Ατομική Διπλωματική Εργασία

ΠΡΟΒΛΕΨΗ ΔΕΥΤΕΡΟΤΑΓΟΥΣ ΔΟΜΗΣ ΤΩΝ ΠΡΩΤΕΪΝΩΝ ΜΕ ΤΗ ΧΡΗΣΗ CLOCKWORK ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ

Παναγιώτης Δημητρίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2018

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόβλεψη Δευτεροταγούς Δομής Των Πρωτεϊνών Με Τη Χρήση Clockwork Νευρωνικών Δικτύων

Παναγιώτης Δημητρίου

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 2018

Ευχαριστίες

Ένα κεφάλαιο τεσσάρων χρόνων γεμάτο κούραση, σκληρή δουλειά και πολύ πείσμα τελειώνει, έχοντας στο αποκορύφωμα του τη συγκεκριμένη διπλωματική εργασία. Η φοιτητική αυτή ζωή κλείνει αλλά φαντάζει σαν να είχε ξεκινήσει χθες. Είχα την τύχη να γνωρίσω τα κατάλληλα άτομα, έτσι ώστε να εισέλθω στον τομέα της έρευνας και να ασχοληθώ με αυτή, σε αυτή την διπλωματική εργασία. Με αυτό τον τρόπο κατάφερα να εξελιχθώ ως άτομο, να αντιμετωπίσω τους φόβους μου, αλλά και να γνωρίσω τα όρια μου.

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου Δρ. Χρίστο Χριστοδούλου, ο οποίος μου έδειξε με τις γνώσεις του τη σημασία και τη μαγεία που υποκρύβεται πίσω από τα νευρωνικά δίκτυα. Επίσης, θέλω να τον ευχαριστήσω για την εμπιστοσύνη αλλά και για το χρόνο που μου αφιέρωσε, για το καλύτερο δυνατό αποτέλεσμα της διπλωματικής αυτής εργασίας.

Στη συνέχεια, θέλω να ευχαριστήσω τον Δρ. Βασίλειο Προμπονά, επίκουρο του Τμήματος Βιολογίας του Πανεπιστημίου Κύπρου, ο οποίος με τη σειρά του συνέβαλε στη πραγματοποίηση αυτής της εργασίας, όχι μόνο με τις γνώσεις του στον τομέα της βιολογίας, αλλά και με τις εισηγήσεις του σε κάθε τυχόν αδιέξοδο.

Πολλές ευχαριστίες, επίσης, στον διδακτορικό φοιτητή του Δρ. Χριστοδούλου, Μιχάλη Αγαθοκλέους, ο οποίος με τις απαράμιλλες, πλέον, γνώσεις του στο παρόν πρόβλημα, βοήθησε στην καλύτερη κατανόηση του αλλά και στην αντιμετώπιση οποιονδήποτε προβλημάτων.

Επιπρόσθετα, θέλω να ευχαριστήσω τους συμφοιτητές μου, Κωνσταντίνο Χαραλάμπους, Αντρέα Διονυσίου αλλά και την Μαρία Μασλιούκοβα, με τους οποίους συνεργαστήκαμε για τα καλύτερα δυνατά αποτελέσματα στο κοινό πρόβλημα που ασχοληθήκαμε.

Τέλος, θέλω να πω ένα μεγάλο ευχαριστώ στην οικογένεια μου, η οποία με έκανε το άτομο το οποίο είμαι σήμερα, που πάντα πίστευε σε μένα και που στάθηκε δίπλα μου σε οτιδήποτε μου επιφύλασσε η ζωή.

Περίληψη

Οι πρωτεΐνες μπορούν να ενταχθούν στα βασικότερα μακροθρεπτικά συστατικά, κατέχοντας πολυδιάστατο ρόλο όσον αφορά την ανάπτυξη και τη συντήρηση όλων των ζώντων οργανισμών. Η γνώση της δομής μιας πρωτεΐνης στον τρισδιάστατο χώρο αποτελεί το κλειδί για τη δημιουργία στοχευμένης θεραπείας, γεγονός ζωτικής σημασίας για την αντιμετώπιση πολλών παθήσεων -μέχρι στιγμής ανίατων- αλλά και για την ικανοποίηση πολλών άλλων αναγκών. Δυστυχώς, οι υφιστάμενες μέθοδοι εξαγωγής της τρισδιάστατης δομής των πρωτεϊνών αποτελούν χρονοβόρες, πολύπλοκες και δαπανηρές διαδικασίες. Γι' αυτό το λόγο, μέχρι σήμερα είμαστε σε θέση να γνωρίζουμε την τρισδιάστατη δομή μόνο ενός μικρού ποσοστού των πρωτεϊνών. Είναι ευρέως γνωστό πως για να γίνει εφικτή η μελέτη της τριτοταγούς δομής μιας πρωτεΐνης, είναι απαραίτητη η γνώση της δευτεροταγούς της δομής, η οποία εξάγεται από την πρωτοταγή, καθώς αυτή καθορίζει τις τοπικές, κανονικές διαμορφώσεις της πολυπεπτιδικής αλυσίδας (α-έλικες, εκτεταμένοι β-κλώνοι). Το πεδίο που αφορά την πρόβλεψη της δευτεροταγούς δομής λαμβάνει συνεχή μελέτη και έρευνα τα τελευταία χρόνια. Καινοτόμες μέθοδοι για την πρόβλεψη των χαρακτηριστικών της δομής των πρωτεϊνών με δεδομένη την αμινοξική τους ακολουθία έχουν ανευρεθεί, όμως αναγνωρίζεται η απόσταση που πρέπει να καλυφθεί, ούτως ώστε η γνώση αυτή να αποτελέσει κτήμα μας στο σύνολό της. Η παρούσα έρευνα αποσκοπεί στη μελέτη και την εφαρμογή Νευρωνικών Δικτύων -και συγκεκριμένα Clockwork νευρωνικών δικτύων ανάδρασης (CW-RNNs)- προκειμένου να επιλυθεί το πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών (Protein Secondary Structure Prediction – PSSP). Συγκεκριμένα, το μοντέλο εκπαιδεύεται, ώστε να δέχεται την πρωτοταγή δομή των πρωτεϊνών (είσοδος του δικτύου) και -κατόπιν επεξεργασίας- να προβλέπει τη δευτεροταγή τους δομή (έξοδος του δικτύου). Το CW-RNN, παρόλο που είναι πολύ πρόσφατο, διαφαίνεται πολλά υποσχόμενο για την επίλυση του προαναφερθέντος ζητήματος, αφού αντιμετωπίζει κάποια προβλήματα από τα οποία πάσχουν άλλα νευρωνικά δίκτυα που χρησιμοποιήθηκαν κατά καιρούς. Στα πλαίσια της παρούσας διπλωματικής εργασίας, έχει διεξαχθεί μια πληθώρα πειραμάτων σε συνδυασμό με την εφαρμογή ensembles και τεχνικών φιλτραρίσματος. Η μέγιστη δυνατή ακρίβεια πρόβλεψης του μοντέλου που παρουσιάζεται έχει υπολογιστεί ως 76.44% για το σύνολο δεδομένων CB513, ποσοστό το οποίο μπορεί να συγκριθεί με αυτά των state-of-the-art μεθόδων.

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή……..1

1.1 Η σημασία και στόχος της έρευνας……2

1.2 Σχετική έρευνα……...5

Κεφάλαιο 2 Βιολογικό Υπόβαθρο...……...…….....14

2.1 Αμινοξέα και πρωτεΐνες..……....15

2.2 Ρόλος και λειτουργίες των πρωτεϊνών…….....18

2.3 Επίπεδα οργάνωσης πρωτεϊνών……...19

Κεφάλαιο 3 Νευρωνικά Δίκτυα…….24

3.1 Η σημασία των τεχνητών νευρωνικών δικτύων……...…25

3.2 Βιολογικός νευρώνας……...27

3.3 Τεχνητός νευρώνας και τεχνητά νευρωνικά δίκτυα……..28

3.3.1 Μοντέλο McCulloh και Pitts……....28

3.3.2 Συναρτήσεις ενεργοποίησης……...…….30

3.3.3 Πολυστρωματικά νευρωνικά δίκτυα perceptron (Multilayer perceptron (MLP)) εμπρόσθιου περάσματος (Feedforward)……...33

3.3.4 Νευρωνικό δίκτυο με ανάδραση (Recurrent Neural Network (RNN))……...34

3.3.5 Clockwork νευρωνικό δίκτυο με ανάδραση (Clockwork RNN (CW-RNN))…….....36

3.4 Αλγόριθμοι μάθησης τεχνητών νευρωνικών δικτύων……..38

3.4.1 Αλγόριθμος μάθησης Perceptron (Perceptron learning algorithm)…….…38

3.4.2 Μέθοδος κατάβασης κλίσης (Gradient Descent (GD))…….....39

3.4.3 Μέθοδος στοχαστικής κατάβασης κλίσης (Stochastic Gradient Descent (SGD))……41

3.4.4 Μέθοδος mini-batch κατάβασης κλίσης (Mini-Batch Gradient Descent (MB-GD))……....42

3.4.5 Adam (Adaptive Moment Estimation) optimizer….....42

3.4.6 Αλγόριθμος μάθησης ανάστροφης μετάδοσης σφάλματος (Backpropagation algorithm)…….....……45
3.4.7 Αλγόριθμος μάθησης ανάστροφης μετάδοσης σφάλματος στο χρόνο (Backpropagation Through Time (BPTT))…….....47
Κεφάλαιο 4 Μεθοδολογία……..49
4.1 Δεδομένα εισόδου……50
4.1.1 Σύνολο δεδομένων CB513……...……....50
4.1.2 Αρχεία πολλαπλής στοίχισης (MSA profiles)……..…52
4.1.3 Δομές δεδομένων εισόδου……...53
4.2 Προσαρμογή υλοποίησης CW-RNN στο πρόβλημα PSSP……...54
4.2.1 Υλοποίηση και αλλαγές…….....54
4.2.2 Αρχιτεκτονική…….……57
4.2.3 Εκπαίδευση του μοντέλου…….....…58
Κεφάλαιο 5 Πειράματα και αποτελέσματα……...61
5.1 Σκοπός των πειραμάτων……..62
5.1.1 Πληροφορίες πειραμάτων……....62
5.1.2 Σημασία πρόβλεψης μεσαίου αμινοξέως…….....64
5.2 Πειράματα για βελτιστοποίηση των παραμέτρων του δικτύου…64
5.2.1 Clock periods…….…..65
5.2.2 Optimizers…….……..68
5.2.3 Συναρτήσεις ενεργοποίησης…….....69
5.2.4 Αριθμός νευρώνων κρυφού επιπέδου…….....71
5.3 10-fold cross-validation……....72
5.3.1 Ensembles……....72
5.3.2 Filtering: Εξωτερικοί κανόνες…….....74
5.3.3 Filtering: Support Vector Machines…….……76
5.4 Χρόνος εκτέλεσης…….....77
Κεφάλαιο 6 Συμπεράσματα και μελλοντική εργασία……..79
6.1 Συμπεράσματα…..…….……80
6.2 Μελλοντική εργασία…….……...85
Αναφορές…...……88
Γενική Βιβλιογραφία……93

Παράρτημα Α……...Α-1
Παράρτημα Β…….Β-10
Παράρτημα Γ…….Γ-12
Παράρτημα Δ…….Δ-13
Παράρτημα Ε…….Ε-17
Παράρτημα ΣΤ……ΣΤ-18

Κεφάλαιο 1

Εισαγωγή

1.2 Η σημασία και στόχος της έρευνας

1.2 Σχετική έρευνα

1.1 Η σημασία και στόχος της έρευνας

Η πρωτεΐνη είναι ένα από τα βασικότερα μακροθρεπτικά συστατικά με πολυσήμαντους ρόλους για την ανάπτυξη και συντήρηση τόσο του ανθρώπινου αλλά και οποιουδήποτε άλλου βιολογικά ζωντανού οργανισμού. Παρά την μεγάλη σημαντικότητα των πρωτεϊνών στον μυϊκό ιστό, αξίζει να σημειωθεί τόσο ο μεταφορικός τους ρόλος σε οξυγόνο όσο και η άμυνα που προσφέρουν στον οργανισμό. Συντελούν ακόμη στην επικοινωνία των κυττάρων, αφού λειτουργούν ως ένζυμα, ορμόνες και υποδοχείς. Συνεπώς, η δομή τους μπορεί στο μέλλον να χρησιμοποιηθεί για την δημιουργία στοχευμένης θεραπείας, γεγονός ζωτικής σημασίας για πολλές παθήσεις.

Η δευτεροταγής δομή μιας πρωτεΐνης περιγράφει κυρίως τις τοπικές κανονικές διαμορφώσεις (α-έλικες, εκτεταμένους β-κλώνους) της πολυπεπτιδικής αλυσίδας στην τρισδιάστατη δομή της. Η τριτοταγής δομή μπορεί να θεωρηθεί σαν η σύνθεση των τοπικών αυτών διαμορφώσεων και η συμπερίληψη όλων των δομικών λεπτομερειών που περιγράφουν την τελική μορφή της πρωτεΐνης στο χώρο αφού παρέχει όλη την απαραίτητη πληροφορία (Χριστοδούλου 2010). Πιο λεπτομερής αναφορά για τα πιο πάνω θα γίνει στο Κεφάλαιο 2.

Οι μέθοδοι που μπορούν να προβλέψουν με ακρίβεια την δευτεροταγή και κατά συνέπεια την τριτοταγή δομή της πρωτεΐνης στο παρόν στάδιο, είναι η κρυσταλλογραφία ακτινών Χ και ο πυρηνικός μαγνητικός συντονισμός (Nuclear Magnetic Resonance - NMR). Δυστυχώς, οι μέθοδοι αυτοί είναι πολύπλοκοι, δαπανηροί και χρονοβόροι (Αγαθοκλέους, 2009). Από την άλλη όμως, η γνώση της πρωτοταγούς δομής μιας πρωτεΐνης μπορεί να γίνει με απλές μεθόδους, αλλά δεν προσφέρει κάποια χρήσιμη πληροφορία για την δευτεροταγή και κατά συνέπεια την τριτοταγή της δομή. Τα τελευταία 50 χρόνια, χρησιμοποιήθηκαν τόσο στατιστικές μέθοδοι, όσο και υπολογιστικές μέθοδοι τεχνητών νευρωνικών δικτύων για την πρόβλεψη της δευτεροταγούς δομής της πρωτεΐνης από την πρωτοταγή της, χωρίς όμως να πετύχουν ακριβής αποτελέσματα. Επιπρόσθετα, είναι γνωστό ότι, οι κατάλληλες συλλογές δεδομένων (δεδομένα εκπαίδευσης και επαλήθευσης), διαδραματίζουν τεράστιο ρόλο στα νευρωνικά δίκτυα. Αυτές οι συλλογές, μεγαλώνουν σε όγκο καθημερινά, με αποτέλεσμα όταν χρησιμοποιούνται, να επιβραδύνουν ακόμη περισσότερο την εκτέλεση των ήδη υφιστάμενων μοντέλων

νευρωνικών δικτύων που χρησιμοποιούνται για αυτό το πρόβλημα. Περισσότερα για τα νευρωνικά δίκτυα θα εξηγηθούν στο Κεφάλαιο 3.

Γνωρίζουμε πως τα αμινοξέα αποτελούνται από μικρότερα μόρια (φορτισμένα + ή -) και λόγω των φορτίων και άλλων παραγόντων αλληλεπιδρούν μεταξύ τους και δημιουργούν την τρισδιάστατη μορφή της πρωτεΐνης. Με τις διάφορες αυτές αλληλεπιδράσεις (μεταξύ των πλευρικών τους αλυσίδων), οι οποίες μπορεί να είναι από οποιοδήποτε προς οποιοδήποτε αμινοξύ της αμινοξικής αλληλουχίας, αναδιπλώνονται στο χώρο και δημιουργούν τη τρισδιάστατη μορφή. Το κάθε αμινοξύ επηρεάζεται από τα γύρω του για το ποια θα είναι η κατηγορία της δευτεροταγούς του δομής. Με βάση τα πιο πάνω, ο αριθμός των πιθανών αλληλεπιδράσεων που μπορεί να υπάρξει είναι πολύ μεγάλος. Επισημαίνουμε ότι, δημιουργείται πάντα η ίδια πρωτεΐνη με την ίδια ακριβείς δομή από κάποια συγκεκριμένη σειρά και αριθμό αμινοξέων, αφού υπάρχουν πάντοτε οι ίδιες αλληλεπιδράσεις μεταξύ αυτών (Αγαθοκλέους 2009). Η πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών (η οποία είναι αλληλένδετα συνδεδεμένη με την τριτοταγή της) από την πρωτοταγή της δομή, συνιστά το πρόβλημα Protein Secondary Structure Prediction (PSSP).

Στόχος μας, σε αυτή την διπλωματική εργασία, είναι η καλύτερη δυνατή πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών από την πρωτοταγή τους δομή, έτσι ώστε να μπορούμε να προβλέψουμε την τριτοταγή τους δομή, αφού γνωρίζοντας την δευτεροταγή δομή μιας πρωτεΐνης, α-έλικες και β-κλώνους, η τριτοταγής της δομή είναι αναδιπλώσεις αυτών. Δηλαδή, η επίλυση του προβλήματος PSSP. Ως αποτέλεσμα, με τη γνώση της τριτοταγούς δομής, θα υπάρξουν τεράστια άλματα τόσο στη φαρμακοβιομηχανία όσο και στην ιατρική, αφού θα μπορούν να αντιμετωπιστούν αυτοάνοσες ασθένειες, όπως για παράδειγμα ο συστηματικός ερυθηματώδης λύκος (ΣΕΛ), που μέχρι τώρα υπάρχει μόνο η δυνατότητα περιορισμού της έκτασης των νόσων αυτών. Συγκεκριμένα, θα μπορούν να βελτιστοποιηθούν τα συμπληρώματα διατροφής και να προληφθούν τυχόν παρενέργειες τους, αλλά και να κατασκευαστούν φάρμακα και αντιβιοτικά, τα οποία ακόμη να δημιουργηθούν, επειδή δεν γνωρίζουμε τη δευτεροταγή δομή συγκεκριμένων πρωτεϊνών που θα βοηθήσουν στην ανακάλυψη τους. Σε πολλούς κλάδους της ιατρικής, η πρόβλεψη της δευτεροταγούς δομής θα θεωρηθεί ως ένα μείζων άλμα, το οποίο θα οδηγήσει στην μακροπρόθεσμη εξέλιξη της φαρμακευτικής μας προσέγγισης.

Σε αυτή την έρευνα, για την πιθανή ικανοποίηση του πιο πάνω στόχου, θα δοκιμαστούν τα Clockwork νευρωνικά δίκτυα (υποκεφάλαιο 3.3.5; Koutník, 2014). Επιλέγοντας τα Clockwork Recurrent Neural Networks (CW-RNNs) αντιμετωπίζουμε καλύτερα το πρόβλημα της εξαφανιζόμενης κλίσης (vanishing gradient). Γνωρίζουμε πως, το πρόβλημα της εξαφανιζόμενης κλίσης, οφείλεται στο ότι η κλίση της συνάρτησης του σφάλματος ως προς την αλλαγή των βαρών αλλάζει εκθετικά, δηλαδή με πολύ μικρό ρυθμό, που έχει ως αποτέλεσμα το δίκτυο να εκπαιδεύεται πολύ δύσκολα. Υποστηρίζεται ότι το πρόβλημα αυτό αντιμετωπίζεται καλύτερα με τα fast/slow τμήματα (modules) τα οποία χρησιμοποιούνται. Δηλαδή, σε συγκεκριμένα χρονικά διαστήματα ενεργοποιούνται συγκεκριμένα τμήματα, έτσι ώστε να παρέχεται πιο χρήσιμη πληροφορία σαν είσοδος στα άλλα τμήματα (τα πιο γρήγορα – μόνο όταν ένα τμήμα είναι πιο αργό από ένα άλλο μπορεί να του στείλει πληροφορία, δηλαδή να υπάρχει σύνδεση από το πιο αργό στο πιο γρήγορο) (Koutnik et al., 2014). Η εκτέλεση συγκεκριμένων τμημάτων σε κάθε χρονική στιγμή, η ύπαρξη λιγότερων συνδέσεων μεταξύ τους, αλλά και η χρήση λιγότερων παραμέτρων, έχουν ως συνέπεια τη μεγάλη μείωση του χρόνου εκτέλεσης του μοντέλου.

Εξαιτίας του συγκεκριμένου προβλήματος, χρησιμοποιώντας την πιο πάνω αρχιτεκτονική, η οποία εξηγείται περεταίρω στα υποκεφάλαια 1.2 και 3.3., με την ανάθεση των κατάλληλων clock speeds στα τμήματα, στο δίκτυο θα εισέρχεται περισσότερη πληροφορία για ένα αμινοξύ. Ως αποτέλεσμα, το δίκτυο αυτό, θα συσχετίζει το συγκεκριμένο αμινοξύ με αυτά που βρίσκονται γύρω του (πριν και μετά από αυτό), τα οποία πιθανότατα να επηρεάζουν τη δομή της πρωτεΐνης στο συγκεκριμένο σημείο. Κατά συνέπεια, το δίκτυο θα εκπαιδεύεται όσο το δυνατόν καλύτερα και θα μπορεί να προβλέπει την δευτεροταγή δομή των πρωτεϊνών με μεγάλο ποσοστό επιτυχίας.

Στη συνέχεια, θα αναφερθούμε σε διάφορες μεθόδους που χρησιμοποιήθηκαν για το συγκεκριμένο πρόβλημα και στο κεφάλαιο 2 και 3 θα εξηγηθούν λεπτομερώς το βιολογικό υπόβαθρο αλλά και τα νευρωνικά δίκτυα αντίστοιχα. Στο κεφάλαιο 4 θα γίνει μια περιγραφή για τα δεδομένα που χρησιμοποιήθηκαν και για την μεθοδολογία που ακολουθήθηκε. Τα πειράματα που έγιναν και τα αποτελέσματα τους θα παρουσιαστούν στο κεφάλαιο 5. Στο κεφάλαιο 6 εξάγονται τα συμπεράσματα και αναφέρονται ιδέες για

μελλοντική εργασία και έρευνα. Τέλος, στο Παράρτημα Α, Β, Γ, Δ και Ε υπάρχουν πληροφορίες σχετικά με τον κώδικα που χρησιμοποιήθηκε, ενώ στο Παράρτημα ΣΤ αναφέρονται επιπλέον clock period σύνολα που δοκιμάστηκαν κατά την εκτέλεση των πειραμάτων.

1.2 Σχετική έρευνα

Όπως έχει ήδη αναφερθεί στο υποκεφάλαιο 1.1, η έρευνα για την πρόβλεψη της δευτεροταγούς δομής της πρωτεΐνης με τη χρήση τόσο στατιστικών όσο και υπολογιστικών μεθόδων, όπου και κατατάσσονται τα νευρωνικά δίκτυα, άρχισε πριν 5 δεκαετίες. Είναι φανερό πως υπάρχει συνεχής πρόοδος στο συγκεκριμένο πρόβλημα, επειδή είναι απαράμιλλης σημασίας και η λύση του θα επιφέρει πολλά θετικά στην ανθρώπινη ζωή, αλλά και γιατί με τη συνεχή μελέτη και έρευνα, ανακαλύπτονται νέες μέθοδοι που συνεχώς βελτιώνουν τα ποσοστά επιτυχίας (Yang et al., 2016).

Σε αυτό το χρονικό διάστημα, έχουν δημιουργηθεί, εφαρμοστεί και τροποποιηθεί πολλοί αλγόριθμοι μάθησης. Η αξιολόγηση των αλγορίθμων για το συγκεκριμένο πρόβλημα έγινε με τον έλεγχο αμινοξέως προς αμινοξύ για μια άγνωστη πρωτεΐνη για το δίκτυο. Δηλαδή, υπολογίζεται το ποσοστό επιτυχίας Q3 (Εξίσωση 1.1; Richards and Kundrot, 1988), με βάση τον αριθμό των αμινοξικών καταλοίπων που προβλέπονται ορθά, σε σχέση με τον ολικό αριθμό των αμινοξικών καταλοίπων που αποτελούν τις άγνωστες πρωτεϊνικές ακολουθίες. Κάποιες από αυτές τις μεθόδους υπολογίζουν και το ποσοστό επιτυχίας Q8. Δηλαδή, προβλέπουν και τις 8 κατηγορίες της δευτεροταγής δομής που αναφέρονται στον Πίνακα 1 στο υποκεφάλαιο 2.1.3. Παράλληλα με τα πιο πάνω

$$Q_3 = \frac{Rcorrect}{TotalR} * 100$$

Εξίσωση 1.1: Το ποσοστό επιτυχίας αμινοξέως προς αμινοξύ για τις τρεις ευρείς κλάσεις Η (έλικες), Ε (κλώνοι), C (coil) της δευτεροταγής δομής των πρωτεϊνών.

Rcorrect : αριθμός αμινοξικών καταλοίπων που έχουν προβλεφθεί ορθά;
TotalR : συνολικός αριθμός αμινοξικών καταλοίπων

υπολογιζόταν και το ποσοστό επιτυχίας SOV[1] (Zemla et al., 1999). Το θεωρητικό όριο στο οποίο μπορεί να φτάσει το πιο πάνω ποσοστό επιτυχίας Q3, έχει ήδη υπολογιστεί και κυμαίνεται μεταξύ 88 και 90% όπως αναφέρουν και δικαιολογούν ο Yang και οι συνεργάτες του (2016).

Στατιστικές μέθοδοι

Οι στατιστικές μέθοδοι προβλέπουν τη δευτεροταγή δομή μιας πρωτεϊνικής ακολουθίας με βάση κάποιες στατιστικές πληροφορίες και την σύνταξη κάποιων κανόνων πρόβλεψης. Συγκεκριμένα, χρησιμοποιούνται στατιστικές πιθανότητες για την ανάπτυξη εμπειρικών κανόνων για το σκοπό αυτό. Η στατιστική πληροφορία που συγκεντρώνεται, μπορεί να περιγραφεί ως η πιθανότητα των διάφορων αμινοξικών καταλοίπων να είναι τοποθετημένα σε συγκεκριμένες στερεοδιατάξεις στην πρωτεϊνική δομή. Το ποσοστό επιτυχίας της κάθε στατιστικής μεθόδου, εξαρτάται, όχι μόνο από το μέγεθος και την ποιότητα των στερεοδομών που θα χρησιμοποιηθούν για τη συγκέντρωση της πιο πάνω πληροφορίας, αλλά και με το πως θα χρησιμοποιηθεί για την εξαγωγή των κανόνων που θα χρησιμοποιηθούν για την πρόβλεψη της δευτεροταγούς δομής.

Η πιο αντιπροσωπευτική, από αυτή την κατηγορία μεθόδων, είναι η Chou-Fasman (Chou and Fasman, 1974), όπου αντιστοίχησαν στα 20 κατάλοιπα κάποιες στερεοδιαταξικές παραμέτρους, που εκφράζουν ποια η πιθανότητα κάθε ένα από αυτά να εμφανιστεί σαν μέρος της δευτεροταγούς δομής (α-έλικας, εκτεταμένη δομή ή στροφή). Το ποσοστό επιτυχίας Q3 αυτής της μεθόδου για μια άγνωστη πρωτεΐνη κυμαινόταν μεταξύ 50 και 60%.

Στη συνέχεια, παρουσιάστηκε η μέθοδος Garnier-Osguthorpe-Robson (GOR), η οποία συνδυάζει στατιστικές μεθόδους με την τεχνική του κοντινότερου γείτονα (Garnier et al., 1978). Για την πρόβλεψη της δευτεροταγούς δομής των πρωτεϊνών, συνδυάζει στατιστική πληροφορία και διάφορους θεωρητικούς αλγορίθμους (Yang et al., 2016). Συγκεκριμένα, όπως αναφέρθηκε και πιο πάνω, με την ανάλυση των στερεοδιατάξεων

[1]SOV(Segment OVerlap): μέθοδος βαθμολόγησης της προβλεπόμενης ακολουθίας της δευτεροταγούς δομής, όπου συγκρίνει διαστήματα από κατάλοιπα που επικαλύπτονται στις δύο ακολουθίες της προβλεπόμενης και πραγματικής ακολουθίας αντίστοιχα

των πρωτεϊνών, συγκεντρώνονται στατιστικές πληροφορίες, δηλαδή κάποιες πιθανότητες, για τις κατηγορίες H (έλικες), E (κλώνοι), C (στροφές). Αυτή η μέθοδος, χρησιμοποιώντας επίσης και την τεχνική του κοντινότερου γείτονα (Yi and Lander, 1993) με ένα παράθυρο 17 αμινοξέων (Zvelebil and Baum, 2008), υπολογίζει τις πιθανότητες για κάθε αμινοξικό κατάλοιπο να ανήκει σε μια συγκεκριμένη δευτεροταγή δομή, λαμβάνοντας υπόψη και τα γειτονικά αμινοξικά κατάλοιπα. Το αρχικό ποσοστό επιτυχίας Q3 αυτής της μεθόδου (GOR) για μια άγνωστη πρωτεΐνη υπολογίζεται στο 64.4%, αλλά μετά από διάφορες, συνεχείς βελτιώσεις (GOR V) κατάφερε να αγγίξει το 73.5% για ποσοστό επιτυχίας Q3 και 0.707 για SOV (Χριστοδούλου, 2010).

Υπολογιστικές μέθοδοι

Στην κατηγορία των υπολογιστικών μεθόδων κατατάσσονται τα τεχνητά νευρωνικά δίκτυα (περισσότερες πληροφορίες στο Κεφάλαιο 3), τα οποία αναδείχθηκαν αποδοτικά εργαλεία για το συγκεκριμένο πρόβλημα, αφού έχουν πετύχει τα καλύτερα αποτελέσματα μέχρι στιγμής. Αυτό οφείλεται στην δυνατότητα τους να μπορούν να ανακαλύπτουν συσχετίσεις δεύτερης και μεγαλύτερης τάξης στα δεδομένα εισόδου, σε αντίθεση με τις στατιστικές μεθόδους, οι οποίες ανακαλύπτουν μόνο συσχετίσεις πρώτης τάξης. Επίσης, κάποιες από αυτές τις μεθόδους, είχαν χρησιμοποιήσει εξελικτική πληροφορία των πρωτεϊνών, η οποία πηγάζει από την πολλαπλή στοίχιση ομόλογων πρωτεϊνικών ακολουθιών (Rost and Sander, 1993; Zvelebil et al., 1987). Πιο κάτω, θα αναφερθούν με χρονολογική σειρά κάποιες από αυτές τις μεθόδους.

Αρχικά, έχουμε τη μέθοδο των Qian και Sejnowski (1988), οι οποίοι ανέπτυξαν ένα πλήρως συνδεδεμένο νευρωνικό δίκτυο εμπρόσθιου περάσματος, το οποίο περιείχε ένα κρυφό επίπεδο. Η μέθοδος αυτή χρησιμοποιούσε ένα παράθυρο εισόδου με μέγεθος 13 αμινοξέα ορθογώνιας κωδικοποίησης, δηλαδή χρησιμοποιούνταν 20 αριθμοί για να περιγράψουν την κατηγορία του αμινοξέως για ένα πρωτεϊνικό κατάλοιπο (Lin et al., 2001). Το δίκτυο αυτό προέβλεπε την κατηγορία (H, E, C) της δευτεροταγούς δομής του κεντρικού αμινοξέως του παραθύρου. Υπήρχε, επίσης, ένα δεύτερο δίκτυο σε αυτή την μέθοδο, όπου βελτίωνε τα αποτελέσματα του πρώτου (Αγαθοκλέους, 2009). Παρά το πρόβλημα υπερεκπαίδευσης της μεθόδου, το ποσοστό επιτυχίας Q3 κατάφερε να φτάσει μέχρι 63.30%.

Διάδοχος του πιο πάνω νευρωνικού δικτύου, ήταν το μοντέλο PHD (Profile network from HeiDelberg) και συγκεκριμένα το PHDsec (Rost and Sander, 1993), το οποίο χρησιμοποιούσε την τεχνική της πολλαπλής στοίχισης πρωτεϊνικών ακολουθιών για την αξιοποίηση εξελικτικής πληροφορίας και με συγκεκριμένες μεθόδους μείωνε το πρόβλημα της υπερεκπαίδευσης. Ως αποτέλεσμα, κατάφερε να αυξήσει το ποσοστό επιτυχίας σε 71.40%.

Ακολούθως, το 1996, οι King και Sternberg, δημιούργησαν το DSC (Discrimination of protein Secondary structure Class) το οποίο είχε ποσοστό επιτυχίας Q3 ίσο με 71.95%. Χρησιμοποιώντας κατηγοριοποίηση, δημιουργεί ομάδες όπου αναθέτει τα αποτελέσματα εξόδου του νευρωνικού αυτού δικτύου και με κάποιους υπολογισμούς από γραμμικές, στατιστικές μεθόδους προσπαθεί να συγκλίνει στην ακριβή δευτεροταγή δομή των άγνωστων πρωτεϊνικών ακολουθιών.

Στη συνέχεια, το δίκτυο NNSSP (Salamov and Solovyev, 1997), ομαδοποιεί πρωτεϊνικές ακολουθίες με βάση τα κοινά τους χαρακτηριστικά, με τη χρήση της τεχνικής του κοντινότερου γείτονα, και τις συγκρίνει με άλλες ακολουθίες δευτεροταγούς δομής. Αρχικά, το δίκτυο αυτό πέτυχε ποσοστό επιτυχίας 68.41%, ενώ μετά τη χρήση της πολλαπλής στοίχισης ακολουθιών κατάφερε να αυξηθεί στο 73.50% (Χριστοδούλου, 2010).

Το 1999, ο Baldi και οι συνεργάτες του κατάφεραν να σχεδιάσουν και να υλοποιήσουν ένα δίκτυο αμφίδρομης ανάδρασης (Bidirectional Recurrent Neural Network (BRRN)), το οποίο είχε επιφέρει ένα από τα καλύτερα αποτελέσματα για εκείνη την χρονική περίοδο (Baldi et al., 1999, 2000). Συγκεκριμένα, το BRNN κατάφερε να φτάσει σε ποσοστό επιτυχίας 73.6% με τη χρήση πολλαπλής στοίχισης ακολουθιών και 76% με το συνδυασμό έξι τέτοιων δικτύων. Το δίκτυο αυτό, προσπαθεί να προβλέψει το μεσαίο στοιχείο-αμινοξύ από ένα κινητό παράθυρο. Το κινητό παράθυρο απαρτίζεται από αμινοξέα και εισέρχεται ως είσοδος στο δίκτυο. Ως αποτέλεσμα, προβλέποντας το μεσαίο αμινοξύ, το δίκτυο εκμεταλλεύεται καλύτερα την πληροφορία που προσφέρουν τα γειτονικά του αμινοξέα, λόγω της μεταξύ τους αλληλεπίδρασης. Με αυτό τον τρόπο,

συγκεντρώνεται περισσότερη πληροφορία για το αμινοξύ που προσπαθεί να προβλέψει το δίκτυο για μια πιο ακριβή πρόβλεψη.

Σημαντικό εδώ να αναφέρουμε ότι το 2001 άρχισαν να υλοποιούνται και να δοκιμάζονται μοντέλα, χρησιμοποιώντας Support Vector Machines (SVM; Ward et al., 2003; Kim and Park, 2003), για το συγκεκριμένο πρόβλημα (Χριστοδούλου, 2010).

Ακολούθως, το 2002, υλοποιήθηκαν τα νευρωνικά δίκτυα SSpro και SSpro8, ως web servers (Pollastri et al., 2002). Για αυτές τις υλοποιήσεις δοκιμάστηκαν διάφορα σύνολα δεδομένων και χρησιμοποιήθηκε ως βασική αρχιτεκτονική το BRNN. Το SSpro προσπαθούσε να προβλέψει τις τρεις ευρείς κλάσεις της δευτεροταγούς δομής, ενώ το SSpro8 και τις 8, όπως περιγράφονται στον Πίνακα 1. Το ποσοστό επιτυχίας Q3, με το μοντέλο SSpro, αυξήθηκε στο 78%, ενώ το μοντέλο SSpro8 είχε ποσοστό επιτυχίας Q8 το οποίο κυμαινόταν μεταξύ 62.6% και 63.3%.

Οι Pollastri και McLysaght, το 2005, δημιούργησαν τον Porter, ο οποίος ήταν και αυτός ένας web server και έδωσε καλύτερα αποτελέσματα από τους πιο πάνω. Όπως παρατηρούμε και στο άρθρο τους (Pollastri and McLysaght, 2005), βασίζεται και αυτό το σύστημα στο BRNN, χρησιμοποιώντας πολλαπλή στοίχιση ακολουθιών και φιλτράρισμα από νευρωνικά δίκτυα ανάδρασης (Recurrent Neural Networks (RNN)). Το ποσοστό επιτυχίας του Porter ήταν 79%.

