

**LEARNING SCHEMES FOR EFFICIENTLY TRAINING NEURAL
NETWORKS ON PROTEIN SECONDARY STRUCTURE PREDICTION**

Michalis Agathocleous

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

July, 2019

© Copyright by

Michalis Agathocleous

All Rights Reserved

2019

Συστήματα Μάθησης για αποτελεσματική εκπαίδευση Νευρωνικών Δικτύων στην Πρόβλεψη Δευτεροταγούς Δομής Πρωτεϊνών

Μιχάλης Αγαθοκλέους

Πανεπιστήμιο Κύπρου, 2019

Το πεδίο της δημιουργίας μη-γραμμικών συναρτήσεων για την ταξινόμηση ή κατηγοριοποίηση δεδομένων από ακολουθίες αντιμετωπίζεται επί του παρόντος από μια ποικιλία μεθόδων Μηχανικής Μάθησης. Η πρόκληση αυτής της διαδικασίας αυξάνεται όταν πέραν από την πληροφορία που προηγείται ενός σημείου μιας ακολουθίας, η πληροφορία που έπεται αυτού του σημείου είναι επίσης σημαντική για την διεκπεραίωση μιας πρόβλεψης ή κατηγοριοποίησης. Πιο συγκεκριμένα, σε αυτή την κατηγορία προβλημάτων, η επεξεργασία ενός συγκεκριμένου σημείου της ακολουθίας δεδομένων γίνεται όχι μόνο με βάση την πληροφορία που προηγείται από τα αριστερά προς τα δεξιά του σημείου αυτού, αλλά και την πληροφορία που έπεται από τα δεξιά προς τα αριστερά του ίδιου σημείου. Η αρχιτεκτονική Νευρωνικών Δικτύων Αμφίδρομης Ανάδρασης (ΝΔΑΑ) σχεδιάστηκε από τους Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999] για να αντιμετωπίσει αυτή την κατηγορία προβλημάτων όπου η πληροφορία που προηγείται και η πληροφορία που έπεται είναι αναγκαίες για την επεξεργασία των δεδομένων μιας ακολουθίας. Η συγκεκριμένη αρχιτεκτονική αποτελείται από δυο απλά Νευρωνικά Δίκτυα Ανάδρασης (ΝΔΑ) τα οποία επεξεργάζονται την πληροφορία που προηγείται και την πληροφορία που έπεται ενός συγκεκριμένου σημείου μια ακολουθίας δεδομένων, αντίστοιχα. Αυτά τα απαιτητικά μοντέλα χρειάζονται αλγόριθμους μάθησης υψηλής απόδοσης όσον αφορά το χρόνο σύγκλισης και τις ανάγκες μνήμης ώστε να εκπαιδεύονται αποδοτικά και αποτελεσματικά. Γενικότερα, η εκπαίδευση των ΝΔΑ είναι

αρκετά περίπλοκη διαδικασία με αποτέλεσμα να είναι ένα ανοιχτό θέμα στον τομέα της Μηχανικής Μάθησης.

Επιπλέον, τα ΝΔΑ μπορούν να συνδυαστούν σε Σύνολα (ensembles) Μοντέλων Μηχανικής Μάθησης (ΣΜΜΜ) με άλλα μοντέλα και τεχνικές Νευρικών Δικτύων (ΝΔ) για περαιτέρω βελτίωση των αποτελεσμάτων σε διάφορα προβλήματα. Τέτοιες τεχνικές ΝΔ είναι τα μπλοκ μνήμης Long-Short Term Memory, τα Echo State Networks, οι Conceptors, τα Clockwork ΝΔΑ ή τα Συνελκτικά Νευρωνικά Δίκτυα (ΣΝΔ). Η αξιολόγηση της απόδοσης των δικτύων αυτών με βάση τον βαθμό ακρίβειας πρόβλεψης ή ταξινόμησης σε κάποιο πρόβλημα δεν είναι ικανοποιητική. Η αυξημένη πολυπλοκότητα αυτών των μεθόδων απαιτεί γρήγορο χρόνο σύγκλισης και χαμηλές απαιτήσεις υπολογιστικής μνήμης για την επίλυση απαιτητικών προβλημάτων.

Ένα σημαντικό πρόβλημα ταξινόμησης δεδομένων από ακολουθίες όπου η πληροφορία που προηγείται και η πληροφορία που έπεται ενός σημείου μιας ακολουθίας είναι σημαντική για την διεκπεραίωση μιας πρόβλεψης, είναι το πρόβλημα της Πρόβλεψης Δευτεροταγούς Δομής Πρωτεϊνών (ΠΔΔΠ). Ένας αυξανόμενος αριθμός πρωτεϊνικών αλληλουχιών τις οποίες συνθέτουν αμινοξέα είναι γνωστός. Ωστόσο, υπάρχει απουσία πληροφορίας σχετικά με την τρισδιάστατη δομή αυτών των πρωτεϊνών, η οποία είναι πολύ σημαντική αφού καθορίζει τη λειτουργία τους. Η δευτεροταγής δομή αυτών των πρωτεϊνικών αλληλουχιών είναι ένα τεχνητό ενδιάμεσο στάδιο μεταξύ της πρωτεϊνικής αλληλουχίας και της τρισδιάστατης δομής μίας πρωτεΐνης, το οποίο υποδεικνύει τις τοπικές πτυχές των αμινοξέων. Αυτές οι τοπικές πτυχές δημιουργούνται με βάση τις αλληλεπιδράσεις των αμινοξέων που βρίσκονται πριν και μετά από μια συγκεκριμένη θέση της πρωτεΐνης. Το πρόβλημα αυτό είναι περίπλοκο και χρειάζεται αλγόριθμους που

μπορούν να επεξεργαστούν και να μάθουν τα χαρακτηριστικά των δεδομένων των ακολουθιών, έτσι ώστε να επιτύχουν το καλύτερο δυνατό αποτέλεσμα.

Σε αυτή τη διατριβή, αρχικά, παρουσιάζουμε μια παραλλαγή της αρχιτεκτονικής NNAA που σχεδιάστηκε από τον Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999] για την αντιμετώπιση του προβλήματος της ΠΔΔΠ. Αυτή η αρχιτεκτονική θεωρείται σήμερα ως μία από τις βέλτιστες αρχιτεκτονικές ΝΔ για την αντιμετώπιση αυτού του προβλήματος. Πιο συγκεκριμένα, εφαρμόζουμε την ίδια αρχιτεκτονική NNAA που προτάθηκε από τον Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999], αλλά χρησιμοποιούμε μια τροποποιημένη διαδικασία εκμάθησης του δικτύου. Στόχος μας είναι να προσδιορίσουμε την συμβολή τοπικής και συνολικής απομακρυσμένης πληροφορίας σε μια πρωτεϊνική ακολουθία για την τοπική αναδίπλωση μιας πρωτεΐνης. Αυτό επιτυγχάνεται μεταβάλλοντας το μήκος του τμήματος της πρωτεϊνικής ακολουθίας που χρησιμοποιείται σαν είσοδος στο ΝΔΑΑ. Τα αποτελέσματα μας για ένα και μόνο NNAA είναι βελτιωμένα σε σχέση με τα αποτελέσματα του Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999] κατά τρεις ποσοστιαίες μονάδες Q3 (ακρίβεια ανά κατάλοιπο). Είναι επίσης συγκρίσιμα με τα αποτελέσματα των μοντέλων συνόλων ΝΔΑΑ που εμφανίζονται στις εργασίες των Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999], Pollastri et al. [Proteins, vol. 47, pp. 228-235, 2002], Cheng et al. [Nucleic Acids Research, vol. 33, pp. W72-W76, 2005] και Magnan & Baldi [Bioinformatics, vol. 30, pp. 2592-2597, 2014]. Επιπλέον, χρησιμοποιήσαμε ΣΜΜΜ με 6 ΝΔΑΑ για περαιτέρω βελτίωση των αποτελεσμάτων μας. Τα αποτελέσματα της μεθόδου μας βελτιώνονται ακόμα περισσότερο όταν η έξοδος των δικτύων φιλτράρεται με μια μέθοδο μετά-επεξεργασίας δεδομένων. Η μέθοδος αυτή είναι βασισμένη στον αλγόριθμο Hidden Markov Model. Η μέθοδος φιλτραρίσματος στο πρόβλημα ΠΔΔΠ στοχεύει στην παροχή

φυσικοχημικά ρεαλιστικών αποτελεσμάτων, ενώ συνήθως βελτιώνει την προβλεπτική απόδοση μιας μεθόδου Μηχανικής Μάθησης. Με βάση αυτή την πληροφορία, πραγματοποιήσαμε μια συγκριτική μελέτη σχετικά με διάφορες μεθόδους φιλτραρίσματος για το πρόβλημα της ΠΔΔΠ. Συγκεκριμένα, χρησιμοποιώντας τόσο τεχνικές Μηχανικής Μάθησης όσο και Εμπειρικούς Κανόνες για να φιλτράρουμε τα αποτελέσματα της μεθόδου μας, διαπιστώσαμε ότι μέσα από αυτή την συγκριτική μελέτη διαπιστώσαμε ότι ο συνδυασμός των δύο μεθόδων οδηγεί σε περαιτέρω βελτίωση των αποτελεσμάτων.

Μετά από αυτή την αρχική μελέτη του προβλήματος της ΠΔΔΠ, προχωρούμε στο βασικό μέρος αυτής της διατριβής όπου ασχολούμαστε με την μελέτη σχεδιασμού και εφαρμογής αλγορίθμων μάθησης για τις αρχιτεκτονικές ΝΔΑΑ οι οποίοι είναι βασισμένοι στην πληροφορία της Δεύτερης Παραγώγου (ΔΠ). Συγκεκριμένα, παρουσιάζουμε μια μέθοδο όπου εφαρμόζεται ο αλγόριθμος Scaled Conjugate Gradient (SCG) για πρώτη φορά σε αυτή την αρχιτεκτονική. Ειδικότερα, παρουσιάζουμε την ανάπτυξη και εφαρμογή μιας δικής μας παραλλαγής του SCG, του Hybrid Rectified Scaled Conjugate Gradient (HR-SCG) για τις αρχιτεκτονικές ΝΔΑΑ. Σε αντίθεση με την συμβατική μέθοδο κατάβασης κλίσης, η οποία χρησιμοποιείται για την εκπαίδευση ΝΔ, ο HR-SCG αξιοποιεί τόσο την κλίση όσο και την καμπυλότητα μια συνάρτησης σε ένα συγκεκριμένο σημείο για να πετύχει την σύγκλιση του αλγορίθμου. Το μοντέλο έχει δοκιμαστεί στο πρόβλημα ΠΔΔΠ και έχει επιτύχει 77,6 % Q3 ακρίβεια ανά κατάλοιπο σε συγκεκριμένο σύνολο δεδομένων. Επιπλέον, έχει αποδειχθεί ότι ο HR-SCG ξεπερνά τον αλγόριθμο μάθησης κατάβασης κλίσης για τα ΝΔΑΑ όσον αφορά το χρόνο σύγκλισης. Συγκεκριμένα χρειάζεται περίπου 75% λιγότερο χρόνο σύγκλισης στο πρόβλημα ΠΔΔΠ. Ως εκ τούτου, ο αλγόριθμος είναι αποδοτικός και αποτελεσματικός

στην εκπαίδευση των πολύπλοκων αρχιτεκτονικών ΝΔΑΑ. Αυτό είναι πολύ σημαντικό στις περιπτώσεις που έχουμε μεγάλα σύνολα δεδομένων ή πολλά παράλληλα ΝΔΑΑ σε ΣΜΜΜ. Επιπλέον, παρουσιάζουμε αποτελέσματα από μια παρόμοια μελέτη όπου ένας άλλος αλγόριθμος μάθησης ΔΠ έχει εφαρμοστεί στην ίδια αρχιτεκτονική. Πιο συγκεκριμένα, ο αλγόριθμος Hessian Free Optimization χρησιμοποιήθηκε για την εκπαίδευση της αρχιτεκτονικής ΝΔΑΑ για το ίδιο πρόβλημα αλλά έχει επιτύχει περίπου 1% χαμηλότερη Q3 ακρίβεια από τον αλγόριθμο HR-SCG.

Επιπλέον, οι τελευταίες εξελίξεις και τα αποτελέσματα στο πεδίο των ΝΔ (δηλ., Βαθιά Μάθηση, Reservoir Computing κλπ.) έχουν δείξει τεράστια βελτίωση σε διάφορα προβλήματα που σχετίζονται με κατηγοριοποιήσεις ή προβλέψεις δεδομένων από ακολουθίες. Παρόλα αυτά, σε πολλά από αυτά τα μοντέλα ΝΔ, οι παραλλαγές τους, καθώς και συγκεκριμένοι αλγόριθμοι μάθησης δεν έχουν χρησιμοποιηθεί ποτέ στο πρόβλημα ΠΔΔΠ. Πιο συγκεκριμένα, υπάρχει ανοιχτό πεδίο μελέτης το οποίο πρέπει να διερευνηθεί σχετικά με το πρόβλημα ΠΔΔΠ και τις μεθόδους Clockwork ΝΔΑ, Συνελκτικά Νευρωνικά Δίκτυα (ΣΝΔ) και Reservoir Computing. Αν και κάθε μια από αυτές τις μεθόδους έχει τα πλεονεκτήματα και τα μειονεκτήματα της, κάθε μία μέθοδος έχει ειδικά χαρακτηριστικά που θα μπορούσαν να βελτιώσουν την ακρίβεια των αποτελεσμάτων, τον χρόνο σύγκλισης των αλγορίθμων μάθησης, την αναγνώριση βραχυπρόθεσμων και μακροπρόθεσμων εξαρτήσεων στα διαδοχικά δεδομένα, κλπ. Επομένως σε αυτή τη διατριβή, παρουσιάζουμε αποτελέσματα μιας νέας μεθόδου όπου δημιουργούμε μια καινοτόμο απεικόνιση των πρωτεϊνικών δεδομένων σε μορφή εικόνας που αναγνωρίζεται από τα ΣΝΔ, αποτελέσματα του ίδιου προβλήματος χρησιμοποιώντας Clockwork ΝΔΑ, μια νέα αρχιτεκτονική ΝΔΑ αμφίδρομων Echo State Networks και αποτελέσματα του ίδιου προβλήματος σε μοντέλα Long-Short Term Memory. Τέλος,

παρουσιάζουμε αποτελέσματα μιας νέας προσέγγισης, όπου απλά ΝΔ εκπαιδεύονται με τον αλγόριθμο βελτιστοποίησης Hessian Free Optimization για το πρόβλημα ΠΔΔΠ. Οι μεθοδολογίες που βασίζονται στα ΣΝΔ και τα απλά ΝΔ εμπρόσθιας τροφοδότησης τα οποία εκπαιδεύονται με τον Hessian Free Optimization έχουν επιτύχει ακρίβεια πρόβλεψης 80.4% Q3 ανά κατάλοιπο η οποία αποτελεί ένα από τα υψηλότερα αποτελέσματα που αναφέρονται σε αυτή τη διατριβή.

Συνολικά, αυτή η διατριβή συμβάλλει στη βελτίωση σχεδιασμού και εκπαίδευσης μοντέλων ΝΔ τα οποία επεξεργάζονται δεδομένα από ακολουθίες όπου επιπλέον από την πληροφορία που προηγείται ενός σημείου μιας ακολουθίας, η πληροφορία που έπεται αυτού του σημείου είναι επίσης σημαντική για την διεκπεραίωση μιας πρόβλεψης. Σε όλους τους τύπους των υπό μελέτη μοντέλων ΝΔ, παρουσιάζουμε μεθοδολογίες, αρχιτεκτονικές και αλγόριθμους που έχουν πλεονεκτήματα έναντι των γνωστών μεθόδων όσον αφορά την ακρίβεια των αποτελεσμάτων ή την ποιότητα των αποτελεσμάτων ή το χρόνο σύγκλισης των αλγορίθμων στο συγκεκριμένο πρόβλημα της ΠΔΔΠ.

LEARNING SCHEMES FOR EFFICIENTLY TRAINING NEURAL NETWORKS ON PROTEIN SECONDARY STRUCTURE PREDICTION

Michalis Agathocleous

University of Cyprus, 2019

The field of deriving non-linear functions to classify sequential data is currently dealt with by a variety of Machine Learning methods. The procedure of learning sequential data becomes even more challenging when the upstream and downstream information of a sequence is also useful for the prediction or classification. More precisely, in this class of problems, a specific location of sequential data is processed based on the information from left to right before that point (the upstream information) and the information from right to left after that point (the downstream information). The Bidirectional Recurrent Neural Network (BRNN) architecture has been designed by Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999] to deal specifically with this class of problems where the upstream and downstream information is needed to process sequential data. This architecture consists of two simple Recurrent Neural Networks (RNNs) which are used to process the upstream and downstream information, respectively. These demanding models need high performance learning algorithms in terms of convergence time and memory usage to be trained effectively and efficiently. In general, training of RNNs is an open topic in Machine Learning because it is usually complex. Furthermore, these models are usually combined in ensembles with other novel Neural Network models such as Long-Short

Term Memory blocks, Echo State Networks, Conceptors, Clockwork RNNs or Convolutional Neural Networks to enhance results. The accuracy rate of these models is not satisfactory for predictive or classification purposes. The complexity of these methods demands low convergence time and memory usage to solve demanding problems.

One main classification problem of sequential data, where the information before and after a specific location of the sequence is important for prediction purposes is the sequence to structure problem of Protein Secondary Structure Prediction (PSSP). A growing number of protein sequences which are composed by amino acids is known. However, there is missing information with respect to their 3D structure, which specifies their function. The Secondary Structure of these protein sequences is an artificial intermediate step between the protein sequence and the 3D structure of a protein, which indicates the local folds of protein sequence amino acids. These local folds are created based on the interactions of amino acids which lie before and after a specific location of the protein. This problem is complex and needs algorithms that can process and learn the characteristics of the sequential data, in order to come up with better results.

In this thesis, we present a variation of the BRNN which was developed by Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999] for tackling the PSSP problem. This architecture is currently considered as one of the optimal computational neural network type architectures for addressing the problem. We implement the same BRNN architecture, but we use a modified training procedure. More specifically, our aim is to identify the effect of the contribution of local versus global information on the PSSP problem, by varying

the length of the segment on which the RNNs operate for each residue position considered. Our results with a single BRNN are better than Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999] by three percentage points (per residue Q3 accuracy) and comparable to ensembles of BRNN models' results which appear in Baldi et al. [Bioinformatics, vol. 15, pp. 937-946, 1999], Pollastri et al. [Proteins, vol. 47, pp. 228-235, 2002], Cheng et al. [Nucleic Acids Research, vol. 33, pp. W72-W76, 2005] and Magnan & Baldi [Bioinformatics, vol. 30, pp. 2592-2597, 2014]. Moreover, we have used ensembles of 6 BRNNs to enhance our results. In addition, our results improve even further when sequence-to-structure output is filtered in a post-processing step, with a novel Hidden Markov Model-based approach. Filtering of protein secondary structure prediction aims to provide physicochemically realistic results, while it usually improves the predictive performance. We performed a comparative study on this challenging problem, utilising both machine learning techniques and empirical rules and we found that combinations of the two lead to the highest improvement.

After these initial investigations, we move to the core part of the thesis, which is the study of second order learning algorithms for the BRNN models. More specifically, we present a second order method for training BRNNs for the PSSP problem where the Scaled Conjugate Gradient (SCG) is applied for the first time on these models. In particular, we present the development and implementation of our variation of the SCG which we call the Hybrid Rectified-Scaled Conjugate Gradient (HR-SCG) learning algorithm

for BRNN architectures. In contrast to the conventional Gradient Descent learning algorithm, the HR-SCG exploits both gradient and curvature information for convergence. The model has been tested on the PSSP problem and achieved 77.6% per residue accuracy on a specific PSSP dataset. Moreover, it has been shown that the HR-SCG outperforms the GD learning algorithm for BRNNs in terms of convergence time, needing approximately 75% less time on PSSP datasets. Hence, the algorithm is efficient for training the complex BRNN architectures, a feature important with big datasets and facilitates fast training of BRNN ensembles. Furthermore, we present results from a similar study where another second order learning algorithm has been applied on the same architecture. More specifically, the Hessian Free optimization algorithm has been used to train the BRNN architecture for the PSSP problem but it has achieved approximately 1% lower accuracy results than the HR-SCG algorithm.

Finally, the latest developments and results in the Neural Networks field (i.e., Deep Learning, Reservoir Computing etc.) have shown huge improvement in several sequential data-related problems. Nevertheless, many of these Neural Network models, their variations and specific learning algorithms have never been used for the PSSP problem. More specifically, there is an open field to be investigated related to the PSSP problem and Clockwork Recurrent Neural Networks, Convolutional Neural Networks and Reservoir Computing methods. Although these techniques have their advantages and disadvantages compared with each other, each one has specific characteristics related to accuracy, execution time, sorting with short and long term dependencies, etc. Hence, we present results

on a novel image-like input representation method for the PSSP problem which is used by Convolutional Neural Networks, results of the same problem with Clockwork Recurrent Neural Networks, with a novel Bidirectional Echo State Network architecture and results of our data with Long-Short Term Memory BRNN methods. Finally, we present results on a novel approach where simple Feed Forward Neural Networks trained with the Hessian Free Optimization algorithm have been used for the PSSP problem showing state of the art results. The methodologies based on Convolutional Neural Networks and Feed Forward Neural Networks trained with the Hessian Free Optimization algorithm have both achieved approximately 80.4% Q3 per residue accuracy which are the highest results reported in this thesis.

Overall, one of the main contributions of this thesis is on the improvement of learning in Neural Network models for sequential data where the upstream and downstream information is important to process a specific location of sequential data. In all types of the studied Neural Network models, we prescribe certain methodologies, architectures and algorithms that have advantages over known methods in terms of accuracy or quality of results or convergence time for the specific PSSP problem.

APPROVAL PAGE


Doctor of Philosophy Dissertation

LEARNING SCHEMES FOR EFFICIENTLY TRAINING NEURAL NETWORKS ON PROTEIN SECONDARY STRUCTURE PREDICTION

Presented by

Michalis Agathocleous

Research Supervisor



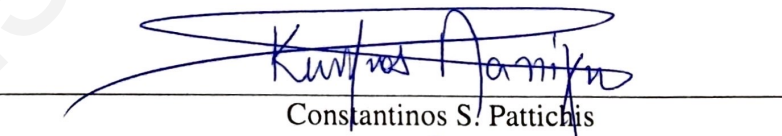
Chris Christodoulou

Committee Member




Christos N. Schizas

Committee Member




Constantinos S. Pattichis

Committee Member



George M. Spyrou

Committee Member



Nicolai Petkov

DECLARATION OF DOCTORAL CANDIDATE

The present doctoral dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Michalis Agathocleous

ACKNOWLEDGEMENTS

At first, I would like to express my appreciation and admirations to my supervisor Dr. Chris Christodoulou. I truly thank him for his strategic guidance and support throughout the years I was working on my PhD research. His knowledge, his expertise and his crucial supervision were the main pillars for the completion of this PhD work. I would also like to thank him because he was always there to help me to overcome any personal difficulties I was facing during my PhD studies. Finally, I am truly thankful to him for introducing me to the field of Neural Networks back in 2008, which since then is my main occupation in the industrial and academic world.

I would also like to thank our main collaborator and very good friend Dr Vassilis Prombonas for his invaluable ideas and targeted comments which were crucial for the development of this thesis. His knowledge in the field of Bioinformatics was decisive for the success of our research ideas.

I am also grateful to my past labmates and current friends Dr Vassilis Vassiliades, Dr Achilleas Koutsou, Dr Margarita Zachariou and Dr Petros Kountouris, for all the good and productive times we had as members of the Computational Intelligence and Neuroscience group. Their stimulating casual discussions on research topics have led to some valuable ideas which have been developed in this PhD work. Furthermore, I would like to thank Andreas Dionyssiou and Konstantinos Charalambous for their ideas, effort and smooth collaboration. In the same scope, I would like to thank all the students (Georgia

Christodoulou, Simos Hajicostas, Irene Petridou, Irene Papakosta, Panayiotis Pavlides, Panagiotis Demetriou, Maria Maslioukova) who have interacted with the field of my PhD work and have helped us in the execution of endless experiments. Finally, I would like to thank my friend Dr Andreas Diavastos for his invaluable support during my PhD studies.

Special thanks to my dissertation committee members, Professors Christos N. Schizas, Constantinos S. Pattichis, George M. Spyrou and Nicolai Petkov, for offering their advice and suggestions for the improvement of this thesis.

I am grateful to the University of Cyprus, the Cyprus State Scholarship Foundation and the A. G. Levendis foundation for funding me through their scholarship programs.

I am deeply thankful to my beloved wife Ellie Demetriou for loving, effectively believing and patiently supporting me all these years. I also thank all my close friends and my family who were understanding and supporting me during my PhD studies.

TABLE OF CONTENTS

List of Abbreviations	1
Abstract	1
Chapter 1: Introduction	1
1.1 Motivation	1
1.2 Problem Statement - Hypothesis	4
1.3 Approach	6
1.4 Outline	9
Chapter 2: Survey: Neural Network algorithms and the PSSP problem	12
2.1 Introduction	12
2.2 Artificial Neural Networks	13
2.2.1 Introduction to Artificial Neural Networks	13
2.2.2 Recurrent Neural Networks	16
2.2.3 Bidirectional Recurrent Neural Networks	20
2.2.4 Clockwork-Recurrent Neural Networks	21
2.2.5 Reservoir Computing	23
2.2.5.1 Echo State Networks	28
2.2.5.2 Other Reservoir Computing Methods	31
2.2.6 Deep Learning and Convolutional Neural Networks	33
2.3 Neural Network Learning Algorithms	38

2.3.1	First and Second Order Learning Algorithms	38
2.3.2	The Scaled Conjugate Gradient algorithm	42
2.3.3	The Hessian Free Optimization algorithm	45
2.3.3.1	Conjugate Gradient for HFO	46
2.3.3.2	Damping	48
2.3.3.3	Gauss-Newton Matrix	50
2.3.3.4	Evaluating the Hessian-Vector Multiplication	51
2.4	Protein Secondary Structure Prediction	55
2.4.1	Introduction	55
2.4.2	Proteins	57
2.4.3	Evaluation Metrics	64
2.4.4	Protein Secondary Structure Prediction	65
2.4.5	Related work	69
2.4.6	Data	73
2.4.6.1	Sequence Databases	75
2.4.6.2	Data Selection Criteria	75
2.4.6.3	Secondary Structure Assignment	76
2.4.6.4	Multiple Sequence Alignment Preprocessing	77
2.4.6.5	The CB513 Dataset	78
2.4.6.6	The PISCES Dataset	79
Chapter 3: BRNN input sliding window and post-processing methods		80
3.1	Introduction	80

3.2	Training BRNN with weight updating for each residue	81
3.2.1	Data Collection and Preprocessing	82
3.2.2	Modified Training Procedure for the BRNN Architecture	83
3.2.3	Results and Discussion	86
3.2.4	Conclusions	90
3.3	Ensemble methods	92
3.3.1	Ensemble of BRNNs	93
3.4	Filtering methods for the PSSP problem	95
3.4.1	Dataset and preprocessing	97
3.4.2	Filtering techniques	98
3.4.3	Prediction accuracy assessment	102
3.4.4	Finding the best local window	103
3.4.5	Results and discussion	104
3.4.5.1	Prediction accuracy per state	105
3.4.5.2	Combining machine learning and empirical techniques	107
3.4.5.3	Combining machine learning techniques	111
3.4.6	Conclusion	112
3.5	Chapter Contribution	113
Chapter 4:	Training BRNNs with Second Order Learning algorithms	115
4.1	Introduction	115
4.2	Training BRNNs with the HR-SCG learning algorithm	116
4.2.1	Introduction	116

4.2.2	Methodology	120
4.2.2.1	The BRNN Architecture	120
4.2.2.2	Development of the HR-SCG Algorithm for BRNNs	124
4.2.2.3	HR-SCG Time Complexity Analysis	134
4.2.2.4	Training BRNNs with the HR-SCG	136
4.2.3	Results and Discussion	136
4.2.3.1	Data Preparation and Simulation Details	137
4.2.3.2	Optimizing BRNN and HR-SCG parameters	139
4.2.3.3	Methodological Enhancements	143
4.2.3.4	Comparison of HR-SCG to BPTT learning algorithm	148
4.2.4	Conclusions	151
4.3	Training BRNN with the HFO learning algorithm	153
4.4	Chapter Contribution	153
Chapter 5:	Efficient and effective methods for the PSSP problem	155
5.1	Introduction	155
5.2	CNNs in Combination with SVMs	156
5.2.1	Introduction	156
5.2.2	Data Representation	157
5.2.3	Support Vector Machines (SVMs)	159
5.2.4	Results and Discussion	160
5.2.4.1	Optimising the Parameters	160
5.2.4.2	10-fold Cross-Validation on CB513	163

5.2.4.3	Ensembles and External Rules Filtering	164
5.2.4.4	Filtering using Support Vector Machines (SVMs)	165
5.2.5	Summary of the Results	165
5.3	Clockwork Recurrent Neural Networks	168
5.3.1	Methodology	170
5.3.1.1	Input Data	170
5.3.1.2	Architecture and implementation	171
5.3.2	Results and Discussion	174
5.3.2.1	Examination of the parameters	174
5.3.2.2	10-fold cross-validation on CB513	180
5.4	Bidirectional ESN	186
5.5	Simple FFNNs trained with HFO learning algorithm	188
5.5.1	Methodology	191
5.5.1.1	Data Representation	191
5.5.1.2	The HFO learning algorithm	192
5.5.2	The proposed Methodology	195
5.5.3	Results and Discussion	196
5.5.3.1	Data Preparation and Simulation Details	198
5.5.3.2	Cross-Validation simulations	199
5.5.3.3	Comparison of our methodology to other methods	202
5.5.4	Conclusion	206
5.6	Long Sort-Term Memory BRNN (LSTM-BRNN)	209

5.7 Chapter Contribution	210
Chapter 6: General Discussion and Conclusions	212
6.1 Overview of the problems	212
6.2 Overview of the approaches, results and conclusions	215
6.3 Presented approaches to other problems	227
6.4 Contributions	230
6.5 Dissemination of PhD work	232
Chapter 7: Future work	238
Publications so far from this thesis	244
References	244
References	245
Appendix A: Publications during PhD work	280

LIST OF TABLES

- 1 Prediction accuracy results with a BRNN architecture with two hidden layers for different sizes of the local context window L_s . MSA profiles were used as input (for all parameter values, see text). 88
- 2 Prediction accuracy results with a BRNN architecture with one hidden layer, randomized input, WTA output encoding and MSA profiles as input. The first four rows correspond to a BRNN architecture with a FFNN of 15 hidden neurons and RNNs of 17 hidden neurons and the last four rows to a BRNN architecture with a FFNN of 51 hidden neurons and RNNs of 41 hidden neurons. The rightmost columns correspond to the performance metrics after filtering with an HMM and a feed forward ANN. 89
- 3 Confusion matrices, showing the distribution of predictions for the three secondary structure states, for the best performing configurations: (a) for the best configuration listed in *Table 1* giving a Q3 of 73.92%, (b) best results of the unfiltered configuration shown in *Table 2* giving a Q3 of 76.07%, (c) best results of the configuration shown in *Table 2* filtered with a HMM giving a Q3 of 76.57%, and (d) best results of the configuration shown in *Table 2* filtered with an ANN giving a Q3 of 76.60%. Note that the displayed values correspond to the fraction of predicted residues with a given observed state, expressed as a percentage. 89

4	Filtering PSSP on the CB513 dataset using the method shown in the first column, sorted by the highest predictive accuracy (Q_3). w is the local window size for filtering that maximises the SEL score for each method. In bold are the highest scores in the corresponding column.	100
5	Filtering PSSP on the PDB-Select25 dataset using the method shown in the first column, sorted by the highest predictive accuracy (Q_3). In bold are the highest scores in the corresponding column.	102
6	Filtering PSSP using combinations of machine learning algorithms and empirical rules. Firstly, a machine learning algorithm is employed for filtering (shown in the first column) and, subsequently, the output is filtered by empirical rules (SS-filt or WH-filt) to further improve PSSP. In bold are the highest Q_3 , SOV and SEL scores.	104
7	Filtering PSSP using combinations of machine learning algorithms with different voting schemes on the CB513 dataset. The last three columns show the results after using the SS-filt empirical rules. In bold are the highest scores in the corresponding column.	110
8	Protein sequences which have been excluded from the CB513 and PISCES dataset. These protein sequences have been excluded from the datasets because of miscalculations of the PSI-BLAST.	138
9	Experimental results using the CB513 dataset, where FP is the network's Free Parameters (see text for description of the parameters)	139

10	Experimental results using the CB513 (PISCES) dataset under 10-(6-) fold cross-validation, where S is a single BRNN trained with SCG algorithm, H is the Hybrid method, D is the Dropout algorithm, E is the Ensemble method and F is the Filtering Technique (see text for description of the parameters)	143
11	Statistics of Q3 results for 10 different training procedures of the HR-SCG algorithm with and without Dropout method.	143
12	Results of an ensemble of six BRNNs combined with an SVM for filtering purposes: 10-fold cross-validation approach on the CB513 dataset.	154
13	Summary of the Results for All Methods.	164
14	Summary of the Results for All Methods.	166
15	CNN Ensembles and SVM Filtering: Q3 and SOV Results for each Fold.	167
16	CNN Ensembles and SVM Filtering: Statistical Analysis	167
17	The sets of clock periods which were used to determine the optimal one. .	175
18	Q3 and SOV accuracy scores for each fold after the application of ensembles.	181
19	Results after ensembles: statistical analysis	181
20	Q3 and SOV accuracy scores for each fold after applying external empirical rules on them.	183
21	Results after external empirical rules: statistical analysis	183
22	Q3 and SOV accuracy scores for each fold after applying SVM filtering on them.	184

23	Experimental results for a single FFNN: Q3 and SOV Results for each Fold of PISCES dataset.	197
24	Experimental results for an ensemble of 5 FFNNs trained with HFO: Q3 and SOV Results for each Fold of PISCES dataset.	197
25	Experimental results for an ensemble of 5 FFNNs trained with HFO and filtered with a SVM: Q3 and SOV Results for each Fold of PISCES dataset.	197
26	FFNN Ensembles and SVM Filtering: Statistical Analysis of Q3 accuracy and SOV score for fold 0-4 results presented in <i>Table 25</i>	199
27	Results of the LSTM-BRNN methods: 5-fold cross-validation approach for the PISCES dataset.	210

LIST OF FIGURES

1	On the left the McCulloch and Pitts model and the right a typical MLP artificial neural network	14
2	Typical dynamic neural network architectures [Peng and Magoulas, 2008](See text for more information)	19
3	The BRNN architecture: Left (forward) and right (backward) context (F_t and B_t) are implemented by recurrent networks , where q^{-1} is the back-shift operator and q^{+1} is the forward-shift operator as explained in Baldi et al. [1999]. The input layer is fed with W_a , W_f , W_b and W_c sliding windows (see text for description of the parameters). The output layer O_t has three softmax units associated with membership in each of the three secondary structure classes for the current residue (position t). Functions $\phi()$, $\beta()$, and $\eta()$ are implemented by feed-forward neural networks [Baldi et al., 1999].	22
4	The CW-RNN architecture is similar to a simple RNN with an input, output and hidden layer. The hidden layer is partitioned into g modules each with its own clock rate. Within each module the neurons are fully interconnected. Neurons in faster modules i are connected to neurons in a slower modules j only if the clock period $T_i < T_j$. This figure is reprinted from Koutnik et al. [2014].	24
5	The ESN network architecture [Ozturk et al., 2007]	30

6	The ESN network architecture as it was firstly designed by [Jaeger, 2001]	31
7	The preconditioned CG , where $x = \delta$, B is the curvature matrix, $A = B_t 1$, $b = \nabla f(w_t 1)$ and P is the preconditioning matrix [Martens and Sutskever, 2011].	48
8	An algorithm for computing the gradient of a FFNN, where $L(y_l; t_l)$ is one of the loss functions of Figure 11 [Martens and Sutskever, 2011] . . .	53
9	An algorithm for computing the $H(w)v$ product in a FFNN, where $L(y_l; t_l)$ is one of the loss functions of Figure 11 [Martens and Sutskever, 2011] . . .	53
10	An algorithm for computing the $G(w)v$ product in a FFNN, where $L(y_l; t_l)$ is one of the loss functions of Figure 11 [Martens and Sutskever, 2011] . . .	54
11	Typical loss functions with their derivatives and HMs [Martens and Sutskever, 2011]	54
12	List of all the 20 essential amino acids [Abo-Elkhier, 2012].	59
13	The Central Dogma of Molecular Biology: DNA makes RNA makes proteins (Nucleic Acids Book, www.atdbio.com, 2018, May 5)	60
14	Example of the central dogma. The first few amino acids for the alpha subunit of hemoglobin [Bolsover et al., 2004].	61
15	The amino acids specified by each codon [Alkatib et al., 2012].	62
16	Layers of protein structure [Gupta et al., 2017].	63
17	Number of publications for PSSP per year [Yang et al., 2016]	70
18	Matrix with the abbreviations of the secondary structures grouped in 8 and 3 classes	79

19	<p>Modified Training Procedure for the BRNN Architecture, where $\gamma \in (0, 1]$ is a modified shift operator, which in effect adds a constant weight based on the importance given to the outputs of the FRNN and BWRNN. Intuitively, we chose $\gamma < 1$ (thus $\gamma - 1 > 1$) to reflect the fact that protein chains are synthesised from the N-terminal to the C-terminal (i.e., from the left to the right side of the sequence respectively) with some secondary structural elements forming co-translationally [Baram and Yonath, 2005].</p>	85
20	<p>The architecture of the ensemble of BRNNs, followed by the filtering of the output. The Position Specific Scoring Matrices (PSSMs) values are given as input to six BRNNs, which predict the secondary structure of each residue in the amino acid sequence. Subsequently, the outputs are averaged and are given as input to the filtering methods investigated in this study.</p>	94
21	<p>Ensemble methods: Results from ensembles of six BRNNs, trained with the 10-fold cross-validation method on the CB513 dataset.</p>	95
22	<p>Experiments with different local window sizes for filtering PSSP (see text for more information) using four machine learning algorithms on the CB513 dataset. The predictive accuracy (left) and the SOV score (right) strongly depend on the size of the local window used for filtering.</p>	99

23	Per state prediction after the application of filtering techniques on the PDB-Select25 dataset. The y -axis corresponds to the number of residues and the x -axis to the combinations of observed and predicted state. For instance, HE corresponds to residues that are observed as helices (H) but are predicted as extended (E). The three columns at each state show the number of residues for the unfiltered classifier (ensemble of BRNNs), the LibSVM filtering and the combination of LibSVM and SS-filt, respectively.	107
24	Five examples that show the effect of filtering on PSSP. The first line in each case shows the PDB ID and the Chain ID. Sequences A and B are taken from CB513 and the remaining sequences from PDB-Select25. The mispredictions are shown in shadow. "PriStr" is the amino acid sequence; "Real SS" is the observed secondary structure; "No-Filt" is the PSSP from the ensemble of BRNNs; "LibSVM" is the PSSP filtered with LibSVM and "SS-Filt" is the application of the SS-Filt empirical rules on the output of LibSVM filtering. Secondary structure states are reported using the reduced three-state scheme (see <i>Section 3.4.1</i>).	108
25	Unfolded architecture of the BRNN: The unfolded architecture of the BRNN is inspired from the work of Baldi et al. [1999]. The time step t represents the relative positions of patterns and is decided by a predefined-length sliding window. While this window is sliding, the corresponding I/O pair and the states of the current time step are determined.	126

26	The HR-SCG BRNN methodology for the PSSP problem:	The methodology has 5 levels of processing. Level 1 is used for Protein Data Pre-processing to feed each one of the BRNNs in Level 2. Level 3 and Level 4 use Ensemble and Filtering Methods, respectively, to present a final output in Level 5.	137
27	Experimental results using the CB513 dataset as a test set for (a) W_a , (b) W_{fb} , (c) W_c , (d) S_{css} , (e) S_{fb} and (f) S_{out} . The six bar charts show the impact of most important BRNN parameters on the final results. A single BRNN has been trained at each time with HR-SCG algorithm on the CB513 dataset. Multiple experiments have been carried out to tune up BRNN parameters to maximize results (see text for description of the parameters and results).		140
28	The Dropout impact in HR-SCG algorithm:	Average results of 10 different training procedures of the HR-SCG algorithm. Each bar shows the minimum and maximum Q3 value of the 10 different training procedures. The dropout technique does not improve the system's accuracy but it reduces the variance of the final Q3 results from multiple experiments of the CB513 and PISCES datasets.	144
29	The HR-SCG BRNN methodology improvements:	The improvements and ML techniques which were applied to HR-SCG methodology have increased the BRNN accuracy by approximately 4%.	144

30	<p>Graph of Confusion Matrix for Actual vs Predicted SS Classes on the CB513 dataset: HH, EE and LL are the True Positive scores of our method for each class after the average ensemble method and filtering technique. EH, LH, HE, LE, HL and EL are the scores for the mispredicted classes where the first letter is the actual class and the second letter is the predicted class. Based on an average of a 10-fold cross validation evaluation, the method can predict correctly with 76,80%, 66.32% and 83.84% the H, E and L classes, respectively.</p>	145
31	<p>Comparison of HR-SCG to BPTT learning algorithm: On 9(a), the HR-SCG learning algorithm needs much less training iterations than the conventional BPTT learning algorithm. On 9(b) the testing error through multiple restarts of the algorithm.</p>	145
32	<p>Example of Data Representation Method: An example of data representation of an input sample using a window size of 15 amino acids. Each line represents the MSA profile vector for the specific amino acid. The SS label for the example input sample showed in this figure, is the SS label for the middle amino acid.</p>	159
33	<p>Optimizers: CNNs Q3 accuracy results using different Gradient Descent (GD) optimization algorithms.</p>	162
34	<p>Window Size: CNNs Q3 accuracy results with different window (W) sizes.</p>	163

35	Confusion Matrix: Predictions and mispredictions of the secondary structure classes H, E and C/L after applying ensembles on each fold using CB513 dataset. Q3 accuracy scores are shown for each class.	167
36	Addition of dummy values: The PSSM profiles is padded with dummy values before and after the sequence termini. $W_a = 11$; X: dummy value. .	169
37	Q3 and SOV accuracy scores for each clock period set shown in <i>Table 17</i> . The red bar corresponds to Q3 and the green bar to SOV.	175
38	Q3 and SOV accuracy scores for each optimizer. The red bar corresponds to Q3 and the green bar to SOV.	178
39	Q3 and SOV accuracy scores for each activation function. The red bar corresponds to Q3 and the green bar to SOV.	179
40	Q3 and SOV accuracy scores for different number of neurons in the hidden layer. The red bar corresponds to Q3 and the green bar to SOV. . .	180
41	Predictions and mispredictions of the secondary structure classes H, E and L after applying ensembles on each fold. The red bar corresponds to Q3 and the green bar to SOV.	182
42	Predictions and mispredictions of the secondary structure classes H, E and L after applying external empirical rules on each fold. The red bar corresponds to Q3 and the green bar to SOV.	183
43	Relationship between the execution time and the number of neurons on the hidden layer of the CW-RNN.	185
44	The architecture of a BESN as described in this study.	186

45	Data Representation: Each amino acid is within a window (W) centered at the residue of interest, encoded by a 20-dimensional MSA profile vector (see text for more details).	191
46	Our proposed methodology for the PSSP problem (see text for more details).	195
47	Graph of Confusion Matrix for Actual vs Predicted SS Classes: HH, EE and LL are the True Positive scores of our method for each class after the average ensemble method and SVM filtering technique. EH, LH, HE, LE, HL and EL are the scores for the mispredicted classes where the first letter is the actual class and the second letter is the predicted class. Based on an average of a 5-fold cross validation evaluation, the method can predict correctly with 84.41%, 70.60% and 83.61% the H, E and L classes, respectively.	200
48	Q3 accuracy and SOV score for each FFNN optimizer. The red bar corresponds to Q_3 accuracy and the yellow bar to SOV score. The SOV score has been multiplied by 100 for presentation purposes (see text for more details).	202
49	Length of training time of the FFNN Optimisers: The length of training time in minutes is calculated based on the average 5-fold cross-validation training time of the 5 FFNNs used in each ensemble. The label of each bar corresponds to 'number of iterations / average iteration execution time (min)' (see text for more details).	203

50 **Q3 accuracy and SOV score** for ANN models used for the PSSP problem. The red bar corresponds to Q_3 accuracy and the yellow bar to SOV score. The SOV score has been multiplied by 100 for presentation purposes (see text for more details). 206

Michalis Agathocleous

LIST OF ABBREVIATIONS

Adam Adaptive Moment Estimation

ANN Artificial Neural Network

BESN Bidirectional Echo State Network

BPDC BackPropagation-Decorrelation

BPTT Backpropagation Through Time

BRNN Bidirectional Recurrent Neural Network

BwRNN Backward Recurrent Neural Network

CG Conjugate Gradient

CNN Convolutional Neural Network

CW-RNN Clockwork-Recurrent Neural Network

DSSP Dictionary for Secondary Structure of Proteins

EKF Extended Kalman Filters

ESN Echo State Network

FFNN Feed-Forward Neural Network

FFTD Feed-Forward Time-Delayed

FRNN Forward Recurrent Neural Network

GD Gradient Descent

HFO Hessian Free Optimization

HMM Hidden Markov Model

HR-SCG Hybrid Rectified-Scaled Conjugate Gradient

iProClass Protein Information Resource

kNN k-Nearest Neighbour

LRN Layered Recurrent Network

LSM Liquid State Machine

LSTM Long-Short Term Memory

ML Machine Learning

MLP Multilayer Perceptron

MSA Multiple Sequence Alignment

NARX Nonlinear Autoregressive Network with Exogenous Inputs

NBPW Naïve Bayesian Parzen Window

NN Neural Networks

PCA Principal Components Analysis

PCG Preconditioned Conjugate Gradient

PDB Protein Data Bank

PDBe Protein Data bank in Europe

PDBj Protein Data bank in Japan

PSR Phoneme Speech Recognition

PSSM Position Specific Scoring Matrices

PSSP Protein Secondary Structure Prediction

QN Quasi-Newton

RBF Radial Basis Function

RC Reservoir Computing

RDTs Recurrent Decision Trees

RepL Representation Learning

RNN Recurrent Neural Network

SCG Scaled Conjugate Gradient

SOV Segment Overlap Score

SVM Support Vector Machine

TDNN Time Delay Neural Network

TMPTP Transmembrane Protein Topology Prediction

WTA Winner-Takes-All

Michalis Agathocleous

LIST OF SYMBOLS

x_i ANN input vector values at position i

t Discrete time index in a protein chain

w Local window size on protein chain

W_a BRNN sliding window

F_t BRNN forward (upstream) context

B_t BRNN backward (downstream) context

q^{-1} BRNN back-shift operator

q^{+1} BRNN forward-shift operator

U BRNN input nodes

$\phi()$ BRNN forward RNN

$\beta()$ BRNN backward RNN

$\eta()$ BRNN forward NN

F BRNN forward states

B BRNN backward states

hF $\phi()$ hidden states

hB $\beta()$ hidden states

hU $\eta()$ hidden states

oF $\phi()$ output states

oB $\beta()$ output states

oU $\eta()$ output states

O_t BRNN output vector

W_a BRNN input sliding window

W_f $\phi()$ input sliding window

W_b $\beta()$ input sliding window

W_c $\eta()$ input sliding window

N_ϕ Numbers of hidden nodes in ϕ

N_n Numbers of hidden nodes in $\eta()$

N_β Numbers of hidden nodes in $\beta()$

N_{uf} Dimensionality of the input vector to hF

N_{uc} Dimensionality of the input vector to hU

N_{ub} Dimensionality of the input vector to hB

$N_{\phi'}$ Number of nodes in boxes oF

N_{β} Number of nodes in boxes oB

N_{hU} Dimensions of hU layer

$w_{F_{ij}}$ Connection weights between hF and oF

$w_{B_{ij}}$ Connection weights between hB and oB

$w_{U_{ij}}$ Connection weights between hU and oU

$f_{i,t}$ Outputs of each neuron at time t of oF

$b_{i,t}$ Outputs of each neuron at time t of oB

$hU_{i,t}$ Outputs of each neuron at time t of oU

$\alpha()$ BRNN ReLU activation function

Ψ Number of training patterns

s Number of neurons BRNN output layer

$\bar{y}_{p,i,t}$ BRNN target output

$y_{p,i,t}$ BRNN system output

T Moving amount of the sliding window size

M Number of GD's iterations

W_{fb} Chapter 4: $\phi()$ and $\beta()$ input sliding window

S_{cSS} Chapter 4: Adaptive step size scaling parameter

n Chapter 4: Number of F and B states

h_n Chapter 4: $\phi()$ and $\beta()$ hidden states

S_{fb} Chapter 4: Number of addition consecutive context vectors in $\phi()$ and $\beta()$

S_{out} Chapter 4: Number of addition consecutive context vectors in BRNN O_t

h_n Chapter 4: Number of hidden units in $\eta()$

h_{fb} Chapter 4: Number of hidden units in $\phi()$ and $\beta()$

s_{in} Chapter 4: Number of input states to S_{fb}

s_{out} Chapter 4: Number of input states to S_{out}

a Chapter 5: Leaking rate

$f()$ Chapter 5: Activation function

$u()$ Chapter 5: Reservoir input value

x Chapter 5: Reservoir echos

Y_{target} Chapter 5: Reservoir desired outputs

β Chapter 5: Reservoir regularization coefficient

I Chapter 5: Identity matrix

H Chapter 5: Hessian matrix

Chapter 1

Introduction

1.1 Motivation

The field of deriving non-linear functions to classify sequential data is currently dealt with by a variety of Machine Learning methods. Such methods include Recurrent Neural Networks (RNNs) [Elman, 1990], Recurrent Decision Trees (RDTs) and Hidden Markov Models [Dietterich, 2002]. The procedure of learning sequential data becomes even more challenging when the upstream and downstream information of a sequence is also useful for the prediction or classification. More precisely, in this class of problems, a specific location of sequential data is processed based on the information from left to right before that point (the upstream information) and the the information from right to left after that point (the downstream information). One main classification problem of sequential data, where the information before and after a specific location of the sequence is important for

prediction purposes is the sequence to structure problem of Protein Secondary Structure Prediction (PSSP) [Baldi et al., 1999]. This problem is complex and needs algorithms that can process and learn the characteristics of the sequential data, in order to come up with better results.

