rhoW
    self.W[random.rand(*self.W.shape) < self.sparsity] = 0
    self.x = zeros((self.resSize, 1))

    self.W_B *= self.spectral_radius / rhoW
    self.W_B[random.rand(*self.W_B.shape) < self.sparsity] = 0
    self.x_B = zeros((self.resSize, 1))

def init_error_accuracy(self):
    self.TrainError = 0
    self.TestError = 0
    self.TrainAccuracy = 0
    self.TestAccuracy = 0
```

```
def createplot(self):
    plot(array(self.TrainErrorplot), 'r--', array(self.TestErrorplot), 'b--')
    title("Train & Test Error")
    ylabel('Error')
    xlabel('Iterations')
    show()

def init_training(self, trainData, testData):
    self.trainData = trainData
    self.testData = testData
    self.trainLen = self.trainData.sizeOfAminoacids
    print(self.trainLen)
    self.testLen = self.testData.sizeOfAminoacids

    self.Yt = self.trainData.yt  # epithimiti exodos
    print(len(self.Yt))

def run(self, train_file,test_file):
    open_file = open(train_file, "r")
    lines = open_file.readlines()
    open_file2 = open(test_file, "r")
    lines2 = open_file2.readlines()
    #subset_size = len(lines) / 10 / 3+1
    #print(subset_size)
    self.TestErrorplot = []
    self.TrainErrorplot = []
    self.TrainAccuracyplot = []
    self.TestAccuracyplot = []

    sum = 0
    sum2 = 0
```

```
        self.bp = BackPropagation.BackPropagation(30, 0, 7*self.inSize, self.outSize, 0.6, self.momentum)
        self.bp.init_weights()
        self.init_weights()
        pTrain = storeProteins.storeProteins()
        pTest = storeProteins.storeProteins()

        testing_this_round = lines2 #lines[0:][:subset_size * 3]
        # xekina apo tin thesi [:] kai pigaine mexri tin [:]
        training_this_round = lines #lines[subset_size * 3:]
        pTrain.readProteins(training_this_round, self.window)
        pTest.readProteins(testing_this_round, self.window)
        self.init_training(pTrain, pTest)
        self.TrainAccuracyplot = []
        self.TestAccuracyplot = []

        print ("Running")
        for i in range(self.maxIteration):
            self.init_error_accuracy()
            self.training()
            self.TrainErrorplot.append(self.TrainError / len(lines)) #(len(lines) / 3 - subset_size)

            self.outTrain.write(
                str(self.TrainError) + " "+ str(self.TrainError / len(lines) )+ " " + str(self.TrainAccuracy) + " " + "\n")
            sum2 = sum2 + self.TrainAccuracy

            self.testing()
            self.TestErrorplot.append(self.TestError / len(lines2))

            self.outTest.write(str(self.TestError) +" " + str(self.TestError / len(lines2))+" " + str(self.TestAccuracy) + " " + "\n")
```

```
            sum = sum + self.TestAccuracy

            print(i)
        self.outTest.write(str(sum / self.maxIteration) + "\n")
        self.outTrain.write(str(sum2 / self.maxIteration) + "\n")

        self.createplot()

    def training(self):
        self.iterationTrain = 0
        error=0.0
                    overlap=3
        for w in range(0, self.trainLen, 1):
            if sum(self.trainData.data[
                (w + self.window - 1) * self.inSize:(w + self.window - 1) * self.inSize +
self.inSize]) == 0.0:
                self.TrainError = self.TrainError + 0.5 * error
                error = 0.0
                continue
            w_B = self.window - 1
            for iterW in range(0, self.window, 1):
                u = zip(*[iter(self.trainData.data[(w + iterW) * self.inSize:(
                                                        w + iterW) * self.inSize +
self.inSize])] * 1)  # eisodos tin dedomeni xroniki stigmi
                u_B = zip(*[iter(self.trainData.data[(w_B + w + self.window - 1) *
self.inSize:(
                                                            w_B + w + self.window -
1) * self.inSize + self.inSize])] * 1)

                self.x = array((1 - self.a) * self.x + self.a * tanh(
                    dot(self.W_in, vstack((u, [1.0]))) + dot(self.W, self.x)))  # anavathmisi
timwn twn nevrwnwn tou DR
                self.x_B = array((1 - self.a) * self.x_B + self.a * tanh(
```