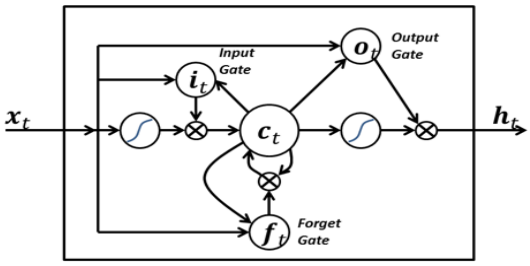
Ο αλγόριθμος μηχανικής μάθησης LAD (Logical Analysis of Data) υλοποιήθηκε το 2005 και είχε ως στόχο την ανάλυση, σε περισσότερο βάθος, των ιδιοτήτων των αμινοξέων, για την συγκέντρωση περισσότερων πληροφοριών (Blazewicz et al., 2005). Αυτή η μέθοδος, βοήθησε στην εξαγωγή συμπερασμάτων για τις αλληλεπιδράσεις μεταξύ των αμινοξέων και πως οι ιδιότητες τους επηρεάζουν την κατηγορία τους στην δευτεροταγή δομή. Το ποσοστό επιτυχίας της ήταν 70.6%.

Εκτός από τις πιο πάνω υλοποιήσεις, χρησιμοποίησαν και οι Chen και Chaudhari (2007) το BRNN. Συγκεκριμένα, εξήγησαν πόσο σημαντικές είναι οι μακρινές εξαρτήσεις μεταξύ των αμινοξέων για την αναδίπλωση της πρωτεΐνης και πως προσπάθησαν να τις συμπεριλάβουν στο μοντέλο που πρότειναν. Το μοντέλο αυτό, συμπεριλάμβανε δύο

επίπεδα BRNN (Cascaded Bidirectional Recurrent Neural Network), όπου το δεύτερο επίπεδο έπαιρνε σαν είσοδο την έξοδο του πρώτου για να φιλτράρει τα αποτελέσματα του. Το Q3 ποσοστό επιτυχίας αυτής της μεθόδου ήταν 74.38%, ενώ το SOV της 0.66.
Οι Kountouris και Hirst (2009), με το μοντέλο DISSPRED (DIhedral angles and Secondary Structure PREDiction), προσπάθησαν να προσεγγίσουν το πρόβλημα με χρήση δύο SVMs, όπου το ένα υπολόγιζε την κατηγορία του αμινοξέως στη δευτεροταγή του δομή, ενώ το άλλο πρόβλεπε την κατηγορία δίεδρης γωνίας στην οποία αυτό ανήκει. Οι δίεδρες γωνίες συνδέονται αλληλένδετα με την δευτεροταγή δομή ενός αμινοξέως, αφού είναι οι γωνίες που χαρακτηρίζουν τα επίπεδα της κοινής ραχοκοκαλιάς από την οποία αποτελούνται τα αμινοξέα. Συνδυάζοντας τα αποτελέσματα από τα δύο SVMs ενισχύεται η είσοδος της επόμενης τους εκτέλεσης. Αυτό το μοντέλο κατάφερε να φτάσει σε Q3 ποσοστό επιτυχίας 80%.

Ο Porter web server, ο οποίος αναφέρθηκε πιο πάνω, αναβαθμίστηκε το 2013 σε Porter 4.0, ενώ ταυτόχρονα αναβαθμίστηκε και ο PaleAle σε PaleAle 4.0 (Mirabello and Pollastri, 2013). Οι κύριες διαφορές με το προηγούμενα μοντέλα είναι ότι στην νέα έκδοση το μέγεθος των συνόλων δεδομένων εκπαίδευσης αυξήθηκε κατά μεγάλο βαθμό, οι παράμετροι των δικτύων για να μπορούν να υποστηρίζουν τον αυξημένο αριθμό παραδειγμάτων σχεδόν διπλασιάστηκε και η διαδικασία εκπαίδευσης γινόταν σε μεγαλύτερο βάθος. Ο Porter 4.0 αύξησε το ποσοστό επιτυχίας Q3 του Porter σε 82.2%.

Μια από τις πιο πρόσφατες μεθόδους, το SPIDER3 (Structural Property prediction with Integrated DEep neuRal network), δεν προσπαθεί μόνο να προβλέψει τη δευτεροταγή δομή μιας άγνωστης πρωτεΐνης, αλλά και να αντιμετωπίσει άλλα προβλήματα παρόμοιου τύπου (Heffernan et al., 2017). Το ενδιαφέρον μας επικεντρώνεται στο PSSP πρόβλημα, γι' αυτό θα αναφερθούμε μόνο σε αυτό. Σε αυτό το σύστημα χρησιμοποιήθηκαν τα Long Short-Term Memory (LSTM) BRNNs, τα οποία είναι ικανά να συλλάβουν μακρινές εξαρτήσεις χωρίς τη χρήση της τεχνικής του παραθύρου. Το LSTM δίκτυο, είναι ένα RNN δίκτυο όπου στο κρυφό επίπεδο υπάρχουν LSTM units, όπως παρουσιάζονται στο Σχήμα 1.1 (Hochreiter and Schmidhuber ,1997). Αποτελούνται από ένα cell, ένα input gate, ένα output gate και ένα forget gate.



Σχήμα 1.1: Ένα LSTM unit

Πάρθηκε από: Wikimedia Commons contributors, "File:Peephole Long Short-Term Memory.svg," Wikimedia Commons, the free media repository, https://commons.wikimedia.org/w/index.php?title=File:Peephole_Long_Short-Term_Memory.svg&oldid=247762287 (accessed April 21, 2018)

Το δίκτυο είναι ικανό να 'θυμάται' τιμές, είτε για μεγάλες, είτε για μικρές χρονικές περιόδους, με αποτέλεσμα να μπορεί να γεφυρώσει το κενό μεταξύ σημείων μεγάλων ακολουθιών που σχετίζονται μεταξύ τους. Οι συγγραφείς αυτού του άρθρου, συγκρίνουν τα αποτελέσματα αυτού του δικτύου με το προηγούμενο μοντέλο - SPIDER2 (Heffernan et al., 2015) -, όπου και επισημάνουν ακριβώς τις βελτιώσεις.

Τονίζουν, επίσης, ότι με τη χρήση του συγκεκριμένου συστήματος, για την πρόβλεψη των δομικών ιδιοτήτων της πρωτεΐνης, η μεγαλύτερη βελτίωση υπήρξε για τα αμινοξικά κατάλοιπα, σε σχέση με την προηγούμενη μέθοδο, τα οποία μεταξύ τους είχαν την μεγαλύτερη απόσταση. Το μοντέλο SPIDER3 έφτασε σε 84% ποσοστό επιτυχίας Q3.

Τα παραπάνω ποσοστά επιτυχίας Q3, Q8 ή SOV, είναι δύσκολο να χρησιμοποιηθούν για την άμεση σύγκριση των μεθόδων, λόγω του ότι αυτές οι μέθοδοι μπορεί να χρησιμοποίησαν διαφορετικά δεδομένα εισόδου, τόσο για εκπαίδευση, όσο και για επαλήθευση. Με τη χρήση μεγαλύτερων συνόλων δεδομένων κάποιος αλγόριθμος μάθησης μπορεί να εκπαιδευτεί σε μεγαλύτερο βαθμό, με αποτέλεσμα να μπορεί να προβλέψει πιο ορθά τις άγνωστες, προς αυτόν, πρωτεϊνικές ακολουθίες.

Σχετική έρευνα Πανεπιστημίου Κύπρου

Από προηγούμενες διπλωματικές εργασίες (και συγκεκριμένα Αγαθοκλέους 2009, Χριστοδούλου 2010, Παυλίδης 2016) μπορούμε να παρατηρήσουμε ότι ο τομέας των νευρωνικών δικτύων και οι υπολογιστικές μέθοδοι, είναι πολύ υποσχόμενες για το συγκεκριμένο πρόβλημα (πρόβλεψη δευτεροταγούς δομής πρωτεϊνών (PSSP)).

Στην έρευνα του, ο Αγαθοκλέους (2009) ανέπτυξε νευρωνικό δίκτυο αμφίδρομης ανάδρασης (BRNN), το οποίο βασίστηκε στην αρχιτεκτονική που εισήγαγε στον χώρο ο Baldi (Baldi et al., 1999, 2000), όπως ήδη αναφέρθηκε και πιο πάνω, και εκπαιδεύτηκε με τον αλγόριθμο ανάστροφης μετάδοσης λάθους για να προβλέπει τη δευτεροταγή δομή πρωτεϊνών παίρνοντας σαν είσοδο μόνο την πρωτοταγή τους δομή (το δίκτυο αυτό είχε μέχρι 64% ποσοστού επιτυχίας Q3). Ο Αγαθοκλέους είχε τονίσει πως το ποσοστό αυτό είναι χαμηλό αλλά επισήμανε πως υπάρχουν προοπτικές βελτίωσης, αφού το δίκτυο είναι φανερό πως, μέσα από τα αποτελέσματα και την ανάλυση τους, μαθαίνει. Τέλος, συμπέρανε ότι δύσκολα εντοπίζεται το ολικό ελάχιστο, το οποίο θα έδινε τα καλύτερα ποσοστά επιτυχίας, λόγω του μεγάλου όγκου δεδομένων.

Στη συνέχεια, η Χριστοδούλου (2010), στη διπλωματική της εργασία, διερεύνησε μεθόδους εκπαίδευσης νευρωνικών δικτύων αμφίδρομης ανάδρασης για το πρόβλημα αυτό. Κρατώντας σταθερή την αρχιτεκτονική του συστήματος που ανέπτυξε ο Αγαθοκλέους, διαμόρφωσε τον αλγόριθμο μάθησης και τον τρόπο προσαρμογής των βαρών του δικτύου. Ως αποτέλεσμα, το σύστημα επεξεργαζόταν περισσότερη πληροφορία όσον αφορά μια συγκεκριμένη πρωτεΐνη κάθε χρονική στιγμή. Εφαρμόζοντας την μέθοδο της πολλαπλής στοίχισης πρωτεϊνών για την κωδικοποίηση των δεδομένων εισόδου είχε ποσοστό επιτυχίας Q3 μέχρι 74%. Χρησιμοποιώντας την μέθοδο WTA (Winner Takes All) και αλλάζοντας την αρχιτεκτονική, το SOV αυξήθηκε μέχρι 0.65 και το Q3 μέχρι 76%. Τέλος, χρησιμοποίησε Hidden Markov Models (HMM) για φιλτράρισμα της εξόδου, όπου το SOV αυξήθηκε μέχρι 0.70.

Στην προσπάθεια επίλυσης του συγκεκριμένου προβλήματος, ο Παυλίδης (2016) προσπάθησε να το προσεγγίσει με τη χρήση συνελικτικών νευρωνικών δικτύων (Convolutional Neural Networks (CNNs)). Το δίκτυο CNN, όπως είχε αναφέρει, ανήκει στην οικογένεια των βαθιών νευρωνικών δικτύων (Deep Neural Networks) και έχει υποσχόμενα αποτελέσματα σε προβλήματα που σχετίζονται με την αναγνώριση χαρακτηριστικών. Αυτό γιατί, τέτοια δίκτυα, όπως είχε επισημάνει, εκμεταλλεύονται τη χωρική δομή των δεδομένων εισόδου και θεωρητικά κάνουν καλύτερη διαχείριση των δεδομένων εισόδου σε προβλήματα που έχουν να κάνουν με αλληλουχίες ή γενικότερα αυτά που λαμβάνουν ως παράμετρο το χώρο. Προσπάθησε με την οπτικοποίηση των

δεδομένων εισόδου, δηλαδή της αλληλουχίας αμινοξέων, να εκπαιδεύσει το δίκτυο για να βρίσκει χαρακτηριστικά που να μας δίνουν την πιθανή κατηγορία που ανήκει το δεδομένο (αμινοξύ). Η καλύτερη κατηγοριοποίηση που δόθηκε από το δίκτυο του, είχε ποσοστό ακριβείας Q3 42.89% .

Με βάση τα πιο πάνω, είναι φανερό ότι μέσα από την βελτίωση (Χριστοδούλου, 2010), σε διάφορους τομείς, του συστήματος που ανέπτυξε ο Αγαθοκλέους (2009) το ποσοστό επιτυχίας αυξήθηκε αρκετά. Ως αποτέλεσμα, τα νευρωνικά δίκτυα, με την κατάλληλη αρχιτεκτονική και χρήση συγκεκριμένων μεθόδων και τεχνικών, έχουν την ικανότητα να δώσουν πολύ καλά αποτελέσματα στο πρόβλημα της πρόβλεψης της δευτεροταγούς δομής των πρωτεϊνών.

Κεφάλαιο 2

Βιολογικό Υπόβαθρο

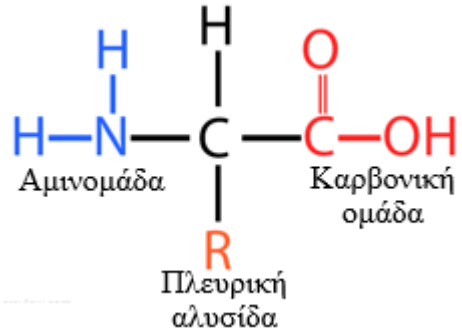
2.1 Αμινοξέα και πρωτεΐνες

2.2 Ρόλος και λειτουργίες των πρωτεϊνών

2.3 Επίπεδα οργάνωσης πρωτεϊνών

2.1 Αμινοξέα και πρωτεΐνες

Τα αμινοξέα είναι χημικές ενώσεις οι οποίες αποτελούνται από ένα άτομο άνθρακα, το οποίο συνδέεται ομοιοπολικά με μια αμινομάδα (-NH2), μια καρβοξυλομάδα (-COOH), μια πλευρική αλυσίδα R και ένα άτομο υδρογόνου (Σχήμα 2.1). Η πλευρική αυτή αλυσίδα τα διαφοροποιεί ως προς τις ιδιότητες τους, αφού σε κάθε αμινοξύ είναι διαφορετική. Η ποικιλομορφία μπορεί να εντοπιστεί, αναλύοντας το μέγεθος, το σχήμα, το φορτίο, την ικανότητα δέσμευσης υδρογόνου και τη χημική δραστικότητα.



Σχήμα 2.1: Δομή αμινοξέως

Πάρθηκε από: http://www.nutrientsreview.com/proteins/amino-acids

Ανάλογα με τις ιδιότητες κάθε πλευρικής αλυσίδας τα αμινοξέα χωρίζονται σε τέσσερις κατηγορίες. Αναλυτικότερα, υπάρχουν οι βασικές πλευρικές ομάδες -λόγω της ύπαρξης της αμινομάδας (-NH2)- που προσδίδουν στο αμινοξύ θετικό φορτίο. Αντίστοιχα, οι όξινες πλευρικές αλυσίδες έχουν αρνητικό φορτίο λόγω του καρβοξυλίου (-COOH). Επίσης, υπάρχουν τα πολικά αμινοξέα στις πλευρικές τους αλυσίδες, τα οποία εμφανίζουν πολικές ομάδες, όπως για παράδειγμα την υδροξυλομάδα (-OH) και την καρβονυλομάδα (=O). Τέλος, υπάρχουν και τα μη πολικά αμινοξέα, των οποίων η πλευρική αλυσίδα αποτελείται μόνο από C και H (με εξαίρεση την κυστεΐνη και τη μεθειανίνη, όπου υπάρχει και S-θείο).

Κάθε αμινοξύ συμβολίζεται είτε με μια τριάδα (τριπλέτα) του γενετικού κώδικα DNA είτε με ένα γράμμα (Σχήμα 2.2). Τοποθετώντας τις τριπλέτες ή τα γράμματα αυτά σε μια συμβολοσειρά, δημιουργείται η αμινοξική ακολουθία ενός πολυπεπτιδίου (πρωτοταγής δομή). Τα πολυπεπτίδια, δηλαδή, προκύπτουν από την ομοιοπολική ένωση των αμινοξέων μεταξύ τους με πεπτιδικούς δεσμούς. Συνδυάζοντας ένα ή περισσότερα πολυπεπτίδια, καταλήγουμε σε μεγάλα βιολογικά μακρομόρια, τις πρωτεΐνες. Παρόλο

που υπάρχουν αρκετά αμινοξέα, μόνο 20 εξ αυτών, τα πρωτεϊνικά αμινοξέα, χρησιμοποιούνται στην κατασκευή των περισσότερων πρωτεϊνών.

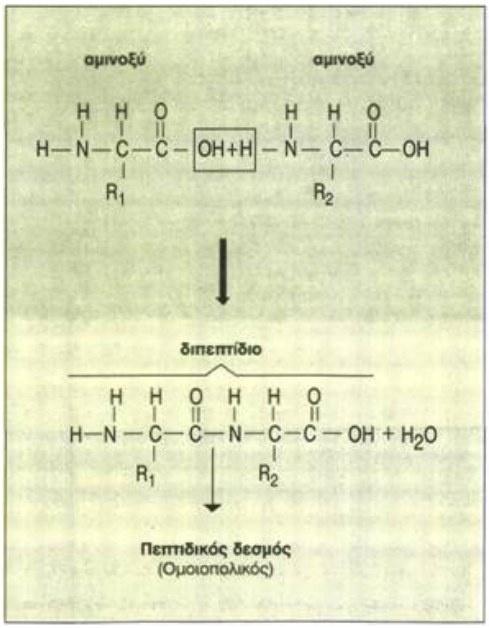
Η διαδικασία για την ένωση δύο αμινοξέων ονομάζεται «συμπύκνωση» και κατά τη διάρκεια της το ένα αμινοξύ αποβάλει ένα άτομο υδρογόνου (H), από την αμινομάδα του (NH2), ενώ το άλλο αποβάλει μια υδροξυλομάδα (OH), από την καρβοξυλομάδα του (COOH). Από τη συγκεκριμένη αντίδραση παράγεται ένα μόριο νερού (H2O) και ένα διπεπτίδιο, το οποίο αποτελεί αποτέλεσμα της ένωσης των δύο αμινοξέων με ένα πεπτιδικό (ομοιοπολικό) δεσμό (-CO-NH-) (Σχήμα 2.3). Με τη βοήθεια του μηχανισμού συμπύκνωσης, λοιπόν, μπορούν να δημιουργηθούν τα μακρομόρια.

Αμινοξύ	*Συντομογραφία τριών γραμμάτων*	*Συντομογραφία ενός γράμματος*	*Αμινοξύ*	*Συντομογραφία τριών γραμμάτων*	*Συντομογραφία ενός γράμματος*
Αλανίνη	Ala	A	Κυστεΐνη	Cys	C
Αργινίνη	Arg	R	Λευκίνη	Leu	L
Ασπαραγίνη	Asn	N	Λυσίνη	Lys	K
Ασπαραγινικό οξύ	Asp	D	Μεθειονίνη	Met	M
Βαλίνη	Val	V	Προλίνη	Pro	P
Γλουταμίνη	Gln	Q	Σερίνη	Ser	S
Γλουταμινικό οξύ	Glu	E	Τυροσίνη	Tyr	Y
Γλυκίνη	Gly	G	Φαινυλαλανίνη	Phe	F
θρεονίνη	Thr	T			
θρυπτοφάνη	Trp	W			
Ισολευκίνη	Ile	I			
Ιστιδίνη	His	H			

Σχήμα 2.2: Τρόποι κωδικοποίησης αμινοξέων

Πάρθηκε από:
http://eclass.uth.gr/eclass/modules/document/file.php/SEYB130/ΒΙΒΛΙΟ_ΚΕΦ_3_ΔΟΜΗ%20ΚΑΙ%20ΛΕΙΤΟΥΡΓΙΑ%20ΤΩΝ%20ΠΡΩΤΕΙΝΩΝ.pdf

Παρατηρούμε στο σχήμα 2.3, όπως προαναφέρθηκε, ότι η πλευρική αλυσίδα R διαφέρει σε κάθε αμινοξύ, ενώ ο υπόλοιπος «σκελετός» του αμινοξέως είναι ο ίδιος. Οι πλευρικές αλυσίδες των αμινοξέων αυτών (R1 και R2) αντιπροσωπεύουν τις διαφορετικές ιδιότητες του κάθε αμινοξέως. Βάσει αυτών των ιδιοτήτων, διαμορφώνεται ανάλογα η όψη της τριτοταγούς δομής της πρωτεΐνης στον χώρο, κάτι που αποτελεί κομβικό σημείο για την συγκεκριμένη έρευνα και κάτι που θα εξηγήσουμε στη συνέχεια. Είναι σημαντικό να τονιστεί ότι για τη δημιουργία της τριτοταγούς δομής προαπαιτείται ο σχηματισμός της δευτεροταγούς από την πρωτοταγή δομή της πρωτεΐνης.

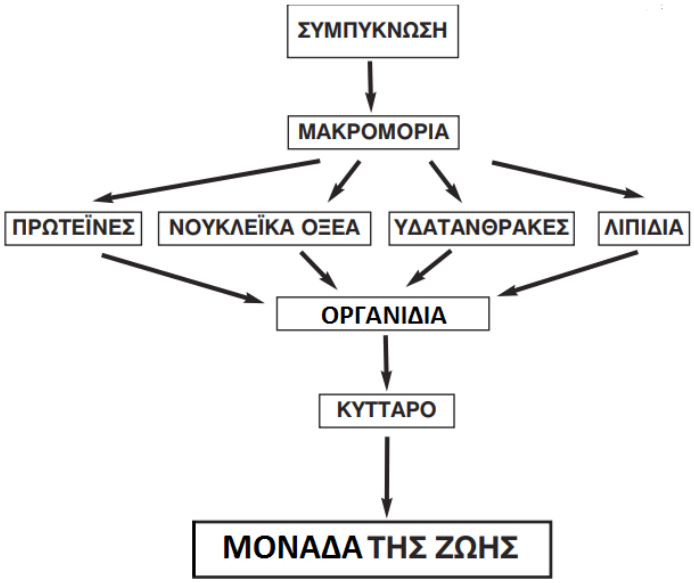


Σχήμα 2.3: Διαδικασία συμπύκνωσης

Πάρθηκε από:
http://ebooks.edu.gr/modules/ebook/show.php/DSGL-B106/85/678,2573/

Αναμφισβήτητα, από τα πιο πάνω μπορούμε να κατανοήσουμε ότι τα αμινοξέα αποτελούν τους δομικούς λίθους των πρωτεϊνών και καθορίζουν τις χαρακτηριστικές τους ιδιότητες. Ανάλογα με τη σειρά που βρίσκονται σε μια πολυπεπτιδική αλυσίδα, αλληλεπιδρούν μεταξύ τους για τον καθορισμό της τελικής δομής της πρωτεΐνης.

Οι πρωτεΐνες, τα νουκλεϊκά οξέα, οι πολυσακχαρίτες (υδατάνθρακες) και τα λιπίδια αποτελούν τις τέσσερις κατηγορίες των μακρομορίων και συνεργάζονται, για να δημιουργήσουν συμπλέγματα μακρομορίων. Αυτά, με τη σειρά τους, απαρτίζουν τα οργανίδια, τα οποία χαρακτηρίζονται ως τα μέρη του κυττάρου που παρουσιάζουν συγκεκριμένο δομικό ή λειτουργικό ρόλο. Το κύτταρο αποτελεί μονάδα ζωής, αφού οι οργανισμοί -τόσο μονοκύτταροι όσο και πολυκύτταροι- συγκροτούνται από αυτά (Σχήμα 2.4). Οι πρωτεΐνες είναι τα πιο διαδεδομένα και ποικιλόμορφα μόρια που υπάρχουν σε αυτό και αποτελούν το μεγαλύτερο μέρος της στερεής μάζας του.



Σχήμα 2.4: Από την συμπύκνωση δημιουργούνται τα μακρομόρια τα οποία χωρίζονται σε 4 κατηγορίες. Τα μακρομόρια δημιουργούν συμπλέγματα τα οποία απαρτίζουν οργανίδια τα οποία είναι μέρη του κυττάρου.

Πάρθηκε από: http://archeia.moec.gov.cy/sm/46/epopt_yliko_b_lykeiou_diafanies.pdf

2.2 Ρόλος και λειτουργίες των πρωτεϊνών

Είναι κοινώς αποδεκτό ότι οι πρωτεΐνες συνιστούν απαράμιλλης σημασίας κομμάτι, τόσο του ανθρώπινου όσο και οποιουδήποτε άλλου βιολογικού οργανισμού, αφού επιτελούν σημαντικές βιολογικές του λειτουργίες.

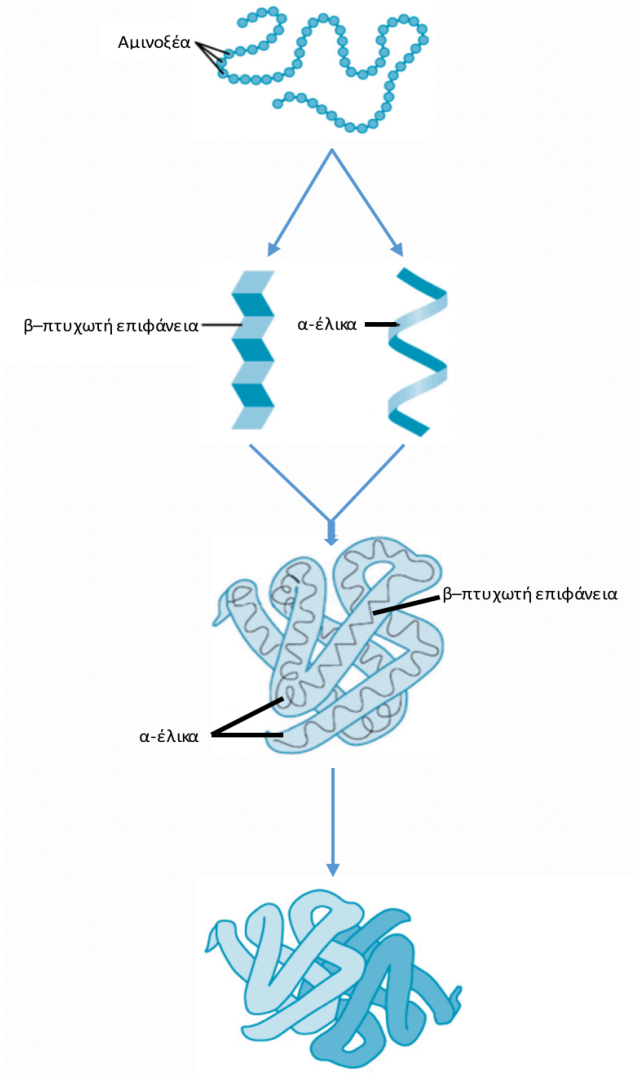
Διαδραματίζουν πολυδιάστατο ρόλο στους οργανισμούς, με τον πιο σημαντικό να διαφαίνεται ο δομικός. Συγκεκριμένα, αποτελούν τα δομικά συστατικά των μεμβρανών του κυττάρου, βοηθώντας στη στήριξή του, και σε μεγαλύτερη κλίμακα, διαδραματίζουν καίριο ρόλο στην κίνηση του σώματος (συσταλτικές πρωτεΐνες: ακτίνη, μυοσίνη). Παράλληλα, μπορούν να έχουν καταλυτικό ρόλο, αφού πολλές από αυτές δρουν ως ένζυμα, για να διεκπεραιωθούν οι χημικές αντιδράσεις. Επιπρόσθετα, δύνανται να επιτελούν ρόλο ορμόνης (π.χ., ινσουλίνη) για να διατηρείται η ομοιόσταση του οργανισμού, καθώς και προστατευτικό, αφού ως αντισώματα (ανοσοσφαιρίνες) αναγνωρίζουν και επιτίθενται σε εισβολείς του οργανισμού. Ο ρόλος τους, όμως, δεν περιορίζεται μόνο σε αυτούς. Οι πρωτεΐνες αποτελούν σημαντική πηγή ενέργειας όταν τα υπόλοιπα αποθέματα του οργανισμού (λίπη, υδατάνθρακες) δεν επαρκούν, ενώ ως

νευροδιαβιβαστές βοηθούν στο συντονισμό των βιολογικών διεργασιών. Τέλος, αξίζει να σημειωθεί και η μεταφορική τους λειτουργία όσον αφορά στο ζωτικής σημασίας αέριο του οργανισμού, το οξυγόνο (αιμοσφαιρίνη, μυοσφαιρίνη).

Η λειτουργία που εκτελεί η κάθε πρωτεΐνη βρίσκεται σε άμεση συνάρτηση με την αναδίπλωση που θα επισυμβεί, ώστε να σχηματιστεί η τριτοταγής της δομή.

2.3 Επίπεδα οργάνωσης πρωτεϊνών

Η επινόηση μιας ιεραρχικής δομής, προκειμένου να κατανοηθούν βαθύτερα τα επίπεδα οργάνωσης των πρωτεϊνών, υπήρξε καθοριστικής σημασίας. Τα πιο πάνω επίπεδα αντιστοιχούν στην πρωτοταγή, δευτεροταγή, τριτοταγή και τεταρτοταγή δομή (Σχήμα 2.5).



Πρωτοταγής δομή
Η ακολουθία των αμινοξέων σε μια πολυπεπτιδική αλυσίδα

καθορίζει

Δευτεροταγής δομή
Η αναδίπλωση της πολυπεπτιδικής αλυσίδας και η μετατροπή της σε ελικοειδή ή πτυχωτή μορφή

καθορίζει

Τριτοταγής δομή
Αποτελεί αποτέλεσμα της τελικής αναδίπλωσης της πολυπεπτιδικής αλυσίδας στο χώρο

καθορίζει

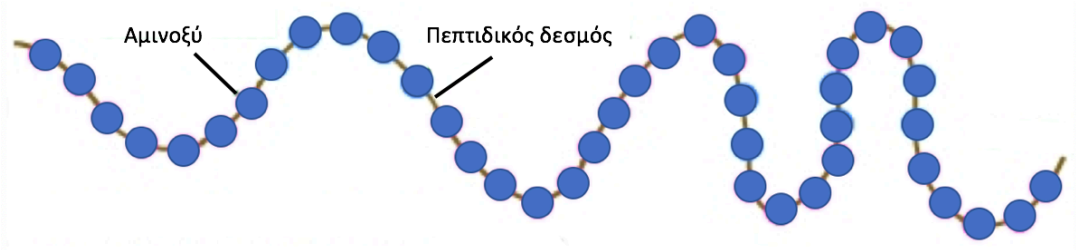
Τεταρτοταγής δομή
Όταν η πρωτεΐνη αποτελείται από περισσότερες από μια πολυπεπτιδικές αλυσίδες

Σχήμα 2.5: Επίπεδα οργάνωσης πρωτεϊνών

Πάρθηκε από: https://www.pathwayz.org/Tree/Plain/PROTEIN+STRUCTURE

Πρωτοταγής δομή

Η πρωτοταγής δομή των πρωτεϊνών, η οποία προκύπτει από την ένωση των αμινοξέων με πεπτιδικούς ομοιοπολικούς δεσμούς, εξαρτάται από την αλληλουχία των νουκλεοτιδίων του DNA (Deoxyribonucleic acid). Η μετάφραση του DNA έγινε πλέον ανθρώπινη κατάκτηση, με αποτέλεσμα να είμαστε σε θέση να γνωρίζουμε την πρωτοταγή δομή ενός τεράστιου αριθμού πρωτεϊνών. Γνωρίζοντας, όμως, την πρωτοταγή δομή μιας πρωτεΐνης, δε μπορούμε να ξέρουμε σημαντικά στοιχεία για αυτήν, όπως τη λειτουργία ή το σχήμα της σε τρισδιάστατο χώρο. Όπως παρατηρήσαμε και στο Σχήμα 2.2, υπάρχουν 20 αμινοξέα, τα οποία μπορούν να ενωθούν μεταξύ τους με ποικίλους συνδυασμούς, με αποτέλεσμα την ύπαρξη ενός τεράστιου αριθμού διαφορετικών αλυσιδών.



Σχήμα 2.6: Πρωτοταγής δομή της πρωτεΐνης

Πάρθηκε από: https://biology.tutorvista.com/biomolecules/proteins.html

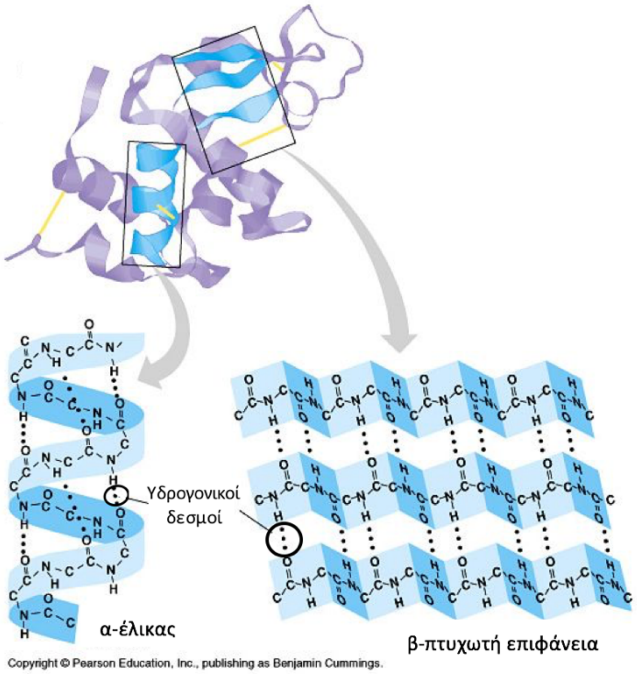
Δευτεροταγής δομή

Οι πολυπεπτιδικές αλυσίδες συσπειρώνονται με βάση τον τρόπο που προκαθορίζεται στην πρωτοταγή δομή και, έτσι, δημιουργείται η τρισδιάστατη μορφή τους στο χώρο, δηλαδή η δευτεροταγής τους δομή. Τα αμινοξέα που απαρτίζουν την αρχική αμινοξική ακολουθία ενώνονται με υδρογονικούς δεσμούς και δημιουργούν α-έλικες, β-πτυχωτές επιφάνειες ή άλλες κατηγορίες δευτεροταγούς δομής. Οι υδρογονικοί αυτοί δεσμοί είναι υπεύθυνοι για τη σταθερότητα της συγκεκριμένης δομής (Αγαθοκλέους, 2009). Το πρόγραμμα DSSP (Define Secondary Structure of Protein), το οποίο αναφέρθηκε και από τον Αγαθοκλέους (2009), χρησιμοποιείται για να καθορίσουμε την δευτεροταγή δομή των πρωτεϊνών, κατηγοριοποιώντας, παράλληλα, τις πρωτεΐνες στις κατηγορίες που παρουσιάζονται στον Πίνακα 1.

Πίνακας 1: Οι τρεις ευρύτερες κατηγορίες H, E, C, στις οποίες συμπτύσσονται οι οκτώ κατηγορίες, όπως ορίζονται από το πρόγραμμα DSSP

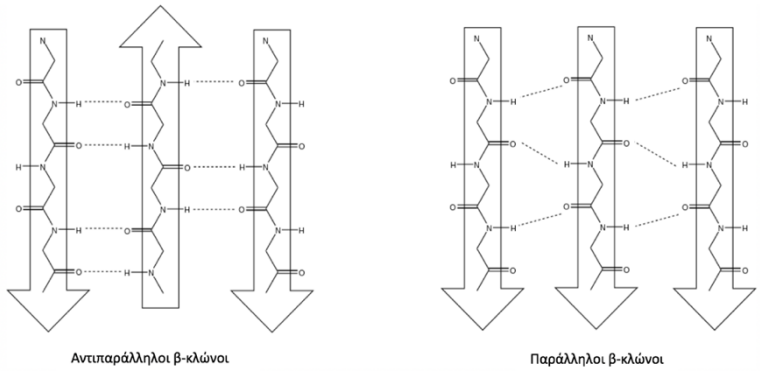
H (έλικες)	**E (κλώνοι)**	**C (coil)**
α-helix (H)	β-strand (E)	β-turn (T)
3-helix (G)	β-bridge (B)	bend (S)
π-helix (I)		(C): Οι υπόλοιπες πρωτεΐνες ανήκουν σε αυτή την κατηγορία

Οι α-έλικες ανήκουν στις ελικοειδείς δευτεροταγείς δομές και οι β-πτυχωτές επιφάνειες ανήκουν στις εκτεταμένες δομές. Συγκεκριμένα, όπως παρατηρούμε στο Σχήμα 2.7, οι α-έλικες έχουν μια ελικοειδή μορφή και οι ασθενείς δεσμοί υδρογόνου που συνδέουν τις στροφές τους, αποτελούν την αιτία για την πιο σταθερή δομή συγκριτικά με άλλες μη ελικοειδείς αλυσίδες πολυπεπτιδίων. Αντίστοιχα, στο ίδιο σχήμα, παρουσιάζεται η αναδίπλωση της β-πτυχωτής επιφάνειας, η οποία είναι αποτέλεσμα της ένωσης των β-κλώνων, που διατάσσονται είτε παράλληλα είτε αντιπαράλληλα με επίσης υδρογονικούς δεσμούς.