The most effective models for this kind of data are RNNs and more specifically the BRNNs [Baldi et al., 1999; Schuster and Paliwal, 1997]. BRNNs are based on RNNs' architecture theory which efficiently incorporates temporal dynamics [Elman, 1990]. In general, training of RNNs is an open topic because it is usually complex. The most important training algorithms for RNNs are the Real Time Recurrent Learning (RTRL) [Ronald and Zipser, 1989], the Backpropagation Through Time (BPTT) [Werbos, 1990], the Conjugate Gradient (CG) [Charalambous, 1992], Hessian Free Optimization (HFO) [Martens, 2010; Martens and Sutskever, 2011] and the Extended Kalman Filters (EKF) [Williams, 1992; Wang and Huang, 2011]. However, only the BPTT algorithm has been modified and developed to train BRNN architectures.

RNNs are usually combined with other methods and algorithms to improve the system's accuracy. Such methods include ensemble [Dietterich, 2000; Zhou et al., 2002, 2010; Li et al., 2018a; Zheng et al., 2019] and filtering techniques [Kountouris et al., 2012], Long-Short Term Memory (LSTM) blocks [Graves and Schmidhuber, 2005; Hochreiter and Schmidhuber, 1997], Echo State Networks (ESNs) [Jaeger, 2001, 2007; Lukoševičius and Jaeger, 2009], Conceptors [Jaeger, 2014] and Clockwork RNNs (CW-RNN) [Koutnik et al., 2014] as architectural improvements of RNNs. Furthermore, Convolutional Neural Networks (CNNs) [Krizhevsky et al., 2012] have shown superior results in

various sequential problems [Schmidhuber, 2015; Srinivas et al., 2016; Rawat and Wang, 2017]. These methods are new established fields in the Machine Learning (ML) field which can build high quality feature extraction procedures and achieve high-performance classification of sequential data.

The PSSP problem [Agathocleous et al., 2010, 2016; Baldi et al., 1999; Kountouris et al., 2012] is an important problem in the biological sciences. The primary structure of a growing number of proteins is known. However, there is missing information with respect to their 3D structure, which specifies their function. Besides being important for a basic understanding of life itself, knowledge of protein structure may assist in research against disease, providing the means for improving the general quality of life.

The motivation of this research work is to develop novel optimization algorithms and modeling methods that can make predictions on sequential data where important information for a specific time-step of the sequence is located upstream and downstream, and more specifically for the PSSP problem. It is important to note that PSSP is a sequence to structure problem which means that by time-step we refer to a specific data point of a protein input sequence and the ‘prediction’ is actually a classification of the protein SS, which is spatial information rather than sequential. In other words, in the PSSP problem we classify the SS of the proteins (rather than ‘predict’ the future in any way), based on the protein sequence. This research has its highest impact in the field of machine learning in general by allowing cross-fertilisation of ideas between the research areas of

analytical learning optimisation algorithms and effective implementation of practical applications for the PSSP problem, which are based on human-technology interaction and in the longer term, improve people's quality of life.

1.2 Problem Statement - Hypothesis

The procedure of learning sequential data with BRNNs is challenging because of the complexity of the data and the short and long range dependencies which can be found in the sequences. The most common algorithm which has been modified and developed to train the BRNN architectures is the BPTT. The disadvantage of this algorithm is that it is based on the computation of a gradient vector of the output error measure with respect to the network weights, which is a time consuming and complex process. In contrast, the Scaled Conjugate Gradient (SCG) algorithm, which has not been developed for the BRNN architecture, has some advantages over the gradient descent algorithms. The SCG algorithm is a second-order algorithm, which although is a gradient descent-based method, it has been found to be superior to the conventional BPTT algorithm in terms of accuracy, convergence rate and the vanishing-gradient problem in simpler RNN architectures [Hochreiter and Schmidhuber, 1997]. Moreover, the Hessian Free Optimization algorithm (HFO) [Martens, 2010; Martens and Sutskever, 2011] is a second order algorithm which, further to the SGC advantages, can work very well for fast optimizing non-convex functions such as the training objective for simple FFNNs and RNNs [Martens, 2010]. Consequently, these algorithms alleviate most of the crucial issues that a training

algorithm for sequential data must overcome [Dietterich, 2002]. Our initial focus is on applying these learning algorithms on the BRNN architecture:

1. How can we develop and implement SCG for the BRNN architecture?
2. How can we do the same for the HFO learning algorithm?
3. Which are the results of these methods on the PSSP problem?
4. Which are the advantages and disadvantages compared to the BPTT learning algorithm?

More challenges arise when studying RNNs and other NN models for the PSSP problem, which raise more questions to be answered:

1. Which is the difference in the results if we use global or local information to train BRNNs?
2. How can we improve our results with ensemble methods?
3. Which is the best machine learning filtering method for the PSSP problem?
4. Is it possible to use filtering rules to improve the PSSP results?
5. Which combination between machine learning filtering methods and filtering rules can show the best results?
6. How can we use ESN for the PSSP problem?
7. How can we use simple Feed-forward Neural Networks trained with SCG or HFO learning algorithms for the PSSP problem?

8. How can we use CW-RNN for the PSSP problem?
9. How can we encode protein data to be classified by CNNs?
10. Can we combine these methods with LSTM-BRNNs to improve results?

Research has shown that some second order learning algorithms can be very efficient and effective for training RNN models to increase the performance for several problems [Chang and Mak, 1999; Martens and Sutskever, 2011]. Furthermore, the combination of these methods with ensemble and filtering methods, algorithm customizations and other novel neural network models may improve the results in terms of convergence time and memory usage. Therefore, how efficient and effective can these methods be for the challenging PSSP problem where the upstream and downstream information of a sequence is also useful for the prediction?

1.3 Approach

This thesis explores several research areas, RNN models and training algorithms from the ML area and PSSP from the Bioinformatics area, with main emphasis on developing second order learning algorithms for BRNNs and other novel neural network models for the PSSP problem.

The BRNN architecture of Baldi et al. [1999] is considered as one of the first and optimal computational neural network type architectures for addressing the PSSP problem. Initially, we implement the same BRNN architecture, but we have used a modified training procedure. More specifically, our aim was to identify the effect of the contribution

of local versus global information, by varying the length of the segment on which the Recurrent Neural Networks operate for each residue position considered. For training the network, the backpropagation learning algorithm with an online training procedure was used, where the weight updates occur for every amino acid, as opposed to [Baldi et al. \[1999\]](#), where the weight updates are applied after the presentation of the entire protein. Furthermore, we have used ensemble of 6 BRNNs to investigate the fluctuations in our results. Finally, our results improve even further when sequence-to-structure output is filtered in a post-processing step. We performed a comparative study on these results, utilizing both machine learning filtering techniques and filtering based on empirical rules to find the combinations which lead to the highest improvement.

After these initial investigations we moved to the core part of the thesis. The BRNN architecture is demanding high performance learning algorithms in terms of convergence time and memory usage to be trained effectively and efficiently. We have designed and implemented a second order method for training BRNNs for the PSSP problem where the SCG is applied for the first time on these models. More specifically, we have presented the development and implementation of the Hybrid Rectified-Scaled Conjugate Gradient (HR-SCG) learning algorithm for BRNN architectures which is based on the SCG learning algorithm. In contrast to the conventional Gradient Descent (GD) learning algorithm, the HR-SCG exploits both gradient and curvature information for convergence. Furthermore, the HFO [[Martens and Sutskever, 2011](#)], another second order learning algorithm, has been developed and applied for the first time on the BRNN architecture.

In addition, we have investigated and combined neural network models which have never been developed for the PSSP problem. We have applied new novel NN methods for the PSSP problem. Firstly, we have used a new RNN model which is called Clockwork-RNNs to investigate their performance on the PSSP results. Furthermore, we have developed a novel Bidirectional ESN architecture for the PSSP problem. Moreover, we have developed and implemented an innovative by design combination of CNNs with Support Vector Machines (SVMs) [Dionysiou et al., 2018] as a solution to the PSSP problem, with a novel two-dimensional (2D) input representation method, where Multiple Sequence Alignment profile vectors are placed one under another. This method gave an image-like representation of the protein data to be manipulated by the CNN models. Finally, we have developed and implemented a novel approach with simple Feed-forward Neural Networks trained the the powerful HFO learning algorithm for the PSSP problem. The results of all these methods are compared with the current most efficient method for the PSSP problem which is based on a LSTM-BRNN network [Heffernan et al., 2017].

In general, the approaches followed in this thesis can be summarised in three specific sections: (a) We have investigated the initial BRNN model [Baldi et al., 1999] and other methods to improve results for the PSSP problem, (b) We have designed and developed new second order learning methods to train the BRNN architecture and (c) we have investigated and combined new novel techniques for the PSSP problem, based on state-of-art neural network models and learning algorithms.

1.4 Outline

This thesis is structured as follows. *Chapter 2* covers the literature review on the fields of Neural Network models and PSSP problem. *Chapters 3, 4 and 5* present our work so far. *Chapter 6* shows a general discussion on our work and finishes with our conclusions. Finally, *Chapters 7* refers to the future work based on this thesis.

Chapter 2 presents a survey on Neural Network models (*Section 2.1*) and learning algorithms (*Section 2.2*) which are related to our work. Then, the *Section 2.3* presents the background and reviews the literature of the PSSP problem. In this section we present the information needed from the field of Bioinformatics to understand the PSSP problem, related work to this problem and the data we have used in this thesis.

The BRNN architecture of [Baldi et al., 1999] is considered as one of the optimal computational neural network type architectures for addressing the PSSP problem. This architecture is investigated in *Chapter 3*. In *Section 3.2*, we present a variation of the Bidirectional Recurrent Neural Network (BRNN) which was initially developed by Baldi et al. [1999]. We implement the same BRNN architecture, but we use a modified training procedure. Moreover, in *Section 3.3*, we present ensembles of 6 BRNNs to enhance our results. Based on this, we show results from multiple ensemble methods. In addition, our results improve even further when sequence-to-structure output is filtered in a post-processing step. In *Section 3.4*, we performed a comparative study on the challenging problem of filtering methods, utilising both machine learning techniques and empirical rules and we found that combinations of the two lead to the highest improvement.

The need to train BRNNs with more efficient algorithms than existing methods, in terms of accuracy and convergence time, has been the initial motivation for *Chapter 4*. The SCG [Møller, 1993b], a second-order learning algorithm, which has been found to be superior to the conventional GD algorithm in terms of accuracy, convergence rate and the vanishing-gradient problem [Hochreiter and Schmidhuber, 1997]. In addition, the original form of the algorithm [Møller, 1993b] does not depend on any parameters. Consequently, *Section 4.2* introduces the mathematical analysis and development of our proposed HR-SCG algorithm to train the BRNN architecture for the PSSP problem which is based on the SCG algorithm. Furthermore, in *Section 4.3*, we present the results from the application of another powerful second order learning algorithm, the HFO [Martens, 2010; Martens and Sutskever, 2011] algorithm, on the same architecture and problem.

The latest developments and results in the NNs field have demonstrated huge potential in several data-related problems. Nevertheless, many of these NN models, their variations and specific learning algorithms have never been used for the PSSP problem. More specifically, there is an open field to be investigated related to the PSSP problem and CW-RNNs (*Section 2.1.4*), CNNs (*Section 2.1.5*), RC (*Section 2.1.6*) methods and HFO algorithm (*Section 2.2.3*). Although these techniques have their advantages and disadvantages compared between each other, each one has specific characteristics related to accuracy, execution time, sorting with short and long term dependencies and quality of results. Hence, in *Chapter 5* we present results on an image-like input representation method for the PSSP problem which is used by Convolutional Neural Networks (*Section 5.2*), results of the same problem on Clockwork Recurrent Neural Networks (*Section*

5.3), results on a novel Bidirectional Echo State Network (*Section 5.4*) architecture and results on a novel method of simple Feed-forward Neural Networks trained with the HFO algorithm (*Section 5.5*).

Finally, in *Chapter 6* and in *Chapter 7*, we present our Conclusions and Future work, respectively. More specifically, we present an overview of the problem (*Section 6.1*), an overview of our approach, results and conclusions (*Section 6.2*), presented approaches to other problems (*Section 6.3*), contributions (*Section 6.4*) and dissemination of this PhD work (*Section 6.5*). In *Chapter 7*, we present future work and potential contributions based on this thesis.

Chapter 2

Survey: Neural Network algorithms and the PSSP problem

2.1 Introduction

This chapter deals with an extensive literature review coverage of the field of Artificial Neural Networks (ANN) and the PSSP problem. More specifically, we present ANN models, algorithms and methodologies which are related to this thesis. Furthermore, we present the PSSP problem and relevant datasets which have been used in this work.

2.2 Artificial Neural Networks

2.2.1 Introduction to Artificial Neural Networks

An ANN is a Biological inspired Machine Learning algorithm which consists of nodes or "neurons" (by analogy with their biological brain neurons). Each neuron of this algorithm is a McCulloch and Pitts model [McCulloch and Pitts, 1943], which can be seen on *Figure 1* (right). Individual neurons take single or multiple inputs and produce an output which is a non-linear transformation, through an activation function, of the product of the input values and their corresponding weights. Those neurons can only perform trivial functions. However, if those neurons are ordered in layered structures, they can solve any complex non-linear non-convex function of any problem. This complex structure is the Multi Layer Perceptron (MLP) structure, which is an ANN. An MLP ANN usually consists of a non active input layer, which does not consists of neurons, one or two active hidden layers and an active output layer. The structure of an MLP ANN can also be seen on *Figure 1* (left), where the Layer i is the input layer, the Layer j is the hidden layer and the Layer k is the output layer.

The output of a feed-forward Neural Network is given by *Equation 1* and *Equation 2*. The input vector values x_i are entered through the input Layer i , which propagate the x_i values to the hidden layer j through the connecting links. Each connecting link is associated with weight vector values v_{ij} that modify the propagated x_i values. Every neuron of

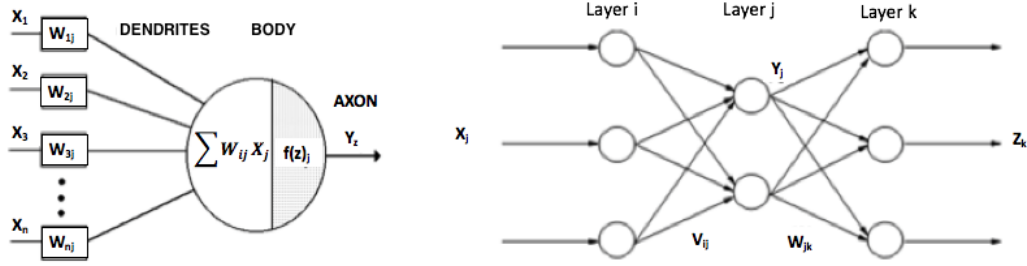


Figure 1: On the left the McCulloch and Pitts model and the right a typical MLP artificial neural network

the hidden Layer j is calculating its output through the *Equation 1*, where v_{ij} are the connection weights of the neurons, x_i are the outputs of the non-active neurons of the input layer, b_j is a bias term and f is an activation function. Similarly, the output of each neuron of the output Layer k is calculated through the *Equation 2*, where w_{jk} are the connection weights of the neurons, y_j is the outputs of the hidden layers neurons, b_k is a bias term and f is an activation function. The activation function can be any monotonically increasing and continuously differentiable function, like the sigmoid function (*Equation 3*), where u denotes the slope of the sigmoid.

$$y_j = f \left(\sum v_{ij} x_i + b_j \right) \quad (1)$$

$$z_k = f \left(\sum_j w_{jk} y_j + b_k \right) \quad (2)$$

$$f(u) = \frac{1}{1 + e^{-u}} \quad (3)$$

In the case where a second hidden layer is used, the feed-forward procedure is similar with the case where we have only one hidden layer. According to the Kolmogorov's

theorem [Kolmogorov, 1957] an MLP ANN with two active hidden layers can create any hyper plane and separate any type of data in space, independently of the input data dimension or the number of classes.

The algorithm that is used to train an MLP ANN is the standard Backpropagation which is based on the gradient decent method [Rumelhart et al., 1986a; Werbos, 1974]. This algorithm is a mathematical model which tries to find a minimization solution for an error function, based on the Gradient decent method. The weights of an MLP ANN are adjusted, through an iterative training procedure, so that the error function is minimized. The Backpropagation algorithm has two phases, the feed-forward phase and the backward phase. During the first phase, the input data is given to the MLP ANN and the actual outputs are calculated through *Equation 1* and *Equation 2*. Then the backward phase takes place. During this phase, the neurons' weights and biases, which are initialized randomly, must be trained in a way that the algorithm creates hyper-planes to separate in the input data space. The difference of the desired and the actual neuron's outputs is calculated and represents the error of the data's propagation through the system. After that, this error is back propagated through the network and is used to adjust the neurons' weights and biases. The adjustments on the neuron's weights are done with the *Equation 4* - *Equation 7*. More specifically, the Delta rule [Widrow and Hoff, 1960] is the summation of the *Equation 4* on a weight at time $t - 1$ to give the new weight at time t . Continuing on the feed-forward Equations, the weights that are attached to the output layer are trained with the *Equation 4* and *Equation 5*, where w_{kj} is the change to the weight from neuron j to neuron k , η is the learning rate, t_k is the desired output of a neuron and z_k is the

actual output of an output's layer neuron. Similarly, the weights that are attached to the hidden layers are trained with the *Equation 6* and *Equation 7*, where v_{kj} is the change to the weight from neuron i to neuron j , η is the learning rate and y_j is the actual output of a hidden layer's neuron.

$$\Delta w_{kj} = \eta \delta_k y_j \quad (4)$$

$$\delta_k = z_k(1 - z_k)(t_k - z_k) \quad (5)$$

$$\Delta v_{ji} = \eta \delta_j x_i \quad (6)$$

$$\delta_j = y_j(1 - y_j) \sum_k \delta_k w_{kj} \quad (7)$$

2.2.2 Recurrent Neural Networks

Further to ANN, Recurrent Neural Networks (RNN), have also processing units, which are called "neurons", and abstract synaptic connections which propagate signals through the network. The feature that distinguished them from ANNs is that the synaptic connection topology can process cycles. This topology is biologically inspired and computationally more powerful than other modelling approaches, such as Hidden Markov Models (HMMs), which have non-continuous internal states, feedforward neural networks and SVMs [Cortes and Vapnik, 1995a], which do not have internal states at all.

RNNs are an architecture that can enhance the power and capacity of simple ANN, which give them the advantage of extracting patterns and learning features of temporal sequence data. RNNs are universal approximators of dynamical systems [Lukoševicius and Jaeger, 2009] and are well-known for their power to memorise time dependencies and model nonlinear systems [Medsker and Jain, 2010]. They can be trained from examples to map input sequences to output sequences and in principle they can implement any kind of sequential behaviour. More specifically, through their recurrent connections they develop self-sustained temporal activation dynamics. This architecture upgrades the simple functional ANN to complex dynamical systems. Furthermore, their internal states manipulate the input history to a non linear transformation which produce a dynamical memory for temporal context information.

The research area around RNNs is divided in two major groups. The first group uses RNNs from their scientific perspective, as modelling biological brain, or as engineering tool in the fields of Computational Neuroscience and Machine Learning, respectively. In contrast, the second group uses RNNs from their dynamical systems perspective. The first class of this group is RNNs with main characteristic of energy-minimizing stochastic dynamics and symmetric connections. In this class, we can find models such as Hopfield networks [Hopfield, 1982], Boltzmann Machines [Ackley et al., 1985] and Deep Belief Networks [Hinton et al., 2006] which are trained mostly with unsupervised learning procedures. The second class of RNNs from the dynamical systems perspective have the characteristic of deterministic update dynamics and directed connections which are

trained with supervised learning algorithms and are used to transform an input time series into an output time series. This category on RNN models is the main target of this research document.

A RNN architecture is the main aspect of the model and usually determines the algorithm's performance. Through the literature we can indicate different categories which have been proposed to organize various RNN architectures. These categories depend on the layer that the recurrence feedbacks are returned as network's input (hidden or output layer), canonical RNNs and dynamic MLPs [Tsoi, 1998], autonomous converging and non-autonomous non-converging [Bengio, 1993], locally and output feedback, fully connected or non-fully connected RNNs. The most famous RNN architecture are Feed-Forward Time-Delayed (FFTD) [Waibel, 1989], Elman-type RNNs [Elman, 1990] and Jordan-type RNNs [Jordan, 1986]. More specifically, all the typical dynamic neural network architectures are described by [Peng and Magoulas, 2008] who specify the main four categories of RNN architecture. These architectures are FFTD, Layered Recurrent Network (LRN) [Elman, 1990], Nonlinear Autoregressive Network with Exogenous Inputs (NARX) [Jordan, 1986] and fully RNNs (*Figure 2*).

The latter type of RNN models have some advantages as ML algorithms. These models under general assumptions they are universal approximators of dynamical systems [Funahashi and Nakamura, 1993]. Furthermore, they exhibit recurrent connection pathways as biological brain modules. These advantages can give solutions for a wide range of applications. Despite their potential dynamical power, these algorithms show limitations on their performance on non-linear modelling which is still an open research area.

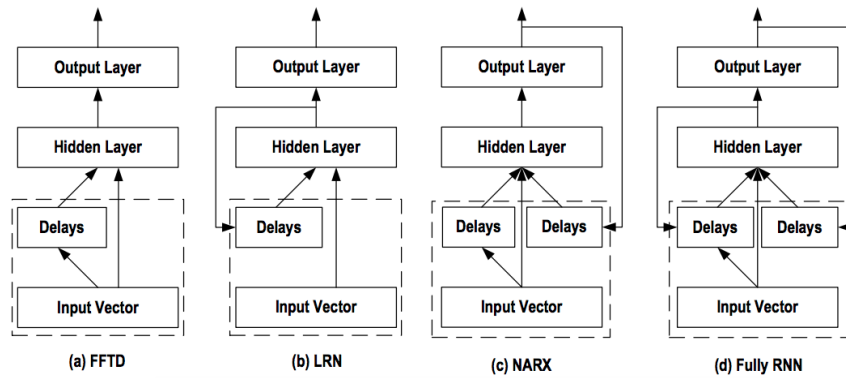


Figure 2: Typical dynamic neural network architectures [Peng and Magoulas, 2008](See text for more information)

These limitations arise mostly because of the gradient-descent based learning algorithms which are used to train RNNs. Independently, a number of training algorithms have been proposed to train RNNs but all suffer from the same issues which are still open problems in the field of RNNs. Firstly, during training the network parameters gradually change in a way that the network dynamics are driven through bifurcations [Doya, 1992]. This leads to the degeneration of gradient information which cannot guarantee the algorithm convergence. Secondly, long training times are shown for each single parameter update. This situation is computationally expensive and creates a bound on network's size. Another problem, is the well known vanishing gradient problem. The necessary information exponentially diffuses over time [Bengio et al., 2004] which destroys the long-range memory of the network. Finally, global control parameters of complex learning algorithms are not easily optimized which easily can create a chaotic system.

The state-of-the-art gradient decent learning algorithms are BPTT [Werbos, 1990] and Real Time Recurrent Learning (RTRL) [Williams and Zipser, 1989]. BPTT unfolds the RNN and builds an equivalent feedforward NN. Then the BP algorithm calculates in

batch learning the gradient vector and derivatives to train the network. In contrast, RTRL calculates the gradient history and derivatives in an online mode and propagates them forward in time. Further to the general problems of gradient decent algorithms, BPTT and RTRL have their own drawbacks. BPTT can learn data only in batch mode and needs a significant amount of storage due to the fact that it stores all the states of the network. RTRL drawbacks are the large computational time of $O(N^4)$ and storage requirements. Another method which shows that converges faster, than the ones previously mentioned, is the Atiya-Parlos Recurrent Learning (APRL) [Atiya and Parlos, 2000]. This method can give more robust and accurate results. The main difference of this method is the gradient vector calculation. This method does not use a pure gradient decent model but it calculates the gradient vector with respect to neuron activations and not only the weights directly. Furthermore, Extended Kalman Filters [Puskorius and Feldkamp, 1994] and Expectation-Maximization algorithm [Ma and Ji, 1998] are used to train RNNs but with no better results.

2.2.3 Bidirectional Recurrent Neural Networks

Predictions on sequential data are particularly challenging when both the upstream and downstream information of a sequence is important for a specific time-step. Application examples include Phoneme Speech Recognition (PSR) [Graves and Schmidhuber, 2005; Wollmer et al., 2009] and Bioinformatics problems, such as the PSSP [Baldi et al., 1999; Agathocleous et al., 2010; Kountouris et al., 2012] and other related problems (e.g., Transmembrane Protein Topology Prediction (TMPTP) [Nugent and Jones, 2009]). In

such sequence-based problems the events are dynamic and located downstream and upstream, i.e., left and right in the sequence. A ML model designed for such data must learn to make predictions based on both directions of a sequence. To predict these events, researchers utilise BRNN architectures [Baldi et al., 1999].

A typical BRNN architecture [Baldi et al., 1999] consists of two RNNs and a Feed Forward Neural Network (FFNN). The novelty of this architecture is the contextual information contained in the two RNNs, the Forward RNN (FRNN) and the Backward RNN (BwRNN). The prediction at time-step t , for a segment at a time-series, is performed based on the information contained in a sliding window W_a . The FRNN iteratively processes the information located on the left side of the position t to compute the forward (upstream) context (F_t). Similarly, the BwRNN iteratively processes the information located on the right side of the position t to compute the backward (downstream) context (B_t). Hence, the two RNNs are used to implement F_t and B_t (Figure 3). These RNNs correlate each sequence separately and hold an internal temporary knowledge to form the network's internal memory [Elman, 1990].

The BRNN architectures are usually trained with an extension of the BPTT algorithm [Frasconi et al., 1998] with the error propagated in both directions of the BRNN.

2.2.4 Clockwork-Recurrent Neural Networks

Clockwork-Recurrent Neural Network (CW-RNN) [Koutnik et al., 2014] is a recently proposed, innovative architecture, which was created in order to address weaknesses of RNNs, namely their difficulty to train successfully on long-term memory (Figure 4). The

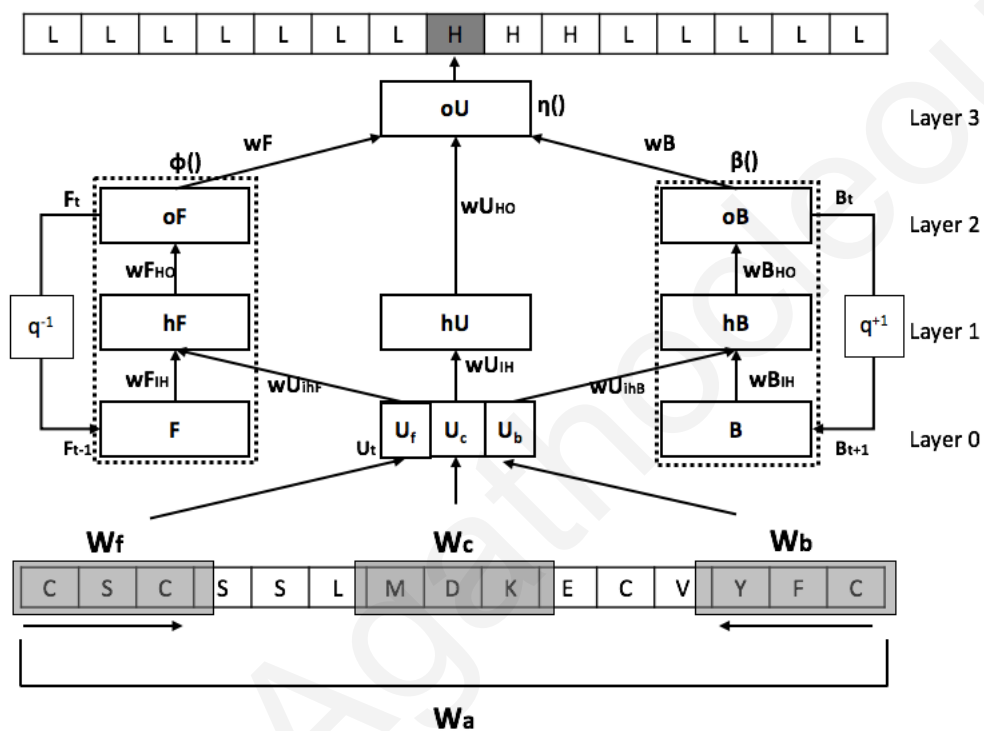


Figure 3: **The BRNN architecture:** Left (forward) and right (backward) context (F_t and B_t) are implemented by recurrent networks, where q^{-1} is the back-shift operator and q^{+1} is the forward-shift operator as explained in Baldi et al. [1999]. The input layer is fed with W_a , W_f , W_b and W_c sliding windows (see text for description of the parameters). The output layer O_t has three softmax units associated with membership in each of the three secondary structure classes for the current residue (position t). Functions $\phi()$, $\beta()$, and $\eta()$ are implemented by feed-forward neural networks [Baldi et al., 1999].

reduction of the number of parameters, its improvement in performance over specific tasks, which were tested and the manner that speeds up evaluation compared to RNNs, are indices that overpower the simple RNNs. These advantages are due to the fact that the hidden layer of this architecture is composed of a number of modules, where each module is making computations at its own clock speed, as they are all assigned a clock period. Having these modules running at different speeds, computations are constrained to take place in different time steps and this is how both the long-term dependency and the vanishing-gradient problems are mitigated. As a result, at each time step only a number of modules are executed which leads to faster training and evaluation. Another feature of this architecture is the connections between the modules. Even though the modules are internally fully-interconnected, only faster modules are connected to slower ones and not vice-versa. This results in a smaller number of weights which is another reason as to why CW-RNN trains and evaluates faster than RNNs.

2.2.5 Reservoir Computing

Recent advances in the field of RNNs, known as Reservoir Computing (RC) [Lukoševičius et al., 2012], facilitated the practical application of RNNs and demonstrated better performance than classical approaches. The two pioneering RC approaches were proposed in 2001, the year which appears as the beginning of this ML field, under the names ESN [Jaeger and Haas, 2004; Ozturk et al., 2007] and Liquid State Machines (LSM) [Maass et al., 2002]. These approaches were proposed independently by [Jaeger, 2001] and [Maass et al., 2002], respectively. These models have their roots in Computational

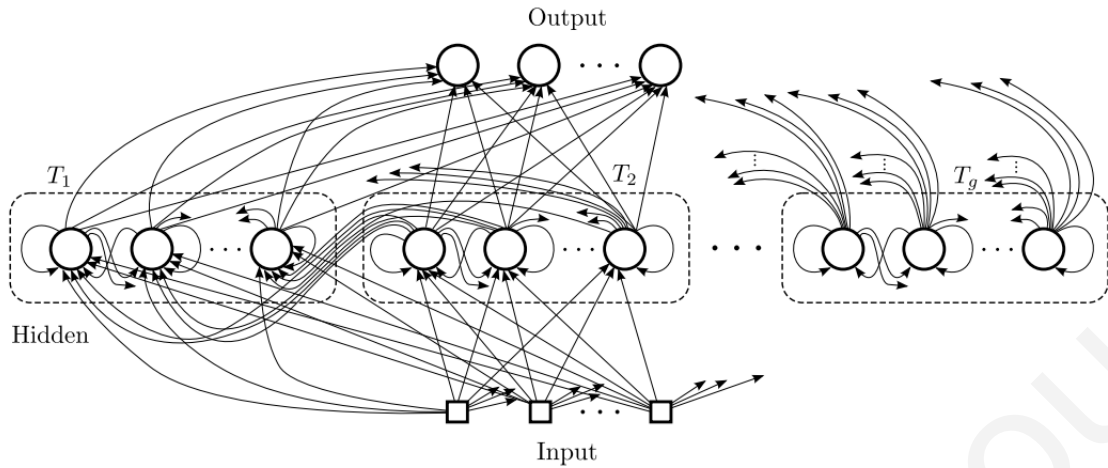


Figure 4: **The CW-RNN architecture** is similar to a simple RNN with an input, output and hidden layer. The hidden layer is partitioned into g modules each with its own clock rate. Within each module the neurons are fully interconnected. Neurons in faster modules i are connected to neurons in a slower module j only if the clock period $T_i < T_j$. This figure is reprinted from [Koutnik et al. \[2014\]](#).

Neuroscience, due to their architecture, and in ML, because of the learning algorithms they use.

RC models overcome the most common problems that appear generally in RNNs. The general curve of RC architectures and learning avoid the gradient descent training algorithms. A RC algorithm consists of two parts. A randomly fixed recurrent neural network is created, which is called the reservoir. This part of the algorithm is responsible for a non linear transformation of the input signal to an output sequence of temporal vectors. Readout is the second part of the algorithm which achieves a linear transformation of reservoir's output to an output signal. This concept was firstly approached by [\[Dominey and Ramus, 2000\]](#). Although their research was based on cortico-striatal circuits in the human brain (empirical cognitive neuroscience, functional neuroanatomy), they mentioned that there is no learning in the recurrent connections, only between the State units and the

Output units. They also said that the adaptation is based on a simple associative learning mechanism. Furthermore they mentioned that the fact that the prefrontal network relies in fixed random recurrent connection is worth nothing. The conclusions of [Dominey and Ramus, 2000] did not have any impact in the evolution to RC field because only in 2008 ML RC researchers became aware of each other [Bengio et al., 2013]. However, the experimental results of [Dominey and Ramus, 2000] have approved the theory behind computational RC research. Since 2001, the RC field has shown great growth in the aspects of RC architectures, reservoir and readout learning algorithms.

Today, RC methods, have become popular and constitute one of the basic paradigms of RNN modelling [Lukoševičius et al., 2012]. Firstly, RC methods have outperformed classical nonlinear prediction and classification algorithms in demanding problems like speech recognition [Maass et al., 2004; Verstraeten et al., 2005; Dominey et al., 2006; Blanc and Dominey, 2003] and EEG signal processing [Kindermans et al., 2010; Sussillo et al., 2012]. Secondly, they are computationally universal for continuous time, continuous value real time system models [Maass et al., 2006]. Furthermore, this kind of algorithms have biological plausibility. The principles of RC connections have been inspired from architectural and dynamical properties of mammalian brains to explain and use the abilities of accurate computations in noisy physical substrate [Haeusler and Maass, 2007], accurate timing and representation of sequential information. The latter is widely used to explain the sequential information processing especially in the speech recognition problem [Maass et al., 2004; Verstraeten et al., 2005; Dominey et al., 2006; Blanc and Dominey, 2003]. Finally, RC models have the ability of extensibility and parsimony

because of the fact that the reservoir is separated from the readout. Consequently, previous learned representations are not destroyed when new items are presented as input to the model. This can be done by adding some new outputs and train them as a different readout.

The work of [Jaeger, 2001] generally defines a basic constructive guide for a powerful ESN. More specifically, a big, sparsely and randomly connected reservoir can achieve very promising results for a lot of problem. Also this work gives guidelines on how sparse or dense the reservoir input and feedback weights should be. Furthermore, [Jaeger, 2001] specifies the relation of the reservoir weight matrix spectral radius with memory and non-linearity that a task requires. The latter ensures the echo state property of each implementation. Nevertheless, lots of ESN implementations have appeared in the recent years which modify the standards of initial the ESN reservoir model. The most common modifications refer to the factor topology. Accordingly, some of these have to do with small world networks [Liebald, 2004] and scale free random networks [Jiang et al., 2008] and biologically inspired connection topologies. The most important conclusion of different topologies is the evidence of "no free lunch" rule. Another topology, which has been suggested for an ESN reservoir, is the layered reservoirs. This idea was motivated by the fact that a reservoir in an abstract view is just one layer of recurrent connected neurons. Finally, problems that have been shown with the amount of time scale of a reservoir, have been solved with the replacement of simple neurons with LSTM neurons [Schmidhuber et al., 2007], which unfortunately increase the complexity of the algorithm.

Training only the readout of RC methods was the main learning procedure of the method which was proved through some findings. More specifically, [Steil, 2005] has executed some training RNN simulations with the APRL [Atiya and Parlos, 2000] learning algorithm and then he investigated the way that the dynamics were changing over time. During training of RNNs this algorithm leads to functional decomposition of the method's architecture into slowly changing reservoir dynamics and fast adapting readout layer. Consequently, these findings become the basis to investigate the situation of training only a readout after a recurrent component. This method shows equivalent results with other methods which train the whole recurrent structure but it is much faster than the others. The complexity time of such learning algorithm is $O(n)$ [Steil, 2005] in contrast of $O(n^2)$ [Weiskopf et al., 2004] and $O(n^4)$ [Ronald and Zipser, 1989].

The most common structure of a readout is the original one single-layer readout [Jaeger, 2001; Maass et al., 2002]. More specifically, the training algorithm which was proposed for this structure was the linear regression algorithm. Linear regression gave the advantages of fast learning and simplicity in a batch learning mode. In the case of ESN, the application of linear regression to train the readout is a straight forward task. In contrast, the case of LSMs with spiking neurons is more complicated and needs a low pass filtering technique between the reservoir and readout [Carnell and Richardson, 2005]. Furthermore, evolutionary algorithms have been used to train the readout [Jiang et al., 2008]. This method did not enhance the performance of a readout but it appears useful when temporal data do not have a desirable output at each time-step.

In the case of on-line weight adaptation several algorithms have been proposed to train the single layer readout. The most common algorithms for this task are Least Mean Square approaches. The most powerful algorithms to train online readouts however is BackPropagation-Decorrelation (BPDC) [Steil, 2005]. BPDC is used to train only the output weights of an ESN. More specifically, BPDC uses a one-step back propagation of errors (virtual teacher forcing), temporal memory by decorrelating of activations and a non-adaptive linear reservoir. The linear dynamical reservoir with fixed weights receives an input signal and provides a dynamical memory. If the states of the reservoir are maximally decorrelated then the information processing capacity is maximal. Then the one-step back propagation is used to map the reservoir *echos* with the desired outputs. Generally, this algorithm produces high quality results with low computational power. Finally, on-line training of LSMs with firing rate-coded readout have been achieved with reward-modulated STDP [Legenstein et al., 2008].

2.2.5.1 Echo State Networks

ESN [Jaeger, 2001] constitute a basic and one of the most powerful RC methods. The untrained RNN of this model is called *dynamic reservoir*. It takes as input a sequence of data and produces in its output temporal representations of those inputs. These representations are called *echoes* and are used for the algorithm's name.

The reservoir of ESNs traditionally has sigmoidal hidden neurons (usually using the $tanh(\cdot)$ function) which generally build a nonlinear relationship between the reservoir's input temporal data and the output *echoes*. This reservoir is randomly generated and does

not need any training procedure. The reservoir of ESNs [Jaeger, 2001] is characterized by the *echo state property*. More specifically, it says that the information from previous inputs and previous state of the reservoir must decrease gradually and should not get amplified. Furthermore, ESN echoes are passed through a trained (usually with batch linear regression) readout and produce the desired output. The architecture of an ESN as it was originally designed by Jaeger [2001] can be seen in *Figures 5 and 6*.

The theory behind the separation of reservoir and readout is based on general ML theory and on the fact that both serve different purposes. The reservoir has the ability, like kernel methods, to expand the input sequence to a higher dimensional, rich enough state space. In contrast, the readout has the ability to map the reservoir output representations to desired output signals [Lukoševicius and Jaeger, 2009]. This separation gives the advantage of generating and training these sub-models separately, which will produce the detail requirement to achieve better performance.

In *Figure 5* we can see the architecture of a simple ESN by [Ozturk et al., 2007]. In order to explain how an ESN works we assume the existence of M input units, N internal units and L output units. Furthermore, $u(n)$ is the value of input units at time n , $x(n)$ is the value of internal units at time n and $y(n)$ is the value of output units at time n . The connection weights between the input and the internal units are given by an $N \times M$ weight matrix W^{in} , the connection weights between the internal units are given by an $N \times N$ weight matrix W , the connection weights between the internal units and the output units are given by an $L \times N$ weight matrix W^{out} and finally the connection weights between the output units and the internal units are given by an $N \times L$ weight matrix W^{back} . The internal

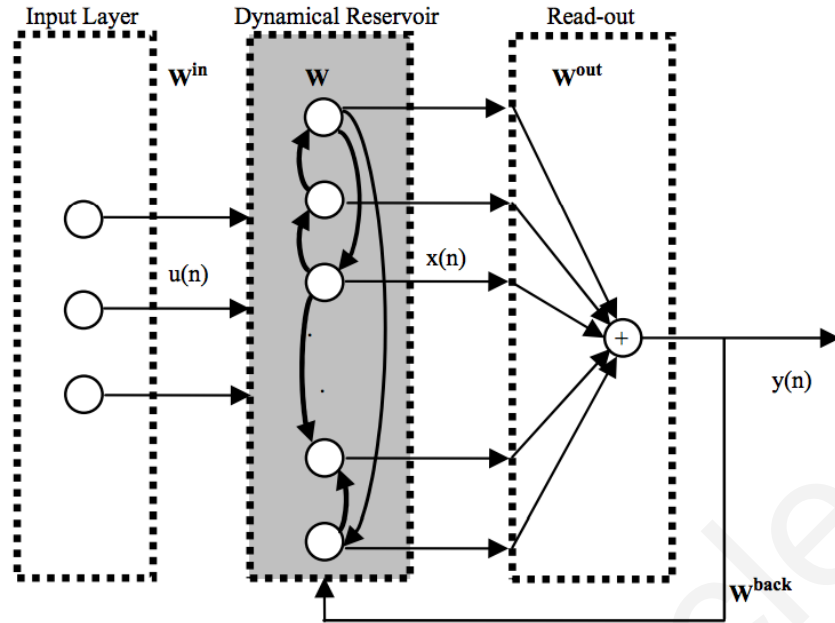


Figure 5: The ESN network architecture [Ozturk et al., 2007]

units' activation is calculated by *Equation 8*, where $f(\cdot)$ is the activation function used for the internal units. Usually the hyperbolic tangent function is used for $f(\cdot)$. The output of internal units feeds the readout, which calculates the network's output in *Equation 9*. The easiest way to train this system is by linearising the system around the current state $x(n)$ and create a Jacobian matrix, which can be seen on *Equation 10*. Then a linear regression algorithm can train the system by calculating the weight vector W (*Equation 10*).

$$x(n+1) = f(W^{in}u(n+1) + Wx(n) + W^{back}y(n)) \quad (8)$$

$$y(n+1) = f^{out}(W^{out}x(n+1)) \quad (9)$$

$$J(n+1) = \begin{bmatrix} f(\text{net}_1(n))w_{11} & f(\text{net}_1(n))w_{12} & \dots & f(\text{net}_1(n))w_{1N} \\ f(\text{net}_2(n))w_{21} & f(\text{net}_2(n))w_{22} & \dots & f(\text{net}_2(n))w_{2N} \\ \dots & \dots & \dots & \dots \\ f(\text{net}_N(n))w_{N1} & f(\text{net}_N(n))w_{N2} & \dots & f(\text{net}_N(n))w_{NN} \end{bmatrix} W = F(n)W \quad (10)$$

ESNs have been used in all kinds of classification and prediction problems and in most cases they outperform the classical ML methods [Maass et al., 2004; Verstraeten et al., 2005; Dominey et al., 2006; Blanc and Dominey, 2003; Kindermans et al., 2010; Sussillo et al., 2012].

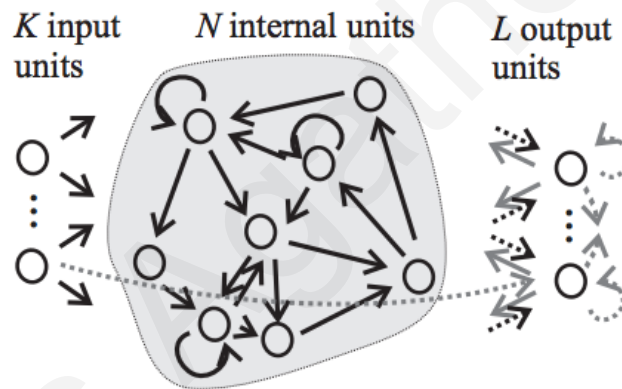


Figure 6: The ESN network architecture as it was firstly designed by [Jaeger, 2001]

2.2.5.2 Other Reservoir Computing Methods

LSM [Maass et al., 2002] is the second of the two initial RC models. This model was introduced from the computational neuroscience field independently but during the same period with ESNs. The basic idea of the model was the concept of randomness and some

computational properties that appear in brain's neural microcircuits [Maass et al., 2002, 2003]. Due to the latter, LSMs are a more biologically realistic model than other RC methods. The initial model of [Maass et al., 2002] uses spiking integrate-and-fire neurons with dynamic synaptic connections to build the reservoir. The LSM's reservoir is called the liquid and temporal vectors of its output sequence are called liquid states.

An approach that typically uses a smaller number of hidden neurons than ESNs and LSMs is Evolino [Schmidhuber et al., 2007]. Evolino uses LSTM cells [Hochreiter and Schmidhuber, 1997] and optimises their connection weights using an evolutionary algorithm.

If we exclude the variations of most known RC methods (ESNs, LSM, and Evolino) some other models are also under the umbrella of RC. One of these models is the Temporal Recurrent Network (TRN) [Dominey and Ramus, 2000]. As we have already mentioned, [Dominey and Ramus, 2000] were the first who introduced the concept of RC. Their algorithms are simple Neural Networks which introduce the idea of the reservoir and are trained with the Least Mean Square methods and a gradient decent algorithm. Over the ML field, the RC concept has also been used for harnessing the computational power unconventional hardware. Such work has been done by [Schürmann et al., 2005] and [Schrauwen et al., 2008] on analog electronics, by [Nikolić et al., 2007] on biological neural tissues and by [Vandoorne et al., 2008] on optical computers. Finally, an other perspective of RC is the work of [Jones et al., 2007] who have built a reservoir with computer-simulated gene regulation network of bacteria [Jones et al., 2007]. This latter

network takes an input sequence of chemical stimuli and produces an output of protein levels and mRNAs measures.