```
            dot(self.W_in_B, vstack((u_B, [1.0]))) + dot(self.W_B, self.x_B)))

        if iterW == self.window - 1:
            self.bp.calculateOutput(
                zip(*[iter(self.trainData.data[(w + self.window - 1 -overlap) *
self.inSize:
                (w + self.window - 1+overlap) * self.inSize + self.inSize])] * 1))

        if w >= self.initLen:  # initial transient
            if iterW == self.window - 1:
                # ipologismos exodou
                y = dot(self.W_out.T, self.x) + dot(self.W_out_B.T, self.x_B)
                y = reshape(y, (1, self.outSize))[0]

                for i in range(0, self.outSize, 1):
                    y[i] += self.bp.outputL[i].output

                for i in range(0, self.outSize, 1):
                    y[i] = 1 / (1 + math.exp((-1.0) * y[i]))
                    self.bp.outputL[i].output = y[i]

                maxPosition = argmax(y)
                yout = zeros(3)
                yout[maxPosition] = 1

                self.delta = y * (1 - y) * (y - self.Yt[w + self.window - 1])
                # output * (1-output) * (output-target)

                newWeight = self.W_out - self.lr * self.delta * self.x \
                        + self.momentum * (self.W_out - self.prev_W_out)
                self.prev_W_out = self.W_out
                self.W_out = newWeight
```

```
                newWeight = self.W_out_B - self.lr * self.delta * self.x_B \
                        + self.momentum * (self.W_out_B - self.prev_W_out_B)
                self.prev_W_out_B = self.W_out_B
                self.W_out_B = newWeight

                self.iterationTrain += 1
                self.TrainAccuracy = self.TrainAccuracy + 1 - sum(
                   abs(self.Yt[w + self.window - 1] - yout)) / 2

                print (str(self.numToA(maxPosition)) + " " + str(
                   self.numToA(argmax(self.Yt[w + self.window - 1]))) +
                     " " + str(self.TrainAccuracy))

                er = self.Yt[w + self.window - 1] - y
                er=er**2
                error = error + sum(er)

            if iterW == self.window - 1:
               if w < self.initLen:
                  for i in range(0, self.outSize, 1):
                     self.bp.outputL[i].output = 1 / (1 + math.exp((-1.0) *
self.bp.outputL[i].output))
                     self.delta[i] = self.bp.outputL[i].output * (1 - self.bp.outputL[i].output)
* \
                              (self.bp.outputL[i].output - self.Yt[w + self.window - 1][i])

               self.bp.calculateDelta(self.delta)
               self.bp.changeWeights(
                  zip(*[iter(self.trainData.data[(w + self.window - 1-overlap) * self.inSize:
                  (w + self.window - 1+overlap) * self.inSize + self.inSize])] * 1))

            w_B -= 1
```

```
        self.TrainAccuracy = (self.TrainAccuracy / self.iterationTrain) * 100
        self.TrainAccuracyplot.append(self.TrainAccuracy)

    def testing(self):
        self.YtTest = self.testData.yt
        self.iterationTest = 0
        error = 0.0
                    overlap=3

        for w in range(0, self.testLen, 1):
            if sum(self.testData.data[
                   (w + self.window - 1) * self.inSize:(w + self.window - 1) * self.inSize +
self.inSize]) == 0.0:
                self.TestError = self.TestError + 0.5 * error
                error = 0.0
                continue
            w_B = self.window - 1
            for iterW in range(0, self.window, 1):
                u = zip(*[iter(self.testData.data[(w + iterW) * self.inSize:(
                                                                w + iterW) * self.inSize +
self.inSize])] * 1)  # eisodos tin dedomeni xroniki stigmi
                u_B = zip(*[iter(self.testData.data[(w_B + w + self.window - 1) *
self.inSize:(
                                                                    w_B + w + self.window -
1) * self.inSize + self.inSize])] * 1)