Σχήμα 2.7: Δευτεροταγής δομή της πρωτεΐνης αποτελείται από α-έλικες και β-πτυχωτές επιφάνειες

Πάρθηκε από: http://slideplayer.com/slide/273474/

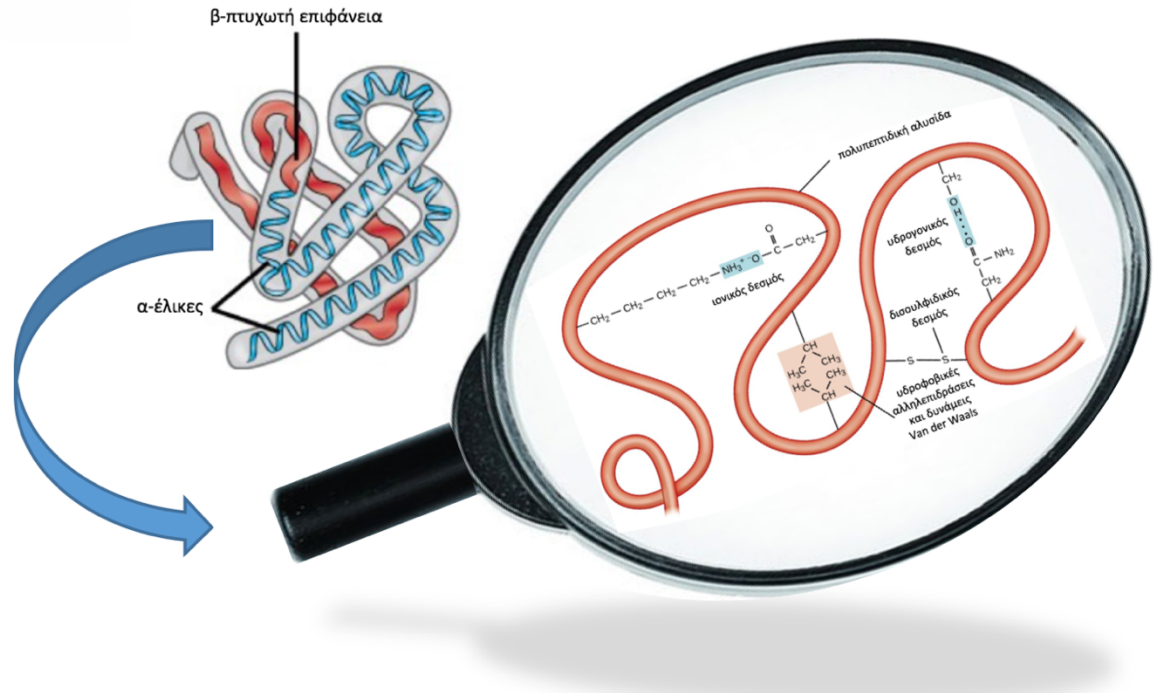


Σχήμα 2.8: Οι β-κλώνοι παρουσιάζονται συχνά στη δευτεροταγή δομή των πρωτεϊνών και μπορεί να είναι είτε παράλληλοι, είτε αντιπαράλληλοι

Πάρθηκε από: https://commons.wikimedia.org/wiki/File:Beta_sheets.svg

Τριτοταγής δομή

Η τρισδιάστατη μορφή της πρωτεΐνης στο χώρο συνδέεται άμεσα με την τριτοταγή της δομή, αφού περιγράφει το τελικό σχήμα της πολυπεπτιδικής αλυσίδας στο χώρο, όπως αυτό προκύπτει από την περαιτέρω αναδίπλωσή της. Για την επίτευξη της σταθερότητας της τριτοταγούς δομής αλληλεπιδρούν μεταξύ τους οι πλευρικές ομάδες R των αμινοξέων, δημιουργώντας δεσμούς υδρογόνου, ιοντικούς δεσμούς, υδρόφοβους δεσμούς, δυνάμεις Van der Waals και ομοιοπολικούς δισουλφιδικούς δεσμούς.

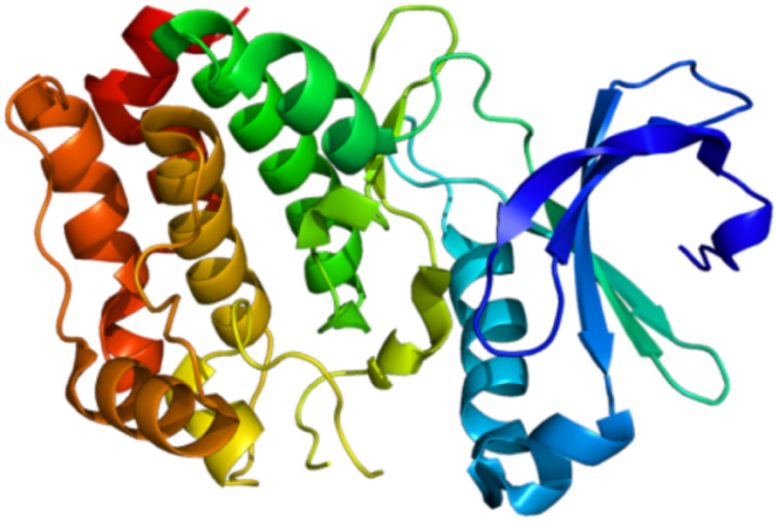


Σχήμα 2.9: Η τριτοταγής δομή μιας πρωτεΐνης και οι δεσμοί που δημιουργούνται από τις αλληλεπιδράσεις των πλευρικών ομάδων R για την σταθερότητα της

Δημιουργήθηκε με χρήση εικόνων από: https://courses.lumenlearning.com/microbiology/chapter/proteins/

Τεταρτοταγής δομή

Η τεταρτοταγής δομή δημιουργείται όταν συνδυαστούν περισσότερες από μία όμοιες ή διαφορετικές πολυπεπτιδικές αλυσίδες μετά την αναδίπλωσή τους. Σε αυτή τη δομή οι πρωτεΐνες ονομάζονται πρωτεϊνικά σύμπλοκα και οι πολυπεπτιδικές αλυσίδες που είχαν λάβει μέρος σε αυτή τη διαδικασία ονομάζονται υπομονάδες.



Σχήμα 2.10: Τεταρτοταγής δομή μιας πρωτεΐνης

Πάρθηκε από: https://medicalxpress.com/news/2014-07-proteins-scientists-drug-discovery-tool.html

Γενικά για τις πρωτεΐνες

Μια πρωτεΐνη, για να είναι λειτουργική – να μπορεί να χρησιμοποιηθεί στο κύτταρο – υποβάλλεται στα πιο πάνω στάδια, για να αποκτήσει την τελική της δομή και λειτουργία. Η δομή και η λειτουργία μιας πρωτεΐνης, όπως έχει ήδη αναφερθεί, είναι αλληλένδετες έννοιες. Δηλαδή, αν αλλάξει η δομή της (επίπεδα οργάνωσης), τότε αλλάζει και η λειτουργία της.

Κεφάλαιο 3

Νευρωνικά Δίκτυα

3.1 Η σημασία των τεχνητών νευρωνικών δικτύων

3.2 Βιολογικός νευρώνας

3.3 Τεχνητός νευρώνας και τεχνητά νευρωνικά δίκτυα

- 3.3.1 Μοντέλο McCulloh και Pitts
- 3.3.2 Συναρτήσεις ενεργοποίησης
- 3.3.3 Πολυστρωματικά νευρωνικά δίκτυα perceptron (Multilayer perceptron (MLP)) εμπρόσθιου περάσματος (Feedforward)
- 3.3.4 Νευρωνικό δίκτυο με ανάδραση (Recurrent Neural Network (RNN))
- 3.3.5 Clockwork νευρωνικό δίκτυο με ανάδραση (Clockwork RNN (CW-RNN)

3.4 Αλγόριθμοι μάθησης τεχνητών νευρωνικών δικτύων

- 3.4.1 Αλγόριθμος μάθησης Perceptron (Perceptron learning algorithm)
- 3.4.2 Μέθοδος κατάβασης κλίσης (Gradient Descent (GD))
- 3.4.3 Μέθοδος στοχαστικής κατάβασης κλίσης (Stochastic Gradient Descent (SGD))
- 3.4.4 Μέθοδος mini-batch κατάβασης κλίσης (Mini-Batch Gradient Descent (MB-GD))
- 3.4.5 Adam (Adaptive Moment Estimation) optimizer
- 3.4.6 Αλγόριθμος μάθησης ανάστροφης μετάδοσης σφάλματος (Backpropagation algorithm)
- 3.4.7 Αλγόριθμος μάθησης ανάστροφης μετάδοσης σφάλματος στο χρόνο (Backpropagation Through Time (BPTT))

3.1 Η σημασία των τεχνητών νευρωνικών δικτύων

Τα τεχνητά νευρωνικά δίκτυα αποτελούν τομέα έρευνας της Τεχνητής Νοημοσύνης. Συγκεκριμένα, διεξάγονταν και διεξάγονται έρευνες και μελέτες για την μίμηση της λειτουργίας του εγκεφάλου από υπολογιστικά μοντέλα. Ο ανθρώπινος εγκέφαλος είναι ένα πανίσχυρο εργαλείο με ένα πολύπλοκο τρόπο λειτουργίας, που ακόμη και σήμερα δεν ανακαλύφθηκε πλήρως το πως λειτουργεί. Η εμφάνιση των νευρωνικών δικτύων οφείλεται τόσο στον περιορισμό των κλασικών υπολογιστικών συστημάτων να επιλύσουν πολύπλοκα, ασαφή και μεγάλα σε όγκο προβλήματα, αλλά και για την καλύτερη κατανόηση της λειτουργίας του ανθρώπινου εγκεφάλου. Με τη χρήση των νευρωνικών δικτύων μας δίνεται η δυνατότητα να προσομοιώσουμε κάποια χαρακτηριστικά του εγκεφάλου, όπως την ταχύτητα που επεξεργάζεται πολύπλοκα δεδομένα μέσω του παραλληλισμού, την ανεκτικότητα σε λάθη αλλά και την ευρωστία που διαθέτει, αφού αν τύχει κάποια βλάβη σε κάποιο νευρώνα, το νευρωνικό δίκτυο θα συνεχίσει να λειτουργεί. Επιπρόσθετα, υιοθετώντας τα χαρακτηριστικά του ανθρώπινου εγκεφάλου, μπορούν να προσαρμόζονται και να μαθαίνουν από παραδείγματα αλλά και να γενικεύουν από αυτά, με αποτέλεσμα να μπορούν να απαντήσουν και να δώσουν λύσεις σε καινούργιες άγνωστες καταστάσεις, που σχετίζονται όμως με τα παραδείγματα που προηγήθηκαν. Επίσης, μπορούν να συλλέξουν και να αναγνωρίσουν σημαντικές πληροφορίες από ασυνεπή ή "θορυβώδη" δεδομένα. Εκτός από τα πιο πάνω, μπορούν να λύσουν μη γραμμικά διαχωρίσιμα προβλήματα, με τη χρήση μη γραμμικών συναρτήσεων ενεργοποίησης (υποκεφάλαιο 3.3.2) και δεν έχουν μεγάλες υπολογιστικές απαιτήσεις μετά την εκπαίδευση του δικτύου. Με βάση τα παραπάνω, τα νευρωνικά δίκτυα θεωρούνται ιδανικά για την επίλυση προβλημάτων τα οποία δεν μπορούν να λυθούν με κάποιους συγκεκριμένους κανόνες, έχουν θορυβώδη δεδομένα ή μεγάλο όγκο από δεδομένα, χρειάζονται μεγάλη ταχύτητα και ισχύ ή γνωρίζουμε τα σύνολα δεδομένων εισόδου και τις επιθυμητές εξόδους τους.

Για να γίνει, όμως, πλήρης αξιοποίηση των πιο πάνω χαρακτηριστικών από το τεχνητό νευρωνικό δίκτυο, είναι αναγκαία η σωστή του εκπαίδευση, όπως γίνεται και στον άνθρωπο. Καθώς εκπαιδεύεται ο ανθρώπινος εγκέφαλος, μπορεί να χρειαστεί αρκετές επαναλήψεις, προκειμένου να αφομοιώσει και να κατανοήσει πλήρως μια νεοεισερχόμενη πληροφορία. Είναι σημαντικό σε αυτό το σημείο να τονιστεί ότι αυτή η

διαδικασία ολοκληρώνεται πιο εύκολα εάν η πληροφορία αυτή αναγνωριστεί από τον ίδιο ως σημαντική για επερχόμενη ανάκληση. Δηλαδή, μετά από έναν αριθμό επαναλήψεων των ίδιων δεδομένων ή πληροφοριών, δημιουργούνται καινούριες συνάψεις μεταξύ των νευρώνων του εγκεφάλου και σχηματίζεται, έτσι, ένα κομμάτι νευρωνικού δικτύου, που είναι υπεύθυνο για τη διατήρηση της καινούριας γνώσης. Υπάρχουν τρία είδη μηχανικής μάθησης, τα οποία προσπαθούν να μιμηθούν τους τρόπους με τους οποίους εκπαιδεύεται ο εγκέφαλος. Τα είδη αυτά περιγράφονται πιο κάτω.

Επιβλεπόμενη μάθηση

Αυτό το είδος μάθησης μπορούμε να το αντιστοιχίσουμε με τον τρόπο που μαθαίνει ένας μαθητής από τον δάσκαλο του, ο οποίος μετά από κάθε απάντηση του μαθητή, τον ενημερώνει αν ήταν ορθή ή λανθασμένη. Παρομοιάζοντας, λοιπόν, το νευρωνικό δίκτυο ως τον μαθητή και τα ζεύγη δεδομένων (δεδομένα εισόδου και επιθυμητής εξόδου) ως τον δάσκαλο, προσπαθεί να μειώσει τη διαφορά (σφάλμα) μεταξύ της επιθυμητής και της πραγματικής εξόδου του για κάθε δεδομένο εισόδου. Αυτό επιτυγχάνεται με την προσαρμογή των βαρών και το δίκτυο σταματάει την εκπαίδευση όταν η πιο πάνω διαφορά μειωθεί μέχρι κάποιο αισθητό βαθμό ή όταν περάσει κάποιος αριθμός εποχών εκπαίδευσης. Μια εποχή ολοκληρώνεται όταν εισέρθουν όλα τα δεδομένα στο δίκτυο για εκπαίδευση.

Ενισχυτική μάθηση

Η ενισχυτική μάθηση είναι πιο βιολογικά ρεαλιστική μάθηση. Υπάρχει κάποιος “κριτής” ο οποίος κρίνει τις πράξεις του μαθητή-νευρωνικού δικτύου με κάποια επιβράβευση ή τιμωρία, χωρίς όμως να τον ανατροφοδοτεί ακριβώς με την ορθή απάντηση. Αυτό γίνεται για να κατευθύνεται ο μαθητής ή το νευρωνικό δίκτυο προς την ελαχιστοποίηση των τιμωριών και για την μεγιστοποίηση των επιβραβεύσεων. Τα βάρη του δικτύου αλλάζουν έχοντας σαν στόχο αυτή την κατάσταση. Ως αποτέλεσμα, ο ανθρώπινος εγκέφαλος προσπαθεί να επαναλάβει κάποια πράξη η οποία έχει τύχει επιβράβευσης, ενώ παράλληλα να μειώσει τη συχνότητα των πράξεων οι οποίες επιφέρουν την τιμωρία. Το νευρωνικό δίκτυο προσπαθεί να μιμηθεί αυτή τη λογική.

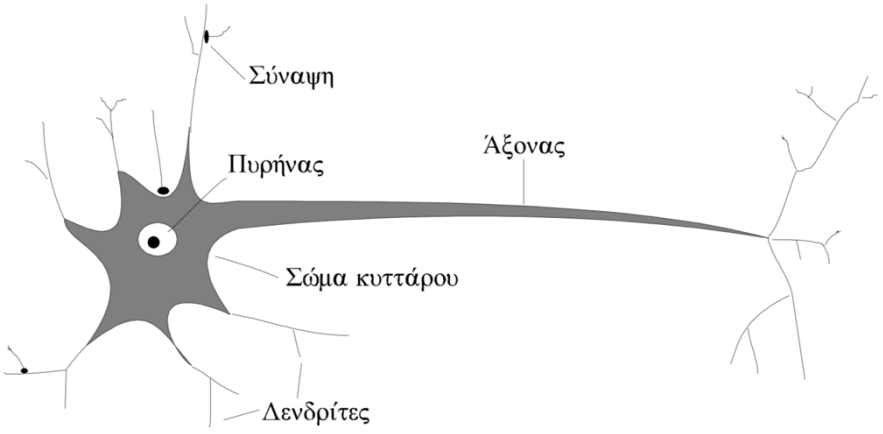
Μη επιβλεπόμενη μάθηση

Σε αυτό το είδος μάθησης δεν υπάρχει δάσκαλος (επιθυμητές έξοδοι), ούτε κριτής, αλλά μόνο τα δεδομένα εισόδου. Ο ανθρώπινος εγκέφαλος προσπαθεί να αναγνωρίσει κοινά χαρακτηριστικά μεταξύ αυτών των δεδομένων για να τα κατηγοριοποιήσει σε διαφορετικές ομάδες. Παράλληλα, το νευρωνικό δίκτυο όταν δέχεται κάποιο δεδομένο εισόδου προσπαθεί να το αντιστοιχήσει με την ομάδα που ταυτίζεται καλύτερα.

3.2 Βιολογικός νευρώνας

Ο νευρώνας είναι το κύτταρο που θεωρείται η βασική μονάδα δόμησης του εγκεφάλου. Το ανθρώπινο σώμα -και συγκεκριμένα ο ανθρώπινος εγκέφαλος- περιέχει περίπου 10 x 10^9 διαφόρων τύπων νευρώνες. Οι νευρώνες συνδέονται μεταξύ τους με "συνάψεις". Ένα σύνολο από διασυνδεδεμένους νευρώνες αποτελούν ένα νευρωνικό δίκτυο.

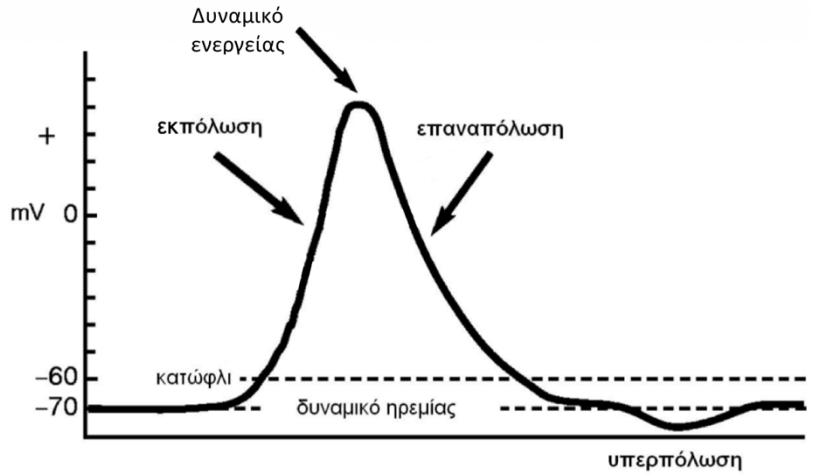
Ένας νευρώνας, ως τμήμα ενός συγκεκριμένου νευρωνικού δικτύου, είναι υπεύθυνος να λαμβάνει τα εισερχόμενα σήματα από άλλους νευρώνες, τα επεξεργάζεται ποικιλομόρφως και να αποστέλλει το αποτέλεσμα της προηγηθείσας εξεργασίας σε άλλους νευρώνες. Η δομή του, όπως παρατηρείται στο Σχήμα 3.1, απαρτίζεται το σώμα με τους δενδρίτες, που είναι προσαρτημένοι σε αυτό, και τον άξονα με τις τελικές απολήξεις. Οι δενδρίτες είναι υπεύθυνοι για την υποδοχή των εισερχόμενων σημάτων, το σώμα με τον πυρήνα του για την επεξεργασία και ο άξονας για την αποστολή των εξερχόμενων σημάτων σε άλλους νευρώνες συνδεδεμένους με αυτόν.



Σχήμα 3.1: Η δομή ενός βιολογικού νευρώνα

Πάρθηκε από: http://kelifos.physics.auth.gr/COURSES/neural/K3.pdf

Ένας νευρώνας ονομάζεται ενεργός όταν πυροδοτεί σήματα και μη-ενεργός όταν βρίσκεται σε μια αδρανή κατάσταση, στην οποία αδυνατεί τόσο να δεχτεί όσο και να στείλει σήματα. Τα σήματα χωρίζονται σε διεγερτικά και ανασταλτικά. Αυτά τα σήματα συναθροίζονται και, αν το αποτέλεσμα φτάσει ή ξεπεράσει κάποια συγκεκριμένη τιμή -δηλαδή κάποιο κατώφλι- τότε λέγεται ότι ο νευρώνας είναι ενεργός και πυροδοτεί. Τα διεγερτικά σήματα προσθέτουν κάποια θετική τιμή στο δυναμικό του νευρώνα, με αποτέλεσμα να πλησιάζει το κατώφλι, ενώ τα ανασταλτικά μειώνουν το δυναμικό του νευρώνα. Την ενδεχόμενη διέγερση του νευρώνα και τη δημιουργία ηλεκτρικού σήματος (παλμό), μετά από το σύνολο των σημάτων που θα έχει δεχτεί, τόσο διεγερτικών όσο και ανασταλτικών, μπορούμε να την παρατηρήσουμε στο Σχήμα 3.2.



Σχήμα 3.2: Το δυναμικό δράσης ενός βιολογικού νευρώνα όταν ενεργοποιείται

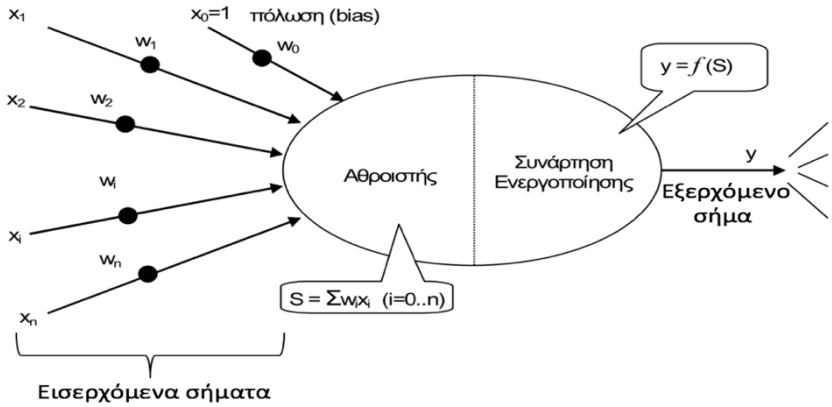
Πάρθηκε από: http://docplayer.gr/221387-Prosomoiosi-diktyoy-viologikon-neyronon-me-hrisi-montelon-izhikevich-odigoymenon-apo-dynamika-topikoy-pedioy-diplomatiki-ergasia.html

3.3 Τεχνητός νευρώνας και τεχνητά νευρωνικά δίκτυα

3.3.1 Μοντέλο McCulloh και Pitts

Οι τεχνητοί νευρώνες προσπαθούν να προσομοιώσουν τη λειτουργία του βιολογικού νευρώνα, όπως αναφέρθηκε στο προηγούμενο υποκεφάλαιο. Δηλαδή, δέχεται κάποια εισερχόμενα σήματα, τα συναθροίζει στον "Αθροιστή", όπως παρατηρείται στο Σχήμα 3.4, και στη συνέχεια στην "Συνάρτηση Ενεργοποίησης" ελέγχει αν το αποτέλεσμα αυτό έφτασε ή ξεπέρασε κάποιο κατώφλι. Αν ικανοποιείται μια από αυτές τις δύο συνθήκες,

τότε, όπως και στον βιολογικό νευρώνα, ο τεχνητός νευρώνας πυροδοτεί, προκαλώντας τη μετάδοση του εξερχόμενου του σήματος.



Σχήμα 3.3: Τεχνητός νευρώνας McCulloh και Pitts

Πάρθηκε από: http://aibook.csd.auth.gr/include/slides/Chap19.pdf

Οι McCulloh και Pitts (1943) είναι οι δημιουργοί του απλού μοντέλου νευρώνα που παρουσιάζεται στο Σχήμα 3.3. Όπως παρατηρούμε οι είσοδοι του τεχνητού νευρώνα, που μπορεί να επέρχονται από άλλους νευρώνες, πολλαπλασιάζονται με τα αντίστοιχα τους βάρη που βρίσκονται στις ακμές και συναθροίζονται. Με τον πολλαπλασιασμό τους, και μετέπειτα με την πρόσθεση αυτών των τιμών μπορεί το σήμα που παράγεται να ενισχύεται ή να αποδυναμώνεται. Η διαδικασία του πολλαπλασιασμού και της συνάθροισης των τιμών για τον υπολογισμό της συνολικής τιμής εισόδου του τεχνητού νευρώνα πριν την είσοδο της στην συνάρτηση ενεργοποίησης γίνεται με την Εξίσωση 3.1. Η είσοδος x_0 έχει πάντα τη θετική τιμή 1 και ονομάζεται πόλωση και ο σκοπός της είναι να εξαφανίζει το κατώφλι, αφού αν το αποτέλεσμα από την Εξίσωση 3.1 είναι μεγαλύτερο αυτού, τότε ο νευρώνας ενεργοποιείται, αλλιώς παραμένει ανενεργός.

$$S = \sum_i w_i x_i + bias$$

Εξίσωση 3.1: Συνολική τιμή εισόδου του τεχνητού νευρώνα

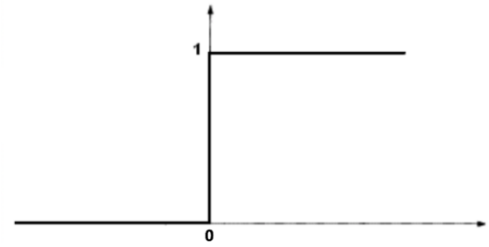
x_i: δεδομένο εισόδου; **w_i:** βάρος ακμής i

Ως αρχική συνάρτηση κατωφλίου στα τεχνητά νευρωνικά δίκτυα, χρησιμοποιήθηκε η βηματική συνάρτηση (Heaviside). Εισάγοντας την τιμή S, από την Εξίσωση 3.1, στην συνάρτηση κατωφλίου, Εξίσωση 3.2, τότε μας δίνει το αποτέλεσμα αν ο νευρώνας θα

πυροδοτήσει ή όχι. Με την πρόσθεση του bias, όπως αναφέραμε πιο πάνω, η συνάρτηση κατωφλίου διαμορφώνεται ανάλογα όπως παρουσιάζεται στο Σχήμα 3.4.

$$f = \begin{cases} 1, & \text{εάν } S \geq 0 \\ 0, & \text{αλλιώς } (S < 0) \end{cases}$$

Εξίσωση 3.2: Εξίσωση βηματικής συνάρτησης (Heaviside) κατωφλίου



Σχήμα 3.4: Βηματική συνάρτηση (Heaviside) κατωφλίου

Πάρθηκε από: https://www.quora.com/What-is-the-unit-step-Function-in-Artificial-Neural-Network

Δυστυχώς, η βηματική συνάρτηση, δε βοηθάει ένα νευρωνικό δίκτυο να μάθει ορθά, λόγω κάποιων μειονεκτημάτων που παρουσιάζει, όπως το ότι δεν υποδεικνύει πόσο κοντά ή μακριά βρίσκεται το τελικό αποτέλεσμα του δικτύου από το κατώφλι για να υπολογιστεί η τιμή απόκλισης λάθους και το ότι δεν είναι παραγωγίσιμη σε όλο το πεδίο ορισμού συντελεί στη δυσλειτουργία της μεθόδου κατάβασης κλήσης (Υποκεφάλαιο 3.4.2). Ως αποτέλεσμα, επιλέγουμε να χρησιμοποιήσουμε άλλες συναρτήσεις ενεργοποίησης, οι οποίες είναι παραγωγίσιμες. Το αποτέλεσμα της Εξίσωσης 3.1 μπορεί να αποτελέσει είσοδο για οποιαδήποτε συνάρτηση ενεργοποίησης και έτσι δημιουργείται η πιο γενικευμένη Εξίσωση 3.3.

$$y = f(\sum_{i} w_i x_i)$$

Εξίσωση 3.3: Η τελική τιμή εξόδου ενός νευρώνα.

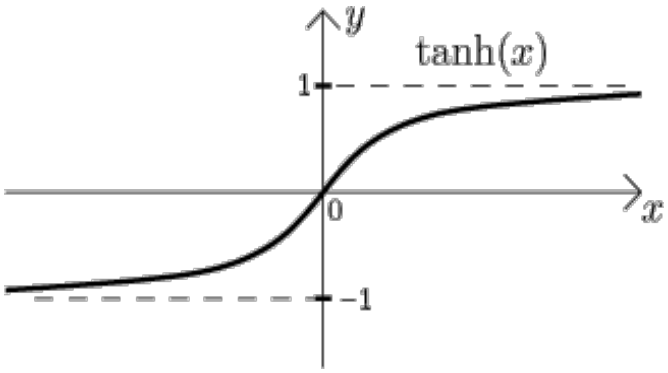
f: κάποια παραγωγίσιμη συνάρτηση ενεργοποίησης

3.3.2 Συναρτήσεις ενεργοποίησης

Όπως έχει ειπωθεί και πιο πάνω, η βηματική συνάρτηση ανήκει στις συναρτήσεις ενεργοποίησης, αλλά τα μειονεκτήματα της την καθιστούν μη αποτελεσματική στις

πολύπλοκες εφαρμογές που προσπαθούμε με κάποιες τεχνικές να προσεγγίσουμε. Άλλες συναρτήσεις ενεργοποίησης, οι οποίες αντιμετωπίζουν τα προβλήματα της συνάρτησης αυτής παρουσιάζονται στη συνέχεια.

Υπερβολική εφαπτομένη – Hyperbolic tangent (tanh)



Σχήμα 3.5: Υπερβολική εφαπτομένη

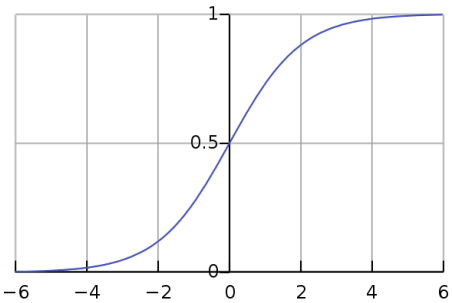
Πάρθηκε από: *http://math.feld.cvut.cz/mt/txtb/4/txe3ba4f.htm*

Η υπερβολική εφαπτομένη είναι μια μη γραμμική συνάρτηση και έτσι μπορεί να μοντελοποιεί μη γραμμικά διαχωρίσιμα προβλήματα. Σε όλο το πεδίο τιμών της είναι παραγωγίσιμη και οι τιμές εξόδου του νευρωνικού δικτύου περιορίζονται στις τιμές -1 και 1. Η συγκεκριμένη συνάρτηση έχει την εξίσωση που παρουσιάζεται στην Εξίσωση 3.4.

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Εξίσωση 3.4: Εξίσωση της υπερβολικής εφαπτομένης

Σιγμοειδής



Σχήμα 3.6: Σιγμοειδής συνάρτηση

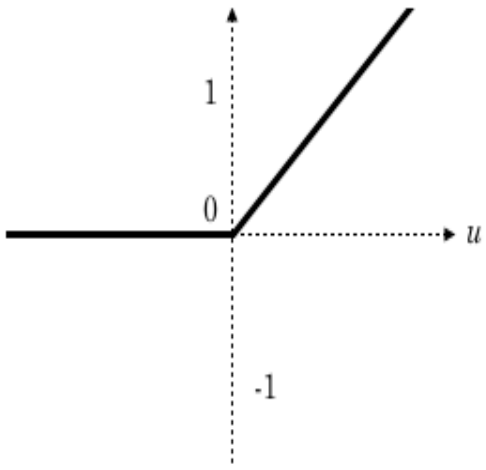
Πάρθηκε από:
https://commons.wikimedia.org/wiki/File:Logistic-curve.svg

Η σιγμοειδής είναι μια μη γραμμική συνάρτηση και έτσι, όπως και η υπερβολική εφαπτομένη που αναφέρθηκε πιο πάνω, μπορεί να μοντελοποιεί μη γραμμικά διαχωρίσιμα προβλήματα. Το πεδίο τιμών της κυμαίνεται μεταξύ 0 και 1. Οι τιμές εξόδου (= πεδίο τιμών) της είναι πολύ απότομες (από X = -2 μέχρι 2), αφού και η παραμικρή αλλαγή στις αντίστοιχες τιμές εισόδου μπορούν να τις επηρεάσουν σε μεγάλο βαθμό. Με αυτό κατανοούμε ότι μπορεί να καταλήξει σε πιο σαφείς διακρίσεις στην πρόβλεψη. Δυστυχώς όμως, το αρνητικό αυτής της συνάρτησης είναι ότι όσο η γραμμή στον άξονα των X γίνεται πιο επίπεδη, η κλίση της συνάρτησης γίνεται ολοένα και πιο μικρή όπως και οι αλλαγές στις τιμές εξόδου στις αντίστοιχες αλλαγές των τιμών εισόδου. Δηλαδή, αφού θα γίνονται μόνο μικρές αλλαγές, το δίκτυο είτε δε θα μαθαίνει, είτε θα μαθαίνει πολύ αργά.

$$S(t) = \frac{1}{1 + e^{-t}}$$

Εξίσωση 3.5: Εξίσωση της σιγμοειδούς συνάρτησης

Ακόμη δύο συναρτήσεις ενεργοποίησης είναι η Rectified Linear Unit (ReLU) (Σχήμα 3.7 και Εξίσωση 3.6) και η Softmax (Εξίσωση 3.7). Αυτές οι συναρτήσεις, όπως θα δούμε και στη συνέχεια (Κεφάλαιο 5 – Αποτελέσματα), δοκιμάστηκαν χωρίς όμως να επιφέρουν καλύτερα αποτελέσματα από την υπερβολική εφαπτομένη και την σιγμοειδή συνάρτηση.



Σχήμα 3.7: ReLU

Πάρθηκε από: https://www.researchgate.net/figure/ReLU-activation-function_fig3_319235847

$$f(x) = \max(0, x)$$

Εξίσωση 3.6: Εξίσωση της συνάρτησης ReLU

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

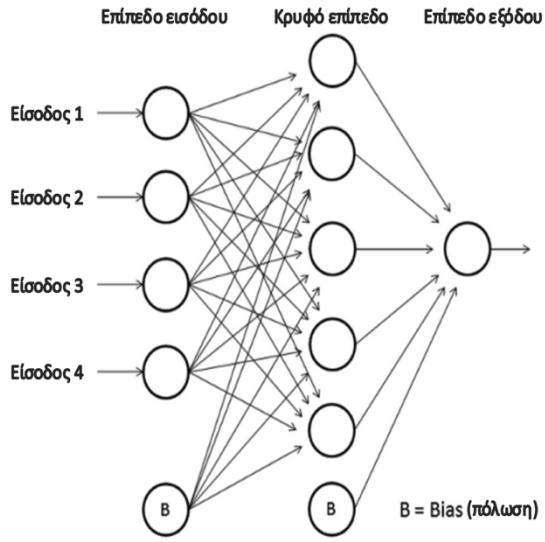
Εξίσωση 3.7: Εξίσωση της συνάρτησης softmax

Πάρθηκε από:
https://jamesmccaffrey.wordpress.com/2016/03/04/the-max-trick-when-computing-softmax/

3.3.3 Πολυστρωματικά νευρωνικά δίκτυα perceptron (Multilayer perceptron (MLP)) εμπρόσθιου περάσματος (Feedforward)

Τα δίκτυα αυτά δημιουργούνται με την ένωση πολλών μη γραμμικών τεχνητών νευρώνων McCulloh και Pitts. Συγκεκριμένα, οι νευρώνες αυτοί είναι οργανωμένοι σε επίπεδο εισόδου (input layer), εξόδου (output layer) και κρυφό επίπεδο (hidden layer). Κάθε νευρώνας είναι ενωμένος με ένα ή περισσότερους νευρώνες μόνο από τα επόμενα επίπεδα. Επίσης, όπως παρατηρήσαμε στο μοντέλο McCulloh και Pitts, υπάρχει και εδώ το bias, το οποίο είναι υπεύθυνο για να εξαφανίζει το κατώφλι. Το επίπεδο εισόδου χαρακτηρίζεται ως ανενεργό, αφού δεν κάνει κάποιαν επεξεργασία, παρά μόνο να εισάγει τα δεδομένα εισόδου στο δίκτυο. Από την άλλη, το κρυφό επίπεδο και το επίπεδο εξόδου χαρακτηρίζονται ως ενεργά επίπεδα, λόγω της επεξεργασίας που εκτελούν. Το κρυφό επίπεδο με βάση τα βάρη, τα οποία προσαρμόζονται συνεχώς κατά τη διάρκεια του δικτύου και που βρίσκονται στις ακμές που ενώνουν τους νευρώνες, αλλά και την τιμή κατωφλίου τους, αποθηκεύουν γνώση για τον υπολογισμό των εξόδων τους. Ο αριθμός των νευρώνων του επιπέδου εισόδου και εξόδου εξαρτάται αντίστοιχα από την κωδικοποίηση των δεδομένων εισόδου και τις κατηγορίες εξόδου ενός συγκεκριμένου προβλήματος. Για να αποφασιστεί ο κατάλληλος αριθμός των νευρώνων που θα υπάρχουν σε ένα ή περισσότερα κρυφά επίπεδα ενός τέτοιου δικτύου, πρέπει να εφαρμοστεί η τεχνική “δοκιμής και λάθους” (trial and error), δηλαδή να εκτελεστούν αρκετά πειράματα. Είναι απαραίτητο να βρεθούν οι κατάλληλες παράμετροι και το

δίκτυο να προσαρμοστεί ανάλογα με το πρόβλημα και αυτό πετυχαίνεται με την πιο πάνω τεχνική.



