Ατομική Διπλωματική Εργασία

ΠΡΟΒΛΕΨΗ ΤΗΣ ΔΕΥΤΕΡΟΤΑΓΟΥΣ ΔΟΜΗΣ ΤΩΝ ΠΡΩΤΕΪΝΩΝ ΜΕ ΤΗΝ ΒΟΗΘΕΙΑ CLOCKWORK NETWORK

ΒΑΝΘΙΑ ΤΟΥΜΠΟΥΡΗ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2017

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόβλεψη της Δευτεροταγούς δομής των Πρωτεϊνών με την βοήθεια Clockwork Network

Βάνθια Τουμπουρή

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

Πρώτα απ’ όλα θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κύριο Χρίστο Χριστοδούλου για την εμπιστοσύνη που μου έδειξε, για την ανάθεση της παρούσας Ατομικής Διπλωματικής Εργασίας, την ηθική ενθάρρυνση και τις πολύτιμες συμβουλές που μου παρείχε για την εκπόνηση της.

Ένα μεγάλο ευχαριστώ στον διδακτορικό φοιτητή κύριο Μ. Αγαθοκλέους για την τόσο σημαντική βοήθεια , τις χρήσιμες συμβουλές και την καθοδήγηση που μου παρείχε. Η βοήθεια του ήταν καθοριστική.

Τέλος, θα ήταν παράλειψη μου να μην ευχαριστήσω τους γονείς μου και τα αδέλφια μου για την αγάπη τους, την ανεκτίμητη συμπαράσταση τους και την ηθική υποστήριξη καθόλη την διάρκεια των σπουδών μου.

Περίληψη

Σκοπός της παρούσας Ατομικής Διπλωματικής Εργασίας είναι η ανάπτυξη μιας διαφορετικής μεθόδου, για την επίλυση του προβλήματος της Πρόβλεψης της Δευτεροταγούς δομής των πρωτεϊνών, με την χρήση δικτύου clockwork-RNN.

Κατ'αρχάς, γίνεται μια σύντομη βιβλιογραφική ανασκόπηση για τις πρωτεΐνες και την σημαντικότητα τους καθώς και για τα νευρωνικά δίκτυα. Στην συνέχεια, ακολουθεί ανάλυση και επεξήγηση των δεδομένων που δόθηκαν σαν είσοδος στο δίκτυο όπως επίσης και για την αρχιτεκτόνική και λειτουργικότητα του δικτύου clockwork-RNN. Επιπρόσθετα, παρουσιάζεται η δομή και η υλοποίηση του δικτύου, το οποίο χρησιμοποιήθηκε για την επίλυση του προβλήματος και είναι υλοποιημένο στην γλώσσα προγραμματισμού Python. Τέλος, η εργασία αυτή ολοκληρώνεται με την ανάλυση των αποτελεσμάτων του δικτύου και την εξαγωγή συμπερασμάτων.

Το clockwork -RNN δίκτυο που χρησιμοποιήθηκε για την επίλυση του προβλήματος είχε ως αρχικό αποτέλεσμα ποσοστό επιτυχίας μέχρι 64.75%. Δεν υπάρχει αμφιβολία, πως τα αποτελέσματα του δικτύου είναι ικανοποιητικά για το παρόν στάδιο , αφού σύμφωνα με προηγούμενες αντιστοιχες μελέτες για την επίλυση του προβλήματος αυτού το καλύτερο ποσοστό ακρίβειας που έχει επιτευχθεί μεχρί στιγμής ανέρχεται στο 76%. Ωστόσο, έχει καταστεί σαφές πως υπάρχουν αρκετά περιθώρια βελτίωσης στο μέλλον για να επιτευχθεί ακόμη καλύτερο ποσοστό επιτυχίας με την χρήση του παρόντος δικτύου.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Πρωτεΐνες

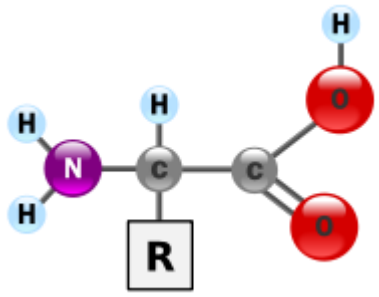
Οι πρωτεΐνες είναι ουσίες με κυρίαρχο και πρωταρχικό ρόλο στη ζωή. Άλλωστε το όνομά τους υποδηλώνει το ρόλο αυτό. Το όνομα πρωτείνη προέρχεται από τη λέξη «πρώτα» που σημαίνει πρωταρχικής σημασίας και περιγράφηκε και ονομάστηκε έτσι από τον Σουηδό χημικό Γιονς Γιάκομπ Μπερτσέλιους το 1838(Βικπαίδεια,2017). Επίσης, λέγονται και λευκώματα λόγω του λευκού χρώματος που έχουν πολλές από αυτές (Σημαντήρης, 2005). Αποτελούν απαραίτητο στοιχείο για κάθε ζωντανό οργανισμό για την ανάπτυξη και ανακατασκευή των ιστών, την καλή λειτουργία και δομή όλων των ζωντανών κυττάρων. Είναι βασικά οι εργάτες της βιοχημείας, συμμετέχοντας σε όλες σχεδόν τις κυτταρικές διεργασίες (Προμπονάς, 2008).

Οι πρωτεϊνες αποτελούν τα πιο διαδεδομένα και πολυδιάστατα, τόσο στη μορφή όσο και στη λειτουργία τους, μακρομόρια . Είναι μεγάλα σύνθετα βιομόρια με μοριακό βάρος απο 10000 μέχρι πάνω από 1 εκατομύριο αποτελούμενα από πολλά αμινοξέα (περισσότερα από εκατό) που ενώνονται μεταξύ τους με πεπτιδικούς δεσμούς και σχηματίζουν την γραμμική αλυσίδα των πολυπεπτιδίων . Κάθε πρωτεϊνη έχει τον δικό της αριθμό αμινοξέων και την δική της αλληλουχία. Τα αμινοξέα μπορούν να τοποθετηθούν με εκατομύρια διαφορετικούς τρόπους. Ανάλογα με την ακολουθία που συνδιάζονται η πρωτείνη που προκύπτει είναι υπεύθυνη για συγκεκριμένες λειτουργίες του σώματος. Όλες οι πρωτεΐνες περιέχουν άνθρακα, οξυγόνο και άζωτο και οι περισσότερες από αυτές και θείο.(Προμπονάς, 2006).

1.2 Αμινοξέα

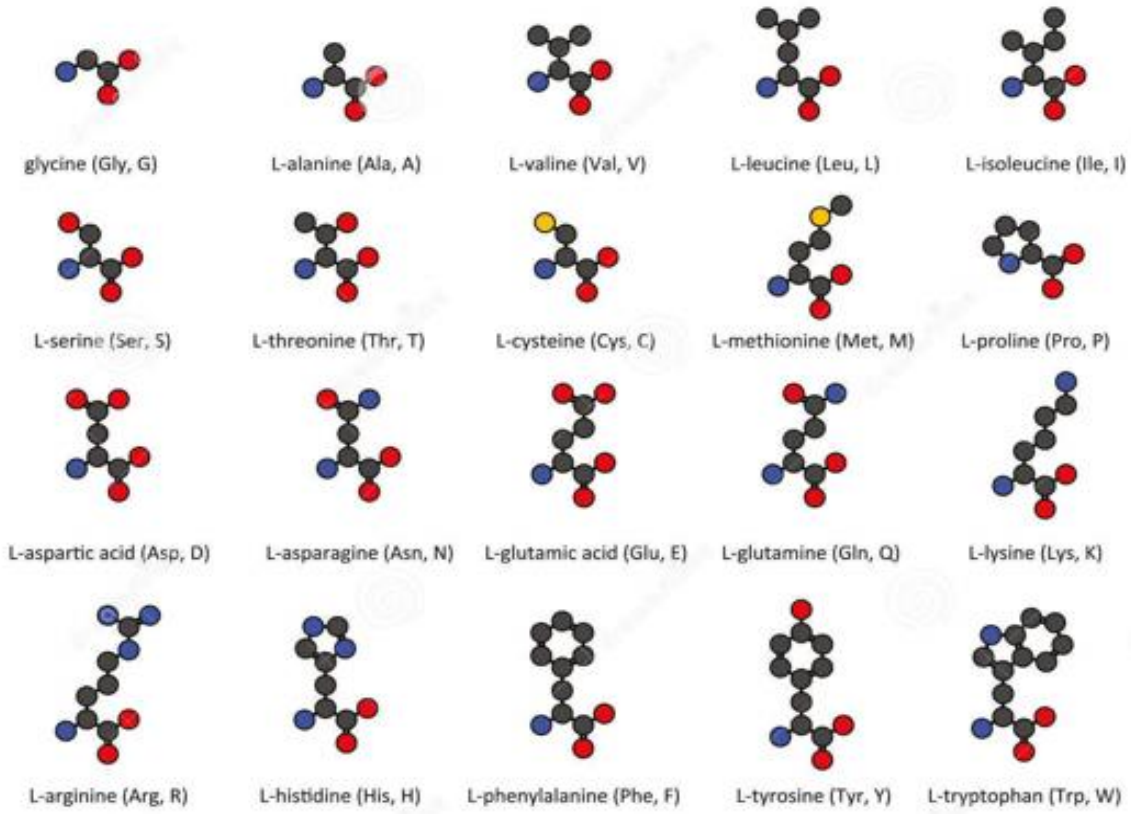
Αμινοξέα λέγονται οι χημικές ενώσεις που περιέχουν μία τουλάχιστον καρβονική ομάδα (από τα καρβονικά οξέα (RCOOH[1]) και μία τουλάχιστον αμινομάδα (-NH_2) και αποτελούν τα βασικά δομικά στοιχεία των πρωτεϊνών που καθορίζουν και τις χαρακτηριστικές ιδιότητές τους. (Σημαντήρης, 2005,Προμπονάς, 2006). Συμβολίζονται

με τον κωδικό του ενός ή των τριών γραμμάτων , επομένως η αμυνοξευκή ακολουθία ενός πολυπεπτιδίου μπορεί να αναπαρασταθεί ως συμβολοσειρά. (Αγαθοκλέους, 2009).



Σχήμα1.2.1: Γενικός Συντακτικός τύπος δομής των α-αμινοξέων. (https://el.wikipedia.org/wiki/Amino_acid)

Οι πρωτεΐνες δομούνται από ένα σύνολο 20 αμινοξέων. Είναι δηλαδή γραμμικά πολυμερή αμινοξέων. Κάθε αμινοξύ αποτελείται από ένα κεντρικό τετρασθενές άτομο άνθρακα που συνδέεται με μια αμινική ομάδα, μια καρβοξυλική ομάδα, μια χαρακτηριστική πλευρική αλυσίδα και ένα υδρογόνο. Αυτά τα τετραεδρικά κέντρα, με εξαίρεση εκείνο της γλυκίνης, είναι χειρόμορφα και μόνο το L ισομερές απαντά στις φυσικές πρωτεΐνες. Όλες οι φυσικές πρωτεΐνες δομούνται από τα ίδια 20 αμινοξέα. Οι πλευρικές αλυσίδες των 20 αυτών δομικών μορίων διαφέρουν κατά πολύ σε μέγεθος, σε σχήμα και σε παρουσία λειτουργικών ομάδων. Τα αμινοξέα μπορούν να ταξινομηθούν σε ομάδες όπως: α) αλειφατικές πλευρικές αλυσίδες γλυκίνη, αλανίνη, βαλίνη, ισολευκίνη, λευκίνη, με- θειονίνη και προλίνη, β) αρωματικές πλευρικές αλυσίδες – φαινυλαλανί- νη, τυροσίνη, θρυπτοφάνη, γ) υδροξυλικές-αλειφατικές πλευρικές αλυσί- δες – σερίνη, θρεονίνη, δ) κυστεΐνες που περιέχουν σουλφυδρύλια, ε) βασικές πλευρικές αλυσίδες – λυσίνη, αργινίνη και ιστιδίνη, στ) όξινες πλευρικές αλυσίδες – ασπαραγινικό και γλουταμινικό οξύ και ζ) αμιδικές πλευρικές αλυσίδες – ασπαραγίνη και γλουταμίνη. Οι ομαδοποιήσεις αυτές είναι κάπως αυθαίρετες και φυσικά είναι πιθανές πολλές άλλες δυνατότητες ομαδοποιήσεων. (Branden and Tooze, 2006).



Σχήμα 1.2.2: Τα 20 βασικά αμινοξέα (https://gr.dreamstime.com/comm-image30574281)

1.3 Δομή των Πρωτεϊνών

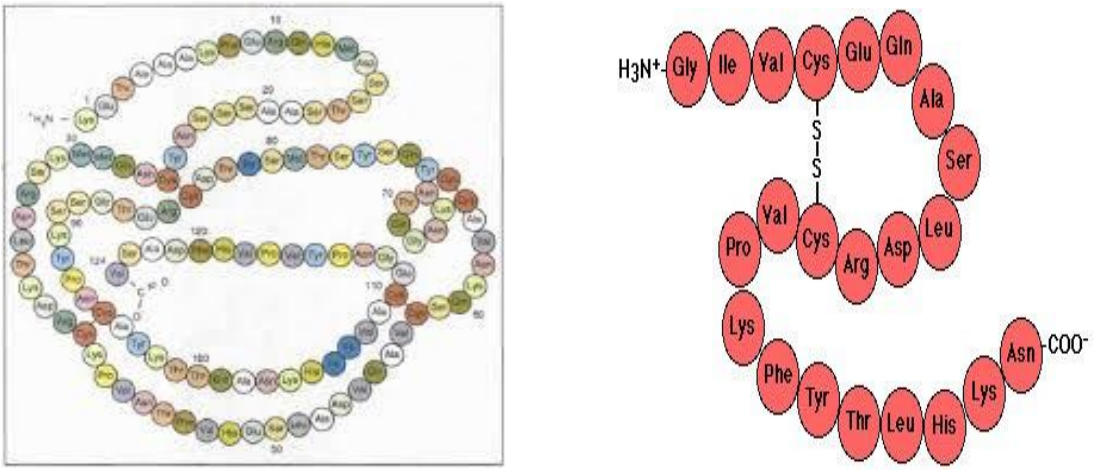
Η μελέτη της δομής των πρωτεινών γίνεται σε τέσσερα επίπεδα και περιλαμβάνει την πρωτοταγή, δευτεροταγή , τριτοταγή και τεταρτοταγή δομή. Σύμφωνα με τον Προμπονά (2008) ακολουθείται μια ιεραρχία στην δομή των πρωτεϊνών.

1.3.1 Πρωτοταγής Δομή

Οι πρωτείνες παράγονται στο κυτταρόπλασμα και πιο συγκεκριμένα στα ριβοσώματα. Η αλληλουχία των αμινοξέων μιας πρωτείνης σχηματίζει την «πρωτοταγή δομή».

Καθοριστικοί παράγοντες είναι τα νουκλεϊνικά οξέα τα οποία παίζουν σημαντικό ρόλο στις λειτουργίες και τα κληρορονομικά χαρίσματα των οργανισμών. Βασικά οι πρωτεΐνες είναι γραμμικά πολυμερή που δημιουργούνται δεσμεύοντας την α-καρβοξυλική ομάδα ενός αμινοξέος στην α-αμινική ομάδα ενός άλλου αμινοξέος με έναν πεπτιδικό δεσμό (αμιδικός δεσμός). Η δημιουργία ενός διπεπτιδίου από δύο αμινοξέα συνοδεύεται από την απώλεια ενός μορίου ύδατος . Η ισορροπία της αντίδρασης βρίσκεται προς την πλευρά της υδρόλυσης παρά της σύνθεσης. Επομένως, η βιοσύνθεση του πεπτιδικού δεσμού χρειάζεται την προσθήκη ελεύθερης ενέργειας. Παρ' όλα αυτά οι πεπτιδικοί δεσμοί είναι αρκετά σταθεροί κινητικά. Μια σειρά αμινοξέων που ενώνονται με πεπτιδικούς δεσμούς δημιουργούν μια πολυπεπτιδική αλυσίδα, και κάθε μονάδα αμινοξέος στο πολυπεπτίδιο ονομάζεται κατάλοιπο (Branden and Tooze, 2006). Μια πολυπεπτιδική αλυσίδα έχει πολικότητα διότι τα δύο άκρα της είναι διαφορετικά: μια α-αμινική ομάδα στο ένα άκρο, μια α-καρβοξυλική ομάδα στο άλλο άκρο. Το αμινοτελικό άκρο θεωρείται η αρχή της πολυπεπτιδικής αλυσίδας και επομένως η αλληλουχία των αμινοξέων σε μια πολυπεπτιδική αλυσίδα γράφεται αρχίζοντας με το αμινοτελικό.(Branden and Tooze, 2006).

Τα αμινοξέα στην πολυπεπτιδική αλυσίδα συνδέονται με αμιδικούς δεσμούς που δημιουργούνται μεταξύ της καρβοξυλικής ομάδας ενός αμινοξέος και της αμινικής ομάδας του επομένου. Η σύνδεση αυτή, που ονομάζεται πεπτιδικός δεσμός, έχει πολλές σημαντικές ιδιότητες. Πρώτον, είναι εντυπωσιακά ανθεκτική στην υδρόλυση, με συνέπεια οι πρωτεΐνες να χαρακτηρίζονται από κινητική σταθερότητα. Δεύτερον, η πεπτιδική ομάδα είναι επίπεδη διότι ο δεσμός C—N έχει χαρακτηριστικά μερικού διπλού δεσμού. Τρίτον, κάθε πεπτιδικός δεσμός έχει έναν δότη (την ομάδα NH) και έναν δέκτη (την ομάδα CO) δεσμών υδρογόνου. Η δημιουργία δεσμών υδρογόνου μεταξύ των ομάδων αυτών του κορμού είναι ιδιαίτερο χαρακτηριστικό της πρωτεϊνικής δομής. Τέλος, ο πεπτιδικός δεσμός δεν έχει φορτίο, επιτρέποντας έτσι στις πρωτεΐνες να δημιουργούν συμπαγείς σφαιρικές δομές με το μεγαλύτερο μέρος του κορμού βυθισμένο στο εσωτερικό της πρωτεΐνης. Βασικά οι πρωτεΐνες, επειδή ειναι γραμμικά πολυμερή, μπορούν να περιγραφούν και ως αλληλουχίες αμινοξέων. Και αυτές οι αλληλουχίες γράφονται με κατεύθυνση από το αμινοτελικό προς το καρβοξυτελικό άκρο.

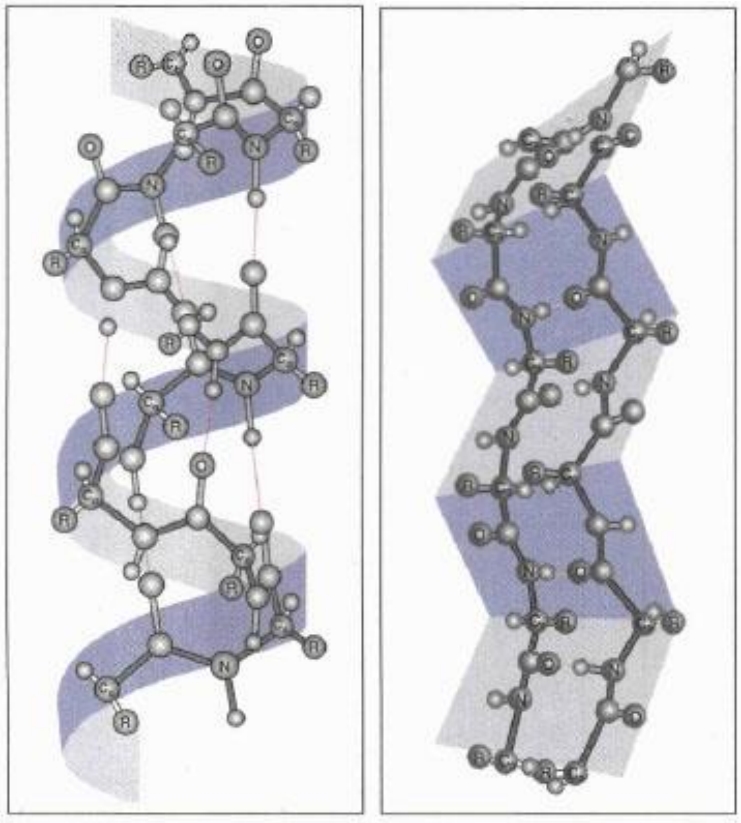


Σχήμα 1.3.1.1 : Η πρωτοταγή δομή μιας πρωτεϊνης
(http://ebooks.edu.gr/modules/ebook/show.php/DSGL-C120/480/3166,12748/)

1.3.2 Δευτεροταγής Δομή

Ακολούθως όλα τα πρωτεϊνικά μόρια υφίστανται μια φυσική αναδιάταξη με σκοπό να δώσουν μια «δευτεροταγή δομή» η οποία αυτή η μορφή είναι η λεγόμενη « α- ελικα» - δεξιόστροφη, η οποία προκαλείται από δεσμούς υδρογόνου που αναπτύσσονται μεταξύ των καρβοξυλομάδων και των αμινομάδων των αμινοξέων. Δηλαδή η δευτεροταγής δομή αναφέρεται στη στερεοδιάταξη των τοπικών περιοχών της πολυπεπτιδικής αλυσίδας. Στην α-έλικα η πολυπεπτιδική αλυσίδα ελίσσεται και δημιουργεί μια συμπαγή ράβδο. Στο εσωτερικό της έλικας η ομάδα CO κάθε αμινοξέος δεσμεύεται με δεσμό υδρογόνου στην ομάδα NH του αμινοξέος που βρίσκεται τέσσερα κατάλοιπα πιο κάτω στην πολυπεπτιδική αλυσίδα.

Ένα άλλο κύριο δομικό στοιχείο της δευτεροταγούς δομής είναι η λεγόμενη «β-πτυχωτή» επιφάνεια. όπου στη περίπτωση αυτή διασταυρώνονται παράλληλες αλυσίδες πολυπεπτιδίων που ενώνονται στις διασταυρώσεις με δεσμούς υδρογόνου μεταξύ NH και CO σχηματίζοντας έτσι μια εξαιρετικά σφιχτή δομή, όπως στο μετάξι. Οι πρωτεινες σε αυτή την δομή ονομάζονται ινώδεις. Αξιοσημείωτο είναι ότι ενώ η πρωτοταγής δομή κάθε πρωτεϊνης είναι μοναδική, η δευτεροταγής δομή πολλών διαφορετικών πρωτεϊνων μπορεί να είναι ίδια.

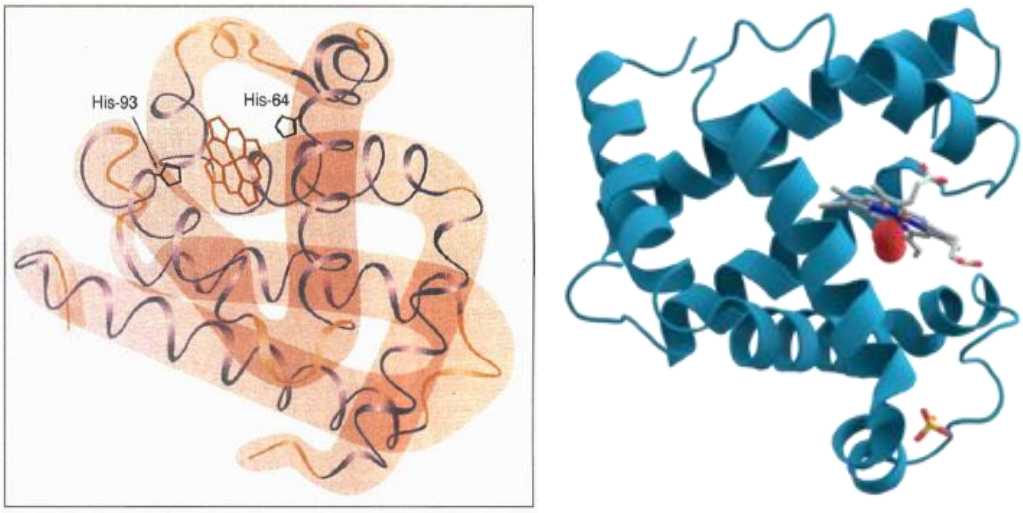


Σχήμα 1.3.2.1: Η δευτεροταγής δομή μιας πρωτεΐνης. Στα αριστερά αναπαραστάται ο α-έλικας και στα δεξιά η β-πτυχωτή μορφή.
(http://ebooks.edu.gr/modules/ebook/show.php/DSGL-C120/480/3166,12748/)

1.3.3 Τριτοταγής Δομή

Στην συνέχεια οι πρωτείνες υφίστανται σε ακόμη πιο πολύπλοκη πτύχωση που ονομάζεται «τριτοταγής δομή» και η οποία είναι το τελικό λειτουργικό σχήμα της πρωτείνης. Εδώ γίνεται η συνολική αναδίπλωση της πολυπεπτιδικής αλυσίδας. Η συμπαγής, ασύμμετρη δομή του κάθε πολυπεπτιδίου στον χώρο λέγεται τριτοταγής δομή. Οι τριτοταγείς δομές των υδατοδιαλυτών πρωτεϊνών έχουν κοινά χαρακτηριστικά: πρώτα από όλα στο εσωτερικό τους τοποθετούνται τα αμινοξέα με υδρόφοβες πλευρικές αλυσίδες και η επιφάνειά τους περιέχει κατά κύριο λόγο τα υδρόφιλα αμινοξέα που αλληλεπιδρούν με το υδάτινο περιβάλλον. Η κινητήρια δύναμη των αναδιπλώσεων των υδατο- διαλυτών πρωτεϊνών είναι οι υδροφοβικές αλληλεπιδράσεις των αμινοξέων του

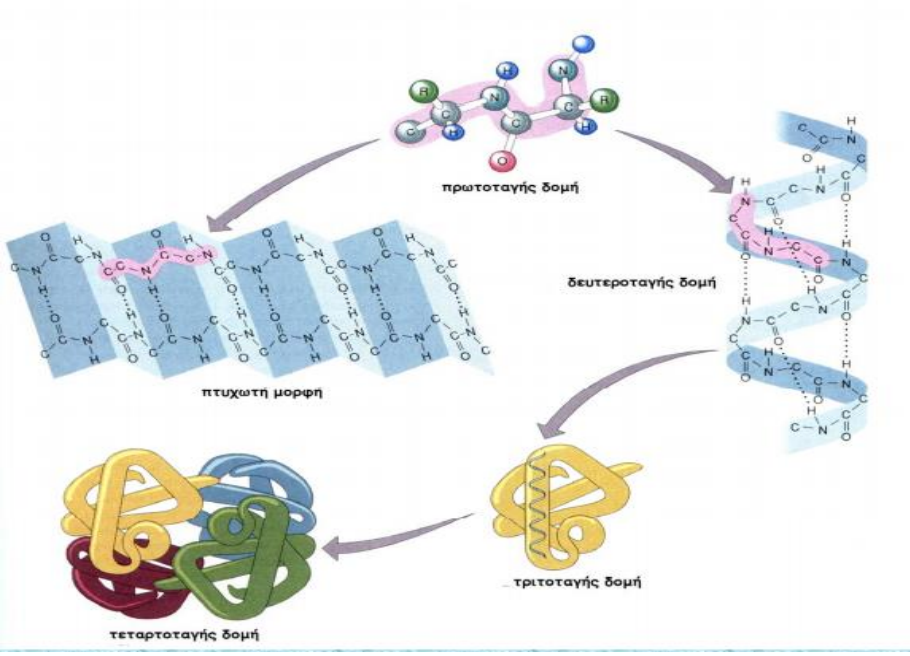
εσωτερικού. Μερικές πρωτεΐνες που βρίσκονται σε υδρόφοβο περιβάλλον, όπως στις μεμβράνες, εμφανίζουν αντίθετη κατανομή υδρόφοβων και υδρόφιλων αμινοξέων. Στις πρωτεΐνες αυτές τα υδρόφοβα αμινοξέα βρίσκονται στην επιφάνεια της πρωτεΐνης για να αλληλεπιδρούν με το περιβάλλον, ενώ οι υδρόφιλες ομάδες βυθίζονται στο εσωτερικό της πρωτεΐνης αποφεύγοντας το περιβάλλον.