2.2.6 Deep Learning and Convolutional Neural Networks

The performance of ML algorithms is heavily dependent on the choice of data representation (features) on which they are applied. For this reason, much of the actual effort in deploying ML algorithms goes into the design of preprocessing pipelines and data transformation that result in a representation of the data that can support effective ML methods [Bengio et al., 2013]. Feature engineering is the process of transforming data into useful information vectors based on prior knowledge. The weakness of such methods is that in most cases we do not have all the prior knowledge due to hidden variables, unknown data distributions and data patterns. Consequently, the main concept of transforming raw data to data with useful information probably does not work and has a negative impact on the performance of applications. The ideal scenario is the construction of powerful algorithms which could identify the hidden features and automatically create the data representations. These algorithms should be able to be applied to any system, with no other human effort, for prediction and classification purposes. This can be done by powerful Representation Learning (RepL) algorithms. In simple words, a good data representation can enhance the performance of a predictor or a classifier, does not need any human effort [Vincent et al., 2010] and does not need any prior information on what exactly the problem is. However, this ideal scenario does not always work and is heavily dependent on the data probability densities. Although representation of data is one of the

roots of ML algorithms, RepL has been established as a ML field only in the last few years.

In general, such categories of algorithms are Manifold Learning for low dimensional data [Cayton, 2005], Sparse Coding for sparse data [Olshausen, 1996; Goodfellow et al., 2012], Auto-encoders [Bourlard and Kamp, 1988; Hinton and Zemel, 1994] and Deep Learning [Hinton et al., 2006].

Deep Learning (DeL) is a group of RepL algorithms which are useful for training multilayer models [Bengio et al., 2013]. These models can be Deep Belief Neural Networks [Hinton et al., 2006], Deep Boltzmann Machines [Sutskever et al., 2009], Recurrent Neural Networks (very deep networks with shared parameters), Convolutional Neural Networks or any other multilayer model. Deep architectures are the ideal models to solve problems with complicated high-dimensional data, which include several tasks. These tasks can be interpreted in multiple layer architectures that have the ability to sequentially extract all the hidden features [Hinton et al., 2006; Bengio, 2009]. The special category of RNNs have the added advantage of processing sequential complicated high dimensional data. However, training such algorithms can be very difficult (Section 2.2.2). Deep architectures can be trained with two main techniques. The first technique is called semi-supervised procedure and the second one is based on generalization and optimization steps. The first technique is well defined by [Weston et al., 2008] and is using unsupervised dimensionality reduction methods to extract features and then use them to train some layers or all the layers of deep architectures. The second technique, which is the one that we are interested in, uses pre-training techniques to initialise deep architecture

parameters and then an optimization algorithm to optimise the learning procedure. The basic idea of these training algorithms, which is called Unsupervised Layerwise Feature Stacking, is that they separate each layer distinct levels of concept which are trained separately but each higher layer in the structure needs data from the previous layers to be trained [Bengio, 2009]. Through this procedure, deep learning algorithms can identify multiple levels of representation which consist a hierarchy of features [Bengio et al., 2013]. Each level of deep architectures can create a higher level of data abstraction based on lower levels of data abstraction. In contrast with manifolds of data which can destroy the input information, the distributions that are created on the data through deep learning are simpler and linearly separable [Bengio et al., 2013]. According to [Erhan et al., 2010], layerwise unsupervised pre-training can help the supervised learner because the training on intermediate representations is guided and representations are easier to be learned than learning all the information at once. Furthermore, [Bengio, 2009] said that is easier to learn simpler concepts first and then build higher-level ones based on the simple ones. Both observations lead to the concept of higher-level abstraction through a process of learning better features from already extracted features. Empirically, the data representations of layerwise feature stacking methods had better results in terms of classification and error minimization of deep architectures [Larochelle et al., 2009; Erhan et al., 2010; Salakhutdinov and Hinton, 2009; Goodfellow et al., 2009].

A CNN is a class of deep, feedforward artificial neural networks (NN) that has successfully been applied to analyzing visual imagery [Krizhevsky et al., 2012; Rawat and Wang, 2017]. CNNs were inspired by the human visual system, where individual cortical

neurons respond to stimuli, only in a restricted region of the visual field, known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs have enjoyed a great success in large-scale image and video recognition [Srinivas et al., 2016]. This has become possible due to the large public image repositories, such as ImageNet [Krizhevsky et al., 2012], and high-performance computing systems, such as GPUs or large-scale distributed clusters [Simonyan and Zisserman, 2014]. Overall, CNNs are in general a good option for feature extraction, immense complexity sequence and pattern recognition problems [Krizhevsky et al., 2012; Rawat and Wang, 2017; Srinivas et al., 2016; Simonyan and Zisserman, 2014; LeCun and Bengio, 1998; Srinivas et al., 2016; Bluche et al., 2013; Graves et al., 2013].

CNNs are biologically-inspired variants of MLPs. The CNN architecture consists of an input layer (inactive), multiple hidden layers and an output layer. Generally speaking, CNNs combine three architectural ideas to ensure some degree of shift, scale, and distortion invariance: local receptive fields, shared weights, and spatial subsampling/pooling [LeCun and Bengio, 1998]. The hidden layers of a CNN typically consist of convolutional layers, pooling layers and fully connected layers. There are four main operations performed by a CNN: (a) convolution, (b) non linearity (Rectifier Linear Unit - ReLU), (c) pooling or sub sampling, and (d) classification. One of the major characteristics of CNNs is that they take advantage of the fact that the input would be like an “image”, so they constrain the architecture in a more sensible way. Every layer of a CNN transforms one volume of activations to another through a differential function. The arrangement of a CNN’s neurons, unlike a regular NN, is in 3 dimensions: width, height and depth.

The Convolutional Layer (CL) is the core building block of a CNN that basically performs the feature extraction process. The key hyperparameter of a CL is the kernel. The kernel is basically a 2D array initialized with random values, and it is used to compute dot products between the entries of the filter and the input volume at any position. The stride is another important hyperparameter that defines the amount of sliding of the kernel across the width and height of the input volume. The result of the kernel sliding over the width and height of the input volume is the feature map, a 2D array holding the responses/activations of the kernel at any spatial position. Moreover, the CNNs' ability to handle complex sequential data relies in part to the sparse connections of neurons. More specifically, each neuron is connected to only a local region of the input volume (i.e., receptive field), and as a result CNNs are capable of encoding complex sequential data correlations in their structure. The Pooling Layer (PL) is another critical block, for building a CNN. Generally speaking, a common technique for constructing a CNN is to insert a pooling layer in-between successive CLs. The main purpose of a pooling layer is to (a) reduce the representation size, (b) reduce the amount of computation in the NN, and (c) control overfitting. The PL uses a filter of a certain dimension and resizes the input given spatially, by striding the filter across the input volume and performing usually the MAX operation.

The last layer of a CNN is usually a fully-connected Softmax output layer. Nevertheless, this final step can be practically realized with any suitable classifier. In particular, a small advantage was reported when the softmax output layer of a CNN was replaced by a linear SVM [Tang, 2013].

2.3 Neural Network Learning Algorithms

2.3.1 First and Second Order Learning Algorithms

A number of first and second order optimization algorithms have been developed [Burney et al., 2007] and are routinely used for training ANNs. More specifically, first order learning algorithms use gradient information for optimization while second order learning algorithms use both gradient and curvature information. Compared to each other, first and second order ANN learning algorithms, have their advantages and disadvantages in terms of convergence rate and memory consumption. Training ANNs with specific learning algorithms for specific problems can improve results and minimize convergence time. Although, many first and second order learning algorithms have been developed for ANNs, not all of them have been applied on RNN architectures.

The most common first order learning algorithm for training ANNs is the Backpropagation learning algorithm (BP) [Rumelhart et al., 1986a; Werbos, 1974] which is based on the Gradient Descent (GD) method. BP is a powerful method which finds the derivatives of an error function with respect to the tunable weights in the network [Riedmiller and Braun, 1993; Burney et al., 2007; Liew et al., 2016]. More specifically, the network weight values are updated along the negative of the gradient of the error function. The BP learning algorithm works in both batch and stochastic learning modes. In batch mode the algorithm is summing the error gradients of all the input records in the training data to update the weights of the model. Consequently, averaging the error gradients makes

the algorithm robust in noise but leads to a slow convergence rate. On the other hand, the stochastic BP is updating the model's weights each time a new input record is presented to the input layer of the model. In contrast to batch BP, this method can reach convergence much faster because it appears to avoid local minima [LeCun et al., 1998; Liew et al., 2016]. In general, the speed of algorithm convergence can be controlled by the learning rate parameter which is basically the step size of the GD method. To control this parameter global and local adaptive algorithms have been suggested [Senior et al., 2013; Liew et al., 2016]. Furthermore, this algorithm has been modified to train RNNs as well, by forming the BPTT learning algorithm [Frasconi et al., 1998][Werbos, 1990]. Finally, the GD method has been improved based on adaptive estimates of lower-order moments to optimize stochastic objective functions. This method is called Adaptive Moment Estimation (Adam) and it has shown improvement of convergence time for several problems [Diederik and Jimmy, 2015].

Superior to first order learning algorithms are second order learning algorithms. These learning algorithms can develop a unique step size for each weight [Burney et al., 2007], which is usually done in two procedures. Firstly, the algorithm has to decide a direction in weight space to minimize the error function. The most common algorithm to decide the optimum direction is the line search algorithm [Johansson et al., 1991; Charalambous, 1992]. Then, the individual step size is calculated by variants of the Newton's method [Robitaille et al., 1993]. Thus, the gradient information is used to update a network's weight vector. Although the second order algorithms have many advantages against the conventional gradient descent optimization algorithms [Møller, 1993b], they have some

computational drawbacks. Calculating the Hessian matrix, which contains the second-order partial derivatives of a function, requires $O(N^2)$ memory complexity and $O(N^3)$ computational complexity, where N is the number of network's free parameters. Thus, they are impractical for very large networks.

One very common second order learning algorithm is the Quasi-Newton method (QN) which is based on the Newton's method [Li and Yan, 1995]. The Newton's method points directly to a minimum of the error surface but it has to recalculate the Hessian matrix repeatedly. The QN method overcomes this problem by calculating an approximation of the Hessian matrix. Furthermore, a line search method is used to calculate the appropriate direction vector, which is time consuming. Clearly, the QN method can converge really fast with a good Hessian matrix approximation [Liew et al., 2016]. Furthermore, the Levenberg-Marquardt algorithm (LM) [Levenberg, 1944; Marquardt, 1963] has been developed following similar principles as the QN method, but with much different mathematical background. This algorithm has also a mechanism to calculate an approximation of the Hessian matrix to be more efficient in terms of time complexity and memory consumption. The LM algorithm has been shown through practice that it is suitable for training modest size ANNs [Liew et al., 2016]. Concluding, the most appropriate algorithm between QN and LM for training a specific ANN depends on the network's size and the problem which has to be solved.

One of the most famous second order learning algorithms is the Conjugate Gradient algorithm (CG) [Johansson et al., 1991; Charalambous, 1992]. Most of the second order learning methods have been derived on the assumption of a quadratic error function with

a positively-defined Hessian matrix. The CG algorithm calculates the direction and step size information without explicit knowledge of the Hessian matrix. Hence, the costly computation and evaluation of the Hessian matrix, which has to be done repeatedly, can be avoided. More specifically, this algorithm is using conjugate vector information to avoid the Hessian matrix calculation. Then, the line search method is used to calculate the direction in weight space to minimize the error function. Finally, a number of search functions have been developed and used for the algorithm to minimize to the most desirable step size [Battiti, 1989]. The CG does not have any great requirements in terms of storage, which makes the algorithm eligible for networks with large numbers of weights. The SCG learning algorithm [Møller, 1993b], which is a variant of the CG, has been developed to eliminate the time consuming line search procedure by using a Levenberg-Marquardt approach to calculate and scale the step size. Consequently, the algorithm is significantly improved in terms of execution time. Finally, CG-based methods converge to the unique global optimum of quadratic functions in at most N steps, where N is the number of a model's weight vector length. This is a great advantage against (stochastic) gradient descent, which takes very large numbers of steps to converge even for simple quadratic problems. Consequently, this method has been used to improve the convergence time of deep learning architectures [Le et al., 2011].

Moreover, the HFO [Martens, 2010; Martens and Sutskever, 2011] second order learning algorithm demonstrated promising results on problems such as the noiseless memorization problem, the 3-bit temporal order problem and the random permutation problem [Martens and Sutskever, 2011]. In this algorithm, the method of finite differences is used

to create a Hessian Matrix approximation. The finite differences method is based on the model's gradient vector, which can be calculated much faster and in a way easier than the Hessian Matrix. The system's gradient vector and Hessian Matrix approximation are used to compute the system's Taylor expansion function. Then, the Preconditioned Conjugate Gradient algorithm (PCG) [Martens and Sutskever, 2011], which is a variant of the CG algorithm, is used to optimize the calculated Taylor expansion function. The PCG algorithm is used to give a new step size and a direction to update the system's parameters. This algorithm which was based on the works of Møller [1993b,a], Pearlmutter [1994] and Gers et al. [2002], has proven that it can manage with the Fundamental Deep Learning Problem in RNNs [Schmidhuber, 2015]. Consequently, HFO has been applied on deep NN architectures and it appeared to outperform other learning algorithms in specific sequential problems [Martens and Sutskever, 2011].

Different ANN learning algorithms appear to have advantages and disadvantages, but only few (i.e., CG and HFO learning algorithm) have been applied to RNNs [Charalambous, 1992; Martens, 2010].

2.3.2 The Scaled Conjugate Gradient algorithm

SCG is an optimization learning algorithm which has been introduced by Møller [1993b,a]. This algorithm is based on the conventional CG learning algorithm but does not contain any user-dependent parameters whose values are crucial for its success. More specifically, this learning algorithm avoids a time consuming line search per learning iteration by using a step size scaling mechanism.

The basic idea in replacing the line search algorithm of CG is based on a theorem related to conjugate directions which can be found in [Hestenes, 1980] and is well explained in [Møller, 1993b,a]. The idea is replacing the term $s_k = E''(w_k)p_k$ with:

$$s_{\kappa} = \frac{E'(w_{\kappa} + \sigma_{\kappa}p_k) - E'(w_k)}{\sigma_{\kappa}}$$

where s_k is the second order information, $E''(w_k)$ is the Hessian Matrix and p_k is set to the negative of the model gradient vector E' .

The SCG learning algorithm to train Feedforward Neural Networks, as appeared in the work of Møller [1993b,a], can be seen in the next steps:

1. Choose weight vector w_k and scalars $\sigma > 0$, $\lambda_l > 0$, $\bar{\lambda}_l = 0$.

Set $p_k = -E'(w_k)$, $r_k = -E'(w_k)$, $\kappa = 1$, **success=true**.

2. If *success = true*, then calculate second order information:

$$\sigma_{\kappa} = \frac{\sigma}{|p_{\kappa}|}$$

$$s_{\kappa} = \frac{E'(w_{\kappa} + \sigma_{\kappa}p_k) - E'(w_k)}{\sigma_{\kappa}}$$

$$\delta_k = p_k^T s_k$$

3. Scale δ_k : $\delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k)|p_k|^2$

4. If $\delta_k \leq 0$ then make the Hessian Matrix positive definite:

$$\bar{\lambda}_k = 2\left(\lambda_k - \frac{\delta_k}{|p_k|^2}\right)$$

$$\delta_k = -\delta_k + \lambda_k |p_k|^2$$

$$\lambda_k = \bar{\lambda}_k$$

5. Calculate step size α_k :

$$\mu_k = p_k^T r_k$$

$$\alpha_k = \frac{\mu_k}{\delta_k}$$

6. Calculate the comparison parameter Δ_k :

$$\Delta_k = \frac{2\delta_k[E(w_k) - E(w_k + \alpha_k p_k)]}{\mu_k^2}$$

7. If $\Delta_k \geq 0$ then a successful reduction in error can be made:

$$w_{k+1} = w_k + S_{cSS}(k)\alpha_k p_k$$

$$r_{k+1} = -E'(w_{k+1})$$

$$\bar{\lambda}_k = 0$$

success = true

If $k \bmod N = 0$ then restart algorithm:

$$p_{k+1} = r_{k+1}$$

$$\text{else: } \beta_k = \frac{|r_{k+1}|^2 - r_{k+1} r_k}{\mu_k}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

If $\Delta_k \geq 0.75$, then reduce the scale parameter:

$$\lambda_k = 1/2\lambda_k$$

else:

$$\bar{\lambda}_k = \lambda_k$$

success=false

8. If $\Delta_k < 0.25$, then increase the scale parameter:

$$\lambda_k = \lambda_k + (\delta_k(1 - \Delta_k)/|p_k|^2)$$

9. if the steepest descent direction $r_k \neq 0$, then set $k = k + 1$ and go to 2 else terminate and return w_{k+1} as desired minimum.

2.3.3 The Hessian Free Optimization algorithm

HFO is a second order optimization learning algorithm which has been introduced by [Martens \[2010\]](#) and [Martens and Sutskever \[2011\]](#). The HFO is a minimization algorithm of the twice-differentiable objective function $f : R^n \Rightarrow R$ with regards to a vector of parameters $w \in R^n$. It is based on iteratively optimizing a sequence of local quadratic approximations of an objective function in order to produce updates to the corresponding weight vector w of a predefined model. This idea was taken by the classic Newton's methods [[Robitaille et al., 1993](#)].

In the simple form of HFO learning algorithm, the iteration t produces a new weight vector w_t by minimizing the local quadratic model $M_{t-1}(\delta)$ of an objective function $f(w_{t-1} + \delta)$, which is formed using gradient and curvature information of the previous weight vector w_{t-1} . This can be seen in [Equation 11](#), where $f(w_{t-1} + \delta)$ is the local

quadratic approximation of the objective function, B_{t-1} is the curvature matrix and δ^* is the direction vector.

$$f(w_{t-1} + \delta) \cong M_{t-1}(\delta) = f(w_{t-1}) + \nabla f(w_{t-1})^T \delta + \frac{1}{2} \delta^T B_{t-1} \delta \quad (11)$$

Minimizing the quadratic model means that an optimal search direction δ^* is found which will be used to update the weight vector w_t based on *Equation 12*, where δ_t^* is the minimizer of the quadratic approximation (*Equation 11*) and $\alpha \in [0, 1]$ is the step size which is calculated by a line search method [Johansson et al., 1991; Charalambous, 1992].

$$w_t = w_{t-1} + \alpha \delta_t^* \quad (12)$$

Solving the system in *Equation 11* in order to find the minimizer δ^* is computationally impractical and sometimes even impossible because of its $O(n^3)$ complexity [Martens and Sutskever, 2011]. In order to avoid this, the linear Conjugate Gradient (CG) (*Section 2.2.3.1*) algorithm is used to partially optimize the quadratic model M . The resulting approximate minimizer δ^* is then used to update the weight vector w of the model *Equation 12*.

2.3.3.1 Conjugate Gradient for HFO

CG is a specialized optimization algorithm which has been developed specifically for quadratic objective functions of the form $q(x) = \frac{1}{2} x^T A x - b^T x$, where $A \in R^{n \times n}$ is positive definite ($x^T A x > 0 \forall$ non zero column vector x) and $b \in R^n$. During the

application of CG on a quadratic model for calculating the new weight vector w (Equation 12), the calculations of $x = \delta$, $A = Bt1$, $b = \nabla f(w_{t1})$ are needed. Also important is taking into account that the constant term $f(w_{t1})$ can be ignored.

In the worst case scenario, CG algorithm will be converged in N steps, where N is the number of model's free parameters (e.g., the length of weight vector w). Even if it does not converge, the algorithm tends to make very good partial progress [Martens and Sutskever, 2011]. More specifically, during the optimization, the preconditioning method is used to accelerate the CG convergence. This is possible by transforming the coordinate system using a preconditioning matrix P . The CG algorithm using preconditioning is described in Figure 7.

Furthermore, a number of stopping criteria are used to give a balance between the quality of a solution and the number of iterations needed to obtain an optimum solution. More specifically, the relative progress of optimizing M is measured by Equation 13, where x_j is the j^{th} iteration of CG and k is the size of the window over which the progress is calculated [Martens and Sutskever, 2011]. CG can be terminated when s_j is below some constant value (e.g., 0.0001). However, deciding when to terminate the algorithm can be an extremely complex and complicated process. Consequently, a number of more advanced stopping criteria have been proposed which have nothing to do with the value of M [Martens and Sutskever, 2011].

$$s_j = \frac{(M(x_j) - M(x_{j-k}))}{M(x_j)} \quad (13)$$

Algorithm 2 Preconditioned conjugate gradient algorithm (PCG)

inputs: b, A, x_0, P
 $r_0 \leftarrow Ax_0 - b$
 $y_0 \leftarrow \text{solution of } Py = r_0$
 $p_0 \leftarrow -y_0$
 $i \leftarrow 0$
while termination conditions do not apply **do**
 $\alpha_i \leftarrow \frac{r_i^\top y_i}{p_i^\top Ap_i}$
 $x_{i+1} \leftarrow x_i + \alpha_i p_i$
 $r_{i+1} \leftarrow r_i + \alpha_i Ap_i$
 $y_{i+1} \leftarrow \text{solution of } Py = r_{i+1}$
 $\beta_{i+1} \leftarrow \frac{r_{i+1}^\top y_{i+1}}{r_i^\top y_i}$
 $p_{i+1} \leftarrow -y_{i+1} + \beta_{i+1} p_i$
 $i \leftarrow i + 1$
end while
output: x_i

Figure 7: **The preconditioned CG**, where $x = \delta$, B is the curvature matrix, $A = B_t 1$, $b = \nabla f(w_t 1)$ and P is the preconditioning matrix [Martens and Sutskever, 2011].

2.3.3.2 Damping

The CG algorithm which is described in *Section 2.2.3.1*, requires the curvature matrix B to be positive-definite. However, in ANN the objective function is usually non-convex and B may not be positive-definite. Consequently, the minimizer of M may not exist, which means that the CG method is not applicable. Moreover, in the early stages of optimization, the minimizer δ^* of a quadratic approximation model M can be very large and aggressive, which means that it is often located far beyond the region where the quadratic approximation is reasonably trustworthy. These problems usually appear in second order optimization algorithms. The damping methods have been introduced to overcome these problems [Martens and Sutskever, 2011].

Damping methods essentially restrict the optimization of M to a trust region by augmenting M with penalty terms. These terms are designed to encourage the minimizer of M to remain in the region where M is a good approximation for the objective function.

There are a number of damping methods proposed by [Martens and Sutskever \[2011\]](#), which are directly applicable to HFO. For the purposes of this thesis, we are using the Tikhonov Damping method with Levenberg-Marquardt heuristic [[Nocedal and Wright, 2006](#)]. Tikhonov regularization or Tikhonov damping is one of the most well-known damping methods. This method is penalizing the quadratic model by introducing an additional quadratic penalty term into the quadratic model M . Thus, instead of minimizing M , we minimize a damped quadratic model. This model can be seen in [Equation 14](#), where $\hat{B} = B + \lambda I$ and $\lambda \geq 0$ are scalar parameters determining the strength of the damping.

$$\hat{M}(\delta) \equiv M(\delta) + \frac{\lambda}{2} \delta^T \delta = f(\theta)^T \delta + \frac{1}{2} \delta^T \hat{B} \delta \quad (14)$$

In [Equation 14](#), a good value of λ is critical for the success of Tikhonov damping. Too high values for λ will result in updates which resemble gradient descent with extremely small learning rate. This usually discards all the benefits of second order learning algorithms. On the other hand, too small values for λ will aggressively optimize the quadratic function which means very large weight updates that may increase the objective instead of decreasing it. Consequently, dynamically adjusting the value of λ based on Levenberg-Marquardt heuristic is addressing these issues. This heuristic defines a reduction ratio

based on *Equation 15*, where the reduction of the objective function is compared to the quadratic model.

$$\rho \equiv \frac{f(\theta_{k-1} + \delta_k) - f(\theta_{k-1})}{M_{k-1}(\delta_k)} \quad (15)$$

Furthermore, the Levenberg-Marquardt heuristic proposes two explicit rules to dynamically adapt the value of λ :

1. If $\rho > \frac{3}{4}$ then $\lambda \leftarrow \frac{2}{3\lambda}$
2. If $\rho > \frac{1}{4}$ then $\lambda \leftarrow \frac{3}{2\lambda}$
3. else $\lambda \leftarrow \lambda$

Despite the clear benefits of damping, it is important to note that they are very tricky and must be used with care. If they are overused, they produce extremely reliable updates which are simultaneously useless since they are too small. On the other hand, if they are not properly calibrated they can produce updates which give the best reductions of the objective function in early stages but may stack in local minima.

2.3.3.3 Gauss-Newton Matrix

A significant problem of CG is the use of HM as curvature matrix. The problem appears because of the inability to apply the CG algorithm to a quadratic model when the curvature matrix is not positive-definite. While the damping methods address this issue, there is a more direct solution to deal with this problem. Instead of using the HM

as the curvature matrix, another matrix can be used which is guaranteed to always be positive semi-definite. This new matrix is the generalized Gauss-Newton matrix, which is an approximation of the HM [Schraudolph, 2002]. The benefits of using this matrix do not only lie in the fact that it is always positive semi-definite but also it tends to work much better than HM in terms of efficiency and performance. This is even applied to situations where HM is positive-definite and there is no problem in using it as curvature matrix.

A combination of Gauss-Newton matrix and a damping method in the HFO learning algorithm produces much better updates for a model's weight vector w .

2.3.3.4 Evaluating the Hessian-Vector Multiplication

Based on the work of Martens [2010], no explicit evaluation and storing of HM is being done for the HFO algorithm. Instead, a dot product of a HM with the arbitrary vectors $v \in R^n$ is being computed and utilized, which cost as much as a gradient evaluation.

By the definition of directional derivatives, if we consider the HM to be the first order derivatives Jacobian matrix of the gradient, the $H(w)v$ product is the directional derivative of the gradient $\nabla f(w)$ in the direction v . $H(w)v$ product is given by Equation 16.

$$H(w)v = \lim_{\epsilon \rightarrow 0} \frac{\nabla f(w + \epsilon v) - \nabla f(w)}{\epsilon} \quad (16)$$

While this may imply a finite-differences algorithm for computing Hv at the cost of a single gradient evaluation, in practice finite-differences suffer from numerical errors, which are extremely undesirable in neural network training. Consequently, another

method is being used which avoids those errors. This method is called Forward Differentiation (FD), originally proposed by Wengert [1964] and later adjusted to neural network training by Pearlmutter [1994].

The idea behind FD is to make repeated use of the chain rule to the value of every node of the gradient, like in the BP learning algorithm. More precisely, an $R_v(X)$ operator is defined, which denotes the directional derivative of X in direction v , as shown in Equation 17.

$$R_v(X) = \lim_{\epsilon \rightarrow 0} \frac{X(w + \epsilon v) - X(\theta)}{\epsilon} = \frac{\partial X}{\partial w} v \quad (17)$$

Since the R operator is a derivative operator, it obeys the usual rules of differentiation which are shown in Equations 18-20. By applying these rules recursively to the gradient calculation algorithm, the Hv product can be efficiently computed in a way similar to BP algorithm.

$$R_v(X + Y) = R_v(X) + R_v(Y) \quad (18)$$

$$R_v(XY) = (R_v X)Y + X R_v Y \quad (19)$$

$$R_v(h(X)) = (R_v X)h'(X) \quad (20)$$

Figure 8 shows the algorithm for a simple gradient evaluation, while Figure 9 shows the modification of the gradient algorithm by applying the rules of differentiation to compute the Hv product. Similarly, Figure 10 shows the algorithm for the Gv product, which

input: $a_0 = x; \theta$ mapped to $(W_1, W_2, \dots, W_\ell, b_1, b_2, \dots, b_\ell)$.

/ Forward pass */*
for all i **from** 1 **to** ℓ **do**
 $s_i \leftarrow W_i a_{i-1} + b_i$
 $a_i \leftarrow \phi_i(s_i)$
end for

/ Loss derivative computation */*
 $\mathcal{D}a_\ell \leftarrow \left. \frac{\partial L(y, z)}{\partial z} \right|_{z=a_\ell}$

/ Backwards pass */*
for all i **from** ℓ **downto** 1 **do**
 $\mathcal{D}s_i \leftarrow \mathcal{D}a_i \odot \phi'_i(s_i)$
 $\mathcal{D}W_i \leftarrow \mathcal{D}s_i a_{i-1}^\top$
 $\mathcal{D}b_i \leftarrow \mathcal{D}s_i$
 $\mathcal{D}a_{i-1} \leftarrow W_i^\top \mathcal{D}s_i$
end for

output: $\mathcal{D}\theta$ as mapped from $(\mathcal{D}W_1, \mathcal{D}W_2, \dots, \mathcal{D}W_\ell, \mathcal{D}b_1, \mathcal{D}b_2, \dots, \mathcal{D}b_\ell)$.

Figure 8: An algorithm for computing the gradient of a FFNN, where $L(y_l; t_l)$ is one of the loss functions of Figure 11 [Martens and Sutskever, 2011]

input: v mapped to $(RW_1, \dots, RW_\ell, Rb_1, \dots, Rb_\ell)$
 $Ra_0 \leftarrow 0$ (since a_0 is not a function of the parameters)

/ Forward pass */*
for all i **from** 1 **to** ℓ **do**
 $R_s_i \leftarrow RW_i a_{i-1} + W_i R a_{i-1} + R b_i$ (product rule)
 $R a_i \leftarrow R_s_i \phi'_i(s_i)$ (chain rule)
end for

$$RD a_\ell \leftarrow R \left(\left. \frac{\partial L(y, z)}{\partial z} \right|_{z=a_\ell} \right) = \frac{\partial \left[\left. \frac{\partial L(y, z)}{\partial z} \right|_{z=a_\ell} \right]}{\partial a_\ell} R a_\ell = \left. \frac{\partial^2 L(y, z)}{\partial z^2} \right|_{z=a_\ell} R a_\ell$$

/ Backwards pass */*
for all i **from** ℓ **downto** 1 **do**
 $RD s_i \leftarrow RD a_i \odot \phi'_i(s_i) + \mathcal{D}a_i \odot R(\phi'_i(s_i))$ (product rule)
 $\quad = RD a_i \odot \phi'_i(s_i) + \mathcal{D}a_i \odot \phi''_i(s_i) \odot R s_i$ (chain rule)
 $RD W_i \leftarrow RD s_i a_{i-1}^\top + \mathcal{D}s_i R a_{i-1}^\top$ (product rule)
 $RD b_i \leftarrow RD s_i$
 $RD a_{i-1} \leftarrow RW_i^\top \mathcal{D}s_i + W_i^\top RD s_i$ (product rule)
end for

output: $H(\theta; (x, y))v$ as mapped from $(RDW_1, \dots, RDW_\ell, RD b_1, \dots, RD b_\ell)$.

Figure 9: An algorithm for computing the $H(w)v$ product in a FFNN, where $L(y_l; t_l)$ is one of the loss functions of Figure 11 [Martens and Sutskever, 2011]

input: v mapped to $(RW_1, \dots, RW_\ell, Rb_1, \dots, Rb_\ell)$
 $Ra_0 \leftarrow 0$

/ Forward pass */*
for all i from 1 to ℓ do
 $Rs_i \leftarrow RW_i a_{i-1} + W_i Ra_{i-1} + Rb_i$
 $Ra_i \leftarrow Rs_i \phi'_i(s_i)$
end for

$$RDa_\ell \leftarrow \left. \frac{\partial^2 L(y, z)}{\partial z^2} \right|_{z=a_\ell} Ra_\ell$$

/ Backwards pass */*
for all i from ℓ downto 1 do
 $RDs_i \leftarrow RDa_i \odot \phi'_i(s_i)$
 $RDW_i \leftarrow RDs_i a_{i-1}^\top$
 $RDb_i \leftarrow RDs_i$
 $RDa_{i-1} \leftarrow W_i^\top RDs_i$
end for

output: $G(\theta; (x, y))v$ as mapped from $(RDW_1, \dots, RDW_\ell, RDb_1, \dots, RDb_\ell)$.

Figure 10: An algorithm for computing the $G(w)v$ product in a FFNN, where $L(y; tl)$ is one of the loss functions of Figure 11 [Martens and Sutskever, 2011]

Name	\hat{y}	$L(y, z)$	$\nabla_z L(y, z)$	H_L
Squared error	$\hat{y} = z$	$\frac{1}{2} \ \hat{y} - y\ ^2$	$\hat{y} - y$	I
Cross-entropy	$\hat{y} = \text{Sigmoid}(z)$	$-y \log \hat{y} - (1 - y) \log(1 - \hat{y})$	$\hat{y} - y$	$\text{diag}(\hat{y} \odot (1 - \hat{y}))$
Cross-entropy (multi-dim)	$\hat{y} = \text{Softmax}(z)$	$-\sum_i [y]_i \log[\hat{y}]_i$	$\hat{y} - y$	$\text{diag}(\hat{y}) - \hat{y} \hat{y}^\top$

Figure 11: Typical loss functions with their derivatives and HMs [Martens and Sutskever, 2011]

is similar to Hv but simpler. All these algorithms use the loss functions which appear in *Figure 11*.

2.4 Protein Secondary Structure Prediction

2.4.1 Introduction

Proteins are an integral part of every living organism. In the human body, there are more than 30,000 unique proteins, which perform a vast array of important functions inside the cells. They are responsible for DNA replicating and defending against infections, as well as for many other functions required to sustain life.

They consist of organic compounds called amino acids connected to each other in long chains. Each protein differentiates from another in structure and in function, depending on the serial sequence of its amino acids. This is because the amino acids that make up a protein interact with each other, which causes the protein to fold into a specific three-dimensional structure. The structure is always the same for a specific protein, under certain conditions, and this is what determines its function.

Studying the structure and functions of proteins facilitate the process of manufacturing food supplements, drugs and antibiotics to further evolve the quality of life and healthiness of people forward. The study of existing proteins is the key for treating diseases and solving a number of biological problems, especially nowadays when technology has made the process computationally easier, faster and significantly cheaper.

In order to facilitate the process of studying proteins, a hierarchical approach has been established to better observe the structure of the proteins in the various phases of their formation. There are four layers of organization, which are the primary structure, the secondary structure, the tertiary structure and finally, the quaternary structure. The primary structure is the linear sequence of the amino acids, namely the order in which amino acids appear in the protein when unfolded. The secondary structure defines the way local segments of a protein are oriented in space, while the tertiary structure is the three-dimensional shape of a protein, when the amino acid chain is folded, and is the one that determines the specific function of a protein. Finally, a number of tertiary structures folding together forms a quaternary structure.

Despite the fact that for millions of proteins, the primary structure is well documented, only for a small fraction of those we know the secondary and tertiary structure. This is because the current state-of-the-art methodologies and instruments for protein structure determination are incredibly costly in terms of both money and time. This is incredibly serious, since the primary structure on its own, tells nothing about the actual function of the protein. This resulted in the emergence of a number of computational techniques and algorithms that attempt to predict the secondary and tertiary structure of a protein, given its primary, which do so significantly faster and cheaper.

One of those techniques used on this problem - PSSP is the use of Machine Learning algorithms. These algorithms are designed based on computational statistics and mathematical optimization techniques, which give computer systems the ability to learn patterns and idiosyncrasies of data, with the goal of being able to predict and classify new ones.

There are a number of machine learning algorithms that have been used over time on this problem, which are discussed in this thesis.

2.4.2 Proteins

Proteins are large, complex molecules made up of hundreds to thousands of smaller units called amino acids, which are attached to one another in long chains. Proteins are responsible for most of the functions within organisms and this is what classifies each protein into a specific type. For example, there are structural proteins, which strengthen cells, tissues and organs and defense proteins, namely the antibodies, which help organisms fight infection, heal damaged tissue and evade predators.

In the human body, proteins are created mostly through the consumption of foods. When food, which contains proteins, is consumed, the digestive system breaks it down into amino acids, which enter the blood stream. The cells then gather the necessary amino acids from the blood stream, to create the proteins it requires to perform any of the vast array of functions possible. A diet poor of proteins results in few amino acids entering the blood stream which weakens the immune system, causes exhaustion, dizziness and possibly a number of other very serious diseases. This is because the cells do not have enough amino acids to create the proteins required for each of the functions necessary to sustain the human body.

Consequently, understanding the significant role of proteins in all aspects of living organisms is important. However, what is necessary is to understand the core structure

and function of each protein, in order to facilitate the process of creating food supplements, drugs and antibiotics to further evolve the quality of life and healthiness of people forward. The study of existing proteins, is the key for treating diseases and solving a number of biological problems, especially nowadays when technology has made the process computationally easier and significantly faster.

Amino acids, or as they are often called, the building blocks of life are the sole component of proteins. There are more than five hundred (500) naturally occurring amino acids known, but only twenty (20) appear in the genetic code and in the formation of proteins (*Figure 13*). Consequently, those amino acids are called the essential amino acids and are found in most, but not in all proteins.

All amino acids are composed by one functional group of amine (-NH₂) and carboxyl (-COOH), along with a side chain, the R group, specific to each amino acid. The unique side chain is what differentiates amino acids in their physical and chemical properties. Moreover, depending on the chemistry of their side chain, amino acids are classified into three (3) different categories. The first and largest group of amino acids has nonpolar side chains, while the second has polar side chains, which are uncharged. The third one has amino acids with positive and negative charges on their side chain. This is extremely critical to the protein structure, since these side chains can interact and bond with one another based on their chemistry, which forms the specific part of the protein in a certain shape. This means that the sequence and location of amino acids in a particular protein determines where the bends and folds occur in its three-dimensional structure. Finally, every single amino acid has its amino group positively charged and its carboxylic group

No.	Amino acid	3-Letter code	1-Letter code	Circles' height	Circles' radii
1	Alanine	Ala	A	0.0	1.0
2	Arginine	Arg	R	0.05	0.95
3	Asparagine	Asn	N	0.10	0.90
4	Aspartic acid	Asp	D	0.15	0.85
5	Cysteine	Cys	C	0.20	0.80
6	Glutamine	Gln	Q	0.25	0.75
7	Glutamic acid	Glu	E	0.30	0.70
8	Glycine	Gly	G	0.35	0.65
9	Histidine	His	H	0.40	0.60
10	Isoleucine	Ile	I	0.45	0.55
11	Leucine	Leu	L	0.50	0.50
12	Lysine	Lys	K	0.55	0.45
13	Methionine	Met	M	0.60	0.40
14	Phenylalanine	Phe	F	0.65	0.35
15	Proline	Pro	P	0.70	0.30
16	Serine	Ser	S	0.75	0.25
17	Threonine	Thr	T	0.80	0.20
18	Tryptophan	Trp	W	0.85	0.15
19	Tyrosine	Tyr	Y	0.90	0.10
20	Valine	Val	V	0.95	0.05

Figure 12: List of all the 20 essential amino acids [Abo-Elkhier, 2012].

negatively charged. This facilitates the sequential connection between amino acids with covalent bonds.

The way amino acids connect to each other is by peptide bonds, in units as small as two or three amino acids, called dipeptides and tripeptides respectively, or in much longer chains called polypeptides, forming a protein molecule. This process is called condensation reaction and it extracts a water molecule as it joins the amino group of one amino acid and the carboxyl group of a neighboring amino acid. What remains of each amino acid after the junction, is called amino acid residue.

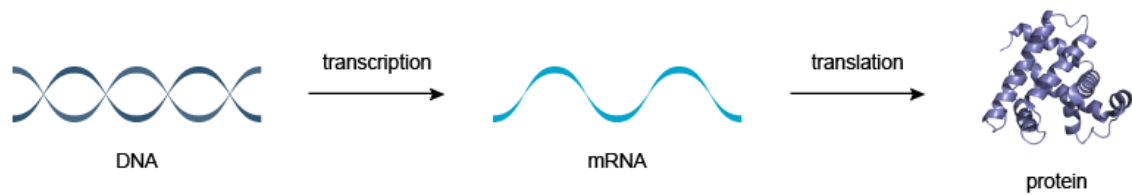


Figure 13: The Central Dogma of Molecular Biology: DNA makes RNA makes proteins (Nucleic Acids Book, www.atdbio.com, 2018, May 5)

Each amino acid is abbreviated into a single (or triple) character from the English alphabet, meaning the amino acid sequence of a polypeptide can be represented as a sequence of characters. This sequence is considered to be the primary structure of the protein. As a result, any change in the sequence of the polypeptide, leads to the formation of a completely different protein, along with a completely different set of properties and functionalities.

The way each protein is assembled is encoded in the genes of an organism, the DNA. More specifically, the unique amino acid sequence, which forms a protein, is specified by the nucleotide sequence of the gene encoding that protein. In the case of the human genome, there are around thirty-thousand (30,000) genes, each of which encodes a single, unique protein.

The way it works is that the DNA makes RNA through a process called transcription and the RNA makes proteins through a process called translation. This constitutes The Central Dogma of Molecular Biology, which is illustrated in *Figure 13*.

The genetic code is basically a set of nucleotide triplets, called codons. Each combination of a triplet designates an amino acid, and since there are four (4) unique nucleotides (adenine - A, uracil - U, guanine - G, and cytosine -C), the total number of triplets that

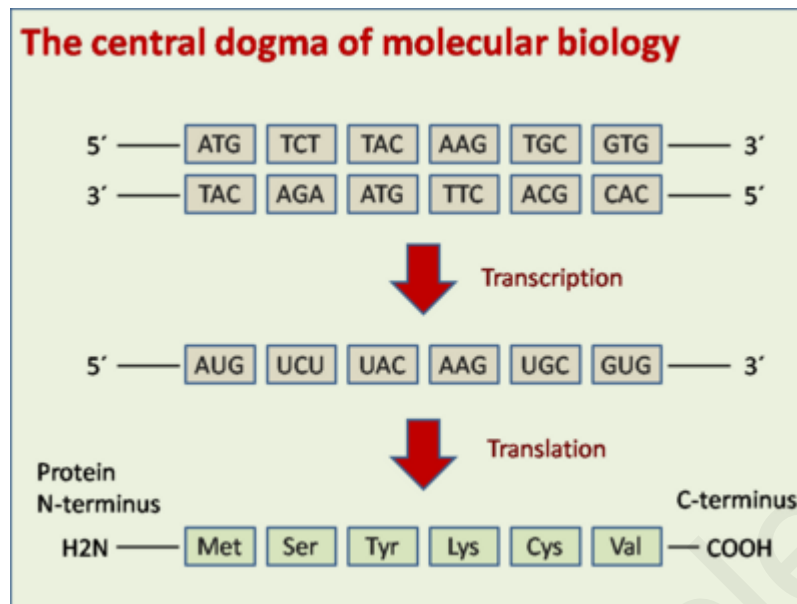


Figure 14: Example of the central dogma. The first few amino acids for the alpha subunit of hemoglobin [Bolsover et al., 2004].

can be arranged is sixty-four ($4^3 = 64$). However, there are only twenty (20) amino acids that can be encoded naturally, which means some amino acids can be described by more than one codon, or some codons do not encode any amino acids. Those codons, which do not encode any amino acids, are called the stop codons and serve as a termination signal for the translation process, meaning that when one is found, the polypeptide, or the protein, translated up to that point is released. *Figure 14* illustrates an example of the translation from DNA to protein (the first few amino acids for the alpha subunit of the protein hemoglobin), while *Figure 15* examines the full table of codons, along with the amino acid or the stop signal they encode.

In order to facilitate the process of studying proteins, a hierarchical approach has been established to better observe the structure of the proteins in the various phases of their formation. There are four layers of organization, which are the primary structure,

		Second codon position					
		U	C	A	G		
First codon position	U	Phe Leu Leu	Ser Ser	Tyr Stop Stop	Cys Stop Trp	Third codon position	U C A G
	C	Leu	Pro	His Gln	Arg		U C A G
	A	Ile Ile Met, fMet	Thr Thr	Asn Lys	Ser Arg		U C A G
	G	Val Val	Ala	Asp Glu	Gly Gly		U C A G



 Number of tRNA species

Figure 15: The amino acids specified by each codon [Alkatib et al., 2012].

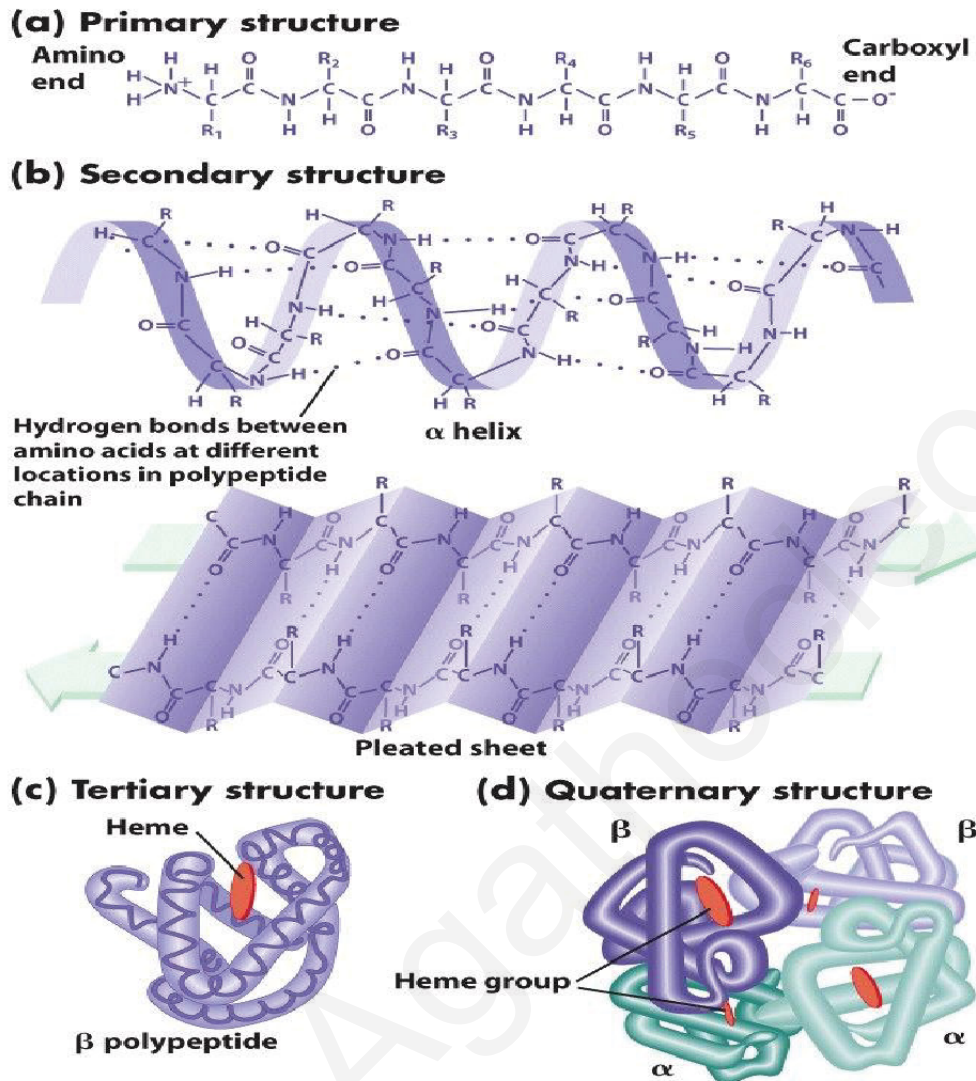


Figure 16: Layers of protein structure [Gupta et al., 2017].

the secondary structure, the tertiary structure and finally, the quaternary structure. It is important to note that this organization of many hierarchical structures is strictly used to make things easier for people to understand how proteins are formed. In organisms, proteins have one single structure, which is three-dimensional.