Σχήμα 3.8: Πολυστρωματικό δίκτυο perceptron εμπρόσθιου περάσματος

Πάρθηκε από: http://www.mdpi.com/1424-8220/12/7/8895/htm

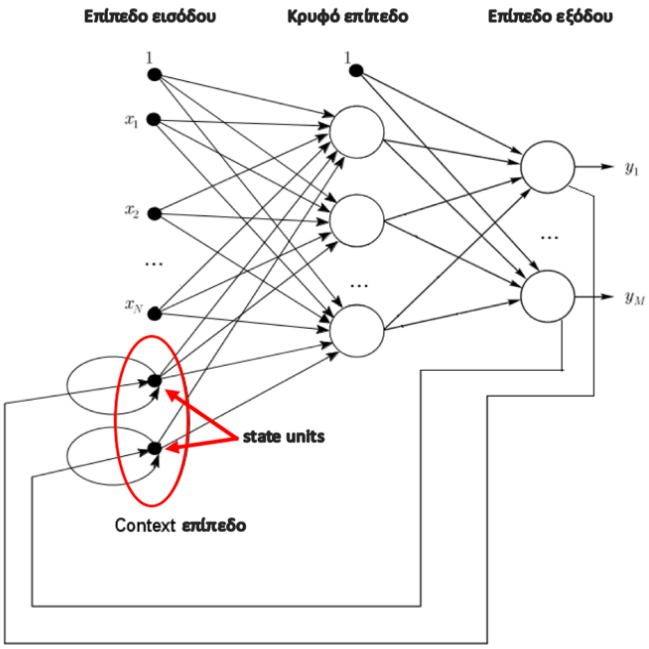
3.3.4 Νευρωνικό δίκτυο με ανάδραση (Recurrent Neural Network (RNN))

Το δίκτυο αυτό είναι παρόμοιο με το προηγούμενο πολυστρωματικό δίκτυο perceptron, αλλά με ανάδραση. Η συγκεκριμένη αρχιτεκτονική είναι κατάλληλη για δυναμικά προβλήματα αφού ένα από τα χαρακτηριστικά του είναι η ιδιότητα του να μοντελοποιεί την αίσθηση του χρόνου. Σε αυτή την αρχιτεκτονική, οι είσοδοι αποτελούνται από τα κύρια δεδομένα εισόδου του επιπέδου εισόδου και από τις εξόδους μιας προηγούμενης χρονικής στιγμής των νευρώνων των άλλων επιπέδων (είτε κρυφού, είτε εξόδου). Με το να λαμβάνει υπόψη τα αποτελέσματα των προηγούμενων χρονικών στιγμών για την έξοδο της παρούσας χρονικής στιγμής, το δίκτυο αυτό δημιουργεί ένα είδος εσωτερικής βραχυπρόθεσμης μνήμης. Οι πιο γνωστές αρχιτεκτονικές αυτού του τύπου δημιουργήθηκαν από τον Jordan το 1986 και από τον Elman το 1990.

Δίκτυο Jordan (Σχήμα 3.9)

Η ανάδραση σε αυτό το δίκτυο υπάρχει μεταξύ του επιπέδου εξόδου και του επιπέδου εισόδου μέσω ενός συνόλου επιπλέον νευρώνων που βρίσκονται στο context επίπεδο και ονομάζονται state units. Ο αριθμός των νευρώνων που υπάρχουν σε αυτό το επίπεδο ισούται με τον αριθμό των νευρώνων που υπάρχουν στο επίπεδο εξόδου. Οι ακμές μεταξύ

των νευρώνων αυτών των δύο επιπέδων έχουν τιμή 1. Το αποτέλεσμα από την έξοδο του δικτύου μεταφέρεται την επόμενη χρονική στιγμή στην είσοδο του (context επίπεδο).



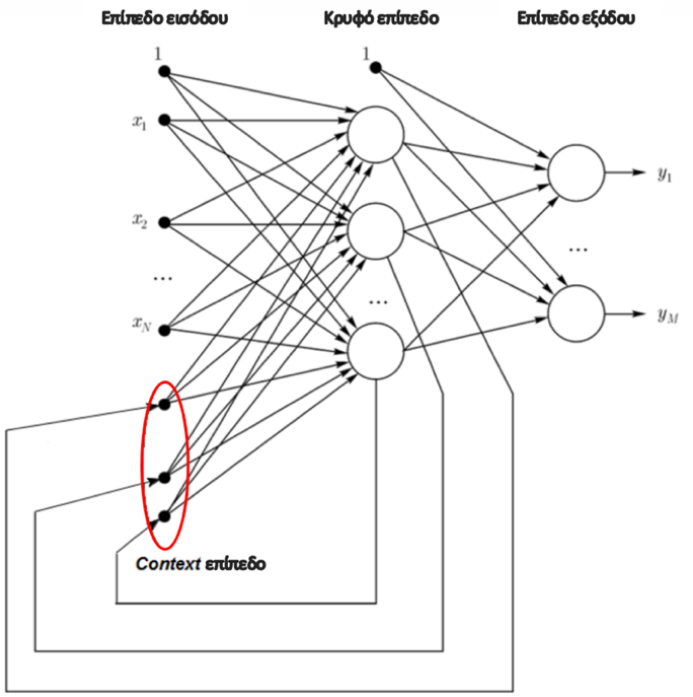
Σχήμα 3.9: Jordan νευρωνικό δίκτυο με ανάδραση

Πάρθηκε από: https://www.mql5.com/en/articles/1103

Με αυτό τον τρόπο υπολογίζεται το αποτέλεσμα της παρούσας χρονικής στιγμής ως συνάρτηση του προηγούμενου αποτελέσματος και της παρούσας εισόδου του δικτύου. Στην τοπικές αναδράσεις που υπάρχουν στο δίκτυο, πολλαπλασιάζουμε τις τιμές των πληροφοριών που ανατροφοδοτούνται με μια χρονική σταθερά. Αν η χρονική σταθερά είναι μικρή τότε το context επίπεδο θα κρατήσει την πληροφορία αυτή για μικρό χρονικό διάστημα, ενώ αν είναι μεγάλη για μεγάλο χρονικό διάστημα. Δηλαδή, η σταθερά αυτή καθορίζει το βάθος της εσωτερικής μνήμης που δημιουργείται στο δίκτυο. Το γεγονός ότι η ανατροφοδότηση γίνεται από το επίπεδο εξόδου, θεωρείται ως μειονέκτημα, επειδή το επίπεδο αυτό περιορίζεται από τα επιθυμητά αποτελέσματα.

Δίκτυο Elman

Το δίκτυο Elman αντιμετωπίζει το πιο πάνω πρόβλημα με το να γίνεται η ανάδραση, όπως φαίνεται στο Σχήμα 3.10, από το κρυφό επίπεδο και όχι από το επίπεδο εξόδου. Με αυτό τον τρόπο το δίκτυο κατασκευάζει τη δική του αναπαράσταση του χρόνου και το επίπεδο εξόδου δεν περιορίζεται από τις επιθυμητές εξόδους και είναι “ελεύθερο”. Η υπόλοιπη αρχιτεκτονική είναι η ίδια με το δίκτυο του Jordan.



Σχήμα 3.10: Elman νευρωνικό δίκτυο με ανάδραση

Πάρθηκε από: https://www.mql5.com/en/articles/1103

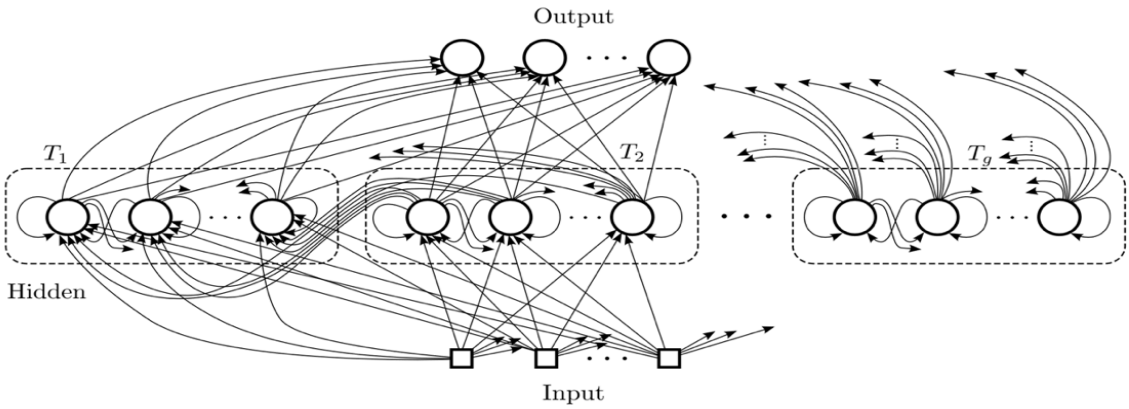
3.3.5 Clockwork νευρωνικό δίκτυο με ανάδραση (Clockwork RNN (CW-RNN); Koutník, 2014)

Η αρχιτεκτονική αυτού του δικτύου, είναι μια τροποποίηση της αρχιτεκτονικής ενός απλού νευρωνικού δικτύου με ανάδραση που μόλις αναφέραμε στο προηγούμενο υποκεφάλαιο. Το σημαντικό όμως είναι πως αυτή η αρχιτεκτονική είναι πιο απλή αλλά παράλληλα και πιο ισχυρή. Τα απλά νευρωνικά δίκτυα δεν μπορούν να αντιμετωπίσουν το πρόβλημα της εξαφανιζόμενης κλίσης (vanishing gradient), όπως αναφέραμε στο υποκεφάλαιο 1.1, με αποτέλεσμα να δυσκολεύονται να εκπαιδευτούν σε long-term εξαρτήσεις. Αντίθετα στα CW-RNNs, αυτό το πρόβλημα αντιμετωπίζεται με τα διάφορα τμήματα που βρίσκονται στο κρυφό επίπεδο, τα οποία χρησιμοποιούν διαφορετικά clock speeds, συγχρονίζουν τους υπολογισμούς τους με διαφορετικά, διακριτά clock periods.

Περιέχει, όπως και ένα απλό νευρωνικό δίκτυο με ανάδραση, ένα επίπεδο εισόδου, εξόδου και το κρυφό επίπεδο. Υπάρχουν εμπρόσθιες συνδέσεις από το επίπεδο εισόδου στο κρυφό επίπεδο και από το κρυφό επίπεδο στο επίπεδο εξόδου. Οι νευρώνες του κρυφού επιπέδου είναι χωρισμένοι σε g τμήματα κάποιου μεγέθους k. Σε κάθε τμήμα ανατίθεται κάποιο clock period $T_n \in \{T_1, \ldots, T_g\}$, όπου η επιλογή των περιόδων $\{T_1, \ldots,$

T_g} είναι τυχαία. Σε κάθε τμήμα οι νευρώνες είναι πλήρως διασυνδεδεμένοι, αλλά υπάρχουν συνδέσεις ανάδρασης από νευρώνες, οι οποίοι βρίσκονται στο πιο αργό τμήμα j, σε νευρώνες που βρίσκονται στο πιο γρήγορο τμήμα i, μόνο αν ισχύει ότι clock period $T_i < T_j$.

Όπως φαίνεται και στο Σχήμα 3.11, ταξινομώντας τα τμήματα με την περίοδο να αυξάνεται από το T_1 στο T_g, τα πιο αργά τμήματα μεταδίδουν το αποτέλεσμα τους στα πιο γρήγορα τμήματα. Σε κάθε χρονική στιγμή t, μόνο οι έξοδοι των τμημάτων i, τα οποία ικανοποιούν τη συνθήκη (t MOD T_i) = 0, είναι ενεργοί. Τα low-clock-rate τμήματα επεξεργάζονται, διατηρούν και παράγουν την long-term πληροφορία, η οποία λήφθηκε από τις ακολουθίες εισόδου. Τα high-speed τμήματα επικεντρώνονται στις τοπικές, ψηλών συχνοτήτων πληροφορίες. Στο πέρασμα προς τα πίσω (backward pass; υποκεφάλαιο 3.4.7), το σφάλμα μεταδίδεται μόνο από τα τμήματα τα οποία εκτελέστηκαν στη χρονική στιγμή t. Το σφάλμα των τμημάτων που δεν εκτελέστηκαν, αντιγράφεται πίσω στον χρόνο, όπου προστίθεται στο back-propagated σφάλμα.

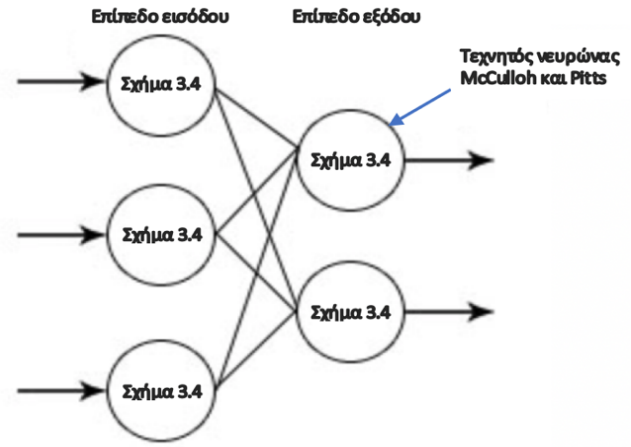


Σχήμα 3.11: Η αρχιτεκτονική του CW-RNN είναι παρόμοια με ενός απλού RNN με ένα επίπεδο εισόδου, εξόδου και κρυφό επίπεδο. Το κρυφό επίπεδο απαρτίζεται από g τμήματα όπου το κάθε ένα έχει το δικό του clock rate και οι νευρώνες του είναι πλήρως διασυνδεδεμένοι. Οι νευρώνες στο πιο γρήγορο τμήμα *i* συνδέονται με τους νευρώνες στο πιο αργό τμήμα *j* μόνο αν το clock period $T_i < T_j$. (Koutník et al., 2014)

3.4 Αλγόριθμοι μάθησης τεχνητών νευρωνικών δικτύων

3.4.1 Αλγόριθμος μάθησης Perceptron (Perceptron learning algorithm)

Τα μονοεπίπεδα δίκτυα (Σχήμα 3.12), τα οποία αποτελούνται από τεχνητούς νευρώνες McCulloh και Pitts, μπορούν να εκπαιδευτούν με τον αλγόριθμο μάθησης Perceptron, ο οποίος είναι αλγόριθμος επιβλεπόμενης μάθησης.



Σχήμα 3.12: Μονοεπίπεδο Perceptron

Πάρθηκε από: http://neuroph.sourceforge.net/tutorials/Perceptron.html

Αρχικά, αρχικοποιούνται τα βάρη και τα κατώφλια με μικρές τυχαίες τιμές. Στο δίκτυο παρουσιάζονται τα δεδομένα εισόδου και οι επιθυμητές έξοδοι. Στη συνέχεια, υπολογίζονται οι τιμές των πραγματικών εξόδων, συγκρίνονται με τις τιμές των επιθυμητών εξόδων και ανάλογα προσαρμόζονται τα βάρη και τα κατώφλια. Η διαδικασία αυτή βασίζεται στον κανόνα Δέλτα (Εξίσωση 3.8) (Widrow and Hoff, 1960), ο οποίος δείχνει την απόκλιση που υπάρχει μεταξύ πραγματικού και επιθυμητού αποτελέσματος. Η αλλαγή στα βάρη είναι ανάλογη του πολλαπλασιασμού του λάθους Δ (Εξίσωση 3.9), της εισόδου του δικτύου x_i και του ρυθμού μάθησης η ($0 \leq \eta \leq 1$), ο οποίος καθορίζει πόσο γρήγορα συγκλίνει η μάθηση.

$$w_i(t+1) = w_i(t) + \eta \Delta x_i(t)$$

Εξίσωση 3.8: Κανόνας Δέλτα (Delta rule)

w_i: βάρος νευρώνα i; **η:** ρυθμός μάθησης;
Δ: Εξίσωση 3.9; **x_i:** είσοδος νευρώνα i

Το λάθος Δ είναι θετικό και το βάρος της παρούσας χρονικής στιγμής $w_i(t+1)$ αυξάνεται εάν το επιθυμητό αποτέλεσμα είναι ένα, ενώ είναι αρνητικό και το συγκεκριμένο βάρος

μειώνεται αν το επιθυμητό αποτέλεσμα είναι μηδέν. Αν το Δ είναι μηδέν, τότε δε γίνεται καμιά αλλαγή και το βάρος $w_i(t+1)$ ισούται με το βάρος της προηγούμενης χρονικής στιγμής $w_i(t)$.

$$\Delta = d(t) - y(t)$$

Εξίσωση 3.9: Το λάθος Δ

d(t): Επιθυμητό αποτέλεσμα τη χρονική στιγμή t;
y(t): Πραγματικό αποτέλεσμα τη χρονική στιγμή t

Η διαδικασία επαναλαμβάνεται ξανά από το βήμα της παρουσίασης των εισόδων και των επιθυμητών εξόδων στο δίκτυο και αυτό συνεχίζεται είτε μέχρι το δίκτυο να συγκλίνει, είτε μέχρι ένα συγκεκριμένο αριθμό εποχών. Μια εποχή είναι η είσοδος όλων των δεδομένων εισόδου και επιθυμητών εξόδων στο δίκτυο.

3.4.2 Μέθοδος κατάβασης κλίσης (Gradient Descent (GD))

Η μέθοδος κατάβασης κλίσης αποτελεί μια από τις πιο διαδεδομένες μεθόδους μάθησης, όπου οι νευρώνες του δικτύου διέπονται από μια παραγωγίσιμη συνάρτηση ενεργοποίησης. Αυτή η μέθοδος χρησιμοποιείται μόνο για επιβλεπόμενη μάθηση και προσπαθεί να προσαρμόσει τα βάρη του δικτύου για να ελαχιστοποιηθεί το ολικό τετραγωνικό σφάλμα E (Εξίσωση 3.10). Ο πιο συνήθης αλγόριθμος που χρησιμοποιείται για να υπολογιστεί το σφάλμα των εξόδων των νευρώνων είναι ο αλγόριθμος ελαχίστων τετραγώνων (Least Mean Square (LMS)), ο οποίος λαμβάνει υπόψη του την πραγματική και επιθυμητή έξοδο.

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

Εξίσωση 3.10: Συνάρτηση σφάλματος E

t_{pj}: επιθυμητό αποτέλεσμα του νευρώνα j;
o_{pj}: πραγματικό αποτέλεσμα του νευρώνα j

Συγκεκριμένα, η κύρια ιδέα της μεθόδου είναι η αλλαγή των βαρών του δικτύου να είναι ανάλογη ως προς το αρνητικό της παραγώγου της συνάρτησης του σφάλματος ως προς τα βάρη (Εξίσωση 3.11).

$$\Delta w_{ij} = -\eta \frac{\theta E_p}{\theta w_{ij}}$$

Εξίσωση 3.11: Η αλλαγή των βαρών του δικτύου είναι ανάλογη ως προς το αρνητικό της παραγώγου της συνάρτησης του σφάλματος ως προς τα βάρη

η: ρυθμός μάθησης; **E_p:** Εξίσωση 3.10;

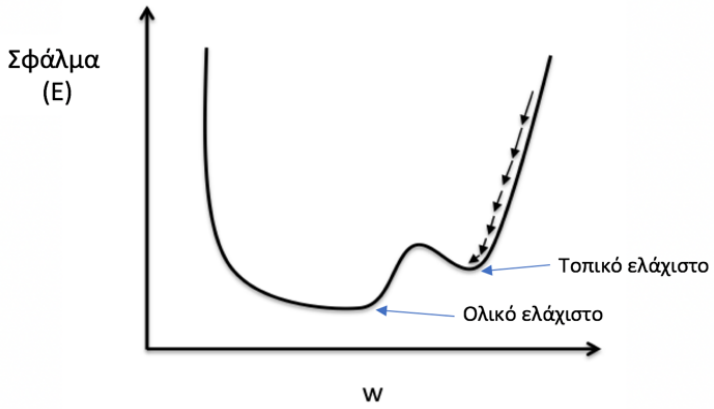
Στη συνέχεια, το αποτέλεσμα αυτό εφαρμόζεται στην Εξίσωση 3.12 για την αλλαγή των βαρών.

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} \sum_i x_i$$

Εξίσωση 3.12: Ανανέωση των βαρών του δικτύου

w_{ij}: βάρος νευρώνα i;
Δw_{ij}: Εξίσωση 3.11; **x_i:** είσοδος νευρώνα i

Η μέθοδος αυτή, προσπαθεί να αλλάξει τα βάρη του δικτύου με τέτοιο τρόπο έτσι ώστε το σφάλμα να είναι το ελάχιστο και οι τιμές των πραγματικών εξόδων του δικτύου να είναι όσο πιο κοντά στις επιθυμητές. Για να επιτευχθεί το ελάχιστο σφάλμα είναι ανάγκη να φτάσουμε στο ελάχιστο σημείο της συνάρτησης που χρησιμοποιείται, το οποίο ονομάζεται ολικό ελάχιστο. Δυστυχώς, όταν προσπαθούμε να μειώσουμε το σφάλμα της συνάρτησης για να το επιτύχουμε αυτό, μπορεί οι τιμές των βαρών να εγκλωβιστούν σε τοπικό ελάχιστο, όπως μπορούμε να παρατηρήσουμε στο Σχήμα 3.13. Αν γίνει αυτό, σημαίνει ότι το δίκτυο δε θα δίνει τα καλύτερα δυνατά αποτελέσματα.



Σχήμα 3.13: Στην προσπάθεια μείωσης του λάθους, μπορεί να εγκλωβιστεί σε τοπικό ελάχιστο

Πάρθηκε από: https://wiki.tum.de/display/lfdv/Adaptive+Learning+Rate+Method

Αυτό μπορεί να αντιμετωπιστεί με διάφορους εμπειρικούς κανόνες, οι οποίοι μπορούν να μειώσουν την εμφάνιση των τοπικών ελαχίστων. Οι κανόνες αυτοί είναι η μείωση του ρυθμού μάθησης, η πρόσθεση επιπλέον εσωτερικών νευρώνων, η πρόσθεση ενός επιπλέον όρου στην εξίσωση προσαρμογής των βαρών, την ορμή, ή η προσθήκη "θορύβου".

3.4.3 Μέθοδος στοχαστικής κατάβασης κλίσης (Stochastic Gradient Descent (SGD))

Στη μέθοδο κατάβασης κλίσης που εξηγήσαμε στο προηγούμενο υποκεφάλαιο, υπολογίζεται το σφάλμα βάση ολόκληρου του συνόλου δεδομένων. Ως επακόλουθο, όταν χρησιμοποιούνται μεγάλα σύνολα δεδομένων, τα βάρη του δικτύου προσαρμόζονται πιο αργά και έτσι χρειάζεται περισσότερο χρόνο για να συγκλίνει στο ολικό ελάχιστο. Σε αυτό το είδος εκπαίδευσης, γίνεται η αναπροσαρμογή των βαρών μετά από κάθε δεδομένο εκπαίδευσης και γι' αυτό ονομάζεται και on-line Gradient Descent.

Αρχικά, αρχικοποιούνται τα βάρη με μικρές τυχαίες τιμές και δίνεται μια τιμή στο ρυθμό μάθησης. Για κάθε δεδομένο εκπαίδευσης, για κάθε του βάρος χρησιμοποιούνται οι πιο κάτω Εξισώσεις 3.13 και 3.14 για τον υπολογισμό της νέας του τιμής. Η διαδικασία αυτή επαναλαμβάνεται για ένα αριθμό εποχών, είτε μέχρι να ελαχιστοποιηθεί το σφάλμα.

$$\Delta w_{ij} = -\eta(t_i - o_i)$$

Εξίσωση 3.13: Η αλλαγή των βαρών του δικτύου

η: ρυθμός μάθησης;
t_i: επιθυμητό αποτέλεσμα του νευρώνα i;
o_i: πραγματικό αποτέλεσμα του νευρώνα i

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} x_i$$

Εξίσωση 3.14: Ανανέωση των βαρών του δικτύου

w_{ij}: βάρος νευρώνα i;
Δw_{ij}: Εξίσωση 3.13; **x_i:** είσοδος νευρώνα i

3.4.4 Μέθοδος mini-batch κατάβασης κλίσης (Mini-Batch Gradient Descent (MB-GD))

Η συγκεκριμένη μέθοδος είναι κάτι ενδιάμεσο των μεθόδων κατάβασης κλίσης (υποκεφάλαιο 3.4.2) και στοχαστικής κατάβασης κλίσης (υποκεφάλαιο 3.4.3). Η διαφορά με τις άλλες μεθόδους είναι ότι η ανανέωση των βαρών του δικτύου γίνεται μετά από ένα σύνολο δεδομένων εκπαίδευσης μεγέθους g, όπου 1 < g < μέγεθος ολόκληρου του συνόλου δεδομένων εκπαίδευσης. Τα δύο άκρα αυτής της σχέσης αντιστοιχούν στις δύο άλλες μεθόδους. Δηλαδή, το 1 αντιστοιχεί στη μέθοδο SGD, αφού μετά από κάθε ένα δεδομένο εκπαίδευσης, υπολογίζονται τα νέα βάρη του δικτύου, ενώ στη μέθοδο GD, ο υπολογισμός για την ανανέωση των βαρών του δικτύου γίνεται αφότου εισαχθεί στο δίκτυο ολόκληρο το σύνολο δεδομένων εκπαίδευσης. Λόγω του ότι η ανανέωση των βαρών γίνεται πιο συχνά από τη GD, η συγκεκριμένη μέθοδος συγκλίνει πιο γρήγορα. Ακόμη, είναι υπολογιστικά πιο αποδοτική από την SGD, αφού με την online εκπαίδευση της SGD έχουμε συνεχή ανανέωση των βαρών και αυτό την καθιστά υπολογιστικά πιο ακριβή.

3.4.5 Adam (Adaptive Moment Estimation) optimizer (Kingma and Lei Ba, 2017)

Πριν να προχωρήσουμε στην εξήγηση και περιγραφή του Adam, θα αναφερθούμε στους αλγορίθμους μάθησης RMSProp και AdaGrad, από τους οποίους υιοθέτησε κάποια χαρακτηριστικά και χρησιμοποίησε κάποια στοιχεία τους.

RMSProp (Root Mean Square Propagation)

Ο RMSProp προσαρμόζει τον ρυθμό μάθησης για κάθε διαφορετική παράμετρο στο δίκτυο. Συγκεκριμένα, υπολογίζει ένα κινούμενο μέσο όρο (moving average) του τετραγώνου της κλίσης για κάθε βάρος. Άρα, όπως παρατηρείται και στην Εξίσωση 3.15, η τιμή αυτή υπολογίζεται με βάση την τιμή του μέσου όρου της προηγούμενης στιγμής και του τετραγώνου της παρούσας κλίσης. Στη συνέχεια, για να υπολογιστεί η νέα τιμή

$$MS(t) = 0.9 * MS(t-1) + 0.1(g(t))^2$$

Εξίσωση 3.15: Κινούμενος μέσος όρος

$(g(t))^2$: το τετράγωνο της κλίσης της συνάρτησης που χρησιμοποιείται

των παραμέτρων χρησιμοποιείται η Εξίσωση 3.16.

$$w(t+1) = w(t) - \eta \frac{g(t)}{\sqrt{MS(t) + \varepsilon}}$$

Εξίσωση 3.16: Ανανέωση των παραμέτρων-βαρών του δικτύου

MS(t): Εξίσωση 3.15;
ε: μικρή σταθερά για αποτροπή διαίρεσης με το μηδέν;
g(t): κλίση της συνάρτησης που χρησιμοποιείται

AdaGrad (Adaptive Gradient Algorithm)

Ο AdaGrad προσαρμόζει τον ρυθμό μάθησης σε κάθε παράμετρο. Συγκεκριμένα, εκτελεί μεγάλες αλλαγές σε παραμέτρους που εμφανίζονται σπάνια και θεωρείται ότι περιέχουν σημαντική πληροφορία και εκτελεί μικρότερες αλλαγές σε παραμέτρους που εμφανίζονται πιο συχνά. Γι' αυτό το λόγο είναι κατάλληλος για πιο "αραιά" δεδομένα.

Adam (Adaptive Moment Estimation)

Ο Adam είναι ένας πιο προηγμένος αλγόριθμος μάθησης από τους προηγούμενους που αναφέρθηκαν στα προηγούμενα υποκεφάλαια. Η κλίση που χρησιμοποιείται σε κάθε επανάληψη, υπολογίζεται με βάση την κλίση της προηγούμενης στιγμής, χρησιμοποιώντας μια τεχνική βασισμένη στην ορμή (momentum). Ο σκοπός του momentum είναι να κάνει πιο γρήγορη την εκπαίδευση του δικτύου, αφού προσπαθεί να βοηθήσει την εκπαίδευση του να "διαφεύγει" από τα τοπικά ελάχιστα, με το να τα προσπερνά. Επίσης, ο Adam, υπολογίζει ανάλογα για κάθε παράμετρο διαφορετικό ρυθμό μάθησης με εκτιμήσεις της πρώτης και δεύτερης στιγμής των κλίσεων. Αυτό έχει ως αποτέλεσμα την πιο γρήγορη εκπαίδευση του δικτύου όταν απαιτείται συγκεκριμένη τιμή του ρυθμού μάθησης για κάθε διαφορετική παράμετρο.

Οι παράμετροι που χρησιμοποιούνται για τον αλγόριθμο αυτό είναι το beta1, το beta2, το η (stepsize) και το ε (epsilon). Το beta1 χρησιμοποιείται για την μείωση του μέσου όρου της κλίσης, ενώ το beta2 χρησιμοποιείται για την μείωση του μέσου όρου του τετραγώνου της κλίσης. Δηλαδή, το beta1 είναι εκθετικός ρυθμός μείωσης για τις εκτιμήσεις της πρώτης στιγμής (μέσος όρος) της κλίσης και το beta2 για τις εκτιμήσεις

της δεύτερης στιγμής (uncentered variance - διαφορά) της κλίσης. Το η είναι ο ρυθμός μάθησης και το ε είναι μια μικρή σταθερά που χρησιμοποιείται για αποτροπή διαίρεσης με το μηδενικό σφάλμα.

Αλγόριθμος

Αρχικά, αρχικοποιούνται τα m(0) και v(0) με 0, τα οποία είναι διανύσματα για την 1η και 2η στιγμή της κλίσης αντίστοιχα. Στη συνέχεια, υπολογίζεται ο καινούργιος ρυθμός μάθησης με την Εξίσωση 3.17.

$$\eta(t) = \eta(t-1) * \frac{\sqrt{1-(beta_2)^t}}{1-(beta_1)^t}$$

Εξίσωση 3.17: Ρυθμός μάθησης τη χρονική στιγμή t

beta$_1$: εκθετικός ρυθμός μείωσης για τις εκτιμήσεις της πρώτης στιγμής;
beta$_2$: εκθετικός ρυθμός μείωσης για τις εκτιμήσεις της δεύτερης στιγμής

Έπειτα, χρησιμοποιώντας το διάνυσμα των κλίσεων g που παίρνουμε από την συνάρτηση που χρησιμοποιούμε, υπολογίζονται η νέα πρώτη και δεύτερη biased εκτίμηση m(t) και v(t) με τις Εξισώσεις 3.18 και 3.19.

$$m(t) = beta_1 * m(t-1) + (1 - beta_1) * g$$

Εξίσωση 3.18: Biased εκτίμηση της **πρώτης** στιγμής τη χρονική στιγμή t

beta$_1$: εκθετικός ρυθμός μείωσης για τις εκτιμήσεις της πρώτης στιγμής;
g: διάνυσμα των κλίσεων που παίρνουμε από την συνάρτηση που χρησιμοποιούμε

$$v(t) = beta_2 * v(t-1) + (1 - beta_2) * g^2$$

Εξίσωση 3.19: Biased εκτίμηση της **δεύτερης** στιγμής τη χρονική στιγμή t

beta$_2$: εκθετικός ρυθμός μείωσης για τις εκτιμήσεις της δεύτερης στιγμής;
g: διάνυσμα των κλίσεων που παίρνουμε από την συνάρτηση που χρησιμοποιούμε

Τα αποτελέσματα από τις προηγούμενες δύο εξισώσεις χρησιμοποιούνται για τον υπολογισμό της bias-corrected πρώτης και δεύτερης εκτίμησης, όπως παρατηρείται στις Εξισώσεις 3.20 και 3.21.

$$\hat{m}(t) = \frac{m(t)}{\left(1 - beta_1{}^t\right)}$$

Εξίσωση 3.20: Bias-corrected εκτίμηση της **πρώτης** στιγμής τη χρονική στιγμή t

beta$_1$: εκθετικός ρυθμός μείωσης για τις εκτιμήσεις της πρώτης στιγμής

$$\hat{v}(t) = \frac{v(t)}{\left(1 - beta_2{}^t\right)}$$

Εξίσωση 3.21: Bias-corrected εκτίμηση της **δεύτερης** στιγμής τη χρονική στιγμή t

beta$_2$: εκθετικός ρυθμός μείωσης για τις εκτιμήσεις της δεύτερης στιγμής

Τέλος, υπολογίζεται η καινούργια τιμή των παραμέτρων με την Εξίσωση 3.22 και αυτά τα βήματα επαναλαμβάνονται μέχρι να συγκλίνουν όλες οι παράμετροι.

$$w(t) = w(t-1) - \eta \frac{\hat{m}(t)}{\sqrt{\hat{v}(t)} + epsilon}$$

Εξίσωση 3.22: Καινούργια τιμή των παραμέτρων

η: ρυθμός μάθησης;
epsilon: μικρή σταθερά (προκαθορισμένη τιμή = 10^{-8})

3.4.6 Αλγόριθμος μάθησης ανάστροφης μετάδοσης σφάλματος (Backpropagation algorithm)

Ονομάζεται επίσης και γενικευμένος κανόνας Δέλτα (Werbos, 1974; Rumelhart et al., 1986) και χρησιμοποιείται για επιβλεπόμενη μάθηση. Τα νευρωνικά δίκτυα Perceptron μπορούν να εκπαιδευτούν με αυτό τον αλγόριθμο, ο οποίος προσαρμόζει ανάλογα τα βάρη και το κατώφλι του δικτύου για την ελαχιστοποίηση του σφάλματος με τη μετάδοση του προς τα πίσω. Ο αλγόριθμος αυτός χωρίζεται σε δύο φάσεις, το εμπρόσθιο πέρασμα (forward pass) και το πέρασμα προς τα πίσω (backward pass). Πριν αρχίσει η διαδικασία αυτή τα βάρη και τα κατώφλια του δικτύου αρχικοποιούνται με μικρές τυχαίες τιμές (-1,1).

Εμπρόσθιο πέρασμα

Σε αυτή τη φάση τα δεδομένα εισόδου και εξόδου παρουσιάζονται στο δίκτυο και υπολογίζονται αρχικά οι πραγματικές έξοδοι των νευρώνων του κρυφού επιπέδου, οι οποίοι εισέρχονται στο επόμενο επίπεδο του δικτύου. Στη συνέχεια, υπολογίζονται οι πραγματικές έξοδοι του επιπέδου εξόδου, όπου και δίνεται το τελικό αποτέλεσμα. Μετέπειτα, γνωρίζοντας την επιθυμητή έξοδο-αποτέλεσμα, υπολογίζεται το σφάλμα για τους νευρώνες του επιπέδου εξόδου με την Εξίσωση 3.23, το οποίο θα χρησιμοποιηθεί στη δεύτερη φάση. Σημαντικό εδώ να αναφέρουμε ότι οι Εξισώσεις 3.23 και 3.25 προκύπτουν αφού εφαρμοσθεί η μέθοδος κατάβασης κλίσης σε ένα πολυστρωματικό δίκτυο Perceptron.

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

Εξίσωση 3.23: Υπολογισμός σφάλματος στους νευρώνες του επιπέδου εξόδου

t_{pj}: επιθυμητό αποτέλεσμα του νευρώνα j;
o_{pj}: πραγματικό αποτέλεσμα του νευρώνα j

Πέρασμα προς τα πίσω

Το σφάλμα που υπολογίστηκε, μεταφέρεται προς τα πίσω. Τα βάρη των ακμών που ενώνουν το κρυφό επίπεδο και το επίπεδο εξόδου προσαρμόζονται ανάλογα με βάση τον γενικευμένο κανόνα Δέλτα (Εξίσωση 3.24).

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

Εξίσωση 3.24: Γενικευμένος κανόνας Δέλτα

w_{ij}: το βάρος που συνδέει τον νευρώνα i και j;
η: ρυθμός μάθησης;
δ: Εξίσωση 3.23 ή 3.25 ανάλογα σε ποιο επίπεδο βρίσκονται τα βάρη

Τα υπόλοιπα βάρη του δικτύου, δηλαδή τα βάρη των κρυφών επιπέδων του δικτύου, υπολογίζονται και προσαρμόζονται ανάλογα με την Εξίσωση 3.25. Με τις ίδιες εξισώσεις ανανεώνονται και οι τιμές των κατωφλίων.

$$\delta_{pj} = o_{pj}(1 - o_{pj}) \sum_{\kappa} (\delta_{p\kappa} - w_{j\kappa})$$

Εξίσωση 3.25: Υπολογισμός σφάλματος στους νευρώνες του κρυφού επιπέδου

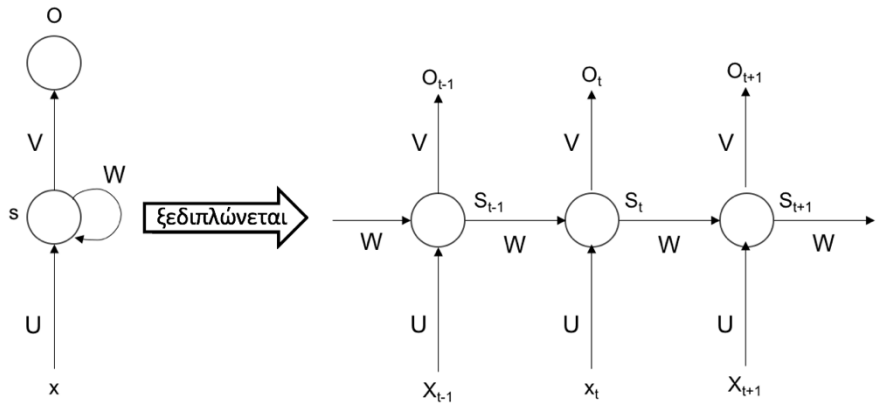
$\delta_{p\kappa}$: το σφάλμα από τον νευρώνα κ ο οποίος είναι συνδεδεμένος με τον νευρώνα j (δίνεται από την Εξίσωση 3.23);
$w_{j\kappa}$: το βάρος που συνδέει τον νευρώνα j και κ;
η: ρυθμός μάθησης;
t_{pj}: επιθυμητό αποτέλεσμα του νευρώνα j;
o_{pj}: πραγματικό αποτέλεσμα του νευρώνα j

Με τα δύο αυτά περάσματα ολοκληρώνεται μια εποχή και η διαδικασία αυτή επαναλαμβάνεται μέχρι να μειωθεί σε ικανοποιητικό βαθμό το σφάλμα.