Σχήμα1.2.3.1: Η τριτοταγής δομή της πρωτεϊνης(https://en.wikipedia.org/wiki/MreB)

1.3.4 Τεταρτοταγής Δομή

Τέλος, η «τεταρτοταγής δομή» αναφέρεται στην ειδική σύνθεση του συνόλου των πολυπεπτιδικών αλυσίδων μιας πρωτεΐνης και στη δημιουργία συμπλόκων πολλών υπομονάδων. Οι πρωτεΐνες που αποτελούνται από περισσότερες της μιας πολυπεπτιδικές αλυσίδες εμφανίζουν τεταρτοταγή δομή και το κάθε ένα από τα ανεξάρτητα πολυπεπτίδια ονομάζεται υπομονάδα. Οι τεταρτοταγείς δομές μπορεί να είναι απλές όταν αποτελούνται από δύο ίδιες υπομονάδες ή πολύπλοκες όταν αποτελούνται από πολλές διαφορετικές υπομονάδες. Στις περισσότερες περιπτώσεις οι υπομονάδες συγκρατούνται με μη ομοιοπολικούς δεσμούς. οι οποίες είναι χαλαρά ενωμένες όπως π.χ η αιμοσφαιρίνη.



Σχήμα1.3.4.1: Η εξέλιξη της δομής της πρωτεϊνης από την πρωτοταγή δομή ως την τεταρτοταγή (http://www.particlesciences.com/news/technical-briefs/2009/protein-structure.html)

Τα επίπεδα δομής των πρωτεϊνών

Επίπεδο δομής	Περιγραφή	Τύποι δεσμών
Πρωτοταγής	Αμινοξέα στην σειρά συνδεδεμένα με πεπτιδικούς δεσμούς	Ομοιπολικοί (πεπτιδικοί) μεταξύ των αμινοξέων
Δευτεροταγής	Σπειροειδής (α-έλικας) ή διάταξη σε ένα απίπεδο (β-επιφάνεια)	Δεσμοί υδρογόνων μεταξύ των στοιχείων που απαρτίζουν τους πεπτιδικούς δεσμούς.
Τριτοταγής	Πολυπεπτιδική αλυσίδα με αναδίπλωση και συστροφές στον χώρο.	Δεσμοί υδρογόνου ιοντικοί 5-5 υδρόφοβες δυνάμεις. Το 5 συστατικό των αμινοξέων κυστεϊνη και μεθιονίνη
Τεταρτοταγής	Μερικές πολυπεπτιδικές αλυσίδες ενώ μπορεί να υπάρχουν και μη πρωτεϊνικές ομάδες	Δεσμοί υδρογόνου και ιοντικοί δεσμοί μεταξύ πολυπεπτιδικών αλυσίδων.

Πίνακας 1.3.4.1: Πίνακας ο οποίος αναπαριστά τα επίπεδα της δομής των πρωτεϊνών και τους αντίστοιχους τύπους δεσμών.

1.4 Είδη των πρωτεϊνών

Οι πρωτείνες ανάλογα με την μορφή τους διακρίνονται σε ινώδεις και σφαιρικές πρωτεΐνες.

Με κριτήριο την σύνθεση τους διακρίνονται σε απλές όταν αποτελούνται μόνο από αμινοξέα και σύνθετες ή αλλοιώς πρωτείδια όταν στο μόριο τους περιλαμβάνονται και μη πρωτεινικά τμήματα όπως μέταλλα, σάγχαρα, λίπη κτλ.

Με βάση την λειτουργία τους διακρίνονται σε δομικές. (όταν αποτελούν τα δομικά υλικά του κυττάρου) και λειτουργικές όταν συμβάλλουν σε κάποιες λειτουργίες.
Με κριτήριο τον αριθμό κια το είδος των αμινοξέων που περιέχουν, χωρίζονται σε πρωτεΐνες υψηλής και χαμηλής βιολογικής αξίας.(Βικπαίδεια 2017).

1.5 Βιολογικός Ρόλος των Πρωτεϊνών

Είναι γενικά αποδεκτό ότι ο ρόλος των πρωτεϊνων είναι πολυδιάστατος και πολύ σωστά ο Προμπονάς(2008) τις χαρακτηρίζει ως την εργαλειοθήκη του κυττάρου. Ο βιολογικός ρόλος των πρωτεϊνων καθορίζεται κάθε φορά από την τρισδιάστατη δομή τους και κάθε ιδιότητα η οποία χαρακτηρίζει ένα ζωντανό οργανισμό επηρεάζεται από αυτές. (Χριστοδούλου, 2010).

- Πολλές πρωτεΐνες δρουν ως ένζυμα που καταλύουν τις βιοχημικές αντιδράσεις, και είναι ζωτικής σημασίας στο μεταβολισμό. Ο ρόλος των ενζύμων είναι να διευκολύνουν τις χημικές αντιδράσεις μέσα σε ένα οργανισμό, λειτουργόντας ως βιολογικοί καταλύτες.

- Άλλες πρωτεΐνες έχουν δομικές ή μηχανικές λειτουργίες, όπως οι πρωτεΐνες του κυτταρικού σκελετού, οι οποίες συμβάλλουν στη διατήρηση της μορφής των κυττάρων. Οι πρωτεϊνες όπ[ως η ελαστίνη και το κολλαγόνο, είναι συστατικά των συνδέσμων των οστών και του συνδετικού ιστού.

- Είναι το κύριο συστατικό του μυϊκού ιστού. Η ακτίνη και η μυοσίνη προκαλούν την σύσπαση των μυίκών ινών. Αυτές είναι οι κινητικές πρωτεΐνες.

- Οι πρωτεΐνες είναι επίσης σημαντικές στη δράση του ανοσοποιητικού συστήματος στην άμυνα εναντίον εισβολέων, τον σχηματισμό κυτταρικών ιστών, και τον κυτταρικό κύκλο. Οι πρωτεϊνες αυτές παράγονται από τον ίδιο τον οργανισμό και η δομή τους είναι τέτοια που έχουν καθοριστικό ρόλο να δεσμεύουν και να εξουδετερόνουν το ξένο σώμα (αντιγόνο).

- Κάποιες άλλες πρωτεϊνες επίσης δρουν ως ορμόνες. Η ινσουλίνη και η γλυκαγόνη που εκκρίνονται από το πάγκρεας ειναι ορμόνες που ρυθμίζουν τα επίπεδα σακχάρου στο αίμα.

- Ορισμένες πρωτεϊνες έχουν μεταφορικό ρόλο. Ένα παράδειγμα είναι η αιμοσφαιρίνη η οποία είναι υπεύθυνη για την μεταφορά οξυγόνου στο αίμα, ενώ η μυοσφαιρίνη είναι υπεύθυνη για την πρόσληψη οξυγόνου από τους μυς.

1.6 Πρόβλημα PSSP

Η τρισδιάστατη δομή της πρωτεΐνης, παρέχει όλη την πληροφορία που χρειαζόμαστε για μια πρωτεΐνη, γι' αυτό και το πρόβλημα της πρόβλεψης του σχήματος των πρωτεϊνών θεωρείται φλέγων ζήτημα για τον κλάδο της ιατρικής, της φαρμακευτικής και της βιοπληροφορικής, αλλά η ανάλυσή της αποτελεί ακόμη δυσκολότερο κομμάτι μελέτης για τη βιολογία. Η γνώση της δομής των πρωτεϊνών είναι πάρα πολύ σημαντική αφού μπορεί να μας δώσει σημαντικές πληροφορίες για τον τρόπο δράσης τους και να καθοδογήσουν την επινόηση νέων πρωτεϊνών με επιθυμιτές ιδιότητες. Οι πειραματικές μέθοδοι προσδιορισμού της τριτοταγούς δομής είναι χρονοβόρες και ασύλληπτα δαπανηρές, αφήνοντας την επιστημονική κοινότητα με μερική μόνο γνώση των λειτουργιών των πρωτεϊνών, αφού μόλις μερικές χιλιάδες πρωτεΐνες έχουν μελετηθεί.

Η αλληλουχία των αμινοξέων μίας πρωτεΐνης αποτελεί την πρωτοταγή της δομή. Αυτή η συγκεκριμένη κατά περίπτωση αλληλουχία των αμινοξέων είναι εκείνη που προσδίδει πολλές από τις βασικές ιδιότητες στις διάφορες πρωτεΐνες και καθορίζει σε μεγάλο βαθμό τις δευτεροταγείς και τριτοταγείς δομές τους. Ως ενδιάμεσο βήμα, για την μελέτη της τριτοταγούς δομής, είναι η εκμάθηση της δευτεροταγούς δομής όπου αυτή, με τη σειρά της, κάνει χρήση της πρωτοταγούς δομής για να προβλέψει το σχηματισμό των δομικών στοιχείων (α-έλικες, β-κλώνοι κ.α.) που προκύπτουν από αυτή(Kabsch and Sander, 1983).

Τα τελευταία χρόνια έχουν γίνει αρκετές ευρευνητικές μεθόδοι και προσπάθειες για την επίλυση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, με την καλύτερη μέθοδο να πετυχαίνει ποσοστό επιτυχίας 76%. Αν και σε γενικές γραμμές είναι ένα πολύ καλό ποσοστό, δεν υπάρχει αμφιβολία ότι υπάρχουν αρκετά περιθώρια βελτίωσης.

Η διπλωματική, αυτή, εργασία έχει σκοπό την επέκταση των ερευνητικών δοκιμών στο θέμα αυτό, με τη χρήση των δικτύων Clockwork Recurrent Neural Networks (CW-RNN) (Koutnik et al., 2014).

1.7 Σχετική έρευνα

Στην προσπάθεια επίλυσης του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, έχουν γίνει αρκετές έρευνες με την χρήση διάφορων μεθόδων, οι οποίες κάνουν χρήση της μεθόδου βαθμολόγησης της ακριβείας πρόβλεψης μιας πρωτεΐνης Q3(εξίσωση αξιολόγησης ποσοστών επιτυχίας) για έλεγχο των αποτελεσμάτων τους.

$$Q_3 = \frac{\text{αριθμός καταλείπων που προβλέφθηκαν ορθά}}{\text{συνολικό αριθμός καταλείπων}} * 100$$

Εξίσωση 1.7.1: Εξίσωση Q3 για την μέτρηση του ποσοστού επιτυχίας μιας μεθόδου.(Παυλίδης, 2016)

Οι μέθοδοι αυτές χωρίζονται σε δύο κατηγορίες, στατιστικές και υπολογιστικές μεθόδους. Αρχίζοντας με τις στατιστικές μεθόδους, οι οποίες είναι βασισμένες σε στατιστικούς κανόνες, η Chou-Fasman(Chou and Fasman, 1974), υπολογίζει την πιθανότητα ύπαρξης ενός αμινοξέως στις α-έλικες ή στις β-πτυχωτές βασισμένη στη συχνότητα εμφάνισης του αμινοξέως. Η μέθοδος αυτή κατάφερε να προβλέψει μόλις το 50-60% της δευτεροταγούς δομής.

Αργότερα, η GOR (Garnier et al., 1978) μέθοδος, εξέφρασε την πιθανότητα ένα αμινοξύ να ανήκει σε μια από τις κατηγορίες helix (H), coil (C) και extended (E) κάνοντας χρήση των γειτονικών αμινοξέων. Η αρχική προσέγγιση της GOR μεθόδου έδωσε ποσοστό 65%, ενώ αργότερα, με κάποιες επεκτάσεις, δόθηκε αποτέλεσμα 73,5% .

Τέλος, η μέθοδος Predator(Frishman and Argos, 1996) η οποία σε αντίθεση με την GOR δεν λαμβάνει υπόψη τοπικές αλληλεπιδράσεις αλλά μεγάλα μήκη ακολουθίας, άρα μεγάλες αλληλεπιδράσεις των αμινοξέων, μπορεί να δώσει ποσοστά Q3(τα οποία αντιστοιχούν στα ποσοστά των καταλοίπων που προβλέπονται ορθά) μέχρι και 75%.

Στη κατηγορία των υπολογιστικών μεθόδων, η οποία έχει ραγδαία ανάπτυξη τα τελευταία χρόνια, σε προβλήματα που αφορούν τη αναγνώριση χαρακτηριστικών ανήκουν και τα ΤΝΔ. Αυτή η κατηγορία μεθόδων σε αντίθεση με τις στατιστικές μεθόδους μαθαίνουν την αντιστοιχία της πρωτοταγούς με την δευτεροταγή δομή κατά την διάρκεια που εκπαιδεύονται.

Παραδείγματα τέτοιων δικτύων που υλοποιήθηκαν είναι το PHD(Rost And Sander 1993), μέθοδος η οποία χρησιμοποιεί τεχνικές όπως το γρήγορο σταμάτημα και την τεχνική ensemble average και έχει πετύχει ποσοστό ακριβείας Q3 μέχρι και 71,3%.

Το NNSSP(Salamov and Soloveyev 1995-1997) το οποίο χρησιμοποιεί την μέθοδο του κοντινότερου γείτονα και έχει πετύχει ποσοστά Q3 μέχρι και 73,5%.

Ο αλγόριθμος DSC(King And Stenberg 1996), ο οποίος ομαδοποιεί σε κατηγορίες τα αποτελέσματα εξόδου του δικτύου και προσπαθεί με στατιστικές μεθόδους να

προσεγγίσει την ακριβή δευτεροταγή δομή των πρωτεϊνών, έχει πετύχει ποσοστό ακριβείας Q3 71,95%.

Porter(Pollastri and McLysaght 2004) το οποίο αποτελεί ένα δίκτυο με αρχιτεκτονική αμφίδρομης ανάδρασης, χρησιμοποιεί για εκπαίδευση δεδομένα MSA και φιλτράρει τις προβλεπόμενες ακολουθίες εισόδου με ανάδραση. Πετυχαίνει υψηλότερη ακρίβεια εκπαιδεύωντας πολλά τέτοια δίκτυα αμφίδρομης ανάδρασης και η τελική ακολουθία υπολογίζεται από το μέσο όρο τους. Έχει πετύχει ακρίβεια Q3 79%.

LAD(Blacewicz et al., 2005), κατά τον οποίο λαμβάνονται υπόψη οι ιδιότητες και η δομή των αμινοξέων, με ποσοστό Q3 να φτάνει μέχρι και 70,6%.

Μια πιο πρόσφατη μέθοδο αποτελεί το δίκτυο Chen And Chaudhary(2007) που αποτελεί ένα δίκτυο δύο επιπέδων ΤΝΔ, πετυχαίνωντας έτσι να λαμβάνονται υπόψην οι μακρινές αλληλεπιδράσεις μεταξύ των αμινοξέων, οι οποίες έχουν μεγάλη σημασία στην αναδίπλωση των πρωτεϊνών. Με αυτή την μέθοδο έχει επιτευχθεί ποσοστό Q3 74,38%.

Τέλος, αξίζει να αναφερθούμε στο δίκτυο Νευρωνικό Δίκτυο Αμφίδρομης Ανάδρασης(Bidirectional Recurrent Neural Network-BRNN) (Baldi et al., 1999) το οποίο έχει την καλύτερη απόδοση στην πρόβλεψη δευτεροταγούς δομής των πρωτεϊνών. Το ΤΝΔ αυτό παίρνει ως είσοδο την ακολουθία από αμινοξέα μέσω ενός κινητού παραθύρου και προβλέπει την δομή του κεντρικού αμινοξέως λαμβάνοντας υπόψη τα αμινοξέα που προηγούνται και έπονται αυτού. Η ερευνητική ομάδα του Π.Κ που ασχολείται με το πρόβλημα αυτό, κατάφερε να πετύχει ακρίβεια πρόβλεψης 78% με το δίκτυο BRNN(Baldi et al., 1999).

Κεφάλαιο 2

Νευρωνικά Δίκτυα

2.1 Τεχνητή Νοημοσύνη

Είναι ευρέως αποδεχτό ότι στο σύγχρονο κόσμο η επιστήμη της Πληροφορικής διαδραματίζει ένα πολύ σημαντικό ρόλο στη ζωή του ανθρώπου. Οι εφαρμογές της στη βιομηχανία, στις θετικές επιστήμες, τη φιλοσοφία, ψυχολογία, οικονομικά και σε τόσα άλλα πεδία της ανθρώπινης δραστηριότητας έχουν απαλλάξει τον άνθρωπο από μονότονες και συχνά κουραστικές εργασίες και του έχουν προσφέρει δυνατότητες που παλαιότερα ήταν αδιανόητες. Αποτελεί το «κύριο» εργαλείο στην κάθε εργασία για την επίλυση προβλημάτων, την εξαγωγή συμπερασμάτων και τη λήψη αποφάσεων.(Βικπαίδεια 2017).

Ένα από τα πιο νέα ερευνητικά πεδία στις θετικές επιστήμες είναι η τεχνητή νοημοσύνη που έχει γίνει γνωστή τις τελευταίες δεκαετίες και αποτελεί ένα από τα πλέον «μαθηματικοποιημένα» και ταχέως εξελισσόμενα πεδία της πληροφορικής.

Ο όρος τεχνητή νοημοσύνη (ΤΝ) αναφέρεται στον κλάδο τη πληροφορικής ο οποίος ασχολείται με τη σχεδίαση και την υλοποίηση υπολογιστικών συστημάτων τα οποία μιμούνται στοιχεία της ανθρώπινης συμπεριφοράς και τα οποία υπονοούν εξυπνάδα, μάθηση, προσαρμοστικότητα, εξαγωγή συμπερασμάτων, κατανόηση από τα συμφραζόμενα, επίλυση προβλημάτων κλπ. Πολύ σωστά ο McCarthy την αποκαλεί «επιστήμη και μεθοδολογία της δημιουργίας νοούν των μηχανών».(Βλαχάβας και άλλοι 2011). Σύμφωνα με τους Barr και Feigenbaum , η τεχνητή νοημοσύνη είναι ο τομέας της επιστήμης των υπολογιστών, που ασχολείται με την σχεδίαση ευφυών υπολογιστικών συστημάτων, δηλαδή συστημάτων που επιδεικνύουν χαρακτηριστικά που σχετίζονται με την νοημοσύνη στην ανθρώπινη συμπεριφορά.(Βλαχάβας και άλλοι 2011).

Τα τελευταία χρόνια η απαιτήσεις των εφαρμογών γίνονται όλο και πιο σύνθετες και δύσκολα βρίσκονται μεμονωμένες κατηγορίες μεθόδων ή ερευνητικών αντικειμένων της ΤΝ επειδή αποκτούν την συνεργασία πολλών μεθόδων και αντικειμένων. Μερικές από τις Εφαρμογές της ΤΝ είναι: Έμπειρα συστήματα, Ρομποτική, Επίλυση προβλημάτων, Σχεδιασμός νευρωνικών δικτύων, Ευφυή συστήματα πρακτόρων, Τεχνητή όραση, Ιατρική, Νομική, Εκπαίδευση, Γλωσσολογία, Γεωλογία, Βιολογία, Αστρονομία, κ τλ. Δηλαδή με απλά λόγια η ΤΝ έχει κύριο στόχο να φτιάξει συστήματα που να σκέφτονται

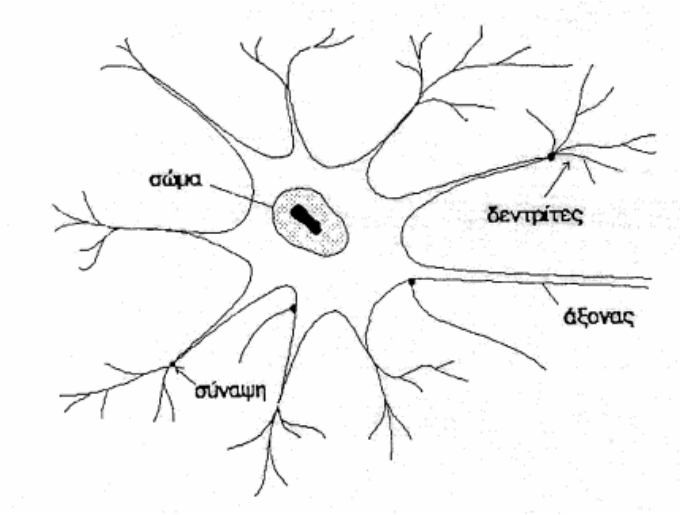
και να συμπεριφέρονται όπως οι άνθρωποι. Ιδού και το κύκριο χαρακτηριστικό τους είναι ότι οι πρώτες αρχές και λειτουργίες τους βασίζονται στο νευρικό σύστημα του ανθρώπου(Βλαχάδας και άλλοι 2011).

2.2 Νευρωνικά Δίκτυα

Τα Νευρωνικά δίκτυα αναπτύχθηκαν μέσα από τις διεξαγωγές ερευνών της Τεχνητής Νοημοσύνης.

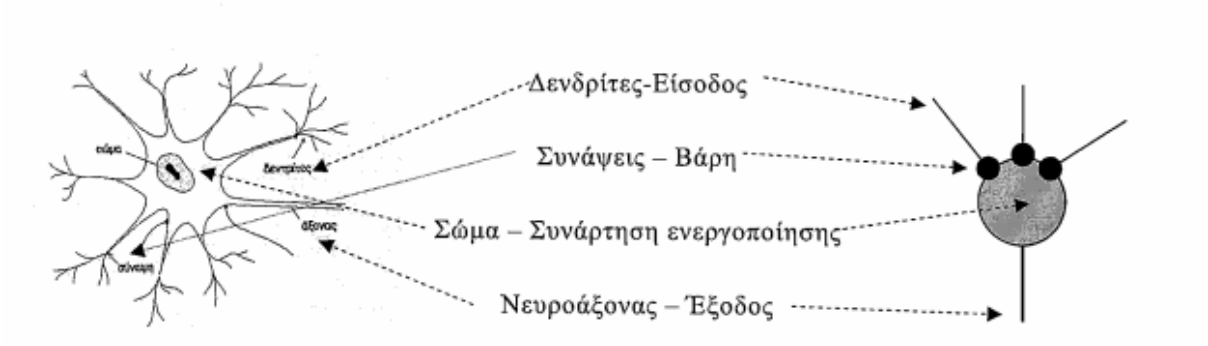
Είναι εμπνευσμένα από το Κεντρικό Νευρικό Σύστημα (ΚΝΣ), το οποίο προσπαθούν να προσομοιώσουν. Ουσιαστικά, αποτελούν απλοποιημένα μοντέλα του κεντρικού νευρικού συστήματος του ανθρώπου, τα οποία προσπαθούν να βελτιστοποιήσουν μια πολύπλοκη μαθηματική συνάρτηση. Αποτελούνται από διασυνδεδεμένα υπολογιστικά στοιχεία που έχουν την ικανότητα να ανταποκρίνονται σε ερεθίσματα που δέχονται στην είσοδό τους και να μαθαίνουν να προσαρμόζονται στο περιβάλλον τους. (Ρεφανίδης 2011).

Ο ανθρώπινος εγκέφαλος, αποτελεί το κλασσικό παράδειγμα Βιολογικού Νευρωνικού Δικτύου, όπου οι κόμβοι είναι στην πραγματικότητα τα νευρικά κύτταρα («νευρώνες»). Στην περίπτωση βιολογικών νευρώνων, πρόκειται για ένα τμήμα νευρικού ιστού όπου κάθε νευρώνας αποτελείται από τέσσερα βασικά τμήματα που είναι οι δενδρίτες, το σώμα, ο άξονας και οι διάφορες νευρικές απολήξεις.



Σχήμα 2.2.1: Αναπαράσταση ενός βιολογικού νευρώνα(https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html)

Ο κάθε νευρώνας έχει πολλούς δενδρίτες με πολλές διακλαδώσεις. Στην άκρη των δενδριτών βρίσκονται οι συνάψεις και από εκεί το κύτταρο λαμβάνει ή μεταδίδει τα διάφορα σήματα. Οι άξονες ενός κυττάρου συνδέονται με τους δενδρίτες ενός άλλου, μέσω μιας σύναψης. Κάθε νευρώνας έχει δύο δυνατές καταστάσεις στις οποίες μπορεί να βρίσκεται και τις ονομάζουμε ενεργό και μη-ενεργό κατάσταση. Όταν ο νευρώνας είναι ενεργός λέμε ότι πυροδοτεί, ενώ όταν είναι μη-ενεργός λέγουμε ότι είναι αδρανής. Σε πλήρη αντιστοιχία με το απλοποιημένο αυτό μοντέλο του βιολογικού νευρώνα αναπτύχθηκε το μοντέλο του τεχνητού νευρώνα.