2.4.3 Evaluation Metrics

The most informative metrics to evaluate PSSP related models are per residue accuracy Q_3 and Segment Overlap (SOV) score [Rost et al., 1994; Zemla et al., 1999].

More specifically, the Q_3 metric is defined as the three-state overall percentage of correctly predicted residues:

$$Q_3 = 100 \frac{1}{N_{res}} \sum_i M_{ii}, \quad (21)$$

where N_{res} is the total number of residues and M_{ij} is the number of residues observed in state i and predicted in state j , with i and $j \in \{H, E, L\}$ (i.e. M_{ii} is the number of residues predicted correctly in state i).

Additionally, we can calculate the per-state accuracy, as the percentage of correctly predicted residues in a particular state:

$$Q_i = 100 \frac{M_{ii}}{obs^i} \quad (22)$$

where obs^i is the number of residues observed in state i .

Furthermore, the SOV score [Zemla et al., 1999] is defined as a measure that is based on the average overlap between the observed and the predicted segments instead of the average per-residue accuracy:

$$SOV_i = 100X \frac{1}{N_i} \sum_{s_i} \frac{\min ov(s_1, s_2) + \delta(s_1, s_2)}{\max ov(s_1, s_2)} X len(s_1) \quad (23)$$

where s_1 and s_2 are segments of secondary structure in conformational state i (i.e. H, E, or C), $len(s_1)$ is the number of residues in segments, $minov(s_1, s_2)$ is the length of the actual overlap of s_1 and s_2 , i.e. for which both segments have residues in state i , $maxov(s_1, s_2)$ is the total extent for which either of the segments s_1 and s_2 has a residue in state i . The normalization value N_i is defined as:

$$N_i = \sum_{s_i} len(s_1) + \sum_{s'_i} len(s_1) \quad (24)$$

and $\delta(s_1, s_2)$ is defined as:

$$\delta(s_1, s_2) = min(maxov(s_1, s_2) - minov(s_1, s_2)) \quad (25)$$

Finally, the Matthews correlation coefficient [Matthews, 1975], C_i , provides a measure for the performance at each state:

$$C_i = \frac{p_i n_i - u_i o_i}{\sqrt{(p_i + u_i)(p_i + o_i)(n_i + u_i)(n_i + o_i)}} \quad (26)$$

$$\text{with } p_i = M_{ii}, \quad n_i = \sum_{j \neq i} \sum_{k \neq i} M_{jk},$$

$$o_i = \sum_{j \neq i} M_{ji} \quad \text{and} \quad u_i = \sum_{j \neq i} M_{ij}.$$

2.4.4 Protein Secondary Structure Prediction

The primary structure of a protein is the discrete sequence of amino acids, which is basically the linear succession of amino acids in the protein, if its three-dimensional

structure was to be unfolded. Using the one-character amino acid abbreviations, countless possibilities of protein formation exist. However, only a tiny subset of them has actually been studied and most of the information that exists today about proteins is about their primary structure. This is because the primary structure of a protein can easily be translated from the genetic material, though no useful information regarding its function can be extracted from it. However, various learning algorithms can be applied to it, to accurately predict cheaply its secondary and tertiary structure, which is also the main focus of this dissertation.

The secondary structure is the three-dimensional form of local segments of proteins. The most common method of describing the secondary structure of proteins was defined by the Dictionary of Protein Secondary Structure, or Dictionary for Secondary Structure of Proteins (DSSP) [Kabsch and Sander, 1983] in short. Single character codes are used, based on hydrogen bond patterns, to define the eight (8) types of secondary structure that the DSSP classifies. These are the α -helix (H), 3-helix (G), π -helix (I), β -strand (E), β -bridge (B), β -turn (T), bend (S), and random coil (C) for residues which are not in any of the other conformations. This last designation is unfortunate as no portion of protein three-dimensional structure is truly random and it is usually not a coil. A number of "other" secondary structures types have been proposed; however, they represent a small fraction of residues and may not be a general structural principle of proteins. It is common to group these eight (8) categories into three (3) to describe the nature of the shape of the specific local segment of the protein. First, the helix conformations that obviously contain the first three categories (H, G, I), and have helical form, the sheet conformations

that contain the β -strand (E) and β -bridge (B) categories, and finally Coil conformations which contain everything else.

The tertiary structure is the way the polypeptide chain coils and turns to form a complex molecular three-dimensional shape. This structure is what actually defines the functions and properties of the protein. Despite its great significance only for a very small portion of known proteins, there is a documented and fully defined tertiary structure. This is because of the very expensive experimental procedures required and it is still today a very important problem. Under certain conditions, such as protein temperature or pH change, the original three-dimensional structure is destroyed and its properties and biological functions are altered, despite of the fact that the amino acid sequence is still the same. This confirms that the 3D structure of the protein is what defines its function and not the amino acid sequence it is made up of. However, under normal conditions, both secondary and tertiary structures remain the same for each protein, since the linear sequence of amino acids (primary structure) is always the same and the following structures are developed through the interactions between the R groups of the amino acids. The layers of protein structure can be seen in *Figure 16*.

The prediction of a protein's SS from its Primary Structure (PS) can be an important intermediate step to the prediction of a protein's three-dimensional (3D) structure [Yue and Dill, 2000; PalÁz et al., 2004]. A protein's PS is a sequence composed of 20 different amino acid types which are connected and interact to create the SS, the local (geometrical) structural patterns defined by hydrogen bonding patterns which may be short, mid- or even long-range. When an experimentally-determined 3D structure is available, each

amino acid can be assigned to a SS class, usually under a commonly accepted scheme: helix (*H*), extended (*E*) and coil/loops (*L*) [Kabsch and Sander, 1983].

Knowledge of a protein's three-dimensional (3D) structure can be an important step in studying the functional properties of protein molecules, which are the functional workhorses in all living cells. Experimental biochemical methods for the characterization of the molecular structures of individual proteins in atomic detail are expensive, time consuming and -frequently- inefficient [Baldi et al., 1999]. Since genomic technologies provide genetic sequences at an ever increasing pace, the gap between our knowledge of protein sequences (primary structures) and the corresponding experimentally determined 3D structures is widening exponentially. Even though it has been reported that a high fraction of residues from proteins encoded in the human genome and other model species can be mapped to a 3D structure (either by experimental or theoretical methods) [Schwede, 2013], the same does not necessarily hold for non-model species, especially for some "exotic" pathogens, whose genomes often encode protein molecules with peculiar sequence features. For example, of the >5000 proteins encoded in the malaria-causing parasite *Plasmodium falciparum* isolate 3D7 for which no experimental structural information exists in the Protein Databank, less than half (2459) can be mapped onto 3D structures using sequence similarity methods (source UniProt: <https://www.uniprot.org/uniprot/> accessed May 9 2019).

2.4.5 Related work

There is more than half a century's worth of work on the PSSP problem. A number of machine learning algorithms have been developed and optimized for this specific problem over the years, which resulted recently in accuracies $>90\%$ [Magnan and Baldi, 2014] in the Q3 accuracy score. However, the algorithms that managed to achieve such high accuracies ($>85\%$) have all used additional information and structural templates from databases, called sequence-based structural similarity of a protein. This makes the learning process and performance much better, relative to the more pure machine learning algorithms. Without relying on these structural templates, the three-state accuracy is now at 82-84%, which is still good, considering the complexity of the problem. There is still room for improvement, however, considering the theoretical limit of the three state prediction of around 88-90% [Magnan and Baldi, 2014].

Observing the *Figure 17* it is clear that despite its long history it was only until the 90's that PSSP started getting more attention. That is because some major breakthroughs were achieved during that period which resulted in gradually increasing the three-state accuracy of the problem significantly.

Over the past 30 years, the accuracy for secondary structure predictive methodologies has improved significantly with machine learning techniques [Qian and Sejnowski, 1988] and evolutionary information from multiple sequence alignments [Rost and Sander, 1993] having a crucial role. Several artificial neural network (ANN) architectures have

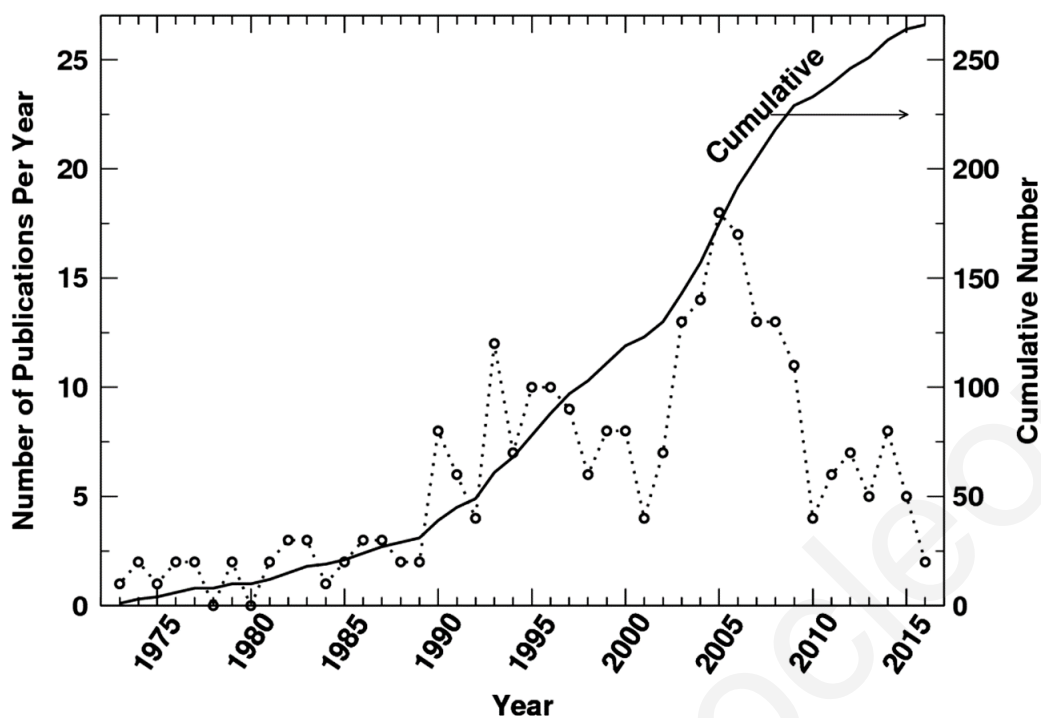


Figure 17: Number of publications for PSSP per year [Yang et al., 2016]

been used, such as feed-forward ANNs [Rost and Sander, 1993], [Jones, 1999], bidirectional recurrent ANNs (BRNNs) [Baldi et al., 1999], [Pollastri et al., 2002], [Pollastri and McLysaght, 2005] and cascade-correlation ANNs [Wood and Hirst, 2005][Chen and Chaudhari, 2007], whilst SVMs have been proven successful over the past decade [Kieslich et al., 2016], [Hua and Sun, 2001], [Karypis, 2006], [Kountouris and Hirst, 2009]. Other methods used hidden Markov models (HMMs) [Karplus et al., 1999], [Lin et al., 2005], multiple linear regression [Pan, 2001], [He and Pan, 2005] and nonlinear dynamic systems [Green et al., 2009], whereas methods like JPred [Cuff et al., 1998] make consensus secondary structure prediction. More recently, knowledge-based methods, such as PROTEUS [Montgomerie et al., 2006] and HYPROSP [Wu et al., 2004], utilised structural information, whilst the predictive accuracy was further improved through the

use of remote homology information [Mooney and Pollastri, 2009]. A brief review of these methodologies can be found in Kountouris and Hirst [2009]. Furthermore, a more recent review presents all the latest developments on the field of PSSP which have achieved more than 80% per residue accuracy [Yang et al., 2016]. These results rely on increasingly larger databases of protein sequences, the use of templates and powerful deep learning models. More specifically, a method based on Deep Convolutional Neural Fields (based on Convolutional Neural Networks) [Wang et al., 2016] and the MUFold-SS method [Fang et al., 2018] have shown promising results. Finally, the work of Heffernan et al. [2017], which is based on LSTM [Hochreiter and Schmidhuber, 1997] BRNNs, has shown an 84% Q3 accuracy which is one of the highest scores reported so far [Yang et al., 2016]. These methods can improve one of the most common drawback of the majority of PSSP methods, i.e., the failure to recognise long-range interactions between β -strands leading to the formation of β -sheets. Moreover, recent research efforts demonstrate evidence that this issue may have arisen by the use of small and/or non-representative datasets [Kieslich et al., 2016]. Current state-of-the-art methods for predicting protein SS from PS are based on ML classifiers fed with sequence profile inputs and achieve around 82%-84% Q3, whereas the SOV metric [Zemla et al., 1999; Zhang et al., 2011] is circa 73-75.

Furthermore, several PSSP prediction methodologies have used a multi-step process with ensemble methods [Dietterich, 2000; Zhou et al., 2002, 2010; Li et al., 2018a; Zheng et al., 2019] and filtering techniques [Kountouris et al., 2012] to improve the quality of

results. Ensembles reduce the mis-predicted residues by averaging the results of multiple classifiers. For example, these methods show improved generalization capabilities that outperform those of single networks [Granitto et al., 2005a]. However, for aggregation to be effective, the individual networks must be as accurate and diverse as possible. The seminal work of Baldi et al. [1999], Pollastri et al. [2002] have shown results on the SSpro method where they have used an ensemble of 11 BRNNs to achieve a prediction accuracy of 78%. Moreover, the SCRATCH server uses the SSpro 5 method trained on newer datasets to achieve an accuracy of 79% [Cheng et al., 2005; Magnan and Baldi, 2014]. This method has been trained and validated on a large dataset of approximately 11000 sequence profiles, using an elaborate training process where individual BRNNs were trained on different subsets of the training data. Furthermore, this method uses an ensemble of 100 BRNNs. With SSpro 5 these authors introduced an additional step of using the observed secondary structure from homologs in the PDB to infer the final prediction, reaching an accuracy of approximately 92%. In addition, filtering techniques remove conformations that are physicochemically unlikely. For instance, helical conformations in proteins usually consist of at least three, four or five residues for 3_{10} -helix, α -helix and π -helix, respectively. Since the different types of helices are usually grouped in a single category by PSSP methods, a predicted helical structure would be expected to have a minimum number of three consecutive residues in order to fulfill geometric and hydrogen-bonding requirements. Hence, predictions of isolated helical residues are physicochemically unrealistic, because one residue cannot form a helix. To tackle this problem, both machine learning algorithms [Chen and Chaudhari, 2007] and empirical

rules have been used in the past [Rost and Sander, 1993; Salamov and Solovyev, 1995; Wood and Hirst, 2005] with variable levels of success, highlighted in a comparative study [Kountouris et al., 2012].

2.4.6 Data

High quality datasets for training and validation purposes are mandatory when constructing a prediction model. For the purposes of this thesis, we have used the CB513 dataset [Cuff and Barton, 1999], which is a non-redundant dataset consisting of 513 protein sequences and has been heavily used as a PSSP benchmark dataset. Furthermore, we have also used a much bigger non-redundant dataset called PISCES [Wang and Dunbrack Jr, 2003; Kieslich et al., 2016], consisting of 8632 protein chains.

Knowledge of a protein's three-dimensional (3D) structure can be an important step in studying the functional properties of protein molecules, which are the functional workhorses in all living cells. Experimental biochemical methods for the characterization in atomic detail of the molecular structures of individual proteins are expensive, time consuming and -frequently- inefficient [Baldi et al., 1999]. Since genomic technologies provide genetic sequences at an ever increasing pace, the gap between our knowledge of protein sequences (primary structures, PS) and the corresponding experimentally determined 3D structures is widening exponentially. Even though it has been reported that a high fraction of residues from proteins encoded in the human genome and other model species can be mapped to a 3D structure (either by experimental or theoretical methods) [Schwede,

2013] the same does not necessarily hold for non-model species, especially for some “exotic” pathogens, whose genomes often encode protein molecules with peculiar sequence features. For example, of the >5000 proteins encoded in the malaria-causing parasite *Plasmodium falciparum* isolate 3D7 for which no experimental structural information exists in the Protein Databank, less than half (2459) can be mapped onto 3D structures using sequence similarity methods (source UniProt: <https://www.uniprot.org/uniprot/> accessed May 9 2019).

As an intermediate step for elucidating protein structures at atomic resolution (or even as a compromise to the lack of such information), protein secondary structures (SS) may provide lower resolution structural information. Protein secondary structure roughly refers to (mostly) local geometrical structural patterns formed by the folded polypeptide chains (linear polymers typically composed of different combination of 20 amino acid residues). At the molecular level SS elements are usually defined by short-, mid- or even long-range hydrogen bonding patterns between amino acid residues within the same or (less frequently) different polypeptide chains. A commonly accepted scheme assigns amino acid residues from an experimentally derived 3D structure into three SS classes: helix (H), extended (E) and coil/loops (L).

Such information can be very useful both for providing constraints in efforts for predicting protein 3D structure, or as low-resolution -yet useful in several practical aspects- structural information. Current state-of-the-art methods for predicting protein SS from PS are based on ML classifiers fed with sequence profile inputs and achieve 75-80% per residue accuracy (Q3); consensus or ensemble methods raise this figure to a few points

higher than 80% [Kieslich et al., 2016]. The most common drawback of the majority of PSSP methods is the failure to recognise long-range interactions between β -strands leading to the formation of β -sheets, even though recent research efforts demonstrate evidence that this issue may have arisen by the use of small and/or non-representative datasets.

2.4.6.1 Sequence Databases

There are millions of documented proteins in various protein databases such as Protein Information Resource (iProClass), Protein Data bank in Europe (PDBe), Protein Data bank in Japan (PDBj) and RCSB Protein Data Bank. In those databases, information regarding protein names, length, structures (primary, secondary, tertiary and quaternary) exists, as well as many other biological information related to proteins. Those databases were used to extract protein information to create the datasets used in PSSP.

2.4.6.2 Data Selection Criteria

Initially, we reduce redundancy of sequence datasets based on sequence similarity using Cd-hit [Li and Godzik, 2006], thus keeping only representative sequences for obtaining meaningful training and validation datasets.

It is of great importance to choose a suitable data set so that our models can be properly trained, and several criteria should be fulfilled for the selection process:

1. We retained only entries determined by X-ray diffraction, for which we can apply the resolution as a quantitative selection measure. In particular, a 3.0 Å threshold was used to discard structures of insufficient/questionable quality.

2. We discarded entries with physical chain breaks, as empirically identified by at least one pair of successive $C\alpha$ atoms with a distance longer than 4.0 \AA , as well as proteins with large segments of undefined secondary structure.
3. Chains with a length of less than 30 amino acids were discarded. This rule has been set after some initial experiments where we have noticed that chains with less than 30 amino acids are predicted with almost 100% Q_3 accuracy. This usually increases the overall results in a way that the Q_3 accuracy is not representative for a specific dataset. This rule has been also used by Baldi et al. [1999].
4. DSSP [Kabsch and Sander, 1983] (see below), should provide a valid output file for any chain retained in the dataset.

2.4.6.3 Secondary Structure Assignment

The DSSP [Kabsch and Sander, 1983] (URL: <http://swift.cmbi.ru.nl/gv/dssp/>, accessed 28/01/2018) defined a standardized format of categorizing the secondary structures of a protein. In this format, there are eight (8) different classes of secondary structures, based on their shape and they are represented by a capital English letter. There are the H (α -helix), G (3_{10} -helix), I (π -helix), E (extended β -strand), B (isolated β -bridge), T (turn), S (bend) and C (other/coil) (Figure 18) for residues which are not in any of the other conformations. This last designation is unfortunate as no portion of protein three-dimensional structure is truly random and it is usually not a coil. A number of "other" secondary structures types have been proposed; however, they represent a small fraction

of residues and may not be a general structural principle of proteins. It is common to group these eight (8) categories into three (3) to describe the nature of the shape of the specific local segment of the protein, which is the way they are categorized in this dissertation. More specifically, we reduce class assignments from the eight secondary structure (SS) types provided by DSSP (i.e. α -helix (H), 3_{10} -helix (G), π -helix (I), β -strand (E), β -bridge (B), β -turn (T), bend (S) and 'other' (')) into three SS states (Helical: H, G; Extended: E, B; Random coil/Loop: I, T, S, '.'). From here onwards, we refer to these states as H, E and C respectively. DSSP results were fetched from the DSSP website (URL: <http://swift.cmbi.kun.nl/gv/dssp/>, accessed 20 April 2009) and transformed to the 3-state representation by an in-house parser. Since several protein chains contain segments of disordered regions where DSSP does not produce any output, for the purposes of this work we have decided to exclude any such entries.

2.4.6.4 Multiple Sequence Alignment Preprocessing

Furthermore, Multiple sequence alignment (MSA) profiles have been used for data preprocessing and PS encoding [Rost, 1996]. MSA profiles have been shown to enhance machine learning-based PSSP, since they incorporate useful evolutionary information for the encoding of each position of a protein.

MSAs have been shown to significantly increase protein secondary structure prediction accuracy in recent applications [Rost and Sander, 1994]. This is because structure is considered to be more conserved than sequence [Rost and Sander, 1994]. Every position within an alignment contains an evolutionary record. We encode each input residue

with a 20-dimensional vector, where ordinates correspond to the frequencies of the different 20 amino acid residues at the respective column of the MSA. Apparently, encoding for single sequences at the input reduces to the orthogonal encoding scheme. For the polypeptide chains collected with the previously described procedure, we have utilised unweighted profiles available from the HSSP database [Schneider and Sander, 1996] and from the PSI-BLAST [Altschul et al., 1997] search against the NCBI-NR (NCBI: <http://www.ncbi.nlm.nih.gov/>) database.

2.4.6.5 The CB513 Dataset

CB513 dataset constitutes the major dataset of this thesis. It has been used as a benchmark dataset in all the methods appeared in this work. The origin of the CB513 [Cuff and Barton, 1999] dataset was the dataset of Heinz-Uwe Hobohm (Pdb Select25, 2009) in 2009. This dataset originally contained 4019 proteins, with maximum similarity per protein pair of 25%. This is incredibly important in order to avoid a problem called selection bias, where the data sample is not truly random and there is no even representation of all classes of the problem. In selection bias, the trained model learns some classes better than others, which results in poor classification/prediction on patterns in the testing set, which belong to a poorly represented class on the training dataset.

From the initial 4019 proteins, only 513 finally remained. This is due to three main reasons. First, proteins had to be in the PDB database and be encoded in the DSSP format. Second, the secondary structure of those proteins should have been determined by the X-Ray crystallography method or by the Nuclear Magnetic Resonance (NMR)

Secondary Structure	8 class code	3 class code
α -helix	H	H
3-helix	G	
π -helix	I	
β -strand	E	E
β -bridge	B	
β -turn	T	C
bend	S	
Random coil	C	

Figure 18: Matrix with the abbreviations of the secondary structures grouped in 8 and 3 classes

method. Finally, there was some additional specific requirements, regarding both the structure of amino acids in a protein as well as the clarity of the structure determination by the X-Rays, Those conditions had to be set and followed, in order to create a dataset which would actually be useful for the PSSP problem without negatively influencing the classifications.

2.4.6.6 The PISCES Dataset

PISCES dataset [Kieslich et al., 2016] has also been used in this thesis. This dataset contains 8632 protein sequences which have been retrieved from the PISCES protein culling server [Wang and Dunbrack Jr, 2003]. The sequence identities are obtained from PSI-BLAST alignments with position-specific substitution matrices derived from the non-redundant protein sequence database.

PISCES dataset is based on a percent identity cutoff of 25% and resolution cutoff of 3.0 . Furthermore, an R-factor cutoff of 1.0 was used for further evaluation of the robustness of developed models.

Chapter 3

BRNN input sliding window and post-processing methods

3.1 Introduction

Successful protein secondary structure prediction is an important step towards modelling protein 3D structure, with several practical applications. In the last four decades several PSSP algorithms have been proposed but still there is room for improvement. This improvement can be even achieved from enhancements on models or application of ML techniques on existing methods.

The BRNN architecture of [Baldi et al., 1999] is considered as one of the optimal computational neural network type architectures for addressing the problem. More specifically, in this chapter, we present a variation of the Bidirectional Recurrent Neural Network (BRNN) which was initially developed by Baldi et al. [1999]. We implement the same BRNN architecture, but we use a modified training procedure. More specifically,

our aim is to identify the effect of the contribution of local versus global information on the PSSP problem, by varying the length of the segment on which the Recurrent Neural Networks operate for each residue position considered. Our results with a single BRNN are better than Baldi et al. [1999] by three percentage points (Q3) and comparable to ensembles of BRNN models' results which appear in Baldi et al. [1999]; Pollastri et al. [2002]; Cheng et al. [2005]; Magnan and Baldi [2014]. Moreover, we have used ensembles of 6 BRNNs to enhance our results. In addition, our results improve even further when sequence-to-structure output is filtered in a post-processing step, with a novel Hidden Markov Model-based approach. Filtering of protein secondary structure prediction aims to provide physicochemically realistic results, while it usually improves the predictive performance. We performed a comparative study on this challenging problem, utilising both machine learning techniques and empirical rules and we found that combinations of the two lead to the highest improvement.

3.2 Training BRNN with weight updating for each residue

We have implemented the same BRNN architecture as Baldi et al. [1999], but we have used a modified training procedure. More specifically, our aim is to identify the effect of the contribution of local versus global information, by varying the length of the segment on which the Recurrent Neural Networks operate for each residue position considered. For training the network, the backpropagation learning algorithm with an online training procedure is used, where the weight updates occur for every amino acid, as opposed to [Baldi et al., 1999], where the weight updates are applied after the presentation of

the entire protein. Our results with a single BRNN are better than [Baldi et al., 1999] by three percentage points (Q3) and comparable to results of [Baldi et al., 1999] when they use an ensemble of 6 BRNNs. In addition, our results improve even further when sequence-to-structure output is filtered in a post-processing step, with a novel Hidden Markov Model-based approach.

In the introductory work of BRNNs for PSSP [Baldi et al., 1999], for each position in a sequence presented to the network, an input window is formed centered around this position. During the training phase, the protein is processed in its entirety before the weight updates are made. In this work, our aim is to investigate how the prediction accuracy could be improved by: (i) updating the weights at every residue, which in a way constitutes a form of dynamic training and is more context-sensitive, and (ii) using different filtering approaches (a novel method based on Hidden Markov Models and a cascaded feed forward Artificial NN).

3.2.1 Data Collection and Preprocessing

In order to train and validate our models, we need a set of high quality data consisting of proteins with experimentally determined 3D structures deposited in the RSCB Protein Data Bank (PDB) (URL: <http://www.pdb.org/>, accessed 20 April 2009). Moreover, the resulting dataset should be maximal (in order to capture the knowledge we currently have available on protein structures) but also non-redundant, to avoid poor generalisation of the BRNN. For this purpose we utilised the PDBSelect25 dataset (URL: <http://bioinfo.tg.fh-giessen.de/pdbselect/>, accessed 20 April 2009), which is regularly produced by analysis

of the PDB with the algorithm described in [Hobohm et al., 1992]. This dataset contained 4019 polypeptide chains that shared less than 25% overall pair-wise sequence identity. These data have been preprocessed based on the methods described in *Sections 2.3.6.1-2.3.6.4*. Consequently, we started with a dataset containing a total of 4019 protein chains, of which 2656 corresponded to structures determined by X-ray crystallography. Following the above procedure we ended up with 612 protein chains.

3.2.2 Modified Training Procedure for the BRNN Architecture

A NN must accept the amino acid at its input with all the necessary information that it needs in order to produce the right output. Taking into account that the formation of different secondary structural elements depends on the interaction between neighboring-in-space (not necessarily in sequence) amino acid residues, we chose the BRNN architecture for its ability to encapsulate information included in the amino acid residues that are coming before and after the residue at the examined position t ; where t denotes the discrete time index in $[1, T]$, with T being the total length of the protein chain.

The BRNN we have used in this work is a modified version of the one shown in *Figure 3 (Section 2.1.3)*. The RNNs are used for taking into account the information contained in a local segment of length L_s centered around position t . The FRNN processes $j = (L_s - 1)/2$ amino acid residues located on the left side of the segment, computing iteratively from the far left side of the window (i.e., in position $t - j$) and moving towards the right until position t (inclusive) by taking into account a sliding sub-segment of length Sl . The BwRNN processes the amino acids located on the right side of t , in

a similar symmetric way. During the recurrent network processing, a kind of memory is being formed since the NN correlates each sequence separately and holds an internal temporary knowledge [Elman, 1990]. The output from the two recurrent NNs and the output from the FFNN are correlated and predict the secondary structure state for residue t as indicated in Equation 27.

$$O_t = \eta(F_t, B_t, I_t) \quad (27)$$

where $\eta(\cdot)$ is realised by the FFNN, F_t is the forward (upstream) context, B_t is the backward (downstream) context and I_t is the input vector at time (sequence position) t . In the current work, we use an input vector encoding a single residue (corresponding to a window size of unity in Baldi's [Baldi et al., 1999] implementation). The contextual information from the protein is gathered into a pair of vectors F_t and B_t . Only after the F_t and B_t are computed, the algorithm can predict the state (as in [Baldi et al., 1999]). In order for the amino acids to be examined, two learnable non-linear state transition functions $\phi()$ and $\beta()$ are applied. Algorithmically this is shown in Figure 19.

Once the data located within the input vector enters the BRNN, the Mean Square Error function is applied and is used by the Backpropagation algorithm [Rumelhart et al., 1986a] for the BRNN to be trained. Training is performed based on two alternative output encoding schemes: (i) an orthogonal, and (ii) a "winner-take-all" (WTA). The former scheme has three output units with binary values giving eight possible combinations, three of which are assigned to the three reduced SS states, and the rest are arbitrarily considered to be classified as random coil. The WTA encoding scheme, has three output units as well


```

1: for every sequence do
2:   for  $t = 1$  to  $T$  do
3:     for  $i_f = t - \frac{L_s-1}{2}$  to  $t$  do
4:       for  $i'_f = i_f - S_l$  to  $i_f - 1$  do
5:         if  $i'_f < t - \frac{L_s-1}{2}$  then
6:            $F_{i'_f} \leftarrow 0$ 
7:         end if
8:       end for
9:        $F_{i_f} \leftarrow \phi_t(\gamma F_{i_f-S_l}, \gamma F_{i_f-S_l+1}, \dots, \gamma F_{i_f-1}, I_{i_f})$ 
10:    end for
11:    for  $i_b = t + \frac{L_s-1}{2}$  to  $t$  step -1 do
12:      for  $i'_b = i_b + S_l$  to  $i_b + 1$  step -1 do
13:        if  $i'_b > t + \frac{L_s-1}{2}$  then
14:           $B_{i'_b} \leftarrow 0$ 
15:        end if
16:      end for
17:       $B_{i_b} \leftarrow \beta_t(\gamma^{-1} B_{i_b+S_l}, \gamma^{-1} B_{i_b+S_l-1}, \dots, \gamma^{-1} B_{i_b+1}, I_{i_b})$ 
18:    end for
19:     $O_t = \eta_t(F_t, B_t, I_t)$ 
20:    update  $\phi_t, \beta_t, \eta_t$ 
21:  end for
22: end for

```

Figure 19: **Modified Training Procedure for the BRNN Architecture**, where $\gamma \in (0, 1]$ is a modified shift operator, which in effect adds a constant weight based on the importance given to the outputs of the FRNN and BWRNN. Intuitively, we chose $\gamma < 1$ (thus $\gamma - 1 > 1$) to reflect the fact that protein chains are synthesised from the N-terminal to the C-terminal (i.e., from the left to the right side of the sequence respectively) with some secondary structural elements forming co-translationally [Baram and Yonath, 2005].

(corresponding to the reduced SS states) and assigns the SS state of the winning neuron as the prediction for the examined residue. For both schemes, once the error is calculated, the delta rule is applied to update the network weights.

3.2.3 Results and Discussion

A set of optimal NN parameters were empirically found following experimentation. For training the network, the dataset of 612 polypeptide chains was randomly split to 513 proteins for training and 99 proteins for testing. Firstly, we explored BRNN architectures with two hidden layers. More specifically, the FFNN was composed of two fully interconnected hidden layers with 12 neurons each, whereas the BwRNN and FRNN consist of two hidden layers each, with the first hidden layer having 13 and the second 12 neurons. Our first experiments, utilising single sequences at the input, achieved a highest result of 66.59% (Q3) with optimal parameters: $\gamma = 0.7$, learning rate $\alpha = 0.8$, momentum $m = 0.0$, $L_s = 15$, $S_l = 3$, and orthogonal output encoding. The next set of experiments was performed using the MSA profiles as input. For 11 out of the 612 protein chains of the initial data set an HSSP profile was not available, thus the data set was slightly reduced to a total of 601 chains, which was again randomly split into a training and a test set of 504 and 97 protein chains respectively. Initially, the predictive accuracy value reached on average 70.82% (Q3, with the aforementioned optimal parameters), almost 4% higher than when training with single sequences. In order to improve the performance, we decided to apply a randomisation procedure on the data at every iteration (i.e., when all the training data is passed through the network). We consider randomisation to be equivalent

to the insertion of noise during training, which could theoretically improve the results. As predicted, randomisation does indeed improve the prediction accuracy, as illustrated in *Table 1*. It has to also be noted that large L_s values gave better results than smaller ones. We experimented with L_s values up to 60 (where data sets were modified accordingly to exclude sequences shorter than L_s), and from the results it was obvious that the optimal L_s was 31 (which concurs with the input window size of [Baldi et al., 1999]). As it can be seen from *Table 1* the best result is 73.92% (Q3) accuracy; the corresponding SOV measure is 63.02. Experimentation with varying the number of neurons in all the layers of the constituent networks of the BRNN did not give any significant improvement in the results reported above.

Secondly, we decided to reduce the complexity of the BRNN architecture by using a single hidden layer. We also changed the output encoding to WTA. In addition, we experimented with different hidden layer sizes and the optimal results (shown in *Table 2*, columns 1 and 2) were obtained with: (i) a FFNN of 15 hidden neurons combined with RNNs of 17 hidden neurons (rows 1-4), and (ii) a FFNN of 51 hidden neurons combined with RNNs of 41 hidden neurons (rows 5-8). In order to assess whether these predictions are significantly different, we performed four repetitions of each configuration (see *Table 2*), and descriptive statistics were calculated (data not shown). For the Q3s resulting for the two configurations no statistically significant difference could be observed, with a non-parametric Mann-Whitney test (p-value=0.57). As we can see from the results shown in *Table 2* there was a significant improvement of the Q3 prediction accuracy measure (up to 76.07%) and a SOV of up to 65.36 compared to architectures with two hidden layers.

L_s	15	17	19	21	23	29	31
Q3%	72.94	73.14	72.79	71.80	73.09	73.41	73.92

Table 1: Prediction accuracy results with a BRNN architecture with two hidden layers for different sizes of the local context window L_s . MSA profiles were used as input (for all parameter values, see text).

Knowing that filtering the initial sequence-to-structure network outputs improves the prediction accuracy (see [Chen and Chaudhari, 2007] and references therein), we decided to explore this possibility on our BRNN architectures which gave the best results (i.e., with one hidden layer). Two filtering approaches were employed, one based on a novel Hidden Markov Model (HMM) and one based on a cascaded Artificial NN (ANN, similar to the structure-to-structure network of [Rost and Sander, 1993]) for comparison. According to the results (shown in Table 2, columns 3-6), both filtering approaches improve the results, in particular there is a significant increase in the SOV values (in the order of up to 10%). More specifically, for both configurations the HMM filtering was producing significantly higher SOV values (Mann-Whitney test: p-value = 0.03 for both cases) compared to the unfiltered results. In addition, the ANN filtering produced significantly higher SOV values only for the second configuration (Mann-Whitney test: p-value = 0.03) but not for the first one (p-value = 0.2). The increase in SOV was expected, since both filtering procedures are known to eliminate biologically irrelevant predictions, especially in cases where isolated residues are predicted in a SS state. For our best achieved results, further analysis regarding the details of the prediction in different SS states has been conducted. More specifically, we counted all possible instances of observed versus predicted outcomes, through which confusion matrices were created, as shown in Table 3.

Three are the main observations from the confusion matrices of Table 3:

<i>Q3%</i>	<i>SOV</i>	<i>Q3% & HMM</i>	<i>SOV & HMM</i>	<i>Q3% & ANN</i>	<i>SOV & ANN</i>
76.07	64.32	76.57	70.32	76.60	71.90
75.32	64.66	75.17	67.67	75.47	72.90
75.14	62.12	76.38	68.13	75.59	63.05
74.81	65.36	74.69	67.99	75.11	69.88
75.26	64.53	75.75	69.55	76.04	70.40
75.49	64.45	75.90	69.51	76.33	72.91
76.07	62.43	76.84	69.15	76.44	71.19
75.26	65.21	75.90	69.09	73.61	71.32

Table 2: Prediction accuracy results with a BRNN architecture with one hidden layer, randomized input, WTA output encoding and MSA profiles as input. The first four rows correspond to a BRNN architecture with a FFNN of 15 hidden neurons and RNNs of 17 hidden neurons and the last four rows to a BRNN architecture with a FFNN of 51 hidden neurons and RNNs of 41 hidden neurons. The rightmost columns correspond to the performance metrics after filtering with an HMM and a feed forward ANN.

<i>Obs vs Pred</i>	<i>H</i>	<i>E</i>	<i>C</i>	<i>H</i>	<i>E</i>	<i>C</i>
		(a)			(b)	
H	62.63	3.39	33.98	71.65	5.46	22.89
E	7.22	45.03	47.74	7.50	59.85	32.65
C	6.55	4.31	89.12	8.01	7.20	84.79
		(c)			(d)	
H	73.24	4.39	22.37	72.74	4.16	23.37
E	5.77	60.62	33.61	0.00	57.99	42.01
C	8.7	57.24	84.01	0.00	14.21	85.79

Table 3: Confusion matrices, showing the distribution of predictions for the three secondary structure states, for the best performing configurations: (a) for the best configuration listed in Table 1 giving a Q3 of 73.92%, (b) best results of the unfiltered configuration shown in Table 2 giving a Q3 of 76.07%, (c) best results of the configuration shown in Table 2 filtered with a HMM giving a Q3 of 76.57%, and (d) best results of the configuration shown in Table 2 filtered with an ANN giving a Q3 of 76.60%. Note that the displayed values correspond to the fraction of predicted residues with a given observed state, expressed as a percentage.

1. Helices (H) and loops (C) are pretty accurately predicted.
2. Extended structures (E) suffer from under-prediction. However, the novel HMM-based filtering method seems to partially overcome this problem.
3. Most of the incorrect predictions involve the loop secondary structure state. Observation 1 was expected since (i) most other works on PSSP report similar trends, and (ii) H and C are the most populated states in our datasets as opposed to strands (observation 2). For observation 3, we believe this could be an artifact of our output encoding scheme. It is worth pointing out that the ANN filtering method (see *Table 3d*) completely eliminates false predictions in the ‘H’ class.

3.2.4 Conclusions

In an attempt to tackle the PSSP problem, our modified training procedure for the BRNN, where the main modification is the updating of the weights for each residue, gives a prediction accuracy of $Q3=76.07\%$ with respective $SOV = 64.32$, which is better than the state-of-the-art results of [Baldi et al., 1999] ($Q3=73.6\%$) and clearly comparable to the results obtained in [Baldi et al., 1999] when an ensemble of 6 BRNN based predictors was used (i.e., $Q3=75.1\%$ on average). Even though we have a slight computational overhead with our approach, where weight updates occur for each residue presented to the BRNN, in practice our BRNNs train (in the worst case) within a couple of hours, and certainly we assume that (for most network parameter sets) our single BRNN would be computationally cheaper than the ensemble of 6 BRNNs reported in [Baldi et al., 1999].

Training time is not an issue for this type of applications, since it not anticipated that the BRNNs will have to be trained very frequently; the most important issue in this application is the prediction accuracy. Certainly, an exact comparison with [Baldi et al., 1999] cannot be made, unless the same training and testing sets are used, but even with that the optimal connectivity of Baldi's architecture is not available (personal communication with one of the authors of [Baldi et al., 1999], G. Pollastri).

In the work of Baldi et al. [Baldi et al., 1999], the BRNN has a short input window of residues centered at the prediction site. Although this small window aims to avoid overfitting, it does not capture variable long-range information, which is overcome by unfolding the RNNs throughout the sequence. Despite the fact that we minimise the input vector in order to contain information for a single residue, the novelty of our approach lies firstly on the fact that we use a more elaborate computation within the recurrent context windows (as described in *Section 3.2.1*), and secondly on updating network weights at every amino acid residue. With the latter, even though we are not able to capture long-range dependencies, we manage to more accurately take into account all available local information and this seems to be justified from our results.

Improved prediction results were obtained when sequence-to-structure output was filtered, in a post-processing step, in order to take higher order SS correlations into account. In particular, our novel HMM-based filtering approach not only improved the unfiltered results, but it was shown to be on average marginally better than a standard feed forward ANN-based filtering approach and much better than the BRNN-based filtering results reported in Chen and Chaudhari [2007]. We believe that if we were to use an ensemble of

BRNN-based predictors with our training scheme and our novel filtering procedures, our results would be even better. This ML techniques are investigated in *Section 3.1.3* and *Section 3.1.4*.

3.3 Ensemble methods

Ensemble methods is a category of well known methods which are used in ML to improve the performance of a learning model [Dietterich, 2000; Zhou et al., 2002, 2010; Li et al., 2018a; Zheng et al., 2019]. More specifically, through ensemble methods, instead of training just one model and get a single prediction, we train multiple instances of same or different methods and we combine the results.

There are a number of ensemble methods, some of which are more advanced and complex than others [Dietterich, 2000; Zhou et al., 2002, 2010; Li et al., 2018a; Zheng et al., 2019]. In this work, as we have seen in *Section 3.1.2* and as we will see in the next sections, we use a relatively simple ensemble method which is called an averaging ensemble. Essentially what it does it averages the outputs of its models. More specifically, in a scenario for the PSSP problem, we calculate the output for each model and classify it into one of the three classes available (H, E, and C). Then, the "winner takes all" method is used to take the results of each model and the class with the most representations is the final class of a specific input. Ensembles reduce the mis-predicted residues by averaging the results of multiple classifiers. For example, these methods show improved generalization capabilities that outperform those of single networks [Granitto et al., 2005a]. However,

for aggregation to be effective, the individual networks must be as accurate and as diverse as possible.

Consequently, random errors which might have occurred in some models are averaged out, which results in ultimately slightly better predictions, given the simplicity of this ensemble method. In more advanced ensembles, significant improvement may be achieved but they are usually computationally more expensive and time consuming.

3.3.1 Ensemble of BRNNs

ANNs were first employed for PSSP in Qian and Sejnowski [1988]. Since then, they have been widely applied in this domain under different settings [Rost and Sander, 1993; Jones, 1999]. In 1999, Baldi and colleagues [Baldi et al., 1999] implemented a BRNN architecture to predict secondary structure, which was proven one of the most successful approaches in the field. The predictive accuracy was boosted in a subsequent study through the use of an ensemble of eleven BRNNs [Pollastri et al., 2002]. The BRNN architecture is well described in *Section 3.1.2*. The FRNN processes the local information contained at the left of the local window (upstream information), whereas the BwRNN takes into account the amino acids at the right-hand side of the local window (downstream information).

In our study which is presented in *Section 3.1.2* [Agathocleous et al., 2010], we used the same BRNN architecture of Baldi et al. [Baldi et al., 1999] but the training procedure resulted in a significant increase of the predictive accuracy when a single BRNN is considered. Based on these results, we employ an ensemble of six BRNNs but, rather than

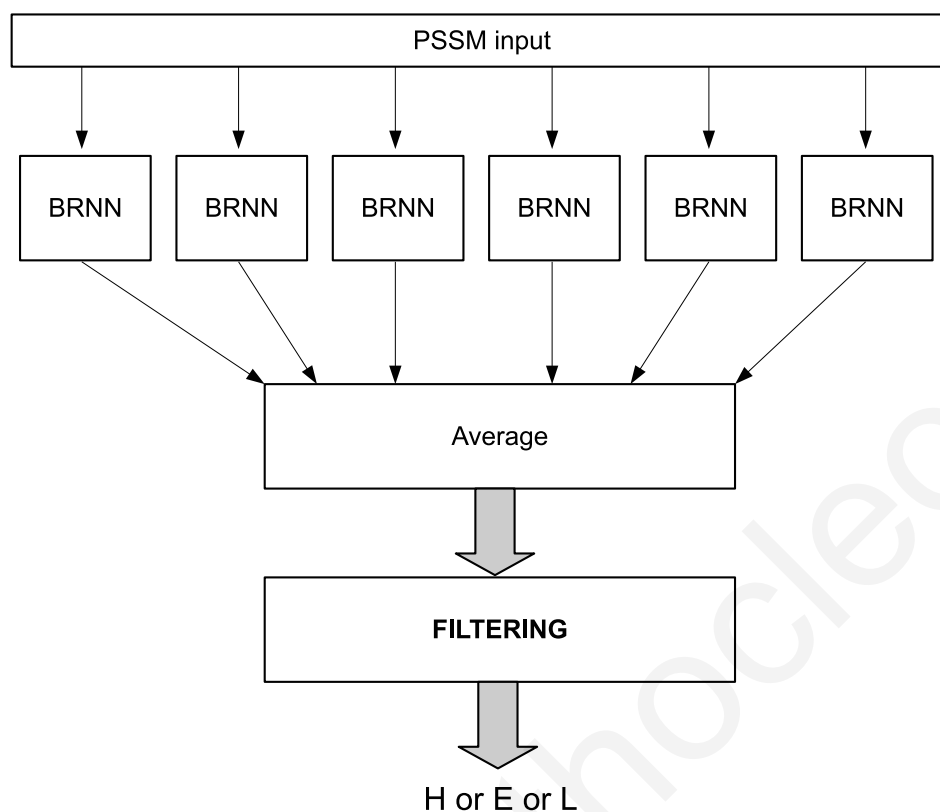


Figure 20: The architecture of the ensemble of BRNNs, followed by the filtering of the output. The Position Specific Scoring Matrices (PSSMs) values are given as input to six BRNNs, which predict the secondary structure of each residue in the amino acid sequence. Subsequently, the outputs are averaged and are given as input to the filtering methods investigated in this study.

using a single residue in the central FFNN, we utilised a local window of five residues, centred around the residue of interest. Thus, the classifier incorporates the local information contained in the neighbouring residues. Each BRNN returns three real values for the central residue of the local window, one for each secondary structure state. Subsequently, the corresponding outputs for each state are averaged and, therefore, the output of the ensemble is an array of three values for each residue. The resulting predictions are then used for filtering. The overall architecture is illustrated in *Figure 20*.

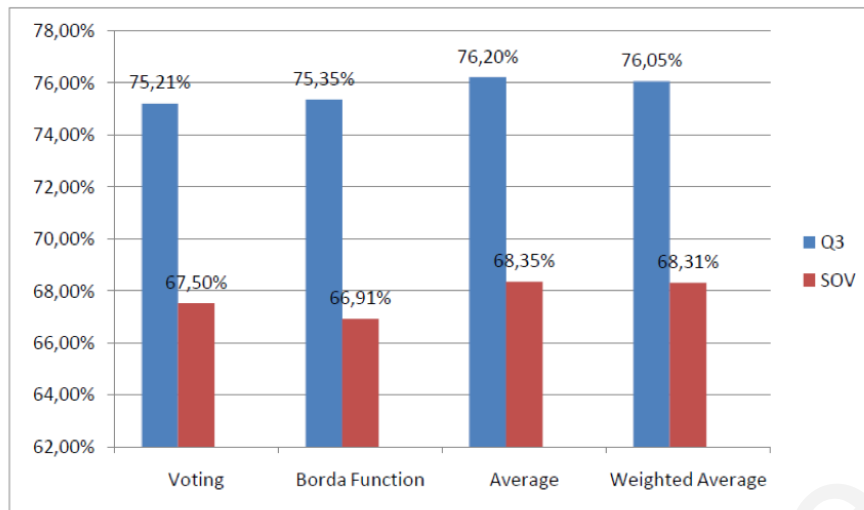


Figure 21: **Ensemble methods:** Results from ensembles of six BRNNs, trained with the 10-fold cross-validation method on the CB513 dataset.