3.4.7 Αλγόριθμος μάθησης ανάστροφης μετάδοσης σφάλματος στο χρόνο (Backpropagation Through Time (BPTT))

Ο αλγόριθμος αυτός είναι μια τροποποίηση του αλγορίθμου που αναφέρθηκε στο υποκεφάλαιο 3.4.3. Συγκεκριμένα, έγιναν κάποιες αναγκαίες αλλαγές στον προηγούμενο αλγόριθμο για να είναι εφαρμόσιμος σε νευρωνικά δίκτυα με ανάδραση, τα οποία δημιουργούν ένα είδος μνήμης.

Αρχικά, όταν χρησιμοποιείται ο αλγόριθμος μάθησης ανάστροφης μετάδοσης σφάλματος στο χρόνο (Werbos, 1990), παρουσιάζονται στο δίκτυο τα δεδομένα εισόδου και τα αντίστοιχα τους επιθυμητά αποτελέσματα. Στη συνέχεια, το δίκτυο "ξεδιπλώνεται" (Σχήμα 3.14) και το σφάλμα, το οποίο θα ληφθεί υπόψη στο επόμενο βήμα, υπολογίζεται και προστίθεται κάθε χρονική στιγμή. Όταν υπολογιστεί το τελικό σφάλμα κινείται προς τα πίσω στο ξεδιπλωμένο δίκτυο και με βάση την τιμή του σφάλματος προσαρμόζονται ανάλογα τα βάρη και το κατώφλι και η διαδικασία αυτή επαναλαμβάνεται.



Σχήμα 3.14: Το αποτέλεσμα ξεδιπλώματος ενός νευρωνικού δικτύου με ανάδραση

X_{t-1}, X_t, X_{t+1}: εισόδοι του δικτύου κάθε χρονική στιγμή
W,U,V: βάρη του δικτύου
O_{t-1}, O_t, O_{t+1}: έξοδοι του δικτύου κάθε χρονική στιγμή

Πάρθηκε από: https://www.safaribooksonline.com/library/view/deep-learning-for/9781788295628/20f639d4-8079-4137-b613-c8a479e6f2cf.xhtml

Όπως έχει αναφερθεί και πιο πάνω, για την εκπαίδευση του δικτύου με το συγκεκριμένο αλγόριθμο απαιτείται το ξεδίπλωμα του (Mozer, 1989). Όπως παρατηρείται και στο Σχήμα 3.15, με το ξεδίπλωμα του νευρωνικού δικτύου με ανάδραση, δημιουργείται ένα δίκτυο χωρίς ανάδραση και για κάθε χρονική στιγμή προστίθεται ένα επίπεδο. Με αυτή τη διαδικασία, αυτός ο αλγόριθμος, μπορεί να γίνει υπολογιστικά ακριβός και το δίκτυο γίνεται πιο πολύπλοκο και απαιτεί μεγάλη χωρητικότητα.

Κεφάλαιο 4

Μεθοδολογία

4.1 Δεδομένα εισόδου

4.1.1 Σύνολο δεδομένων CB513

4.1.2 Αρχεία πολλαπλής στοίχισης (MSA profiles)

4.1.3 Δομές δεδομένων εισόδου

4.2 Προσαρμογή υλοποίησης CW-RNN στο πρόβλημα PSSP

4.2.1 Υλοποίηση και αλλαγές

4.2.2 Αρχιτεκτονική

4.2.3 Εκπαίδευση του μοντέλου

4.1 Δεδομένα εισόδου

Είναι ευρέως γνωστό και αποδεκτό πως ένας σημαντικός παράγοντας για την επιτυχή εκπαίδευση ενός νευρωνικού δικτύου, η οποία θα του επιτρέψει να δώσει τα καλύτερα αποτελέσματα, θεωρείται το σύνολο δεδομένων εισόδου που θα χρησιμοποιηθεί για την εκπαίδευση του. Ανάλογα με το πρόβλημα, τα δεδομένα εισόδου επεξεργάζονται με κατάλληλο τρόπο, έτσι ώστε το νευρωνικό δίκτυο να έχει τη δυνατότητα να βρίσκει πιο εύκολα συσχετίσεις μεταξύ τους για να μπορεί να γενικεύει καλύτερα. Εκτός όμως από την εκπαίδευση του δικτύου, σημαντικό κομμάτι είναι και η επαλήθευση του. Συγκεκριμένα, ένα σύνολο δεδομένων εισόδου, χωρίζεται σε δύο υποσύνολα, τα δεδομένα εκπαίδευσης και τα δεδομένα επαλήθευσης. Με αυτό τον τρόπο, όταν το δίκτυο εκπαιδευτεί, του παρουσιάζονται τα δεδομένα επαλήθευσης, ένα υποσύνολο δεδομένων άγνωστα προς αυτό, για να ελεγχθεί κατά πόσο το δίκτυο μαθαίνει και μέχρι ποιο βαθμό. Στόχος του παρόντος προβλήματος είναι η πρόβλεψη της δευτεροταγούς δομής μιας πρωτεΐνης εισάγοντας στο δίκτυο την πρωτοταγή της δομή. Η μάθηση που χρησιμοποιεί το δίκτυο είναι επιβλεπόμενη, καθιστώντας την γνώση της δευτεροταγούς δομής των πρωτεϊνών απαραίτητη για την εκπαίδευση του.

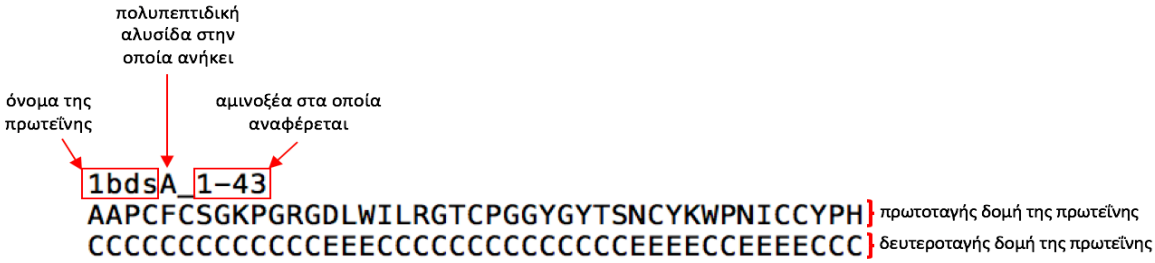
Στα πλαίσια αυτής της διπλωματικής εργασίας, χρησιμοποιήθηκε το σύνολο δεδομένων CB513 (Cuff and Burton, 1999) σε συνδυασμό με τα αρχεία πολλαπλής στοίχισης (Multiple Sequence Alignment (MSA) profiles).

4.1.1 Σύνολο δεδομένων CB513

Το σύνολο CB513 δημιουργήθηκε με βάση ένα μέρος του μεγαλύτερου συνόλου δεδομένων pdb_select25 (Cuff and Burton, 1999). Για την επιλογή των πρωτεϊνών που χρησιμοποιήθηκαν για τη δημιουργία του συνόλου αυτού, τέθηκαν κάποια κριτήρια. Οι πρωτεΐνες που περιέχει το σύνολο pdb_select25 έχουν μικρό συντελεστή ομοιότητας (μικρότερο ή ίσο με 25%). Αυτό είναι ένα κρίσιμο κριτήριο, αφού έτσι το δίκτυο μπορεί να εκπαιδευτεί σε ένα σύνολο δεδομένων με ποικιλομορφία. Ως αποτέλεσμα, το ικανοποιητικό ποσοστό ανομοιότητας που υπάρχει μεταξύ των πρωτεϊνών, βοηθάει το νευρωνικό δίκτυο να μην υπερεκπαιδευτεί. Όταν ένα δίκτυο εκπαιδεύεται με δεδομένα τα οποία είναι όμοια μεταξύ τους, τότε υπερεκπαιδεύεται και χάνει τη δυνατότητα του

να γενικεύει. Ακόμη, επιλέχθηκαν οι πρωτεΐνες των οποίων η δευτεροταγής δομή προβλέφθηκε με χρήση της μεθόδου κρυσταλλογραφίας ακτινών Χ ή με τη μέθοδο πυρηνικού μαγνητικού συντονισμού (NMR), ανήκουν στη βάση δεδομένων PDB (Protein Data Bank), η δευτεροταγής τους δομή είναι κωδικοποιημένη με τη μορφή του προγράμματος DSSP (Πίνακας 1) και τα αμινοξέα που τις απαρτίζουν έπρεπε να τηρούν συγκεκριμένα κριτήρια που σχετίζονται με αυτά. Επίσης, το σύνολο δεδομένων CB513 χρησιμοποιείται ευρέως και έτσι κάνει εφικτή και πιο εύκολη τη σύγκριση μεταξύ των διάφορων μεθόδων ως προς την ευκρίνεια και την αποδοτικότητα τους.

Το σύνολο αυτό χωρίστηκε σε δύο αρχεία. Το ένα αρχείο περιείχε σχεδόν το 90% του συνόλου όπου αντιπροσώπευε τα δεδομένα εκπαίδευσης, ενώ το υπόλοιπο (≈)10% ήταν τα δεδομένα επαλήθευσης. Η μορφή των πρωτεϊνών σε αυτά τα αρχεία διαδραματίζεται στο σχήμα 4.1. Οι πρώτοι τέσσερις χαρακτήρες της πρώτης γραμμής είναι το όνομα της πρωτεΐνης, ο πέμπτος χαρακτήρας αντιπροσωπεύει την πολυπεπτιδική αλυσίδα στην οποία ανήκει, ενώ οι αριθμοί μετά τον χαρακτήρα "_" αναφέρονται στα συγκεκριμένα αμινοξέα της πρωτεΐνης. Στη δεύτερη γραμμή βρίσκεται η πρωτοταγής δομή της συγκεκριμένης πρωτεΐνης και στην τρίτη η δευτεροταγής της δομή. Στα αρχεία αυτά υπάρχουν πρωτεΐνες με ελλιπή πληροφορία. Στις γραμμές της πρωτοταγής και δευτεροταγής δομής, κάποια αμινοξέα μπορεί να μην είναι γνωστά με αποτέλεσμα σε αυτή τη θέση να υπάρχει ο χαρακτήρας "!". Όπου υπήρχε ο χαρακτήρας αυτός, αποβάλλαμε την πληροφορία αυτής της θέσης.

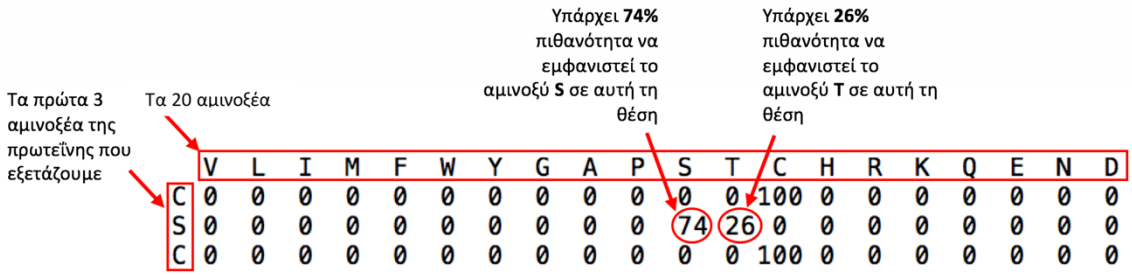


Σχήμα 4.1: Αναπαράσταση μιας πρωτεΐνης (1bdsA_1-43) που βρίσκεται στα αρχεία των δεδομένων εισόδου

4.1.2 Αρχεία πολλαπλής στοίχισης (MSA profiles)

Για κάθε πρωτεΐνη που βρίσκεται στο σύνολο δεδομένων CB513 υπάρχει το αντίστοιχο της MSA profile (Rost and Sander, 1993). Τα MSA profiles περιέχουν εξελικτική πληροφορία για κάθε πρωτεΐνη και με βάση την πληροφορία αυτή, οι ομόλογες πρωτεΐνες, δηλαδή αυτές που πιθανότατα να έχουν την ίδια δομή στο χώρο, κατηγοριοποιούνται στην ίδια κατηγορία. Συγκεκριμένα, εδώ έχουμε την ευθυγράμμιση μιας πρωτεΐνης με άλλες που έχουν αρκετά κοινά χαρακτηριστικά με αυτή. Οι κατηγορίες που δημιουργούνται αντιστοιχούν σε διάφορες πρωτεϊνικές οικογένειες με αποτέλεσμα την ποικιλομορφία των δεδομένων εισόδου.

Στο Σχήμα 4.2 μπορούμε να παρατηρήσουμε πως κωδικοποιείται μια πρωτεΐνη από το CB513. Κάθε στήλη αντιστοιχεί σε ένα από τα 20 αμινοξέα και την πιθανότητα εμφάνισης του στη συγκεκριμένη θέση, ενώ ο αριθμός των γραμμών ισούται στον αριθμό των αμινοξέων που αναφέρονται σε αυτή την πρωτεΐνη (Σχήμα 4.1, 1η γραμμή). Δηλαδή, η κύρια πληροφορία που μας ενδιαφέρει και που μας προσφέρουν τα MSA profiles, είναι η πιθανότητα εμφάνισης ενός αμινοξέως σε κάθε θέση. Τα αρχεία πολλαπλής στοίχισης δημιουργήθηκαν χρησιμοποιώντας το πρόγραμμα PSI-BLAST (Position-Specific Iterated BLAST) με αλληλουχίες που βρίσκονται στο CB513 ως είσοδοι στη βάση NCBI (National Center for Biotechnology Information).



Σχήμα 4.2: Κομμάτι του MSA profile μιας πρωτεΐνης. Η συγκεκριμένη πρωτεΐνη είναι η 1ednA_1-21. Οι αριθμοί αντιπροσωπεύουν την πιθανότητα εμφάνισης του αμινοξέως που εξετάζουμε στην αντίστοιχη θέση.

Μετά από τον έλεγχο των MSA profiles που δημιουργήθηκαν, ανακαλύψαμε ότι 8 από αυτά δημιουργήθηκαν λανθασμένα και έτσι τα αφαιρέσαμε, αφαιρώντας παράλληλα και τις αντίστοιχες πρωτεΐνες από το σύνολο δεδομένων CB513. Οι πρωτεΐνες αυτές ήταν η

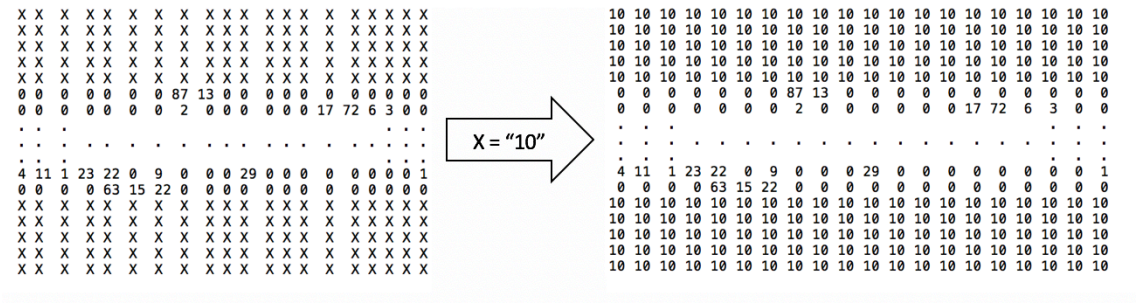
1coiA_1-29, η 1mctI_1-28, η 1tiiC_195-230, η 2erlA_1-40, η 1ceoA_202-254, η 1mrtA_31-61, η 1wfbB_1- 37 και η 6rlxC_-2-20.

Για την ορθή επαλήθευση του μοντέλου-δικτύου, που θα χρησιμοποιηθεί σε αυτή τη διπλωματική εργασία, χρησιμοποιήσαμε 10-fold διασταυρωμένη επικύρωση (cross-validation). Το σύνολο δεδομένων CB513 χωρίστηκε σε 10 τμήματα όπου το 1 χρησιμοποιείται για την επαλήθευση του μοντέλου, ενώ τα υπόλοιπα 9 χρησιμοποιούνται για την εκπαίδευση του. Χρησιμοποιώντας αυτή την τεχνική το δίκτυο μπορεί να ελέγχεται κάθε φορά σε διαφορετικά δεδομένα επαλήθευσης για μια πιο αντικειμενική εκτίμηση του ποσοστού πρόβλεψης.

4.1.3 Δομές δεδομένων εισόδου

Το αρχείο που περιέχει τα δεδομένα εκπαίδευσης από το σύνολο δεδομένων CB513, χρησιμοποιείται για την αντιστοίχιση των πρωτεϊνών με τα MSA profiles τους. Επίσης, χρησιμοποιείται για τη δημιουργία μιας δομής δεδομένων, η οποία περιέχει όλες τις πληροφορίες για κάθε πρωτεΐνη που υπάρχουν στο αρχείο αυτό και μετά από τα στοιχεία κάθε πρωτεΐνης προστίθεται ο χαρακτήρας "+". Αυτό γίνεται για να προστεθεί εδώ η δευτεροταγής δομή της συγκεκριμένης πρωτεΐνης όταν γίνει η πρόβλεψη της.

Η χρήση ενός κινούμενου παραθύρου Wa ήταν απαραίτητη για να μπορεί το δίκτυο να έχει πρόσβαση σε όλα τα δεδομένα, τόσο εκπαίδευσης όσο και επαλήθευσης. Για τη δημιουργία της δομής δεδομένων, η οποία περιέχει τις πληροφορίες των MSA profiles όλων των πρωτεϊνών, αναγκαία ήταν η πρόσθεση κάποιων "άχρηστων" τιμών (dummy values) στην αρχή και στο τέλος των στοιχείων κάθε πρωτεΐνης. Μετά την εισαγωγή των πληροφοριών που υπάρχουν σε ένα MSA profile, δηλαδή των στοιχείων για τα αμινοξέα μιας πρωτεΐνης, ένας πίνακας με μέγεθος (Wa-1) / 2 προστίθεται πριν και μετά από αυτές. Κάθε γραμμή του πίνακα περιέχει "άχρηστες" τιμές και στη συγκεκριμένη υλοποίηση οι τιμές αυτές είναι ο αριθμός "10" (Σχήμα 4.3). Χρησιμοποιώντας την τεχνική του κινούμενου παραθύρου με αυτό τον τρόπο, μας παρέχεται η δυνατότητα να προβλέπουμε κάθε πρωτεΐνη από το σημείο που αρχίζει και μας επιτρέπει να προβλέπουμε το μεσαίο αμινοξύ του κάθε κινούμενου παραθύρου.



Σχήμα 4.3: Πρόσθεση "άχρηστων" τιμών πριν και μετά τα δεδομένα που υπάρχουν στο MSA profile μιας πρωτεΐνης. Η τιμή που επιλέχθηκε σε αυτή την υλοποίηση ήταν ο αριθμός "10".

Η πρωτεΐνη που παρουσιάζεται στο σχήμα είναι η 1a45A_1-174;
$\mathbf{W_a}$ =11;
X: "άχρηστη" τιμή (dummy value)

Όπως ήδη αναφέρθηκε, ένα αμινοξικό κατάλοιπο επηρεάζεται άμεσα από τα γειτονικά του αμινοξέα, τα οποία καθορίζουν την κατηγορία της δευτεροταγούς του δομής, και αυτή η πληροφορία μπορεί να αξιοποιηθεί χρησιμοποιώντας την τεχνική του κινούμενου παραθύρου.

Στη συνέχεια, μετά την επεξεργασία της πιο πάνω δομής δεδομένων, κατασκευάζεται η τελική δομή δεδομένων, η οποία απαρτίζεται από όλα τα κινούμενα παράθυρα όλων των πρωτεϊνών που θα χρησιμοποιηθούν για την εκπαίδευση του δικτύου. Παράλληλα, κατασκευάζεται και η τελική μορφή της δομής δεδομένων των επιθυμητών εξόδων των δεδομένων εκπαίδευσης, όπου κάθε ένα στοιχείο της αντιστοιχεί σε ένα κινούμενο παράθυρο. Η πιο πάνω μέθοδος χρησιμοποιείται και για τη δημιουργία της τελικής μορφής της δομής των δεδομένων επαλήθευσης.

4.2 Προσαρμογή υλοποίησης CW-RNN στο πρόβλημα PSSP

4.2.1 Υλοποίηση και αλλαγές

Στα υποκεφάλαια 1.1 και 3.3.5 εξηγήσαμε τη σημασία και το λόγο που χρησιμοποιούμε το CW-RNN για το συγκεκριμένο πρόβλημα. Επίσης, περιγράψαμε και εξηγήσαμε την γενική του αρχιτεκτονική όπως την παρουσίασε ο Koutník (2014). Όπως μπορείτε να κατανοήσετε, είναι ένα πολύ πρόσφατο μοντέλο το οποίο εφαρμόστηκε και ελέγχθηκε

σε πολύ μικρό αριθμό προβλημάτων-εφαρμογών, όπως παραγωγή και κατηγοριοποίηση ακολουθιών με μεγάλης εμβέλειας εξαρτήσεις. Σε αυτού του είδους εφαρμογές έδωσε πολύ καλά αποτελέσματα και ποσοστά επιτυχίας.

Η υλοποίηση που χρησιμοποιήθηκε (υλοποιήθηκε από τον Tom Runia το 2017, https://github.com/tomrunia/ClockworkRNN, last accessed 15 July 2017) ήταν όπως την παρουσίασε ο Koutník με μια μικρή διαφορά που δεν επηρέασε την αποδοτικότητα του δικτύου. Η διαφορά ήταν το ότι ο αριθμός των κρυφών νευρώνων είναι απαραίτητο να διαιρείται ακριβώς με τον αριθμό των τμημάτων που απαρτίζεται το κρυφό επίπεδο. Στο άρθρο του ο Koutník είχε αναφέρει ότι αν δεν διαιρείται ακριβώς ο αριθμός των νευρώνων με τα τμήματα, οι εναπομείναντες νευρώνες διανέμονται στα πιο γρήγορα τμήματα.

Έγιναν αρκετές γενικές αλλαγές για να μπορεί η συγκεκριμένη υλοποίηση να προσαρμοστεί στο PSSP πρόβλημα. Στο προηγούμενο υποκεφάλαιο αναφερθήκαμε στη μορφή που πρέπει να έχουν τα δεδομένα εισόδου για να μπορεί να τα δεχτεί το δίκτυο και να μπορεί να εκπαιδευτεί ορθά από αυτά με την εκμετάλλευση όσης περισσότερης πληροφορίας. Το μέγεθος του επιπέδου εισόδου ισούται με Wa * 20. Κάθε είσοδος – μια γραμμή του κινούμενου παραθύρου Wa – αντιπροσωπεύει ένα αμινοξικό κατάλοιπο και οι 20 αριθμοί αντιστοιχούν στις συχνότητες διαφορετικών τύπων αμινοξέων στοιχισμένων με την αρχική ακολουθία. Το δίκτυο αποτελείται από τρείς αριθμητικές εξόδους, οι οποίες αντιστοιχούν στις τρεις ευρείς κλάσεις της δευτεροταγούς δομής, E, H και C. Η απόφαση για την κλάση του προβλεπόμενου αμινοξικού καταλοίπου αποφασίζεται χρησιμοποιώντας την απλή μέθοδο «ο νικητής τα παίρνει όλα» (Winner-Takes-All (WTA)).

Επιπρόσθετα, το μοντέλο-δίκτυο εκπαιδεύτηκε χρησιμοποιώντας ένα αριθμό από σταθερού μεγέθους σύνολα δεδομένων (mini-batches). Το μη δυναμικό μέγεθος των συνόλων αυτών, μας περιόρισε στην επιλογή του αριθμού των δεδομένων εκπαίδευσης και επαλήθευσης. Έτσι, το δίκτυο δε μπορούσε να εκμεταλλευτεί όλα τα δεδομένα εκπαίδευσης με αποτέλεσμα να μην εκπαιδεύεται στο μέγιστο που θα μπορούσε, αλλά επίσης σχημάτιζε λάθος συσχετίσεις μεταξύ των δεδομένων, αφού η τελευταία πρωτεΐνη δεν ήταν ολοκληρωμένη. Αυτό ήταν ένα κρίσιμο πρόβλημα, αφού το δίκτυο θα μπορούσε να δώσει καλύτερα ποσοστά επιτυχίας με τη χρήση ολόκληρου του συνόλου δεδομένων

εκπαίδευσης. Για την αντιμετώπιση αυτού του περιορισμού αποφασίσαμε να προσθέσουμε μηδενικά (“0”) στο τέλος των δεδομένων επαλήθευσης για να επιτρέπεται στο δίκτυο να χρησιμοποιήσει όλα τα δεδομένα εκπαίδευσης. Για να χρησιμοποιηθεί ολόκληρο το σύνολο των δεδομένων εκπαίδευσης, με βάση τη συγκεκριμένη υλοποίηση, πρέπει το μέγεθος του συνόλου αυτού να μπορεί να διαιρεθεί ακριβώς με το μέγεθος του συνόλου δεδομένων επαλήθευσης.

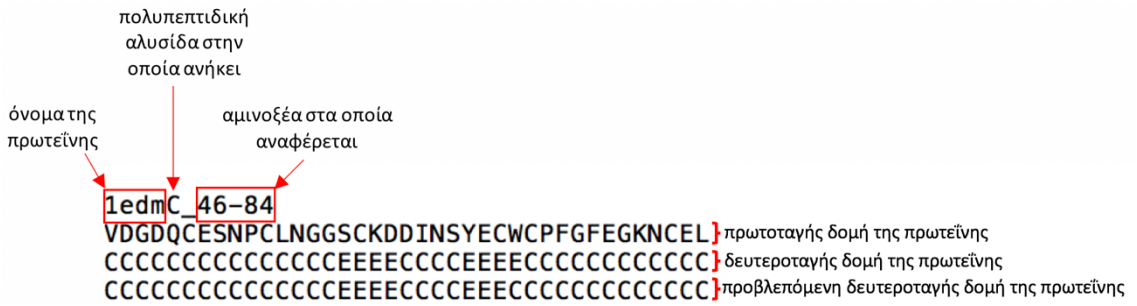
Μετά από κάθε batch εκπαίδευσης, υπολογίζεται το σφάλμα εκπαίδευσης (training loss) και από τις προβλέψεις εκπαίδευσης, υπολογίζεται επίσης για το συγκεκριμένο batch ο αριθμός των ορθών προβλέψεων με τη μέθοδο WTA. Ο αριθμός αυτός προστίθεται κάθε φορά για τον υπολογισμό του συνολικού αριθμού ορθών προβλέψεων για κάθε εποχή. Ως αποτέλεσμα, στο τέλος της εποχής, δηλαδή όταν εισαχθούν στο δίκτυο όλα τα batches εκπαίδευσης, η τιμή με το σύνολο των ορθών προβλέψεων είναι ήδη υπολογισμένη. Διαιρώντας την ως προς τον συνολικό αριθμό δεδομένων εκπαίδευσης υπολογίζουμε το ποσοστό επιτυχίας για τα δεδομένα εκπαίδευσης (training accuracy). Στη συνέχεια, το δίκτυο ελέγχεται με τα δεδομένα επαλήθευσης και υπολογίζεται παρομοίως το σφάλμα επαλήθευσης και το ποσοστό επιτυχίας τους (validation accuracy).

Ακόμη ένα πρόβλημα που έπρεπε να αντιμετωπιστεί σε αυτό το σημείο, ήταν όταν άρχιζε μια νέα εποχή. Συγκεκριμένα, επειδή τα δεδομένα εκπαίδευσης χωρίζονται σε batches σταθερού μεγέθους, υπάρχει μεγάλη πιθανότητα κάποιες πληροφορίες σε κάποιο σημείο μιας πρωτεΐνης να μοιραστούν, είτε στην αρχή, είτε στο τέλος του batch. Με αυτό τον τρόπο το δίκτυο θα μάθαινε συγκεκριμένα τεχνητά πρότυπα σχετικά με τα δεδομένα και δε θα μπορούσε εύκολα να ανακαλύψει τις πραγματικές συσχετίσεις μεταξύ τους. Για την αντιμετώπιση αυτού του προβλήματος χρειάστηκε να γίνει ένα ανακάτεμα των δεδομένων εκπαίδευσης πριν την επόμενη εποχή. Δηλαδή, οι πρωτεΐνες εκπαίδευσης ανακατεύονται μεταξύ τους έτσι ώστε σε κάθε εποχή να κόβεται διαφορετική πρωτεΐνη σε κάθε batch, για να ελαχιστοποιηθεί το ποσοστό που επηρεάζει αυτός ο παράγοντας την εκπαίδευση του δικτύου.

Κατά την τελευταία εποχή, δημιουργούνται δύο δομές δεδομένων, οι οποίες περιέχουν τις τελικές προβλέψεις του δικτύου, τόσο των δεδομένων εκπαίδευσης όσο και των δεδομένων επαλήθευσης. Αυτές οι δομές χρησιμοποιούνται για την αντικατάσταση του

χαρακτήρα “+” στις δομές που περιγράψαμε στο υποκεφάλαιο 4.1.3, με την προβλεπόμενη δευτεροταγή δομή κάθε πρωτεΐνης.

Στο τέλος όλων των εποχών, δημιουργούνται δύο αρχεία εξόδου, χρησιμοποιώντας τις δομές που μόλις αναφέραμε. Αυτά τα αρχεία, έχουν την ίδια δομή όπως τα αρχεία εισόδου που αναφέραμε στο υποκεφάλαιο 4.1.1 και Σχήμα 4.1, εκτός από το γεγονός ότι μετά τις πληροφορίες που υπάρχουν για κάθε πρωτεΐνη, προστίθεται και η δευτεροταγής δομή της συγκεκριμένης πρωτεΐνης στην αμέσως επόμενη γραμμή. Ως αποτέλεσμα, κάθε πρωτεΐνη απαιτεί τέσσερις γραμμές για την εγγραφή των πληροφοριών της. Οι πληροφορίες αυτές είναι το όνομα της πρωτεΐνης, η πρωτοταγής της δομή, η δευτεροταγής της δομή και η προβλεπόμενη δευτεροταγής της δομή (Σχήμα 4.4). Τα αρχεία αυτά έχουν τη συγκεκριμένη μορφή, επειδή απαιτείται στη συνέχεια για τα ensembles και το φιλτράρισμα (filtering), τα οποία θα εξηγηθούν στο Κεφάλαιο 5.



Σχήμα 4.4: Αναπαράσταση μιας πρωτεΐνης (1edmC_46-84) που βρίσκεται στα αρχεία των δεδομένων εξόδου.

4.2.2 Αρχιτεκτονική

Η καλύτερη αρχιτεκτονική που προσδιορίστηκε ήταν μέσω της τεχνικής “δοκιμής και λάθους”. Πιο συγκεκριμένα, το σύνολο των clock periods που χρησιμοποιείται είναι συμμετρικό και ένα κομμάτι του εμπνεύστηκε από τη σειρά Fibonacci, όπως πρότεινε και ο Koutník στο άρθρο του (Koutník et al., 2014). Η διαδικασία που ακολουθήθηκε για την επιλογή του συνόλου των clock periods περιγράφεται στο υποκεφάλαιο 5.2.1. Το μέγεθος του κινούμενου παραθύρου W_a ισούται με 11 και το κρυφό επίπεδο του δικτύου απαρτίζεται από 28 τμήματα και σε κάθε ένα από αυτά ανατίθεται 1 clock period. Το σύνολο των clock periods που χρησιμοποιείται είναι {89, 55, 34, 21, 13, 8, 5, 3, 3, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 3, 3, 5, 8, 13, 21, 34, 55, 89}. Τα πιο γρήγορα τμήματα είναι αυτά με

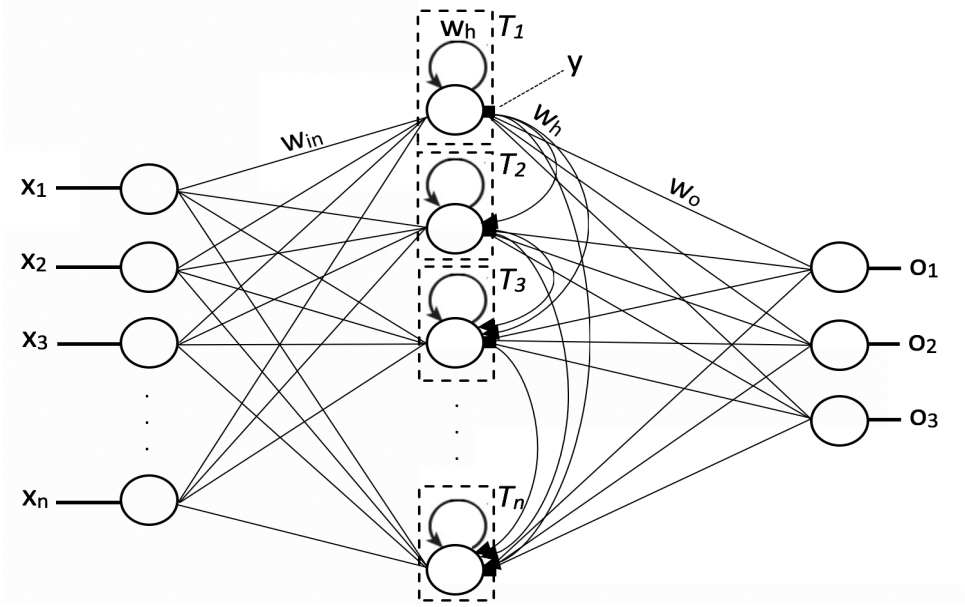
περίοδο 1, δηλαδή ενεργοποιούνται σε κάθε χρονικό βήμα ενώ τα πιο αργά τμήματα είναι αυτά με περίοδο 89. Τα βάρη του δικτύου αρχικοποιούνται με ιδιαίτερα μικρές τιμές με τυπική απόκλιση (standard deviation) 0.01 και τα biases με την τιμή 0. Η συνάρτηση ενεργοποίησης για το κρυφό επίπεδο ήταν η υπερβολική εφαπτομένη και για το επίπεδο εξόδου η σιγμοειδής. Για την εκπαίδευση του δικτύου και για τον υπολογισμό των κλίσεων (gradients) χρησιμοποιείται ο Adam optimizer. Ο συγκεκριμένος optimizer επιλέχθηκε ανάμεσα σε άλλους (υποκεφάλαιο 5.2.2), επειδή είναι υπολογιστικά αποδοτικός και είναι κατάλληλος για το συγκεκριμένο πρόβλημα με μεγάλο σύνολο δεδομένων (Kingma and Lei Ba, 2017). Ο Adam χρησιμοποιείται με τις προκαθορισμένες τιμές του για το beta1 (0.9) και το beta2 (0.99) και με ρυθμό μάθησης ίσο με 0.01. Επίσης, χρησιμοποιείται το gradient clipping, το οποίο είναι το "clipping" των κλίσεων μεταξύ δύο αριθμών για να τις αποτρέψουν από το να γίνουν πολύ μεγάλες. Αυτό έχει ως αποτέλεσμα την μείωση του προβλήματος της έκρηξης κλίσεων (gradient exploding problem).

4.2.3 Εκπαίδευση του μοντέλου

Στο υποκεφάλαιο 4.1.2 αναφέραμε το cross-validation, το οποίο έγινε με 10 διαφορετικά folds. Για να μπορούμε να χρησιμοποιήσουμε όλα τα δεδομένα εκπαίδευσης από κάθε fold, σε κάποια από αυτά, το δίκτυο εκπαιδευόταν με mini-batches, ενώ σε άλλα με ολόκληρο το batch, δηλαδή ολόκληρο το σύνολο των δεδομένων εκπαίδευσης. Δηλαδή, στην πρώτη περίπτωση, οι νέες τιμές των βαρών του δικτύου υπολογίζονταν στο τέλος κάθε mini-batch, ενώ στη δεύτερη περίπτωση, μετά το πέρασμα όλων των δεδομένων εκπαίδευσης.