Σχήμα 2.2.2 : Στα αριστερά αναπαριστάται το μοντέλο του βιολογικού νευρώνα και στα δεξιά το μοντέλο του τεχνητού νευρώνα. (http://repfiles.kallipos.gr/html_books/93/04a-

main.html)

Έτσι, ένας τεχνητός νευρώνας d με συνδέσεις εισόδου x1,…,xd με αντίστοιχες τιμές βαρών w1, …, wd, για τον υπολογισμό της εξόδου y του νευρώνα εκτελούνται δύο βήματα. Το πρώτο βήμα είναι ο υπολογισμός της ενεργοποίησης u όπως φαίνεται στο σχήμα, όπου θ η πόλωση του νευρώνα

$$u = \sum_{i=1}^{d} w_i \cdot x_i + \theta$$

Εξίσωση 2.2.1: Συνάρτηση υπολογισμού ενεργοποίησης

και το τελευταίο βήμα είναι ο υπολογισμός της εξόδου του νευρώνα περνώντας την ενεργοποίηση u μέσα από μια συνάρτηση ενεργοποίησης f: y=f(u). Η συνάρτηση f μπορεί να είναι γραμμική, για παράδειγμα f(u)=u, ή μη γραμμική, για παράγειγμα συνάρτηση καμπάνας. Πιο συνηθησμένα παραδείγματα συναρτήσεων αποτελούν οι βηματική και η σιγμοειδής συνάρτηση. Έτσι κάθε νευρώνας δέχεται πολλά σήματα ως είσοδο και μετά την επεξεργασία τους διαδίδει μόνο ένα σε όλους τους νευρώνες με τους οποίους συνδέεται (Golden,1996).

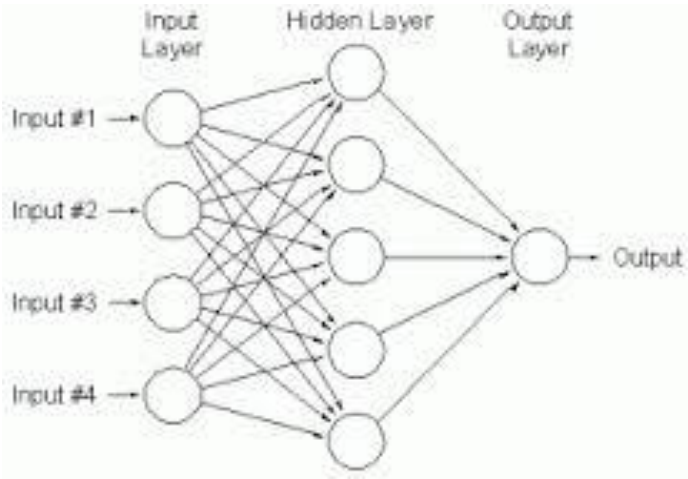
$$\psi = \begin{cases} 1 & \text{εάν } u \geq \theta \\ 0 & \text{εάν } u < \theta \end{cases}$$

Εξίσωση 2.2.2 : Εξίσωση βηματικής συνάρτησης

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

Εξίσωση 2.2.3 : Εξίσωση σιγμοειδούς συνάρτησης

Ένα νευρωνικό δίκτυο είναι μια μηχανή η οποία έχει σχεδιαστεί να δουλέψει με τον ίδιο τρόπο με τον όποιο ο εγκέφαλος εκτελεί διαφορές λειτουργίες. Στο τεχνητό νευρωνικό δίκτυο (ΤΝΔ), οι νευρώνες αποτελούν τα δομικά στοιχειά του δικτυού. Οι τεχνητοί νευρώνες είναι πυκνά διασυνδεδεμένοι μεταξύ τους και οργανωμένοι σε επάλληλα επίπεδα. Συνήθως υπάρχει ένα επίπεδο εισόδου όπου εισάγεται το διάνυσμα εκπαίδευσης ή το διάνυσμα έλεγχου, ένα η περισσότερα κρυφά επίπεδα όπου γίνεται η μη γραμμική επεξεργασία των πληροφοριών και, τέλος, ένα επίπεδο εξόδου που μεταφέρει τα αποτελέσματα στον έξω κόσμο. Γι΄αυτό υπάρχουν δυο είδη νευρώνων, οι νευρώνες εισόδου και οι υπολογιστικοί νευρώνες. Ο αριθμός των τεχνητών νευρώνων μπορεί να κυμαίνεται από μερικές μονάδες έως μερικές χιλιάδες. Ένα τυπικό παράδειγμα νευρωνικού δικτυού φαίνεται στο Σχήμα 2.2.3.



Σχήμα 2.2.3 : Παράδειγμα τυπικού νευρωνικού δικτύου(https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm)

Ένα νευρωνικό δίκτυο είναι ένας συμπαγής παράλληλος κατανεμημένος επεξεργαστής, που έχει τη φυσική κλίση να αποθηκεύει εμπεριστατωμένη γνώση και να την κάνει

διαθέσιμη για χρήση. Στα ΤΝΔ όπως και στον εγκέφαλο η γνώση αποκτάται από το δίκτυο μέσα από μια διαδικασία μάθησης και οι δυνάμεις σύνδεσεις των νευρώνων, γνωστές σαν συναπτικά βάρη, χρησιμοποιούνται για την αποθήκευση γνώσης.

Η διαδικασία για την εκμάθηση ονομάζεται "αλγόριθμος μάθησης". Για να κατασκευάσουμε ένα νευρωνικό δίκτυο, πρέπει πρώτα να ορίσουμε την τοπολογία του, δηλαδή τον αριθμό των νευρώνων που θα το απαρτίζουν και τη δομή τους. Στη συνέχεια, πρέπει να υπολογίσουμε τους συντελεστές βαρύτητας των συνάψεων (ή τη μνήμη του νευρωνικού δικτύου). Κάθε σήμα που μεταδίδεται από ένα νευρώνα σε ένα άλλο μέσα στον νευρωνικό δίκτυο συνδέεται με την τιμή βάρους, w, και η οποία υποδηλώνει πόσο στενά είναι συνδεδεμένοι οι δύο νευρώνες που συνδέονται με το βάρος αυτό. Η διαδικασία μάθησης υπολογίζει τους συντελεστές αυτούς, συνήθως από ένα σύνολο παραδειγμάτων. Μπορούμε να πούμε ότι με τη διαδικασία της μάθησης προσπαθούμε να ανακαλύψουμε την σωστή συνάρτηση μεταξύ των διανυσμάτων εισόδου και εξόδου.

2.3 Εκπαίδευση Νευρωνικών Δικτύων

Υπάρχουν τρία είδη μάθησης, η επιβλεπόμενη μάθηση, η μη επιβλεπόμενη μάθηση και η ενισχυτική μάθηση.

2.3.1 Επιβλεπόμενη Μάθηση

Επιβλεπόμενη Μάθηση (Supervised Learning) είναι η διαδικασία *όπου ο* αλγόριθμος κατασκευάζει μια συνάρτηση που απεικονίζει δεδομένες εισόδους (σύνολο εκπαίδευσης) σε γνωστές επιθυμητές εξόδους, με απώτερο στόχο τη γενίκευση της συνάρτησης αυτής και για εισόδους με άγνωστη έξοδο. Κατά την διαδικασία αυτή τα συναπτικά βάρη τροποποιούνται έτσι ώστε να ελαχιστοποιηθεί η διαφορά μεταξύ της επιθυμητής απόκρισης και της πραγματικής απόκρισης του δικτύου. Τότε το δίκτυο φτάνει σε μια ευσταθή κατάσταση όπου δεν υπάρχουν άλλες αλλαγές βαρών. Έτσι το δίκτυο αποκτά την ικανότητα μάθησης μέσω παραδειγμάτων, οργανώνωντας την πληροφορία των δεδομένων εισόδου σε χρήσιμες μορφές. Αυτές οι μορφές αποτελούν στην ουσία ένα μοντέλο που αναπαριστά τη σχέση που ισχύει μεταξύ των δεδομένων εισόδου και εξόδου.

2.3.2 Μη Επιβλεπόμενη Μάθηση

Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning), το σύστημα χωρίς να γνωρίζει τις επιθυμητές εξόδους εντοπίζει τα κοινά χαρακτηριστηκά των δεδομένων εισόδου και προσδιορίζει τις διάφορες ομάδες κατηγοροποιώντας τα δεδομένα σε διάφορα υποσύνολα.

2.3.3 Ενισχυτική Μάθηση

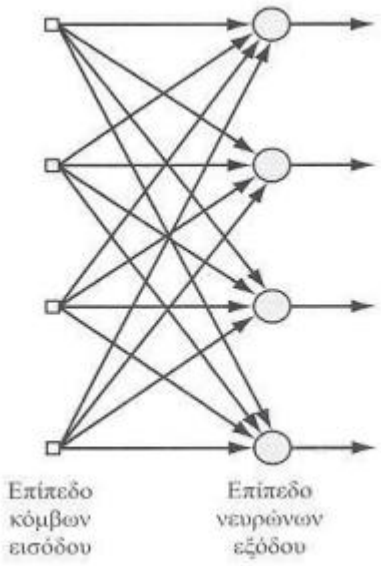
Ενισχυτική Μάθηση (Reinforcement Learning), όπου ο αλγόριθμος μαθαίνει μια στρατηγική ενεργειών μέσα από άμεση αλληλεπίδραση με το περιβάλλον. Το περιβάλλον λειτουργεί σαν κριτής για το σύστημα το οποίο δεν προσδιορίζει την σωστή απάντηση αλλά κρίνει τις εξόδους του δικτύου μέσω μιας επιβράβευσης ή μιας τιμωρίας. Με αυτό το τρόπο το δίκτυο αποκτά την ικανότητα να εκπαιδεύεται μέσω της προσπαθειάς του να παίρνει επιβράβευση και να μην παίρνει τιμωρία.

2.4 Δομή Νευρωνικών Δικτύων

Παρόλο που δεν υπάρχει περιορισμός για τον τρόπο με τον όποιο οργανώνονται οι συνάψεις των νευρώνων, υπάρχουν μερικές ομάδες δομών οι οποίες έχουν μελετηθεί εκτενεστέρα και είναι αυτές που χρησιμοποιούνται συχνότερα σε διαφορές εφαρμογές.

2.4.1 Νευρωνικά Δίκτυα ενός επιπέδου

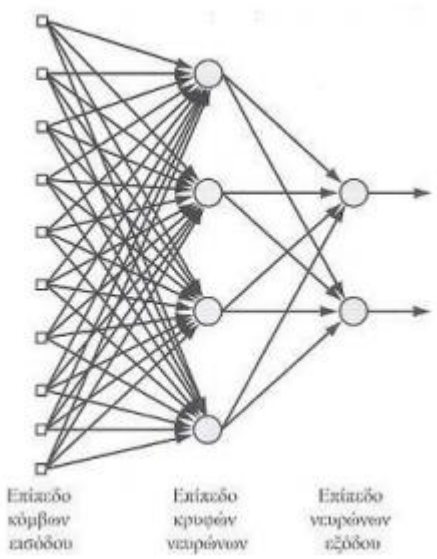
Αποτελεί την πλέον απλή περίπτωση οργάνωσης ενός νευρωνικού δικτύου. Οι είσοδοι κάθε νευρώνα συνδέονται με τις αντίστοιχες εισόδους του δικτυού και η έξοδος κάθε νευρώνα αποτελεί και έξοδο δικτύου.Οι νευρώνες εισόδου απλά μεταφέρουν το σήμα στο επίπεδο εξόδου χωρίς να κάνουν καμία επεξεργασία. Ο χαρακτηρισμός ενός επιπέδου αναφέρεται στο επίπεδο εξόδου.



Σχήμα 2.4.1.1: Δίκτυο πρόσθιας τροφοδότησης ενός επιπέδου(https://www.slideshare.net/MohammedBennamoun/artificial-neural-network-lect4-single-layer-perceptron-classifiers)

2.4.2 Πολυεπίπεδα Νευρωνικά Δίκτυα

Προσθέτοντας ένα ή περισσότερα κρυφά επίπεδα, των οποίων οι νευρώνες ονομάζονται κρυφοί νευρώνες, μπορούμε να αυξήσουμε απεριόριστα τους βαθμούς ελευθερίας του νευρωνικού δικτυού. Συνήθης τακτική είναι οι νευρώνες κάθε επιπέδου να συνδέονται με τις εξόδους των νευρώνων που βρίσκονται στο προηγούμενο επίπεδο. Το σήμα ρέει από το επίπεδο εισόδου προς το επίπεδο εξόδου μέσω των κρυφών επιπέδων. Άρα η είσοδος των νευρώνων για κάθε επίπεδο εξαρτάται από την έξοδο των νευρώνων στο προηγούμενο επίπεδο. Έτσι δημιουργούμε τα πολυεπίπεδα νευρωνικά δίκτυα. Ένα τέτοιο δίκτυο με ένα κρυφό επίπεδο φαίνεται στο Σχήμα 2.4.2.1.



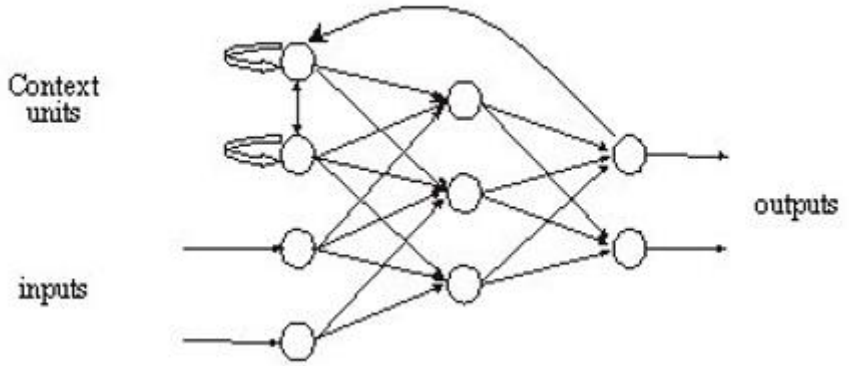
Σχήμα 2.4.2.1 : Δίκτυο πρόσθιας τροφοδότησης με ένα κρυφό επίπεδο.(https://www.researchgate.net/figure/230735806_fig1_Schematic-drawing-of-multilayer-perceptron-neural-networks-16)

2.4.3 Νευρωνικά Δίκτυα Ανάδρασης

Στο υποκεφάλαιο αυτό θα παρουσιάσουμε ένα άλλο είδος νευρωνικού δικτύου, τα Νευρωνικά Δίκτυα Ανάδρασης (Recurrent Neural Networks - RNN). Ένα Αναδραστικό Τεχνητό Νευρωνικό Δίκτυο (ΑΤΝΔ) διαφέρει από ένα Feedforward δίκτυο στο γεγονός ότι περιέχει έναν τουλάχιστον βρόγχο ανατροφοδότησης. Αυτό σημαίνει ότι σε έναν τουλάχιστον νευρώνα, το σήμα εξόδου του επηρεάζει το σήμα που έρχεται στην είσοδο του νευρώνα. Σε ορισμένες περιπτώσεις, τα δίκτυα αυτά περιέχουν μία κρυφή βαθμίδα, οι κόμβοι της οποίας είναι διασυνδεδεμένοι μεταξύ τους. Έχουν τη δυνατότητα να λειτουργούν σε πολλαπλά βήματα, όπου σε κάθε βήμα κάθε κρυφός κόμβος λαμβάνει σαν είσοδο, μαζί με το διάνυσμα εισόδου και τις τιμές των υπόλοιπων κόμβων της κρυφής βαθμίδας. Στα δίκτυα αυτά γίνεται μοντελοποίηση της αίσθησης του χρόνου. Αυτή η αρχιτεκτονική προσδίδει στο δίκτυο πολύ περισσότερες δυνατότητες, αλλά είναι πιο δύσκολο να αντιμετωπισθεί μαθηματικά.

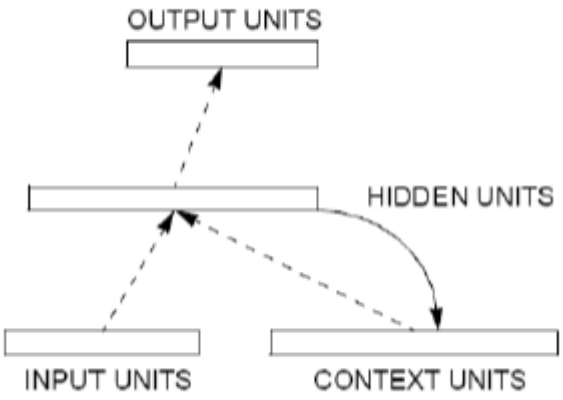
Παράδειγμα τέτοιου δικτύου αποτελεί το δίκτυο Jordan(Jordan, 1986) το οποίο φαίνεται στο Σχήμα 2.4.3.1. Η μνήμη μοντελοποιείται με την εφαρμογή των context units, στα

οποία αποθηκεύεται η πληροφορία εξόδου της προηγούμενης χρονικής στιγμής και έχουν μέγεθος όσο ο αριθμός των εξόδων του δικτύου. Τα βάρη στις αναδράσεις είναι σταθερά για να μην αλλιώνεται η πληροφορία στο δίκτυο. Στα context units η πληροφορία πολλαπλασιάζεται με κάποια σταθερά, μέσω τοπικών αναδράσεων, η οποία όσο πιο μεγάλη είναι τόσο πιο πολύ τα context units κρατούν την πληροφορία



Σχήμα 2.4.3.1 : Δίκτυο Jordan με ανάδραση.(Jordan,1986)

Ένα άλλο παράδειγμα δικτύου με ανάδραση αποτελεί το δίκτυο του Elman(Elman,1990). Σε αυτή την αρχιτεκτονική η ανάδραση ξεκινά από το κρυφό επίπεδο και πηγαίνει στο context layer, με αυτό το τρόπο η ανάδραση γίνεται εντός του δικτύου και οι νευρώνες εξόδου είναι ελεύθεροι.



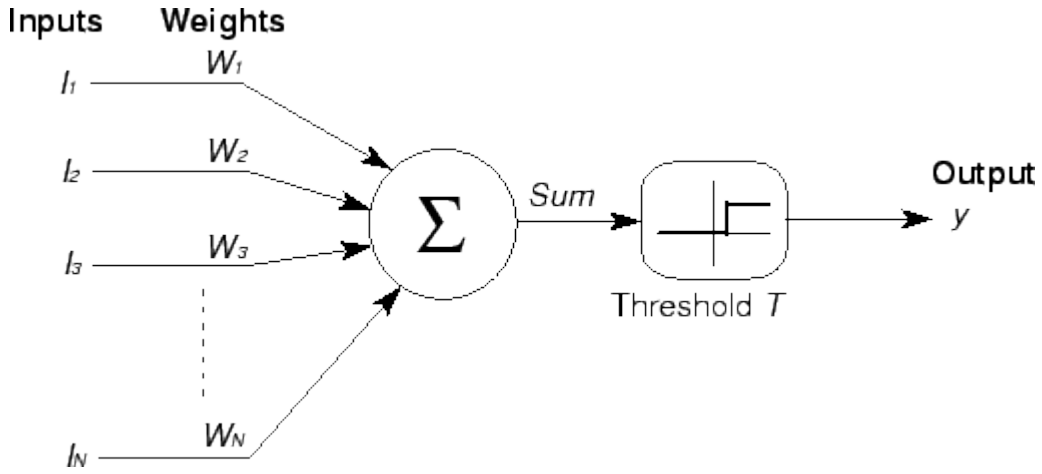
Σχημα 2.4.3.2 : Δίκτυο με ανάδραση Elman(Elman,1990)

2.5 Ιστορικό Υπόβαθρο

Τα Νευρωνικά Δίκτυα είναι σχετικά μια νέα περιοχή και δεν υπάρχει ουσιαστικά μεγάλη προϊστορία, όπως σε άλλες επιστήμες. Έκαναν την εμφάνιση τους τις τελευταίες δεκαετίες αλλά η μεγάλη ώθηση δόθηκε μετά το 1980.

Το 1943 δημιουργήθηκε το πρώτο μοντέλο νευρωνικού δικτύου από τους McCulloch και Pitts. Στην ουσία είχαν παρουσιάσει την ιδέα ότι ένα Νευρωνικό Δίκτυο αποτελείται από ένα σύνολο ενός μεγάλου αριθμού νευρώνων και έδειξαν πώς θα μπορούσαν να λειτουργούν οι νευρώνες με τις διασυνδέσεις τους.

Το 1947, οι ίδιοι προχώρησαν σε ένα πιο εξελιγμένο πρότυπο, κατά το οποίο ο νευρώνας μπορεί να βρίσκεται σε μια εκ των δύο καταστάσεων, να πυροδοτεί ή να βρίσκεται σε ηρεμία. Δεν υπάρχει ποτέ ένωση των εξόδων από δύο διαφορετικούς νευρώνες, πρέπει υποχρεωτικά να οδηγούν σε είσοδο άλλου νευρώνα. Ένας νευρώνας έχει πολλές εισόδους αλλά μια έξοδο. Τα δίκτυα αυτά προσπαθούν να εξηγήσουν πως δουλεύει η μνήμη. Θεωρούν ότι ένας πιθανός μηχανισμός μνήμης μπορεί να είναι η ύπαρξη κλειστών διαδρομών του σήματος μέσα στο δίκτυο. Αν υπάρχει σύνδεση μεταξύ της εξόδου και της εισόδου σε ένα κύτταρο, έχουμε μηχανισμό ανάδρασης.



Σχήμα 2.5.1 : Ένα τυπικό μοντέλο τεχνητού νευρώνα McCulloch and Pitts(http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node12.html)

Το 1957, παρουσιάστηκε από τον F. Rosenblatt το μοντέλο του αισθητήρα (perceptron). Είναι ένα πολύ απλό μοντέλο με δύο επίπεδα, της εισόδου και της εξόδου, όπου το σήμα προχωρά μονοδρομικά από την είσοδο στην έξοδο.

Λίγα χρόνια αργότερα, το 1959 οι Widrow και Hoff ανέπτυξαν δύο νέα μοντέλα το Adaline και το Madaline. Αυτά τα δύο μοντέλα χρησιμοποιήθηκαν ως φίλτρα για να εξαλείψουν την ηχώ σε τηλεφωνικές γραμμές.

Το 1982 ο Hopfield έδειξε με μαθηματική απόδειξη πώς ένα νευρωνικό δίκτυο μπορεί να χρησιμοποιηθεί ως αποθηκευτικός χώρος (storage device) και πώς μπορεί να επανακτήσει όλη την πληροφορία ενός συστήματος αν του δοθούν μερικά τμήματα μόνο και όχι ολόκληρο το σύστημα.

Σημαντικό ρόλο διαδραμάτισε η πρόοδος στην διαδικασία εκπαίδευσης των δικτύων όταν επινοήθηκε ο κανόνας της διόρθωσης του σφάλματος. Κατά την εκπαίδευση ενός δικτύου μεγάλη σημασία έχει η απόκλιση που δίνει στην έξοδό του το δίκτυο από την αναμενόμενη τιμή. Η απόκλιση αυτή αποτελεί το σφάλμα του δικτύου και ενεργοποιεί ένα μηχανισμό ελέγχου ώστε να επιφέρει μια σειρά από διορθωτικές αλλαγές στα βάρη των νευρώνων.

Τέλος το 1986 οι McClelland και Rumelhart, στο «Parallel Distributed Processing» προτείνουν μία νέα διαδικασία εκπαίδευσης, την μέθοδο της οπισθοδιάδοσης (back–propagation), η οποία είναι σήμερα η πιο χρήσιμη τεχνική εκπαίδευσης δικτύων.

Τα τελευταία χρόνια έχει υπάρξει μεγάλο ενδιαφέρον για τα νευρωνικά δίκτυα καθώς εφαρμόζονται με μεγάλη επιτυχία σε ένα ασυνήθιστα μεγάλο φάσμα στον τομέα της επιστήμης και της τεχνολογίας. Πράγματι είναι προφανές ότι τα νευρωνικά δίκτυα εισάγονται οπουδήποτε τίθεται θέμα πρόβλεψης, ταξινόμησης ή ελέγχου. Αυτή η επιτυχία, μπορεί να αποδοθεί στα δύο βασικά τους στοιχεία: την ισχύ και την ευχρηστία. Πολύ σωστά χαρακτηρίζονται σαν ένα χρήσιμο υπολογιστικό εργαλείο, με πολλαπλές εφαρμογές.

Κεφάλαιο 3

Δεδομένα δικτύου

3.1 Δεδομένα Εισόδου

Πολύ σημαντικό βήμα κατά την εκπαίδευση του Νευρωνικού δικτύου αποτελεί η ορθή επιλογή των δεδομένων εκπαίδευσης. Στην περίπτωση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών τα δεδομένα εισόδου του δικτύου μας είναι η πρωτοταγής δομή τους. Όπως προαναφέρθηκε η πρωτοταγής δομή των πρωτεϊνών ορίζεται από την αναπαράσταση της αλληλουχίας των αμινοξέων από τα οποία αποτελείται η κάθε πρωτεΐνη. Δεδομένου ότι η εκπαίδευση του δικτύου γίνεται με επιβλεπόμενη μάθηση, τα δεδομένα εκπαίδευσης του δικτύου είναι πρωτεϊνες των οποίων είναι γνωστή τόσο η πρωτοταγής όσο και η δευτεροταγής δομή τους. Ακολούθως, τα δεδομένα αυτά πρέπει να κωδικοποιηθούν ώστε να μπορούν να δοθούν σαν είσοδος στο νευρωνικό δίκτυο. Δεδομένου ότι απαιτείται η όσο το δυνατό καλύτερη αναπαράσταση των δεδομένων αυτών από τα πραγματικά δεδομένα θα γίνει χρήση της τεχνικής της πολλαπλής ευθυγραμισμένης ακολουθίας πρωτεϊνών (MSA- Multiple Sequence Alignment). Με τη χρήση αυτής της μεθόδου επιτυγχάνεται η προσθήκη περισσότερης πληροφορίας στο δίκτυο.