                self.x = array((1 - self.a) * self.x + self.a * tanh(
                    dot(self.W_in, vstack((u, [1.0]))) + dot(self.W, self.x)))  # anavathmisi
timwn twn nevrwnwn tou DR
                self.x_B = array((1 - self.a) * self.x_B + self.a * tanh(
                    dot(self.W_in_B, vstack((u_B, [1.0]))) + dot(self.W_B, self.x_B)))

                if iterW == self.window - 1:
```

```
self.bp.calculateOutput(
   zip(*[iter(self.testData.data[(w + self.window - 1-overlap) * self.inSize:
   (w + self.window - 1+overlap) * self.inSize + self.inSize])] * 1))



# ipologismos exodou
y = dot(self.W_out.T, self.x) + dot(self.W_out_B.T, self.x_B)
y = reshape(y, (1, self.outSize))[0]

for i in range(0, self.outSize, 1):
   y[i] += self.bp.outputL[i].output

for i in range(0, self.outSize, 1):
   y[i] = 1 / (1 + math.exp((-1.0) * y[i]))
   self.bp.outputL[i].output = y[i]

maxPosition = argmax(y)
yout = zeros(3)
yout[maxPosition] = 1



self.iterationTest += 1
self.TestAccuracy = self.TestAccuracy + 1 - sum(
      abs(self.YtTest[w + self.window - 1] - yout)) / 2

print (str(self.numToA(maxPosition)) + " " + str(
      self.numToA(argmax(self.YtTest[w + self.window - 1]))) +
        " " + str(self.TestAccuracy))

er = self.YtTest[w + self.window - 1] - y
er = er ** 2
error = error + sum(er)
```

```
            w_B -= 1

    self.TestAccuracy = (self.TestAccuracy / self.iterationTest) * 100
    self.TestAccuracyplot.append(self.TestAccuracy)

def numToA(self, possition):

    if (possition == 0):
        return ("C")
    elif (possition == 1):
        return ("E")
    elif (possition == 2):
        return ("H")
    else:
        print ("ERROR !!!")
```

Παράρτημα Γ

storePoteins.py

```
##
# Read data from file and organize them into input
# and target output
##
from numpy import *



class storeProteins:
    def __init__(self):
        self.sizeOfAminoacids = 0
        self.aminoacids = []
        self.data = []
        self.yt = []

    def readProteins(self, lines, window):
        leadingZeros = zeros((1, (window - 1) * 20))
        for i in range(0, len(lines), 3):
            name = lines[i].rstrip()
            secondary = lines[i + 2].rstrip()
            msa = loadtxt("../input/msaFilesCorrect/" + name + '.hssp')
            self.data = append(self.data, leadingZeros)
            self.data = append(self.data, msa / 100)
            self.data = append(self.data, leadingZeros)
            self.sizeOfAminoacids += len(msa) + 2 * (window - 1)
            self.normilizeOutput(secondary, window)
            """print("Store protein\n\n")
            print(self.data)
            print(self.sizeOfAminoacids)
            print(self.yt)"""
```

```
def readAminoAcids(self, fileName):
    with open("../input/msaFilesCorrect/" + fileName + ".hssp", "r+") as f_in:
        lines = filter(None, (line.rstrip() for line in f_in))
    return lines

def normilizeOutput(self, secondary, window):
    se = secondary.split()

    for i in range(0, (window - 1), 1):
        self.yt.append([0, 0, 0])

    for t in se[0]:

        if t == "C":
            self.yt.append([1, 0, 0])
            self.aminoacids.append(t)
        elif t == "E":
            self.yt.append([0, 1, 0])
            self.aminoacids.append(t)
        elif t == "H":
            self.yt.append([0, 0, 1])
            self.aminoacids.append(t)
        else:
            continue
            yt = [0, 1, 0]
            random.shuffle(yt)
            self.yt.append(yt)

    for i in range(0, (window - 1), 1):
        self.yt.append([0, 0, 0])
```

Παράρτημα Δ

BackPropagation.py