Furthermore, we have investigated how other ensemble methods can improve our results. More specifically, we have used six BRNNs to compare four different ensemble methods: Voting, Borda Function, Average and Weighted Average. In this work, we have used 10-fold cross validation method based on the CB513 dataset. We have trained six BRNNs for each fold of the CB513 dataset and then we have used the output results to test and compare the four ensemble methods. In general, ensemble methods have managed to improve the Q_3 accuracy of single BRNNs approximately 2-3%. The results can be seen in *Figure 21*. The Average Ensemble method of six BRNNs has been shown the best results for the PSSP problem.

3.4 Filtering methods for the PSSP problem

Several PSSP methods used a multi-step process and the final step includes filtering the predictions to improve the quality of the results. This is accomplished by removing

conformations that are physicochemically unlikely. For instance, helical conformations in proteins are repetitive structures that consist of at least three, four or five residues for respectively. Since the different types of helices are usually grouped in a single category by PSSP methods, a predicted helical structure would be expected to have a minimum number of three consecutive residues in order to fulfill geometric and hydrogen-bonding requirements. Hence, predictions of single helical residues are physicochemically unrealistic, because one residue cannot make a turn in order to form a helix. To tackle this problem, both machine learning algorithms [Chen and Chaudhari, 2007] and empirical rules have been used in the past [Rost and Sander, 1993; Wood and Hirst, 2005; Salamov and Solovyev, 1995]. Despite being employed widely, there is no clear indication for the most effective filtering method in PSSP and, to the best of our knowledge, no study has been carried out to find the most suitable filtering technique.

We have performed a comparative study on the challenging problem of filtering PSSP, utilising both widely used empirical smoothing rules and machine learning techniques. Using an ensemble of six BRNNs with per-residue weight updating [Agathocleous et al., 2010], we predict the secondary structure on two non-redundant, non-homologous datasets and, subsequently, we apply a number of filtering techniques to smooth the predictions. Importantly, the SOV increases significantly in most cases. On the other hand, some classifiers increase the per-residue accuracy, whereas others decrease it. The Logistic function, the MLP and the SVMs were found to be superior to the tested methods in terms of both Q3 and SOV score. Notably, the results improve even further when we

use combinations of machine learning algorithms and empirical filtering rules. The work described in this section has been published in [Kountouris et al. \[2012\]](#).

3.4.1 Dataset and preprocessing

The study was carried out using two non-redundant, non-homologous datasets. The first, denoted as CB513, was compiled by [Cuff and Barton \[1999\]](#) and is well defined in [Section 2.3.5.1](#). CB513 has been widely utilised to compare several secondary structure prediction methods in the literature, e.g. [[Karypis, 2006](#); [Kountouris and Hirst, 2009](#)]. Because of its small size, this dataset was used to study the impact of various input coding schemes. The second dataset was PDB-Select25 (version October 2008) [[Hobohm et al., 1992](#)], a set of 4018 high quality X-ray and NMR structures with less than 25% sequence similarity. From the initial set, we removed chains for which DSSP [[Kabsch and Sander, 1983](#)] did not return valid output, which resulted in a final set of 3977 protein chains. Even though most typical PSSP methods are optimised to work with globular proteins, we decided not to remove around 90 transmembrane proteins contained in this dataset.

Secondary structure was assigned based on the experimentally determined 3D structures using the established DSSP program [[Kabsch and Sander, 1983](#)], which assigns secondary structure in eight states: H (α -helix), G (3_{10} -helix), E (extended β -strand), B (isolated β -bridge), T (turn), S (bend) and “_” (other/coil). Most of the existing methods predict secondary structure using a three-state assignment. Hence, we reduce the above representation into a three-state scheme, by assigning H, G, and I to the helix state (H),

E and B to the extended state (E) and the rest to the loop state (L). This three-state representation is also followed by the EVA secondary structure prediction validation server [Rost and Eyrich, 2001].

Since their first use in PSIPRED [Jones, 1999], PSI-BLAST's [Altschul et al., 1997] PSSMs are utilised by the majority of PSSP methods. PSSMs are constructed using multiple sequence alignments and provide crucial evolutionary information about the protein structure. PSSMs consist of $N \times 20$ elements, where the N rows correspond to the length of the amino acid sequence and the columns correspond to the 20 standard amino acids. We generated a PSSM for each chain in the dataset using the BLOSUM62 substitution matrix [Henikoff and Henikoff, 1992] with an e-value of 0.001 and three iterations against the NCBI non-redundant (nr) database, downloaded in February 2009. The database was filtered by *pfilt* [Jones and Swindells, 2002] to remove low complexity regions, transmembrane spans and coiled coil regions. This filtering could be important for dealing with transmembrane proteins.

3.4.2 Filtering techniques

We evaluate an array of machine learning algorithms to identify those that perform better in filtering PSSP. More specifically, we employed the WEKA software package [Witten and Frank, 2005] to test the following classification algorithms: Naive Bayes, Simple Cart, Radial Basis Function (RBF) network initialised by k -Means clustering ($k = 3$), IBk (nearest neighbour algorithm) with $k = 3$, MLP, Random Forest, J48

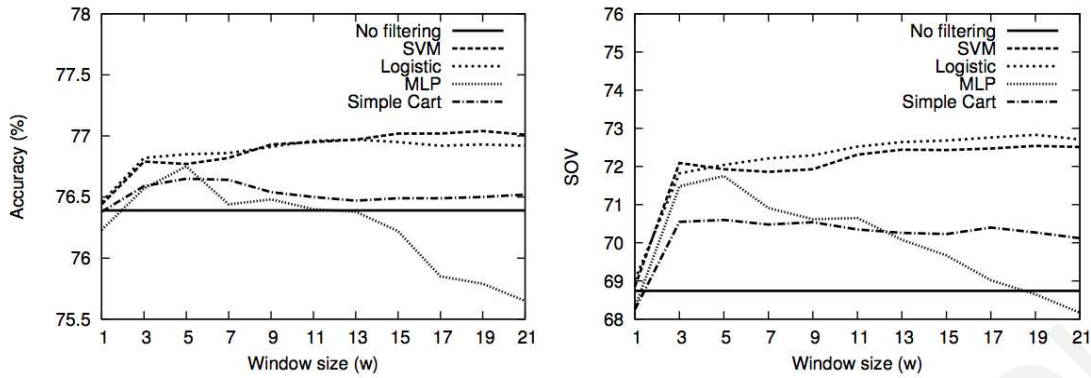


Figure 22: Experiments with different local window sizes for filtering PSSP (see text for more information) using four machine learning algorithms on the CB513 dataset. The predictive accuracy (left) and the SOV score (right) strongly depend on the size of the local window used for filtering.

decision tree (C4.5) and Logistic function. The latter is an implementation of a multinomial logistic regression function with a ridge estimator to avoid overfitting [leCessie and vanHouwelingen, 1992].

Additionally, we employed SVMs to filter the predictions. More specifically, we used the default one-against-one multi-class SVM provided by the LibSVM software package [Chang and Lin, 2011]. We utilised the RBF kernel function and we set the kernel parameter, γ , at $\frac{1}{3w}$, where w is the length of the local window. Finally, the misclassification penalty parameter, C , was equal to unity.

Moreover, we used a meta-classifier to combine two or more from the above algorithms by using several voting schemes. More specifically, we employed the following voting schemes (implemented in WEKA): (i) Majority Vote, (ii) Maximum probability, (iii) Minimum probability, (iv) Product of probabilities and (v) Average of probabilities.

Table 4: Filtering PSSP on the CB513 dataset using the method shown in the first column, sorted by the highest predictive accuracy (Q_3). w is the local window size for filtering that maximises the SEL score for each method. In bold are the highest scores in the corresponding column.

Filtering method	w	Q_3 (%)	Q_H (%)	Q_E (%)	Q_L (%)	SOV	SOV _H	SOV _E	SOV _L	C_H	C_E	C_L	SEL
LibSVM	19	77.04	78.02	65.81	82.40	72.54	71.92	68.64	70.80	0.718	0.635	0.583	74.79
Logistic	19	76.93	78.69	67.33	80.76	72.83	72.43	68.74	71.31	0.716	0.633	0.582	74.88
MLP	5	76.75	77.94	66.02	81.68	71.75	70.72	68.77	70.17	0.717	0.626	0.579	74.25
Simple Cart	5	76.65	79.06	66.78	80.11	70.60	70.91	67.57	69.66	0.712	0.625	0.580	73.63
SS-filt	—	76.43	75.98	62.23	84.58	71.25	70.53	66.92	70.56	0.711	0.622	0.578	73.84
No filtering	—	76.39	77.12	63.53	82.87	68.74	68.75	66.07	69.63	0.706	0.618	0.580	72.57
WH-filt	—	76.24	74.77	62.33	85.06	69.43	67.31	65.90	70.35	0.710	0.616	0.577	72.84
RBF Network	1	76.23	81.52	69.88	75.44	69.30	71.73	68.84	66.86	0.705	0.618	0.578	72.77
Naive Bayes	3	76.10	78.68	71.99	76.27	71.75	71.37	70.46	68.57	0.710	0.629	0.561	73.92
Viterbi	1	75.98	77.77	63.93	81.14	69.59	69.58	65.57	67.53	0.705	0.619	0.562	72.79
J48	3	75.98	78.97	65.87	79.10	68.53	69.33	66.84	67.84	0.704	0.610	0.569	72.25
Random Forest	19	75.19	79.64	68.58	75.23	66.76	68.58	66.68	64.94	0.696	0.608	0.550	70.98
IBk (k=3)	13	72.03	78.67	62.64	71.81	62.66	67.98	62.71	60.86	0.640	0.566	0.499	67.34

The combinations of filtering methods are selected based on the performance of the individual learning techniques and they are discussed in *Section 3.4.5*. For the same purpose, we implemented a HMM using the Viterbi algorithm [Viterbi, 1967]. A detailed description of the above algorithms is beyond the scope of this article, but a comprehensive survey for many of them in the context of their potential in data mining was written by Wu and co-workers [Wu et al., 2007]. All algorithms from WEKA and LibSVM were used with the default parameters. The results presented here can be possibly improved by optimising each algorithm individually.

The ultimate goal of a classification algorithm is not to achieve high training accuracy, but to classify successfully previously unseen examples. Hence, we use n -fold cross-validation to estimate the generalisation error. More specifically, we divide the training set into n subsets and, sequentially, we use $n - 1$ for training and the remaining one for testing. This procedure is repeated n times, until all subsets are used once for testing. In this paper, we report the results from 10-fold cross-validation on the CB513 dataset and 5-fold cross-validation on the PDB-Select25 dataset. For both datasets, the folds have similar representation of helical, extended and loop residues. Moreover, in the case of CB513, we ensure similar distributions of small/large protein chains as well as of the four main SCOP classes (all- α , all- β , $\alpha + \beta$ and α/β) [Murzin et al., 1995]. The subsets are available on request.

Additionally, we filtered PSSP using two empirical techniques that were previously used to filter other PSSP methods. For this purpose, we implemented a custom software, which uses regular expressions to perform the empirical filtering step. The first set of smoothing rules (denoted as SS-filt) was compiled by Salamov and Solovyev [Salamov and Solovyev, 1995] and contains the following three filtering rules: (i) replace single helical residues with loop, i.e. $!H H !H \rightarrow !H C !H$; (ii) replace single strand residues with loop, i.e. $!E E !E \rightarrow !E C !E$; and (iii) all strands of length two surrounded with helices are replaced by helices, i.e. $H E E H \rightarrow H H H H$. The second set of filtering rules (denoted as WH-filt) consists of ten empirical rules that have been used to filter PSSP from DESTRUCT and can be found in [Wood and Hirst, 2005]. The above rules are based on empirical knowledge and aim to remove physicochemically unrealistic predictions.

Table 5: Filtering PSSP on the PDB-Select25 dataset using the method shown in the first column, sorted by the highest predictive accuracy (Q_3). In bold are the highest scores in the corresponding column.

Filtering method	Q_3 (%)	Q_H (%)	Q_E (%)	Q_L (%)	SOV	SOV_H	SOV_E	SOV_L	C_H	C_E	C_L	SEL
LibSVM	77.53	79.60	65.98	82.13	72.29	72.46	71.44	71.20	0.724	0.647	0.589	74.91
MLP	77.28	79.44	69.08	79.97	72.14	72.55	73.07	70.87	0.719	0.645	0.585	74.72
Logistic	77.04	78.21	70.20	79.81	71.82	71.13	73.76	70.99	0.713	0.643	0.584	74.43
Simple Cart	77.02	80.08	68.43	79.15	70.96	71.86	72.46	70.16	0.712	0.636	0.586	73.99
J48	76.59	79.89	67.63	78.74	69.13	70.72	71.44	69.05	0.705	0.628	0.581	72.86
RBF Network	76.43	80.34	68.90	77.29	68.63	70.85	71.51	68.55	0.705	0.619	0.581	72.53
SS-filt	76.19	73.49	73.79	79.76	70.80	67.40	76.28	70.88	0.703	0.628	0.579	73.49
No filtering	76.12	74.42	74.53	78.41	67.95	65.00	76.17	69.22	0.702	0.622	0.582	72.03
Random Forest	76.00	81.25	70.09	74.83	65.78	69.01	70.94	64.97	0.703	0.635	0.563	70.89
Naive Bayes	75.99	77.33	76.70	74.48	71.36	70.74	75.73	68.81	0.709	0.633	0.563	73.68
Viterbi	75.88	75.72	74.57	76.73	70.35	67.99	73.84	68.23	0.704	0.626	0.566	73.12
WH-filt	75.74	71.67	73.62	80.31	68.76	63.69	75.38	70.57	0.696	0.618	0.577	72.25
IBk (k=3)	73.45	80.03	66.10	71.97	61.80	67.66	68.38	60.76	0.652	0.601	0.522	67.63

Finally, combinations of machine learning algorithms and empirical algorithms were also used. More specifically, a machine learning algorithm was applied at first and, subsequently, an empirical rule was used to filter the outputs. This approach resulted in further improvement of the machine learning algorithms as discussed below.

3.4.3 Prediction accuracy assessment

To facilitate an objective comparison of the above learning methods, several measures were used to assess the performance of each filtering technique, most of them defined in the EVA server [Rost and Eyrich, 2001]. All the evaluation matrices have been defined in *Section 2.3.3*.

In addition, we define a selection criterion, denoted as SEL, which takes equally into account the achieved Q_3 and SOV scores, the most established assessment measures. Therefore, the SEL score is calculated as follows:

$$SEL = \frac{Q_3 + SOV}{2} \quad (28)$$

3.4.4 Finding the best local window

A window of neighbouring residues is often used in secondary structure prediction to capture additional information about local interactions [Rost, 2001] and, hence, we investigate the use of a local window, w , for filtering, centred around the residue to be predicted. More specifically, the ensemble of BRNNs has three output values for each residue, one for each secondary structure state. Therefore, a local window of size w will result in an input vector of $3 \times w$ attributes for the filtering classifier. Due to different design and capabilities, the size of the local window that maximises the predictive accuracy or the SOV is different for each classifier employed in this study. Using the CB513 dataset, we tested different input coding schemes for each method to find the best local window in each case. *Figure 22* shows how the predictive accuracy and the SOV measure changes by varying the size of the local window. The selected window size was the one that maximised the SEL score (*Equation 28*) for each machine learning technique tested, thus taking into account both the Q_3 and the SOV score. The drop of accuracy in the case of MLP for $w > 5$ is most probably due to overfitting. After optimising the local window

Table 6: Filtering PSSP using combinations of machine learning algorithms and empirical rules. Firstly, a machine learning algorithm is employed for filtering (shown in the first column) and, subsequently, the output is filtered by empirical rules (SS-filt or WH-filt) to further improve PSSP. In bold are the highest Q_3 , SOV and SEL scores.

Classifier	CB513 dataset						PDB-Select25 dataset					
	SS-filt			WH-filt			SS-filt			WH-filt		
	Q_3 (%)	SOV	SEL	Q_3 (%)	SOV	SEL	Q_3 (%)	SOV	SEL	Q_3 (%)	SOV	SEL
LibSVM	77.02	72.94	74.98	76.85	72.21	74.53	77.50	72.66	75.08	77.33	71.98	74.65
Logistic	76.92	73.42	75.18	76.76	72.52	74.64	77.04	72.64	74.84	76.89	71.78	74.34
MLP	76.74	72.50	74.62	76.58	71.48	74.03	77.27	72.91	75.09	77.15	72.15	74.65
Simple Cart	76.67	72.64	74.65	76.51	71.14	73.82	77.06	72.70	74.88	76.89	71.53	74.21
RBF Network	76.54	72.52	74.53	76.39	70.61	73.50	76.66	71.54	74.10	76.44	69.91	73.17
J48	76.21	71.42	73.81	75.85	69.34	72.60	76.75	71.73	74.24	76.54	70.14	73.34
Naive Bayes	76.11	72.10	74.10	75.96	71.50	73.73	76.01	71.79	73.90	75.82	71.11	73.47
Viterbi	75.98	68.59	72.29	75.88	69.15	72.52	75.88	70.35	73.12	75.81	70.01	72.91
Random Forest	75.71	71.58	73.64	75.49	69.12	72.30	76.47	71.27	73.87	76.16	68.42	72.29
IBk (k=3)	73.33	68.73	71.03	73.01	65.45	69.23	74.55	68.30	71.42	74.14	65.14	69.64

sizes for each method on the CB513 dataset, we utilised them for filtering PSSP on the PDB-Select25 dataset.

3.4.5 Results and discussion

Table 4 shows the performance of each filtering method sorted by the highest accuracy, after finding the local window size, w , that maximises the predictive accuracy (Q_3) on the CB513 dataset. The SVM achieved the highest predictive accuracy of 77.04%, an absolute improvement of 0.65% compared to the unfiltered performance, while the SOV score increased by 3.8, reaching the value of 72.54. However, it is the Logistic function that achieved the highest SOV score of 72.83, an increase of 4.09, whereas its predictive accuracy of 76.93% is ranked second to the tested methods. Notably, the Logistic function

has higher SEL score than the SVM, while it is also a faster classifier for this problem. In addition, the MLP and the Simple Cart also achieve improved accuracies and SOV scores higher than 70. Despite that only half of the machine learning algorithms increase their accuracy after filtering, the majority of them increase the SOV significantly.

Table 5 shows the performance of each technique on the PDB-Select25 dataset, for which the window sizes were optimised in the CB513 dataset (see *Table 4*). It is worth mentioning that the overall performance of the unfiltered BRNN ensemble (“No Filtering” row) is slightly lower for this larger dataset. This observation is consistent when all three overall performance measures are considered, i.e. Q_3 , SOV and SEL. Nonetheless, the majority of the applied methods improve the predictive performance and a higher increase is observed compared to the results on the CB513 dataset. The SVM is the most accurate filtering method based on all three basic measures (Q_3 , SOV and SEL), showing an improvement of around 1.4%, 4.4 and 2.9, respectively, while the Logistic function and the MLP are amongst the most accurate techniques. In fact, the MLP performs particularly well on this dataset and is ranked second to the tested methods based on the achieved Q_3 , SOV and SEL. Apart from the MLP, the RBF network and the J48 decision tree perform better on the PDB-Select25 dataset than on the CB513 dataset.

3.4.5.1 Prediction accuracy per state

Notably, some machine learning techniques perform particularly well in the prediction of individual states. More specifically, the RBF network and the Random Forest achieve the highest per-state accuracy for helical residues on both datasets, even though their

overall Q_3 score is lower than that of the best performing methods. In fact, the RBF Network increases the Q_H accuracy by 4.4% on the CB513 dataset, which is more than 2.5% higher than the Q_H score of the two best performing classifiers. On the PDB-Select25 dataset, the Random Forest achieves a remarkable improvement of the Q_H score by 6.8%.

Similarly, the Naive Bayes classifier is very accurate in the prediction of extended residues achieving Q_E of 71.99% on the CB513 dataset, which is more than 4% higher than the Q_E achieved by the Logistic function, whereas it is more than 6% higher than the Q_E of the SVM. Importantly, its achieved Q_E score of 76.7% on the PDB-Select25 dataset is around 10.7% higher than that of the SVM. In contrast, all three algorithms perform relatively poor in the prediction of loop residues, resulting in a low overall per-residue accuracy.

Interestingly, the ensemble of BRNNs overpredicts extended residues with the utilisation of the PDB-Select25 dataset (see *Table 5*), but the application of filtering techniques significantly affects the predictive performance. Some classifiers, such as the Naive Bayes and the Viterbi algorithm, perform well for the prediction of extended residues, while others, such as the LibSVM, decrease the Q_E score. The explanation can be derived from the size of secondary structure elements. While α -helices are usually long repetitive structures with an average length of about ten residues, most extended structures in proteins are shorter than eight residues. Therefore, using a long local window may improve the prediction of longer structures (helix and coil), but it may also decrease the predictive performance of short extended structures. In fact, the Q_E scores shown in *Table 5* are usually

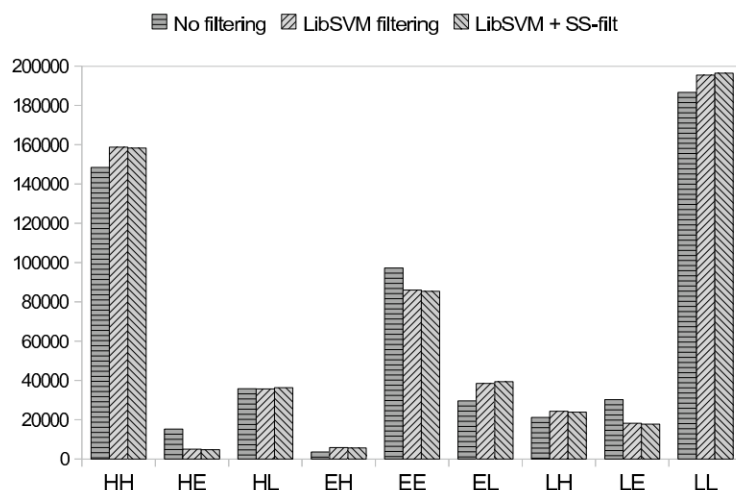


Figure 23: Per state prediction after the application of filtering techniques on the PDB-Select25 dataset. The y -axis corresponds to the number of residues and the x -axis to the combinations of observed and predicted state. For instance, HE corresponds to residues that are observed as helices (H) but are predicted as extended (E). The three columns at each state show the number of residues for the unfiltered classifier (ensemble of BRNNs), the LibSVM filtering and the combination of LibSVM and SS-filt, respectively.

higher for classifiers that use short local windows (from one to three residues) and lower for classifiers that use long windows, such as 19 residues. In addition, the dataset size seems to be important for the prediction of the extended class, which underrepresented in experimentally determined structures and this is an additional reason it is more difficult to predict. Thus, increasing the size of the dataset can provide crucial additional information about the extended state.

3.4.5.2 Combining machine learning and empirical techniques

Interestingly, the empirical filtering does not increase the Q_3 score dramatically, but it improves the SOV. The smoothing rules used in SS-filt are more effective than those in WH-filt in terms of both Q_3 and SOV. The empirical rules have better performance

```

A 1AHB_A, 180-246
PriStr  VPSLATISLENSWSGLSKQIQLAQGN...
Real SS  LLLLLLLLLLLLLLLLLLLLLLLLLLL...
No-Filt  LLLLLEEEHLHLHLHLHHHEHLL...
LibSVM   LLLHHHEEHLHLHLHLHHHHLL...
SS-Filt  LLLLLLLLLLLLLLLLLLLLLLLLLLL...

B 1HIW_S, 7-109
PriStr  ...QTGSEELRSLYNTI AVLVCVHQRIDVKDTKEA...
Real SS  ...LLELHHHHHHHHHHHHHHLLLELHH...
No-Filt  ...LLLHHHHHHHLHEEEHELEHHEH...
LibSVM   ...LLLHHHHHHHHHEEHLHLHLHH...
SS-Filt  ...LLLHHHHHHHHHHHHLLHLHLHH...

C 2DA7_A
PriStr  ...LPQEFVKEWFQKRVYQYSNSRSGPSSG
Real SS  ...LLLLLLLLLLLLLLLLLLLLLLLLLL
No-Filt  ...LLLLEEEHHHHHHHHHHLLLL
LibSVM   ...LLHHHEEHHHHHHHHHHLLLL
SS-Filt  ...LLHHHHHHHHHHHHHHLLLL

D 1MSO_B
PriStr  FVNQHLCGSHLVEALYLVCGERGFFYTPKT
Real SS  LLELLLLHHHHHHHHHHHLEEEELLL
No-Filt  LLLLHLHHHEEEELLLLEEEELLL
LibSVM   LLLLHLHHHHHHHHHHHLLLEELLL
SS-Filt  LLLLHHHHHHHHHHHHLLLEELLL

E 2ELN_A
PriStr  GSSGSSGILLKPTDGDYSTPDKYKLQAHKVTALD
Real SS  LLLLLLLEELLLLEELHHHHHHHHLL
No-Filt  LLLLLLEELLLLEELHEEEEEELHL
LibSVM   LLLLLLLEELLLLEELHHHHHEHHL
SS-Filt  LLLLLLLEELLLLEELHHHHHHL

```

Figure 24: Five examples that show the effect of filtering on PSSP. The first line in each case shows the PDB ID and the Chain ID. Sequences A and B are taken from CB513 and the remaining sequences from PDB-Select25. The mispredictions are shown in shadow. “PriStr” is the amino acid sequence; “Real SS” is the observed secondary structure; “No-Filt” is the PSSP from the ensemble of BRNNs; “LibSVM” is the PSSP filtered with LibSVM and “SS-Filt” is the application of the SS-Filt empirical rules on the output of LibSVM filtering. Secondary structure states are reported using the reduced three-state scheme (see [Section 3.4.1](#)).

than some of the machine learning algorithms shown in [Tables 4](#) and [5](#), but they come short when they are compared with the best performing techniques, such as the SVM, the Logistic function and the MLP.

Nevertheless, a combination of machine learning techniques and empirical rules can lead to a generally improved filtering in secondary structure prediction. As shown in *Table 6*, the quality of the prediction is improved when we apply empirical rules after filtering by a machine learning algorithm. The empirical rules are manually created by scientists and, hence, they provide physicochemically realistic information, which sometimes cannot be captured by a learning algorithm. On both datasets, the predictive accuracy is not improved for the best performing classifiers, but it is higher for algorithms that achieved lower accuracies without the employment of the empirical rules (compare with *Tables 4* and *5*), such as the RBF Network and the Random Forest. Most importantly, combinations of machine learning techniques and empirical rules give a major boost to the SOV score leading to an improvement of 2% in most cases, demonstrating the crucial information provided by the empirical rules. On CB513, the Logistic function achieves the highest SOV score of 73.42 when we apply the SS-filt rules, while the SVM has the highest predictive accuracy. On PDB-Select25, the SVM remains the most accurate method in terms of Q_3 , whilst the MLP achieves the highest SOV (72.91). In both datasets, the best performing methods have SOV score greater than 72.5. As discussed above, the SS-filt rules are more effective than the WH-filt rules and this is also observed in the results of *Table 6*.

Figure 23 illustrates how the number of correct predictions or mispredictions changes after filtering PSSP with LibSVM and the subsequent application of the SS-filt empirical rules on the PDB-Select25 dataset. As stated above, the ensemble of BRNNs overpredicts extended residues and this is reflected on the high percentage of correctly predicted

Table 7: Filtering PSSP using combinations of machine learning algorithms with different voting schemes on the CB513 dataset. The last three columns show the results after using the SS-filt empirical rules. In bold are the highest scores in the corresponding column.

Classifiers	Voting	w	Machine learning only			SS-filt		
			Q ₃ (%)	SOV	SEL	Q ₃ (%)	SOV	SEL
Logistic + RBF + Random Forest	Prod	3	76.71	71.74	74.23	76.74	72.76	74.75
Logistic + Simple Cart + RBF	Prod	5	76.54	70.50	73.52	76.53	72.02	74.27
Logistic + Naive Bayes	Prod	3	76.30	71.93	74.12	76.31	72.37	74.34
Logistic + Naive Bayes	Avg	3	76.28	71.92	74.10	76.28	72.37	74.33
Logistic + Simple Cart + MLP	Prod	9	77.11	72.25	74.68	77.08	73.43	75.26
Logistic + Simple Cart + MLP	Min	5	76.93	72.17	74.54	76.92	73.11	75.01
Logistic + Simple Cart + MLP	Max	11	76.95	72.16	74.55	76.93	73.22	75.07
Logistic + Simple Cart + MLP	Avg	9	77.12	72.36	74.73	77.09	73.46	75.27
Logistic + Simple Cart + MLP	Maj	11	76.90	71.75	74.33	76.89	72.62	74.76

extended residues, but also on the large number of mispredicted helical and loop residues as extended (HE and LE states). This behaviour is smoothed after filtering with LibSVM, which decreases the number of both correct predictions and mispredictions to extended state, while improving the performance of helix and loop prediction. The application of the empirical rules does not have a significant effect on the number of correctly predicted residues, but, as discussed above, it improves the SOV score. The analysis of mispredictions based on their dihedral angles, ϕ and ψ , did not reveal any particular trend since the mispredicted residues are distributed all over the Ramachandran plot (data not shown).

Figure 24 shows five examples which demonstrate the possible effect of filtering on the overall PSSP. Importantly, the use of the SVM filtering improves PSSP noticeably and in some cases, such as examples D and E, leads in a significant improvement of the predictive performance. Subsequently, the application of the SS-Filt rules is a final refinement step, which smoothes the LibSVM predictions using rules that are well-defined

a priori (*Section 3.4.2*), but its effect is not always significant for the overall predictive performance. Importantly, the filtering step (LibSVM with SS-Filt) should be used as a post-processing step which will refine the results of a PSSP method. The success of any filtering technique strongly depends on the success of the initial prediction method. Filtering can be highly beneficial for state-of-the-art PSSP methods because the output of the initial method is fed as input to the filtering algorithm and, thus, this information must be as accurate as possible. This is certainly a challenging task which is outside the scope of this article, where we consider our initial prediction method satisfactory given its results.

3.4.5.3 Combining machine learning techniques

Based on the results shown in Tables 4 and 5, we tested various combinations of machine learning techniques using different voting schemes implemented in WEKA (see *Section 3.4.2*) and the results are shown in *Table 7* for the CB513 dataset. More specifically, a number of machine learning algorithms are initially used for filtering and their output is fed into a voting function, which decides for the final prediction. The voting schemes based on the average probabilities and the product of probabilities achieve the highest accuracies. The predictions are then filtered by the SS-filt empirical rules to further improve the predictive performance. Given the extensive computational needs of the SVM, we did not use it for the combinations presented in *Table 7*. Instead, the Logistic function was employed in combination with other machine learning techniques. The voting based on the average probability using the Logistic function, the Simple Cart and the

MLP slightly improved the Q_3 and SOV scores. However, the improvement is insignificant compared to the performance of the Logistic function alone. Moreover, combining the Logistic function with classifiers that performed particularly well in the prediction of helical and extended residues, such as the RBF Network and the Naive Bayes, did not have positive impact on the overall performance. Due to the computational requirements and given the insignificant improvement of predictive accuracy on the CB513 dataset, we did not apply the above combinations of machine learning techniques on the PDB-Select25 dataset.

3.4.6 Conclusion

The aim of [Kountouris et al. \[2012\]](#) was to compare the performance of a variety of filtering methods to the problem of PSSP, which has not been studied systematically, although it is utilised by a plethora of prediction methods. We employed both machine learning algorithms and empirical methods and, using two non-redundant, non-homologous sets of 513 and 3977 protein chains, respectively, we showed that the SVM, the Logistic function and the MLP are the most suitable learning techniques to tackle this problem. More importantly, combinations of machine learning techniques and empirical smoothing rules can improve the quality of the predictions even further, particularly the SOV score.

Based on the results presented in this article, we suggest the utilisation of the Logistic function or the MLP followed by the application of the SS-filt empirical rules to filter

PSSP. Despite achieving slightly lower predictive accuracy than the SVM, these classifiers are much faster compared to the SVM and can lead to reliable filtering of the predictions.

Our findings are based on initial (sequence-to-structure) secondary structure predictions obtained by a BRNN with per-residue weight updating. Different approaches at this starting step are expected to yield different results, thus the impact of filtering methods on alternative initial data, or their combinations, should be further investigated.

We are currently conducting a similar study to evaluate learning algorithms used for ANN ensembles, instead of just averaging the outputs of a number of ANNs. Finally, since filtering is a common step in many protein structure prediction problems, such as β -turn prediction [Shepherd et al., 2001], this comparative study can be useful for other research fields in structural bioinformatics.

3.5 Chapter Contribution

A list of contributions resulted from this chapter is presented below:

1. The proof that the upper limit of a window size which can be captured from a simple BRNN trained with the BPTT algorithm for the PSSP problem is 31 amino acids.
2. The proof that in a BRNN trained with BPTT, the local information of a window around a specific residue is sufficient, compared to a method where the whole sequence is used for each residue. The results are comparable because in the second case the information of long range dependencies is lost due to the vanishing gradient problem.

3. The introduction of the γ parameter, a modified shift operator, which in effect adds a constant weight based on the importance given to the outputs of the FRNN and BwRNN.
4. The comparison of PSSP filtering methods where it was shown that the SVM, the Logistic function and the MLP are the most suitable learning techniques to tackle this problem.
5. The MLP, which is faster than SVM, can lead to reliable filtering of PSSP predictions despite achieving slightly lower predictive accuracy than SVM.
6. Combinations of machine learning techniques and empirical smoothing rules can improve the quality of the predictions, particularly the SOV score.

Chapter 4

Training BRNNs with Second Order Learning algorithms

4.1 Introduction

The need to train BRNNs with more efficient algorithms than existing methods in terms of accuracy and convergence time has been the initial motivation for this Chapter. The SCG [Møller, 1993b], a second-order learning algorithm, which has been found to be superior to the conventional GD algorithm in terms of accuracy, convergence rate and the vanishing-gradient problem [Hochreiter and Schmidhuber, 1997]. In addition, the original form of the algorithm [Møller, 1993b] does not depend on any parameters. Consequently, *Section 4.2* introduces the mathematical analysis and development of our proposed Hybrid Rectified-Scaled Conjugate Gradient (HR-SCG) algorithm to train the BRNN architecture for the PSSP problem which is based on the SCG algorithm. Furthermore, in *Section 4.3*, we present the results from the application of another powerful

second order learning algorithm, the HFO [Martens, 2010; Martens and Sutskever, 2011] algorithm, on the same architecture and problem.

4.2 Training BRNNs with the HR-SCG learning algorithm

4.2.1 Introduction

ML is a subfield of Artificial Intelligence (AI) that provides a system the ability to learn from data. Usually, ML methods consist of a trainable model and a learning algorithm [Kotsiantis et al., 2007; Liew et al., 2016]. These methods are mostly used for classification and prediction on static and sequential data.

Even though a number of theoretical ML algorithms has been designed to process and make predictions on sequential data, the mining of such data types is still an open field of research due to its complexity. Analysis and development of optimisation algorithms for specific ML techniques for sequential data must take into account how (a) to capture and exploit sequential correlations, (b) to represent and incorporate loss functions, (c) to identify long-distance dependencies, and (d) to make the optimisation algorithm fast [Dietterich, 2002]. One of the most successful classes of models which has been designed to deal with these questions is Recurrent Neural Networks (RNNs) [Elman, 1990] which belongs to the class of Artificial Neural Networks (ANNs). More specifically, ANNs are universal function approximators [Hornik et al., 1989; Park and Sandberg, 1991; Scarselli and Tsoi, 1998] and RNNs are universal approximators of dynamical systems [Funahashi and Nakamura, 1993]. The connections between the nodes of these networks have the

distinction of forming directed cycles which create an internal memory in the model to process classification tasks with sequential data. Consequently, RNNs can associate patterns which are located far away from each other on a sequence and create a kind of short and long range dependencies between data.

Training RNNs is a demanding task in terms of time and space efficiency. At the same time it is a difficult task because of the vanishing gradient and exploding gradient problems, where the gradient is getting smaller as the information is moving backward through hidden layers or is getting very big at the early layers of a model [Bengio et al., 2004]. In order to train RNNs, several learning algorithms have been developed which involve adjusting the network weights till a desirable mapping between input and output sequence is formed. Those algorithms are mostly related to learning algorithms which have been developed to train FFNNs and are well described in Battiti [1992]. The most common class of RNN training algorithms is based on the gradient descent minimization method which is applied on unfolded (through sequence) version of a RNN [Rumelhart et al., 1986b], with the most well known learning algorithm for such models being the GD based on Backpropagation Through Time (BPTT) [Werbos, 1990; Frasconi et al., 1998]. Unfortunately, this kind of algorithms have a poor convergence rate [Møller, 1993b]. Moreover, they depend on parameters which have to be specified by the user and are usually crucial for the performance of the algorithm. In order to improve these drawbacks, more efficient algorithms need to be developed. As we have already mentioned, one candidate algorithm is the SCG [Møller, 1993b]. Since then many variants [Andrei, 2007; Cetişli and Barkana, 2010; Fatemi, 2017] and many applications [Khadse et al., 2017;

Gayathri and Kumarappan, 2015; Borkar et al., 2016] have been proposed but not all of these have been designed or adopted for RNNs.

Moreover, predictions on sequential data are particularly challenging when both the upstream and downstream information of a sequence is important for a specific time-step. Application examples include Phoneme Speech Recognition (PSR) [Graves and Schmidhuber, 2005; Wollmer et al., 2009] and Bioinformatics problems, such as Protein Secondary Structure Prediction (PSSP) [Baldi et al., 1999; Agathocleous et al., 2010; Kountouris et al., 2012] and other related problems (e.g., Transmembrane Protein Topology Prediction (TMPTP) [Nugent and Jones, 2009]). In such sequence-based problems the events are dynamic and located downstream and upstream, i.e., left and right in the sequence. A ML model designed for such data must learn to make predictions based on both directions of a sequence. To predict these events, researchers utilise Bidirectional Recurrent Neural Network (BRNN) architectures [Baldi et al., 1999]. The BRNN architectures are usually trained with an extension of the BPTT algorithm [Frasconi et al., 1998] with the error propagated in both directions of the BRNN. To the best of our knowledge the SCG algorithm has not been applied in the past for these models. Since the potential of BRNNs has not been fully realized, methods that address the difficulty of training this kind of models are of great importance.

BRNNs have been used widely on the PSSP problem [Baldi et al., 1999], PSR [Graves and Schmidhuber, 2005; Wollmer et al., 2009] and healthcare [Kollias et al., 2018]. Knowledge of a protein's secondary structure is important as a step towards elucidation of its 3D structure, which is crucial to understand its function. This problem is even more

pronounced since the rapid growth of the number of available protein sequences has far outpaced the experimental determination of their structures. Thus, there is a growing need for a computational approach to the problem of protein structure prediction. The prediction of Secondary Structure (SS), i.e., the local structure commonly defined by hydrogen bond patterns and local geometry, is a critical first step towards this end and, therefore, it has attracted a great amount of interest over the past 60 years [Yang et al., 2016]. With respect to their secondary structure, amino acid residues in protein chains are usually assigned into three main classes, namely helix, extended and coil/loop [Kabsch and Sander, 1983]. Ensembles of BRNNs trained with the GD algorithm have proved to be a very efficient method for the PSSP problem with accuracy of approximately of 76%-79% [Baldi et al., 1999; Pollastri et al., 2002; Magnan and Baldi, 2014].

The need to train BRNNs with more efficient algorithms than existing methods, in terms of accuracy and convergence time, has been the initial motivation for this work. Consequently, this section introduces the mathematical analysis and development of our proposed HR-SCG algorithm to train the BRNN architecture for the PSSP problem. It has been recently shown that BRNNs combined with LSTM units [Hochreiter and Schmidhuber, 1997] outperform typical BRNN architectures in terms of accuracy in the PSSP problem [Heffernan et al., 2017].

4.2.2 Methodology

Our methodology is based on BRNN architectures. This methodology has been developed to handle problems with sequential data where both the upstream and the downstream information of a sequence is important for predicting a specific time-step. Based on the advantages and disadvantages of first and second order learning algorithms which have been analysed in *Chapter 2*, we have chosen to apply the SCG learning algorithm for BRNNs to handle this unique class of problems.

4.2.2.1 The BRNN Architecture

The BRNN architecture is based on the work of Baldi et al. [1999] and is well described in (*Section 2.1.3*) (*Figure 3*). In contrast to this work, we use:

1. Rectified Linear Unit (ReLU) activation function instead of sigmoid activation function. ReLU in many cases appears to minimize the vanishing gradient problem [LeCun et al., 2015]. Moreover, it has been shown that unfolded BRNNs can be trained more efficiently with ReLU [LeCun et al., 2015].
2. Mean Square Error (MSE) function instead of log likelihood error function.

In this RNN architecture, Layer 0 (*Figure 3*) is not an active layer. However, layers 1 and 2 have a ReLU activation function (*Equation 29*), while layer 3 is a softmax output layer. The individual transfer function takes a summed input of the previous layer's output multiplied by its corresponding weights, except for boxes U , F and B , where box U stands

for input nodes, F for the set of forward states and B for the set of backward states, where state is the output vectors of forward and backward RNNs at each previous time-step. The links between boxes oF and F and between boxes oB and B represent the recursive connections providing the information of the given number of states of current input U .

$$a(x) = \max(0, x) \quad (29)$$

$$F_t = \phi(F_{t-1}, U_{f,t}) \quad (30)$$

$$B_t = \beta(B_{t+1}, U_{b,t}) \quad (31)$$

where $U_{f,t}$ is the input nodes to hF and $U_{b,t}$ is the input nodes to hB .

The state functions of the BRNN are given by (30) and (31), with boundary conditions as defined in Baldi et al. [1999]. In Figure 3, Layer 0 is the input layer which takes F_{t-1} , U_t and B_{t+1} directly as its inputs, and Layer 1 is the hidden layer with inputs (32), (33) and (34). Moreover, the hidden layer outputs are given by (35), (36) and (37).

$$\begin{aligned} hF_{j,t-1} &= \sum_{l=1}^{N_{\phi'}} wF_{j,l}^{IH} \cdot f_{l,t-1} \\ &+ \sum_{l=1}^{N_{u_f}} wU_{j,l}^{IH-F} \cdot u_{l,t} \quad j = 1 \dots N_{\phi} \end{aligned} \quad (32)$$

$$hU_{j,t} = \sum_{l=1}^{N_{uc}} wU_{j,l}^{IH} \cdot u_{l,t} \quad j = 1 \dots N_n \quad (33)$$

$$\begin{aligned}
hB_{j,t+1} &= \sum_{l=1}^{N_{\beta'}} wB_{j,l}^{IH} \cdot b_{l,t+1} \\
&+ \sum_{l=1}^{N_{ub}} wU_{j,l}^{IH-B} \cdot u_{l,t} \quad j = 1 \dots N_{\beta}
\end{aligned} \tag{34}$$

$$h_{j,t}^F = \alpha(hF_{j,t-1}) \tag{35}$$

$$h_{j,t}^U = \alpha(hU_{j,t}) \tag{36}$$

$$h_{j,t}^B = \alpha(hB_{j,t+1}) \tag{37}$$

where N_{ϕ} , N_n and N_{β} are the numbers of hidden nodes in the corresponding sets, that is, forward state, input and backward state, N_{uf} is the dimensionality of the input vector to hF , N_{uc} is the dimensionality of the input vector to hU , N_{ub} is the dimensionality of the input vector to hB , $N_{\phi'}$ and $N_{\beta'}$ represent the number of nodes in boxes oF and oB and $\alpha()$ is the ReLU activation function (Equation 29).

Furthermore, Layer 2 in *Figure 3* represents the state layer of the architecture which has inputs as defined by (38) and (39) outputs are defined by (40) and (41).

$$oF_i = \sum_{j=1}^{N_\phi} wF_{ij}^{HO} h_{j,t}^F \quad i = 1 \dots N_{\phi'} \quad (38)$$

$$oB_i = \sum_{j=1}^{N_\beta} wB_{ij}^{HO} h_{j,t}^B \quad i = 1 \dots N_{\beta'} \quad (39)$$

$$f_{i,t} = \alpha(oF_i) \quad (40)$$

$$b_{i,t} = \alpha(oB_i) \quad (41)$$

Finally, Layer 3 in *Figure 3* represents the output layer oU of the network, which is a function of F_{t-1} , U_t and B_{t+1} as shown in (42). The network's output is given by $Y_{i,t} = \text{softmax}(oU_{i,t})$ with:

$$\begin{aligned}
 oU_{i,t} = & \sum_{j=1}^{N_{\phi'}} wF_{ij} \cdot f_{i,t} \\
 & + \sum_{j=1}^{N_{\beta'}} wB_{ij} \cdot b_{i,t} \\
 & + \sum_{j=1}^{N_{hU}} wU_{ij} \cdot hU_{i,t}
 \end{aligned} \tag{42}$$

where N_{hU} is the dimensions of hU layer; i stands for the position of a neuron in oU ; wF_{ij} , wB_{ij} and wU_{ij} are the connection weights between layer 2 and layer 3. Finally, $f_{i,t}$, $b_{i,t}$ and $hU_{i,t}$ are the outputs of each neuron at time t of oF , oB and oU , respectively.

4.2.2.2 Development of the HR-SCG Algorithm for BRNNs

We have mathematically analysed and developed the corresponding training formulas and optimisation procedure of the SCG learning algorithm for our BRNN architecture. We call our methodology HR-SCG algorithm. Our methodology is hybrid because it uses both the SCG and GD learning algorithms. The HR-SCG formulas were derived from the unfolded version of our architecture. Since stationarity is assumed, the connection weights do not change over time and the unfolding architecture of the BRNN (*Figure 25*)

is as indicated in the work of Baldi et al. [1999]. In our work, we have used the cost function:

$$E = \frac{1}{2\psi} \sum_{p=1}^{\Psi} \sum_{i=1}^s (\bar{y}_{p,i,t} - y_{p,i,t})^2 \quad (43)$$

where Ψ is the number of training patterns, s the number of neurons in the output layer, $\bar{y}_{p,i,t}$ the target output and $y_{p,i,t}$ the system output.

The general formula of the partial derivative of E with respect to any weight in Figure 3 can be written as below (16):

$$\begin{aligned} \frac{\partial E}{\partial w} &= \frac{1}{\psi} \sum_{t=1}^T \left(\frac{\partial E}{\partial y_{i,t}} \cdot \frac{\partial y_{i,t}}{\partial oU_t} \cdot \frac{\partial oU_t}{\partial w} \right) \\ &= \frac{1}{\psi} \sum_{t=1}^T [(\bar{y}_{i,t} - y_{i,t}) \cdot y_{i,t} \cdot (1 - y_{i,t}) \cdot \frac{\partial oU_t}{\partial w}] \end{aligned} \quad (44)$$

where T is the moving amount of the sliding window size.