Τα Data Set και η κωδικοποίηση τους τα οποία χρησιμοποιούνται για την εκπαίδευση και επαλήθευση του δικτύου είναι αυτά που έχει προτείνει η Ειρήνη Παπακώστα στη δική της διπλωματική εργασία, η οποία ολοκληρώθηκε το 2016 και είχε τον ίδιο στόχο με την δική μου, χρησιμοποιώντας μια διαφορετική μέθοδο. Στα δεδομένα αυτά έγιναν κάποιες βελτιστοποιήσεις. Οι ευθυγραμμισμένες πρωτεΐνες τις οποίες χρησιμοποίησε η Παπακώστα ως είσοδο στο δίκτυο της, διαχωρίστηκαν με 0 όσα και το μέγεθος του κινητού παραθύρου στο χρόνο, ούτως ώστε να αποφευχθεί η συσχέτιση των αμινοξέων μιας πρωτεΐνης με τα αμινοξέα μιας άλλης(της προηγούμενης) πρωτεΐνης κάνοντας χρήση της τεχνικής window. Οι εξαρτήσεις μεταξύ των αμινοξέων υπάρχουν μόνο μεταξύ των αμινοξέων που ανήκουν στην ίδια πρωτεΐνη.

3.1.1 Τεχνική MSA

Η τεχνική MSA αποτελεί μια από τις πιο διαδεδομένες τεχνικές στην βιοπληροφορική για την απεικόνιση των σχέσεων μεταξύ των αμινοξέων σε ένα σύνολο πρωτεϊνών. Σύμφωνα με την τεχνική αυτή, οι διάφορες αλληλουχίες των αμινοξέων σε ένα σύνολο

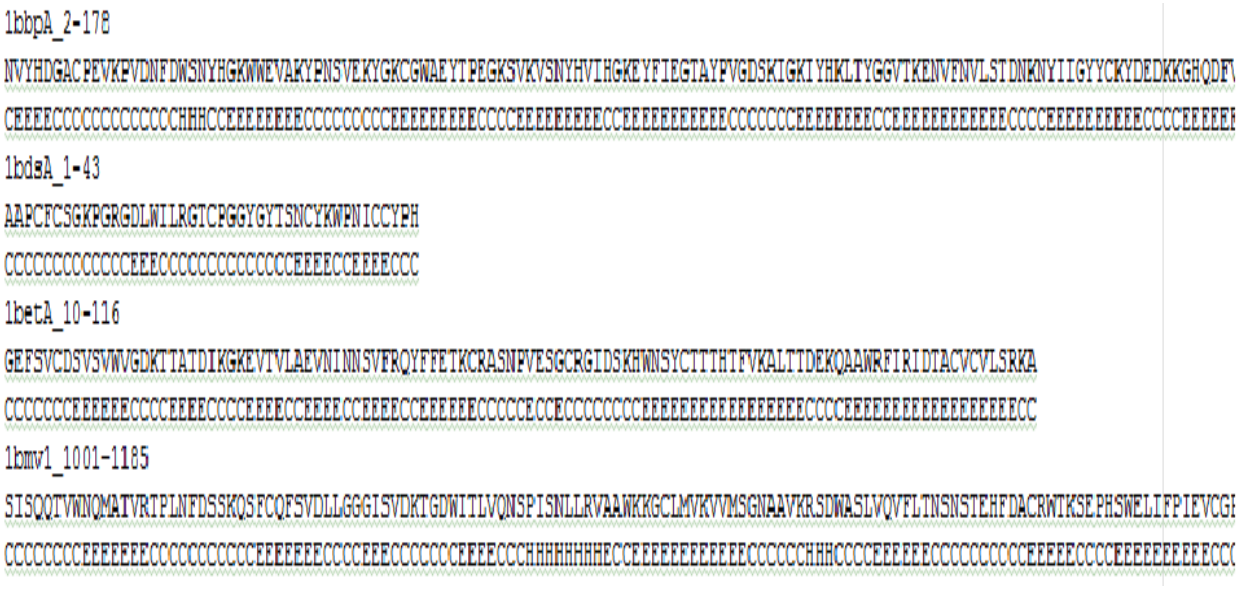
πρωτεϊνών συγκρίνονται και παράγουν μια ευθυγράμμιση που εμφανίζει τα κατάλοιπα αμινοξέος για κάθε πρωτεΐνη σε μια ευθεία. Κάθε στήλη της ευθείας απεικονίζει ενα ξεχωριστό κατάλοιπο έτσι ώστε, τα κατάλοιπα που έχουν κοινή εξελικτική καταγωγή για την δομική βιολογία, ή που αντιστοιχούν σε ανάλογες θέσεις σε ομοιογενείς πτυχώσεις σε ένα σύνολο πτωτεϊνών για την μοριακή βιολογία και διαδραματίζουν τον ίδιο ρόλο αντίστοιχα στις πρωτεϊνες που ανήκουν, να εμφανίζονται στην ίδια στήλη. Με αυτό τον τρόπο επιτυγχάνεται η γενική αναπαράσταση των αμινοξέων που αποτελούν μια πρωτεϊνη.(Chuong B. and Katoh K. ,2016). Ένα παράδειγμα μιας MSA αναπαράστασης φαίνεται στο Σχήμα 3.1.1.1.

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

Σχήμα 3.1.1.1 (Χριστοδούλου, 2010): Αναπαράσταση MSA τεχνικής. Στην πρώτη σειρά αναγράφονται τα 20 αμινοξέα τα οποία αποτελούν τις πρωτεΐνες. Στην πρώτη στήλη αναγράφονται τα αμινοξέα της πρωτεΐνης (πρωτοταγής δομή της). Στην κάθε γραμμή αναπαριστάται η πιθανότητα του κάθε αμινοξέως να εμφανιστεί στην αντίστοιχη θέση της πρωτεΐνης. Για παράδειγμα, σύμφωνα με την Τρίτη γραμμή, στην θέση C του αμινοξέως υπάρχει πιθανότητα 100% να εμφανιστεί το αμινοξύ C.

3.1.2 Δομή Δεδομένων εισόδου

Τα αρχεία που περιέχουν τα δεδομένα που θα χρησιμοποιηθούν σαν είσοδος στο δίκτυο είναι τα αρχεία που χρησιμοποιήθηκαν από την Ειρήνη Παπακώστα κατά την υλοποίηση της διπλωματικής της εργασία. Το αρχείο msaProteinsTrainBigDtaset_afterProcess corr.txt περιέχει το σύνολο με τις πρωτεΐνες οι οποίες θα χρησιμοποιηθούν για εκπαίδευση του δικτύου. Η δομή των πρωτεϊνών μέσα στο αρχείο φαίνεται στο Σχήμα 3.1.2.1.



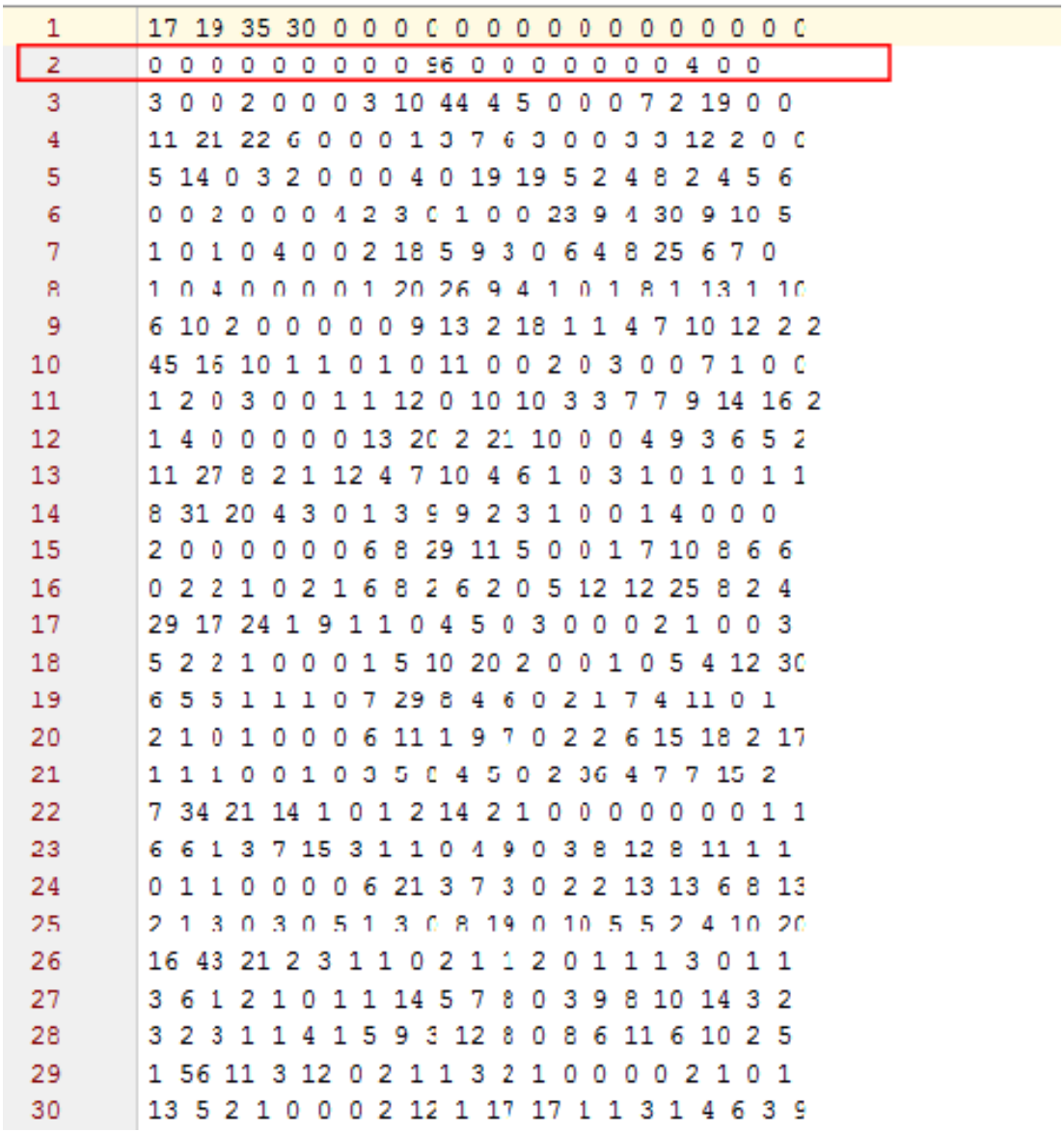
Σχήμα 3.1.2.1: Αναπαράσταση των πρωτεϊνών στο αρχείο msaProteinsTrainBigDtaset_afterProcess corr.txt

Κάθε πρωτεΐνη αναπαριστάται σε τρεις γραμμές. Η πρωτή γραμμή δείχνει το όνομα την πρωτεΐνης, η δεύτερη γραμμή την πρωτοταγή δομή της και η Τρίτη γραμμή την δευτεροταγή δομή της. Για παράδειγμα, στο Σχήμα 3.1.2.1 η πρώτη γραμμή μας δίνει το όνομα της πρωτεΐνης 1bbpA_2-178, η δεύτερη γραμμή την αλληλουχία των αμινοξέων που την αποτελούν(πρωτοταγής δομή) NVYHDGA… και η Τρίτη γραμμή αποτελείται από την δευτεροταγή δομή της πρωτεΐνης CEEEECCCC… Στην επόμενη γραμμή αρχίζει η αναπαράσταση μιας άλλης πρωτεΐνης.

Η δεύτερη γραμμή για κάθε πρωτεΐνη θα αποτελέσει την είσοδο του δικτύου ενώ η Τρίτη γραμμή θα αποτελέσει την επιθυμιτή έξοδο του δικτύου. Κάθε πρωτεΐνη έχει ένα αρχείο με το όνομα της στο οποίο υπάρχουν τα κατάλοιπα από τα 20 αμινοξέα που θα αντιστοιχηθούν για να παράξουν την είσοδο του δικτύου για την συγκεκριμένη πρωτεΐνη. Έτσι, έχοντας το όνομα της πρωτεΐνης από το αρχείο

msaProteinsTrainBigDtaset_afterProcess corr.txt (πρώτη γραμμή για κάθε πρωτεΐνη) εντοπίζει το αρχείο με την MSA αναπαράσταση των αμινοξέων για την πρωτεΐνη αυτή. Για παράδειγμα απο το σχήμα για την πρώτη πρωτεΐνη που αναπαριστάται με όνομα 1bbpA_2-178 το αντίστοιχο αρχείο με την MSA αντιστοίχηση της είναι το 1bbpA_2-178.hssp.

Ένα παράδειγμα τέτοιου αρχείου φαίνεται στο Σχήμα 3.1.2.2



1	17 19 35 30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 0 0 96 0 0 0 0 0 0 0 4 0 0
3	3 0 0 2 0 0 0 3 10 44 4 5 0 0 0 7 2 19 0 0
4	11 21 22 6 0 0 0 1 3 7 6 3 0 0 3 3 12 2 0 0
5	5 14 0 3 2 0 0 0 4 0 19 19 5 2 4 8 2 4 5 6
6	0 0 2 0 0 0 4 2 3 0 1 0 0 23 9 4 30 9 10 5
7	1 0 1 0 4 0 0 2 18 5 9 3 0 6 4 8 25 6 7 0
8	1 0 4 0 0 0 0 1 20 26 9 4 1 0 1 8 1 13 1 10
9	6 10 2 0 0 0 0 0 9 13 2 18 1 1 4 7 10 12 2 2
10	45 16 10 1 1 0 1 0 11 0 0 2 0 3 0 0 7 1 0 0
11	1 2 0 3 0 0 1 1 12 0 10 10 3 3 7 7 9 14 16 2
12	1 4 0 0 0 0 0 0 13 20 2 21 10 0 0 4 9 3 6 5 2
13	11 27 8 2 1 12 4 7 10 4 6 1 0 3 1 0 1 0 1 1
14	8 31 20 4 3 0 1 3 9 9 2 3 1 0 0 1 4 0 0 0
15	2 0 0 0 0 0 0 6 8 29 11 5 0 0 1 7 10 8 6 6
16	0 2 2 1 0 2 1 6 8 2 6 2 0 5 12 12 25 8 2 4
17	29 17 24 1 9 1 1 0 4 5 0 3 0 0 0 2 1 0 0 3
18	5 2 2 1 0 0 0 1 5 10 20 2 0 0 1 0 5 4 12 30
19	6 5 5 1 1 1 0 7 29 8 4 6 0 2 1 7 4 11 0 1
20	2 1 0 1 0 0 0 6 11 1 9 7 0 2 2 6 15 18 2 17
21	1 1 1 0 0 1 0 3 5 0 4 5 0 2 36 4 7 7 15 2
22	7 34 21 14 1 0 1 2 14 2 1 0 0 0 0 0 0 1 1
23	6 6 1 3 7 15 3 1 1 0 4 9 0 3 8 12 8 11 1 1
24	0 1 1 0 0 0 0 6 21 3 7 3 0 2 2 13 13 6 8 13
25	2 1 3 0 3 0 5 1 3 0 8 19 0 10 5 5 2 4 10 20
26	16 43 21 2 3 1 1 0 2 1 1 2 0 1 1 1 3 0 1 1
27	3 6 1 2 1 0 1 1 14 5 7 8 0 3 9 8 10 14 3 2
28	3 2 3 1 1 4 1 5 9 3 12 8 0 8 6 11 6 10 2 5
29	1 56 11 3 12 0 2 1 1 3 2 1 0 0 0 0 2 1 0 1
30	13 5 2 1 0 0 0 2 12 1 17 17 1 1 3 1 4 6 3 9

Σχήμα 3.1.2.2 :(Ειρήνη Παπακώστα,2016): MSA αναπαράσταση της πρωτοταγούς δομής μιας πρωτεΐνης όπως φαίνεται στο MSA αρχείο της.

Η κωδικοποίηση της θέσης του αμινοξέως δείχνει την πιθανότητα εμφάνισης των 20 αμινοξέων σε εκείνη την θέση συμπεριλαμβανομένου και του αμινοξέως αυτού. Αυτό επαναλαμβάνεται για κάθε θέση της ακολουθίας των αμινοξέων που αποτελούν την κάθε πρωτεΐνη η οποία αναπαρίσταται στο αρχείο MSA. Για την πρωτεϊνη που αναπαρίσταται η πρωτοταγής δομή της με την τεχνική MSA στο σχήμα, έχουμε την πληροφορία ότι στην δεύτερη θέση της αλληλουχίας των αμινοξέων που την αποτελούν βρίσκεται κατά 96% το αμινοξύ που αναπαριστάται στην στήλη 10 και κατά 4% το αμινοξύ που αναπαρίσταται στην στήλη 18. Κάθε στήλη αντιπροσωπεύει ένα από τα 20 αμινοξέα από τα οποία αποτελείται η πρωτοταγής δομή των πρωτεϊνών.

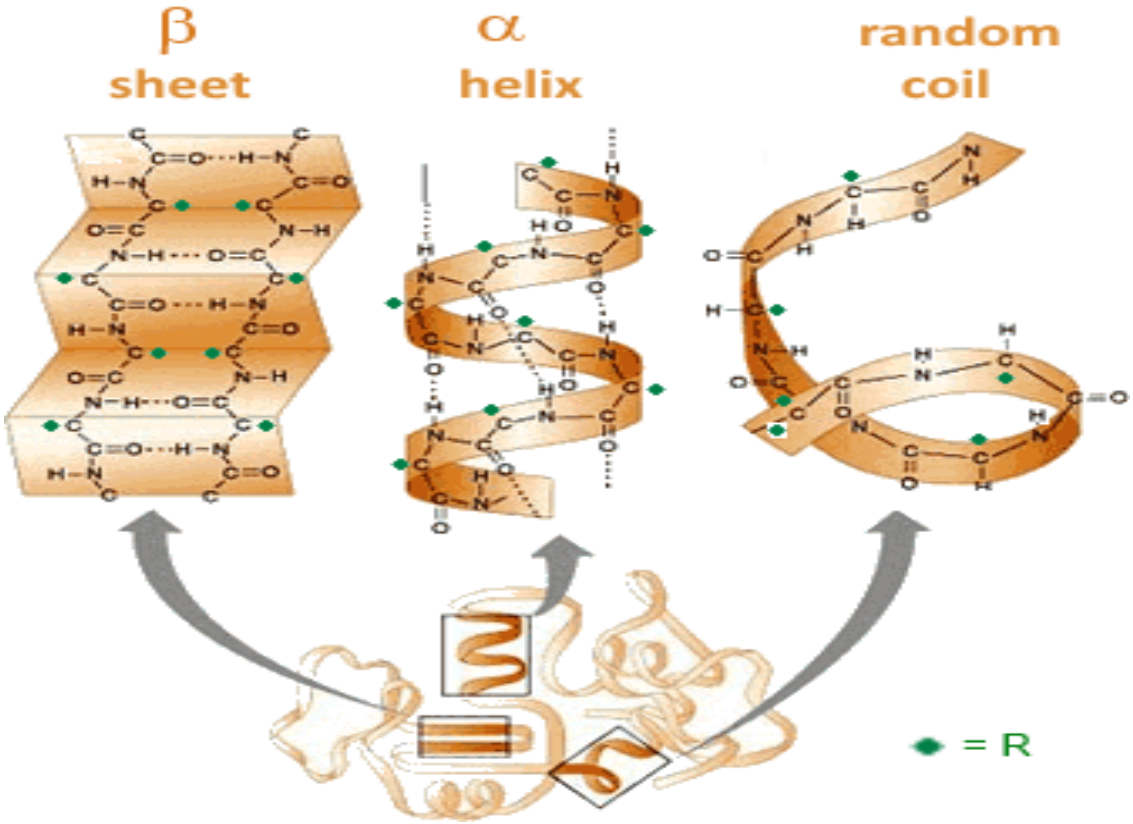
3.2 Προεπεξεργασία Δεδομένων εισόδου

Για την κατάλληλη προεπεξεργασία των δεδομένων εισόδου του δικτύου έχει γίνει χρήση της κλάσης StoreProtein.py(Ειρήνη Παπακώστα, 2016) στην οποία γίνεται η απαραίτητη επεξεργασία των δεδομένων αυτών ούτος ώστε να τροποποιηθούν στην μορφή εισόδου που απαιτεί το δίκτυο. Δημιουργούνται δύο ξεχωριστά αντικείμενα της κλάσης αυτής το pTrain και το pTest για να χρησιμοποιηθούν αργότερα σαν δεδομένα εκπαίδευσης και δεδομένα ελέγχου αντίστοιχα. Στην κλάση αυτή διαβάζονται όλα τα δεδομένα και συγκεκριμένα, όλες οι πρωτεΐνες που είναι γνωστή τόσο η πρωτοταγής όσο και η δευτεροταγής δομή τους, από το αρχείο msaProteinsTrainBigDtaset_afterProcess corr.txt και με τον τρόπο που επεξηγείται πιο πάνω, ανατρέχει στο αρχείο με την MSA μορφή για κάθε πρωτεΐνη.

Ακολούθως, εκτελείται η επεξεργασία των δεδομένων εισόδου του αρχείου msa κάθε πρωτεΐνης. Αρχικά, γίνεται η ευθυγράμμιση των δεδομένων και διαίρεση του κάθε κατάλοιπου ξεχωριστά με την σταθερά 100 έτσι ώστε να έρθουν στο διάστημα 0 με 1. Τα δεδομένα αυτά αποθηκεύοντα στον πίνακα **data** για να αποτελέσουν αργότερα την είσοδο του δικτύου. Παράλληλα στην μεταβλητή SizeOfAminoacids αποθηκεύεται το μέγεθος της αλληλολουχίας των αμινοξέων.

3.3 Προεπεξεργασία Δεδομένων Αναμενόμενης Εξόδου

Στην κλάση storeProtein.py γίνεται επίσης και η επεξεργασία της αναμενόμενης εξόδου του δικτύου για κάθε είσοδο του. Η πληροφορία αυτή βρίσκεται στο αρχείο msaProteinsTrainBigDtaset_afterProcess corr.txt και αποτελεί την τρίτη γραμμή σε κάθε αναπαράσταση μιας πρωτεΐνης. Η δευτεροταγής δομή της πρωτεΐνης χωρίζεται σε τρις κατηγορίες, α-έλικας, β-πτυχωτή και κάτι διαφορετικό από τα προηγούμενα. Η τρεις κατηγορίες αναπαραστούνται με τον συμβολισμό H(helix), E(extended) και L (loop) (Singh M., 2005). Έτσι, η δευτεροταγής δομή της κάθε πρωτεΐνης στο αρχείο msaProteinsTrainBigDtaset_afterProcess corr.txt εμφανίζεται σαν μια αλληλουχια που αποτελείται από τα σύμβολα των τριών κατηγοριών που προαναφέρθηκαν.



Σχήμα 3.3.1: Στο σχήμα αναπαρίσταται η αναδίπλωση των πρωτεϊνών στον χώρο. Υπάρχουν 3 κατηγορίες.. (H) Κατηγορία α-έλικας, (E) Κατηγορία β-πτυχωτή και (L) Κάτι άλλο.

Παρ' όλα αυτά για να χρησιμοποιηθεί η επιθυμητή έξοδος του δικτύου για κάθε πρωτεΐνη που μας παρέχεται από το αρχείο msaProteinsTrainBigDtaset_afterProcess corr.txt πρέπει να κωδικοποιηθεί. Αφού έχουμε τρεις διαφορετικές κατηγορίες θα χρησιμοποιήσουμε ένα συνδιασμό τριών εξόδων για αναπαράσταση της κάθε εξόδου. Η επιθυμητή έξοδος του δικτύου έχει κωδικοποιηθεί όπως φαίνεται στο πίνακα 3.3.1.

Κατηγορία	**Κωδικοποίηση**
α- έλικας (H)	[1,0,0]
β- πτυχωτή (E)	[0,1,0]
Κανένα από τα πιο πάνω (L)	[0,0,1]

Πίνακας 3.3.1 : Στην πρώτη στήλη του πίνακα εμφανίζονται οι τρις κατηγορίες και στην δεύτερη στήλη του πίνακα οι αντίστοιχες κωδικοποιήσεις.

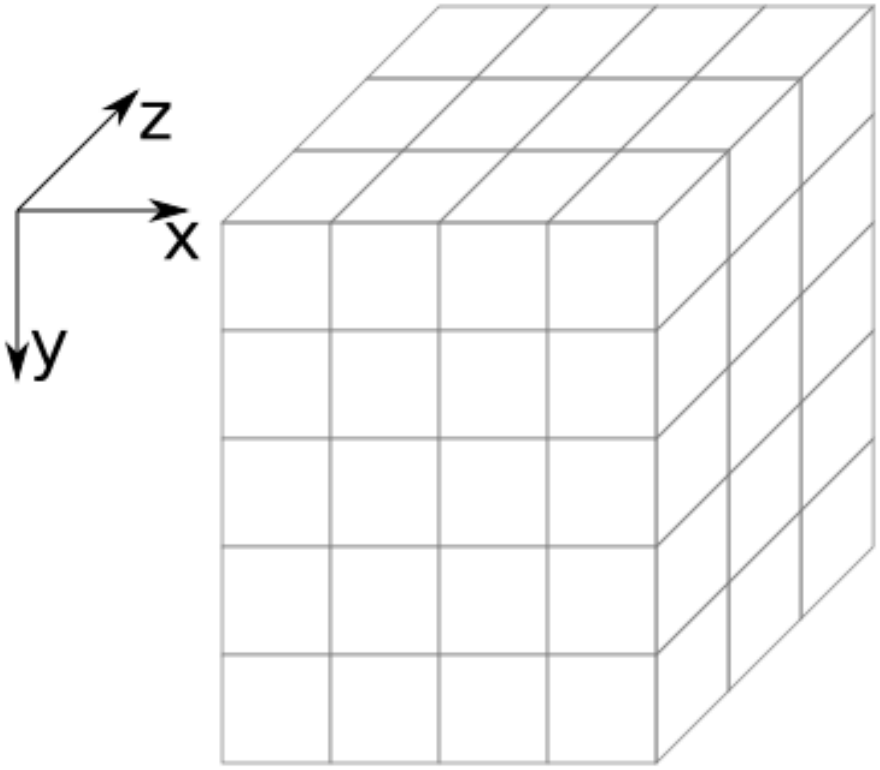
Οι κωδικοποιημένες επιθυμητές εξόδοι του δικτύου για κάθε πρωτεΐνη αποθηκεύονται στην λίστα Yt.