Αρχικά, αρχικοποιούνται τα βάρη του δικτύου με μικρές τυχαίες τιμές με τυπική απόκλιση 0.01 και τα biases με τιμή 0. Μετέπειτα, ξεκινάει η φάση του εμπρόσθιου περάσματος. Στο πρώτο βήμα αποφασίζεται ποια τμήματα θα ενεργοποιηθούν βάση της παρούσας χρονικής στιγμής. Στη συνέχεια, υπολογίζεται η τιμή εξόδου του κάθε νευρώνα που βρίσκεται στα ενεργοποιημένα τμήματα. Για τον υπολογισμό αυτό λαμβάνονται υπόψη οι τιμές εισόδου (Σχήμα 4.5; x_1, x_2, …), τα βάρη που βρίσκονται μεταξύ επιπέδου εισόδου και κρυφού επιπέδου (w_{in}), τα βάρη των recurrent συνδέσεων (w_h) αλλά και τα αποτελέσματα από τα πιο αργά τμήματα. Η τιμή του κάθε νευρώνα που υπολογίζεται με τις τιμές που μόλις αναφέραμε, εισέρχεται στην συνάρτηση

ενεργοποίησης, η οποία στην περίπτωση μας είναι η υπερβολική εφαπτομένη, για να εξαχθούν οι τελικές έξοδοι (y). Ακολουθεί ο υπολογισμός των πραγματικών εξόδων του δικτύου. Οι τιμές αυτές υπολογίζονται με βάση τις τιμές εξόδου των κρυφών νευρώνων (y) και των βαρών που βρίσκονται μεταξύ του κρυφού επιπέδου και του επιπέδου εξόδου (w_o). Όπως και στους νευρώνες του κρυφού επιπέδου, και εδώ οι τιμές που μόλις υπολογίσθηκαν εισέρχονται σε μια συνάρτηση ενεργοποίησης, η οποία εδώ είναι η σιγμοειδής, για την εξαγωγή των τελικών πραγματικών εξόδων του δικτύου (o_1, o_2, o_3). Σειρά έχει ο υπολογισμός του σφάλματος. Μετά τον υπολογισμό του τετραγωνικού σφάλματος για τους νευρώνες εξόδου για κάθε δεδομένο εισόδου που υπολογίζεται βάση των πραγματικών και επιθυμητών εξόδων, υπολογίζεται το σφάλμα για ολόκληρο το mini-batch. Με τον ίδιο τρόπο γίνεται αν έχουμε ολόκληρο το batch. Στο πέρασμα προς τα πίσω, ο Adam optimizer, με βάση το σφάλμα που μόλις υπολογίστηκε, υπολογίζει τις κλίσεις για κάθε μεταβλητή.



Σχήμα 4.5: Clockwork-RNN

Το τμήμα με T_1 είναι το πιο γρήγορο ενώ το T_n το πιο αργό ($T_1 < T_2 < T_3 < ... < T_n$) και γι' αυτό υπάρχουν αυτές οι συνδέσεις (από τα πιο γρήγορα τμήματα στα πιο αργά)

x_1, …, x_n: είσοδοι του δικτύου;
w_{in}: βάρη μεταξύ επιπέδου εισόδου και κρυφού επιπέδου;
w_h: βάρη που βρίσκονται στις recurrent συνδέσεις;
y: τελική έξοδος των νευρώνων στο κρυφό επίπεδο;
w_o: βάρη μεταξύ κρυφού επιπέδου και επιπέδου εξόδου;
o_1, o_2, o_3: πραγματικές έξοδοι του δικτύου

Όπως έχουμε ήδη αναφέρει στο υποκεφάλαιο 4.2.2, πραγματοποιείται και “clipping” στις κλίσεις πριν εφαρμοστούν στις μεταβλητές. Τέλος, γίνονται οι κατάλληλες αλλαγές στα βάρη του δικτύου που τηρούν την προϋπόθεση ότι οι νευρώνες στους οποίους αντιστοιχούν βρίσκονται σε ενεργοποιημένα τμήματα. Μετά το τέλος της εκπαίδευσης σε κάθε εποχή, ακολουθεί η αξιολόγηση του δικτύου με τα δεδομένα επαλήθευσης, όπου υπολογίζεται το σφάλμα επαλήθευσης.

Κεφάλαιο 5

Πειράματα και αποτελέσματα

5.1 Σκοπός των πειραμάτων

5.1.1 Πληροφορίες πειραμάτων

5.1.2 Σημασία πρόβλεψης μεσαίου αμινοξέως

5.2 Πειράματα για βελτιστοποίηση των παραμέτρων του δικτύου

5.2.1 Clock periods

5.2.2 Optimizers

5.2.3 Συναρτήσεις ενεργοποίησης

5.2.4 Αριθμός νευρώνων κρυφού επιπέδου

5.3 10-fold cross-validation

5.3.1 Ensembles

5.3.2 Filtering: Εξωτερικοί κανόνες

5.3.3 Filtering: Support Vector Machines

5.4 Χρόνος εκτέλεσης

5.1 Σκοπός των πειραμάτων

5.1.1 Πληροφορίες πειραμάτων

Τα πειράματα τα οποία διεξάχθηκαν κατά τη διάρκεια της παρούσας διπλωματικής εργασίας, είχαν ως αρχικό στόχο την βελτιστοποίηση των υπερπαραμέτρων του δικτύου για το συγκεκριμένο πρόβλημα, PSSP. Όπως έχουμε ήδη αναφέρει, το σύνολο δεδομένων που χρησιμοποιήθηκε ήταν το CB513, από το οποίο αφαιρέσαμε 8 πρωτεΐνες, οι οποίες είχαν λανθασμένα MSA profiles. Για την ανακάλυψη των βέλτιστων παραμέτρων και πριν προχωρήσουμε στο 10-fold cross-validation, βασιστήκαμε μόνο σε ένα από τα δέκα folds (το fold με ονομασία "fold0"), το οποίο περιέχει 456 πρωτεΐνες εκπαίδευσης και 49 πρωτεΐνες επαλήθευσης. Η επιλογή των υπερπαραμέτρων εξαρτάται από διάφορους παράγοντες. Μερικοί από αυτούς είναι η αρχιτεκτονική του δικτύου, τα δεδομένα εισόδου και το πρόβλημα που προσπαθούμε να λύσουμε. Τα υπόλοιπα 9 folds, περιέχουν διαφορετικό αριθμό πρωτεϊνών εκπαίδευσης και επαλήθευσης από αυτό που χρησιμοποιήσαμε για την βελτιστοποίηση των υπερπαραμέτρων ("fold0"), λόγω των 8 πρωτεϊνών που αφαιρέσαμε. Μπορούμε να παρατηρήσουμε τους ακριβείς αριθμούς των folds αυτών στον Πίνακα 2.

Πίνακας 2: Ακριβής αριθμός πρωτεϊνών εκπαίδευσης και επαλήθευσης για κάθε fold και πόσα batches χρησιμοποιήθηκαν στην εκπαίδευση του δικτύου

Αρ. fold	**Αρ. batches**	**Αρ. πρωτεϊνών εκπαίδευσης**	**Αρ. πρωτεϊνών επαλήθευσης**
0	9	456	49
1	11	457	48
2	1	456	49
3	5	455	50
4	1	456	49
5	9	458	47
6	2	459	46
7	1	457	48
8	1	456	49
9	6	457	48

Ακόμη ένας λόγος για την διεξαγωγή των πειραμάτων ήταν για επαλήθευση της θεωρίας του Koutník (2014), τόσο για την ταχύτητα του δικτύου, όσο και για την ικανότητα του να μπορεί να ανακαλύψει μακρινές εξαρτήσεις μεταξύ των δεδομένων. Για τους λόγους αυτούς, διεξήγαμε διάφορα πειράματα για να ελέγξουμε πώς η κάθε υπερπαράμετρος του CW-RNN επηρεάζει τα αποτελέσματα και για να αποφασίσουμε το βέλτιστο σύνολο των παραμέτρων για να πετύχουμε το καλύτερο δυνατό ποσοστό για το πρόβλημα αυτό.

Τα ποσοστά εκπαίδευσης και επαλήθευσης υπολογίστηκαν χρησιμοποιώντας την ανά αμινοξικό κατάλοιπο ακρίβεια (per-residue accuracy) Q3, και το Segment OVerlap (SOV) score (Zemla et al., 1999).

Μετά την βελτιστοποίηση των παραμέτρων του δικτύου, κάθε fold του συνόλου δεδομένων, εκτελέστηκε 8 φορές. Αυτό έγινε με στόχο την εφαρμογή ensembles σε κάθε fold ξεχωριστά για ένα καλύτερο ποσοστό επιτυχίας. Τα ensembles είναι ο συνδυασμός των διάφορων αποτελεσμάτων που εξάχθηκαν από τις διαφορετικές εκτελέσεις του ίδιου fold και το καινούργιο αποτέλεσμα είναι ο μέσος όρος τους. Όταν εφαρμόστηκαν τα ensembles υπολογίσθηκαν ξανά τα καινούργια ποσοστά ακριβείας Q3 και SOV και για κάθε fold εξάγεται και το confusion matrix του. Αυτό μας περιγράφει ξεχωριστά την ακρίβεια κάθε κατηγορίας (H, E, C) για βαθύτερη γνώση της ποιότητας του classifier μας. Τέλος, στις εξόδους των ensembles, εφαρμόστηκε φιλτράρισμα. Δοκιμάστηκαν συγκεκριμένα δύο είδη φιλτραρίσματος, εξωτερικοί εμπειρικοί κανόνες (empirical rules) και Support Vector Machines (SVMs). Οι εμπειρικοί κανόνες προσπαθούν να διορθώσουν και να κάνουν πιο ομαλές (“smooth”) τις προβλέψεις του μοντέλου, μεταβάλλοντας κομμάτια από την προβλεπόμενη δευτεροταγή δομή, τα οποία είναι φυσικοχημικά απίθανα. Η εμπειρική τεχνική που χρησιμοποιήθηκε ήταν η SS-filt (Salamov and Solovyev, 1995), η οποία αποτελείται από τους κανόνες φιλτραρίσματος οι οποίοι αντικαθιστούν τα single-helical (H) και τα single-strand (E) αμινοξικά κατάλοιπα με coil (C) και όλα τα strands (E), τα οποία έχουν μήκος δύο και περιβάλλονται από helices (H), αντικαθίστανται με helices (H) (Kountouris et al., 2012). Τα SVMs (Cortes and Vapnik, 1995), σε ένα γραμμικά διαχωρίσιμο πρόβλημα, προσπαθούν να εντοπίσουν τα οριακά κοντινότερα σημεία (support vectors) δύο διαφορετικών κλάσεων και στη συνέχεια τοποθετούν την επιφάνεια διαχωρισμού (hyperplane) στη μέση των σημείων αυτών. Στην περίπτωση που το πρόβλημα δεν είναι

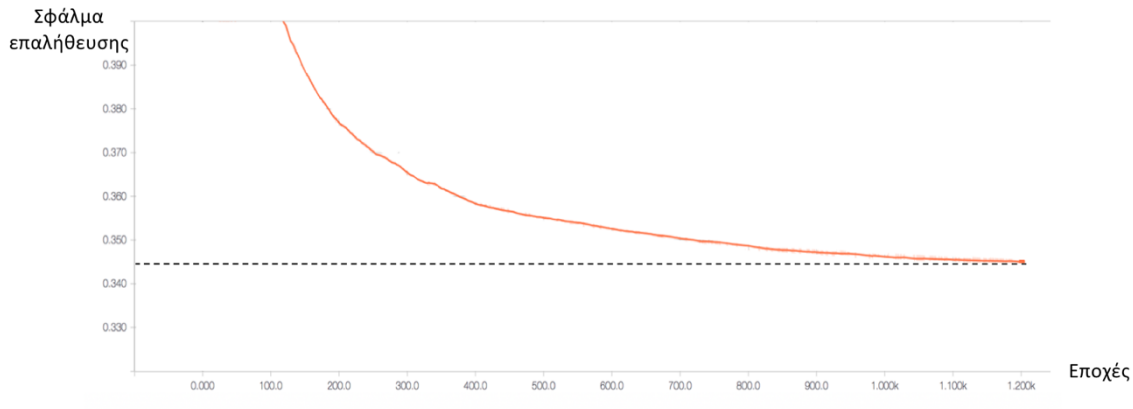
γραμμικά διαχωρίσιμο, τα SVMs προσπαθούν να ανάγουν το πρόβλημα σε υψηλότερη διάσταση έτσι ώστε να γίνει πλέον γραμμικά διαχωρίσιμο, χρησιμοποιώντας κάποιες γνωστές τεχνικές kernel.

5.1.2 Σημασία πρόβλεψης του μεσαίου αμινοξέος

Σε αυτό το σημείο καλό θα ήταν να τονίσουμε τη σημασία πρόβλεψης του μεσαίου αμινοξέως του κινητού παραθύρου. Πριν να γίνουν τα πειράματα για την πρόβλεψη του μεσαίου αμινοξέως, είχαμε επικεντρωθεί στην πρόβλεψη του τελευταίου αμινοξέως του κινητού παραθύρου. Δηλαδή, τα dummy values που αναφέραμε στο υποκεφάλαιο 4.1.3, εισάγονταν όλα στην αρχή των στοιχείων κάθε πρωτεΐνης κατά τη δημιουργία της δομής δεδομένων. Με αυτό τον τρόπο όμως, το δίκτυο έπαιρνε μόνο μερική πληροφορία για το αμινοξύ που προσπαθούσαμε να προβλέψουμε, επειδή χρησιμοποιούσε πληροφορία μόνο από τα αμινοξέα που προηγούνταν. Γνωρίζουμε όμως, ότι ένα αμινοξύ αλληλεπιδρά τόσο με τα αμινοξέα που βρίσκονται πριν από αυτό, όσο και με τα αμινοξέα που βρίσκονται μετά από αυτό, δηλαδή όλα τα γειτονικά του αμινοξέα. Ως αποτέλεσμα, αρχίζοντας να προβλέπουμε το μεσαίο αμινοξύ, αντί το τελευταίο αμινοξύ του κινητού παραθύρου, τα αποτελέσματα του δικτύου βελτιώθηκαν κατά ένα σεβαστό ποσοστό. Σε αυτή την διπλωματική εργασία δε θα επικεντρωθούμε στην σειρά πειραμάτων που διεξάχθηκαν όταν προβλεπόταν το τελευταίο αμινοξύ. Το ποσοστό επιτυχίας Q3 με αυτή την τεχνική ήταν $\approx 65\%$.

5.2 Πειράματα για βελτιστοποίηση των παραμέτρων του δικτύου

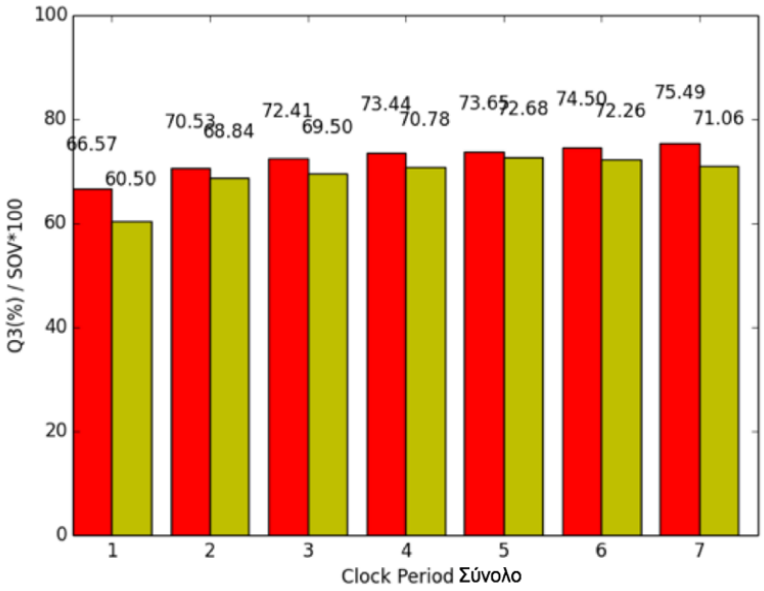
Έχουμε αναφέρει στο υποκεφάλαιο 5.1.1 πως όλα τα πειράματα για τη βελτιστοποίηση των παραμέτρων έχουν διεξαχθεί με τη χρήση του fold με όνομα “fold0”. Η πιο κάτω γραφική παράσταση δημιουργήθηκε για να μας δείξει μέχρι που μειώνεται το σφάλμα επαλήθευσης στο συγκεκριμένο fold και να μας επιβεβαιώσει ότι το δίκτυο μαθαίνει, αφού σε κάθε εποχή μπορεί να προβλέψει καλύτερα άγνωστες προς αυτό πρωτεϊνικές ακολουθίες. Συγκεκριμένα, το δίκτυο εκπαιδεύτηκε για 1200 εποχές, όπου στο τέλος κάθε εποχής υπολογιζόταν το σφάλμα επαλήθευσης, το οποίο παρατηρούμε στο Σχήμα 5.1 ότι μειώνεται μέχρι και το 0.3451.



Σχήμα 5.1: Η γραφική του σφάλματος επαλήθευσης.

5.2.1 Clock periods

Τα πιο αργά τμήματα του CW-RNN, λειτουργούν ως ένας μηχανισμός που γεφυρώνει το κενό που υπάρχει μεταξύ των πληροφοριών και των κλίσεων των παραμέτρων, οι οποίες βρίσκονται μεγάλα χρονικά διαστήματα μακριά. Τα πιο γρήγορα τμήματα επεξεργάζονται την πληροφορία που δέχονται και "παίρνουν" βραχυπρόθεσμες αποφάσεις, λόγω του ότι έχουν πιο μικρές περιόδους και εκτελούνται πιο συχνά από τα πιο αργά τμήματα. Η πληροφορία, την οποία επεξεργάζονται τα πιο γρήγορα τμήματα, γίνεται διαθέσιμη για επεξεργασία και στα πιο αργά τμήματα, όταν αυτά ενεργοποιηθούν, προκειμένου να γίνουν οι σωστές συσχετίσεις μεταξύ αυτής της πληροφορίας και της πληροφορίας που ήταν ήδη αποθηκευμένη στις καταστάσεις τους από την τελευταία τους ενεργοποίηση. Ως αποτέλεσμα, τα clock periods, είναι η καινοτομία της συγκεκριμένης αρχιτεκτονικής και θεωρούνται ως το κλειδί για την επίτευξη καλών αποτελεσμάτων και γι' αυτό ξεκινήσαμε να πειραματιζόμαστε και να δοκιμάζουμε διάφορους συνδυασμούς πρώτα σε αυτά.



Σχήμα 5.2: Q3 ποσοστό ακριβείας και SOV score για κάθε σύνολο των clock periods το οποίο παρουσιάζεται στον Πίνακα 3.

Κόκκινη στήλη: Q3 ποσοστό ακριβείας
Πράσινη στήλη: SOV score

Πίνακας 3: Τα σύνολα των clock periods που χρησιμοποιήθηκαν για τον καθορισμό του βέλτιστου για το μοντέλο για το πρόβλημα PSSP.

Αρ.	Clock Period Σύνολα
1	1, 2, 4, 8, 16, 32, 64
2	64, 32, 16, 8, 4, 2, 1
3	1, 2, 4, 8, 16, 8, 4, 2, 1
4	89, 55, 34, 21, 13, 8, 5, 3, 2, 1, 1
5	64, 32, 16, 8, 4, 2, 1, 2, 4, 8, 16, 32, 64
6	256, 128, 64, 32, 16, 8, 4, 2, 2, 2, 1, 1, 2, 2, 4, 8, 16, 32
7	89, 55, 34, 21, 13, 8, 5, 3, 3, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 3, 3, 5, 8, 13, 21, 34, 55, 89

Αρχικά, δοκιμάσαμε τα clock periods που ήταν ήδη καθορισμένα στην CW-RNN υλοποίηση που χρησιμοποιήσαμε (Σύνολο 1 στον Πίνακα 3), τα οποία έδωσαν σχετικά

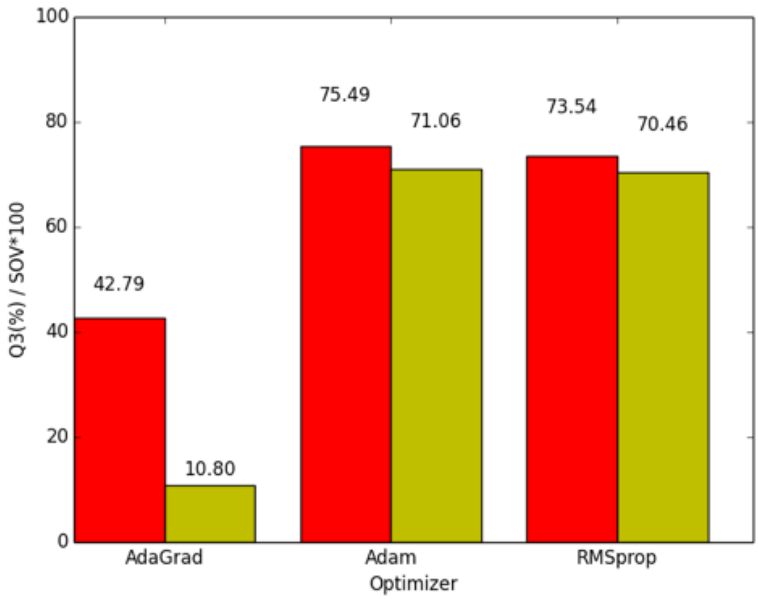
χαμηλό ποσοστό ακριβείας Q3 και SOV score (Σχήμα 5.2). Χρησιμοποιώντας τις ίδιες ακριβώς περιόδους με αντίστροφη σειρά, δηλαδή από την πιο μεγάλη περίοδο στην πιο μικρή (Σύνολο 2 στον Πίνακα 3), το ποσοστό ακριβείας Q3 αυξήθηκε σχεδόν 4% και το SOV αυξήθηκε περισσότερο από 8 πόντους. Έπειτα, συνδυάσαμε κομμάτια από τα πιο πάνω σύνολα για τη δημιουργία του Clock Period Συνόλου 3, το οποίο κατάφερε να δώσει ακόμη καλύτερα αποτελέσματα από τα προηγούμενα δύο σύνολα. Η σκέψη για αυτό τον συνδυασμό των πρώτων δύο συνόλων, πήγαζε από τη συμμετρία του κινητού παραθύρου σε σχέση με το μεσαίο αμινοξύ για το οποίο εκτελείται η πρόβλεψη. Έχοντας αυτό υπόψη μας, χρησιμοποιήσαμε ένα συμμετρικό σύνολο περιόδων. Το 5ο σύνολο χρησιμοποιήθηκε βασισμένο στην ίδια λογική με το 3ο και τα ποσοστά αυξήθηκαν λίγο. Το 6ο σύνολο περιόδων δοκιμάστηκε για να εξετάσουμε την περίπτωση όπου δεν υπήρχε συμμετρία και τοποθετούνταν περισσότερα γρήγορα τμήματα για να επεξεργάζονται πληροφορία σε κάθε χρονική στιγμή. Αυτό είχε ως επακόλουθο την πολύ μικρή αύξηση του ποσοστού Q3, ενώ παράλληλα την λιγοστή μείωση του SOV score.

Έχοντας στο μυαλό μας την υποψία ότι μπορεί να υπάρχει κάποια σύνδεση μεταξύ των περιόδων του δικτύου, σκεφτήκαμε να χρησιμοποιήσουμε μια σειρά από περιόδους με λίγο καλύτερη σύνδεση μεταξύ τους. Το Clock Period Σύνολο 4 εμπνεύστηκε από ένα μικρό μέρος της σειράς Fibonacci, το οποίο οδήγησε σε ένα σχετικά καλό ποσοστό. Τελικά, ενώσαμε όλη τη γνώση που απεκτήσαμε από τα προηγούμενα πειράματα, όπως τη συμμετρία των clock periods, ένα μέρος της σειράς Fibonacci και την προσθήκη περισσότερων γρηγορότερων, δηλαδή πιο μικρών, περιόδων για τη δημιουργία του 7ου συνόλου. Αυτό το σύνολο επίφερε το καλύτερο αποτέλεσμα στο Q3 score και το χρησιμοποιήσαμε για όλα τα επόμενα πειράματα. Παρόλα αυτά, το SOV score μειώθηκε συγκρίνοντας το με τα αποτελέσματα του 5ου και 6ου συνόλου περιόδων.

Καλό είναι να σημειώσουμε εδώ ότι τα διαφορετικά clock period σύνολα που μόλις αναφέρθηκαν αντιστοιχούν σε CW-RNNs με τμήματα διαφορετικού μεγέθους (7, 7, 9, 11, 13, 18, 28 αντίστοιχα). Οι διαφορές που παρουσιάζονται στο Σχήμα 5.2 αφορούν την επίδραση των διαφορετικών συνόλων περιόδων και όχι το μέγεθος του δικτύου, και συγκεκριμένα των τμημάτων. Τα clock period σύνολα 1 και 2 μας δείχνουν ξεκάθαρα ότι παρόλο που έχουν τον ίδιο αριθμό από τμήματα, το αποτέλεσμα εξαρτάται από τη σωστή επιλογή και σειρά των clock periods.

Εκτός από τα πιο πάνω clock period σύνολα, δοκιμάστηκαν και άλλα όπως μπορούμε να παρατηρήσουμε στο Παράρτημα ΣΤ. Αυτά δοκιμάστηκαν για να ελέγξουμε, όταν υπάρχει συμμετρία, αν τα αποτελέσματα του δικτύου βελτιώνονταν με την αλλαγή των clock periods, είτε με πιο μικρές, είτε με πιο μεγάλες περιόδους. Επίσης, δοκιμάστηκαν ενδιάμεσα των clock period συνόλων που παρουσιάσαμε πιο πάνω, και μας βοήθησαν να καταλήξουμε στο βέλτιστο clock period σύνολο. Δηλαδή, αυτά μας βοήθησαν να συμπεράνουμε ότι στο καλύτερο clock period σύνολο για το πρόβλημα μας θα έπρεπε να υπάρχει συμμετρία, να έχουμε περισσότερα γρήγορα τμήματα και να περιέχει ένα κομμάτι από τη σειρά Fibonacci.

5.2.2 Optimizers



Σχήμα 5.3: Q3 ποσοστό ακριβείας και SOV score που πέτυχε ο κάθε optimizer

Κόκκινη στήλη: Q3 ποσοστό ακριβείας
Πράσινη στήλη: SOV score

Στην υλοποίηση που χρησιμοποιήθηκε στα πλαίσια αυτής της διπλωματικής εργασίας, ήταν διαθέσιμοι τρεις διαφορετικοί optimizers, οι οποίοι δοκιμάστηκαν για την αποδοτικότητα και την απόδοση τους για το πρόβλημα PSSP. Οι optimizers αυτοί ήταν ο Adam, ο RMSprop και ο AdaGrad, τους οποίους έχουμε εξηγήσει στο υποκεφάλαιο 3.4.5. Οι υπόλοιπες παράμετροι του δικτύου παράμειναν σταθερές κατά τον έλεγχο των

optimizers. Συγκεκριμένα, τα clock periods τα οποία χρησιμοποιήθηκαν, ήταν τα βέλτιστα, τα οποία αναφέρθηκαν στο υποκεφάλαιο 5.2.1, οι συναρτήσεις ενεργοποίησης για το κρυφό επίπεδο και το επίπεδο εξόδου ήταν οι υπερβολική εφαπτομένη και η σιγμοειδής συνάρτηση αντίστοιχα και χρησιμοποιήθηκαν 28 νευρώνες στο κρυφό επίπεδο, ένας για κάθε τμήμα.

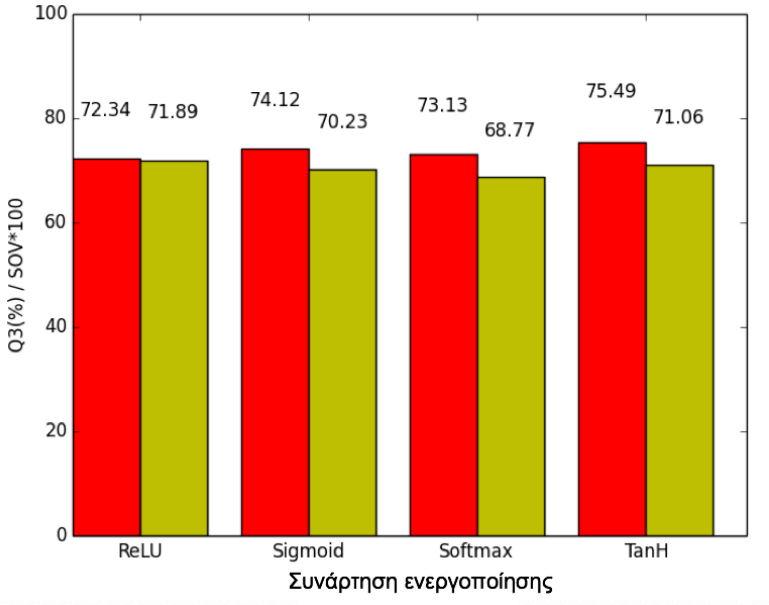
Ο Adam optimizer συνδυάζει τα πλεονεκτήματα του AdaGrad και του RMSprop. Ενισχύει την αποδοτικότητα σε προβλήματα με "αραιές" κλίσεις (sparse gradients) διατηρώντας ένα διαφορετικό ρυθμό μάθησης για κάθε παράμετρο του δικτύου και έχει αναφερθεί ότι είναι αποτελεσματικός σε on-line αλλά και μη στατικά (non-stationary) προβλήματα. Παρόλα αυτά, κάναμε κάποια πειράματα και για τους άλλους δύο για να εξεταστεί κατά πόσο ήταν κατάλληλοι για αυτό το πρόβλημα.

Τα αποτελέσματα του AdaGrad optimizer ήταν πολύ χαμηλά. Το Q3 score του ήταν 42.79% καθώς το SOV score του ήταν σχεδόν 4 φορές μικρότερο του. Αυτή η διαφορά ανάμεσα στο Q3 και SOV scores του είναι κατά μεγάλο βαθμό πιο μικρή στους άλλους optimizers. Από την άλλη, ο RMSprop optimizer ήταν πιο κατάλληλος για αυτό το πρόβλημα, αφού όπως παρατηρείται στο Σχήμα 5.3 τα αποτελέσματα του ήταν πολύ κοντά σε αυτά που επιτεύχθηκαν με τη χρήση του Adam optimizer, ο οποίος πέτυχε τα καλύτερα αποτελέσματα στο συγκεκριμένα σύνολο δεδομένων.

5.2.3 Συναρτήσεις ενεργοποίησης

Όπως ήδη αναφέραμε στο υποκεφάλαιο 4.2.2, οι συναρτήσεις ενεργοποίησης που χρησιμοποιήθηκαν αρχικά ήταν η υπερβολική εφαπτομένη για το κρυφό επίπεδο και η σιγμοειδής συνάρτηση για το επίπεδο εξόδου. Οι υπολογισμοί στα βάρη στο κρυφό επίπεδο είναι κρίσιμοι για το τελικό αποτέλεσμα, γι' αυτό το λόγο αποφασίσαμε να αλλάζουμε μόνο τη συνάρτηση ενεργοποίησης του κρυφού επιπέδου σε κάθε πείραμα που θα ακολουθήσει σε αυτό το υποκεφάλαιο. Οι υπόλοιπες παράμετροι του δικτύου έμειναν αμετάβλητες. Σε περισσότερη λεπτομέρεια, χρησιμοποιήθηκε ο Adam optimizer, τα clock periods τα οποία χρησιμοποιήθηκαν ήταν τα βέλτιστα (υποκεφάλαιο 5.2.1) και ο αριθμός των νευρώνων του κρυφού επιπέδου ήταν 28, ένας νευρώνας για κάθε τμήμα.

Διεξάχθηκαν ακόμη τρία πειράματα μετά το αρχικό μας, για να αποφασίσουμε την κατάλληλη συνάρτηση ενεργοποίησης για το κρυφό επίπεδο του δικτύου. Όπως παρατηρούμε στο Σχήμα 5.4, χρησιμοποιώντας την ReLU, το ποσοστό Q3 ήταν το πιο χαμηλό, αλλά το SOV score ήταν το πιο ψηλό, σε σύγκριση με τις υπόλοιπες. Παρόλο που η ReLU κατάφερε να πετύχει το πιο ψηλό SOV score, αποφασίσαμε να επιλέξουμε την υπερβολική συνάρτηση. Αυτό γιατί το SOV score της είναι ελάχιστα πιο χαμηλό από αυτό της ReLU, ενώ το ποσοστό Q3 της αρκετά πιο ψηλό. Στη συνέχεια, χρησιμοποιώντας τη σιγμοειδή συνάρτηση και στα δύο επίπεδα (χρησιμοποιείται ήδη ως συνάρτηση ενεργοποίησης για το επίπεδο εξόδου), τα αποτελέσματα τα οποία επιτευχθήκαν πλησίαζαν αυτά με τα καλύτερα ποσοστά. Στο τελευταίο πείραμα χρησιμοποιήσαμε τη softmax στο κρυφό επίπεδο όπου και είχαμε το μικρότερο SOV score, αλλά το Q3 ήταν μόνο 2% πιο χαμηλό από το καλύτερο αποτέλεσμα.

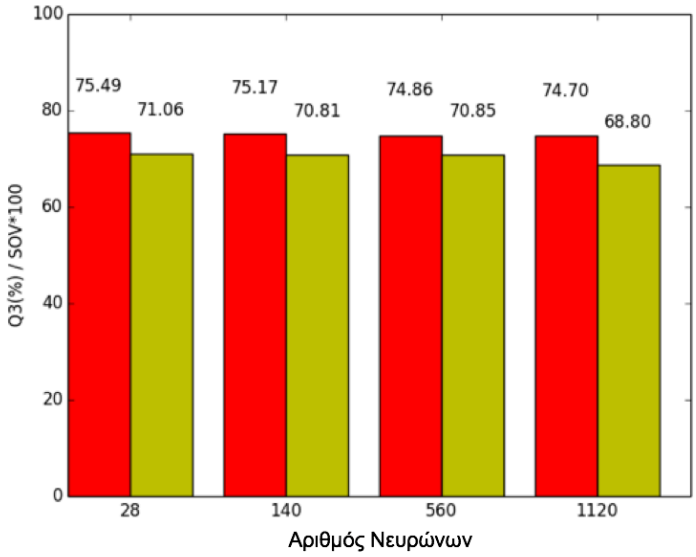


Σχήμα 5.4: Q3 ποσοστό ακριβείας και SOV score που πέτυχε η κάθε συνάρτηση ενεργοποίησης που χρησιμοποιήθηκε στο κρυφό επίπεδο του δικτύου

Κόκκινη στήλη: Q3 ποσοστό ακριβείας
Πράσινη στήλη: SOV score

5.2.4 Αριθμός νευρώνων κρυφού επιπέδου

Τα πρώτα πειράματα έχουν διεξαχθεί χρησιμοποιώντας όσο το δυνατό πιο λίγους νευρώνες, επειδή ο χρόνος εκτέλεσης του μοντέλου εξαρτάται επίσης και από το μέγεθος του, δηλαδή από πόσους νευρώνες απαρτίζεται. Γι' αυτό το λόγο προσπαθήσαμε να κρατήσουμε τον αριθμό των νευρώνων όσο πιο μικρό γίνεται για να επιτύχουμε ένα πολύ γρήγορο χρόνο εκτέλεσης, κάτι το οποίο θεωρείται ως κύριο χαρακτηριστικό για τα CW-RNNs. Ο μικρότερος αριθμός των νευρώνων που δικαιούμαστε να χρησιμοποιήσουμε περιορίζεται από τον αριθμό των τμημάτων (ο οποίος είναι ίσος με τον αριθμό των clock periods του μοντέλου) και κάθε τμήμα πρέπει να έχει τουλάχιστον ένα νευρώνα. Χρησιμοποιώντας μόλις 28 νευρώνες (δηλαδή ένα νευρώνα σε κάθε τμήμα), καταφέραμε να επιτύχουμε ικανοποιητικά Q3 και SOV scores. Στη συνέχεια, προσθέσαμε περισσότερους νευρώνες σε κάθε τμήμα, για να εξετάσουμε αν το μοντέλο θα μπορούσε να επεξεργαστεί περισσότερη πληροφορία με αυτό τον τρόπο για ένα καλύτερο αποτέλεσμα. Όπως παρατηρείται και στο Σχήμα 5.5, προσθέτοντας περισσότερους νευρώνες δεν επηρεάζονται ιδιαίτερα τα Q3 και SOV scores. Η αύξηση του μεγέθους του δικτύου με την πρόσθεση περισσότερων κρυφών νευρώνων, προσθέτει μόνο περισσότερη πολυπλοκότητα στο μοντέλο, το οποίο έχει ως επακόλουθο την αύξηση του χρόνου εκτέλεσης του, αλλά όχι την αύξηση των Q3 και SOV scores, αφού παραμένουν περίπου τα ίδια. Για την ταχύτητα εκτέλεσης του μοντέλου με τον αριθμό των νευρώνων που δοκιμάστηκαν σε αυτό το υποκεφάλαιο θα αναφερθούμε ξανά στο υποκεφάλαιο 5.4.



Σχήμα 5.5: Q3 ποσοστό ακρίβειας και SOV score για διαφορετικό αριθμό νευρώνων στο κρυφό επίπεδο

Κόκκινη στήλη: Q3 ποσοστό ακριβείας
Πράσινη στήλη: SOV score

5.3 10-fold cross-validation

Η πλήρης αξιολόγηση των αποτελεσμάτων του συνόλου δεδομένων CB513, ολοκληρώθηκε με έναν 10-fold cross-validation έλεγχο. Αυτός ο έλεγχος έπρεπε να γίνει για να επικυρωθεί η ευρωστία του μοντέλου και να αποδειχθεί η αποδοτικότητα του σε διάφορα δεδομένα εκπαίδευσης και επαλήθευσης. Όλες οι εκτελέσεις ολοκληρώθηκαν με τη χρήση των βελτιστοποιημένων παραμέτρων του μοντέλου, οι οποίες διεξάχθηκαν και περιεγράφηκαν στο υποκεφάλαιο 5.2.