```
import Node
from numpy import *


class BackPropagation:
    def __init__(self, hidden1, hidden2, num_input, num_output, learningRate,
momentum):
        self.lr = learningRate
        self.momentum = momentum
        self.input = num_input
        self.hidden1 = hidden1
        self.hidden2 = hidden2
        self.output = num_output
        self.layer1 = []
        if hidden2 != 0:
            self.layer2 = []
        self.outputL = []


    def init_weights(self):
        self.layer1 = [Node.Node(self.input) for i in range(self.hidden1)]  # nodes of layer
1
        if self.hidden2 != 0:
            self.layer2 = [Node.Node(self.hidden1) for i in range(self.hidden2)]  # nodes of
layer 2
            self.outputL = [Node.Node(self.hidden2) for i in range(self.output)]  # nodes of
output layer
        else:
            self.outputL = [Node.Node(self.hidden1) for i in range(self.output)]  # nodes of
output layer
```

```
def calculateOutput(self, protein):
    y = 0.0
    a = 1.0
    protein = vstack(protein + [1.0])  # protein+bias
    for i in range(0, self.hidden1, 1):
        for j in range(0, self.input + 1, 1):
            # add all weight * input for each input of the nodes
            y = y + protein[j] * self.layer1[i].weight[j]
        # pass the sum through the bias function
        y = 1 / (1 + math.exp((-a) * y))
        self.layer1[i].output = y  # save output in node
        y = 0.0

    if self.hidden2 != 0:
        for i in range(0, self.hidden2, 1):
            for j in range(0, self.hidden1, 1):
                # add all weight * input for each input of the nodes
                y = y + self.layer1[j].output * self.layer2[i].weight[j]
            y += self.layer2[i].weight[self.hidden1]  # add the weight*bias
            # pass the sum through the bias function
            y = 1 / (1 + math.exp((-a) * y))
            self.layer2[i].output = y  # save output in node
            y = 0.0

    for i in range(0, self.output, 1):
        if self.hidden2 != 0:
            for j in range(0, self.hidden2, 1):
                # add all weight * output of 2nd hidden layer for each input of the nodes
                y = y + self.layer2[j].output * self.outputL[i].weight[j]
            y += self.outputL[i].weight[self.hidden2]  # add the weight * bias
        else:
            for j in range(0, self.hidden1, 1):
```

```
            # add all weight * output of 1st hidden layer for each input of the nodes
            y = y + self.layer1[j].output * self.outputL[i].weight[j]
        y += self.outputL[i].weight[self.hidden1]

    self.outputL[i].output = y  # save output in node
    y = 0.0

def calculateDelta(self, delta):

  for i in range(0, self.output, 1):
    self.outputL[i].delta = delta[i]

  delta = 0.0
  if self.hidden2 != 0:
    for i in range(0, self.hidden2, 1):
      for j in range(0, self.output, 1):
        # add all weight * delta(of next layer) for each node
        delta = delta + self.outputL[j].weight[i] * self.outputL[j].delta

      # sum of weights and deltas(of next layer) * output * (1-output)
      delta = delta * self.layer2[i].output * (1 - self.layer2[i].output)
      self.layer2[i].delta = delta  # save delta
      delta = 0.0

  for i in range(0, self.hidden1, 1):
    if self.hidden2 != 0:
      # add all weight * delta(of next layer) for each node
      for j in range(0, self.hidden2, 1):
        delta = delta + self.layer2[j].weight[i] * self.layer2[j].delta
    else:
      for j in range(0, self.output, 1):
        delta = delta + self.outputL[j].weight[i] * self.outputL[j].delta
```