3.4 Διαμόρφωση πινάκων

Στην συνέχεια, εκτελείται επεξεργασία του πίνακα data ο οποίος περιέχει τα δεδομένα εισόδου και της λίστας Yt που περιέχει τις επιθυμητές εξόδους του δικτύου ούτως ώστε να περάσουμε τα δεδομένα στο δίκτυο.

Ο πίνακας data μετασχηματίζεται σε ένα τρισδιάστατο πίνακα της μορφής (n x m x y). Όπου n ο αριθμός τον δεδομένων εισόδου που θα πάρει το δίκτυο μέχρι να εκπαιδευτεί. Δηλαδή το sizeOfTrainingSet, όπου m το μέγεθος του window, δηλαδή το μέγεθος της

αλληλουχίας με την οποία θα συσχετίζονται τα δεδομένα και y το μέγεθος της εισόδου που δεχετε το δίκτυο (InputSize=20 => όσα είναι τα αμινοξέα που αποτελούν τις πρωτεΐνες). Για παράδειγμα, αν ορίσουμε το μέγεθος του window ίσο με 15, για κάθε χρονική στιγμή το Input του δικτύου θα αποτελείτε από ένα matrix διαστάσεων 20 x 15. Αυτή η επεξεργασία εκτελείτε στην συνάρτηση _load_data().



Σχήμα 3.4.1: Αναπαράσταση του πίνακα δεδομένων του δικτύου(Με μικρότερες διαστάσεις). Η διάσταση X αντιπροσωπεύει τον αριθμό των εισόδων του δικτύου, η διάσταση Z αντιπροσωπεύει το WindowSize και η διάσταση Y αντιπροσωπεύει το trainingSize. Ο matrix X x 1 αποτελεί την είσοδο του δικτύου για μια χρονική στιγμή t.

Κεφάλαιο 4

Clockwork-RNN

4.1 Εισαγωγή

Τα προβλήματα πρόβλεψης μιας αλληλουχίας δεδομένων, αποτελούν πρόκληση για την μηχανική μάθηση αφού απαιτείται η αναγνώριση των πολύπλοκων εξαρτήσεων μεταξύ των στοιχείων που αποτελούν την αλληλουχία. Σε αυτή την κατηγορία προβλημάτων ανήκει και το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών, αφού καλούμαστε να αναγνωρίσουμε ποια αμινοξέα, από την αλληλουχία των αμινοξέων τα οποία αποτελούν την πρωτοταγή δομή τους, αλληλεπιδρούν μεταξύ τους.

Παρόλο που μέχρι σήμερα έχουν γίνει πολλές προσπάθειες, με διάφορες μεθόδους για την επίλυση του προβλήματος PSSP, εντούτοις υπάρχουν ακόμη περιθώρια βελτίωσης αφού καλύτερα αποτελέσματα έχει ο αλγόριθμος του Baldi υλοποιημένος με το BRNN δίκτυο με ποσοστό επιτυχίας 76%.(P.Baldi et al., 1999). Αν και τα δίκτυα αμφίδρομης ανάδρασης(RNN), όπως το δίκτυο BRNN, θεωρητικά μπορούν να αντιμετωπίσουν τέτοιου είδους προβλήματα, όπου υπάρχουν εξαρτήσεις ανάμεσα στα δεδομένα, με το πλεονέκτημα της βραχυπρόθεσμης μνήμης που δημιουργείται μέσω των feedback ενώσεων τους, δυστυχώς δεν μπορούν να επιφέρουν το ίδιο καλά αποτελέσματα όταν απαιτήται μακροχρόνια μνήμη.

Έχοντας υπόψην τα παραπάνω, σε αυτή την Διπλωματική Εργασία θα παρουσιαστεί μια διαφορετική μέθοδος για την επίλυση του PSSP προβλήματος με την χρήση των Clockwork RNN δικτύων.(Koutnik et al., 2014).

4.2 Δίκτυα Clockwork-RNN

Το δίκτυο Clockwork Recurent Neural Network(CW-RNN) ανήκει στην κατηγορία των Recurent Neural Networks(RNN). Για την ακρίβεια, αποτελεί μια απλή αλλά ισχυρή τροποποίηση του απλού RNN, όπου το κρυφό επίπεδο είναι χωρισμένο σε ξεχωριστές ενότητες. Κάθε ενότητα εκτελεί υπολογισμούς μόνο στον δικό της προκαθορισμένο χρόνο(clock- rate). (Koutnik et al.,2014). Με αυτή την τροποποίηση του δικτύου RNN, όχι μόνο το δίκτυο δεν γίνεται πιο σύνθετο,αλλά αντιθέτως επιτυγχάνεται η μείωση των παραμέτρων του δικτύου. Το δίκτυο αυτό δημιουργήθηκε από των Koutnik το 2014(Koutnik et al., 2014) και αποτελεί μια σχετικά νέα αλλά πολύ υποσχόμενη μέθοδο

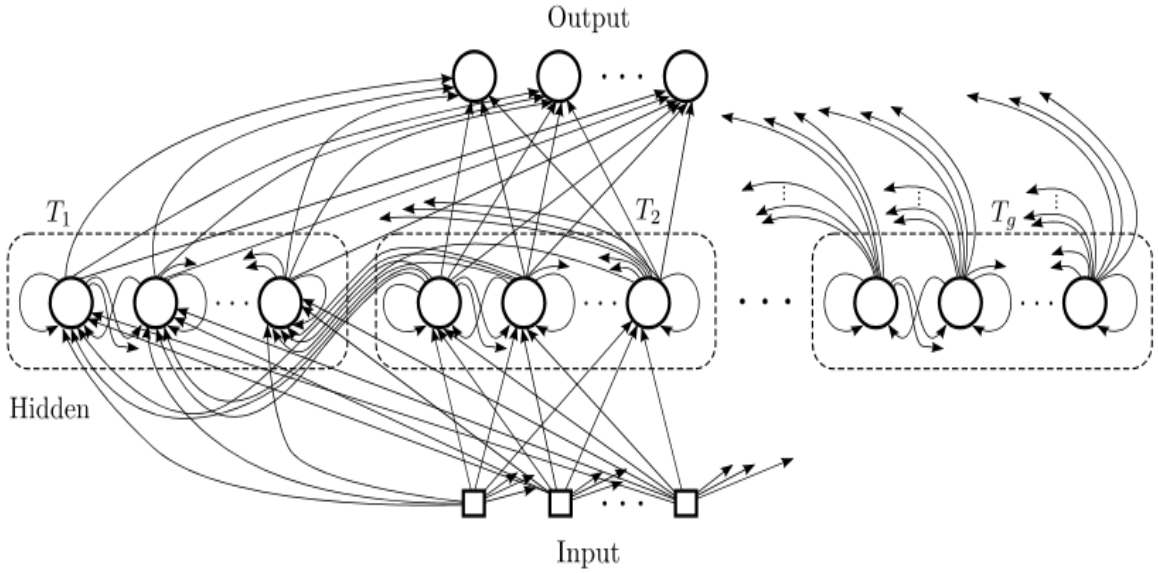
στον τομέα της μηχανικής μάθησης για την επίλυση προβλημάτων κατηγοριοποίησης και πρόβλεψης αλληλουχιών.

Αυτή η καινούργια παραλλαγή του δικτύου RNN, εκπαιδεύεται με τον αλγόριθμο λάθους back-propagation διαμέσω του χρόνου και έχει επιφέρει εξαιρετικά αποτελέσματα, μέχρι στιγμής, σε προβλήματα δημιουργίας και κατηγοριοποίησης αλληλουχιών όπου υπάρχουν long-term εξαρτήσεις μεταξύ των δεδομένων.

Το clockwork RNN δίκτυο επιτυγχάνει να επιλύσει το πρόβλημα των μακρυνών χρονικά εξαρτήσεων με τα διαφορετικά τμήματα στο οποίο είναι χωρισμένο το κρυφό του επίπεδο, τα οποία λειτουργούν σε διαφορετικό χρόνο, εκτελώντας τους υπολογισμούς τους σε διαφορετικές, διακριτές χρονικές περιόδους.

Επιπρόσθετα, το δίκτυο Clockwork RNN εκπαιδεύεται και ελέγχεται αρκετά πιο γρήγορα από το δίκτυο RNN αφού σε κάθε χρονική στιγμή δεν λειτουργούν(επεξεργάζονται) όλα τα τμήματα του κρυφού επιπέδου. Επίσης, έχει λιγότερα βάρη να επεξεργαστεί αφού στο κρυφό επίπεδο τα πιο αργά(slower) τμήματα δεν λαμβάνουν δεδομένα εισόδου από τα πιο γρήγορα(faster) τμήματα.

4.3 Αρχιτεκτονική Clockwork-RNN



Σχήμα 4.3.1(Koutnik et al, 2014) : Αναπαράσταση της αρχιτεκτονικής του δικτύου Clockwork RNN. Είναι πλήρες συνδεδεμένο δίκτυο και έχει ένα επίπεδο εισόδου, ένα κρυφό επίπεδο και ένα επίπεδο εξόδου. Το κρυφό επίπεδο είναι χωρισμένο σε g τμήματα, το καθένα με το δικό του χρόνο. Οι νευρώνες στο τμήμα i(faster) είναι συνδεδεμένοι με το τμήμα j(slower) αν ισχύει ότι Ti<Tj.

Το δίκτυο Clockwork RNN όπως φαίνεται και στο Σχήμα 4.3.1 έχει πολλά κοινά στοιχεία με τα δίκτυα αμφίδρομης ανάδρασης. Αποτελείται από το επίπεδο εισόδου, το κρυφό επίπεδο και το επίπεδο εξόδου. Υπάρχουν πρόσθιες(forward) συνδέσεις από το επίπεδο εισόδου στο κρυδό επίπεδο και από το κρυφό επίπεδο προς το επίπεδο εξόδου. Διαφέρει στο κρυφό επίπεδο όπου είναι χωρισμένο σε g τμήματα, το καθένα μεγέθους k. Κάθε τμήμα έχει μια προκαθορισμένη χρονική περίοδο Tn Ε {T1,…,Tg}. Όλα τα τμήματα είναι πλήρως συνδεδεμένα μεταξύ τους, αλλά οι συνδέσεις από το τμήμα j στο τμήμα i υπάρχουν μόνο αν ι χρονική περίοδος Ti είναι μικρότερη από την χρονική περίοδο Tj. Ταξινομόντας τα τμήματα με βάση την χρονική τους περίοδο κατά αύξουσα σειρά, οι συνδέσεις μεταξύ των τμημάτων θα ήταν από δεξιά προς τα αριστερά όπως φαίνεται στο Σχήμα 4.3.1.

4.4 Λειτουργία Clockwork-RNN

Είναι γνωστό, ότι στα δίκτυα αμφίδρομης ανάδρασης η έξοδος(output, yo(t)), για μια χρονική περίοδο t υπολογίζεται με τις ακόλουθες εξισώσεις:

$$\mathbf{y}_H^{(t)} = f_H(\mathbf{W}_H \cdot \mathbf{y}^{(t-1)} + \mathbf{W}_I \cdot \mathbf{x}^{(t)}), \qquad (1)$$

$$\mathbf{y}_O^{(t)} = f_O(\mathbf{W}_O \cdot \mathbf{y}_H^{(t)}), \qquad (2)$$

Σχήμα 4.4.1(Koutnik et al,2014) : Στο πάνω μέρος αναπαριστάται η εξίσωση που υπολογίζει της έξοδο ενός δικτύου αμφίδρομης ανάδρασης στο κρυφό επίπεδο (1) και στο κάτω μέρος η εξίσωση που υπολογίζει την έξοδο ενός νευρωνικού δικτύου αμφίδρομης ανάδρασης(2) για μια χρονική περίοδο t.

Στο Σχήμα 4.4.1 φαίνονται οι εξισώσεις εξόδου του δικτύου για το κρυφό επίπεδο(1) και το επίπεδο εξόδου του δικτύου(2). Όπου WH, WI και WO είναι οι πίνακες βαρών για το κρυφό επίπεδο, το επίπεδο εισόδου και το επίπεδο εξόδου αντίστοιχα. X(t) αναπαριστά το διάνυσμα εισόδου του δικτύου και το Y(t) είναι το διάνυσμα που δείχνει τους ενεργούς νευρώνες του κρυφού επιπέδου την χρονική στιγμη t. Οι συναρτήσεις που χρησιμοποιούνται και στις δύο εξισώσεις Fh και Fo δεν είναι γραμμικές.

Η βασική διαφορά μεταξύ των δύο δικτύων RNN και clockwork RNN είναι ότι στο δεύτερο για κάθε χρονική στιγμή t, μόνο οι εξόδοι των τμημάτων i που ικανοποιούν την εξίσωση (t MOD Ti = 0) είναι ενεργοί. Η επιλογή των περιόδων {T1, ... , Tg} είναι αυθαίρετη. Μια συνηθισμένη επιλογή για καθορισμό των χρονικών περιόδων αποτελεί ο καθορισμός εκθετικής σειράς : το τμήμα i έχει χρονική περίοδο Ti = 2^(i-1).

$$\mathbf{W}_H = \begin{pmatrix} \mathbf{W}_{H_1} \\ \vdots \\ \mathbf{W}_{H_g} \end{pmatrix} \qquad \mathbf{W}_I = \begin{pmatrix} \mathbf{W}_{I_1} \\ \vdots \\ \mathbf{W}_{I_g} \end{pmatrix}, \qquad (3)$$

Σχήμα 4.4.2(Koutnik et al, 2014): Οι πίνακες βαρών WH και WI.

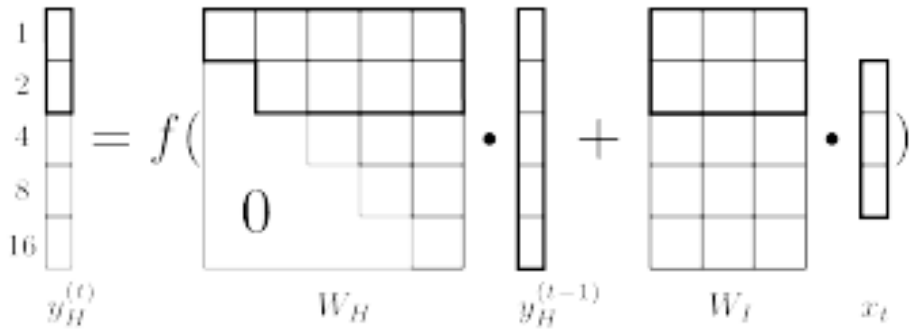
Στο Σχήμα 4.4.2 γίνεται αναπαράσταση των πινάκων με τα τα βάρη των συνδέσεων στο κρυφό επίπεδο και στο επίπεδο εισόδου. Οι πίνακες περιέχουν g σειρές (όσα και τα τμήματα του κρυφού επιπέδου). Ο πίνακας WH είναι άνω τριγωνικός πίνακας όπου κάθε γραμμή του πίνακα, WHi, είναι χωρισμένη σε στήλες {O1,....,Oi-1, WHi,I ,……, WHi,g}.

$$\mathbf{W}_{H_i} = \begin{cases} \mathbf{W}_{H_i} & \text{if } (t \bmod T_i) = 0 \\ \mathbf{0} & \text{otherwise,} \end{cases} \qquad (4)$$

Σχήμα 4.4.3(Koutnik et al,2014): Εξίσωση για έλεγχο των ενεργών τμημάτων του κρυφού επιπέδου για μια χρονική στιγμή t.

Σε κάθε χρονική στιγμή μπρόσθιου περάσματος, μόνο οι σειρές του WH και WI οι οποίες συνδέονται με τμήματα ενεργών νευρώνων, χρησιμοποιούνται για την εκτίμηση του διανύσματος εξόδου των αντίστοιχων τμημάτων, αναβαθμίζοντας το Yh. Τα υπόλοιπα τμήματα (μη ενεργά) διατηρούν τις τιμές εξόδου τους από την προηγούμενη χρονική στιγμή.

Ο υπολογισμός για την κατάσταση του κρυφού επιπέδου για μια χρονική στιγμή t=6 φαίνεται στο ακόλουθο Σχήμα 4.4.4.



Σχήμα 4.4.4: (Koutnik et al., 2014) Υπολογισμός των κόμβων του κρυφού επιπέδου που είναι ενεργοί την χρονική στιγμή t=6 σύμφωνα με την εξίσωση (1) στο (σχήμα). Την χρονική στιγμή 6, τα πρώτα δύο τμήματα με περιόδους T1=1 και T2=2 αντίστοιχα, συμμετέχουν στον υπολογισμό για αλλαγή των εξόδων.(Αναπαριστούνται σκιασμένα στο σχήμα).

Έτσι, με τον τρόπο αυτό επιτυγχάνεται η διατήρηση των δεδομένων στο δίκτυο μακροπρόθεσμα αφού στα τμήματα που δεν ικανοποιούν την εξίσωση ενεργοποίησης για μια χρονική στιγμή διατηρείται η πληροφορία από προηγούμενες χρονικές στιγμές. Με αυτό τον τρόπο συνδιάζονται τα δεδομένα μιας χρονικής στιγμής με τα δεδομένα προηγούμενων χρονικών στιγμών δημιουργώντας στο δίκτυο ένα είδος μακροπρόθεσμης μνήμης.

Για υπολογισμό του σφάλματος, όπως και στα δίκτυα αμφίδρομης ανάδρασης, χρησιμοποιείται ο αλγόριθμος οπισθοδρόμισης σφάλματος, με την διαφορά ότι η διάδοση σφάλματος γίνεται μόνο μέσα από τους κόμβους των τμημάτων που εκτελέστηκαν (ενεργοποιήθηκαν) την χρονική στιγμή t ενώ οι υπόλοιποι κόμβοι διατηρούν το σφάλμα τους από της προηγούμενες χρονικές στιγμές.

Κεφάλαιο 5

Υλοποίηση δικτύου

5.1 Υλοποίηση

Στο κεφάλαιο αυτό θα περιγραφεί ο σχεδιασμός και η υλοποίηση του δικτύου Clockwork RNN το οποίο χρησιμοποιήθηκε για την επίλυση του προβλήματος. Το δίκτυο το οποίο χρησιμοποιήθηκε έχει ανακτηθεί από το διαδίκτυο (https://github.com/tomrunia/ClockworkRNN) και είναι σε γλώσσα προγραμματισμού Python.

5.1.1 Γλώσσα Προγραμματισμού Python

Για την επίλυση του προβλήματος επιλέγηκε η γλώσσα προγραμματισμού Python λόγω των βοηθητικών βιβλιοθηκών που παρέχει και είναι πολύ χρήσιμες στον τομέα των Νευρωνικών Δικτύων.Επίσης η ανάγνωση των δεδομένων(πρωτεϊνών) από το δίκτυο το οποίο χρησιμοποιήθηκε ήταν αυτό που υλοποίησε η φοιτήτρια Ειρήνη Παπακώστα στην δική της διπλωματική εργασία και ήταν υλοποιημένο σε γλώσσα προγραμματισμού Python. Το ίδιο και η υλοποίηση του δικτύου η οποία ανακτήθηκε από το διαδίκτυο.

Μας παρέχει χρήση της βιβλιοθήκης **tensorflow**, όπου προσφέρει υψηλού επιπέδου διεπαφές για την εκπαίδευση νευρωνικών δικτύων. Το πλεονέκτημα που προσφέρεται με την χρήση του tensorflow είναι ότι πολλές από τις κοινές χαμηλού επιπέδου εργασίες έχουν υλοποιηθεί βελτιστοποιημένα στη βιβλιοθήκη του, οπότε δίνει την δυνατότητα η υλοποίηση του συστήματος να μπορεί να ξεκινήσει σε αρκετά υψηλό επίπεδο.

Άλλες βιβλιοθήκες οι οποίες ήταν πολύ χρήσιμες για την εκπόνιση του συστήματος και μας παρείχε η γλώσσα προγραμματισμού Python ήταν οι **Numpy** για την υλοποίηση και επεξεργασία πινάκων και η **Matplotlib** για την δημιουργία γραφικών παραστάσεων.

5.2 Κλάσεις του δικτύου

5.2.1 Κλάση StoreProtein.py

Η κλάση αυτή έχει υλοποιηθεί από την Ειρήνη Παπακώστα και χρησιμοποιήθηκε στο δίκτυο για την ανάγνωση και κωδικοποίηση των πρωτεϊνών από τα αρχεία τους ούτως

ώστε να δοθούν σαν είσοδος στο δίκτυο μας. Η πλήρης λειτουργία της έχει επεξηγηθεί στο Κεφάλαιο 3.

5.2.2 Κλάση Config.py

Στην κλάση αυτή ορίζονται οι διάφορες παράμετροι του δικτύου οι οποίες παίζουν σημαντικό ρόλο για το τελικό αποτέλεσμα. Αυτές οι παράμετροι θα χρησιμοποιηθούν κατα την εκπαίδευση και τον έλεγχο του δικτύου.

5.2.2.1 Παράμετροι Δικτύου

Periods

Με την παράμετρο αυτή καθορίζονται οι περιόδοι στις οποίες θα χωριστεί το κρυφό επίπεδο του δικτύου και η χρονική περίοδος του κάθε τμήματος Ti όπως εξηγήται στο Κεφάλαιο 4.

Num_steps

Αυτή η παράμετρος μεταβάλλεται στην συνέχεια μέσα στο πρόγραμμα αφού δείχνει τα συνολικά χρονικά βήματα που θα εκτελέσει το δίκτυο. Υπολογίζεται ως το γινόμενο των βημάτων ανά εποχή (steps_per_epoch) επι τον αριθμό των εποχών(num_epochs).

Num_Input

Καθορίζει τον αριθμό των κόμβων εισόδου του δικτύου. Στην περίπτωση του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών είναι ο αριθμός 20, αφού τα δεδομένα εισόδου για μια χρονική στιγμή είναι 20.

Num_hidden

Καθορίζει τον αριθμό των κόμβων που αποτελούν το κρυφό επίπεδο του δικτύου. Είναι οι κόμβοι οι οποίοι αργότερα σύμφωνα με την ιδιότητα του clockwork-RNN θα χωριστούν σε ξεχωριστά τμήματα.

Num_output

Καθορίζει τον αριθμό των κόμβων που αποτελούν το επίπεδο εξόδου του δικτύου. Στην περίπτωση του προβλήματος PSSP είναι ο αριθμός 3, αφού στην έξοδο οι τρείς κατηγορίες καθορίζονται από τον συνδιασμό των τριών εξόδων του δικτύου [0,0,1], [0,1,0], [1,0,0].

Num_epochs

Καθορίζει τον αριθμό των εποχών που θα έχει το δίκτυο. Μια ολοκληρομένη εποχή περιλαμβάνει την εκπαίδευση και τον έλεγχο του δικτύου. Επαναλαμβάνεται όσες είναι οι εποχές με όλα τα δεδομένα στο δίκτυο.

Batch_size

Είναι η παράμετρος που καθορίζει το μέγεθος των πρωτεϊνών που επεξεργάζεται το δίκτυο σε κάθε time_step.

Learning_rate

Ο ρυθμός μάθησης καθορίζει πόσο γρήγορα συγκλίνει η μάθηση. Χρειάζεται προσοχή γιατί μεγάλος ρυθμός μάθησης μπορεί να οδηγήσει σε ταλάντωση γύρω από τις βέλτιστες τιμές βαρών, ενώ σε αντίθεση περίπτωση μια μικρή τιμή στο ρυθμό μάθησης μπορεί να οδηγήσει σε παγίδευση σε τοπικά ακρότατα.

Window_size

Αν και αυτή η παράμετρος δεν ορίζεται από το αρχείο config.py αλλά μέσα στο αρχείο train.py παίζει καθοριστικό ρόλο για το δίκτυο. Το χρονικό κινητό παράθυρο είναι μια τεχνική που χρησιμοποιήται στα νευρωνικά δίκτυα με ανάδραση και ορίζει την έννοια της αλληλεξάρτησης μεταξύ των δεδομένων εισόδου. Το μέγεθος του χρονικού παραθύρου καθορίζει το χρονικό διάστημα μέσα στο οποίο θα μεταβάλλονται τα βάρη του δικτύου.