5.3.1 Ensembles

Όπως έχουμε ήδη αναφέρει και στην αρχή αυτού του κεφαλαίου εφαρμόσαμε ensembles στα τελικά αποτελέσματα του κάθε fold. Τα αποτελέσματα των ensembles παρουσιάζονται στον Πίνακα 4. Οι Q3 τιμές κυμαίνονται από 73.74% μέχρι 77.07% και το SOV από 0.68 μέχρι 0.74. Αυτό δείχνει ότι τα αποτελέσματα από τα διαφορετικά folds μπορούν να συγκριθούν σε ποιότητα (δες επίσης και Πίνακα 5). Ακόμη, τα ποσοστά των τριών κλάσεων, H, E, C, υπολογίστηκαν ξεχωριστά (δες Q_H, Q_E, Q_C και SOV_H, SOV_E,

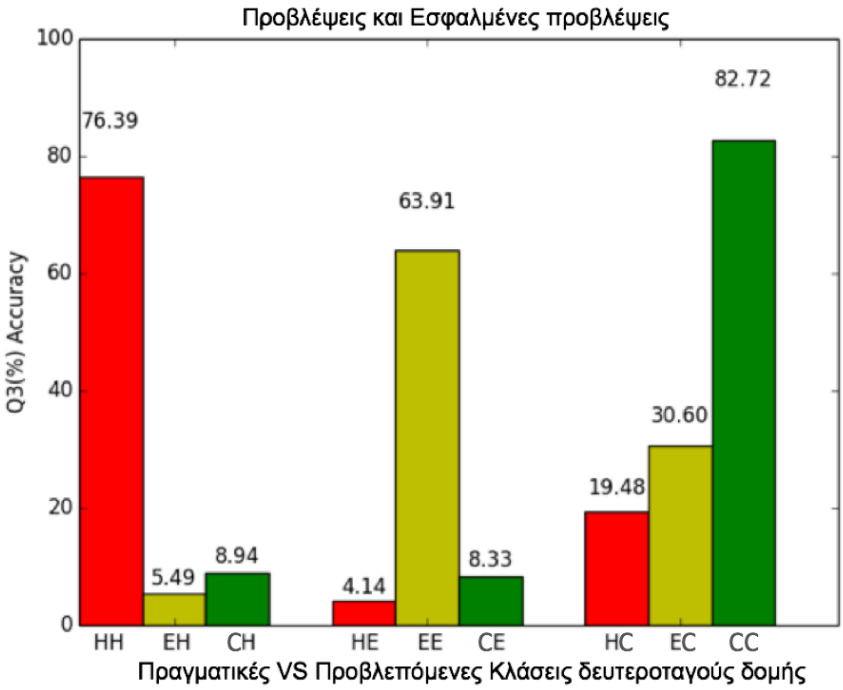
SOV_C στον Πίνακα 4) για βαθύτερη γνώση όσον αφορά την ποιότητα του classifier μας, και μπορούμε επίσης να παρατηρήσουμε τα ποσοστά των εσφαλμένων εκτιμήσεων τους (mispredictions) σε ένα confusion matrix, το οποίο παρουσιάζεται στο Σχήμα 5.6. Τα αποτελέσματα που παρουσιάζονται εδώ είναι συγκρίσιμα με αυτά που επιτεύχθηκαν με πιο πολύπλοκα BRNN μοντέλα (Kountouris et al,. 2012), απαιτούν όμως πολύ λιγότερο χρόνο για την εκπαίδευση των ξεχωριστών μοντέλων. Με βάση τα πιο πάνω, το μοντέλο μπορεί να εκπαιδευτεί επιτυχώς και να ελεγχθεί στις ακολουθίες των δεδομένων που υπάρχουν στο κάθε fold σε διαφορετική σειρά.

Πίνακας 4: Q3 ποσοστά ακριβείας και SOV scores για κάθε fold μετά την εφαρμογή των ensembles

Αρ. fold	Q_3(%)	Q_H(%)	Q_E(%)	Q_C(%)	SOV	SOV_H	SOV_E	SOV_C
0	76.72	75.80	65.81	80.92	**0.74**	0.78	0.70	**0.73**
1	75.49	73.35	67.08	81.49	0.70	0.72	0.70	0.70
2	75.70	73.39	64.48	81.37	0.72	0.74	0.69	0.71
3	76.90	68.52	60.35	84.37	**0.74**	0.72	0.63	**0.73**
4	75.10	72.02	56.70	82.15	0.73	0.74	0.62	0.72
5	73.74	63.30	62.06	82.35	0.70	0.65	0.66	0.71
6	76.00	71.53	61.94	**85.35**	0.72	0.73	0.68	0.72
7	74.59	69.14	67.54	81.22	0.68	0.67	0.70	0.69
8	76.13	67.61	**68.61**	82.24	**0.74**	0.70	**0.71**	**0.73**
9	**77.02**	**77.63**	63.39	84.17	0.72	**0.81**	0.67	0.72
Average	75.74	71.23	63.80	82.56	0.72	0.73	0.67	0.72

Πίνακας 5: Αποτελέσματα μετά την εφαρμογή των ensembles: στατιστική ανάλυση

	Q_3(%)	**SOV**
Sample Standard Deviation (**s**)	1.05	0.02
Variance ($\mathbf{s^2}$)	1.11	0.00
Mean (**Average**)	75.74	0.72
Standard Error of the Mean ($\mathbf{SE_{\bar{x}}}$)	0.33	0.01



Σχήμα 5.6: Προβλέψεις και εσφαλμένες προβλέψεις από τις κλάσεις H, E, C της δευτεροταγούς δομής, μετά την εφαρμογή ensembles σε κάθε fold. Παρουσιάζονται τα Q3 ποσοστά ακριβείας για κάθε κλάση.

5.3.2 Filtering: Εξωτερικοί κανόνες

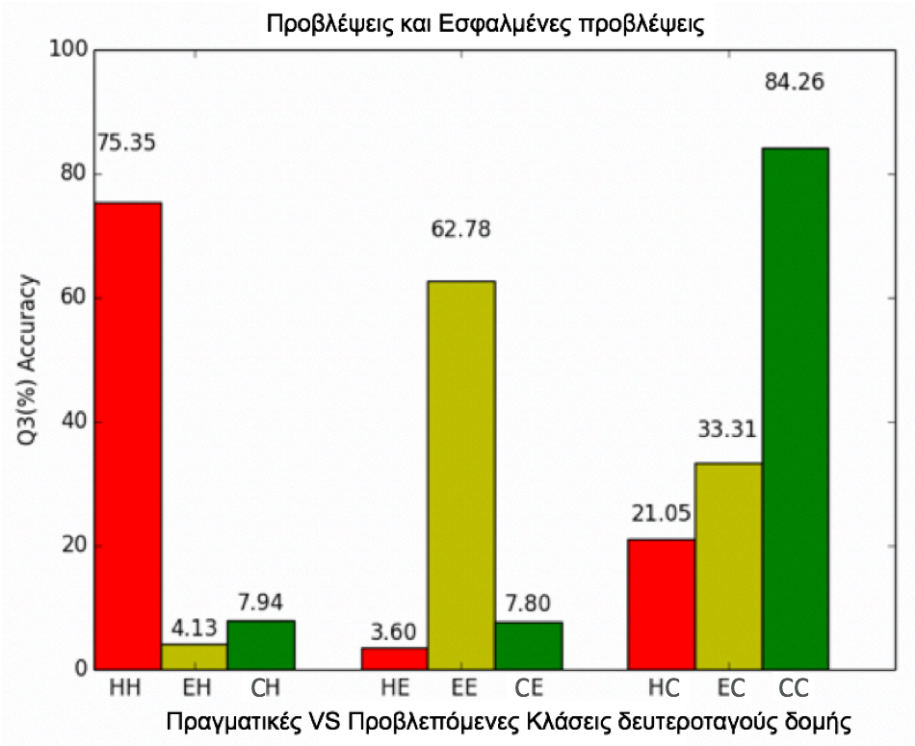
Η εφαρμογή φιλτραρίσματος στα folds είχε ως αποτέλεσμα την πολύ μικρή αύξηση του ολικού ποσοστού Q3 (κυμαίνεται μεταξύ των folds από 73.69% μέχρι 77.23%) και του ολικού SOV score (κυμαίνεται από 0.69 μέχρι 0.75) όπως παρατηρείται στον Πίνακα 6. Τα μισά από τα folds έχουν μειωμένο ποσοστό Q3, ενώ μόνο σε ένα από αυτά υπάρχει μια μείωση στο SOV score. Παρόλα αυτά και τα δύο, ολικό Q3 και SOV scores, στην τελική είχαν μια μικρή αύξηση λόγω των folds που είχαν αυξημένα ποσοστά μετά την εφαρμογή των εξωτερικών εμπειρικών κανόνων. Η τυπική απόκλιση (standard deviation) και η διαφορά (variance) του Q3 έχει αυξηθεί κατά πολύ λίγο σε σχέση με τις προβλέψεις οι οποίες δεν είχαν υποστεί φιλτράρισμα (σύγκριση Πίνακα 7 με Πίνακα 5), ενώ των SOV scores παρέμειναν στις ίδιες τιμές παρά την λιγοστή αύξηση του μέσου όρους τους. Το Q_H και το Q_E μειώθηκαν μετά το φιλτράρισμα κατά ≈1%, καθώς υπάρχει μια αύξηση στο ποσοστό του Q_C (Πίνακας 6).

Πίνακας 6: Q3 ποσοστά ακριβείας και SOV scores για κάθε fold μετά την εφαρμογή των εξωτερικών εμπειρικών κανόνων

Αρ. fold	Q_3(%)	Q_H(%)	Q_E(%)	Q_C(%)	SOV	SOV_H	SOV_E	SOV_C
0	**77.23**	75.06	64.82	83.17	**0.75**	0.79	0.70	**0.73**
1	75.41	71.49	65.06	83.53	0.72	0.73	0.69	0.71
2	75.41	71.49	65.06	83.53	0.72	0.73	0.69	0.71
3	76.90	67.57	58.88	85.55	0.74	0.73	0.63	**0.73**
4	75.06	70.56	55.10	83.15	0.72	0.73	0.61	0.70
5	73.69	62.30	60.23	83.97	0.72	0.66	0.64	0.71
6	76.28	70.42	61.27	**87.12**	0.74	0.75	0.68	**0.73**
7	75.00	67.79	65.50	83.10	0.69	0.71	0.68	0.68
8	76.05	65.27	**67.21**	83.96	**0.75**	0.69	**0.71**	**0.73**
9	**77.23**	**77.22**	61.24	85.48	**0.75**	**0.84**	0.65	**0.73**
Average	75.83	69.92	62.44	84.26	0.73	0.74	0.67	0.72

Πίνακας 7: Αποτελέσματα μετά την εφαρμογή εξωτερικών εμπειρικών κανόνων: στατιστική ανάλυση

	Q_3(%)	**SOV**
Sample Standard Deviation (**s**)	1.13	0.02
Variance ($\mathbf{s^2}$)	1.28	0.00
Mean (**Average**)	75.83	0.73
Standard Error of the Mean ($\mathbf{SE_{\bar{x}}}$)	0.36	0.01



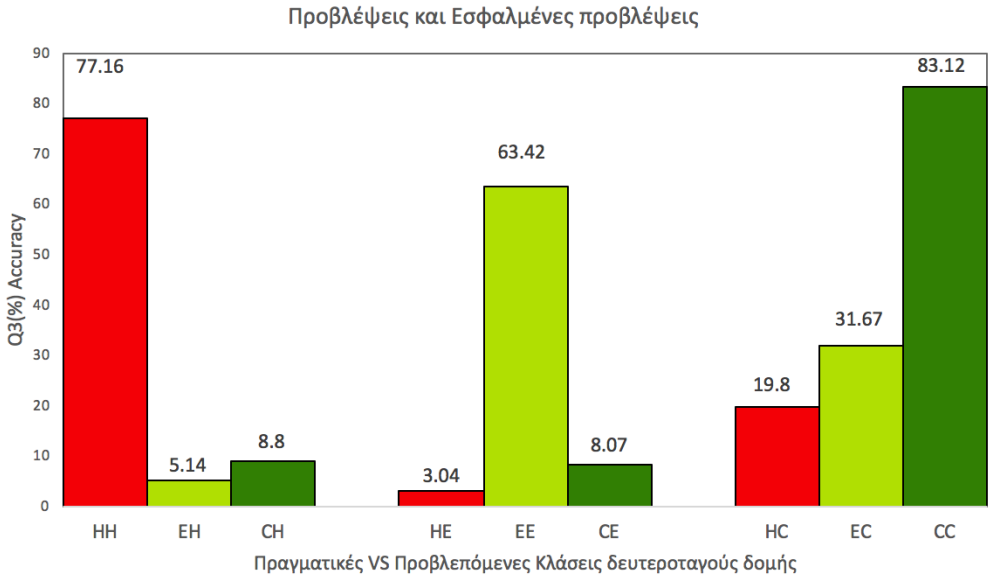
Σχήμα 5.7: Προβλέψεις και εσφαλμένες προβλέψεις από τις κλάσεις H, E, L της δευτεροταγούς δομής, μετά την εφαρμογή των εξωτερικών εμπειρικών κανόνων σε κάθε fold. Παρουσιάζονται τα Q3 ποσοστά ακριβείας για κάθε κλάση.

5.3.3 Filtering: Support Vector Machines

Η εφαρμογή των μηχανών διανυσμάτων υποστήριξης (support vector machines) τεχνικής για φιλτράρισμα στα folds είχε ως αποτέλεσμα την μικρή αύξηση του ολικού ποσοστού Q3 (κυμαίνεται μεταξύ των folds από 74.54% μέχρι 77.55%) όπως παρατηρείται στον Πίνακα 8. Σε όλα τα folds, εκτός από το fold με αριθμό 8, αυξήθηκε το ποσοστό Q3, με μέγιστο το 77.55%. Ως αποτέλεσμα, μετά την εφαρμογή των Support Vector Machines, το ολικό ποσοστό Q3 αυξήθηκε κατά 0.70%, φτάνοντας στο 76.44%. Το Q_C και το Q_E μειώθηκαν μετά το συγκεκριμένο φιλτράρισμα κατά πολύ λίγο, καθώς υπάρχει μια αύξηση στο ποσοστό του Q_H σχεδόν 5% (Πίνακας 8).

Πίνακας 8: Q3 ποσοστά ακριβείας για κάθε fold μετά την εφαρμογή Support Vector Machines

Αρ. fold	Q_3(%)	Q_H(%)	Q_E(%)	Q_C(%)
0	**77.55**	**79.00**	**67.00**	82.00
1	76.35	**79.00**	62.00	82.00
2	76.90	**79.00**	62.00	81.00
3	77.36	78.00	63.00	83.00
4	76.17	76.00	60.00	83.00
5	74.54	75.00	59.00	81.00
6	76.60	74.00	63.00	**84.00**
7	76.14	73.00	64.00	83.00
8	75.70	74.00	65.00	82.00
9	77.10	78.00	62.00	83.00
Average	76.44	76.5	62.70	82.40



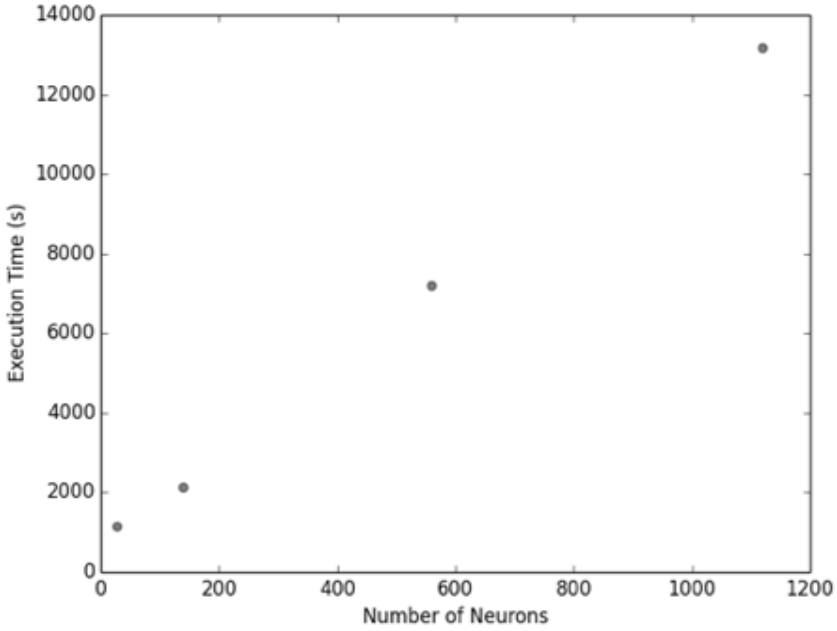
Σχήμα 5.8: Προβλέψεις και εσφαλμένες προβλέψεις από τις κλάσεις H, E, C της δευτεροταγούς δομής, μετά την εφαρμογή των Support Vector Machines σε κάθε fold. Παρουσιάζονται τα Q3 ποσοστά ακριβείας για κάθε κλάση.

5.4 Χρόνος εκτέλεσης

Όπως έχει ήδη ειπωθεί, ένα από τα κύρια χαρακτηριστικά της αρχιτεκτονικής του CW-RNN είναι ο γρήγορος χρόνος εκτέλεσης του. Έχει ελεγχθεί σε συγκεκριμένα προβλήματα-εφαρμογές όπου υπερισχύει του RNN σε αυτό τον τομέα (Koutník et al., 2014). Αυτό συμβαίνει γιατί χρησιμοποιεί λιγότερες παραμέτρους και εκτελεί λιγότερες λειτουργίες σε κάθε χρονική στιγμή λόγω του μικρότερου αριθμού συνδέσεων μεταξύ

των νευρώνων του κρυφού επιπέδου. Το CW-RNN παρουσίασε την απόδοση του στην ταχύτητα και στο πρόβλημα PSSP.

Καθώς αυξάνεται ο αριθμός των νευρώνων στο κρυφό επίπεδο, η πολυπλοκότητα του μοντέλου αυξάνεται παράλληλα. Για τα πειράματα που εκτελέσθηκαν για να αποφασίσουμε το βέλτιστο αριθμό κρυφών νευρώνων για το CW-RNN (υποκεφάλαιο 5.2.4), καταγράφαμε τον χρόνο εκτέλεσης τους, για να εξετάσουμε αν υπάρχει κάποια σχέση μεταξύ του χρόνου εκτέλεσης και του αριθμού των νευρώνων στο κρυφό επίπεδο. Όπως παρουσιάζεται στο Σχήμα 5.9, όταν ο αριθμός των νευρώνων αυξάνεται, ο χρόνος εκτέλεσης αυξάνεται γραμμικά. Παρατηρώντας ότι τα αποτελέσματα του CW-RNN για το συγκεκριμένο πρόβλημα, με την αύξηση των νευρώνων, παραμένουν τα ίδια, μας δίνει τη δυνατότητα να χρησιμοποιούμε μόνο απλά μοντέλα τα οποία μπορούν να εκπαιδευτούν σε πολύ μικρό χρονικό διάστημα (της τάξης των λεπτών στη συγκεκριμένη περίπτωση).



Σχήμα 5.9: Σχέση μεταξύ του χρόνου εκτέλεσης και του αριθμού των νευρώνων στο κρυφό επίπεδο του CW-RNN.

Κεφάλαιο 6

Συμπεράσματα και μελλοντική εργασία

6.1 Συμπεράσματα

6.2 Μελλοντική εργασία

6.1 Συμπεράσματα

Στο Κεφάλαιο 5, αναλύθηκαν και περιεγράφηκαν λεπτομερώς τα πειράματα και τα αποτελέσματα τους. Εφαρμόζοντας μόνο ensembles, το CW-RNN πετυχαίνει ποσοστά επιτυχίας Q3 75.74% και SOV 0.72. Τα ποσοστά αυτά θεωρούνται πολύ καλά αφού και οι state-of-the-art μέθοδοι έχουν παρόμοια και μπορούν να συγκριθούν με αυτές. Τα τελικά όμως αποτελέσματα κρίθηκαν από το φιλτράρισμα που ακολουθήθηκε. Τα φιλτραρίσματα αυτά έγιναν ξεχωριστά και είναι ανεξάρτητα πειράματα. Το φιλτράρισμα που δοκιμάστηκε αρχικά ήταν οι εξωτερικοί εμπειρικοί κανόνες. Με αυτό, το ποσοστό Q3 αυξήθηκε στο 75.83% και το SOV στο 0.73. Το φιλτράρισμα με χρήση SVMs εφαρμόστηκε στα αποτελέσματα των ensembles και αύξησε περισσότερο το αποτέλεσμα του ποσοστού Q3 από τους εμπειρικούς κανόνες, φτάνοντας στο 76.44%.

Γενικά, παρά την πολυπλοκότητα του προβλήματος, το μοντέλο ολοκληρώνει την εκτέλεση του σε σύντομο χρονικό διάστημα και σίγουρα πιο γρήγορα σε σύγκριση με τις τυπικές αρχιτεκτονικές, οι οποίες είναι βασισμένες σε RNNs και BRNNs. Ταυτόχρονα, έχει παρόμοια ποιότητα πρόβλεψης, όπως παρατηρούμε από τα πιο πάνω αποτελέσματα, όπως αυτά με πολύ πιο πολύπλοκες αρχιτεκτονικές ανάδρασης που δοκιμάστηκαν μέχρι στιγμής.

Υψηλής σημασίας η πρόβλεψη του μεσαίου αμινοξέως του κινητού παραθύρου

Για την επίτευξη όμως των πιο πάνω αποτελεσμάτων καίριας σημασίας ήταν η πρόβλεψη του μεσαίου αμινοξέως του κινητού παραθύρου, παρά οποιοδήποτε άλλο. Συμπεράναμε ότι με την πρόβλεψη του τελικού αμινοξέως του παραθύρου, όπου δοκιμάσαμε αρχικά, το πιο ψηλό ποσοστό Q3 του δικτύου ήταν μόλις 65%. Όταν όμως αρχίσαμε να προβλέπουμε το μεσαίο αμινοξύ τα αποτελέσματα Q3 αυξήθηκαν σχεδόν 10%, λόγω του ότι αρχίσαμε να λαμβάνουμε υπόψη περισσότερη πληροφορία για κάθε αμινοξύ από τα γειτονικά του. Ως αποτέλεσμα, κατανοούμε ότι η επιλογή για το ποια θα είναι η θέση του αμινοξέως που θα προβλέπουμε αποτελεί έναν πολύ σημαντικό παράγοντα στα αποτελέσματα μας.

Το μοντέλο μαθαίνει

Το δίκτυο καθώς εκπαιδεύεται μαθαίνει, αφού μειώνεται το σφάλμα, με αποτέλεσμα να μαθαίνει βασικές και κυρίως τοπικές εξαρτήσεις μεταξύ των δεδομένων εισόδου. Από την άλλη όμως δεν μπορεί να ανακαλύψει και να μάθει σε μεγάλο βαθμό κάποιες συγκεκριμένες εξαρτήσεις. Όπως παρατηρείται στους Πίνακες 4, 6, και 8 η κλάση δευτεροταγούς δομής E (β-strands) έχει τα πιο χαμηλά ποσοστά επιτυχίας, πράγμα το οποίο σημαίνει ότι το δίκτυο δε μπορεί να μάθει αποτελεσματικά τη συγκεκριμένη κλάση, συντελώντας στη μείωση του ολικού ποσοστού. Αυτό μπορεί να οφείλεται είτε στο γεγονός ότι το σύνολο δεδομένων που χρησιμοποιούμε δεν είναι αντιπροσωπευτικό για τη συγκεκριμένη κλάση, είτε στο γεγονός ότι τα τμήματα της κλάσης E μπορεί να βρίσκονται μακριά το ένα από το άλλο διαχωρισμένα από άλλες κλάσεις δευτεροταγούς δομής. Αυτό συντελεί στο γεγονός ότι το CW-RNN πιθανότατα να μην μπορεί να ανακαλύψει μακρινές εξαρτήσεις στο μέγιστο βαθμό. Παρόλα αυτά, το δίκτυο καταφέρνει να προβλέψει τις άλλες κλάσεις με ψηλά ποσοστά με αποτέλεσμα το τελικό ποσοστό να είναι αρκετά ψηλό και να μπορεί να συγκριθεί με state-of-the-art μεθόδους.

Γρήγορος χρόνος εκτέλεσης

Όπως αναγράφεται και στον Πίνακα 2, χρησιμοποιήθηκαν διαφορετικά μεγέθη batches και όσο μικρότερα ήταν αυτά σε μέγεθος, τόσο πιο γρήγορα ολοκληρώνει την εκτέλεση του το μοντέλο. Αυτό μας επέτρεψε την συστηματική εκτέλεση πειραμάτων για τη διεξαγωγή των υπερπαραμέτρων και την ολοκλήρωση του 10-fold cross-validation σε μικρό χρονικό διάστημα. Ως αποτέλεσμα, μπορούμε να συμπεράνουμε ότι η ταχύτητα εκπαίδευσης των CW-RNNs, τα καθιστά κατάλληλα για τη δημιουργία ensemble classifiers συνδυάζοντας ένα μεγάλο αριθμό από δίκτυα. Τα δίκτυα αυτά θα εκπαιδεύονται με διαφορετικές υπερπαραμέτρους για να μαθαίνουν διαφορετικά, συγκεκριμένα χαρακτηριστικά των δεδομένων εισόδου, όπου ένα μόνο δίκτυο θα ήταν ανίκανο να καλύψει όλο το εύρος των σημαντικών χαρακτηριστικών.

Ακόμη ένα συμπέρασμα που εξήγαμε από τα παραπάνω πειράματα είναι ότι τα αποτελέσματα του δικτύου δεν αλλάζουν καθώς το δίκτυο μεγαλώνει, δηλαδή καθώς αυξάνεται ο αριθμός των κρυφών νευρώνων σε κάθε τμήμα. Η σχέση μεταξύ του χρόνου

εκτέλεσης και του αριθμού των κρυφών νευρώνων είναι γραμμική. Αυτό έχει ως συνέπεια ότι το δίκτυο αυτό μπορεί να χρησιμοποιηθεί σε προβλήματα που χρειάζονται περισσότερους νευρώνες για την επεξεργασία των δεδομένων τους, χωρίς όμως να υπάρξει εκθετική αύξηση στο χρόνο εκτέλεσης.

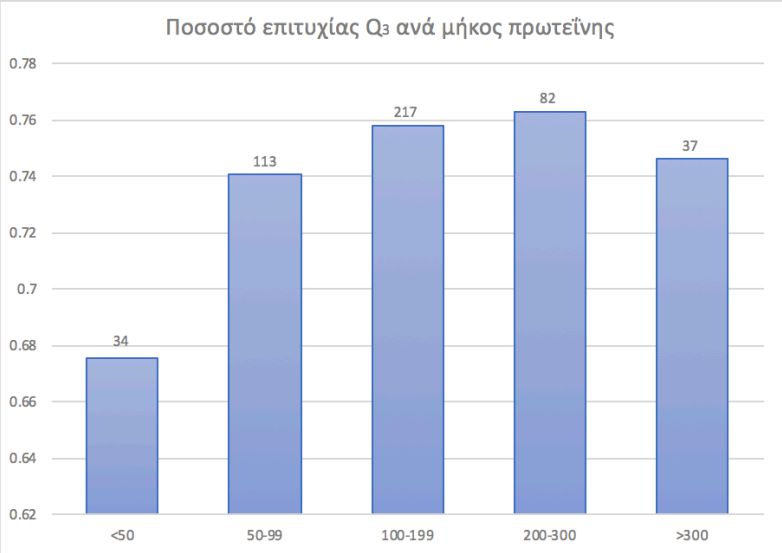
Ζωτικής σημασίας η σωστή επιλογή και σειρά των clock periods

Έχουμε ανακαλύψει ότι τα clock periods είναι το κλειδί για καλά αποτελέσματα. Τα clock periods είναι απαραίτητο να προσαρμόζονται ανάλογα με το πρόβλημα που χρησιμοποιείται το παρόν μοντέλο (CW-RNN). Έχουμε συμπεράνει από τα αποτελέσματα των Clock Period Συνόλων 1 και 2 (υποκεφάλαιο 5.2.1), ότι τα αποτελέσματα του μοντέλου δεν εξαρτώνται από τον αριθμό των clock periods, δηλαδή τον αριθμό των τμημάτων του δικτύου. Αυτό που έχει σημασία είναι η σωστή επιλογή και σειρά των clock periods, αφού κάθε διαφορετικό σύνολο περιόδων έχει διαφορετική επίδραση στο πρόβλημα. Στο συγκεκριμένο πρόβλημα το βέλτιστο σύνολο clock periods ήταν αυτό το οποίο είχε μια συμμετρία, λόγω της συμμετρίας που υπάρχει στο κινητό παράθυρο σε σχέση με το μεσαίο αμινοξύ για το οποίο εκτελείται η πρόβλεψη. Επίσης, περιείχε αρκετά γρήγορα τμήματα για να μπορεί να επεξεργάζεται περισσότερη πληροφορία σε κάθε χρονική στιγμή.

Το ποσοστό επιτυχίας πιθανόν να εξαρτάται από το μήκος των πρωτεϊνών

Όπως παρατηρούμε στο Σχήμα 6.1, υπάρχουν 34 πρωτεΐνες στο σύνολο δεδομένων CB513, οι οποίες έχουν μέγεθος <50 αμινοξέα και δίνουν τα χαμηλότερα ποσοστά επιτυχίας, τα οποία βρίσκονται γύρω στο 67.5%. Το ποσοστό επιτυχίας Q_3 αυξάνεται στο 74% στις πρωτεΐνες με μέγεθος 50-99 αμινοξέα, οι οποίες είναι 113. Η επόμενη κατηγορία που έχουμε είναι οι πρωτεΐνες με μέγεθος 100-199 αμινοξέα, οι οποίες καλύπτουν ένα μεγάλο μέγεθος του συνόλου δεδομένων και έδωσαν ποσοστό επιτυχίας 75.8%. Στη συνέχεια, έχουμε την κατηγορία με το καλύτερο ποσοστό επιτυχίας Q_3, η οποία περιέχει πρωτεΐνες με μέγεθος 200-300 αμινοξέα και έδωσε ποσοστά επιτυχίας γύρω στο 76.2%. Τέλος, έχουμε την κατηγορία με τις μεγαλύτερες σε μέγεθος πρωτεΐνες, οι οποίες είναι μόλις 37 σε αριθμό και έδωσαν ποσοστό επιτυχίας περίπου 74%. Συμπεράναμε ότι οι πρωτεΐνες που δίνουν τα καλύτερα αποτελέσματα είναι αυτές με

μέγεθος το οποίο κυμαίνεται από 100 μέχρι 300 αμινοξικά κατάλοιπα. Σε μικρότερες ή μεγαλύτερες σε μέγεθος πρωτεΐνες παρατηρούμε ένα μειωμένο ποσοστό επιτυχίας, ενώ σε πρωτεΐνες οι οποίες έχουν πολύ μικρό μέγεθος (<50 αμινοξικά κατάλοιπα) παρατηρήσαμε το πιο χαμηλό ποσοστό επιτυχίας.



Σχήμα 6.1: Ποσοστό επιτυχίας Q_3 ανά μήκος πρωτεΐνης. Ο αριθμός που βρίσκεται πάνω από τις μπάρες είναι ο αριθμός των πρωτεϊνών που βρίσκεται σε κάθε κατηγορία.

y-axis: ποσοστό επιτυχίας Q_3;
x-axis: το μήκος των πρωτεϊνών που βρίσκονται σε κάθε κατηγορία

Για πιο ακριβή όμως αποτελέσματα, υπολογίσαμε, από μια εκτέλεση του fold με ονομασία "fold0", τις δέκα καλύτερες και δέκα χειρότερες πρωτεΐνες σε ποσοστό επιτυχίας Q_3 (Σχήμα 6.2, Σχήμα 6.3). Ταυτόχρονα, υπολογίστηκε και το SOV τους. Με αυτό τον τρόπο επαληθεύσαμε ότι οι μικρότερες σε μέγεθος πρωτεΐνες δίνουν χειρότερα αποτελέσματα αφού όπως βλέπουμε στο Σχήμα 6.2, οι περισσότερες χειρότερες πρωτεΐνες είναι σχετικά μικρές. Αντίθετα, στο Σχήμα 6.3, παρατηρούμε ότι οι πρωτεΐνες που έδωσαν τα καλύτερα ποσοστά επιτυχίας Q_3, είναι μεγάλες, εκτός δύο, που αποτελούνται από λίγα αμινοξικά κατάλοιπα. Εκτός από αυτά όμως, είναι φανερή η διαφορά που υπάρχει στο ποσοστό επιτυχίας Q_3 και SOV σε κάποιες από τις πρωτεΐνες. Η χειρότερη πρωτεΐνη έχει το πιο χαμηλό ποσοστό επιτυχίας Q_3, αλλά ένα σχετικά ψηλό SOV, ενώ η καλύτερη πρωτεΐνη έχει χαμηλότερο SOV από αυτήν, περίπου 7 μονάδες.

Πρωτεΐνη	Q3	SOV
1ednA_1-21	52.4%	0.762
4cpaI_3-38	61.1%	0.421
1bdoA_77-156	62.5%	0.574
1dikA_373-520	63.9%	0.622
1gkyA_33-82	64%	0.497
1bfgA_19-144	64.3%	0.625
1azuA_4-127	66.1%	0.608
1glnA_323-370	66.7%	0.67
1eclA_2-163	67.6%	0.624
1smnB_5-245	68.5%	0.612

Σχήμα 6.2: Οι 10 χειρότερες πρωτεΐνες σε ποσοστό επιτυχίας Q_3

Τα αποτελέσματα πάρθηκαν από το fold με ονομασία “fold0”

Πρωτεΐνη	Q3	SOV
2asrA_38-179	90.1%	0.694
1edmC_46-84	89.7%	0.796
4pfkA_1-320	89.3%	0.827
1lbuA_1-82	87.6%	0.846
2cmdA_147-312	86.7%	0.832
3ecaB_213-326	84.2%	0.886
2ccyA_2-128	81.9%	0.586
1scuE_240-388	81.2%	0.818
1aorB_1-211	81%	0.869
1powB_360-549	80.5%	0.879

Σχήμα 6.3: Οι 10 καλύτερες πρωτεΐνες σε ποσοστό επιτυχίας Q_3

Τα αποτελέσματα πάρθηκαν από το fold με ονομασία “fold0”

Γενικά συμπεράσματα

Με βάση τον Koutník, η χρήση των CW-RNNs είναι καθοριστική σε προβλήματα παραγωγής και κατηγοριοποίησης ακολουθιών οι οποίες περιέχουν μακρινές εξαρτήσεις

(Koutník et al., 2014). Συνοψίζοντας όμως τα πιο πάνω συμπεράσματα, παρατηρήσαμε ότι μπορεί να χρησιμοποιηθεί σε τέτοιου είδους προβλήματα και ακόμη πιο πολύπλοκα, εφόσον όμως ανακαλυφθεί το κατάλληλο σύνολο clock periods. Επίσης, εξαρτάται και από την πολυπλοκότητα του προβλήματος. Δοκιμάζοντας το CW-RNN στο PSSP, το οποίο θεωρείται ένα πολύπλοκο πρόβλημα, συνειδητοποιήσαμε ότι δεν μπορεί να ανακαλύψει μακρινές εξαρτήσεις στο μέγιστο βαθμό, με αποτέλεσμα να περιορίζεται στα παρόντα αποτελέσματα. Ακόμη, συμπεράναμε ότι το CW-RNN είναι ιδανικό και για προβλήματα στα οποία χρησιμοποιούνται μεγάλα σύνολα δεδομένων. Χρησιμοποιώντας το CW-RNN, μπορούν να εκτελεστούν πολλά πειράματα, αφού θα χρειάζονται σχετικά μικρό χρόνο εκτέλεσης, σε σχέση με άλλα δίκτυα με πιο πολύπλοκες αρχιτεκτονικές. Με αυτό τον τρόπο, έχουμε την ευκαιρία να ανακαλύψουμε τις υπερπαραμέτρους του δικτύου για τα προβλήματα αυτά σε σύντομο χρονικό διάστημα για το καλύτερο δυνατό αποτέλεσμα. Τέλος, υπάρχει η δυνατότητα εκτέλεσης πειραμάτων με διαφορετικές υπερπαραμέτρους για να μπορούν να καλύψουν το εύρος όλων των χαρακτηριστικών που μπορεί να υπάρχουν στο πρόβλημα. Στοχεύοντας σε διαφορετικά χαρακτηριστικά του προβλήματος σε κάθε εκτέλεση και εξάγοντας πολλά τέτοια πειράματα, μπορούμε να δημιουργήσουμε έναν ensemble classifier, ο οποίος θα είναι πιο αποδοτικός. Το CW-RNN, συνιστάται να χρησιμοποιηθεί σε μεγάλου μεγέθους, πολύπλοκα προβλήματα με βάση τους προαναφερθέντες λόγους.