```
            # sum of weights and deltas(of next layer) * output * (1-output)
            delta = delta * self.layer1[i].output * (1 - self.layer1[i].output)
            self.layer1[i].delta = delta  # save delta
            delta = 0.0

    def changeWeights(self, protein):
        newWeight = 0.0
        protein = vstack(protein + [1.0])  # protein+bias
        # changes the weights of the 1st hidden layer
        for i in range(0, self.hidden1, 1):
            for j in range(0, self.input + 1, 1):
                # calculate new weight
                newWeight = self.layer1[i].weight[j] - self.lr * self.layer1[i].delta * \
                                            protein[j] + self.momentum * (
                    self.layer1[i].weight[j] - self.layer1[i].prevWeight[j])
                # save current weight as previous
                self.layer1[i].prevWeight[j] = self.layer1[i].weight[j]
                self.layer1[i].weight[j] = newWeight  # set new weight value
                newWeight = 0.0

        if self.hidden2 != 0:
            for i in range(0, self.hidden2, 1):
                for j in range(0, self.hidden1, 1):
                    # calculate new weight
                    newWeight = self.layer2[i].weight[j] - self.lr * self.layer2[i].delta *
self.layer1[j].output \
                                + self.momentum * (self.layer2[i].weight[j] -
self.layer2[i].prevWeight[j])
                    # save current weight as previous
                    self.layer2[i].prevWeight[j] = self.layer2[i].weight[j]
                    self.layer2[i].weight[j] = newWeight  # set new weight value
                    newWeight = 0.0
```

```
            # new weight for bias
            newWeight = self.layer2[i].weight[self.hidden1] - self.lr * self.layer2[i].delta \
                    + self.momentum * (
               self.layer2[i].weight[self.hidden1] -
self.layer2[i].prevWeight[self.hidden1])
            # save current weight as previous
            self.layer2[i].prevWeight[self.hidden1] = self.layer2[i].weight[self.hidden1]
            self.layer2[i].weight[self.hidden1] = newWeight  # set new weight value
            newWeight = 0.0

      for i in range(0, self.output, 1):
         if self.hidden2 != 0:
            for j in range(0, self.hidden2, 1):
               # calculate new weight
               newWeight = self.outputL[i].weight[j] - self.lr * self.outputL[i].delta *
self.layer2[
                  j].output \
                       + self.momentum * (self.outputL[i].weight[j] -
self.outputL[i].prevWeight[j])
               # save current weight as previous
               self.outputL[i].prevWeight[j] = self.outputL[i].weight[j]
               self.outputL[i].weight[j] = newWeight  # set new weight value
               newWeight = 0.0

            # new weight for bias
            newWeight = self.outputL[i].weight[self.hidden2] - self.lr *
self.outputL[i].delta \
                    + self.momentum * (
               self.outputL[i].weight[self.hidden2] -
self.outputL[i].prevWeight[self.hidden2])
            # save current weight as previous
            self.outputL[i].prevWeight[self.hidden2] =
self.outputL[i].weight[self.hidden2]
```

```
            self.outputL[i].weight[self.hidden2] = newWeight  # set new weight value
        else:
            for j in range(0, self.hidden1, 1):
                newWeight = self.outputL[i].weight[j] - self.lr * self.outputL[i].delta *
self.layer1[j].output \
                            + self.momentum * (self.outputL[i].weight[j] -
self.outputL[i].prevWeight[j])

                # save current weight as previous
                self.outputL[i].prevWeight[j] = self.outputL[i].weight[j]
                self.outputL[i].weight[j] = newWeight  # set new weight value
                newWeight = 0.0

            # new weight for bias
            newWeight = self.outputL[i].weight[self.hidden1] - self.lr *
self.outputL[i].delta + self.momentum \
                        * (self.outputL[i].weight[self.hidden1] -
self.outputL[i].prevWeight[self.hidden1])
            # save current weight as previous
            self.outputL[i].prevWeight[self.hidden1] =
self.outputL[i].weight[self.hidden1]
            self.outputL[i].weight[self.hidden1] = newWeight

        newWeight = 0.0
```

Παράρτημα Ε

Node.py

```
from numpy import *


class Node:
    def __init__(self, inputs):
        self.prevWeight = [0.0 for x in range(inputs+1)]
        self.delta = 0
        self.output = 0
        random.seed()
        self.weight = random.uniform(-0.1, 0.1, (inputs+1))
```