5.2.3 Κλάση Train.py

Στην κλάση αυτή γίνεται αρχικά η επεξεργασία των δεδομένων και η διαμόρφωση των πινάκων με τα δεδομένα εισόδου και τα αναμενόμενα αποτελέσματα εξόδου με την χρήση των τριών συναρτήσεων generate data, train_test_split και load_data. Η συνάρτηση generate_data δημιουργεί ένα data frame με τα δεδομένα εισόδου και τα αντίστοιχα αναμενόμενα αποτελέσματα τα οποία διαβάστηκαν από τα αρχεία με την βοήθεια της κλάσης StoreProtein.py. Η συνάρτηση load_data δημιουργεί τα matrixes με τα δεδομένα εισόδου και τα αναμενόμενα αποτελέσματα εξόδου του δικτύου σε κάθε περίπτωση. Τέλος, η συνάρτηση train_test_split χωρίζει τους πίνακες με τα δεδομένα σε δεδομένα εκπαίδευσης και δεδομένα ελέγχου.

```
    #Create the correct form of data!!!
def _load_data(data,data2, n_prev=15):
    docX, docY = [], []
    for i in range(15,len(data)):
            if(not (int(data2.iloc[i,0])==0 and int(data2.iloc[i,1])==0 and int(data2.iloc[i,2])==0)):
                    docX.append(data.iloc[i-n_prev:i].as_matrix())
                    docY.append(data2.iloc[i-n_prev].as_matrix())
    alsX = np.array(docX)
    alsY = np.array(docY)
    return alsX, alsY

def train_test_split(df,df2,data_size,test_size=0.1):
    X_data, Y_data= _load_data(df.iloc[0:],df2.iloc[0:])
    ntrn = int(round(len(X_data) * (1 - test_size)))
    X_train = X_data[0:ntrn]
    X_test = X_data[ntrn:]
    y_train = Y_data[0:ntrn]
    y_test = Y_data[ntrn:]
    return (X_train, y_train), (X_test, y_test)

def generate_data():
    print("[x] Generating training examples...")
    pTrain.data=pTrain.data.reshape((pTrain.sizeOfAminoascids),20)
    pTrain.yt=np.array(pTrain.yt)
    pTrain.yt.reshape(len(pTrain.aminoacids),3)
    pdata = pd.DataFrame({'a':pTrain.data[:,0],'b':pTrain.data[:,1],'c':pTrain.data[:,2],'d':pTrain.data[:,3]
                          ,'e':pTrain.data[:,4],'f':pTrain.data[:,5],'g':pTrain.data[:,6],'h':pTrain.data[:,7]
                          ,'i':pTrain.data[:,8],'j':pTrain.data[:,9],'k':pTrain.data[:,10],'l':pTrain.data[:,11]
                          ,'m':pTrain.data[:,12],'n':pTrain.data[:,13],'o':pTrain.data[:,14],'p':pTrain.data[:,15]
                          ,'q':pTrain.data[:,16],'r':pTrain.data[:,17],'s':pTrain.data[:,18],'t':pTrain.data[:,19]})
    pdata2 = pd.DataFrame({'a':pTrain.yt[:,0],'b':pTrain.yt[:,1],'c':pTrain.yt[:,2]})
    return train_test_split(pdata,pdata2,pTrain.data.shape[0])


# Load the training data
(X_train, y_train), (X_validation, y_validation) = generate_data()
```

Στην συνέχεια, γίνεται αρχικοποίηση των μεταβλητών step_in_epoch, steps_per_epoch, num_steps, num_train, num_validation και train_step. Επίσης, γίνεται αρχικοποίηση των μεταβλητών που θα χρησιμοποιηθούν για τον υπολογισμό του Accuracy του δικτύου στο τέλος κάθε εποχής αλλά και στο τέλος εκτέλεσης ολόκληρου του προγράμματος (count,count1,sum,sum1,count2,count3,sum2,sum3). Η μεταβλητή step_in_epoch μας δείχνει σε πιο βήμα της εποχής βρισκόμαστε κάθε χρονική στιγμή. Η μεταβλητή αυτή αυξάνεται μέχρι να φτάσει στα συνολικά βήματα που έχει κάθε εποχή(steps_per_epoch) και στο τέλος της μηδενίζεται και αρχίζει μια νέα εποχή. Η μεταβλητή num_steps αντιπροσωπεύει τα συνολικά βήματα που θα γίνουν κατά την εκτέλεση του

προγράμματος σε όλες τις εποχές. Τέλος, οι τρεις μεταβλητές num_train, num_validation και train_step δείχνουν των συνολικό αριθμό βημάτων εκπαίδευσης και ελέγχου του δικτύου και το βήμα εκπαίδευσης την συγκεκριμένη χρονική στιγμή, αντίστοιχα.

```
num_train      = X_train.shape[0]
num_validation = X_validation.shape[0]

config.num_steps  = X_train.shape[1]
config.num_input  = X_train.shape[2]
config.num_output = y_train.shape[1]


sum=0
sum1=0
count=0
count1=0

sum2=0
sum3=0
count2=0
count3=0
```

Ακολούθως, γίνεται η δημιουργία του μοντέλου του Clockwork δικτύου καλώντας την κλάση Clockwork.py με τις αντίστοιχες παραμέτρους του δικτύου που ορίσαμε στην αρχή του προγράμματος μέσω της κλάσης config.py

```
# Initialize TensorFlow model for counting as regression problem
print("[x] Building TensorFlow Graph...")
model = ClockworkRNN(config)
```

Το σημαντικότερο κομμάτι της κλάσης αποτελεί η εκπαίδευση του δικτύου. Ουσιαστικά, η εκπαίδευση του δικτύου γίνεται στην κλάση clockwork.py όμως, μέσω της κλάσης train.py ορίζεται το σύνολο των δεδομένων με τα οποία θα γίνει η εκπαίδευση σε κάθε βήμα. Επίσης, ορίζεται ο αριθμός τον επαναλήψεων που θα γίνουν μέχρι να εκπαιδευτεί το δίκτυο και αναγνωρίζεται το τέλος κάθε εποχής ούτως ώστε το δίκτυο να προχωρήσει σε έλεγχο με τα δεδομένα ελέγχου. Τέλος, γίνεται εκτύπωση στην οθόνη σε ποιό βήμα βρίσκεται το δίκτυο και το step error(tain_loss).

```
for _ in range(num_steps):

    ########################################################################
    ########################### TRAINING ###################################
    ########################################################################

    index_start = step_in_epoch*config.batch_size
    index_end   = index_start+config.batch_size


    # Actual training of the network
    _, train_step, train_loss, train_predictions, targets, learning_rate, train_summary = sess.run(
        [model.train_op,
         model.global_step,
         model.loss,
         model.predictions,
         model.targets,
         model.learning_rate,
         model.train_summary_op],
        feed_dict={
            model.inputs:  X_train[index_start:index_end,],
            model.targets: y_train[index_start:index_end,],
        }
    )

    print("[%s] Step %05i/%05i, LR = %.2e, Loss = %.5f" %
          (datetime.now().strftime("%Y-%m-%d %H:%M"), train_step, num_steps, learning_rate, train_loss))
```

Στο τέλος κάθε step γίνεται υπολογισμός του accuracy του δικτύου. Ο υπολογισμός γίνεται ένα προς ένα ελέγχοντας αν το αποτέλεσμα για κάθε αμινοξύ είναι ίδιο με το αναμενόμενο αποτέλεσμα. Αν αυτό ισχίει τότε αυξάνουμε ένα μετρητή. Το accuracy υπολογίζεται με την εξίσωση (counter*100/sum) όπου sum ο συνολικός αριθμός τον δεδομένων που εξετάστηκαν.

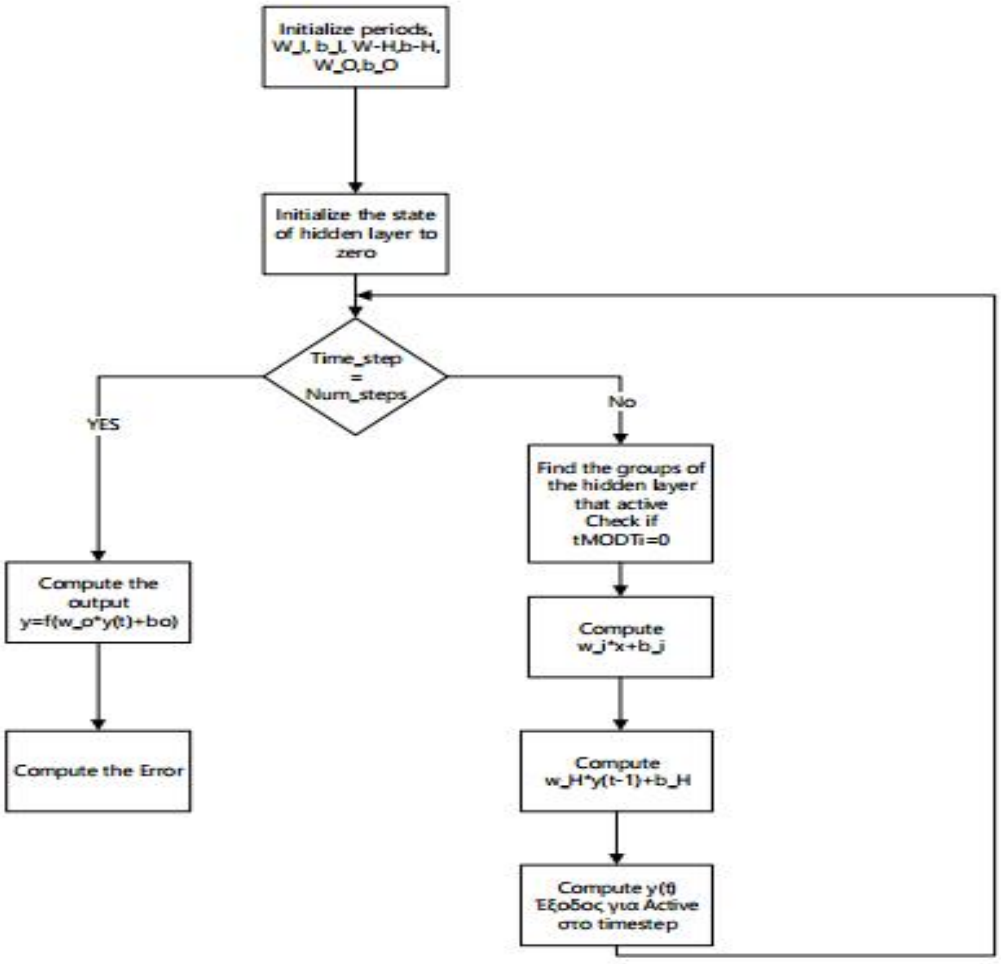
```
for i in range(train_predictions.shape[0]):
    max=train_predictions[i,0]
    pos=0
    for j in range(train_predictions.shape[1]):
        if(train_predictions[i][j]>max):
            max=train_predictions[i][j]
            pos=j
        train_predictions[i][j]=0
    train_predictions[i][pos]=1


res=targets-train_predictions

for i in range(res.shape[0]):
    flag=1
    for j in range(res.shape[1]):
        if(res[i][j]!=0):
            flag=0
    if(flag==1):
        count=count+1

sum=sum+config.batch_size
#print(count)
print("train accuracy")
print(count*100/sum)

sum2=sum2+sum
count2=count2+count
```

Στο τέλος κάθε εποχής, δηλαδή όταν το step_in_epoch είναι ίσο με το steps_per_epoch εκτελείται έλεγχος του δικτύου με τα δεδομένα ελέγχου. Η διαδικασία είναι αντίστοιχη με την διαδικασία εκπαίδευσης χωρίς όμως να έχουμε μεταβολή των βαρών του δικτύου. Όπως και στην εκπαίδευση του δικτύου στην κλάση train.py καθορίζεται ποια δεδομένα από τον πίνακα με τα δεδομένα ελέγχου θα σταλούν στο δίκτυο για έλεγχο σε κάθε βήμα. Αφού ολοκληρωθεί η διαδικασία ελέγχου, δηλαδή να σταλούν όλα τα δεδομένα ελέγχου στο δίκτυο, συνεχίζεται η εκπαίδευση του δικτύου. Στο τέλος κάθε βήματος ελέγχου, γίνεται υπολογισμός του accuracy του δικτύου για το testing με τον ίδιο τρόπο που γίνεται και για το training και εκτυπώνεται το error για κάθε βήμα ελέγχου στην οθόνη.

```
######################################################################
############### MODEL TESTING ON EVALUATION DATA ###############
######################################################################

if step_in_epoch == steps_per_epoch:

    # End of epoch, check some validation examples
    print("#" * 100)
    print("MODEL TESTING ON VALIDATION DATA (%i examples):" % num_validation)

    for validation_step in range(int(math.floor(num_validation/config.batch_size))):

        index_start = validation_step*config.batch_size
        index_end   = index_start+config.batch_size

        validation_loss, test_predictions, targets = sess.run(
            [model.loss,
             model.predictions,
             model.targets],
            feed_dict={
                model.inputs:  X_validation[index_start:index_end,],
                model.targets: y_validation[index_start:index_end,],
             }
        )


for i in range(test_predictions.shape[0]):
    max=test_predictions[i,0]
    pos=0
    for j in range(test_predictions.shape[1]):
        if(test_predictions[i][j]>max):
            max=test_predictions[i][j]
            pos=j
        test_predictions[i][j]=0
    test_predictions[i][pos]=1


res=targets-test_predictions

for i in range(res.shape[0]):
    flag=1
    for j in range(res.shape[1]):
        if(res[i][j]!=0):
            flag=0
    if(flag==1):
        count1=count1+1



sum1=sum1+config.batch_size
#print(count1)
print("test accuracy")
print(count1*100/sum1)

sum3=sum3+sum1
count3=count3+count1



print("[%s] Validation Step %03i. Loss = %.5f" % (datetime.now().strftime("%Y-%m-%d %H:%M"), validation_step, validation_loss))
```

5.2.4 Κλάση Clockwork.py

Η κλάση αυτή αποτελεί το κύριο μέρος του προγράμματος αφού είναι η προσομοίωση του μοντέλου του clockwork δικτύου. Σε αυτή την κλάση, έχουμε τη δημιουργία και τις αρχικοποιήσεις των πινάκων που θα περιέχουν τα βάρη τα κατώφλια και την κατάσταση των κόμβων σε κάθε βήμα. Στην κλάση clockwork γίνεται η ουσιαστική εκπαίδευση και ο έλεγχος του δικτύου, όπου περνούν τα δεδομένα στο δίκτυο για κάθε χρονική στιγμή και υπολογίζεται το αποτέλεσμα. Με την εφαρμογή του αλγορίθμου κατά την εκπαίδευση του δικτύου γίνονται update τα βάρη και αλλάζουν οι τιμές. Τέλος, σε αυτή την κλάση γίνεται υπολογισμός του error σε κάθε βήμα. Η λειτουργία της κλάσης αυτής απεικονίζεται στο πιο κάτω διάγραμμα ροής.



Η συνάρτηση build_model() η οποία ανήκει στην κλάση clockwork.py, αποτελεί την κύρια συνάρτηση της κλάσης. Στην κλάση αυτή, αρχικά, χωρίζονται τα δεδομένα εισόδου ούτως ώστε να υπάρχει μια σειρά από 20 δεδομένα (όσοι είναι και οι κόμβοι εισόδου του δικτύου) για κάθε χρονική στιγμή. Γίνεται αρχικοποίηση των χρονικών περιόδων στις οποίες χωρίστηκε το κρυφό επίπεδο και δημιουργία ανω τριγωνικού πίνακα που να αναπαριστά τις συνδέσεις μεταξύ τους. Στην συνέχεια, γίνεται δημιουργία και αρχικοποίηση πινάκων οι οποίοι αναπαριστούν τα βάρη και τα κατώφλια για τις συνδέσεις μεταξύ του επιπέδου εισόδου και κρυφού επιπέδου, μεταξύ του κρυφού επιπέδου και του επιπέδου εξόδου και από το κρυφό επίπεδο στο κρυφό επίπεδο.(κοκκινο κουτί) Ακολούθως, έχουμε δημιουργία και αρχικοποίηση πίνακα όπου αντιπροσωπεύει την κατάσταση κάθε κόμβου του κρυφού επιπέδου για κάθε χρονική στιγμή. Σε κάθε χρονικό βήμα εκτελεί τα εξής: ελέγχει ποιά μέρη του κρυφού επιπέδου είναι ενεργά σύμφωνα με την εξίσωση (t MOD T_i == 0)(αναπαριστάται με κίτρινο πιο κάτω), εκτελεί τις ανάλογες πράξεις για υπολογισμό της εξόδου του συστήματος(αναπαριστάται με μπλε πιο κάτω) και υπολογίζει το σφάλμα(αναπαριστάται με πράσινο πιο κάτω).

```
def _build_model(self):

    # Weight and bias initializers
    initializer_weights = tf.contrib.layers.variance_scaling_initializer()
    initializer_bias    = tf.constant_initializer(0.0)

    # Activation functions of the hidden and output state
    activation_hidden = tf.tanh
    activation_output = tf.nn.relu

    # Split into list of tensors, one for each timestep
    x_list = [tf.squeeze(x, squeeze_dims=[1]) for x in tf.split(self.inputs, self.config.num_steps, 1, name="inputs_list")]

    # Periods of each group: 1,2,4, ..., 256 (in the case num_periods=9)
    self.clockwork_periods = self.config.periods

    # Mask for matrix W_I to make sure it's upper triangular
    self.clockwork_mask = tf.constant(np.triu(np.ones([self.config.num_hidden, self.config.num_hidden])), dtype=tf.float32, name="mask")

    with tf.variable_scope("input"):
        self.input_W = tf.get_variable("W", shape=[self.config.num_input, self.config.num_hidden], initializer=initializer_weights)    # W_I
        self.input_b = tf.get_variable("b", shape=[self.config.num_hidden], initializer=initializer_bias)                              # b_I

    with tf.variable_scope("hidden"):
        self.hidden_W = tf.get_variable("W", shape=[self.config.num_hidden, self.config.num_hidden], initializer=initializer_weights)  # W_H
        self.hidden_W = tf.multiply(self.hidden_W, self.clockwork_mask)  # => upper triangular matrix                                    # W_H
        self.hidden_b = tf.get_variable("b", shape=[self.config.num_hidden], initializer=initializer_bias)                             # b_H

    with tf.variable_scope("output"):
        self.output_W = tf.get_variable("W", shape=[self.config.num_hidden, self.config.num_output], initializer=initializer_weights)  # W_O
        self.output_b = tf.get_variable("b", shape=[self.config.num_output], initializer=initializer_bias)                             # b_O
```

```
for time_step in range(self.config.num_steps):

    # Only initialize variables in the first step
    if time_step > 0: scope.reuse_variables()

    # Find the groups of the hidden layer that are active
    group_index = 0
    for i in range(len(self.clockwork_periods)):
        # Check if (t MOD T_i == 0)
        if time_step % self.clockwork_periods[i] == 0:
            group_index = i+1  # note the +1

    # Compute (W_I*x_t + b_I)
    WI_x = tf.matmul(x_list[time_step], tf.slice(self.input_W, [0, 0], [-1, group_index]))
    WI_x = tf.nn.bias_add(WI_x, tf.slice(self.input_b, [0], [group_index]), name="WI_x")

    # Compute (W_H*y_{t-1} + b_H), note the multiplication of the clockwork mask (upper triangular matrix)
    self.hidden_W = tf.multiply(self.hidden_W, self.clockwork_mask)
    WH_y = tf.matmul(self.state, tf.slice(self.hidden_W, [0, 0], [-1, group_index]))
    WH_y = tf.nn.bias_add(WH_y, tf.slice(self.hidden_b, [0], [group_index]), name="WH_y")

    # Compute y_t = (...) and update the cell state
    y_update = tf.add(WH_y, WI_x, name="state")
    y_update = activation_hidden(y_update)

    # Copy the updates to the cell state
    self.state = tf.concat([y_update, tf.slice(self.state, [0, group_index], [-1,-1])], 1)

# Save the final hidden state
self.final_state = self.state

# Compute the output, y = f(W_O*y_t + b_O)
self.predictions = tf.matmul(self.final_state, self.output_W)
self.predictions = tf.nn.bias_add(self.predictions, self.output_b)

# Compute the loss
self.error = tf.reduce_sum(tf.square(self.targets - self.predictions), reduction_indices=1)
self.loss  = tf.reduce_mean(self.error, name="loss")
```

Κεφάλαιο 6

Αποτελέσματα, συμπεράσματα και μελλοντική εργασία

6.1 Μετρικές που χρησιμοποιηθηκάν

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα του Clockwork-RNN δικτύου στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών. Συγκεκριμένα, παρουσιάζεται το πως αντιδρά το δίκτυο στην αλλαγή των διάφορων παραμέτρων.

Η μετρική που χρησιμοποιήθηκε για την αξιολόγηση των αποτελεσμάτων του δικτύου είναι ο έλεγχος ακριβείας του αποτελέσματος σε σχέση με το επιθυμιτό αποτέλεσμα ένα προς ένα αμινοξύ. Σε κάθε βήμα έγινε σύγκριση του αποτελέσματος του δικτύου με το αναμενόμενο αποτέλεσμα και αν το επιθυμητό με το πραγματικό αποτέλεσμα ήταν ίδια τότε είχαμε αύξηση ενός μετρητή. Η κωδικοποίηση του αποτελέσματος για να συγκριθεί με το επιθυμητό αποτέλεσμα γίνεται με βάση την κωδικοποίηση που είχε ακολουθήσει η Ειρήνη Παπακώστα στην δική της διπλωματική εργασία. Η μεγαλύτερη τιμή από τις τρεις εξόδους του δικτύου μετατρέπεται σε 1 και οι άλλες δύο εξόδοι σε 0, έτσι μπορούμε να συγκρίνουμε τα δύο αποτελέσματα. Το accuracy του δικτύου έχει υπολογιστεί σύμφωνα με την εξίσωση

$$Q_3 = \frac{\alpha\rho\iota\theta\mu\acute{o}\varsigma\ \kappa\alpha\tau\alpha\lambda\epsilon\acute{\iota}\pi\omega\nu\ \pi o\upsilon\ \pi\rho o\beta\lambda\acute{\epsilon}\phi\theta\eta\kappa\alpha\nu\ o\rho\theta\acute{\alpha}}{\sigma\upsilon\nu o\lambda\iota\kappa\acute{o}\ \alpha\rho\iota\theta\mu\acute{o}\varsigma\ \kappa\alpha\tau\alpha\lambda\epsilon\acute{\iota}\pi\omega\nu} * 100$$

Εξίσωση 6.1.1: εξίσωση υπολογισμού του ποσοστού επιτυχίας(accuracy)

Στο τέλος κάθε step είχαμε υπολογισμό του step_error με βάση την πιο κάτω εξίσωση. Όπου tpj είναι το αναμενόμενο αποτέλεσμα και Opj είναι το πραγματικό αποτέλεσμα.

$$E^p = \frac{1}{2}\sum_j (t_{pj} - o_{pj})^2$$

Εξίσωση 6.1.2: Εξίσωση υπολογισμού του σφάλματος

6.2 Πειράματα

Για την διεξαγωγή συμπερασμάτων σχετικά με τα αποτελέσματα του δικτύου, έγινε η εκτέλεση πολλών πειραμάτων με διάφορες παραμέτρους του δικτύου. Τα αποτελέσματα των πειραμάτων αυτών θα σχολιαστούν στην συνέχεια αναλυτικά.

6.2.1 Πείραμα 01

ΠΕΙΡΑΜΑ 1- WINDOW=15

Σε αρχική φάση η εκτέλεση του προγράμματος έγινε με τυχαίες τιμές στις παραμέτρους οι οποίες έδωσαν σχετικά μέτρια αποτελέσματα. Οι παράμετροι του δικτύου που εφαρμόστηκαν για την εκτέλεση του πειράματος 1 είναι οι ακόλουθες:

```
# Clockwork RNN parameters
periods     = [1, 2, 4, 8, 16, 32, 64] #, 128, 256]
num_steps   = 100
num_input   = 20
num_hidden  = 294
num_output  = 3
num_folds   = 10

# Optmization parameters
num_epochs          = 10
batch_size          = 100
learning_rate       = 1e-3
```

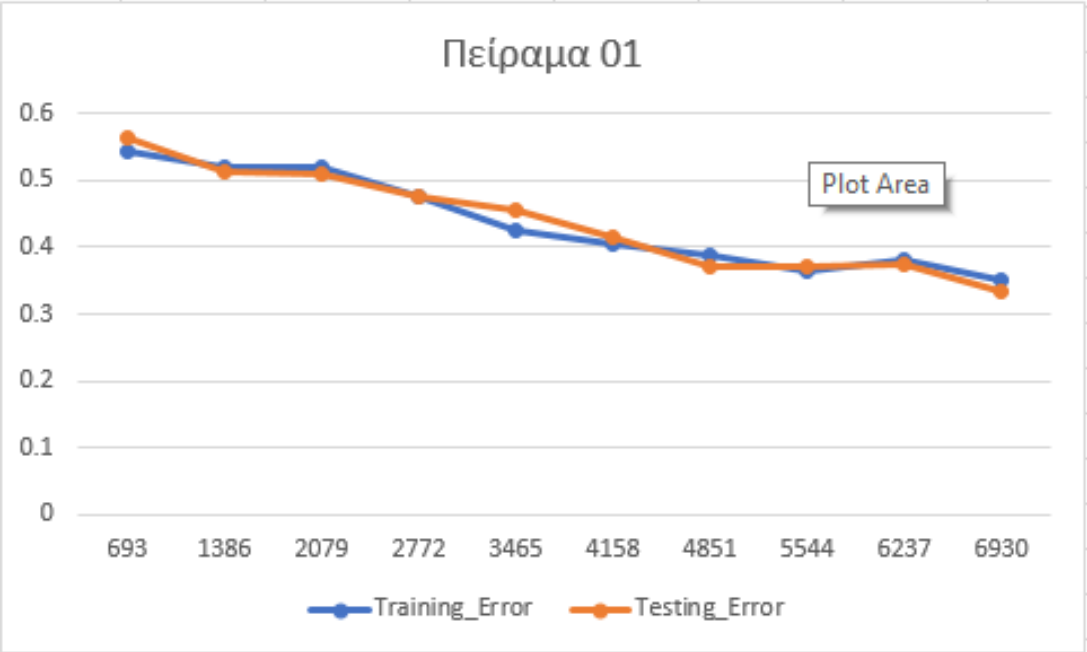
Εκτός από τις πιο πάνω παραμέτρους στο δίκτυο έχει εφαρμοστεί και η τεχνική του κινητού παράθυρου στο χρόνο έτσι ώστε να επιτύχουμε συσχέτιση των δεδομένων εισόδου. Το μέγεθος του window καθορίζει την συχέτιση μεταξύ των αμινοξέων. Στο συγκεκριμένο πείραμα το μέγεθος του κινητού χρονικού παραθύρου είναι ίσο με 15. Αυτό το μέγεθος έχει επιφέρει τα καλύτερα αποτελέσματα στην υλοποίηση με το Echo State Network(Παπακώστα, 2016).

Τα αποτελέσματα του δικτύου που πήραμε από την εφαρμογή του πειράματος 1 με τις πιο πάνω παραμέτρους είναι τα ακόλουθα:

	train_accuracy	test_accuracy
Epoch_01	41.39	51.36
Epoch_02	50.19	52.64
Epoch_03	51.06	64.9
Epoch_04	55.06	59.98
Epoch_05	57.98	61.15
Epoch_06	58.42	61.38
Epoch_07	58.55	61.85
Epoch_08	58.36	61.12
Epoch_09	58.58	61.57
Epoch_10	58.36	61.15

Πίνακας 6.2.1.1: Στα αριστερά φαίνεται το accuracy του δικτύου κατά την εκπαίδευση του και στα δεξιά κατά τον έλεγχο του στον τέλος κάθε περιόδου με την εφαρμογή του πειράματος 1.

Σχήμα 6.2.1.1: Αναπαράσταση του Accuracy του δικτύου σε κάθε εποχη κατά την εκπαίδευση και τον έλεγχο του δικτύου σύμφωνα με τα δεδομένα του σχήματος. Στον οριζόντιο άξονα αναπαριστάται η εποχή και στον κάθετο άξονα το ποσοστό επιτυχίας.



Σχήμα6.2.1.2: Γραφική Παράσταση που αναπαριστά το step_error του δικτύου κατά την εκπαίδευση και τον έλεγχο του. Στον οριζόντιο άξονα αναπαριστάται το step και στον κάθετο άξονα το αντίστοιχο error.