The unfolded architecture of the BRNN (Figure 25) is again inspired from the work of Baldi et al. [1999]. The time step t represents the relative positions of patterns and is decided by a sliding window of predefined-length. While this window is sliding, the corresponding I/O pair and the states of the current time step are determined. As we have already mentioned, since stationarity is assumed, the connection weights do not change over time.

$$k_{i,t} = (\bar{y}_{i,t} - y_{i,t}) \cdot y_{i,t} \cdot (1 - y_{i,t}) \quad (45)$$

$$hB_t = wB_{IH} \cdot B_{t+1} + wU_{ihB} \cdot U_{b,t} \quad (51)$$

$$\begin{aligned} a'(x) &= 0 & \text{for } & x < 0 \\ a'(x) &= 1 & \text{for } & x \geq 0 \end{aligned} \quad (52)$$

Finally, based on (44), we have calculated the derivatives which are used from the HR-SCG algorithm for training the weights of the BRNN architecture in *Figure 3*. Based on (30)-(42), we have set the auxiliary formulas $k_{i,t}$, oU_t , F_t , n , B_t , hF_t and hB_t , which appear in (45)-(51), respectively. These equations have been used with the partial derivative function (44) and the derivative of ReLU activation function (52) to calculate the partial derivatives on each layer of the BRNN weight vector as it appears in *Figure 3*. The final formulas can be seen in (53)-(62).

$$\frac{\partial E}{\partial w_{F_{HO}}} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot w_F \cdot hF_t) \quad (53)$$

$$\frac{\partial E}{\partial w_{B_{HO}}} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot w_B \cdot hB_t) \quad (54)$$

$$\frac{\partial E}{\partial w_F} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot F_t) \quad (55)$$

$$\frac{\partial E}{\partial w U_{HO}} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot n) \quad (56)$$

$$\frac{\partial E}{\partial w B} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot B_t) \quad (57)$$

$$\frac{\partial E}{\partial w F_{IH}} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot w F \cdot w F_{HO} \cdot F_{t-1}) \quad (58)$$

$$\frac{\partial E}{\partial w B_{IH}} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot w B \cdot w B_{HO} \cdot B_{t+1}) \quad (59)$$

$$\frac{\partial E}{\partial w U_{IH}} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot w U_{HO} \cdot U_t) \quad (60)$$

$$\frac{\partial E}{\partial w U_{ihF}} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot w F \cdot w F_{HO} \cdot U_{f,t}) \quad (61)$$

$$\frac{\partial E}{\partial w U_{ihB}} = \frac{1}{\Psi} \sum_{t=1}^T (k_{i,t} \cdot w B \cdot w B_{HO} \cdot U_{b,t}) \quad (62)$$

$$E'(w) = \left[\frac{\partial E}{\partial w_{F_{HO}}}, \frac{\partial E}{\partial w_{B_{HO}}}, \frac{\partial E}{\partial w_{F_{HO}}}, \frac{\partial E}{\partial w_{U_{HO}}}, \frac{\partial E}{\partial w_B}, \frac{\partial E}{\partial w_{F_{IH}}}, \frac{\partial E}{\partial w_{B_{IH}}}, \frac{\partial E}{\partial w_{U_{IH}}}, \frac{\partial E}{\partial w_{U_{ihF}}}, \frac{\partial E}{\partial w_{U_{ihB}}} \right] \quad (63)$$

After the partial derivatives of the cost function (43) with respect to individual weights are calculated, they are directly fed to the HR-SCG algorithm (*Algorithm 1*). These formulas (53)-(62) are used to calculate the unfolded BRNN gradient vector (63) which will be used by the HR-SCG algorithm to update the BRNN weight vector.

Based on the analytical calculation of the BRNN gradient vector E' , we then use the HR-SCG algorithm to train BRNNs. At the beginning of the algorithm, we initialize the model's weight vector W . This is done by the Xavier initialization method [Glorot and Bengio, 2010]. In the case of RNNs, initialization of a weight vector with very small or very big weights must be avoided because it can lead to the vanishing and exploding gradient problems. Through our work many different weight vector initialization approaches have been used based on a constant value W_{init} and an initialization range of $-W_{init} < W < W_{init}$ but they failed. In contrast, the Xavier weight initialization method is based on the network's size and it helps the error signal to reach deep into the network. Although this method has been justified statistically only for sigmoid units [Glorot and Bengio, 2010], it has been used to initialize the weights of our model in a distribution which helps the optimization process to converge to a meaningful solution. Most of them have been failed. Furthermore, we have initialized all the network's recurrent weights

with an identity matrix [Le et al., 2015]. In this way, all the new hidden state vectors are copies of previous hidden vectors with an addition of the effect of current inputs replacing all negative states by zero. Therefore, the error derivatives for the hidden units remain constant and no extra error-derivatives are added when they are backpropagated through time. Generally, this method also helps to overcome the vanishing gradient and the exploding gradient problems. Finally, the softmax output layer helps to avoid flattened saddle points [Dunne and Campbell, 1997], which can lead to a slow convergence rate.

As it can be seen in *Algorithm 1*, at the beginning of each HR-SCG iteration, the direction vector p_k is set to the negative of the model gradient vector E' and the Scaling Parameter S_{css} is recalculated. Then the second order information S_k is calculated. The Hessian Matrix calculation demands $O(N^2)$ memory usage and $O(N^3)$ calculation complexity. To surpass this complexity, the SCG uses a Hessian Matrix approximation calculation based on the model's gradient vector and the direction vector to calculate S_k . Then, a LM approach is used to set the δ_k parameter, which identifies if the Hessian Matrix is positively defined. This is a necessary condition for function minimization procedures. If δ_k is negative, the algorithm has to make the matrix positively defined. This information is used to calculate the algorithm's step size. Thus, the step size and direction vector are used to calculate the new weight vector. At this step, if the error value is less than the error at the previous iteration, the new weight vector is applied to the model. If the error is not decreased for a number of iterations, it would mean that we have approached a minimum. At this final point the GD algorithm is set for further optimization. Otherwise, if the algorithm cannot converge, it is restarted.

One of the most undesirable difficulties during the training of a RNN is the vanishing gradient problem. A mechanism which introduces shortcut connections between the forward and backward states of the time series has been used to form shorter paths along the sequence where gradients can be propagated [Baldi et al., 1999]. Therefore, the gradient information at each time step t includes a strong signal from the whole sequence to encounter for the vanishing gradient problem and consequently avoid long range dependencies elimination. Thus, (64), (65) and (66) are used to calculate the final F_t , B_t and O_t , respectively.

$$F_t = \phi(F_{t-1}, F_{t-s}, U_{f,t}) \quad (64)$$

$$B_t = \beta(B_{t-1}, B_{t-s}, U_{b,t}) \quad (65)$$

$$O_t = \eta(F_{t-s}, \dots, F_{t+s}, B_{t-s}, \dots, B_{t+s}, U_{c,t}) \quad (66)$$

where F_{t-s}/F_{t+s} and B_{t-1}/B_{t+1} are the previous/next context states of F_t and B_t , respectively and U_t is the input vector at t .

Furthermore, compared to the original SCG, we have introduced some minor modifications to the HR-SCG algorithm in order to increase the convergence rate and the ability of the algorithm to search for the best solution in a complicated error surface of such a BRNN network:

Gradient Descent Method: In this work, the GD method has been combined with the SCG learning algorithm to create a Hybrid first and second order learning algorithm. Along the training procedure of a large BRNN model with complicated sequential data, the error surface tends to be rough with multiple local minima. In this case, the SCG step size might be too long for the algorithm to converge in a narrow valley. As a result the error value may remain stable and the algorithm proceeds to restart mode. To surpass this problem, before the algorithm is restarted, the HR-SCG algorithm switches to GD mode with a very small learning rate. Consequently, the GD component of the HR-SCG learning algorithm is using very small steps to reach the minimum. This can be seen in lines 34-41 of *Algorithm 1*. Since the SCG has already reached a minimum, only a few steps of the GD are necessary.

Adaptive Step Size Scaling Parameter: We have introduced an adaptive step size scaling parameter S_{cSS} at step 7 of the SCG algorithm as defined in [Møller, 1993b]. This parameter is recalculated in each iteration of the algorithm as in (67). As it is well known, a learning rate parameter, which appears in the GD learning algorithm, is highly related to the optimization algorithm's step size. Thus, we have modified the algorithm's update weight vector rule to introduce a learning rate like parameter. This can be seen in (67),(68).

$$S_{cSS}(k) = S_{cSS} \exp(-k) \quad (67)$$

$$w_{k+1} = w_k + S_{cSS}(k)a_k p_k \quad (68)$$

where k is the iteration number, a_k is the step size and p_k is the direction vector.

During the first iterations of the algorithm this scalar is high, assuming that the algorithm has identified a direction to a minimum. Hence, we force the algorithm to use a bigger step size in a specific direction to approach a minimum faster. The adaptive scaling parameter is exponentially decreasing as the algorithm approaches a minimum to avoid losing the lowest point of the curve. Furthermore, this parameter is redefined each time the SCG algorithm restarts (line 32 of *Algorithm 1*). Empirically, the use of this parameter is mandatory for training a complicated BRNN architecture with the SCG algorithm. This parameter can be seen in the weight vector update rule at lines 27 and 28 of *Algorithm 1*.

Restart Algorithm Condition: The original SCG algorithm is restarted if the number of learning iterations surpasses the number of the network's parameters. However, this condition is meaningful only if the algorithm is used to optimize a quadratic function. Clearly, in the case of the BRNN architecture this condition fails because the error surface is much more complicated than a quadratic function. Hence, we have chosen to restart the algorithm only if the algorithm shows slow development (training error $< 10^{-7}$) and after the constant number of 20 iterations. Furthermore, our implementation algorithm, before a restart, stores all the weight vectors and the respective training errors. Finally,

after the algorithm reaches the final training iteration, it returns a trained model with the weight vector which was assigned to the lowest training error. Consequently, this version of the algorithm is widely exploring the respective error surface and is less likely to get stuck for a long time in a local minimum. This procedure can be seen in lines 33 to 53 of *Algorithm 1*.

Dropout Method: During the training of BRNN architectures and more generally in any large deep network, over-fitting may become a big problem. Dropout is a method which was introduced in [Srivastava et al. \[2014\]](#) to address this problem by randomly dropping units during the training procedure. Consequently, this prevents units from co-adapting too much. This procedure is executed by *doDropout()* function in line 3 of *Algorithm 1*.

4.2.2.3 HR-SCG Time Complexity Analysis

The time complexity of the HR-SCG algorithm can be calculated based on the SCG and GD time complexities. More specifically, each iteration of the SCG algorithm has one call of $E(w)$ and two calls of $E'(w)$. For a network with N free parameters, the time complexity of SCG per iteration is $O(3N^2)$, which can be reduced to $O(2N^2)$ because the calculation of $E'(w)$ may include $O(N^2)$ [[Møller, 1993b](#)]. Furthermore, the time complexity of the GD algorithm is $O(N^2)$. However, our algorithm is not using the GD optimization at each iteration, so the correct time complexity is:

$$O(N^2) + O(2N^2) + O(M * N^2) = O(N^2 + (M * N^2))O(N^2) \quad (69)$$

Algorithm 1 The HR-SCG BRNN learning algorithm

Require: initialize weight vector w_k , maximum training epochs T , Scalar S_{cSS} , scalars $\sigma > 0$, $\lambda_l > 0$, $\bar{\lambda}_l = 0$
 1: $k = 0$, $p_k = -E'(w_k)$, $r_k = -E'(w_k)$, $\kappa = 1$, success=true
 2: **repeat**
 3: *doDropout()* //as in Srivastava et al. [74]
 4: **if** success = true **then**
 5: //calculate second order information
 6: $\sigma_\kappa = \frac{\sigma}{|p_\kappa|}$
 7: $s_\kappa = \frac{E'(w_\kappa + \sigma_\kappa p_\kappa) - E'(w_\kappa)}{\sigma_\kappa}$
 8: $\delta_k = p_k^T s_k$
 9: **end if**
 10: //scale s_k
 11: $s_k = s_k + (\lambda_k - \bar{\lambda}_k)p_k$
 12: $\delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k)|p_k|^2$
 13: //make the Hessian Matrix positive definite
 14: **if** $\delta_k \leq 0$ **then**
 15: $s_k = s_k + (\lambda_k - 2\frac{\delta_k}{|p_k|^2})p_k$
 16: $\bar{\lambda}_k = 2(\lambda_k - \frac{\delta_k}{|p_k|^2})$
 17: $\delta_k = -\delta_k + \lambda_k|p_k|^2$
 18: $\lambda_k = \lambda_k$
 19: **end if**
 20: //calculate step size α_k
 21: $\mu_k = p_k^T r_k$
 22: $\alpha_k = \frac{\mu_k}{\delta_k}$
 23: //calculate the comparison parameter Δ_k
 24: $\Delta_k = \frac{2\delta_k[E(w_k) - E(w_k + \alpha_k p_k)]}{\mu_k^2}$
 25: **if** $\Delta_k > 0$ **then**
 26: //successful reduction in error can be made
 27: $S_{cSS}(k) = S_{cSS} \exp(-k)$
 28: $w_{k+1} = w_k + S_{cSS}(k)\alpha_k p_k$
 29: $r_{k+1} = -E'(w_{k+1})$
 30: $\bar{\lambda}_k = 0$
 31: success=true
 32: //restart algorithm if N=0 or error remains stable
 33: **if** k mode N=0 or error is stable for 10 iterations **then**
 34: //Gradient descent mode
 35: **if** error is very low **then**
 36: $t = 0$
 37: **repeat**
 38: $w_t = w_k$
 39: $w_{t+1} = w_k + \alpha_k E'(w_t)$
 40: $t = t + 1$
 41: **until** error is stable
 42: **end if**
 43: //iterations surpasses the number of the network's parameters or error is stable \rightarrow restart algorithm
 44: $p_{k+1} = r_{k+1}$
 45: **else**
 46: $\beta_k = \frac{|r_{k+1}|^2 - r_{k+1}r_k}{\mu_k}$
 47: $p_{k+1} = r_{k+1} + \beta_k p_k$
 48: **end if**
 49: **if** $\Delta_k \geq 0.75$ **then**
 50: //reduce the scale parameter
 51: $\lambda_k = 1/2\lambda_k$
 52: **end if**
 53: **else**
 54: //reduction in error is not possible
 55: $\lambda_k = \lambda_k$
 56: success=false
 57: **end if**
 58: **if** $\Delta_k < 0.25$ **then**
 59: //increase the scale parameter
 60: $\lambda_k = 4\lambda_k$
 61: **end if**
 62: **if** $r_k = 0$ **then**
 63: //terminate the algorithm
 64: return w_{k+1}
 65: **end if**
 66: $k = k + 1$
 67: **until** $k = T$

where M is the number of GD's iterations.

4.2.2.4 Training BRNNs with the HR-SCG

In this work, the HR-SCG learning algorithm has been used for training BRNNs to handle the PSSP problem. Our methodology appears in *Figure 26*. As it can be seen, we have 5 levels of processing. In the first level of our methodology, the PS of protein sequences appears in MSA encoding. Input data is used from multiple BRNN classifiers in the second level of our methodology. These BRNN classifiers are trained with the HR-SCG method. Each BRNN returns three real values for the central residue of the local window, one for each secondary structure state. Subsequently, the corresponding outputs of each BRNN for each state are averaged through the ensembles level. Then, the resulting predictions from the ensembles level are used for filtering [Kountouris et al., 2012]. Finally, the last level of our methodology returns three real values, which represent the predicted SS class for specific input data.

4.2.3 Results and Discussion

The HR-SCG methodology has been applied to the PSSP problem and through a thorough experimental analysis, various results have been extracted. These results demonstrate the efficiency and the effectiveness of the algorithm.

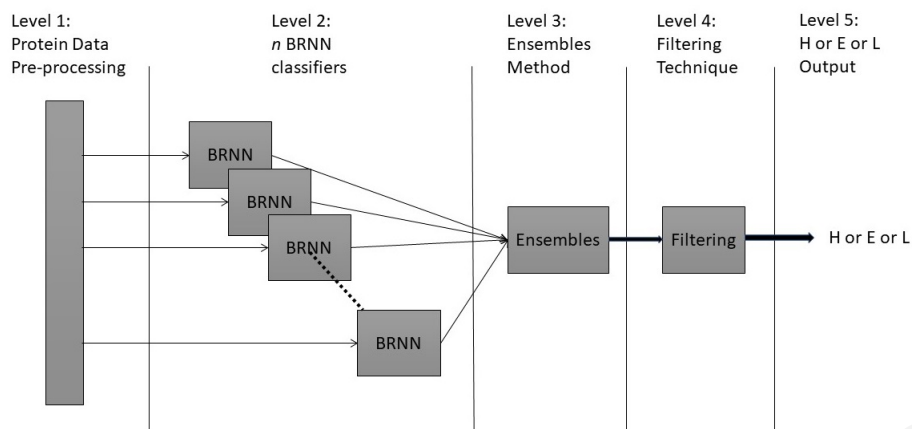


Figure 26: **The HR-SCG BRNN methodology for the PSSP problem:** The methodology has 5 levels of processing. Level 1 is used for Protein Data Pre-processing to feed each one of the BRNNs in Level 2. Level 3 and Level 4 use Ensemble and Filtering Methods, respectively, to present a final output in Level 5.

4.2.3.1 Data Preparation and Simulation Details

Special care has been taken to retrieve datasets of the highest possible quality for the PSSP problem relying in specialized resources. As we have already mentioned in *Sections 2.3.6*, for the purposes of this work, we have used the CB513 [Cuff and Barton, 1999] and the PISCES [Wang and Dunbrack Jr, 2003; Kieslich et al., 2016] datasets, which consist of 513 and 8632 protein chains respectively. More specifically, high resolution protein structural data have been obtained from the RCSB Protein Data Bank (PDB; <http://www.rcsb.org/>). The data have been preprocessed as it has been described in *Sections 2.3.6.1-2.3.6.4*. During this procedure, the MSA files of CB513 and PISCES datasets have been analyzed and cleaned up from data with short or no information. More specifically, the PSI-BLAST has created some MSA files with arrays of zeros or arrays with less amount of elements than expected. The respective protein sequences (342 in

total) have been excluded from the training and testing datasets (*Table 8*). Furthermore, we have followed a strict 10-fold cross-validation approach for the CB513 dataset and 6-fold cross-validation approach for the PISCES dataset as described in [Kountouris et al. \[2012\]](#) and [Wang and Dunbrack Jr \[2003\]](#); [Kieslich et al. \[2016\]](#), respectively.

Dataset Name	Protein Sequence Name
CB513	1COIA, 1MCTI, 1TIIC, 2ERLA, 1CEOA, 1MRTA, 1WFBB, 6RLXC
PISCES	1BB1A, 1BB1B, 1BB1C, 1C94A, 1DPJB, 1DTDB, 1F8VD, 1GWMA, 1HX6A, 1KD8A, 1KP6A, 1KVEA, 1L2WI, 1M3WA, 1M45B, 1M46B, 1MCTI, 1MQSB, 1MW5A, 1O06A, 1P9IA, 1PJMA, 1PJNA, 1SVFB, 1T01B, 1TQEX, 1TTWB, 1U6HB, 1UVQC, 1ZVZB, 1ZW2B, 2BPA3, 2BPTB, 2C5IP, 2C5KP, 2DS2A, 2ERLA, 2GWWB, 2HZSI, 2MLTA, 2P06A, 2PBDV, 2PLXB, 2QUOA, 2W4YA, 2WBYC, 2WFUA, 2WFVA, 2WWXB, 2X5CA, 2X5RA, 2XF7A, 2XZEQ, 3AJBB, 3C5TB, 3DT5A, 3E8YX, 3FBLA, 3GP2B, 3H0TC, 3HE4B, 3HE5A, 3HE5B, 3L9AX, 3LCNC, 3LJMA, 3M6ZA, 3NK4C, 3OWTC, 3P06A, 3PLVC, 3QKSC, 3R46A, 3R4AA, 3RA3B, 3RF3C, 3RKLA, 3S1BA, 3S6PE, 3SHPA, 3SJHB, 3TQ2A, 3TWEA, 3TZ1B, 3U4VA, 3U4ZA, 3UC7A, 3UKXC, 3UL1A, 3V62C, 3V86A, 3VU5B, 3VU6B, 3VVIA, 3W8VA, 3W92A, 3WKNE, 3WOEB, 3WX4A, 3WY9C, 3ZTAA, 4A94C, 4BFHA, 4BLQA, 4BPLB, 4C1AA, 4CAYC, 4CU4B, 4CXFB, 4DACA, 4EHQG, 4F87A, 4FBWC, 4FTBD, 4FZ0M, 4G1AA, 4GVBB, 4H62V, 4H7RA, 4H8MA, 4H8OA, 4HB1A, 4HBEA, 4HLBA, 4HR1A, 4I7ZE, 4IIKA, 4J4AA, 4JGLA, 4JHKC, 4KVTA, 4KYTB, 4LOOB, 4M1XA, 4M6BC, 4MGPA, 4MI8C, 4N3BB, 4N3CB, 4OGQE, 4OQ9A, 4OYDB, 4OZKA, 4PC0C, 4PN8A, 4PN9A, 4PNA, 4PNBA, 4PNDA, 4PW1A, 4QMFA, 4R0RA, 4R80A, 4R8TA, 4RIQC, 4TTLA, 4UEBB, 4W6YA, 1M1QA, 1Q2HA, 1Q9UA, 1QV9A, 1SQWA, 1TU9A, 1U9LA, 1UV7A, 1V96A, 1VPRA, 1VR4A, 1WPNA, 1WV3A, 1WWPA, 1YPYA, 1Z4RA, 2BDRA, 2C0NA, 2CVIA, 2D59A, 2D68A, 2D7EA, 2E1FA, 2ERWA, 2FB0A, 2F1A, 2G7SA, 2HL7A, 2HQLA, 2HX5A, 2IP6A, 2NPTA, 2NS0A, 2O1QA, 2O38A, 2O4AA, 2O99A, 2OL5A, 2OU1A, 2OU5A, 2OX7A, 2P63A, 2P9XA, 2PW9A, 2Q3SA, 2Q3TA, 2R19A, 2R5SA, 2R85A, 2UVPA, 2V73A, 2V9PA, 2VS0A, 2WG8A, 2WVQA, 2Y5PA, 2YF2A, 2ZEXA, 3B0FA, 3BPQA, 3BRVA, 3C0DA, 3C4RA, 3CPTA, 3CRYA, 3CSXA, 3D33A, 3D30A, 3D55A, 3DFUA, 3DNXA, 3E0RA, 3ESLA, 3ESMA, 3F43A, 3F67A, 3F95A, 3FF5A, 3FH3A, 3G21A, 3GO9A, 3GP6A, 3H0DA, 3H0NA, 3H16A, 3H35A, 3HLSA, 3I00A, 3I4UA, 3I76A, 3IBWA, 3IV4A, 3JQOA, 3K8RA, 3KTOA, 3KUPA, 3KVPA, 3L49A, 3L60A, 3L7HA, 3LQ9A, 3M5QA, 3MD9A, 3N7XA, 3NOQA, 3NR5A, 3NS4A, 3O12A, 3O65A, 3O6QA, 3ONQA, 3OP6A, 3P51A, 3PD7A, 3PL0A, 3Q0HA, 3Q18A, 3Q6CA, 3QH6A, 3RK6A, 3TE8A, 3TS9A, 3TUOA, 3TVQA, 3U5WA, 3U97A, 3UV0A, 3UV1A, 3VS8A, 3W6SA, 3ZC0A, 4AP5A, 4AQOA, 4BOQA, 4BSVA, 4BSXA, 4DHXA, 4E6SA, 4E6WA, 4F27A, 4F2LA, 4F4WA, 4F87A, 4FX7A, 4G97A, 4G0FA, 4GT9A, 4GUCA, 4H41A, 4HHVA, 4I16A, 4I86A, 4IHQA, 4J1VA, 4J5RA, 4J91A, 4JX0A, 4K12A, 4K92A, 4KQIA, 4KTWA, 4L2WA, 4L3UA, 4LKUA, 4LQBA, 4LTBA, 4MO0A, 4MTUA, 4MYVA, 4N74A, 4NUAA, 4P2VA, 4P49A, 4P78A, 4PF3A, 4PSFA, 4Q53A, 4Q7OA, 4QRNA, 4QSGA, 4R7RA, 4U9OA, 4X33A

Table 8: Protein sequences which have been excluded from the CB513 and PISCES dataset. These protein sequences have been excluded from the datasets because of miscalculations of the PSI-BLAST.

As it has already been explained in *Section 2.1.3*, the BRNN architecture consists of a FFNN and two RNNs (FRNN and BwRNN), where the contextual information is contained. The network’s input vector at each time step t consists of the MSA information

A/A	FP	W_a	W_{fb}	W_c	S_{cSS}	n	h_n	N_ϕ	S_{fb}	S_{out}	$Q_3(\%)$
1	2761	25	3	3	1000	11	11	11	2	2	69.10
2	2761	25	3	3	10	11	11	11	2	2	69.64
3	2761	31	3	3	100	11	11	11	2	2	70.26
4	4178	31	3	15	10	7	10	7	2	0	72.90
5	5735	27	3	13	10	10	15	10	2	0	73.40
6	6484	23	3	11	10	11	20	11	2	0	71.80
7	7650	31	3	15	10	10	20	10	1	0	64.00
8	7850	27	3	15	10	10	20	10	2	0	73.84
9	8050	31	3	15	10	10	20	10	3	0	73.70
10	9038	27	3	13	10	7	30	7	2	0	73.10

Table 9: Experimental results using the CB513 dataset, where FP is the network’s Free Parameters (see text for description of the parameters)

contained in the sliding window W_α and the target output is the respective class of the amino acid which is located in the center of W_α . The *FRNN* iteratively processes the $(W_\alpha - 1)/2$ residues located on the left side of the position t and the *BwRNN* iteratively processes the $(W_\alpha - 1)/2$ residues located on the right side of the position t .

4.2.3.2 Optimizing BRNN and HR-SCG parameters

A single BRNN has been trained on different single folds of the CB513 dataset. At this stage, we carried out multiple experiments to tune up BRNN and HR-SCG parameters to extract the desirable results. The final Q3 and SOV results of each simulation were extracted as the average of 10 different simulations on the same parameters of the BRNN and HR-SCG algorithm.

One of the most important parameters with a big impact on the results is the sliding window size. Particularly, we have used 3 window size parameters (*Figure 3*). The W_a parameter stands for the sliding window size on the PS. The W_{fb} window parameter

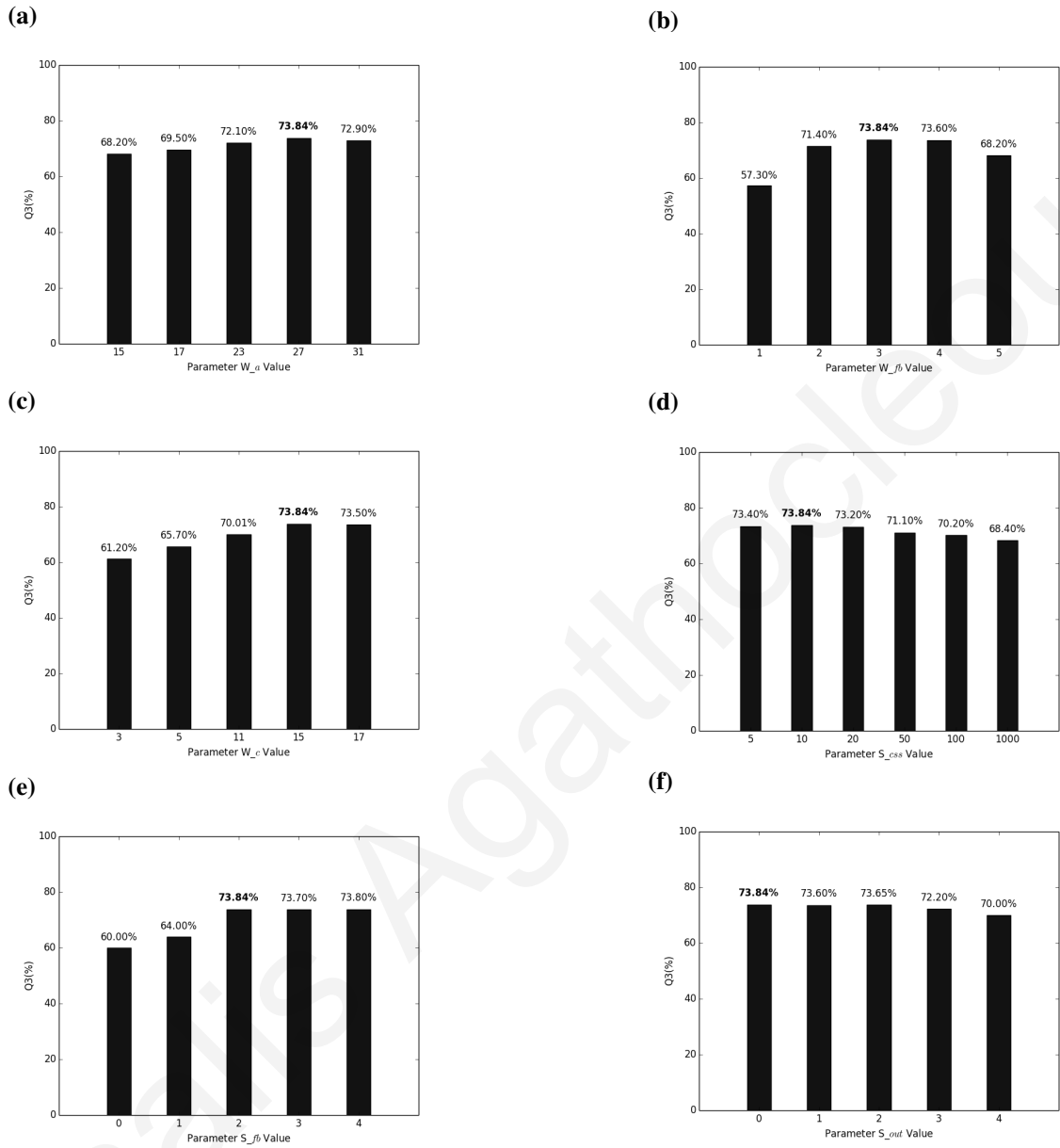


Figure 27: Experimental results using the CB513 dataset as a test set for (a) W_a , (b) W_{fb} , (c) W_c , (d) S_{css} , (e) S_{fb} and (f) S_{out} . The six bar charts show the impact of most important BRNN parameters on the final results. A single BRNN has been trained at each time with HR-SCG algorithm on the CB513 dataset. Multiple experiments have been carried out to tune up BRNN parameters to maximize results (see text for description of the parameters and results).

represents the number of residues that are used as input to F_t and B_t at each time step t . Subsequently, a sliding window of W_{fb} amino acids is moving from left to right for $(W_a - 1)/2$ residues to feed the F_t and respectively a sliding window of W_{fb} amino acids is moving from right to left for $(W_a - 1)/2$ residues to feed the B_t . Finally, the W_c window parameter represents the number of W_a residues which are located at the center of the window and are used as input to h_U layer of the FFNN. The exact number of input nodes for each window is $20 \cdot w$ (w :window size), since each position in the PS is encoded by 20 values in the MSA representation. Furthermore, we had to tune up the parameters that determine the network's architecture: n is the length of the context vectors oF and oB , while N_u indicates the number of hidden units in h_U layer and similarly N_ϕ indicates the number of hidden units in h_F and h_B layers. We have also used the already mentioned adaptive step size scaling parameter S_{cSS} . Finally, we have used the S_{fb} and S_{out} , which are the number of additional consecutive context vectors in the future and the past of F_t/B_T and O_t , respectively. The performance of our model has been evaluated by the percent of correctly predicted residues Q3 [Richards and Kundrot, 1988]. To extract the optimized parameters, we have performed grid search to find the set of optimum parameters. The most indicative results can be seen in *Table 9*. The highest Q3 accuracy which has been achieved was 73.84%, for: $W_a = 27$, $W_{fb} = 3$, $W_c = 15$, $S_{cSS} = 10$, $n = 10$, $N_u = 20$, $N_\phi = 10$ and $S_{fb} = 2$ and $S_{out} = 0$.

After we have tuned up the network architecture, we have investigated the impact of the most important parameters on the results. These results can be seen in *Figure 27* where we have kept the parameters which gave the highest accuracy constant except the

one which is investigated. We have noticed that in order to maximize the algorithm's performance the three windows W_a , W_{fb} and W_c must have values of 27, 3 and 15, respectively. Thus, shorter sliding windows do not provide the network with enough information and longer sliding windows cannot be captured by the network. As it can be seen from experiments in *Figure 27(a)*, *28(b)* and *28(c)*, the tuning up of these parameters can vary the algorithm's performance by more than 15%. The biggest variation in the algorithm's performance around the optimum parameters comes from the variation of W_{fb} . Finally, a window of 1 or 2 amino acids does not contain enough information and information of 5 amino acids cannot be captured by the network. Finally, the overall results obtained using a window of 4 amino acids looks very similar to those obtained by a window of 3 amino acids but we have chosen the later since it leads to a simpler network. Furthermore, it is obvious that the network is fed with more information as we increase the center window W_c length. Consequently, as we can see from *Figure 27(c)* the network's accuracy is increased. The upper bound for this statement is when length W_c reaches 15. Further to this value the network's accuracy remains practically stable.

In addition, we have noticed that the S_{cSS} parameter should be set to 10 to increase the convergence rate for this problem. During our simulations, we have used multiple values for this parameter, but only if this number was in power of 10 we have noticed significant changes on the results. As it can be seen in *Figure 27(d)*, by dropping this parameter from 100 to 10, Q3 increased approximately 3.5%. The final Q3 metric has also increased by 7% after we increase the S_{fb} parameters from 1 to 2, as it can be seen in *Figure 27(e)*. In addition, as it can be seen from experiments in *Figure 27(f)*, the network's accuracy does

<i>Method</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	SOV	SOV_H	SOV_E	SOV_L
S	73.24 (73.69)	72.13 (75.92)	61.58 (62.77)	80.38 (78.87)	67.81 (68.01)	66.56 (66.07)	62.45 (62.02)	69.02 (70.56)
H	73.80 (74.12)	72.53 (75.92)	62.31 (62.77)	80.96 (78.87)	67.91 (68.12)	66.63 (65.98)	62.34 (62.08)	69.81 (70.32)
HD	73.85 (74.24)	72.43 (76.02)	62.47 (63.00)	81.09 (78.96)	67.93 (68.20)	66.59 (66.12)	62.29 (62.03)	70.14 (70.52)
HDE	76.24 (76.52)	75.50 (78.86)	65.22 (65.37)	82.73 (80.62)	70.83 (71.94)	70.13 (71.36)	65.96 (66.15)	71.02 (71.63)
HDEF	77.41 (77.61)	76.80 (80.35)	66.32 (65.30)	83.84 (81.99)	71.96 (72.15)	71.55 (71.85)	65.92 (66.06)	71.80 (71.65)

Table 10: Experimental results using the CB513 (PISCES) dataset under 10-(6-) fold cross-validation, where S is a single BRNN trained with SCG algorithm, H is the Hybrid method, D is the Dropout algorithm, E is the Ensemble method and F is the Filtering Technique (see text for description of the parameters)

	CB513	CB513 with Dropout	PISCES	PISCES with Dropout
Sample Standard Deviation s	0.74	0.22	0.54	0.11
Variance (Sample Standard) s^2	0.55	0.05	0.29	0.01
Population Standard Deviation σ	0.70	0.20	0.51	0.11
Variance (Population Standard) σ^2	0.49	0.04	0.26	0.01
Mean (Average)	73.80	73.85	74.12	74.24
Standard Error of the Mean	0.23	0.07	0.17	0.04

Table 11: Statistics of Q3 results for 10 different training procedures of the HR-SCG algorithm with and without Dropout method.

not have big variations when the S_{out} parameter is increased more than 0. Consequently, we have chosen to ignore this layer of network's complexity.

4.2.3.3 Methodological Enhancements

Because of the size of the BRNN, which can have more than 7000 free parameters, the weight initialization methodology was very important for the algorithm's training procedure. Weight vector initialization with values in $[-0.01, 0.01]$ were destructive for the BRNN. The Q3 accuracy was 50%-55% (results not shown) and the exploding gradient

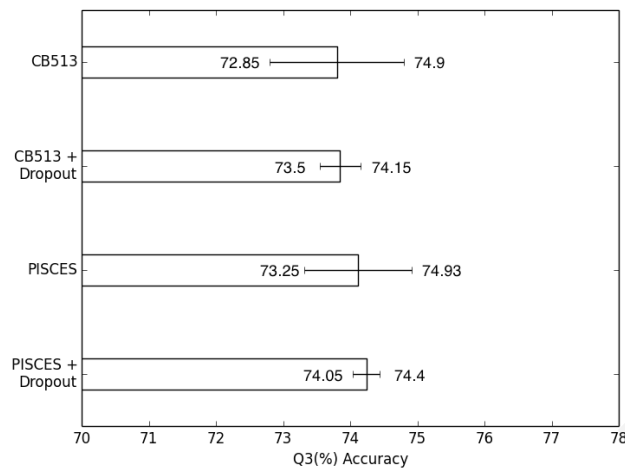


Figure 28: **The Dropout impact in HR-SCG algorithm:** Average results of 10 different training procedures of the HR-SCG algorithm. Each bar shows the minimum and maximum Q3 value of the 10 different training procedures. The dropout technique does not improve the system's accuracy but it reduces the variance of the final Q3 results from multiple experiments of the CB513 and PISCES datasets.

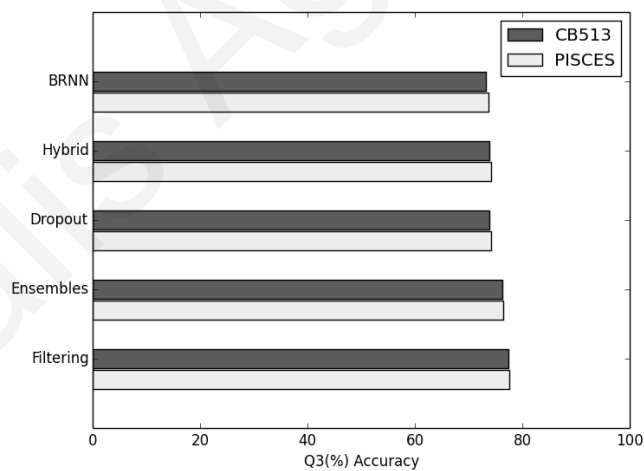


Figure 29: **The HR-SCG BRNN methodology improvements:** The improvements and ML techniques which were applied to HR-SCG methodology have increased the BRNN accuracy by approximately 4%.

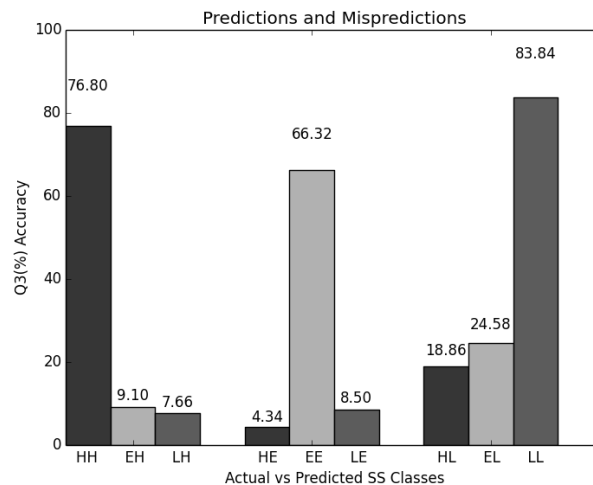
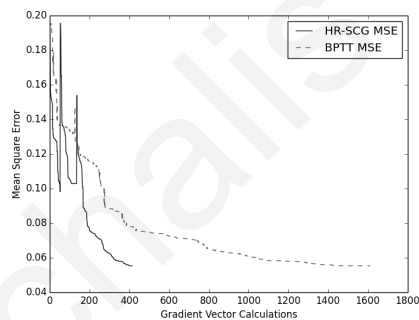


Figure 30: **Graph of Confusion Matrix for Actual vs Predicted SS Classes on the CB513 dataset:** HH, EE and LL are the True Positive scores of our method for each class after the average ensemble method and filtering technique. EH, LH, HE, LE, HL and EL are the scores for the mispredicted classes where the first letter is the actual class and the second letter is the predicted class. Based on an average of a 10-fold cross validation evaluation, the method can predict correctly with 76,80%, 66.32% and 83.84% the H, E and L classes, respectively.

(a) Square error for HR-SCG vs BPTT for CB513



(b) Multiple executions of HR-SCG algorithm for CB513

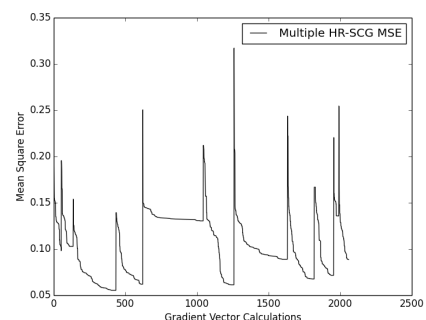


Figure 31: **Comparison of HR-SCG to BPTT learning algorithm:** On 9(a), the HR-SCG learning algorithm needs much less training iterations than the conventional BPTT learning algorithm. On 9(b) the testing error through multiple restarts of the algorithm.

problem was demonstrated. The solution to this problem was the Xavier weight initialization method which is initializing the network's weight vector with much smaller values based on the network's size. Consequently, the range of weight vector values was in $[-1 \times 10^{-4}, 1 \times 10^{-4}]$. As it can be seen in *Table 10*, the initial Q3 accuracy for a single BRNN which was trained on the CB513 and PISCES datasets before any improvements and ML techniques on the SCG learning algorithm were 73.24% and 73.69%, respectively. This accuracy level was approximately 3% lower than the 76% of [Baldi et al. \[1999\]](#) BRNN which is the obvious baseline benchmark for our algorithm.

One big improvement of our methodology is the transition between second to first order learning algorithm. As it can be seen in *Table 10*, when the GD is included in the methodology, the accuracy for both datasets is increased about 0.5%-0.6%. Consequently, the Q3 accuracy for CB513 and PISCES datasets reached the 73.80% and 74.1% of correct predictions respectively. This improvement makes our methodology hybrid and through the small step size of GD algorithm, it can approach better the local minimum. Furthermore, dropout has also been used for generalization purposes. In *Figure 28*, we can see the average results of 10 different cross-validation experiments of our methodology for the CB513 and PISCES datasets. As it can be seen, the dropout technique does not significantly improve the system's overall accuracy but it reduces the range between the lowest and highest Q3 scores in simulations with the same parameters. This can also be seen through the statistical results shown in *Table 11*. The standard deviation and consequently the variance of the Q3 results for both datasets are decreased when the Dropout method is introduced but without affecting any generalized information, as it can be seen

from the results which are explained in the next paragraphs. This result allows us to get more generalized results in less executions of the algorithm.

The major improvement in our results has been achieved with ensemble methods and filtering techniques [Kountouris et al., 2012], which follows Baldi et al. [1999]. In this paper, we employ an ensemble of six BRNNs. Each BRNN returns three real values for the central residue of the local window, one for each secondary structure state. Subsequently, the corresponding outputs for each state are averaged and, therefore, the output of the ensemble is an array of three values for each residue. The resulting predictions are then used for Multilayer Perceptron filtering, as explained in Kountouris and Hirst [2009]. These techniques improved the accuracy and made our results comparable to Baldi et al. [1999] work.

As it can be seen in *Figure 29*, the improvements and ML techniques which were applied to our methodology have increased the BRNN accuracy by approximately 4%, which make our results comparable to the results of other algorithms on the same problems. Furthermore, in *Figure 30* we can see how the algorithm can manage with each SS class. Obviously, in the case of CB513 dataset, the algorithm can predict correctly with 76.80%, 66.32% and 83.84% the H, E and L classes, respectively. Furthermore from *Figure 29* we can obtain the mispredicted residues for each class. As it can be observed, the H and E classes have much less mispredictions than the L class. Consequently, through all the enhancements appeared in *Table 10*, our methodology has achieved 77.41% Q3 accuracy and 71.96 SOV for the CB513 dataset. Furthermore, the methodology has achieved 77.61% Q3 accuracy and 72.15 SOV for the PISCES dataset.

Here, it is important to mention that for comparison purposes, the CB513 and PISCES datasets have been used to train the LSTM-BRNN of [Heffernan et al. \[2017\]](#), which has been reported as the method with highest results on the same type of data for the PSSP problem. The 10-fold cross validation on the CB513 dataset has achieved 76.2% Q3 and the 6-fold cross validation on the PISCES dataset has achieved 81.5% Q3. As it can be seen, the results of our method are slightly higher than the results of the LSTM-BRNN method for the CB513 dataset and slightly lower for the PISCES dataset, so overall we can say that for the same type of data the two methods give comparable results. Furthermore, the results of this method on our datasets are much lower than the 84% Q3 which has been reported in [Heffernan et al. \[2017\]](#). This is due to the fact that different methods have been used to create each dataset. In contrast to our datasets (*Section 2.3.6*), [Heffernan et al. \[2017\]](#) combine two different types of profiles, PSI-BLAST and Hblits.

4.2.3.4 Comparison of HR-SCG to BPTT learning algorithm

Finally, in order to compare the computational performance of HR-SCG to BPTT learning algorithm, we have used the same architecture and methodologies which gave as the optimum results for the HR-SCG learning algorithm and has been explained in detail in *Sections 5.2 and 5.3*, but trained with the BPTT learning algorithm. The results have been taken from 10-fold and 6-fold cross validation approach for CB513 and PISCES datasets, respectively.

The results on the PSSP problem have shown that a BRNN trained with both learning algorithms can capture patterns and make predictions on complicated sequences where

the information in both upstream and downstream direction is important. The BRNN trained with BPTT has achieved 77.14% and 77.8% Q3 accuracy for the CB513 and PISCES datasets, respectively. As it can be seen from the results, the Q3 accuracy on both methodologies and datasets is comparable. Furthermore, as it can be seen on *Figure 31(a)*, the HR-SCG learning algorithm needs much less training iterations than the conventional BPTT learning algorithm for the CB513 dataset. The HR-SCG algorithm is executing approximately 400 gradient vector calculations in 200 epochs against the BPTT which needs 1600 gradient calculations in 1600 epochs. The execution time of each gradient vector calculation in both algorithms is comparable so it is obvious that our HR-SCG algorithm is approximately 4 times faster than the BPTT algorithm. This is the most significant advantage of our methodology against training the BRNN with a conventional BPTT learning algorithm. Furthermore, *Figure 31(b)* is showing the training error of multiple executions of the HR-SCG learning algorithm for the CB513 dataset. As it can be observed, the algorithm is converging to the best available minimum after approximately 400 gradient vector calculations which is achieved after the algorithm is set 3 times in restart mode. Moreover, *Figure 31(c)* shows the training error before several restart modes of the algorithm further to the 400 gradient vector calculations. In this case, the algorithm is converging to worst solutions. Consequently, the training early stopping technique has been used and empirically set to the constant of 400 gradient vector calculations.

The need of less executions of the algorithm to converge is very important if we take into account the latest developments in the field which demand very big datasets and

network architectures, which consequently increase exponentially the amount of training time. In addition, many of these methods are combined in ensembles, as in Baldi et al. [1999], where the training amount of time is even more increased. The number of iterations needed to train BRNNs with HR-SCG linearly increases with the number of ensembles with much smaller slope than the BRNNs trained with the BPTT algorithm. This is extremely important if we take into consideration that an increasing size of ensembles is used to improve the results of ML methods [Zhou et al., 2002; Granitto et al., 2005a; Zhou et al., 2010].

Moreover, we have replicated the work of Baldi et al. [1999] (BaBRNN) and we have trained it on the PICES dataset. The purpose of this exercise was the direct comparison of our results to the BaBRNN method. The main difference between the BaBRNN and our BRNN model (both trained with the BPTT method) is the way that the sequences are fed to the network. In contrast to our window based method, BaBRNN is using the whole sequence to predict the SS state of a sequence at a specific time-step. More specifically, as described in the work of Baldi et al. [1999], we have built an ensemble of 6 BaBRNNs ($n=12$, $h_f b=9$, $N_u=11$, $s_i n=3$ and $s_o ut=3$). The 6-fold cross validation results on the PICES dataset has achieved the accuracy of 76.8%. Although BaBRNN is fed with the whole sequence at each time-step and has achieved an accuracy of approximately 76%, the results are comparable to our window based BRNN method. In Baldi et al. [1999], the vanishing gradient problem is responsible for the lack of capturing long range dependencies in BaBRNN model. Consequently, shorter input sequences in a BRNN have produced comparable results to the BaBRNN method. The comparable results can

be justified because both methods are using the same simplified shortcut connections between the forward and backward states to capture long-range information.