6.2 Μελλοντική εργασία

Έχοντας υπόψη το πόσο πρόσφατη είναι η συγκεκριμένη αρχιτεκτονική του CW-RNN, υπάρχουν περιθώρια για βελτίωση και έρευνα.

Αρχικά, το σύνολο των clock periods, μπορεί να αντικατασταθεί από άλλα που υπάρχει ακόμη μεγαλύτερη σχέση μεταξύ των περιόδων τους. Η επιλογή αυτή είναι πολύ δύσκολη και πρέπει να διεξαχθούν πολλά πειράματα και να γίνει αρκετή έρευνα. Προσθέτοντας σε αυτό, με τη βοήθεια εξελικτικών αλγορίθμων θα μπορούσαν να εκπαιδεύονται τα clocks, ταυτόχρονα με τα βάρη σε κάθε χρονική στιγμή, για διαφορετική επεξεργασία των δεδομένων ανάλογα με τη σημασία τους για το πρόβλημα (Koutník, 2014).

Ακόμη ένα σημείο το οποίο θα μπορούσε να δοκιμαστεί είναι να αφαιρεθεί ο περιορισμός που υπάρχει στην παρούσα υλοποίηση του δικτύου, ότι τα τμήματα πρέπει να έχουν τον ίδιο αριθμό νευρώνων. Αφαιρώντας αυτό τον περιορισμό τα διαφορετικού μεγέθους τμήματα θα είχαν την ευκαιρία να προσαρμοστούν καλύτερα στα δεδομένα της εισόδου του δικτύου.

Επιπρόσθετα, ακόμη μια εισήγηση για μελλοντική εργασία είναι η υλοποίηση ενός BCWRNN. Η αρχιτεκτονική αυτή θα συνδυάζει δύο CW-RNNs στα άκρα και ένα δίκτυο εμπρόσθιου περάσματος στο κέντρο. Τα δύο CW-RNNs, θα εκπαιδεύονται με διαφορετικές υπερπαράμετρους με αποτέλεσμα το κάθε ένα από αυτά να μαθαίνει διαφορετικά χαρακτηριστικά του προβλήματος. Θα γίνεται χρήση κινητού παραθύρου και τα αμινοξέα πριν του προβλεπόμενου αμινοξέως θα δίνονται ως είσοδος στο ένα CW-RNN, ενώ τα αμινοξέα που βρίσκονται μετέπειτα αυτού, στο άλλο. Στη συνέχεια, μετά την επεξεργασία τους, θα συνδυάζονται, αφού το αποτέλεσμα του κάθε δικτύου θα δίνεται ως είσοδος στο δίκτυο εμπρόσθιου περάσματος για το τελικό αποτέλεσμα. Με αυτό τον τρόπο θα γίνεται καλύτερη επεξεργασία των μικρότερων κομματιών του κινητού παραθύρου κάθε φορά και με τον συνδυασμό τους θα συντελούν σε πιθανόν καλύτερα τελικά αποτελέσματα.

Επιπλέον, είναι πολύ σημαντικό στο μέλλον να δοκιμαστεί ένα διαφορετικό σύνολο δεδομένων, το οποίο είναι πιο μεγάλο αλλά και πιο αντιπροσωπευτικό και για τις τρεις ευρείες κλάσεις της δευτεροταγούς δομής. Ως αποτέλεσμα, θα μπορούμε να καταλάβουμε αν τελικά το μοντέλο μας είναι ικανό να αναγνωρίζει αποτελεσματικά μακρινές εξαρτήσεις μεταξύ των δεδομένων εισόδου. Το σύνολο όμως αυτό καλό είναι να το χωρίσουμε σε N+1 τμήματα, όπου οι υπερπαράμετροι του δικτύου θα βελτιστοποιηθούν με ένα από αυτά και το δίκτυο στη συνέχεια θα εκπαιδεύεται και θα ελέγχεται με τα υπόλοιπα N για έναν αντικειμενικό έλεγχο της απόδοσης του δικτύου.

Τα πειράματα που είχαν χρησιμοποιηθεί για τα ensembles στην παρούσα διπλωματική εργασία έχουν εξαχθεί με τη χρήση ίδιων υπερπαραμέτρων. Θα μπορούσαν να δοκιμαστούν επίσης ensembles με χρήση πειραμάτων τα οποία εξάχθηκαν από δίκτυα με διαφορετικές υπερπαραμέτρους. Χρησιμοποιώντας αυτή την τεχνική, στο κάθε πείραμα

το δίκτυο θα μπορούσε να μάθει διαφορετικά χαρακτηριστικά των δεδομένων εισόδου και κατά τον συνδυασμό τους τα αποτελέσματα να βελτιωθούν περισσότερο.

Τέλος, θα ήταν καλό το CW-RNN να δοκιμαστεί για την επίλυση άλλων προβλημάτων, αφού είναι περιορισμένος ο αριθμός των προβλημάτων στα οποία εφαρμόστηκε μέχρι στιγμής και υπόσχεται πολύ καλά αποτελέσματα με την κατάλληλη τροποποίηση.

Αναφορές

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

Παυλίδης Π., “Πρόβλεψη Δευτεροταγούς Δομής Των Πρωτεϊνών Με Τη Χρήση Των Convolutional Neural Networks Για Οπτική Αναγνώριση Αντικειμένων”, Προπτυχιακή διπλωματική, Πανεπιστήμιο Κύπρου, 2016.

Πετρίδου Ει., “ΠΡΟΒΛΕΨΗ ΔΕΥΤΕΡΟΤΑΓΟΥΣ ΔΟΜΗΣ ΠΡΩΤΕΙΝΩΝ” , Προπτυχιακή διπλωματική, Πανεπιστήμιο Κύπρου, Λευκωσία, 2015.

Χριστοδούλου Γ., “Διερεύνηση μεθόδων εκπαίδευσης νευρωνικών δικτύων αμφίδρομης ανάδρασης για πρόβλεψη δευτεροταγούς δομής πρωτεϊνών” , Προπτυχιακή διπλωματική, Πανεπιστήμιο Κύπρου, Λευκωσία, 2010.

Baldi P., Brunak S., Frasconi P., Soda G. and Pollastri G. “Bidirectional Dynamics for Protein Secondary Structure Prediction”, Sequence Learning, R. Sun and L. Giles Editors, Springer Verlag, Lecture Notes in Artificial Intelligence 1828, 80-104, 2000.

Baldi P., Brunak S., Frasconi P., Soda G. and Pollastri G. ”Exploiting the Past and the Future in Protein Secondary Structure Prediction”, Bioinformatics, 15(11), 937-946, 1999.

Blazewicz J., Hammer P. L. and Lukasiak P., "Predicting secondary structures of proteins," in IEEE Engineering in Medicine and Biology Magazine, 24(3), 88-94, 2005.

Chen J. and Chaudhari N. S. “Cascaded Bidirectional Recurrent Neural Networks for Protein Secondary Structure Prediction”, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 14(4), 572-582, 2007.

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

Cortes, C. & Vapnik, V. Support-vector network. Machine Learning, 20, 1–25, 1995.

Cuff J.A. and Barton G.J. ‘‘Evaluation and Improvement of Multiple Sequence for Protein Secondary Structure Prediction”, Proteins, 34, 508-519, 1999.

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

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

Heffernan R, Yang Y, Paliwal K, Zhou Y. “Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility”. Bioinformatics, 33(18), 2842-2849, 2017.

Heffernan R, Paliwal K, Lyons J, Dehzangi A, Sharma A, Wang J, Sattar A, Yang Y, Zhou Y, “Improving prediction of secondary structure, local backbone angles, and solvent accessible surface area of proteins by iterative deep learning”, Sci Rep 5, 11476, 2015.

Hochreiter S., Schmidhuber J., "Long short-term memory", Neural Computation. 9(8), 1735–1780, 1997.

Jordan, M. I. *“*Attractor dynamics and parallelism in a connectionist sequential machine*”*. Proceedings of the Cognitive Science Society, 531-546, 1986.

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

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

Kingma, D. P. and Lei Ba, J., “Adam: A method for stochastic optimization” , arXiv preprint arXiv: 1412.6980, 2017.

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

Koutník, J., Greff, K., Gomez, F., Schmidhuber, J. “A clockwork RNN”, arXiv preprint arXiv:1402.3511v1, 2014.

Lin, C.W. May, R. Taylor, “Amino Acid Encoding Schemes from Protein Structure Alignments: Multi-dimensional Vectors to Describe Residue Types”, Journal of Theoretical Biology, 216(3), 361-365, 2002.

Mirabello, C., & Pollastri, G. “Porter, PaleAle 4.0: high-accuracy prediction of protein secondary structure and relative solvent accessibility”. Bioinformatics, 29, 2056-2058, 2013.

Mozer C.M ‘‘A Focused Backpropagation Algorithm for Temporal Pattern Recognition’’, Complex Systems, 3, 349-381, 1989.

Pollastri, G., & Mclysaght, A. “Porter: a new, accurate server for protein secondary structure prediction”. Bioinformatics, 21(8), 1719-1720, 2005.

Pollastri, G., Przybylski, D., Rost, B., & Baldi, P. “Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles”. Proteins: Structure, Function, and Bioinformatics, 47(2), 228-235, 2002.

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

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

Rost B, Sander C. “Improved prediction of protein secondary structure by use of sequence profiles and neural networks”. Proc Natl Acad Sci USA, 90, 7558–62, 1993.

Rumelhart D.E., Hinton G.E. and Williams R.J. ‘‘Learning Internal Representation by Error Propagation. In Rumelhart D. E. and McClelland J. L. (Eds), Parallel Distributed

Processing: Explorations in the Microstructure of Cognition'', MIT Press, Cambridge, MA, 1, 318-362, 1986.

Salamov A. A. and Solovyev V. V., "Prediction of protein secondary structure by combining nearest-neighbor algorithms and multiple sequence alignments". J. Mol. Biol., 247(1), 11–15, 1995.

Salamov A. A. and Solovyev V. V., "Protein secondary structure prediction using local alignments" J. Mol. Biol., 268, 31-36, 1997.

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

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

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

Yang, Yuedong & Gao, Jianzhao & Wang, Jihua & Heffernan, Rhys & Hanson, Jack & Paliwal, Kuldip & Zhou, Yaoqi. "Sixty-five years of the long march in protein secondary structure prediction: the final stretch?", Briefings in bioinformatics. 10.1093/bib/bbw129, 2016.

Yi TM, Lander ES. "Protein secondary structure prediction using nearest-neighbor methods". J Mol Biol, 232, 1117–29, 1993.

Zemla, A, Venclovas, C., Fidelis, K. and Rost, B. "A modified definition of Sov, a segment-based measure for protein secondary structure prediction assessment", Proteins, 34(2), 220-223, 1999.

Zvelebil MJ. and Baum J., "Understanding Bioinformatics", Garland Science, Taylor & Francis Group, New York – London, 2008.

Zvelebil MJ, Barton GJ, Taylor WR and Sternberg MJ . “Prediction of protein secondary structure and active-sites using the alignment of homologous sequences”. J Mol Biol, 195, 957–61, 1987.

Γενική Βιβλιογραφία

"Άλφα έλικα", ΒΙΚΙΠΑΙΔΕΙΑ, Τελευταία τροποποίηση 16:56, 6 Μαΐου 2016, https://el.wikipedia.org/wiki/Άλφα_έλικα, (accessed 7 April 2018).

"Αμινοξέα", ΒΙΚΙΠΑΙΔΕΙΑ, Τελευταία τροποποίηση 10:07, 5 Μαΐου 2017, https://el.wikipedia.org/wiki/Αμινοξέα, (accessed 20 October 2017).

Αργυράκη Π., "Κεφάλαιο 3. Βιολογικά νευρωνικά δίκτυα", Computational Physics Group, A.V.Th., http://kelifos.physics.auth.gr/COURSES/neural/K3.pdf (accessed 25 April 2018).

Βούτου Ε., "Η σημασία της πρωτεΐνης στη διατροφή μας", medNutrition , 31 Δεκεμβρίου 2014, https://www.mednutrition.gr/portal/lifestyle/diatrofi/8910-i-simasia-tis-proteinis-sti-diatrofi-mas, (accessed 20 October 2017).

Επιχειρησιακό Πρόγραμμα - Εκπαίδευση και δια βίου μάθηση, "Κεφάλαιο 3 – Πρωτεΐνες", Βιοχημεία – Βιβλίο Μαθητή, http://ebooks.edu.gr/modules/ebook/show.php/DSGL-C120/480/3166,12748/ (accessed 7 April 2018).

Ηλιόπουλος Η., 'Μοριακή Αναγνώριση', Γεωπονικό Πανεπιστήμιο Αθηνών, Αθήνα 2001, https://mediasrv.aua.gr/eclass/modules/document/file.php/BIOTECH139/MOPIAKH_ANAΓΝΩΡΙΣΗ_ΘΕΩΡΙΑ3.pdf, (accessed 19 April 2017).

Οικονομίδου Β., Χαμόδρακας Σ., "Εφαρμογή στην πρόβλεψη της δευτεροταγούς δομής της πρωτεΐνης bovine pancreatic trypsin inhibitor (BPTI)", Τομέας Βιολογίας Κυττάρου και Βιοφυσικής, Τμήμα Βιολογίας, Παν/μιο Αθηνών, Φεβρουάριος 2002, http://biophysics.biol.uoa.gr/courses/biophysics/prognwsh/ASKISI_1[2].html (accessed 19 April 2017).

Σπηλιόπουλος Ι., Ξαπλαντέρη Μ., "Πρωτεΐνες", Kallipos Repository, https://repository.kallipos.gr/bitstream/11419/934/1/02_chapter_22.pdf, (accessed 7 April 2018).

Christodoulou C. EPL442 notes., Nicosia 2016 – 2017.

Brownie J., "A Gentle Introduction to Backpropagation Through Time", Machine Learning Mastery, Τελευταία τροποποίηση 23 Ιουνίου 2017, https://machinelearningmastery.com/gentle-introduction-backpropagation-time/ (accessed 3 May 2018).

Hinton G., Srivastava N., Swersky K., "Lecture 6a Overview of mini-batch gradient descent", https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (accessed 10 May 2018).

Lauralee Sherwood, "HUMAN PHYSIOLOGY: From Cells to Systems" , Eighth Edition, 2013.

Raschka S., "What's the difference between gradient descent and stochastic gradient descent?", Τελευταία τροποποίηση 18 Νοεμβρίου 2015, https://www.quora.com/Whats-the-difference-between-gradient-descent-and-stochastic-gradient-descent (accessed 8 May 2018).

Ruder S. "An overview of gradient descent optimisation algorithms". arXiv preprint arXiv:1609.04747, 2017.

Sharma V A., "Understanding Activation Functions in Neural Networks", The Theory Of Everything, https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 (accessed 25 April 2018).

Παράρτημα Α (train.py)

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

from datetime import datetime
import time
import os
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random

from models.clockwork_rnn import ClockworkRNN
import storeProteins
from config import Config

def train(config):

    window = 11  # Size of sliding window
    shuffleSwitch = 1

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

        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.sizeOfAminoacids

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

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

    # start time of the program
    startTime = time.time()

    # train proteins
    open_file = open("../clockworkRNN-master/input/trainCB513/trainSet0.txt",
"r")
    lines = open_file.readlines()

    pTrain = storeProteins.storeProteins()
    pTest = storeProteins.storeProteins()

    training = lines[0:]
    proteinListTrainData = pTrain.readProteins(training, window - 1)  # for
the +(PSS) field to be added
    init_training(config, pTrain)

    # test proteins
    open_file = open("../clockworkRNN-master/input/testCB513/testSet0.txt",
"r")
    lines = open_file.readlines()
```

```
    testing = lines[0:]
    proteinListTestData = pTest.readProteins(testing, window - 1)  # for the
+(PSS) field to be added
    init_testing(config, pTest)

    # the size of each test protein
    sizeOfEachProtein = []
    for i in range(2, len(proteinListTestData), 4):
        sizeOfEachProtein.append(len(proteinListTestData[i]))

    # the size of each train protein
    sizeOfEachTrainProtein = []
    for i in range(2, len(proteinListTrainData), 4):
        sizeOfEachTrainProtein.append(len(proteinListTrainData[i]))

    # Shuffling proteins before new epoch
    shuffledIndex = list(range(0,len(sizeOfEachTrainProtein)))

    # No. of proteins train/validation
    print("Train proteins ",len(sizeOfEachTrainProtein))
    print("Validation proteins ",len(sizeOfEachProtein))

    # Create the correct form of data!!!
    def _load_data(data, data2, n_prev=window):

        # Step 4
        # data should be pd.DataFrame()
        docX, docY = [], []

        addData = 1
        for i in range(n_prev, len(data)+1):

                # REMOVE IF: amino acids in the beginning, middle row are tens
(dummy value)
                if int(data.iloc[i-n_prev, 0]) != 10 and int(data.iloc[i-
int(window/2)-1, 0]) == 10:
                    addData = 0

                # REMOVE IF: amino acids in the end, middle row are tens
(dummy value)
                if int(data.iloc[i-1, 0]) != 10 and int(data.iloc[i-
int(window/2)-1, 0]) == 10:
                    addData = 0

                if addData == 1:
                    docX.append(data.iloc[i-n_prev:i].as_matrix())

                addData = 1

        for j in range(0, len(data2)):
            docY.append(data2.iloc[j].as_matrix())

        # Step 5 from List to Array
        alsX = np.array(docX)
        alsY = np.array(docY)

        return alsX, alsY

    def generate_data(pTrain):

        # Step 2
        pTrain.data = pTrain.data.reshape(pTrain.sizeOfAminoacids, 20)
        pTrain.yt = np.array(pTrain.yt)
```

```
        pTrain.yt.reshape(pTrain.sizeOfCleanAminoacids, 3)

        # Step 3 DataFraming both pTrain.data/pTest.data and
pTrain.yt/pTest.yt
        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 _load_data(pdata.iloc[0:], pdata2.iloc[0:])

    # Load the training data
    print("[x] Generating training examples...")
    (X_train, y_train) = generate_data(pTrain)  # Production of train data
Step 1
    print("[x] Generating test examples...")
    (X_validation, y_validation) = generate_data(pTest)  # Production of test
data Step 1

    # Zero padding the test batch in order to feed all the train data to the
network
    validationSizeBeforePatch = len(y_validation)
    paddingSize = config.batch_size - len(X_validation)

    # Zero padding validation data
    paddingZeros = np.zeros((paddingSize, window, 20))
    X_validation = np.concatenate((X_validation, paddingZeros))

    # Zero padding validation targets
    paddingZeros = np.zeros((paddingSize, 3))
    y_validation = np.concatenate((y_validation, paddingZeros))

# ------------------------------------------------------------------------ #

    num_train      = X_train.shape[0]       # No. of sliding windows which
will be fed to the network (training)
    num_validation = X_validation.shape[0]  # No. of the sliding windows
(testing)

    config.num_steps  = X_train.shape[1]    # Size of the window
    config.num_input  = X_train.shape[2]    # No. of inputs
    config.num_output = y_train.shape[1]    # No. of outputs

    print("----------- ABOUT TRAINING ----------")
    print("X_train.shape")
    print(X_train.shape[0])  # No. of sliding windows which will be fed to the
network (training)
    print(X_train.shape[1])  # Size of the window
    print(X_train.shape[2])  # No. of inputs
    print("y_train.shape")
    print(y_train.shape[0])  # No. of sliding windows which will be fed to the
network (train targets)
```

```
    print(y_train.shape[1])  # No. of outputs
    print("----------- ABOUT TESTING -----------")
    print("X_validation.shape")
    print(X_validation.shape[0])  # No. of sliding windows which will be fed
to the network (testing)
    print(X_validation.shape[1])  # Size of the window
    print(X_validation.shape[2])  # No. of inputs
    print("y_validation.shape")
    print(y_validation.shape[0])  # No. of sliding windows which will be fed
to the network (validation targets)
    print(y_validation.shape[1])  # No. of outputs

    strTime = time.time()  # Execution of the actual model

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

    step_in_epoch = 0
    steps_per_epoch = int(math.floor((len(X_train))/config.batch_size))
    num_steps = steps_per_epoch*config.num_epochs

    print("------------- MORE INFO -------------")
    print("config.batch_size ", config.batch_size)
    print("steps_per_epoch ", steps_per_epoch) # No. of batches
    print("num_steps ", num_steps)
    print("-"*40)

# ------------------------------------------------------------------------ #

    # 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)

    # Can be viewed using Tensorboard
    # Initialize summary writer
    summary_out_dir = os.path.join(config.output_dir, "trainsummaries")
    summary_writer  = tf.summary.FileWriter(summary_out_dir, sess.graph)

    # Initialize summary test writer
    summary_out_dirtest = os.path.join(config.output_dir, "testsummaries")
    summary_writertest = tf.summary.FileWriter(summary_out_dirtest,
sess.graph)

# ------------------------------------------------------------------------ #

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

```
    sumTotal = 0.0  # test accuracy
    tmpPredictedStructure = []  # contains the predicted SS as a list => will
be divided into each protein
    tmpTrainPredictedStructure= []  # contains the PSS as a list (train) =>
will be divided into each protein
    sumTrainEpoch = 0
    sumTrainTotal = 0.0  # total train accuracy
    y_train_counter = 0
    epoch = 1

    for t 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, learning_rate, train_summary,
train_predictions = sess.run(
            [model.train_op,
             model.global_step,
             model.loss,
             model.learning_rate,
             model.train_summary_op,
             model.predictions,
             ],
            feed_dict={
                model.inputs:  X_train[index_start:index_end, ],
                model.targets: y_train[index_start:index_end, ],
            }

        )

        '''
        After each epoch calculate the train accuracy and in the last training
epoch create a list containing
        the predicted secondary structure of all the proteins
        '''
        # Calculate total train accuracy
        sumTrainCorr = 0  # train accuracy of each batch

        for k in range(0, config.batch_size):

            # Winner Takes All
            if train_predictions[k, 0] > train_predictions[k, 1] and
train_predictions[k, 0] > train_predictions[k, 2]:

                if (num_steps - train_step) <= int(len(X_train) /
config.batch_size) - 1:
                    tmpTrainPredictedStructure.append('C')
                if y_train[y_train_counter, 0] == 1:
                    sumTrainCorr += 1

            elif train_predictions[k, 1] > train_predictions[k, 0] and
train_predictions[k, 1] >train_predictions[k, 2]:

                if (num_steps - train_step) <= int(len(X_train) /
config.batch_size) - 1:
                    tmpTrainPredictedStructure.append('E')
                if y_train[y_train_counter, 1] == 1:
                    sumTrainCorr += 1
```

```
            elif train_predictions[k, 2] > train_predictions[k, 0] and
train_predictions[k, 2] >train_predictions[k, 1]:

                if (num_steps - train_step) <= int(len(X_train) /
config.batch_size) - 1:
                    tmpTrainPredictedStructure.append('H')
                if y_train[y_train_counter, 2] == 1:
                    sumTrainCorr += 1

            y_train_counter += 1

        sumTrainEpoch += sumTrainCorr

        # End of train accuracy's calculation

        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))

        # 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:

            print("Train Accuracy: %.3f" % (( float(sumTrainEpoch) /
(config.batch_size*steps_per_epoch)) * 100), "%")

            sumTrainTotal += sumTrainEpoch
            sumTrainEpoch = 0
            y_train_counter = 0
            print("Epoch No ", epoch, " ended.")
            epoch += 1

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

            sumCorr = 0  # test accuracy

            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, predictions, test_summary = \
                    sess.run([model.loss, model.predictions,
model.train_summary_op],

                    feed_dict={
                        model.inputs:  X_validation[index_start:index_end, ],
                        model.targets: y_validation[index_start:index_end, ],
```

```
                    }
                )

            for k in range(0, validationSizeBeforePatch):

                    # WTA
                    if predictions[k, 0] > predictions[k, 1] and
predictions[k, 0] > predictions[k, 2]:

                        if train_step == num_steps:
                            tmpPredictedStructure.append('C')
                        if y_validation[k, 0] == 1:
                            sumCorr += 1

                    elif predictions[k, 1] > predictions[k, 0] and
predictions[k, 1] > predictions[k, 2]:

                        if train_step == num_steps:
                            tmpPredictedStructure.append('E')
                        if y_validation[k, 1] == 1:
                            sumCorr += 1

                    elif predictions[k, 2] > predictions[k, 0] and
predictions[k, 2] > predictions[k, 1]:

                        if train_step == num_steps:
                            tmpPredictedStructure.append('H')
                        if y_validation[k, 2] == 1:
                            sumCorr += 1

            # Show a plot of the ground truth and prediction of the signal
            if validation_step == 0:
                    plt.clf()
                    plt.title("Ground Truth and Predictions")
                    plt.plot(y_validation[index_start:index_start+50, 0],
label="signal 0 (input)")
                    plt.plot(predictions[0:50, 0], ls='--', label="signal 0
(prediction)")
                    plt.plot(y_validation[index_start:index_start+50, 1],
label="signal 1 (input)")
                    plt.plot(predictions[0:50, 1], ls='--', label="signal 1
(prediction)")

                    legend = plt.legend(frameon=True)
                    legend.get_frame().set_facecolor('white')
                    plt.draw()
                    plt.pause(0.001)


            sumTotal = sumTotal + sumCorr
            print("No. of amino acids predicted correct - Testing: ",sumCorr,
" / ", validationSizeBeforePatch)
            print("Test Accuracy (per Epoch): %.3f" % ((float(sumCorr) /
validationSizeBeforePatch) * 100), "%")
            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
```

```
            # checking for test summary
            summary_writertest.add_summary(test_summary, epoch)

            # Shuffle the proteins after each epoch

            if shuffleSwitch == 1 and train_step != num_steps:

                proteinStart = 0
                proteinList = []
                proteinTargets = []
                for i in range(0, len(X_train)):

                    if int(X_train[i, int(window/2) +1, 0]) == 10 and
int(X_train[i, window - 1, 0]) == 10:
                        endBool = 1
                        proteinEnd = i


                        if endBool == 1:
                            if proteinStart == 0:

proteinList.append(X_train[proteinStart:proteinEnd+1])

proteinTargets.append(y_train[proteinStart:proteinEnd+1])
                            else:

proteinList.append(X_train[proteinStart+1:proteinEnd+1])

proteinTargets.append(y_train[proteinStart+1:proteinEnd + 1])

                            proteinStart = proteinEnd

                # Shuffle the train data
                c = list(zip(proteinList, proteinTargets, shuffledIndex))
                random.shuffle(c)
                proteinList, proteinTargets , shuffledIndex = zip(*c)

                # Assembling training data to input form
                tempFinal = np.zeros((len(X_train), window, 20))
                pos = 0
                for i in proteinList:
                    tempFinal[pos:pos + i.shape[0]] = i
                    pos += i.shape[0]

                X_train = tempFinal

                # Assembling training targets to input form
                tempFinal = np.zeros((len(y_train), 3))
                pos = 0
                for i in proteinTargets:
                    tempFinal[pos:pos + i.shape[0]] = i
                    pos += i.shape[0]

                y_train = tempFinal

            # End of Shuffling
            np.set_printoptions(threshold=np.inf)


            print("#" * 100)

# ------------------------------------------------------------------------ #

    # Finished all epochs
    endTimeSeconds = time.time()
```

```
    # Printing the proteinListTestData with the Predicted Secondary Structure
added
    f = open("outputData.txt", 'w')
    predictedStructurePlace = 3
    tmp = 0
    for i in range(0, len(sizeOfEachProtein)):

        del proteinListTestData[predictedStructurePlace]
        proteinListTestData.insert(predictedStructurePlace, ''

.join(tmpPredictedStructure[tmp:(tmp+sizeOfEachProtein[i] - 1)] + ['\n']))

        tmp = tmp + sizeOfEachProtein[i] - 1
        predictedStructurePlace += 4

    for ele in proteinListTestData:
        f.write(ele)
    f.close

    # Printing the proteinListTrainData with the Predicted Secondary Structure
added
    f = open("outputTrainData.txt", 'w')
    tmp = 0

    for i in range(0, len(sizeOfEachTrainProtein)):
        for j in range(0, len(sizeOfEachTrainProtein)):
            if j == shuffledIndex[i]:

                del proteinListTrainData[j*4+3]
                proteinListTrainData.insert((j*4+3), ''.join(
                tmpTrainPredictedStructure[tmp:(tmp +
sizeOfEachTrainProtein[j]-1)] + ['\n']))

                tmp = tmp + sizeOfEachTrainProtein[j] - 1


    for ele in proteinListTrainData:
        f.write(ele)
    f.close

    print("Test Accuracy: ", (sumTotal /
(validationSizeBeforePatch*config.num_epochs)) * 100)
    print("Train Accuracy: ", (sumTrainTotal /
(config.batch_size*steps_per_epoch*config.num_epochs)) * 100)

    endTime = time.time()
    print("Program's execution: ", (endTime-startTime)," seconds")
    print("Model's execution: ", (endTimeSeconds-strTime)," seconds")

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


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

Παράρτημα Β (storeProteins.py)

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

class storeProteins:

    def __init__(self):
        self.sizeOfAminoacids = 0
        self.aminoacids = []
        self.data = []
        self.yt = []
        self.sizeOfCleanAminoacids = 0
        self.aminoC = 0
        self.aminoE = 0
        self.aminoH = 0

    '''
    Reads data from file and loads the MSA representation which corresponds
to.
    Adds tens (dummy value), equal to (windowSize-1) / 2, at the beginning and
the end of the protein.
    The sliding window that is created is to help us predict the middle amino
acid.
    The main data structure that is created is all the sliding windows of each
protein combined.
    It also returns a data structure with the format:
    (Protein name\nPrimary Structure\nCorrect secondary structure\n +\n) where
+ is to be replaced with the predicted
    secondary structure. If the sequence has unknown symbols i.e. '!' it
removes them.
    '''
    def readProteins(self, lines, window):

        proteins = []
        tens = zeros((1, int(window/2) * 20)) + 10

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

            name = lines[i].rstrip()

            proteins.append(name+'\n')

            secondary = lines[i + 2].rstrip()
            proteins.append(lines[i + 1].rstrip().replace('!', '') + '\n')  #
protein's first structure

            proteins.append(secondary.replace('!', '') + '\n')  # protein's
secondary structure
            proteins.append('+\n')  # protein's predicted secondary structure
which will be added later in the + field

            msa = loadtxt("../clockworkRNN-master/input/msaFilesCorrect/" +
name + '.hssp')

            self.data = append(self.data, tens)
            self.data = append(self.data, (msa * 1.0) / 100.0)
            self.data = append(self.data, tens)

            self.sizeOfAminoacids += len(msa) + window
```

```
                self.sizeOfCleanAminoacids += len(msa)
                self.normilizeOutput(secondary, window)

        return proteins

    # Converts the secondary structure class into neuron activation.
    def normilizeOutput(self, secondary, window):

        se = secondary.split()

        for i in range(0, int(window/2), 1):
            self.aminoacids.append("10")

        for t in se[0]:

            if t == "C":
                self.yt.append([1, 0, 0])
                self.aminoacids.append(t)
                self.aminoC +=1

            elif t == "E":
                self.yt.append([0, 1, 0])
                self.aminoacids.append(t)
                self.aminoE +=1

            elif t == "H":
                self.yt.append([0, 0, 1])
                self.aminoacids.append(t)
                self.aminoH += 1

            else:
                continue  # ignores if ! or anything different than the three
above

        for i in range(0, int(window / 2), 1):
           self.aminoacids.append("10")
```

Παράρτημα Γ (config.py)

```
class Config(object):

    output_dir = "./output/"

    # Clockwork RNN parameters
    periods     = [89, 55, 34, 21, 13, 8, 5, 3, 3, 2, 2, 2, 1, 1, 1, 1, 2, 2,
2, 3, 3, 5, 8, 13, 21, 34, 55, 89]
    num_steps   = 100
    num_input   = 20
    num_hidden  = 28
    num_output  = 3

    # Optmization parameters
    num_epochs          = 1200
    batch_size          = 8535
    optimizer           = 'adam'
    max_norm_gradient   = 10.0

    # Learning rate decay schedule
    learning_rate       = 1e-2
    learning_rate_decay = 1.0
    learning_rate_step  = 1000
    learning_rate_min   = 1e-5
```

Παράρτημα Δ (clockwork_rnn.py)

```
import numpy as np
import tensorflow as tf

class ClockworkRNN(object):
    '''
    A Clockwork RNN - Koutnik et al. 2014 [arXiv,
https://arxiv.org/abs/1402.3511]

    The Clockwork RNN (CW-RNN), in which the hidden layer is partitioned into
separate modules,
    each processing inputs at its own temporal granularity, making
computations only at its prescribed clock rate.
    Rather than making the standard RNN models more complex, CW-RNN reduces
the number of RNN parameters,
    improves the performance significantly in the tasks tested, and speeds up
the network evaluation

    '''

    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.random_normal_initializer(stddev=0.01)

        initializer_bias = tf.constant_initializer(0.0)

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

        # 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]))

                # 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)

            self.predictions = activation_output(self.predictions,
name="output")

            # 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])
```

Παράρτημα Ε

################ README ################

Requires:

Python	v.2.7
TensorFlow	v.1.8.0
Pandas	v.0.22.0
NumPy	v.1.14.3
MatplotLib	v.1.5.3

Running the code:

1. Navigate to the folder of the file train.py
2. Open a terminal in this destination
3. Type: python train.py and hit enter
4. Watch the magic happens!

Παράρτημα ΣΤ (**Επιπλέον δοκιμές διαφορετικών clock periods)**

Επεξήγηση αυτού του παραρτήματος στο υποκεφάλαιο 5.2.1.

Οι υπερπαράμετροι παραμένουν σταθερές.

Μέγεθος κινητού παραθύρου = 11
Αριθμός κρυφών νευρώνων = αριθμός των clock periods
Optimizer = Adam (beta1= 0.9, beta2= 0.99, ε= 10^{-9}, ρυθμός μάθησης = 0.01)
Συνάρτηση ενεργοποίησης κρυφού επιπέδου = Υπερβολική εφαπτομένη
Συνάρτηση ενεργοποίησης επιπέδου εξόδου = Σιγμοειδής
Αρχικοποίηση βαρών δικτύου με μικρές τιμές με τυπική απόκλιση 0.01

Αρ. πειράματος	**Clock period σύνολο**	**Ποσοστό επιτυχίας Q_3 (%)**
1	{3,3,2,2,1,1,1,1,2,2,3,3}	72.92
2	{14,5,3,2,2,1,1,1,2,2,3,5,14}	72.97
3	{5,4,3,2,1,1,1,2,3,4,5}	73.29
4	{8,5,3,2,2,1,1,1,2,2,3,5,8}	73.65
5	{4,3,2,2,1,1,1,2,2,3,4}	73.66
6	{13,8,5,3,2,2,1,1,1,1,2,2,3,5,8,13}	73.71
7	{1,1,2,2,3,3,5,8,5,3,2,2,1,1}	73.83
8	{27,9,3,3,2,2,1,1,1,1,2,2,3,3,9,27}	73.87
9	{1,1,1,2,2,3,89,144,233,144,89,3,2,2,1,1,1}	74.01
10	{21,13,8,5,3,2,2,1,1,1,1,2,2,3,5,8,13,21}	74.07
11	{8,5,3,2,2,1,1,1,1,2,2,3,5,8}	74.07
12	{21,13,8,5,3,2,2,1,1,2,2,3,5,8,13,21}	74.12
13	{3,3,3,2,2,2,1,1,1,1,2,2,2,3,3,3}	74.14
14	{1,1,1,2,2,3,3,5,8,13,21,13,8,5,3,3,2,2,1,1,1}	74.15
15	{34,21,13,8,5,3,2,2,1,1,2,2,3,5,8,13,21,34}	74.16
16	{20,15,10,5,3,3,2,2,1,1,1,1,2,2,3,3,5,10,15,20}	74.26
17	{32,18,6,3,3,2,2,1,1,1,1,2,2,3,3,6,18,32}	74.31
18	{1,1,1,1,2,2,2,3,3,5,8,13,21,34,34,21,13,8,5,3,3,2,2,2,1,1,1,1}	74.34
19	{3,3,3,2,2,1,1,1,1,2,2,3,3,3}	74.35
20	{1,1,1,2,2,3,3,5,8,13,21,34,55,81,55,34,21,13,8,5,3,3,2,2,1,1,1}	74.52
21	{4,4,3,3,2,2,1,1,1,1,2,2,3,3,4,4}	74.54
22	{13,8,5,3,2,2,2,1,1,1,1,2,2,2,3,5,8,13}	74.55
23	{32,18,6,3,2,2,1,1,1,1,2,2,3,6,18,32}	74.70
24	{1,1,1,2,2,2,3,3,5,6,13,21,34,21,13,8,5,3,3,2,2,2,1,1,1}	74.78
25	{18,6,3,3,2,2,1,1,1,1,2,2,3,3,6,18}	74.80
26	{1,1,1,2,2,3,3,5,8,13,21,34,55,34,21,13,8,5,3,3,2,2,1,1,1}	74.82
27	{21,13,8,5,3,3,2,2,1,1,1,1,2,2,3,3,5,8,13,21}	74.87
28	{21,13,8,5,3,2,2,2,2,1,1,1,1,2,2,2,2,3,5,8,13,21}	74.88
29	{21,13,8,5,3,3,2,2,2,1,1,1,1,2,2,2,3,3,5,8,13,21}	74.88
30	{1,2,1,3,2,5,3,8,5,13,8,21,13,21,13,21,8,13,5,8,3,5,2,3,1,2,1}	74.89