Όπως φαίνεται από τον πίνακα 6.2.1.1 σύμφωνα με το αποτέλεσμα του δικτύου το ποσοστό επιτυχίας κατά την εκπαίδευση του δικτύου φτάνει το 58.36%, ενω το ποσοστό ελέγχου φτάνει το 61.15%. Παρόλο που και τα δύο ποσοστά επιτυχίας δεν είναι και τόσο καλά αν λάβουμε υπόψην πως υπάρχει λύση όπου το ποσοστό επιτυχίας φτάνει το 76%, θεωρούνται όμως ικανοποιητικά ως πρώτο πείραμα και επίσης καθοριστικά για τα επόμενα πειράματα του δικτύου.

Στο σχήμα 6.2.1.2 αναπαρίσταται το σφάλμα κατά την εκπαίδευση και τον έλεγχο του δικτύου. Όπως φαίνεται στο σχήμα το σφάλμα και στις δύο περιπτώσεις το σφάλμα αρχίζει με τιμές γύρω στο 0.6 και με το πέρασμα του χρόνου εκτέλεσης του δικτύου βλέπουμε να μειώνεται γύρω στο 0.3. Δυστυχώς, όμως, το σφάλμα καθ'όλη την διάρκεια εκπαίδευσης και ελέγχου του δικτύου και με αύξηση των επαναλήψεων εκτέλεσης του δικτύου δεν μειώνεται. Η μικρότερη τιμή που παίρνει είναι 0.3. Το γεγονός αυτό ίσως να οφείλεται στην παγίδευση του δικτύου σε κάποιο τοπικό ακρότατο το οποίο θα

προσπαθήσουμε να λύσουμε με αύξηση της τιμής του learning rate(ρυθμός μάθησης) και των κόμβων στο κρυφό επίπεδο του δικτύου σε επόμενα πειράματα.

6.2.2 Πείραμα 02

ΠΕΙΡΑΜΑ 2-WINDOW=10

Επηρεασμένοι από τα αποτελέσματα που πείραμε από το πείραμα 1, στην προσπαθειά να αυξήσουμε το ποσοστό επιτυχίας του δικτύου θα μειώσουμε το window_size έτσι ώστε να ελέξουμε αν το δίκτυο με τις δεδομένες παραμέτρους μπορεί να αναγνωρίσει τις μακρινές εξαρτήσεις μεταξύ των δεδομένων εισόδου.

```
# Clockwork RNN parameters
periods     = [1, 2, 4, 8, 16, 32, 64] #, 128, 256]
num_steps   = 100
num_input   = 20
num_hidden  = 294
num_output  = 3
num_folds   = 10

# Optmization parameters
num_epochs          = 10
batch_size          = 100
learning_rate       = 1e-3
```

Όπως φαίνεται πιο πάνω, οι υπόλοιπες παράμετροι του δικτύου δεν έχουν υποστεί κάποια αλλαγή, παραμόνο το window_size το οποίο μειώσαμε σε 10. Τα αποτελέσματα που πείραμε από το δίκτυο μετά την αλλαγή του window_size φαίνονται στο Σχήμα 6.2.2.1:

	train_accuracy	test_accuracy
Epoch_01	39.5	48.27
Epoch_02	50.99	57.23
Epoch_03	57.67	62.14
Epoch_04	59.47	62.79
Epoch_05	60.13	62.88
Epoch_06	60.33	63.23
Epoch_07	60.49	63.07
Epoch_08	60.58	62.85
Epoch_09	60.14	63.55
Epoch_10	60.65	63.58

Πίνακας 6.2.2.1: Στα αριστερά φαίνεται το accuracy του δικτύου κατά την εκπαίδευση του και στα δεξιά κατά τον έλεγχο του στον τέλος κάθε περιόδου με την εφαρμογή του πειράματος 2.

Σχήμα 6.2.2.1: Αναπαράσταση του Accuracy του δικτύου σε κάθε εποχη κατά την εκπαίδευση και τον έλεγχο του δικτύου σύμφωνα με τα δεδομένα του σχήματος. Στον οριζόντιο άξονα αναπαριστάται η εποχή και στον κάθετο άξονα το ποσοστό επιτυχίας.

Σύμφωνα με τα αποτελέσματα που πήραμε με την εφαρμογή του πειράματος 2 κατά την εκπαίδευση του δικτύου επιτυγχάνεται ποσοστό επιτυγχίας 60.65% και κατά τον έλεγχο

του 63.58%. Σε σχέση με τα αποτελέσματα του πειράματος 1 είναι αρκετά ικανοποιητικά αφού έχουμε αύξηση του Accuracy του δικτύου και κατά την εκπαίδευση και κατά τον έλεγχο του. Οδηγούμαστε στο συμπέρασμα ότι με την μείωση του window_size τα αποτελέσματα του δικτύου είναι φαναιρά καλύτερα. Αφού με την μείωση του μεγέθους του κινητού παραθύρου στο χρόνο είχαμε αύξηση του ποσοστού επιτυχίας του δικτύου οδηγούμαστε στο συμπέρασμα πως το δίκτυο αδυνατεί να αναγνωρίσει τις μακρινές εξαρτήσεις που υπάρχουν μεταξύ των αμινοξέων της αλληλουχίας. Θα προχωρήσουμε σε ένα τρίτο πείραμα στο οποίο θα μειώσουμε ακόμα περισσότερο το μέγεθος του ούτως ώστε να δούμε αν υπάρχει βελτίωση στην πρόβλεψη των αμινοξέων.

6.2.3 Πείραμα 03

ΠΕΙΡΑΜΑ 3-WINDOW=5

Στο πείραμα αυτό θα επιχειρήσουμε να μειώσουμε το μέγεθος του κινητού παραθύρου στο χρόνο ακόμα περισσότερο αφού στο πείραμα 2 είχαμε καλύτερα αποτελέσματα από το πείραμα 1. Θα κρατήσουμε τις υπόλοιπες παραμέτρους σταθερές

```
# Clockwork RNN parameters
periods      = [1, 2, 4, 8, 16, 32, 64] #, 128, 256]
num_steps    = 100
num_input    = 20
num_hidden   = 294
num_output   = 3
num_folds    = 10

# Optmization parameters
num_epochs          = 10
batch_size          = 100
learning_rate       = 1e-3
```

Με την εφαρμογή των πιο πάνω παραμέτρων και την μείωση του μεγέθους του κινητού παραθύρου στο χρόνο να είναι ίσο με 5 παρατηρούμε τα αποτελέσματα που φαίνονται στο Σχήμα6.2.3.1:

	Training_accuracy	Test_accuracy
Epoch_01	46.24	55.94
Epoch_02	56.64	60.61
Epoch_03	59.89	62.71
Epoch_04	61.58	63.51
Epoch_05	61.66	63.8
Epoch_06	61.89	64.44
Epoch_07	61.83	64.14
Epoch_08	61.96	64.5
Epoch_09	62.24	64.59
Epoch_10	62.02	63.96

Πίνακας 6.2.3.1: Στα αριστερά φαίνεται το accuracy του δικτύου κατά την εκπαίδευση του και στα δεξιά κατά τον έλεγχο του στον τέλος κάθε περιόδου με την εφαρμογή του πειράματος 3.

Σχήμα 6.2.3.1: Αναπαράσταση του Accuracy του δικτύου σε κάθε εποχη κατά την εκπαίδευση και τον έλεγχο του δικτύου σύμφωνα με τα δεδομένα του σχήματος. Στον οριζόντιο άξονα αναπαριστάται η εποχή και στον κάθετο άξονα το ποσοστό επιτυχίας.

Σύμφωνα με τα αποτελέσματα του δικτύου κατά την εφαρμογή του πειράματος 3 το ποσοστό επιτυγχίας κατά την εκπαίδευση είναι ίσο με 62.02% και κατά τον έλεγχο του δικτύου είναι 63.96%. Μετά την εφαρμογή του πειράματος 3 οδηγούμαστε στο συμπέρασμα πως όσο πιο μικρό είναι το μέγεθος του κινητού παραθύρου στο χρόνο τόσο

πιο πολλά αμινοξέα προβλέπει σωστά το δικτυό μας. Είναι φανερό πως το δίκτυο με τις συγκεκριμένες παραμέτρους δεν αναγνωρίζει τις μακρινές εξαρτήσεις των δεδομένων. Έχοντας υπόψην πως υπάρχουν αυτές οι εξαρτήσεις μεταξύ των δεδομένων στο πρόβλημα PSSP θα επιχειρήσουμε να μεταβάλουμε τις υπόλοιπες παραμέτρους του δικτύου για βελτίωση του.

6.2.4 Πείραμα 04

ΠΕΙΡΑΜΑ 4-PERIODS=9

Στο πείραμα αυτό θα δοκιμάσουμε να κρατήσουμε σταθερό το μέγεθος του κινητού παραθύρου στον χρόνο και να μεταβάλουμε τον αριθμό τον περιόδων(periods) στο οποίο χωρίζεται το κρυφό επίπεδο του δικτύου μας. Συγκεκριμένα θα αυξήσουμε τον αριθμό των περιόδων από 7 σε 9. Επίσης θα αυξήσουμε τον αριθμό των κόμβων στο κρυφό επίπεδο από 294 σε 450 έτσι ώστε να μπορούν να χωριστούν σε 9 ίσες περιόδους κατά την εκτέλεση του προγράμματος. Στο πρόγραμμά μας γίνεται έλεγχος αν ο αριθμός των κόμβων του κρυφού επιπέδου στο δίκτυο μπορεί να διαιρεθεί στον αριθμό περιόδων χωρίς υπόλοιπο. Οι παράμετροι που εφαρμόστηκαν για το πείραμα 4 είναι οι ακόλουθες

```
# Clockwork RNN parameters
periods     = [1, 2, 4, 8, 16, 32, 64, 128, 256]
num_steps   = 100
num_input   = 20
num_hidden  = 450
num_output  = 3

# Optmization parameters
num_epochs          = 10
batch_size          = 100
```

Όπως αναπαριστάται στο σχήμα έχουμε μεταβάλει τον αριθμό των περιόδων. Συγκεκριμένα, προσθέσαμε ακόμη δύο περιόδους με χρονική περίοδο 128 & 256 αντίστοιχα και αυξήσαμε τον αριθμό των κρυφών κόμβων σε 450. Με την εφαρμογή του πειράματος 4 έχουμε τα εξής αποτελέσματα(Σχήμα 6.2.4.1)

	Training_accuracy	Testing_accuracy
Epoch_01	45.97	56.07
Epoch_02	57.01	61.44
Epoch_03	60.18	63.16
Epoch_04	61.9	63.76
Epoch_05	61.91	64.38
Epoch_06	62.03	64.36
Epoch_07	61.98	64.71
Epoch_08	61.78	64.71
Epoch_09	62.1	64.57
Epoch_10	62.22	64.74

Πίνακας 6.2.4.1: Στα αριστερά φαίνεται το accuracy του δικτύου κατά την εκπαίδευση του και στα δεξιά κατά τον έλεγχο του στον τέλος κάθε περιόδου με την εφαρμογή του πειράματος 4.

Σχήμα6.2.4.1: Αναπαράσταση του Accuracy του δικτύου σε κάθε εποχη κατά την εκπαίδευση και τον έλεγχο του δικτύου σύμφωνα με τα δεδομένα του σχήματος. Στον οριζόντιο άξονα αναπαριστάται η εποχή και στον κάθετο άξονα το ποσοστό επιτυχίας.

Όπως βλέπουμε από τα αποτελέσματα του δικτύου(Σχήμα 6.2.4.2) το ποσοστό επιτυχίας εκπαίδευσης του είναι 62.22% ενώ το ποσοστό επιτυχίας κατά τον έλεγχο του δικτύου είναι 64.74%. Σε σχέση με τα αποτελέσματα του πειράματος 3 δεν παρατηρείται ιδιαίτερη αύξηση στο ποσοστό επιτυχίας κατά την εκπαίδευση του δικτύου αφου είναι μόλις 0.2%, αλλά σημειώνεται αύξηση στο ποσοστό επιτυχίας κατά τον έλεγχο του δικτύου όπου ανέρχεται γύρω στο 1%. Δεν μπορούμε να έχουμε ξεκάθαρα συμπεράσματα αφου η αλλαγή στα αποτελέσματα είναι ελάχιστη. Παρ' όλα αυτά μπορούμε να διαπιστώσουμε πως με την αύξηση των κόμβων στο κρυφό επίπεδο και των περιόδων επιτυγχάνεται καλύτερη ανάλυση των δεδομένων και έτσι καλύτερα αποτελέσματα. Για τον λόγο αυτό θα δοκιμάσουμε σε επόμενο πείραμα να αυξήσουμε ακόμη περισσότερο τις περιόδους του δικτύου.

6.2.5 Πείραμα 05

ΠΕΙΡΑΜΑ 5-PERIODS=11

Στο πείραμα αυτό θα επιχειρήσουμε να αυξήσουμε τον αριθμό των περιόδων του δικτύου από 9 σε 11. Θα αυξήσουμε παράλληλα και τους κόμβους στο κρυφό επίπεδο του δικτύου για τον λόγο που εξηγήθηκε στο πείραμα 4. Πρέπει να είμαστε προσεχτικοί όμως γιατί η μεγάλη αύξηση των κόμβων στο κρυφό επίπεδο μπορεί να οδηγήσει σε υπερανάλυση των δεδομένων και άρα σε υπερεκπαίδευση του δικτύου. Οι παραμέτροι που θα εφαρμοστούν είναι οι εξής:

```
# Clockwork RNN parameters
periods     = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
num_steps   = 100
num_input   = 20
num_hidden  = 495
num_output  = 3

# Optmization parameters
num_epochs          = 10
batch_size          = 100
```

Με την εφαρμογή του πειράματος 5 έχουμε τα πιο κάτω αποτελέσματα που φαίνονται στο σχήμα 6.2.5.1.

	training_accuracy	testing_accuracy
Epoch_01	47.74	50.15
Epoch_02	58.87	62.29
Epoch_03	60.98	63.5
Epoch_04	61.54	64.25
Epoch_05	62.05	64.27
Epoch_06	62.21	64.67
Epoch_07	62.05	64.47
Epoch_08	62.57	64.08
Epoch_09	62.16	64.67
Epoch_10	62.35	64.75

Πίνακας 6.2.5.1: Στα αριστερά φαίνεται το accuracy του δικτύου κατά την εκπαίδευση του και στα δεξιά κατά τον έλεγχο του στον τέλος κάθε περιόδου με την εφαρμογή του πειράματος 5.

Σχήμα6.2.5.1: Αναπαράσταση του Accuracy του δικτύου σε κάθε εποχη κατά την εκπαίδευση και τον έλεγχο του δικτύου σύμφωνα με τα δεδομένα του σχήματος. Στον οριζόντιο άξονα αναπαριστάται η εποχή και στον κάθετο άξονα το ποσοστό επιτυχίας.

Παρατηρώντας τα αποτελέσματα του δικτύου για το πείραμα 5, το ποσοστό επιτυχίας για την εκπαίδευση του ανέρχεται στο 62.35% ενω το ποσοστό ελέγχου είναι 64.75%. Σε σχέση με το πείραμα 4, έχουμε μια πολύ μικρη αύξηση στα ποσοστά επιτυχίας η οποία όμως δεν είναι αξιοσημείωτη και ούτε ανάλογη της αύξησης των περιόδων. Λόγω αύξησης των κόμβων του κρυφού επιπέδου και των περιόδων στις οποίες χωρίζεται το κρυφό επίπεδο έχουμε αύξηση παράλληλα και στην πολυπλοκότητα του δικτύου. Με αυτό τον τρόπο το δίκτυο γίνεται πιο αργό. Έχοντας υπόψην την απειροελάχιστη αύξηση του ποσοστού επιτυχίας του δικτύου και την αυξημένη πολυπλοκότητα του, οδηγούμαστε στο συμπέρασμα πως δεν είναι μια καλή λύση η αύξηση των περιόδων και των κόμβων στο κρυφό επίπεδο.

6.2.6 Πείραμα 06

Στο πείραμα αυτό θα μεταβάλουμε την παράμετρο learning rate και συγκεκριμένα θα την αυξήσουμε με σκοπό την βελτίωση του ποσοστού επιτυχίας. Οι παράμετροι του δικτύου που χρησιμοποιήθηκαν για το πείραμα αυτό είναι οι ακόλουθες.

```
# Clockwork RNN parameters
periods      = [1, 2, 4, 8, 16, 32, 64] #, 128, 256]
num_steps    = 100
num_input    = 20
num_hidden   = 294
num_output   = 3
num_folds    = 10

# Optmization parameters
num_epochs          = 10
batch_size          = 100
learning_rate       = 1e-1
```

Τα αποτελέσματα που πήραμε από το δίκτυο φαίνονται στον πίνακα 6.2.6.1

	Train_Accuracy	Test_Accuracy
Epoch_01	48.6	50.58
Epoch_02	52.92	54.42
Epoch_03	54.94	60.15
Epoch_04	55.93	58.88
Epoch_05	56.82	60.37
Epoch_06	58.32	62.39
Epoch_07	58.49	60.45
Epoch_08	59.78	59.35
Epoch_09	60.3	63.2
Epoch_10	60.74	63.15

Πίνακας 6.2.6.1: Στα αριστερά φαίνεται το accuracy του δικτύου κατά την εκπαίδευση του και στα δεξιά κατά τον έλεγχο του στο τέλος κάθε περιόδου με την εφαρμογή του πειράματος_06.

Σύμφωνα με τα αποτελέσματα του δικτύου το ποσοστό επιτυχίας, κατά τον έλεγχο και την εκπαιδευσή του, με την αύξηση του ρυθμού μάθησης μειώνεται, αφού ανέρχεται μόλις στο 60.74% και 63.15% αντίστοιχα. Θεωρούμε το πείραμα 06 ένα αποτυχημένο πείραμα και θα διατηρήσουμε τον ρυθμό μάθησης του δικτύου σε χαμηλά επίπεδα.

6.2.7 Πείραμα 07

Στο πείραμα 07 θα μεταβάλουμε την παράμετρο batch_size του δικτύου και συγκεκριμένα θα την μειώσουμε με στόχο να πετύχουμε μεγαλύτερη ανάλυση των δεδομένων εισόδου του δικτύου. Οι παράμετροι του δικτύου κατά το πείραμα 07 είναι οι ακόλουθοι

```
# Clockwork RNN parameters
periods      = [1, 2, 4, 8, 16, 32, 64] #, 128, 256]
num_steps    = 100
num_input    = 20
num_hidden   = 294
num_output   = 3
num_folds    = 10

# Optmization parameters
num_epochs          = 10
batch_size          = 10
learning_rate       = 1e-3
```

Με την εκτέλεση του πειράματος 07 έχουμε τα αποτελέσματα που φαίνονται στον πίνακα 6.2.7.1.

	Train_Accuracy	Test_Accuracy
Epoch_01	56.68	60.77
Epoch_02	61.97	64.5
Epoch_03	62.4	64.59
Epoch_04	62.14	64.8
Epoch_05	62.5	64.83
Epoch_06	62.74	64.75
Epoch_07	62.36	64.83
Epoch_08	62.19	64.45
Epoch_09	62.15	64.74
Epoch_10	62.64	64.75

Πίνακας 6.2.7.1: Στα αριστερά φαίνεται το accuracy του δικτύου κατά την εκπαίδευση του και στα δεξιά κατά τον έλεγχο του στο τέλος κάθε περιόδου με την εφαρμογή του πειράματος_07.

Το ποσοστό επιτυχίας του δικτύου κατά την εκπαίδευση του στο πείραμα 07 ανέρχεται στο 62.64% που είναι το μεγαλύτερο ποσοστό επιτυχίας κατά την εκπαίδευση που έχουμε επιτύχει μέχρι στιγμής. Το ποσοστό επιτυχίας του δικτύου κατά τον έλεγχο του είναι 64.75% και είναι ίσο με το μεγαλύτερο ποσοστό επιτυχίας που έχουμε μέχρι στιγμής κατά τον έλεγχο του δικτύου. Το καλύτερο ποσοστό επιτυχίας μέχρι στιγμής είναι 64.75% και το έχουμε πετύχει με την αύξηση τον κόμβων του δικτύου στο κρυφό επίπεδο(Πείραμα 05) και την μείωση του batch_size του δικτύου(Πείραμα 07). Οδηγούμαστε στο συμπέρασμα πως με περισσότερη ανάλυση των δεδομένων στο δίκτυο έχουμε καλύτερα αποτελέσματα. Θα επιχειρήσουμε ένα ακόμη πείραμα κατά το οποίο θα μειώσουμε το batch_size του δικτύου σε 1.

6.2.8 Πείραμα 08

Στο πείραμα 08 έγινε προσπάθεια για μείωση του batch_size του δικτύου σε 1. Ουσιαστικά, έγινε εκτέλεση του online αλγορίθμού του δικτύου αφου με την μείωση του

batch_size σε 1 είχαμε απενεργοποίηση της τεχνικής αυτής. Ο online αλγόριθμος βοηθά στην αλλαγή των βαρών με βάση το προσωρινό σφάλμα που προκύπτει σε κάθε βήμα. Οι υπόλοιπες παράμετροι του δικτύου διατηρήθηκαν σταθερές

```
# Clockwork RNN parameters
periods      = [1, 2, 4, 8, 16, 32, 64] #, 128, 256]
num_steps    = 100
num_input    = 20
num_hidden   = 294
num_output   = 3
num_folds    = 10

# Optmization parameters
num_epochs           = 10
batch_size           = 1
learning_rate        = 1e-3
```

Τα αποτελέσματα του δικτύου με το πείραμα 08 φαίνονται στον πίνακα 6.2.8.1. Η πολυπλοκότητα του δικτύου αυξήθηκε κατα 10% και ο χρόνος εκτέλεσης του δικτύου δεκαπλασιάστηκε.

	Train_Accuracy	Test_Accuracy
Epoch_01	47.01%	50.52%
Epoch_02	59.40%	63.74%
Epoch_03	61.27%	63.89%
Epoch_04	61.68%	61.62%
Epoch_05	61.71%	63.63%
Epoch_06	61.63%	63.61%
Epoch_07	61.19%	63.63%
Epoch_08	61.54%	63.69%
Epoch_09	61.41%	63.73%
Epoch_10	61.49%	63.75%

Πίνακας 6.2.8.1: Στα αριστερά φαίνεται το accuracy του δικτύου κατά την εκπαίδευση του και στα δεξιά κατά τον έλεγχο του στο τέλος κάθε περιόδου με την εφαρμογή του πειράματος_08.

Με την μεταβολή αυτή της παραμέτρου batch_size του δικτύου σε ένα (1), έχουμε ποσοστά επιτυχίας κατά την εκπαίδευση και τον έλεγχο του δικτύου, 61.49% και 63.75% αντίστοιχα. Με την απενεργοποίηση του batch_mode έχουμε μείωση του ποσοστού

επιτυχίας του δικτύου κατά 1%. Οδηγούμαστε στο συμπέρασμα πως η τεχνική του batch size σε χαμηλές τιμές μας δίνει τα καλύτερα αποτελέσματα με τις συγκεκριμένες παραμέτρους του δικτύου μας. Επίσης κάνει το δίκτυο μας αρκετά πιο γρήγορο. Έτσι θα κρατήσουμε το batch_size του δικτύου σε χαμηλές τιμές.

6.3 Συμπεράσματα

Σε αυτή την διπλωματική εργασία έγινε μια προσπάθεια για επίλυση του προβλήματος της πρόβλεψης δευτεροταγούς δομής των πρωτεϊνών με την χρήση Clockwork-RNN δικτύου. Εκτελέστηκαν 8 πειράματα με διαφορετικές παραμέτρους στο κάθε ένα και πήραμε από το δίκτυο μας τα αποτελέσματα που φαίνονται στον πίνακα 6.3.1:

	Train_Accuracy	Test_Accuracy
Πείραμα 01	58.36%	61.15%
Πείραμα 02	60.65%	63.58%
Πείραμα 03	62.02%	63.96%
Πείραμα 04	62.22%	64.74%
Πείραμα 05	62.35%	64.75%
Πείραμα 06	60.74%	63.15%
Πείραμα 07	62.64%	64.75%
Πείραμα 08	61.49%	63.75%

Πίνακας 6.3.1: Στην δεξιά στήλη γίνεται αναπαράσταση του ποσοστού επιτυχίας κατά τον έλεγχο του δικτύου σε κάθε ένα από τα 8 πειράματα που έγιναν και στην αριστερή στήλη αναπαράσταση του ποσοστού επιτυχίας κατά την εκπαίδευση του δικτύου σε κάθε πείραμα.

Όπως βλέπουμε από το σχήμα 6.3.1 τα καλύτερα αποτελέσματα είχε το Πείραμα 07 κατά το οποίο μειώσαμε το batch_size του δικτύου και ορίσαμε την τιμή του batch_size=10. Το καλύτερο ποσοστό επιτυχίας του Clockwork δικτύου στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών ανέρχεται στο 64.75%. Θεωρείται ένα ικανοποιητικό ποσοστό επιτυχίας ως αρχική προσέγγιση στο πρόβλημα αυτό αλλά σίγουρα υπάρχουν αρκετά περιθώρια βελτίωσης.

Σε όλα τα πειράματα που εκτελέστηκαν παρατηρούμε πως το ποσοστό επιτυχίας ελέγχου του δικτύου είναι καλύτερο από το ποσοστό εκπαίδευσης του δικτύου, φαινόμενο το οποίο δεν συνηθίζεται. Η αιτία για τα αποτελέσματα αυτά δεν είναι ξεκάθαρη, όμως μπορούμε να αποδόσουμε ένα μερίδιο ευθύνης στο ότι πιθανότατα οι πρωτεΐνες οι οποίες χρησιμοποιήθηκαν για έλεγχο του δικτύου έχουν μεγάλη ομοιότητα με τις πρωτεΐνες που χρησιμοποιήθηκαν για την εκπαίδευση του. Αναμφισβήτητα, η αιτία για τα αποτελέσματα αυτά πρέπει να διερευνηθεί περαιτέρο αφού το γεγονός αυτό δεν συνηθίζεται.