4.2.4 Conclusions

In this work, we present a second order method for training BRNNs for the PSSP problem where the SCG is applied for first time on these models. More specifically, we present the development and implementation of the HR-SCG learning algorithm for BRNN architectures. In contrast to the conventional GD learning algorithm, the HR-SCG exploits both gradient and curvature information for fast convergence. Through this paper, the formulas of SCG learning algorithm have been developed in detail and modified for the BRNN architecture. Furthermore, the HR-SCG methodology has also been enriched with a version of GD and Dropout algorithms for faster, better and more accurate results. Finally, ensemble methods and filtering techniques have been used to produce the final results. To the best of our knowledge, the SCG algorithm has not previously been used for training of a BRNN architecture.

The efficiency and effectiveness of our model has been tested on the PSSP problem. It has achieved 77.4% per residue accuracy on the CB513 dataset and 77.6% per residue accuracy on the PISCES dataset which compare well with other works based on BRNNs. Moreover, the SOV values for the CB513 dataset is 71.96 and for the PISCES dataset is 72.15. Finally, as it has already been mentioned, the HR-SCG outperforms the GD learning algorithm for BRNNs in terms of convergence time, needing approximately 75% less convergence time on the CB513 and PISCES datasets. Hence, the algorithm is efficient

for training the complex BRNN architectures, especially when there are big datasets or even for fast training ensembles of BRNNs.

The final results on the PSSP problems have demonstrated that a BRNN trained with our version of the SCG learning algorithm can capture patterns and make predictions on complicated sequences where the information in both upstream and downstream direction is important. Furthermore, the SCG learning algorithm needs much less training iterations and parameters to tune up than the conventional BPTT learning algorithm. This is very important if we take into account the latest developments in the field which demand very big datasets and network architectures, which consequently increase exponentially the training amount of time. In addition, many of these methods are used in ensemble methods where the training amount of time is even more increased.

The accuracy of 100% will never be achieved for the PSSP problem because of the presence of disordered regions, the ambiguities inherent in the definitions of secondary structure, the errors and uncertainties contained in databases and the role of the solvent and other molecules [Magnan and Baldi, 2014]. Nevertheless, the improvement and systematic combination of sequence profiles, machine learning methods and sequence-based structural similarity methods seem to be the best strategy to improve the results related to the PSSP problem. A contribution on any of these three categories, combined with other data preprocessing and algorithmic methods may play a catalytic role on the general improvement of the PSSP problem results.

4.3 Training BRNN with the HFO learning algorithm

Further to our work on training BRNNs with the HR-SCG, we have chosen to do the same work but training our BRNN architecture with the HFO learning algorithm which has been reported as one of the most powerful second order learning algorithms [Martens, 2010; Martens and Sutskever, 2011] for ANNs but has never been used for the BRNN architecture. In contrast to our work in HR-SCG, we have chosen a direct implementation of the original form of HFO to train our models (*Section 2.2.3*). More specifically, we have implemented the ML architecture which is described in *Figure 3*, but this time we have chosen as a filtering technique the SVM models as reported in the work of Kountouris et al. [2012]. The method which has been used for validation purposes was 10-fold cross-validation on the CB513 datasets. As it can be seen on *Table 12*, the method has achieved an accuracy of 76.52% Q3 and 0.7161 SOV. The results are lower than the results reported in *Table 10* for the HR-SCG learning algorithm for the CB513 dataset by 0.89% in terms of Q3 and 0.0035 in terms of SOV. The complexity of the model could not be handled by the original form of the HFO learning algorithm.

4.4 Chapter Contribution

A list of contributions resulted from this chapter is presented below:

1. The SCG, a second order method, is applied for the first time for training BRNNs for the PSSP problem.

<i>Fold</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	SOV
Fold0	77.26	79.52	69.92	79.12	0.6982
Fold1	76.12	74.02	68.01	79.02	0.7076
Fold2	76.91	75.02	69.51	78.11	0.7142
Fold3	77.01	79.23	69.12	76.72	0.7131
Fold4	76.12	76.82	69.92	77.13	0.7314
Fold5	75.94	78.91	68.11	75.92	0.7075
Fold6	77.41	79.33	68.54	78.81	0.7231
Fold7	76.22	77.61	71.94	76.03	0.7381
Fold8	75.35	76.51	68.25	77.11	0.7061
Fold9	76.82	79.14	70.12	75.15	0.7212
Average	76.52	77.61	69.34	77.31	0.7161

Table 12: **Results of an ensemble of six BRNNs combined with an SVM for filtering purposes:** 10-fold cross-validation approach on the CB513 dataset.

2. The introduction of a hybrid learning algorithm where the GD method is used to optimize the weights of a BRNN model updated with the SCG learning algorithm.
3. The introduction of an Adaptive Step Size Scaling Parameter which combined with the SCG learning algorithm can train a simple BRNN in 75% less training time.
4. The HFO, a second order method, is applied for the first time for training BRNNs for the PSSP problem.

Chapter 5

Efficient and effective methods for the PSSP problem

5.1 Introduction

As it has already been shown in *Chapter 2*, the latest developments and results in the NNs field have demonstrated huge potential in several data-related problems. Nevertheless, many of these NN models, their variations and specific learning algorithms have never been used for the PSSP problem. More specifically, there is an open field to be investigated related to the PSSP problem and CW-RNNs (*Section 2.1.4*), CNNs (*Section 2.1.5*) methods and HFO algorithm (*Section 2.2.3*). Although these techniques have their advantages and disadvantages compared between each other, each one has specific characteristics related to accuracy, execution time, sorting with short and long term dependencies and quality of results. Hence, we present results on an image-like input

representation method for the PSSP problem which is used by Convolutional Neural Networks, results of the same problem on Clockwork Recurrent Neural Networks, results on a novel Bidirectional Echo State Network architecture and results on a novel method of simple Feed-forward Neural Networks trained with the HFO algorithm.

5.2 CNNs in Combination with SVMs

5.2.1 Introduction

As we already discussed in this thesis, trying to extract features from complex sequential data for classification and prediction problems is an extremely difficult task. Deep Machine Learning techniques, such as CNNs, have been exclusively designed to face this class of problems. Furthermore, SVMs are a powerful technique for general classification problems, regression, and outlier detection.

One challenging task for ML techniques is to make predictions on sequential data which encode high complexity of interdependencies and correlations, such as the PSSP problem. A ML model designed for such data has to be in position to extract relevant features, and at the same time reveal any long/short range interdependencies in the sequence of data given. The major key point that needs to be considered when trying to solve the PSSP problem is the complex sequence correlations and interactions between the amino acid residues of a protein molecule. In order to maximize the prediction accuracy of a proposed NN technique for a specific amino acid in a protein molecule, the adjacent amino acids have to be considered by the proposed NN architecture.

We have developed and implemented a hybrid machine learning method based on the application of CNNs in combination with SVMs, for complex sequential data classification and prediction [Dionysiou et al., 2018]. The main architecture of our methods is based on *Figure 3*, but we have replaced the BRNNs with CNNs. Furthermore, we have designed and implemented a novel two dimensional (2D) input representation method for sequential data and then we have tested it on the PSSP problem for 3-state secondary structure (SS) prediction.

5.2.2 Data Representation

CNNs are capable of analyzing image-like inputs. The major obstacle on trying to solve a complex sequential data classification problem with CNNs is the representation of the data, in such a way that the network is able not only to understand the shape of the input volume, but also to track the complex sequence correlations among the input volume. Transforming the sequential data shape so as to make it look like an image, allows CNNs to capture the complex sequence-structure relationship, including to model the SS interactions among adjacent or distant amino acid residues in the PSSP problem.

Along these lines, we reorganised the input data shape so that the vectors of each sample in the sequential data are placed one under another, and in such a way create an image-like input that will be effectively read correctly and understood by the CNN. More specifically, we have created images based on the MSA [Wallace et al., 2005] encoding of each protein sequence. The size of each image was specified by:

1. **Image Width:** A sliding window size W on the PS of each protein.

2. **Image Height:** The length of each MSA profile vectors of each amino acid on the PS, which is a constant value that equals to 20 [Wallace et al., 2005].

At each time step, the sliding window is moving on a protein's PS one step to the right to create a new image representation for that specific segment of size W . As it can be seen in *Figure 32*, at each position of the image matrix, we have a normalized value out of 100. This value indicates the appearance of a specific amino acid at a specific location of the sequence based on its MSA profile. The MSA profile vector of the amino acid for which we are looking at its SS is located vertically at the position $(W/2) - 1$ along the image width. This is the center of the image matrix. Consequently, at the rest of the image locations, we have the MSA profiles of all the neighboring amino acids which are located at the PS before and after the centered amino acid. This is the local information needed to infer the SS of the centered amino acid. In particular, for PSSP we have created a new input image by placing MSA profile vectors of each amino acid one under another to construct a 2D representation of the MSA profiles of a certain number of neighbouring amino acid residues (*Figure 32*). The basic idea behind this sequence to image transformation is that the CNNs have been designed to extract features from images. With this 2D input representation the CNN filters can scan the image and extract local patterns and local dependencies from the neighboring amino acids to infer the SS of the centered amino acid.

By sliding the kernel over the newly constructed input volume, CNNs are able to perform feature extraction for each record data, but also consider neighboring correlations and interactions, if any exist. Note that unlike other techniques, the attention given to any

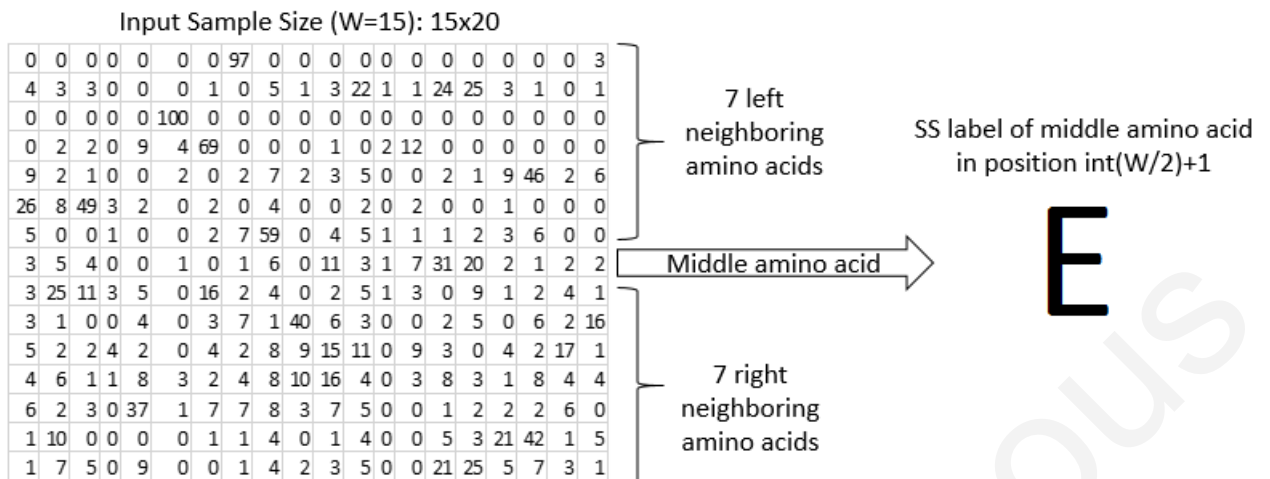


Figure 32: Example of Data Representation Method: An example of data representation of an input sample using a window size of 15 amino acids. Each line represents the MSA profile vector for the specific amino acid. The SS label for the example input sample showed in this figure, is the SS label for the middle amino acid.

neighboring record correlations is equally weighted across all the input volume, for each sample given. This lets the CNN discover and capture any short, mid- and long range correlations among the input records and consider them all equally in terms of the output volume created. One of the major contributions of this work is this innovative input data representation, especially designed for the complex sequential data of the PSSP problem.

5.2.3 Support Vector Machines (SVMs)

The main idea behind SVMs is that the input vectors are non-linearly mapped to a higher dimensional feature space using an appropriate kernel function with the hope that a linearly inseparable problem in the input space becomes linearly separable in the new feature space, i.e., a linear decision surface can be constructed [Vapnik, 1999]. An important advantage of SVMs is that the search for the decision surface that maximizes the margin

among the target class instances ensures high generalization ability of the learning machine [Meyer and Wien, 2001]. Their robust performance with respect to sparse and noisy data makes them a good choice in a number of applications from text categorization to protein function prediction [Furey et al., 2000]. Moreover, SVMs were shown to be the best technique for filtering on the PSSP problem [Kountouris et al., 2012]. Given this, we decided to test the filtering capabilities of SVMs on the CNNs' SS prediction results, to see whether the accuracy is improved, and correct the predicted SS of a protein molecule gathered from an ensemble of CNNs.

5.2.4 Results and Discussion

5.2.4.1 Optimising the Parameters

The CNN implementation using the innovative input data representation described in *Section 5.2.2* has been used and tested on the PSSP problem. To train the CNN, we have used the already mentioned CB513 dataset. More specifically, the model's input was a combination of a certain number of neighboring amino acids MSA profile record vectors, one under another, forming a 2D array. The target output label was the SS class for the middle point amino acid that had been examined.

A single CNN has been trained each time. We have decided to track the optimal hyperparameter values using a specific fold after dividing CB513 dataset into ten (10) folds. The main reason for optimizing the hyperparameters on a specific fold is the small size of CB513 dataset. Accuracy results using different hyperparameter values on the other

folds are not expected to vary considerably. During this phase, multiple experiments were performed in order to tune up our model and finally achieve the highest results using the CNN. These were Q3 of 75.155% and SOV of 0.713. CNNs with different numbers of CLs, PLs, kernel sizes, strides, number of parallel filters in each CL, and GD optimization algorithms (*Figure 33*) have been tested for optimising the parameter values. The optimization algorithms used are: Gradient Descent (GD), Gradient Descent with momentum (GD with momentum), Adaptive Gradient Algorithm (AdaGrad) [Duchi et al., 2011], RMSprop [Tieleman and Hinton, 2012], AdaDelta [Zeiler, 2012], Adaptive Moment Estimation (Adam) [Kingma and Ba, 2015]. The two most critical hyperparameters that showed a big impact on the results are: (a) the optimization method used and (b) the number of neighboring amino acids to be considered in each sample (window size). More specifically, the parameter W is the number of total amino acids to be considered by the CNN when trying to predict the SS of the floor($W/2$) + 1 amino acid. Then, according to the W parameter we reconstruct the input sample so as to become a 2D array with shape $W \times 20$. The results are shown in *Figure 34*. Unlike Wang et al. [2016] method, where they use 42 input features for each residue in an one dimensional input vector format, we use $20 \times W$ (20 input features for each amino acid \times window size) input features for each residue in a two dimensional input vector format where each line represents the MSA profile of an amino acid at any specific position. Generally speaking Wang's et al. Wang et al. [2016] 42 input features used include our 20 input features (MSA profile for each amino acid) plus extra 22 input features for each amino acid. In this way, our method

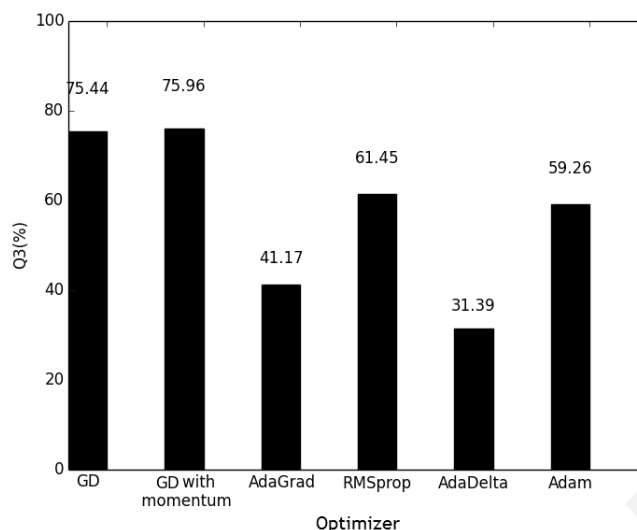


Figure 33: **Optimizers:** CNNs Q3 accuracy results using different Gradient Descent (GD) optimization algorithms.

reduces the dimensionality of the problem without losing too much important information. Moving forward, we had to tune up the parameters that determine the network's architecture.

To get a general idea about the CNN performance we have trained it using the CB513 dataset. After tuning up the network architecture, the following optimal CNN parameter values resulted: (a) Number of convolutional layers: 3, (b) Number of Pooling Layers: 0, (c) Kernel/Filter size: 2×2 , (d) Stride: 1, (e) Number of Parallel Filters per Layer: 5, (f) Neurons Activation Function: Leaky ReLU, and (g) Optimization method: Gradient Descent with momentum=0.85. The number of neighboring amino acids (W) that leads to some among the highest Q3 results and at the same time limiting the complexity of information been used (i.e., minimizing the window) was 15. Moreover, no significant change on Q3 accuracy results was noticed using larger window (W) sizes (*Figure 34*). Based on the results, we realized that (i) smaller W values do not provide enough information

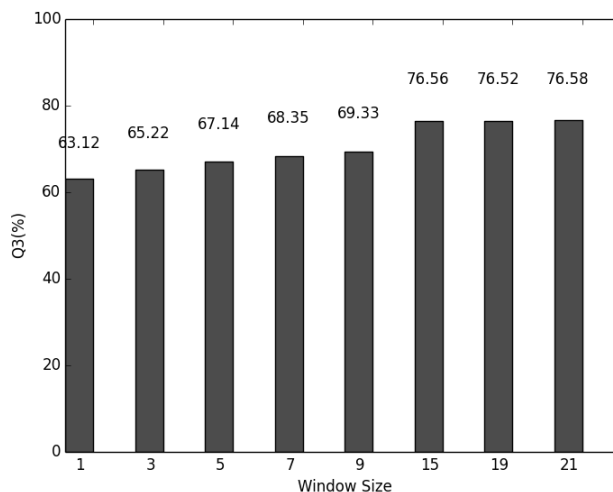


Figure 34: **Window Size:** CNNs Q3 accuracy results with different window (W) sizes.

to the network regarding the adjacent interactions between amino acids, and (ii) larger W values contain way too much (unnecessary in some way) information for the network to be handled and decoded properly.

We did not use pooling layers for our CNN architecture due to the fact that subsampling the features gathered from CNN is not relevant in the PSSP problem. Getting only the maximum value of a spatial domain does not work in PSSP as every value extracted from CLs may represent interactions of amino acids in a certain region. These are the most important factors that lead to low Q3 and SOV results using PLs.

5.2.4.2 10-fold Cross-Validation on CB513

In order to to validate the robustness of the model as well as to prove its efficiency to the exposure of various training and testing data, we had to complete the evaluation of the PSSP problem on the CB513 dataset, using a 10-fold cross-validation test. All the

<i>Method</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	SOV	SOV_H	SOV_E	SOV_L
CNN	75.155	69.474	67.339	84.566	0.713	0.696	0.669	0.734
CNN Ensembles	78.914	72.748	68.854	85.385	0.744	0.738	0.722	0.737
CNN Ens. + ER Filt.	78.692	70.147	66.921	87.053	0.756	0.669	0.713	0.731
CNN Ens. + SVM Filt.	80.40	80.911	70.578	85.165	0.736	0.724	0.716	0.743

Table 13: Summary of the Results for All Methods.

experiments made are with the optimal parameters of the model as described in *Section*

5.2.4.1. As shown in *Table 13*, the Q_3 and SOV accuracy results of CNN with 10-fold cross-validation are 75.15% and 0.713 respectively.

5.2.4.3 Ensembles and External Rules Filtering

After tracking the optimal parameters for the CNN, we have performed six (6) experiments for each fold. Then, in an attempt to maximize the quality of the results gathered as well as to increase the Q_3 and SOV accuracy, we proceeded with using the winner-take-all ensembles technique [Granitto et al., 2005b; Fukai and Tanaka, 1997] on every single fold separately. This technique obtains the predictions of a number of same ML model experiments, and applies the winner takes all method on each amino acid residue SS class predicted. The dramatically improved results are shown in *Table 13*.

Filtering the SS prediction using external empirical rules is usually the last step made, as a final attempt to improve the quality of the results. This is accomplished by removing conformations that are physicochemically unlikely to happen [Salamov and Solovyev, 1995]. Applying the external rules filtering on the CNN's SS prediction, interestingly, does not improve the Q_3 score, but it improves the SOV. The results are shown in *Table 13*.

5.2.4.4 Filtering using Support Vector Machines (SVMs)

CNNs showed very good results on the PSSP (*Figure 33, 34* and *Table 13*). Nevertheless, as mentioned above, we tried to use SVMs to perform the filtering task. More specifically, after gathering the predictions from the CNN we have trained a SVM using a window of SS states predicted by the CNN. After performing several experiments using different kernels, misclassification penalty parameters (C) [Cortes and Vapnik, 1995b], Gamma values (G) [Cortes and Vapnik, 1995b] and window sizes (WIN), we have decided for the optimal SVM parameters that lead to the highest Q3 and SOV accuracy on the PSSP problem and which are: (a) Kernel: Radial Basis Function, (b) $C = 1$, (c) $G = 0.001$ and (d) $WIN = 7$. The results are shown in *Table 14* and *15*.

5.2.5 Summary of the Results

The results shown in *Table 14* summarize the Q3 accuracy and SOV results gathered, with all the methods discussed in this work, using 10-fold cross-validation. It is shown that the CNN can achieve relatively high Q3 and SOV results (75.155% and 0.713 respectively) by its own. Nevertheless, the CNN using ensembles improved the Q3 accuracy results by approximately 3% and SOV score by 0.031. Moving on, filtering the results using External Rules mentioned above, decreases the overall Q3 accuracy results to 78.692%, but dramatically increases the SOV score from 0.744 to 0.756. This was expected as filtering with External Rules has previously been reported to improve SOV scores, but at the same time decrease the overall Q3 accuracy [Baldi et al., 1999]. Finally,

<i>Method</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	SOV	SOV_H	SOV_E	SOV_L
CNN	75.155	69.474	67.339	84.566	0.713	0.696	0.669	0.734
CNN Ensembles	78.914	72.748	68.854	85.385	0.744	0.738	0.722	0.737
CNN Ens. + ER Filt.	78.692	70.147	66.921	87.053	0.756	0.669	0.713	0.731
CNN Ens. + SVM Filt.	80.40	80.911	70.578	85.165	0.736	0.724	0.716	0.743

Table 14: Summary of the Results for All Methods.

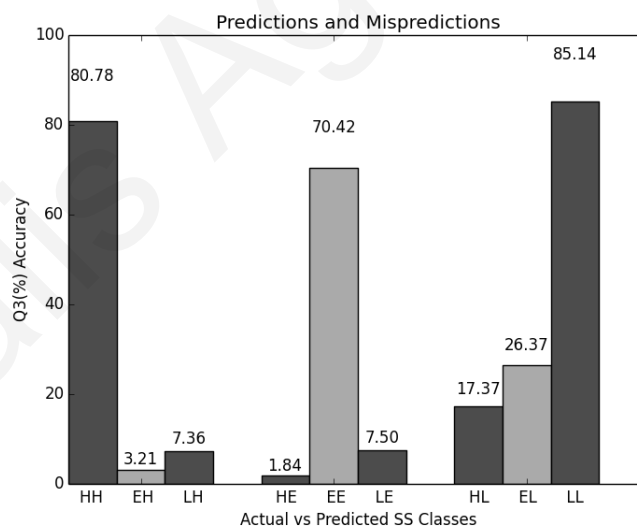
using the combination of CNN ensembles and SVM as a filtering technique, achieves the highest Q3 accuracy results (80.40%). The Q3 values for different folds vary from 78.96% to 83.91% and the SOV from 0.71 to 0.78 *Table 15*. This indicates that the results for different folds are of comparable quality. Moreover, the accuracies for the three classes, H, E, L, are calculated separately (see Q_H , Q_E , Q_L and SOV_H , SOV_E , SOV_L in *Table 15*) for getting deeper insight on the quality of the classifier, and mispredictions are quantified in a confusion matrix, graphically represented in *Figure ??*. As we can see from *Table 15*, Q3 accuracy results gathered using CNN Ensembles and SVM filtering are just over **80%**, which is considered to be a high enough percentage when it comes to PSSP, and which also makes this combination of NN techniques a good option when it comes to complex sequential data classification and prediction problems. **Heffernan et al. [2017]** method achieves 84.16% Q3 accuracy using Bidirectional Recurrent Neural Networks without using a window, but these results are not directly comparable with our results, as they make use of a much larger dataset that contains 5789 proteins, compared to CB513 which contains 513 proteins.

As a conclusion to all the results presented in this section, we can see that the CNNs can effectively detect and extract features from complex sequential data, by utilizing our proposed image-like data representation method used to train the CNNs for the PSSP problem. This is due to the fact that our CNN architecture was exclusively designed

<i>Fold</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	SOV	SOV_H	SOV_E	SOV_L
0	79.69	79.77	70.05	84.75	0.74	0.73	0.71	0.75
1	79.74	78.69	68.06	86.77	0.73	0.73	0.71	0.74
2	78.96	78.64	68.27	84.94	0.72	0.71	0.71	0.73
3	79.55	79.09	67.89	86.12	0.71	0.72	0.70	0.73
4	79.26	78.55	70.00	84.79	0.73	0.72	0.73	0.72
5	79.70	80.27	70.18	84.31	0.73	0.71	0.72	0.73
6	79.64	79.85	68.87	85.26	0.73	0.73	0.71	0.74
7	83.70	87.68	76.86	83.91	0.76	0.73	0.71	0.77
8	83.91	87.53	76.33	84.62	0.78	0.75	0.74	0.79
9	79.85	79.04	69.27	86.18	0.73	0.71	0.72	0.73
Avg.	80.40	80.91	70.57	85.16	0.736	0.724	0.716	0.743

Table 15: **CNN Ensembles and SVM Filtering:** Q3 and SOV Results for each Fold.

	Q3	SOV
Sample Standard Deviation (s)	1.8140	0.0141
Variance (Sample Standard) (s^2)	3.2906	0.0002
Mean (Average)	80.4	0.736
Standard Error of the Mean ($SE_{\bar{x}}$)	0.5736	0.0044

Table 16: **CNN Ensembles and SVM Filtering:** Statistical AnalysisFigure 35: **Confusion Matrix:** Predictions and mispredictions of the secondary structure classes H, E and C/L after applying ensembles on each fold using CB513 dataset. Q3 accuracy scores are shown for each class.

to face such problems. In addition, SVMs seem to be a good technique to be used for filtering the CNN output. The combination though, of these two ML algorithms seems to be a particularly good option for complex feature extraction and prediction on sequential data, as we take advantage of the benefits of both techniques. Finally, by observing the results from the confusion matrix of *Figure 35*, we can conclude that the combination of CNNs with SVMs filtering is a robust and high quality methodology and architecture, as it maximizes the correct predictions for each SS class. Results are expected to be improved by collecting more experiments for each fold, using larger datasets (e.g., PISCES) and deploying more sophisticated ensemble techniques.

5.3 Clockwork Recurrent Neural Networks

In this work we apply a recently proposed recurrent neural network (RNN) architecture, the Clockwork RNN (CW-RNN) [Koutnik et al., 2014] to the PSSP problem for the first time. The CW-RNN is capable of overcoming the complexity of the problem as one of its main advantages is its low execution time, due less parameters and computation of less operations at each time step.

One of the drawbacks of even the most advanced PSSP methods, relates to capturing existing long-distance dependencies, which are particularly important in β -stands interacting to form β -sheets. As we have already mentioned through out this thesis, among the first type of methods that demonstrated a significant improvement towards better handling these long-range dependencies are BRNNs. Unfortunately, even the BRNN approach results to models incapable of fully learning long-range dependencies, which is key to

```

X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X
0 0 0 0 0 0 87 13 0 0 0 0 0 0 0 0
0 0 0 0 0 0 2 0 0 0 0 0 0 17 72 6 3 0 0
. . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . .
4 11 1 23 22 0 9 0 0 0 29 0 0 0 0 0 0 0 1
0 0 0 0 63 15 22 0 0 0 0 0 0 0 0 0 0 0 0
X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X
X X X X X X X X X X X X X X X X X

```

Figure 36: **Addition of dummy values:** The PSSM profiles is padded with dummy values before and after the sequence termini. $W_a = 11$; X: dummy value.

succeed in the PSSP problem. This incapability largely rises because of the vanishing gradient problem [Hochreiter and Schmidhuber, 1997]. This problem has been generally (i.e., not in the PSSP problem) addressed by modifying the architecture of the RNN and as such this is the case in the recently proposed CW-RNN [Koutnik et al., 2014]. The novelty of this RNN model is that the hidden layer is partitioned into modules, where each module is assigned a different clock period, corresponding to different levels of memory span. As a consequence, every module is activated and executes computations only at the time step which corresponds to it. These different modules attempt to provide a solution to the long-range dependency problem.

5.3.1 Methodology

5.3.1.1 Input Data

In this work the CB513 dataset has been used. Before using the MSA profiles, a much more comprehensible form of input data was created. All of the input proteins were added in a file with specific information about them. More specifically, each protein is represented in the file in three lines. There is the name of the protein and the residues which is referred to in the first line, the primary structure of the protein is written in the second line and in the third one its secondary structure. This file was used in order to match each protein with its MSA profile. It was also used for the construction of a data structure, which contained all the information provided by that file and added a line gap after each protein's information to be filled in later on with its predicted secondary structure.

The use of a sliding window W_a was required for the model to be able to have access to all the data available, both for training and validation. After loading the information of each MSA profile, a matrix whose size is equal to $(W_a - 1)/2$, with each of its rows containing dummy values, is being added before and after it (*Figure 36*). By using the sliding window technique in this way provides us with the capability to predict each protein from its starting point and allows us to predict the middle amino acid residue of each sliding window. As it is well known, an amino acid residue is directly influenced by its surrounding amino acid residues, which determine the category of its secondary structure, and this information is captured by using the sliding window.

Next, after processing all of the above data, the final data structure, which consists of all the sliding windows of all the proteins used for training is being constructed at its final form. In parallel, the final form of the target values, where each value corresponds to each sliding window is also being created. The above method is used to create the final form of the validation data too.

5.3.1.2 Architecture and implementation

The implementation used was exactly as proposed by Koutnik et al. [2014]. Several general changes had to be made for this specific implementation to be adjusted to the PSSP problem. The input data had to be in a specific form for the model to accept it and train properly. The size of the network's input layer is equal to $W_a * 20$. Each input - a row of W_a - represents an amino acid residue and the twenty numbers correspond to the frequencies of different residue types aligned with the original sequence. The network consists of three numerical outputs, which are mapped to the three broad secondary structure classes E, H and C. The decision of the class of the predicted amino acid residue is taken following a simple Winner-Takes-All (WTA) method. Additionally, the model was trained on a number of fixed size batches. The non-dynamic size of the batch, constrained us to the choice of the size of training and validation data. In order to overcome this constraint we decided to pad the validation data to the minimum number possible, which would allow all of training data to be used. After each training batch, the training loss is calculated and from the training predictions, the training accuracy for this specific batch is also calculated. When an epoch ends, the model is tested on validation data and the

validation loss and accuracy are computed similarly. Furthermore, because the training data is divided into batches of fixed size, some information about a protein may be cut in half at the beginning or the end of the batches. This requires the shuffling of the training input to prevent the model on learning specific artifactual patterns about the data. At the end of all epochs, two output files are created, using the predictions of training and validation data respectively. These files have the same structure as the input files except that after the information of each protein a line with the predicted secondary structure of that protein is added. As a result, each protein requires four lines of information in the output files, which is its name, its primary and secondary structure and its predicted secondary structure. These are required for ensembles and filtering (*Section 3.1.3 and 3.1.4*).

We identified an optimal architecture and relevant hyperparameters by trial and error. More specifically, the set of clock periods which is used is symmetric and a part of it was influenced by Fibonacci series, as proposed in the initial publication [Koutnik et al., 2014]. The hidden layer of the model was partitioned in 28 modules and each one of them was assigned a clock period. The weights were initialized with particularly small values with a standard deviation of 0.01 and the biases with the value of 0. The activation function for the hidden layer was the hyperbolic tangent and for the output layer the sigmoid. They are both non binary activators, their output is always bounded within a specific range, so the activations will not blow up and the sigmoid can make clear distinctions on predictions as it tends to bring the activations on either side of its boundaries. For the training of the model and the computation of the gradients the Adam optimizer was selected among the available options, as it is computationally efficient and is well

suites for this specific problem with a large dataset. Adam [Kingma and Ba, 2015] was employed using the default values for beta1 (0.9) and beta2 (0.99) and a learning rate of 0.01. Gradient clipping, which is the clipping of the gradients between two numbers to prevent them from becoming too large, was also used to alleviate the gradient exploding problem.

After the optimization of the parameters each fold of the dataset has been executed 8 times. The application of ensembles was then carried out on each one of these folds separately for a better outcome and their Q3 and SOV accuracy scores were computed. Finally, external empirical rules were applied to the ensemble outputs. These specific rules try to correct and smooth the predictions of the model, by altering those parts of the predicted secondary structure, which are physicochemically improbable. The empirical technique used was SS-filt [Salamov and Solovyev, 1995], which consists of the filtering rules that replace single-helical and single-strand residues with loop and all strands of length two surrounded with helices are replaced by helices [Kountouris et al., 2012].

As it is already mentioned, one of the main features of the CW-RNN architecture is its fast execution speed. It is already tested in specific tasks, where it overpowers RNN in this sector [Koutnik et al., 2014]. This is because it is using less parameters and carries less operations at each time step due to the smaller number of connections between the neurons of its hidden layer. In the current paper the CW-RNN architecture has demonstrated its speed capabilities at the PSSP problem as well.

As the number of neurons in the hidden layer grows, more complexity is added to the model. For the runs used to determine the optimal number of neurons for the CW-RNN

we recorded the execution times, to identify whether there is a clear relationship between the execution time and the number of neurons in the hidden layer. As shown in *Figure 43*, when the number of neurons increases, the execution time increases in a linear manner. Given the observation that the results of the CW-RNN for the problem at hand did not lead to better prediction accuracy, gives us the opportunity to stick to very simple models which can get trained very fast (only a few minutes in our case).

5.3.2 Results and Discussion

We conducted several experiments in order to examine how each CW-RNN hyperparameter affects the results and decide an optimal set of parameters for achieving the best result possible. The training and validation accuracies were estimated using the per-residue accuracy Q3, and the Segment Overlap (SOV) score [Zemla et al., 1999]. After a satisfying number of experiments, we have applied ensembles and filtering on them. The main reason we were able to execute numerous experiments was the model's astonishingly fast time of execution.

5.3.2.1 Examination of the parameters

Clock periods

The slower modules of a CW-RNN act as a mechanism that fills up the gap between information and gradients which are a large number of time steps apart. The faster modules make short term decisions, as they have shorter periods and are executed more often than

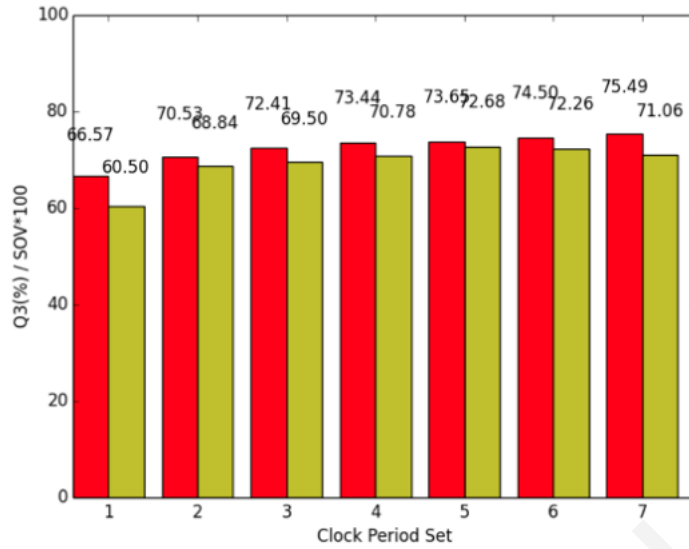


Figure 37: Q3 and SOV accuracy scores for each clock period set shown in Table 17. The red bar corresponds to Q3 and the green bar to SOV.

No.	Clock Period Sets
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

Table 17: The sets of clock periods which were used to determine the optimal one.

the slower ones. The information that the faster modules process are seen and also processed by the slower ones, when they are active, in order to make the correct correlations between this information and the information that was already stored in their states since the last time they were active. As a result, the clock periods are effectively the novelty of the model and are considered to be the key in achieving good results, so we have started experimenting on these first.

Initially, we have tested the default clock periods in the CW-RNN implementation (Set 1 in *Table 17*) resulting in relatively low Q3 and SOV values (*Figure 37*). Using the same exact periods in reverse order (Set 2 in *Table 17*) increased the Q3 accuracy by almost 4% and the SOV by more than 8 points. Next, we combined a part of each of the above sets, hence the Clock Period Set 3, which resulted in an even better result than the last two sets. The thought of this combination resulted due to the symmetry of the sliding window with respect to the middle amino acid for which the prediction is performed, so we used a symmetric set of periods. The 5th set was also tested using the same rationale as the 3rd set and the accuracies slightly increased. The 6th set of periods was tried in order to examine what would happen if there was no symmetry and faster modules would be added to process more information at each time step. This had as a consequence the marginal rise of the Q3 accuracy, but the moderate reduction of the SOV score.

Having in mind that there may be a connection between the periods, we thought of using a series of periods with a slightly better connection between them. Clock Period Set 4 was inspired by a small part of the Fibonacci series, which had as a result a quite good accuracy. Finally, we combined the knowledge of the above experiments, which was the symmetry of the clock periods, a part of the Fibonacci series and the addition of more faster periods for the creation of 7th set. This set has caused the best result at the Q3 score and we used it for all the next experiments. However, the SOV score moderately decreased comparing it with the results from the 5th and 6th set.

We should note here, that the different clock period sets mentioned above correspond to CW-RNNs with different numbers of modules (7, 7, 9, 11, 13, 18, 28 respectively).

The differences displayed in *Figure 37* relate to the effect of the different period sets and not to the size of the network. Clock period sets 1 and 2 can show that even having the same number of modules, the result depends on the right choice of the clock periods.

Optimizers

Three different optimizers available in this CW-RNN implementation were tested for their efficiency and performance. Those were Adam, RMSprop and AdaGrad optimizers. The rest of the parameters of the network were unchanged while testing the optimizers. More specifically, the activation functions for hidden and output layers were the hyperbolic tangent function and the sigmoid function respectively and the number of neurons in the hidden layer was 28, one neuron for each module.

The Adam optimizer combines the advantages of both AdaGrad and RMSprop optimizers. It enhances the efficiency of the optimizer on problems with sparse gradients by maintaining a per-parameter learning rate and it is reported to be efficient on online and non-stationary problems. Nevertheless, we examined the other two optimizers as well. AdaGrad optimizer resulted in a low Q3 score and almost 4 times lower SOV score than Q3, whereas the difference between these scores in other experiments is much smaller. RMSprop optimizer was more suitable for this problem as the results were really close to the ones obtained by the Adam optimizer which achieves the best results in this particular dataset (*Figure 38*).

Activation functions

As it is already explained, the activation functions which were initially used were the

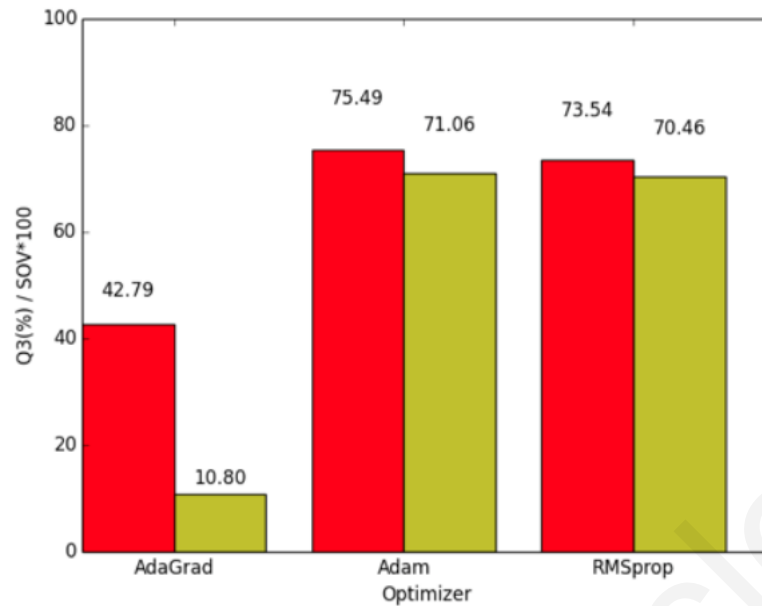


Figure 38: **Q3 and SOV accuracy scores** for each optimizer. The red bar corresponds to Q3 and the green bar to SOV.

hyperbolic tangent for the hidden layer and the sigmoid for the output layer. The computations on the weights of the hidden layer are crucial for the final result, so we decided to change only the activation function of the hidden layer at each experiment. The rest of the parameters of the network were unchanged. In more detail, the clock periods which were used were the optimal ones, using the Adam optimizer and the number of neurons in the hidden layer was 28, one neuron for each module. Three more experiments were executed, after our initial one, in order to decide the proper activation function for the hidden layer. As it can be seen in the bar charts of *Figure 39*, using the ReLU, the Q3 score was the lowest, but the SOV score was the highest, compared to the others. Despite the fact that ReLU resulted in the highest SOV score, we decided to choose the hyperbolic tangent. Its SOV is just a bit smaller than ReLU's, but its Q3 is quite bigger. Next, by using the sigmoid in both layers (already using it as the activation function of the output

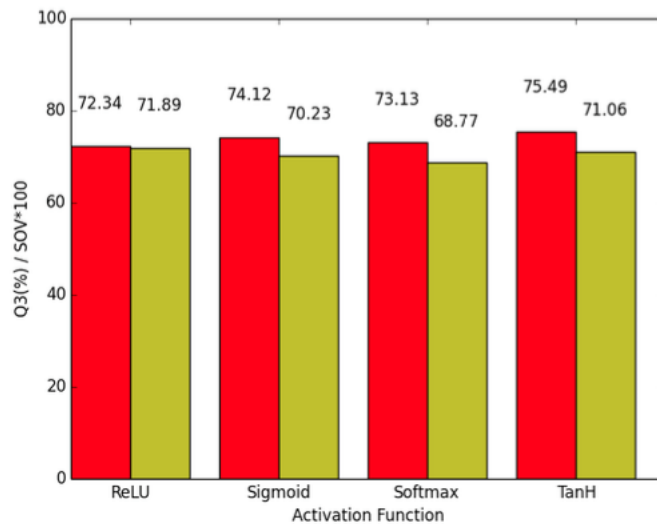


Figure 39: **Q3 and SOV accuracy scores** for each activation function. The red bar corresponds to Q3 and the green bar to SOV.

layer), the results which were achieved were close to the best ones. In the last experiment we used the softmax in the hidden layer and we obtained the lowest SOV score, but the Q3 score was only almost 2% less than the best one.

Number of neurons

The first experiments were carried out by using the lowest number of neurons possible, as the execution time also depends on the size of the network. Thus, we had to keep the number of neurons as low as possible to achieve a really fast execution time. The lowest number of neurons we were able to use is bounded by the number of modules (which is equal to the number of clock periods of the model) and each module must have at least 1 neuron. After achieving a satisfying Q3 and SOV scores with only 28 neurons (i.e. 1 neuron per module), more neurons were added per module, in order to examine if the model would be able to process more information for a better result. As shown in *Figure 40*, by adding more neurons does not quite affect the Q3 and SOV scores. The

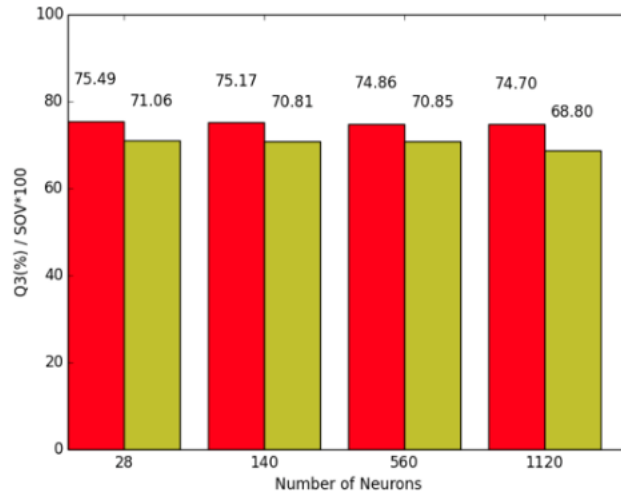


Figure 40: **Q3 and SOV accuracy scores** for different number of neurons in the hidden layer. The red bar corresponds to Q3 and the green bar to SOV.

enlargement of the network by adding more neurons, only adds more complexity to the model, which has as a consequence the increase of its execution time but with roughly the same scores being achieved.

5.3.2.2 10-fold cross-validation on CB513

The complete evaluation of the PSSP CB513 dataset results was performed using a 10-fold cross-validation test. This validation had to be done in order to validate the robustness of the model and to prove its efficiency to the exposure of various training and testing data. All the executions were completed with the optimum parameters of the model.

The results of the ensembles are shown in *Table 18*. Q3 values vary from 73.74% to 77.02% and SOV from 0.68 to 0.74. This indicates that the results for different folds are of comparable quality (see also *Table 19*). Moreover, the accuracies for the three classes,

<i>Fold</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	<i>SOV</i>	SOV_H	SOV_E	SOV_L
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
Avg.	75.83	69.92	62.44	84.26	0.73	0.74	0.67	0.72

Table 18: Q3 and SOV accuracy scores for each fold after the application of ensembles.

	Q3	SOV
Sample Standard Deviation (s)	1.05	0.02
Variance (Sample Standard) (s^2)	1.11	0.00
Mean (Average)	75.74	0.72
Standard Error of the Mean ($SE_{\bar{x}}$)	0.33	0.01

Table 19: Results after ensembles: statistical analysis

H, E, L, are calculated separately (see Q_H , Q_E , Q_L and SOV_H , SOV_E , SOV_L in [Table 18](#)) for getting deeper insight on the quality of the classifier, and mispredictions are quantified in a confusion matrix, graphically represented in [Figure 41](#). The results presented herein are comparable to those we have obtained using more complex BRNN models [[Kountouris et al., 2012](#)], however requiring much less time for training the individual models. Considering the above, the model is able to successfully train and be tested on the different order of sequences of data each fold provides.

The application of filtering on the folds resulted in the slight increase of the overall Q3 (ranging between folds from 73.69% to 77.23%) and the overall SOV (ranging from 0.69 to 0.75) (see [Table 20](#)). Even though most of the folds have a decreased Q3 accuracy,

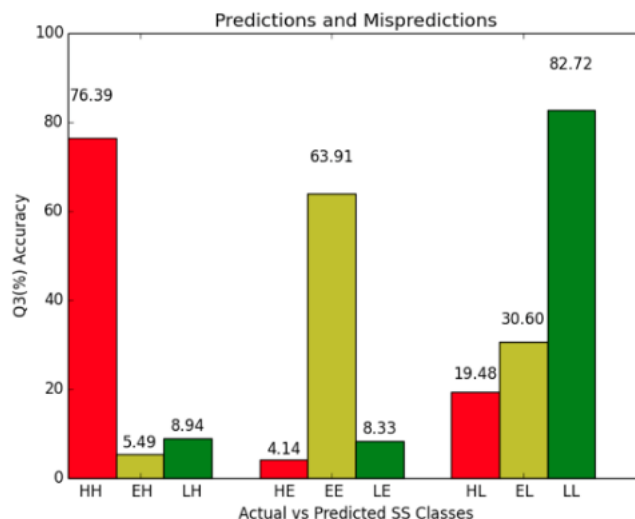


Figure 41: **Predictions and mispredictions** of the secondary structure classes H, E and L after applying ensembles on each fold. The red bar corresponds to Q3 and the green bar to SOV.

whereas only one of the folds has a decline in its SOV score, both those scores have marginally risen due the rest of the folds with increased scores. The standard deviation and variance of Q3 have slightly risen compared to the unfiltered predictions (compare [Table 20](#) to [Table 21](#)), whereas SOVs remain the same despite their marginal increase of their average. QH and QE are both decreased after filtering by approximately 1%, while there is an increase in QL ([Figure 42](#)).