Συνοψίζοντας, λοιπόν , συνάγεται το συμπέρασμα πως τα αποτελέσματα του δικτύου σε αρχική φαση θεωρούνται ικανοποιητικά αφου το ποσοστό επιτυχίας εκπαίδευσης του δικτύου φτάνει μέχρι 62.64% και το ποσοστό ελέγχου του δικτύου ανέρχεται στο 64.75%. Γίνεται, επομένος, εύκολα αντιληπτό, πως τα αποτελέσματα του δικτύου έχουν αρκετά περιθώρια βελτίωσης αφου ως γνωστό το δίκτυο BRNN έχει επιτύχει ποσοστό επιτυχίας 76%, με την χρήση μεθόδων filtering και Ensembles.

6.4 Μελλοντική Εργασία

Η προσπάθεια επίλυσης του προβλήματος της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών με την χρήση Clockwork-RNN που έγινε μέσα από την εκπόνηση της παρούσας διπλωματικής αποτελεί την αρχή για την ανάπτυξη μιας νέας μεθόδου για την λύση του υπάρχοντος προβλήματος. Ωστόσο, έχοντας υπόψην τα αποτελέσματα του δικτύου, γίνεται εύκολα αντιληπτό πως υπάρχουν αρκετά περιθώρια βελτίωσης των αποτελεσμάτων αυτών. Κατά συνέπεια είναι ανάγκη να εκτελεστούν κάποιες έρευνες στο μέλλον για να επιτευχθεί αυτό.

Αρχικά, πρέπει να εξεταστεί αναλυτικότερα γιατί τα ποσοστά επιτυχίας του δικτύου κατά τον έλεγχο του είναι καλύτερα από τα ποσοστά επιτυχίας που σημείωσε το δίκτυο κατά την εκπαίδευση του.

Επιπλέον, θα ήταν χρήσιμη η εφαρμογή μιας διαφορετικής μετρικής, της μετρικής SOV, ούτως ώστε να αποκτήσουμε μια πιο ξεκάθαρη εικόνα των αποτελεσμάτων του δικτύου

και να μπορούμε να οδηγηθούμε σε κάποιο εμφανές συμπερασμά κατά πόσο τα αποτελέσματα του δικτύου είναι ενθαρρυντικά για κάποια περεταίρω μελλοντική ανάπτυξη του δικτύου.

Αναφορές

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

Blazewicz J., Hammer L.P. and Lukasiak P.(2005) ‘‘Predicting Secondary Structures of Proteins Recognizing Properties of Amino Acids with the Logical Analysis of Data Algorithm”, IEEE Engineering in Medicine and Biology Magazine, Vol. 24(3), pp. 88-94.

Branden C Tooze J. (2006), Εισαγωγή στη δομή των πρωτεϊνων, Ακαδημαϊκές Εκδόσεις, Κεφάλαιο 2, Μοτίβα πρωτεϊνικής δομής, Ανάκτηση από διαδίκτυο, http://www.academicbooks.gr/sites/default/files/introduction_to_protein-structure_sample_ch02.pdf

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

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

Elman L.J. “Finding Structure in time”, CognitiveScience, Vol. 14, pp.179-211, 1990.

Frishman D. and Argos P.(1996), “Incorporation of non-local interactions in protein secondary structure prediction from the amino acid sequence” Protein Eng, Vol. 9(2), pp. 133-145.

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

Golden, R.., (1996), “Mathematical Methods for Neural Network Analysis and Design”. MIT Press.

C.B. Do and K. Katoh,”Protein Multiple Sequence Alignment”, Μεταφορτώθηκε στις 17/5/2016: http://ai.stanford.edu/~chuongdo/papers/alignment_review.pdf.

Kabsch, W. and Sander, C.(1983) “Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features”, Biopolymers, Vol22(12), pp.2577–2637

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

Koutnik J, Greff K, Gomes F, Schmidhuber J (2014). A Clockwork RNN, Proceedings of The 31st International Conference on Machine Learning, 1863-1871.

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

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

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

Singh M., “Predicting protein secondary and supersecondary structure”, Handbook of Computational Molecular Biology, Pinceton, 2005.

Αγαθοκλέους Μ. (2009), ΑΔΕ, «Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών με Νευρωνικά Δίκτυα Αμφίδρομης Ανάδρασης», Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

Βικπαίδεια. (2017) Πρωτεϊνες. πρόσβαση 10/03/2017. https://en.wikipedia.org/wiki/Protein

Βλαχάβας Ι, Κεφάλας Π, Βασιλειάδης Ν, Κόκκορας Φ, Σακελλαρίου Η, (2011), Τεχνητή Νοημοσύνη. Θεσσαλονίκη.

Παπακώστα Ε. (2016), ΑΔΕ, «Πρόβλεψη της δευτεροταγούς δομής των Πρωτεϊνών με την βοήθεια Echo-state Network », Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

Παυλίδης Π. (2016), ΑΔΕ, «Πρόβλεψη Δευτεροταγούς Δομής των Πρωτεϊνών με την χρήση των CNN για οπτική αναγνώριση αντικειμένων», Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

Προμπονάς Β. (2006) , Στοιχεία Αρχιτεκτονικής της τρισδιάστατης δομής των πρωτεϊνών, Σύγκριση και Κατηγοιοποίηση πρωτεϊνικών δομών – Σημειώσεις βιοπληροφορικής Αθήνα.

Προμπονάς Β. (2008) , Αρχιτεκτονική της τρισδιάστατης δομής πρωτεινων. Ερευνητικό εργαστήριο βιοπληροφορικής. Κύπρος.

Ρεφανίδης Γ.(2011), Νευρωνικά δίκτυα. Διάλεξη. Πρόσβαση 13/03/2017, http://opencourses.uom.gr/assets/site/public/259/187-Texnhth-Nohmosynh-01-Refanidis.pdf

Σημαντήρης Ν. (2005),Εισαγωγή στις βιοχημικές διεργασίες Χανια, http://lydakis.chania.teicrete.gr.

Βιβλιογραφία

Κεραυνού Ε.(2000), Τεχνητή Νοημοσύνη και έμπειρα συστήματα. Διάλεξη. Πρόσβαση 17 Φεβρουαρίου 2017.

Χριστοδούλου Γ.(2010), ΑΔΕ, «Διερεύνηση μεθόδων εκπαίδευσης Νευρωνικών Δικτύων Αμφίδρομης Ανάδρασης για πρόβλεψη δευτεροταγούς δομής των Πρωτεϊνών», Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

Χριστοδούλου Χ. (2016) – Διαλέξεις Μαθήματος ΕΠΛ442 Νευρωνικά Δίκτυα, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

Παραρτήματα

Readme File

This is a TensorFlow implementation of the Clockwork RNN proposed by Koutnik et al. (ICML, 2014) that is used for the prediction of the secondary structure of proteins.

Dependencies

- NumPy
- TensorFlow (tested with v1.0)
- MatplotLib and Pandas

How to Run:

1. Install Python 3.5
2. Install the Dependencies: NumPy, TensorFlow, MatplotLib and Pandas
3. Open the file train.py with Idle
4. Go to tab Run
5. Select Run Module

Παράρτημα Α

Κλάση storeProtein.py

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

import numpy as np
import protein
import sys


class storeProteins:

    def __init__(config):
        config.sizeOfAminoascids=0
        config.aminoacids=[]
        config.data=[]
        config.yt = []

    def readProteins(config,lines):
        for i in range(0, len(lines), 3):

            name = lines[i].rstrip()
            secondary = lines[i+2].rstrip()
            msa=np.loadtxt("../clockworkRNN-
master/input/msaFilesCorrect/"+name+'.hssp')

            #added from mike
            leadingZeros=np.zeros((15,20))
            config.data=np.append(config.data,leadingZeros)

            config.data=np.append(config.data,msa/100)
            config.sizeOfAminoascids+=len(msa)+15
            config.normilizeOutput(secondary)

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

    def normilizeOutput(config,secondary):


        for i in range(0,15):
           config.yt.append([0,0,0])
           config.aminoacids.append("0")
```

```
        se=secondary.split()

        for t in se[0]:

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

Παράρτημα Β

Κλάση Config.py

```
class Config(object):

    output_dir = "./output/"

    # Clockwork RNN parameters
    periods     = [1, 2, 4, 8, 16, 32, 64]
    num_steps   = 100
    num_input   = 20
    num_hidden  = 294
    num_output  = 3

    # Optmization parameters
    num_epochs          = 10
    batch_size          = 10
    optimizer           = "rmsprop"
    max_norm_gradient   = 10.0

    # Learning rate decay schedule
    learning_rate       = 1e-1
    learning_rate_decay = 0.975
    learning_rate_step  = 1000
    learning_rate_min   = 1e-5
```

Παράρτημα Γ

Κλάση Train.py

```
import tensorflow as tf
from tensorflow.python.framework import ops

from datetime import datetime
import os
import math
import numpy as np

from sklearn.utils import shuffle

from models.clockwork_rnn import ClockworkRNN
import storeProteins
from config import Config
#from utils.data_generator import *

import matplotlib as mpl
import matplotlib.pyplot as plt
#import seaborn as sns
import pandas as pd
import sys


def train(config):

    def init_training(self, trainData):
        self.trainData = trainData
        self.trainLen=self.trainData.sizeOfAminoascids

        self.Yt = self.trainData.yt
        self.Y = np.zeros((config.num_output,config.num_steps))

    def init_testing(self, testData):
        self.testData = testData
        self.testLen=self.testData.sizeOfAminoascids

        self.Ytest = self.testData.yt
        self.YtestR =
np.zeros((config.num_output,config.num_steps))



    plt.ion()
    plt.plot(1)
    #plt.show()

    open_file = open("../clockworkRNN-
master/input/TrainingSets/msaProteinsTrainBigDataset_afterProces
s corr.txt","r")
    #open_file = open("../clockworkRNN-
master/input/TrainingSets/one.txt","r")
```

```
    lines = open_file.readlines()

     #Create a storeProtein object
    pTrain=storeProteins.storeProteins()
    pTest=storeProteins.storeProteins()

    training=lines[0:]
    pTrain.readProteins(training)
    init_training(config,pTrain)

    testing=lines[0:]
    pTest.readProteins(testing)
    init_testing(config,pTest)

     #Create the correct form of data!!!
    def _load_data(data,data2, n_prev=5):
        docX, docY = [], []
        for i in range(15,len(data)):
           if(not (int(data2.iloc[i,0])==0 and
int(data2.iloc[i,1])==0 and int(data2.iloc[i,2])==0)):
                docX.append(data.iloc[i-n_prev:i].as_matrix())
                docY.append(data2.iloc[i-n_prev].as_matrix())
        alsX = np.array(docX)
        alsY = np.array(docY)
        return alsX, alsY

    def train_test_split(df,df2,data_size,test_size=0.1):
        X_data, Y_data= _load_data(df.iloc[0:],df2.iloc[0:])
        ntrn = int(round(len(X_data) * (1 - test_size)))
        X_train = X_data[0:ntrn]
        X_test = X_data[ntrn:]
        y_train = Y_data[0:ntrn]
        y_test = Y_data[ntrn:]
        return (X_train, y_train), (X_test, y_test)

    def generate_data():
        print("[x] Generating training examples...")

pTrain.data=pTrain.data.reshape((pTrain.sizeOfAminoascids),20)
        pTrain.yt=np.array(pTrain.yt)
        pTrain.yt.reshape(len(pTrain.aminoacids),3)
        pdata =
pd.DataFrame({'a':pTrain.data[:,0],'b':pTrain.data[:,1],'c':pTra
in.data[:,2],'d':pTrain.data[:,3]

,'e':pTrain.data[:,4],'f':pTrain.data[:,5],'g':pTrain.data[:,6],
'h':pTrain.data[:,7]

,'i':pTrain.data[:,8],'j':pTrain.data[:,9],'k':pTrain.data[:,10]
,'l':pTrain.data[:,11]

,'m':pTrain.data[:,12],'n':pTrain.data[:,13],'o':pTrain.data[:,1
4],'p':pTrain.data[:,15]
```

```
,'q':pTrain.data[:,16],'r':pTrain.data[:,17],'s':pTrain.data[:,1
8],'t':pTrain.data[:,19]})
        pdata2 =
pd.DataFrame({'a':pTrain.yt[:,0],'b':pTrain.yt[:,1],'c':pTrain.y
t[:,2]})
        return
train_test_split(pdata,pdata2,pTrain.data.shape[0])


    # Load the training data
    (X_train, y_train), (X_validation, y_validation) =
generate_data()


    num_train      = X_train.shape[0]
    num_validation = X_validation.shape[0]

    config.num_steps  = X_train.shape[1]
    config.num_input  = X_train.shape[2]
    config.num_output = y_train.shape[1]


    # Initialize TensorFlow model for counting as regression
problem
    print("[x] Building TensorFlow Graph...")
    model = ClockworkRNN(config)


    # Compute the number of training steps
    step_in_epoch, steps_per_epoch = 0,
int(math.floor(len(X_train)/config.batch_size))
    num_steps = steps_per_epoch*config.num_epochs
    train_step = 0

    # Checkpoint directory. Tensorflow assumes this directory
already exists so we need to create it
    checkpoint_dir =
os.path.abspath(os.path.join(config.output_dir, "checkpoints"))
    checkpoint_prefix = os.path.join(checkpoint_dir, "model")
    if not os.path.exists(checkpoint_dir):
        os.makedirs(checkpoint_dir)

    # Initialize the TensorFlow session
    gpu_options =
tf.GPUOptions(per_process_gpu_memory_fraction=0.75)
    sess = tf.Session(config=tf.ConfigProto(
        gpu_options=gpu_options,
        log_device_placement=False
    ))

    # Create a saver for all variables
    tf_vars_to_save = tf.trainable_variables() +
[model.global_step]
    saver = tf.train.Saver(tf_vars_to_save, max_to_keep=5)
```

```
    # Initialize summary writer
    summary_out_dir = os.path.join(config.output_dir,
"summaries")
    summary_writer  = tf.summary.FileWriter(summary_out_dir,
sess.graph)

    # Initialize the session
    init = tf.initialize_all_variables()
    sess.run(init)


    sum=0
    sum1=0
    count=0
    count1=0

    sum2=0
    sum3=0
    count2=0
    count3=0


    for _ in range(num_steps):


##################################################################
        ########################## TRAINING
############################

##################################################################

        index_start = step_in_epoch*config.batch_size
        index_end   = index_start+config.batch_size


        # Actual training of the network
        _, train_step, train_loss, train_predictions, targets,
learning_rate, train_summary = sess.run(
            [model.train_op,
             model.global_step,
             model.loss,
             model.predictions,
             model.targets,
             model.learning_rate,
             model.train_summary_op],
            feed_dict={
                model.inputs:  X_train[index_start:index_end,],
                model.targets: y_train[index_start:index_end,],
            }
        )

        print("[%s] Step %05i/%05i, LR = %.2e, Loss = %.5f" %
             (datetime.now().strftime("%Y-%m-%d %H:%M"),
train_step, num_steps, learning_rate, train_loss))
```

```
        for i in range(train_predictions.shape[0]):
            max=train_predictions[i,0]
            pos=0
            for j in range(train_predictions.shape[1]):
                if(train_predictions[i][j]>max):
                    max=train_predictions[i][j]
                    pos=j
                train_predictions[i][j]=0
            train_predictions[i][pos]=1


        res=targets-train_predictions

        for i in range(res.shape[0]):
            flag=1
            for j in range(res.shape[1]):
                if(res[i][j]!=0):
                    flag=0
            if(flag==1):
                count=count+1

        sum=sum+config.batch_size
        #print(count)
        print("train accuracy")
        print(count*100/sum)

        sum2=sum2+sum
        count2=count2+count







        # Save summaries to disk
        summary_writer.add_summary(train_summary, train_step)

        if train_step % 1000 == 0 and train_step > 0:
            path = saver.save(sess, checkpoint_prefix,
global_step=train_step)
            print("[%s] Saving TensorFlow model checkpoint to
disk." % datetime.now().strftime("%Y-%m-%d %H:%M"))



        step_in_epoch += 1


############################################################
        ############### MODEL TESTING ON EVALUATION DATA
###############
```

```
#################################################################

        if step_in_epoch == steps_per_epoch:

            # End of epoch, check some validation examples
            print("#" * 100)
            print("MODEL TESTING ON VALIDATION DATA (%i
examples):" % num_validation)

            for validation_step in
range(int(math.floor(num_validation/config.batch_size))):

                index_start = validation_step*config.batch_size
                index_end   = index_start+config.batch_size

                validation_loss, test_predictions, targets =
sess.run(
                    [model.loss,
                     model.predictions,
                     model.targets],
                    feed_dict={
                        model.inputs:
X_validation[index_start:index_end,],
                        model.targets:
y_validation[index_start:index_end,],
                     }
                )



                for i in range(test_predictions.shape[0]):
                     max=test_predictions[i,0]
                     pos=0
                     for j in range(test_predictions.shape[1]):
                         if(test_predictions[i][j]>max):
                             max=test_predictions[i][j]
                             pos=j
                         test_predictions[i][j]=0
                     test_predictions[i][pos]=1


                res=targets-test_predictions

                for i in range(res.shape[0]):
                    flag=1
                    for j in range(res.shape[1]):
                        if(res[i][j]!=0):
                            flag=0
                    if(flag==1):
                        count1=count1+1



                sum1=sum1+config.batch_size
```

```
                #print(count1)
                print("test accuracy")
                print(count1*100/sum1)

                sum3=sum3+sum1
                count3=count3+count1



                print("[%s] Validation Step %03i. Loss = %.5f" %
(datetime.now().strftime("%Y-%m-%d %H:%M"), validation_step,
validation_loss))



            # Reset for next epoch
            step_in_epoch = 0

            print("train accuracy in epoch")
            print(count*100/sum)
            print("test accuracy_in epoch")
            print(count1*100/sum1)



            sum=0
            sum1=0
            count=0
            count1=0



            # Shuffle training data
        perm = np.arange(num_train)
        np.random.shuffle(perm)
        X_train = X_train[perm]
        y_train = y_train[perm]

    print("#" * 100)

    print("train_accuracy of program")
    print(count2*100/sum2)
    print("test_accuracy of program")
    print(count3*100/sum3)


    # Destroy the graph and close the session
    ops.reset_default_graph()
    sess.close()


if __name__ == "__main__":
    train(Config())
```

Παράρτημα Δ

Κλάση Clockwork_rnn.py

```
import numpy as np
import tensorflow as tf
import storeProteins


class ClockworkRNN(object):


    def __init__(self, config):

        self.config = config

        # Check if the number of groups (periods) in the hidden
layer
        # is compatible with the total number of units in the
layer. Note that
        # this is not a requirement in the paper; there the
extra neurons are
        # divided over the higher frequency groups.
        assert self.config.num_hidden % len(self.config.periods)
== 0

        # Global training step
        self.global_step = tf.Variable(0, name='global_step',
trainable=False)

        # Initialize placeholders
        self.inputs  = tf.placeholder(tf.float32, shape=[None,
self.config.num_steps, self.config.num_input], name="inputs")
        self.targets = tf.placeholder(tf.float32, shape=[None,
self.config.num_output], name="targets")

        # Build the complete model
        self._build_model()

        # Initialize the optimizer with gradient clipping
        self._init_optimizer()

        # Operations for creating summaries
        self._build_summary_ops()


    def _build_model(self):

        # Weight and bias initializers
        initializer_weights =
tf.contrib.layers.variance_scaling_initializer()
        initializer_bias    = tf.constant_initializer(0.0)

        # Activation functions of the hidden and output state
```

```
        activation_hidden = tf.tanh
        activation_output = tf.nn.relu

        # Split into list of tensors, one for each timestep
        x_list = [tf.squeeze(x, squeeze_dims=[1]) for x in
tf.split(self.inputs, self.config.num_steps, 1,
name="inputs_list")]

        # Periods of each group: 1,2,4, ..., 256 (in the case
num_periods=9)
        self.clockwork_periods = self.config.periods

        # Mask for matrix W_I to make sure it's upper triangular
        self.clockwork_mask =
tf.constant(np.triu(np.ones([self.config.num_hidden,
self.config.num_hidden])), dtype=tf.float32, name="mask")

        with tf.variable_scope("input"):
            self.input_W = tf.get_variable("W",
shape=[self.config.num_input, self.config.num_hidden],
initializer=initializer_weights)    # W_I
            self.input_b = tf.get_variable("b",
shape=[self.config.num_hidden], initializer=initializer_bias)
# b_I

        with tf.variable_scope("hidden"):
            self.hidden_W = tf.get_variable("W",
shape=[self.config.num_hidden, self.config.num_hidden],
initializer=initializer_weights)  # W_H
            self.hidden_W = tf.multiply(self.hidden_W,
self.clockwork_mask)  # => upper triangular matrix
# W_H
            self.hidden_b = tf.get_variable("b",
shape=[self.config.num_hidden], initializer=initializer_bias)
# b_H

        with tf.variable_scope("output"):
            self.output_W = tf.get_variable("W",
shape=[self.config.num_hidden, self.config.num_output],
initializer=initializer_weights)  # W_O
            self.output_b = tf.get_variable("b",
shape=[self.config.num_output], initializer=initializer_bias)
# b_O

        with tf.variable_scope("clockwork_cell") as scope:

            # Initialize the hidden state of the cell to zero
(this is y_{t_1})
            self.state = tf.get_variable("state",
shape=[self.config.batch_size, self.config.num_hidden],
initializer=tf.zeros_initializer(), trainable=False)

            for time_step in range(self.config.num_steps):

                # Only initialize variables in the first step
```

```
                if time_step > 0: scope.reuse_variables()

                # Find the groups of the hidden layer that are
active
                group_index = 0
                for i in range(len(self.clockwork_periods)):
                    # Check if (t MOD T_i == 0)
                    if time_step % self.clockwork_periods[i] ==
0:
                        group_index = i+1  # note the +1

                # Compute (W_I*x_t + b_I)
                WI_x = tf.matmul(x_list[time_step],
tf.slice(self.input_W, [0, 0], [-1, group_index]))
                WI_x = tf.nn.bias_add(WI_x,
tf.slice(self.input_b, [0], [group_index]), name="WI_x")

                # Compute (W_H*y_{t-1} + b_H), note the
multiplication of the clockwork mask (upper triangular matrix)
                self.hidden_W = tf.multiply(self.hidden_W,
self.clockwork_mask)
                WH_y = tf.matmul(self.state,
tf.slice(self.hidden_W, [0, 0], [-1, group_index]))
                WH_y = tf.nn.bias_add(WH_y,
tf.slice(self.hidden_b, [0], [group_index]), name="WH_y")

                # Compute y_t = (...) and update the cell state
                y_update = tf.add(WH_y, WI_x, name="state")
                y_update = activation_hidden(y_update)

                # Copy the updates to the cell state
                self.state = tf.concat([y_update,
tf.slice(self.state, [0, group_index], [-1,-1])], 1)

            # Save the final hidden state
            self.final_state = self.state

            # Compute the output, y = f(W_O*y_t + b_O)
            self.predictions = tf.matmul(self.final_state,
self.output_W)
            self.predictions = tf.nn.bias_add(self.predictions,
self.output_b)


           # y =
reshape(self.predictions,(1,self.num_output))[0]
            #maxPosition=argmax(y)
            #yout=zeros(3)
            #yout[maxPosition]=1
            #self.predictions =
activation_output(self.predictions, name="output")

          # print (str(self.numToA(maxPosition)) + " " +
str(self.numToA(argmax(self.Yt[num_steps])))
```

```
            # Compute the loss
            self.error = tf.reduce_sum(tf.square(self.targets -
self.predictions), reduction_indices=1)
            self.loss  = tf.reduce_mean(self.error, name="loss")


    def _init_optimizer(self):

        # Learning rate decay, note that is
self.learning_rate_decay == 1.0,
        # the decay schedule is disabled, i.e. learning rate is
constant.
        self.learning_rate = tf.train.exponential_decay(
            self.config.learning_rate,
            self.global_step,
            self.config.learning_rate_step,
            self.config.learning_rate_decay,
            staircase=True
        )
        self.learning_rate = tf.maximum(self.learning_rate,
self.config.learning_rate_min)
        tf.summary.scalar("learning_rate", self.learning_rate)

        # Definition of the optimizer and computing gradients
operation
        if self.config.optimizer == 'adam':
            # Adam optimizer
            self.optimizer =
tf.train.AdamOptimizer(learning_rate=self.learning_rate)
        elif self.config.optimizer == 'rmsprop':
            # RMSProper optimizer
            self.optimizer =
tf.train.RMSPropOptimizer(learning_rate=self.learning_rate)
        elif self.config.optimizer == 'adagrad':
            # AdaGrad optimizer
            self.optimizer =
tf.train.AdagradOptimizer(learning_rate=self.learning_rate)
        else:
            raise ValueError("Unknown optimizer specified")

        # Compute the gradients for each variable
        self.grads_and_vars =
self.optimizer.compute_gradients(self.loss)

        # Optionally perform gradient clipping by max-norm
        if self.config.max_norm_gradient > 0:
            # Perform gradient clipping by the global norm
            grads, variables = zip(*self.grads_and_vars)
            grads_clipped, _ = tf.clip_by_global_norm(
                grads, clip_norm=self.config.max_norm_gradient)

            # Apply the gradients after clipping them
            self.train_op = self.optimizer.apply_gradients(
                zip(grads_clipped, variables),
```

```
                global_step=self.global_step
            )

        else:
            # Unclipped gradients
            self.train_op = self.optimizer.apply_gradients(
                self.grads_and_vars,
                global_step=self.global_step
            )

        # Keep track of gradient values and their sparsity
        grad_summaries = []
        for g, v in self.grads_and_vars:
            if g is not None:
                grad_hist_summary =
tf.summary.histogram("gradients/{}/hist".format(v.name), g)
                sparsity_summary  =
tf.summary.scalar("gradients/{}/sparsity".format(v.name),
tf.nn.zero_fraction(g))
                grad_summaries.append(grad_hist_summary)
                grad_summaries.append(sparsity_summary)
        self.gradient_summaries_merged =
tf.summary.merge(grad_summaries)


    def _build_summary_ops(self):

        # Training summaries
        training_summaries = [
            tf.summary.scalar("train/loss", self.loss),
            tf.summary.scalar("train/learning_rate",
self.learning_rate),
        ]

        # Combine the training summaries with the gradient
summaries
        self.train_summary_op =
tf.summary.merge([training_summaries,
self.gradient_summaries_merged])
```