As it is already mentioned, one of the main features of the CW-RNN architecture is its fast execution speed. It is already tested in specific tasks, where it overpowers RNN in this sector. This is because it is using less parameters and carries less operations at each time step due to the smaller number of connections between the neurons of its hidden layer. In the current paper the CW-RNN architecture has demonstrated its speed capabilities at the PSSP problem as well.

<i>Fold</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	<i>SOV</i>	SOV_H	SOV_E	SOV_L
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
Avg.	75.83	69.92	62.44	84.26	0.73	0.74	0.67	0.72

Table 20: Q_3 and SOV accuracy scores for each fold after applying external empirical rules on them.

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

Table 21: Results after external empirical rules: statistical analysis

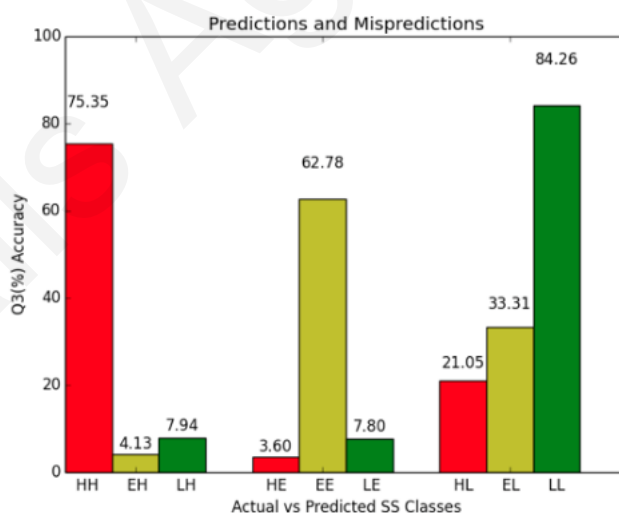


Figure 42: **Predictions and mispredictions** of the secondary structure classes H, E and L after applying external empirical rules on each fold. The red bar corresponds to Q_3 and the green bar to SOV.

<i>Fold</i>	<i>Q₃(%)</i>	<i>Q_H(%)</i>	<i>Q_E(%)</i>	<i>Q_L(%)</i>
0	77.55	79	67	82
1	76.35	79	62	82
2	76.9	79	62	81
3	77.36	78	63	83
4	76.17	76	60	83
5	74.54	75	59	81
6	76.6	74	63	84
7	76.14	73	64	83
8	75.7	74	65	82
9	77.1	78	62	83
Average	76.44	76.5	62.7	82.4

Table 22: Q3 and SOV accuracy scores for each fold after applying SVM filtering on them.

Further to the empirical rules, we have also used a SVM as a filtering technique. The results can be seen on *Table 22*. This filtering technique had improved the method's Q3 per residue accuracy by approximately 0.5%.

As the number of neurons in the hidden layer grows, more complexity is added to the model. For the runs used to determine the optimal number of neurons for the CW-RNN we recorded the execution times, to identify whether there is a clear relationship between the execution time and the number of neurons in the hidden layer. As shown in *Figure 43*, when the number of neurons increases, the execution time increases in a linear manner. Given the observation that the results of the CW-RNN for the problem at hand did not lead to better prediction accuracy, gives us the opportunity to stick to very simple models which can get trained very fast (only a few minutes in our case).

In general, regardless of the complexity of the problem, the model finishes its execution in a short period of time and definitely much faster compared to typically used architectures based on RNNs or BRNNs, with prediction quality in par with that of the

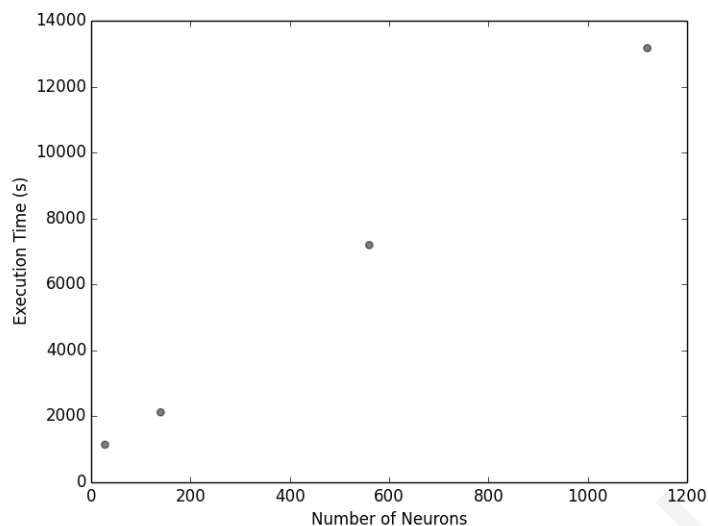


Figure 43: **Relationship between the execution time and the number of neurons** on the hidden layer of the CW-RNN.

much more complex recurrent architectures tested so far. Smaller batch sizes seem to support the model to finish its execution faster. This feature has also allowed us to systematically experiment on the selection of the hyperparameters and complete the 10-fold cross-validation really fast.

In addition, the training speed of CW-RNNs make them ideal for building ensemble classifiers of a large number of networks with different hyperparameters trained to learn different aspects of the same training data. Moreover, our work strongly indicates that CW-RNNs may be appropriate for successfully tackling other related problems in protein sequence analysis. This work has been presented in [Dimitriou et al. \[2018\]](#).

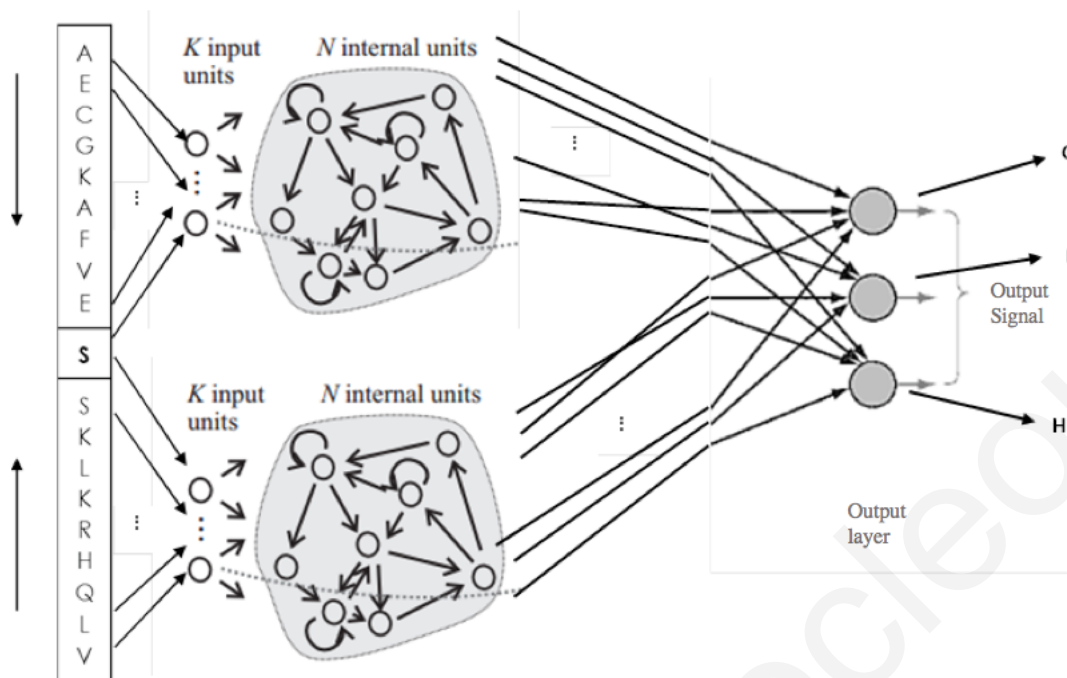


Figure 44: The architecture of a BESN as described in this study.

5.4 Bidirectional ESN

A Bidirectional Echo State Network (BESN) consists of a ESN and a common output layer (Figure 44). Furthermore, the ForESN processes the residue of interest and the information from the preceding amino-acids (upstream information), whereas the BackESN takes the residue of interest and the subsequent amino-acids (downstream information). This way we incorporate in our network the interactions among adjacent amino-acids.

The training procedure of the BESN is similar to the simple ESN (Section 2.1.5.1). A sliding window is used to introduce the data to the reservoir, but instead of passing one residue in each slide we pass a compound of residues. This smaller (internal) window is called w_f and w_b for the *ForESN* and *BackESN*, respectively; this way the network also

takes into account patterns of neighbouring amino-acids which help it improve its predictions. After the residue of interest is processed from both reservoirs, we compute their output by applying *Equations 70 and 71* to each ESN (*ForESN* and *BackESN*) and then adding their results. The outcome is three real values, one for each possible secondary structure (C, E, H), the largest value is our prediction. In addition, for each prediction we save the state X (threshold, reservoir values $x(n)$ and input $u(n)$) in order to update the output weights (W_{out}) in each ESN based on *Equations 72*, after all data were presented.

$$x(n) = (1 - a)x(n - 1) + af(W_{in}[1; u(n)] + Wx(n - 1) + W_{fb}y(n - 1)) \quad (70)$$

where $a(0; 1]$ is the leaking rate, the 1 in matrix $[1; u(n)]$ is the threshold and $f()$ is the activation function used for the reservoir neurons. Usually the hyperbolic tangent function is used for $f()$.

$$y(n) = W_{out}[1; u(n); x(n)] \quad (71)$$

where $[1; u(n); x(n)]$ is a matrix with the concatenation of the threshold, the input value $u(n)$ and reservoir values $x(n)$.

$$W_{out} = Y_{target}X^T(XX^T + \beta I) - 1 \quad (72)$$

where Y_{target} is a matrix with all the desired outputs $y(n)$ of the shown input data until now, X a matrix with the threshold and all the reservoir values $x(n)$ and inputs $u(n)$ until now, X^T its transpose, β the regularization coefficient and I the identity matrix.

Results on the CB513 dataset has shown the potentials of this model. The BESN, with two reservoirs of 800 neurons each, has achieved an accuracy around 74% $Q3$, without any ensemble methods or filtering techniques. This model is using non-linear neurons to achieve acceptable results in no time compared to the rest of the models which have used in this work. More specifically, the BESN is trained with simple linear regression learning algorithm which can update the model's parameter in one time-step.

5.5 Simple FFNNs trained with HFO learning algorithm

RNNs have been successful in handling sequential data. Training RNNs is a demanding task in terms of time and space efficiency because of the complexity of the models where many parameters have to be handled to specify the model's architecture and characteristics. Furthermore, training of these models is a difficult task because of the vanishing gradient and exploding gradient problems, where the gradient is getting smaller as the information moving backward through hidden layers is getting very big at the early layers of a model [Bengio et al. \[2004\]](#). The most common algorithm to train these models is based on the GD minimization method. Unfortunately, this kind of algorithms have a poor convergence rate [Møller \[1993b\]](#). Moreover, they depend on parameters which have to be specified by the user and are usually crucial for the performance of the algorithm.

In order to improve these drawbacks, simple models and more efficient algorithms need to be used. Simple FFNNs can be trained really fast with low demands in terms of time and space. The most common learning algorithm for training FFNNs is the Back-propagation learning algorithm (BP) [Rumelhart et al. \[1986b\]](#) which is based on the GD

method. Unfortunately, it has been proven that FFNNs have poor results in this kind of problems [Qian and Sejnowski \[1988\]](#); [Rost and Sander \[1993\]](#). On the other hand, latest developments in the field of ANN training algorithms, such as the Hessian Free Optimization (HFO) [Martens \[2010\]](#); [Martens and Sutskever \[2011\]](#) second order learning algorithm, can converge faster and more accurately. This algorithm has been found to be superior to the conventional BP algorithm in terms of accuracy, convergence rate and the vanishing-gradient problem [Martens \[2010\]](#); [Martens and Sutskever \[2011\]](#). In addition, the original form of the algorithm [Martens \[2010\]](#); [Martens and Sutskever \[2011\]](#) does not depend on any parameters.

The HFO [Martens \[2010\]](#); [Martens and Sutskever \[2011\]](#) second order learning algorithm demonstrated promising results on problems such as the noiseless memorization problem, the 3-bit temporal order problem and the random permutation problem [Martens and Sutskever \[2011\]](#). In this algorithm, the finite differences method is applied on the model's gradient vector in order to quickly calculate an approximation of the Hessian Matrix. The system's gradient vector and Hessian Matrix approximation are used to compute the system's Taylor expansion function. Then, the Preconditioned Conjugate Gradient algorithm (PCG) [Martens and Sutskever \[2011\]](#), which is a variant of the CG algorithm, is used to optimize the calculated Taylor expansion function. The PCG algorithm is used to give a new step size and a direction to update the system's parameters. Moreover, it has been proven that this algorithm, which was based on the works [Møller \[1993b,a\]](#), [Pearlmutter \[1994\]](#) and [Gers et al. \[2002\]](#), can manage well to train RNNs. Consequently,

HFO has been applied on deep ANN architectures and it was demonstrated to outperform other learning algorithms in specific sequential problems [Martens and Sutskever \[2011\]](#).

The need to build efficient methods in terms of accuracy, convergence time and simplicity for sequential data where both the upstream and downstream information of a sequence is important for a specific time-step and more specifically for the PSSP problem, has been the initial motivation for this work.

Consequently, in this work we demonstrate that providing profiles of protein sequences for training simple FFNNs with the HFO algorithm yield results comparing well with much more complicated ANN architectures in the 3-state PSSP problem. Furthermore, we demonstrate that a simple majority vote of 5 HFO-trained FFNNs, coupled with a SVM classifier for filtering purposes, achieves performance directly comparable to the current state-of-the-art methods. In our approach, a single FFNN trained with the HFO learning algorithm can achieve an approximately 79.6% per residue (Q_3) accuracy on the PISCES dataset. Despite the simplicity of our method, the results are comparable to some of the state of the art methods which have been designed for this problem. An average ensemble method and filtering with Support Vector Machines have also been applied, which increase our results to 80.4% per residue (Q_3) accuracy. Moreover, the HFO does not require tuning of any parameters which makes training much faster than other state of the art methods, a very important feature with big datasets and facilitates fast training of FFNN ensembles.

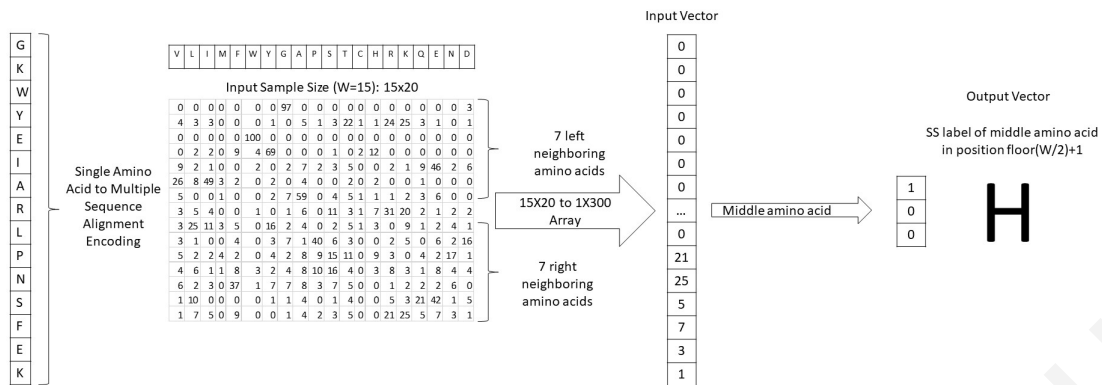


Figure 45: **Data Representation:** Each amino acid is within a window (W) centered at the residue of interest, encoded by a 20-dimensional MSA profile vector (see text for more details).

5.5.1 Methodology

5.5.1.1 Data Representation

The major obstacle on trying to solve a complex sequential data classification problem with any ANN is the representation of the data, in such a way that the network is able not only to understand the shape of the input volume, but also to track the complex sequence correlations among the input volume. Hence, multiple sequence alignment (MSA) profiles have been used for data preprocessing and PS encoding [Rost, 1996]. MSA profiles have been shown to enhance machine learning-based PSSP, since they incorporate useful evolutionary information for the encoding of each position of a protein. More specifically, each amino acid on a protein PS is replaced by a 20-dimensional vector, which

corresponds to the frequencies of 20 different amino acid types aligned to the query sequence after a PSI-BLAST [Altschul et al., 1997] search against the NCBI-NR (NCBI: <http://www.ncbi.nlm.nih.gov/>) database.

As shown in *Figure 45*, the network's input vector at each time-step t consists of the MSA information contained in a sliding window W on a protein PS. More specifically, we have created an input volume by placing MSA [Wallace et al., 2005] profile vectors of each amino acid one after another to construct a 1D representation of the MSA profiles of a certain number of neighboring amino acid residues. Through this technique, the attention given to any neighboring amino acid correlations is equally weighted across all the input volume, for each W given. This lets the FFNN discover and capture any strong short range correlations among the input records and consider them all equally in terms of the output volume created. The target output is the respective class of the amino acid which is located in the center of a specific input W . The FRNN processes simultaneously the residues located on the left and on the right side of the position t to predict the corresponding SS class.

5.5.1.2 The HFO learning algorithm

HFO [Martens, 2010; Martens and Sutskever, 2011] is a second order optimization algorithm of real-valued objective functions. It is a variation of the standard Newton's method [Li and Yan, 1995] which uses local quadratic approximations to generate update proposals. In high dimensionality problems, for which large ANN models with many hidden layers are employed, first order optimization algorithms like GD can be extremely

slow and ineffective due to the vanishing gradient problem [Hochreiter and Schmidhuber, 1997]. In GD methods, the updates are proportional to the gradient of the error function which is back-propagated through the layers of the model. Each time the error is back-propagated, the gradient becomes vanishingly small which results in the front layers having close to zero information on how to update their weights, meaning slow to completely ineffective training [Martens, 2010; Martens and Sutskever, 2011].

The advantage of using a second order optimization algorithm (i.e., Newton's method or HFO) is that these algorithms consider the curvature of the error surface (Hessian Matrix (HM)) in their optimization process which results in extremely better step-wise performance. More specifically, instead of fitting a plane at an initial solution and then determining the step-wise jump like first order algorithms, second order methods find a tightly fitting quadratic curve at that point and directly find the minimum of that curvature, which is supremely fast and efficient. Computing the HM for a large ANN with thousands to millions of free parameters however is not always possible due to the extremely high memory requirements needed to store it. Consequently, while there have been a number of Newton's variations like Newton-CG, CG-Steihaug, Newton-Lanczos [Nash, 1984] and Truncated Newton [Nash, 2000], none of them has been applied effectively to ANN models and consequently their applications in this domain have been extremely limited [Martens and Sutskever, 2011].

The Hessian Free method proposes solutions to the high memory requirements of second order learning algorithms, which enable it to be effective for ANN training. First of all, it does not compute and store the whole HM. Instead, it computes just the dot

product of the HM H with an arbitrary vector u ($H \cdot u$) [Martens and Sutskever, 2011], using mathematical methods like finite differences which cost as much as a single gradient evaluation. This works really well for the HFO algorithm since it does not require the explicit use of the HM, but rather many dot products based on $H \cdot u$. Secondly, the local quadratic objectives, which are approximated with second-order methods, can be efficiently optimized using the linear conjugate gradient (CG) method [Johansson et al., 1991; Charalambous, 1992] in order to compensate for the lack of the HM. While the CG method needs N iterations to converge in quadratic function, where N is the number of the free parameters of the network, there is a number of stopping criteria, which terminate it at early stages when significant progress in the minimization process has been made. This is extremely important since it is clearly impractical to wait for a complete CG convergence when there is a very low margin of further minimization.

It is important to note that even though in HFO no HM is calculated, there are no approximations done and the $H \cdot u$ product is calculated accurately. The only difference between HFO and Newton's method is that while the standard Newton's method performs a complete optimization of the approximated quadratic information, HFO does not. This is because the CG does not fully converge [Martens, 2010]. However, the efficiency-related benefits of avoiding the full HM calculation and inversion are clearly more beneficiary than the extremely small difference in accuracy by the not fully converged CG.

Finally, although the $H \cdot u$ product can be calculated efficiently and accurately, it is not the one usually used in HFO. Based on the same theory, the $G \cdot u$ product is used, where G is an approximation of the HM which is called the Gauss-Newton matrix [Schraudolph,

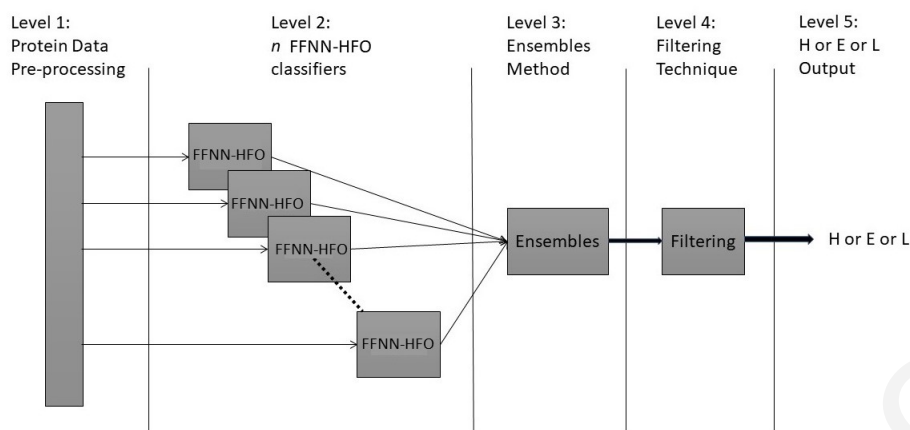


Figure 46: **Our proposed methodology for the PSSP problem** (see text for more details).

2002]. While it seems pointless to use an approximation instead of the correct curvature matrix when there is no problem in efficiency, Gauss-Newton avoids some of the problems that the HM may face and cause the algorithm to be completely ineffective. In fact, comparing to the usage of the HM, the use of the $G \cdot u$ matrix consistently results in better search directions utilizing half the memory and running twice as fast [Martens, 2010].

5.5.2 The proposed Methodology

In this work, the HFO learning algorithm has been used for training FFNNs to handle the PSSP problem. Our methodology appears in *Figure 46*. As it can be seen, we have 5 levels of processing. In the first level of our methodology, the protein PS appears in MSA encoding (*Figure 45*). Input data is used from 5 individual FFNNs in the second level of our methodology. The small number of FFNNs has been chosen based on the work

of Baldi et al. [1999] and Kountouris et al. [2012]. These FFNN classifiers are trained with the HFO learning algorithm. Each FFNN returns three real values in the range (0,1) for the central residue of a sliding local window W , one for each secondary structure state. Subsequently, the corresponding outputs of each FFNN for each state are averaged through the ensembles level [Zhou et al., 2002, 2010]. Then, the resulting predictions from the ensembles level are used for the filtering [Kountouris et al., 2012] procedure. In the case of the PSSP problem, the filtering method which appears to have the best results is based on SVM models [Kountouris et al., 2012]. Consequently, we have created a stacked network architecture where an SVM model is used for filtering purposes. More specifically, the predicted SS sequence of the average ensemble method is used as input to the SVM model. Each predicted SS state is within a window (W_{svm}) centered at the position t of the residue of interest. For each position t of the sequence, this window is used as an input vector to the SVM which predicts the corresponding corrected SS state. Finally, the last level of our methodology returns three real values, which represent the predicted SS class for a specific input vector of a local window.

5.5.3 Results and Discussion

Our proposed methodology has been applied to the PSSP problem and through experimental analysis, various results have been extracted. These results demonstrate the efficiency and the effectiveness of the method.

<i>Fold</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	<i>SOV</i>	SOV_H	SOV_E	SOV_L
0	79.43	78.32	74.09	79.04	0.7172	0.7365	0.7488	0.6957
1	79.38	77.28	73.51	79.62	0.7226	0.7278	0.7447	0.7055
2	79.57	78.05	72.84	79.45	0.7180	0.7319	0.7360	0.6994
3	79.87	78.63	74.29	79.80	0.7229	0.7398	0.7500	0.7056
4	79.58	79.18	73.64	79.81	0.7205	0.7470	0.7437	0.7027
Average	79.57	78.29	73.67	79.54	0.7202	0.7366	0.7446	0.7018

Table 23: Experimental results for a single FFNN: Q3 and SOV Results for each Fold of PISCES dataset.

<i>Fold</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	<i>SOV</i>	SOV_H	SOV_E	SOV_L
0	80.02	77.28	73.58	81.80	0.7396	0.7464	0.7605	0.7214
1	80.11	77.19	73.69	81.44	0.7560	0.7560	0.7624	0.7323
2	80.19	76.83	72.22	82.33	0.7409	0.7395	0.7474	0.7284
3	80.40	77.32	73.62	82.60	0.7468	0.7495	0.7618	0.7317
4	80.12	78.08	73.07	82.58	0.7460	0.7598	0.7563	0.7314
Average	80.17	77.35	73.24	82.15	0.7459	0.7503	0.7577	0.7291

Table 24: Experimental results for an ensemble of 5 FFNNs trained with HFO: Q3 and SOV Results for each Fold of PISCES dataset.

<i>Fold</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	<i>SOV</i>	SOV_H	SOV_E	SOV_L
0	80.24	77.75	73.30	81.78	0.7641	0.7800	0.7642	0.7313
1	80.27	77.05	73.62	81.71	0.7682	0.7684	0.7670	0.7369
2	80.43	77.28	71.80	82.45	0.7653	0.7725	0.7504	0.7374
3	80.55	77.64	73.41	82.35	0.7699	0.7818	0.7659	0.7357
4	80.33	78.51	72.84	82.57	0.7683	0.7912	0.7598	0.7390
Average	80.37	77.65	72.99	82.17	0.7671	0.7788	0.7615	0.7361

Table 25: Experimental results for an ensemble of 5 FFNNs trained with HFO and filtered with a SVM: Q3 and SOV Results for each Fold of PISCES dataset.

5.5.3.1 Data Preparation and Simulation Details

Special care has been taken to retrieve datasets of the highest possible quality for the PSSP problem relying in specialized resources. For the purposes of this work, we have used the PISCES [Kieslich et al., 2016] dataset, which consists of 8632 protein chains. More specifically, high resolution protein structural data have been obtained from the RCSB Protein Data Bank (PDB; <http://www.rcsb.org/>). We have used the DSSP program (URL: <http://swift.cmbi.ru.nl/gv/dssp/>, accessed 01/04/2019), to extract the SS class for each amino acid in each dataset. The DSSP program uses the atomic coordinates and hydrogen bond patterns for assigning each amino acid in one of eight classes: H (α -helix), G (3_{10} -helix), I (π -helix), E (extended β -strand), B (isolated β -bridge), T (turn), S (bend) and C (other/coil). Then, we have reduced the eight classes to the three predefined SS classes as: H, G, and I to the helix state (H), E and B to the extended state (E) and the rest to the loop state (L). Moreover, MSA profiles have been used for data preprocessing and PS encoding. Each protein sequence position is replaced by a 20-dimensional vector. During this procedure, the MSA files of PISCES dataset have been analyzed and cleaned up from data with short or no information (*Table 8*). Furthermore, we have followed a strict 5-fold cross-validation approach as described in [Kieslich et al., 2016].

The network's input vector at each time-step t consists of the MSA information contained in a local sliding window W and the target output is the respective class of the amino acid which is located in the center of that local window. Firstly, a single FFNN has been trained on different single folds of the PISCES dataset. At this stage, we carried out

	Q3	SOV
Sample Standard Deviation (s)	0.1268	0.0023
Variance (Sample Standard) (s^2)	0.0160	5.678E-6
Mean (Average)	80.37	0.7671
Standard Error of the Mean ($SE_{\bar{x}}$)	0.0567	0.0010

Table 26: **FFNN Ensembles and SVM Filtering:** Statistical Analysis of Q3 accuracy and SOV score for fold 0-4 results presented in *Table 25*

multiple experiments to tune up the architectural parameters of the FFNN trained with the HFO learning algorithm. More specifically, we have concluded that, for the purposes of our methodology, the optimum FFNN architecture is 1 hidden layer FFNN of 75 Sigmoid neurons with Mean Square Error (MSE) function. Furthermore, we have used 3 neurons of this category for the output layer. This architecture has full connectivity between input-hidden and hidden-output layers. The only difference between the multiple FFNNs is the random weight initialization. These models have been evaluated based on the Q3 and SOV metrics, but only the Q3 metric was used for tuning. Then, this specific FFNN architecture has been successfully used for our proposed methodology (*Section III*). For our experiments, a machine with 6 cores and 32GB RAM of a CISCO UCS C240 M5 server (2 x Intel Xeon 6140 Gold Processors) has been used.

5.5.3.2 Cross-Validation simulations

In order to validate the robustness of our proposed methodology as well as to prove its efficiency to the exposure of various training and testing data, we had to complete the evaluation of the PSSP problem on the PISCES dataset, using a 5-fold cross-validation test. All the experiments made are with the optimal parameters of the proposed model. As shown in *Table 23*, the Q_3 accuracy and SOV score results of a single FFNN with a 5-fold

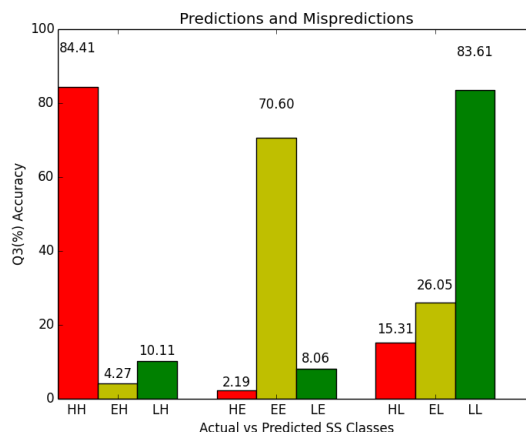


Figure 47: **Graph of Confusion Matrix for Actual vs Predicted SS Classes:** HH, EE and LL are the True Positive scores of our method for each class after the average ensemble method and SVM filtering technique. EH, LH, HE, LE, HL and EL are the scores for the mispredicted classes where the first letter is the actual class and the second letter is the predicted class. Based on an average of a 5-fold cross validation evaluation, the method can predict correctly with 84.41%, 70.60% and 83.61% the H, E and L classes, respectively.

cross-validation are 79.57% Q_3 and 0.728 SOV respectively. These results, compared to the results of other methods which are mentioned in this thesis, are high enough to be considered as a good solution for the PSSP problem. Given the simplicity of the model, we can say that the learning algorithm is powerful enough to optimize the problem and relate the neighboring amino acids to a SS based on the strong information which is coming from local dependencies of a protein PS. Furthermore, as it can be seen from *Table 23*, our method can predict better the H and L classes, where some difficulty is shown in the prediction of E class.

A major improvement in the results of single FFNNs trained with the HFO algorithm has been achieved when the average ensemble method and filtering techniques [Kountouris et al. \[2012\]](#) have been used. In this thesis, we employ an ensemble of 5 FFNNs as it has already been described. The results of the average ensemble method are shown in

Table 24. Clearly, this method corrects some missclassified SS states, thus increasing Q_3 by 0.55% and SOV by 0.0234 overall. Based on these results, the average ensemble method improves the method's accuracy. Moreover, based on the SOV results, it improves significantly the quality of the predicted SS sequence.

The resulting predictions are then used for SVM filtering, as explained in [Kountouris et al. \[2012\]](#). More specifically, after gathering the predictions from the average ensemble method of FFNNs, we have trained a SVM using a window of SS states predicted by the FFNNs. In this case, we have used a 5-fold cross-validation approach based on the folds which have been used for the training of our FFNNs. After performing several experiments using different kernels, misclassification penalty parameters (C) [Cortes and Vapnik \[1995b\]](#), Gamma values (G) [Cortes and Vapnik \[1995b\]](#) and window sizes (W_{svm}), we have decided on the optimal SVM parameters that lead to the highest Q_3 accuracy and SOV score on the PSSP problem and which are: (a) Kernel: Radial Basis Function, (b) $C = 1$, (c) $G = 0.001$ and (d) $W_{svm} = 7$. The final results are shown in *Table 25*. The final results of our methodology are approximately 80.4% Q_3 accuracy and 0.77 SOV for the PISCES dataset. More specifically, the SVM filtering technique can increase the accuracy of our proposed methodology by only 0.2% but it can improve significantly the quality of results by approximately 0.02 SOV units.

As it can be seen in *Tables 23 - 25*, the ensemble methods and filtering techniques which were applied to our methodology have increased the single FFNN accuracy by approximately 0.8% and the SOV metric by 0.05. Furthermore, in *Figure 47* we can see how the algorithm can manage with each SS class. Obviously, in the case of the PISCES

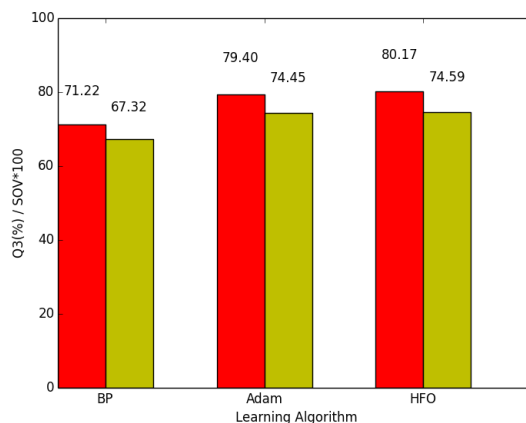


Figure 48: **Q3 accuracy and SOV score** for each FFNN optimizer. The red bar corresponds to Q_3 accuracy and the yellow bar to SOV score. The SOV score has been multiplied by 100 for presentation purposes (see text for more details).

dataset, the algorithm can predict correctly with 84.41%, 70.60% and 83.61% the H, E and L classes, respectively. As it can be observed, the H and L classes are most often predicted as such compared to the E class, a known shortcoming when applying FFNNs in the PSSP problem. Moreover, in *Table 26*, we can see that the values of Standard Deviation and Variance for the case of the Q_3 metric are very small which indicates that the algorithm behaves similarly between the different protein sequences of several dataset folds.

5.5.3.3 Comparison of our methodology to other methods

Finally, in order to evaluate the accuracy, quality of results and computational performance of our method, we have implemented other well known methods to compare their results with our method using the same data. More specifically, we have used the same

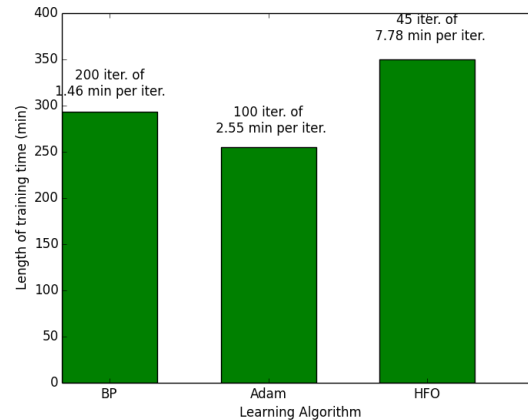


Figure 49: **Length of training time of the FFNN Optimisers:** The length of training time in minutes is calculated based on the average 5-fold cross-validation training time of the 5 FFNNs used in each ensemble. The label of each bar corresponds to 'number of iterations / average iteration execution time (min)' (see text for more details).

5-fold cross validation approach with the PISCES dataset. Hence, we have used two different strategies. At first we have used other optimizers to train and compare the ensemble of 5 FFNNs. Then, we have tested (using the same 5-fold cross validation procedure on our dataset) well established ML methods that have been used on the PSSP.

The optimizers which were chosen to train the ensemble of FFNNs on the PISCES dataset were the BP [Rumelhart et al. \[1986b\]](#) learning algorithm, as the most common benchmark training algorithm in training FFNNs, and the Adam [Diederik and Jimmy \[2015\]](#) learning algorithm, as one of the latest and most powerful developments in the field of training algorithms for ANNs. Compared to the HFO learning algorithm, which is a second order learning algorithm, both BP and Adam are first order gradient descent learning algorithms. Although all the training algorithms were applied to the same FFNN architecture, which is described in detail in *Section IV(A)*, the results are indicative for this problem. Nevertheless, further optimization can be done for the FFNN architectures

which are trained with the BP and Adam algorithms. The parameters of each one of the training algorithms were tuned through many executions on a single FFNN trained for the PISCES dataset. In the case of BP we have identified as optimal the following set of parameters: learning rate=0.01 and momentum=0.0. Similarly, in the case of Adam we have used learning rate=0.001, beta1=0.9, beta2=0.999. As it can be seen from *Figure 48*, the results of BP compared to Adam and HFO learning algorithms are very poor. On the other hand, the HFO learning algorithm has achieved 0.8% better Q_3 accuracy and comparable SOV score compared to the Adam learning algorithm. As it can be seen from *Figure 49*, although the HFO algorithm needs only 45 iterations to be trained, each iteration requires an average execution time of 7.78 minutes. On the other hand, the BP and the Adam algorithms need 200 iterations of 1.46 minutes and 100 iterations of 2.55 minutes each to be trained, respectively. An HFO iteration is executing multiple times the PCG algorithm, a number which cannot be estimated before the completion of an iteration. Therefore, the training time needed for HFO is approximately 290 minutes. The length of this execution time is 17% and 27% more than the training time needed for BP and Adam learning algorithms, respectively, despite needing much less iterations. Nevertheless, the entire process of building FFNN classifiers is much faster with the HFO learning algorithm because there are no parameters to be tuned compared to the three parameters of the Adam learning algorithm which need to be optimised by trial and error.

Then, we have chosen to compare our method with an ensemble of 6 BRNNs trained with the BPTT learning algorithm, as one of the most established methods for this problem [Baldi et al. \[1999\]](#); [Pollastri et al. \[2002\]](#); [Cheng et al. \[2005\]](#); [Magnan and Baldi](#)

[2014] and also with an LSTM-BRNN, as the method which has been reported as obtaining the highest results for the PSSP problem Heffernan et al. [2017]. The BRNN architecture consisted of 1 hidden layer of 20 hyperbolic tangent neurons for the feed forward subnetwork and 1 layer of 11 hyperbolic tangent neurons for each one of the recurrent subnetworks. Furthermore, the BPTT has been tuned to learning rate=0.1 and momentum=0.001. The LSTM-BRNN consisted of 25 Bidirectional LSTM neurons and it has been trained with the Adam learning algorithm where the parameters of learning rate=0.001, beta1=0.9, beta2=0.999 were used. Both methods use the Mean Square Error function. As it can be seen in *Figure 50*, the BRNN trained with BPTT has achieved 77.61% Q_3 accuracy for the PISCES dataset, which is much lower than our results which were approximately 80.4%. Furthermore, although LSTM-BRNNs are reported to capture the long range dependencies of a sequence, it gave comparable overall results to our (more local) method. Definitely, there needs to be more work in order to compare the results of the two methods in such detail (work in progress). Given that our method can capture very well the short range dependencies, the LSTM-BRNNs may face difficulties in short range dependencies which results in the same accuracy to our methodology. Furthermore, our methodology provides an upper limit on what methods may be able to capture when they rely on only local sequence patterns. The biggest advantage of our methodology to these two methods is the simplicity of the models we use. This is very important if we take into account the latest developments in the field which demand very big datasets and network architectures, which consequently increase exponentially

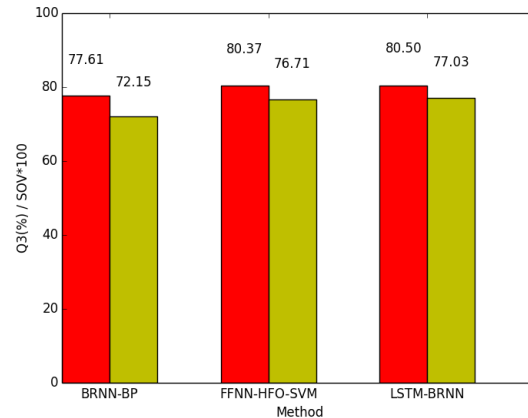


Figure 50: **Q3 accuracy and SOV score** for ANN models used for the PSSP problem. The red bar corresponds to Q_3 accuracy and the yellow bar to SOV score. The SOV score has been multiplied by 100 for presentation purposes (see text for more details).

the amount of training time. In addition, many of these methods are combined in ensembles, as in Baldi et al. [1999], where the training amount of time is even more increased. This is extremely important if we take into consideration that an increasing size of ensembles is often used to improve the results of ML methods Zhou et al. [2002]; Granitto et al. [2005a]; Zhou et al. [2010].

5.5.4 Conclusion

In this work, we present a second order-based methodology for training simple FFNNs for the challenging PSSP problem where both the upstream and downstream information of a sequence is important for a specific time-step. In this methodology, to the best of our knowledge the HFO learning algorithm is applied for the first time to this problem. More specifically, we present the development and implementation of a methodology where ensembles of FFNNs are trained with the HFO learning algorithm for the PSSP problem.

In contrast to the conventional GD learning algorithm, the HFO exploits both gradient and curvature information for fast convergence. The results from the ensembles of FFNNs are combined through average ensemble methods which are then fed for filtering purposes to a SVM model for producing the final results.

The efficiency and effectiveness of simple FFNNs trained with the HFO learning algorithm have been tested on the PISCES dataset, achieving approximately 79.6% Q_3 accuracy and approximately 0.72 SOV score which compare well with other state of the art methods. The average ensemble method and SVM filtering techniques have improved even more the single FFNN Q_3 accuracy by approximately 0.8% and the SOV metric by 0.05. The final results of our methodology are approximately 80.4% Q_3 accuracy and approximately 0.77 SOV score. In terms of accuracy, our proposed method outperforms similar methodologies which are trained based on GD or Adam algorithms. Although, the GD and Adam algorithms seem to converge faster than the HFO learning algorithm, the process of building FFNN classifiers is much faster with the HFO because there are no parameters to be tuned. Therefore, our ML method seems to be a particularly good option for complex feature extraction and prediction on sequential data, as it takes advantage of the benefits of these techniques.

At first glance, this method seems to have good chances to outperform in terms of accuracy and convergence time some of the state of the art methods, such as SSpro and SCRATCH Baldi et al. [1999]; Pollastri et al. [2002]; Cheng et al. [2005]; Magnan and Baldi [2014] where hundreds of BRNNs are used to achieve an accuracy near

to 80%. Hence, to conclude, the predicted sequences from many simulations of our proposed methodology on different PSSP datasets must be carefully extracted and analyzed compared to the results of these state of the art methods. Furthermore, our method has comparable results with the method based on the LSTM-BRNN models which can handle long range dependencies and it has been reported as the method with highest results on this problem. In contrast, our approach provides an upper limit on what methods may be able to capture when they rely on only local sequence patterns. Consequently, this method takes advantage of the strong local dependencies of amino acids. Furthermore, the simplicity of our models is very important if we take into account the latest developments in the field with very big datasets, network architectures and ensembles of networks. Finally, latest developments in the 3D structure prediction of a protein can benefit from our solution on the PSSP problem. More specifically, [AlQuraishi \[2019a\]](#) has presented work, based on deep learning methods, to predict the 3D structure of a protein from its PS. The author has indicated a possible improvement to his methodology if he incorporate PSSP results from other algorithms [AlQuraishi \[2019a\]](#).

The accuracy of 100% will probably never be achieved for the PSSP problem because of the presence of disordered regions, the ambiguities inherent in the definitions of secondary structure, the errors and uncertainties contained in databases and the role of the solvent and other molecules, as well as the inherent protein structural dynamics [Magnan and Baldi \[2014\]](#). Nevertheless, the improvement and systematic combination of sequence profiles, machine learning methods and sequence-based structural similarity methods seem to be the best strategy to improve the results related to the PSSP problem.

A contribution on any of these three categories, combined with other data preprocessing and algorithmic methods may play a catalytic role on the general improvement of the PSSP problem results.

As a conclusion to all the results presented in this work, we can see that the second order HFO learning algorithm can effectively detect and extract features from complex sequential data like the PSSP problem. Furthermore, the combination of multiple ML algorithms seem to be a particularly good option for complex feature extraction and prediction on sequential data, as it takes advantage of the benefits of all techniques. Finally, we demonstrate that a powerful learning algorithm, such as the HFO algorithm, applied on simple ANN models for the PSSP problem can produce comparable results to the most complicated ANN architectures which have been utilized for this problem. Also, this work, gives the initiative to other techniques where powerful learning algorithms can be used for complicated ANN architectures to produce even more accurate results for the PSSP and other related problems. Moreover, these methods may be applied to more problems where the upstream and downstream information is important for a specific time-step of the sequence for more generic conclusions.

5.6 Long Sort-Term Memory BRNN (LSTM-BRNN)

The method which has been reported with the highest results for the PSSP problem is based on the LSTM-BRNN architecture [Heffernan et al., 2017]. This method uses the LSTM cells to capture the long-range dependencies of sequential data where both the upstream and downstream information is important to make predictions on a specific

<i>Fold</i>	$Q_3(\%)$	$Q_H(\%)$	$Q_E(\%)$	$Q_L(\%)$	SOV	SOV_H	SOV_E	SOV_L
Fold 0	80.99	78.38	73.50	80.19	0.7788	0.7828	0.7734	0.7292
Fold 1	80.50	76.86	73.44	80.25	0.7763	0.7648	0.7671	0.7353
Fold 2	80.39	74.52	68.35	84.54	0.7643	0.7483	0.7322	0.7374
Fold 3	80.14	75.82	65.20	85.82	0.7595	0.7618	0.7148	0.7339
Fold 4	80.46	75.14	71.93	83.79	0.7728	0.7641	0.7578	0.7441
Average	80.50	76.14	70.48	82.92	0.7703	0.7644	0.7491	0.7360

Table 27: **Results of the LSTM-BRNN methods:** 5-fold cross-validation approach for the PISCES dataset.

time-step of the sequence. Based on the work of [Heffernan et al. \[2017\]](#), we have trained LSTM-BRNNs with our PISCES dataset. The results of 5-fold cross validation are shown in [Table 27](#). This method has achieved an accuracy of 80.5% Q_3 and 0.7703 SOV . This work has been implemented for comparison purposes to our methods.

5.7 Chapter Contribution

A list of contributions resulted from this chapter is presented below:

1. The introduction of an image-like input representation of the protein MSA profiles for the CNN networks.
2. The achievement of approximately 80.4% Q_3 accuracy for the PSSP problem with CNNs.
3. The introduction of a new BESN architecture to handle bidirectional sequences. This architecture can be trained with a simple linear regression learning algorithm.
4. The application of the CW-RNN model to the PSSP problem with comparable results to BRNN but much faster training.

5. The achievement of approximately 80.4% Q3 accuracy for the PSSP problem with a novel method of simple Feed-forward Neural Networks trained with the HFO learning algorithm.

Michalis Agathocleous

Chapter 6

General Discussion and Conclusions

6.1 Overview of the problems

Learning, is a many-faceted phenomenon. The learning process includes the acquisition of new declarative knowledge, the development of cognitive skills through instructions and practice, the organizing of new knowledge into general, the effective representation of data and finally, the discovery of new theories and facts through practice and experimentation. Analysis of sequential data, feature extraction, prediction algorithms/techniques and ML methods have been excessively studied. The procedure of learning sequential data becomes even more challenging when both the upstream and downstream information of a sequence is useful for making predictions at a specific time-step. Processing this category of sequential data for classification, forecasting or data mining purposes is still an open field of research due to its complexity. ML models,

learning algorithms and methodologies for sequential data must take into account how to (a) capture and exploit sequential correlations, (b) represent and incorporate loss functions, (c) identify long-distance dependencies, and (d) make the optimisation algorithm fast [Dietterich, 2002]. Many ML techniques have been developed under these standards to handle sequential data but not many of them have been modified or developed to process sequential data when both upstream and downstream information is important to make predictions or classification at a specific time-step of a time-series. These methods have to be developed and evaluated based on the accuracy, efficiency and effectiveness of the models, evaluated in terms of computational time and space but also on the quality of the results on related problems.

The most efficient and effective models to process sequential data are RNNs [Elman, 1990]. Based on the theory of these models, a specific architecture of RNNs has been designed to handle the specific class of sequential data where both upstream and downstream information are needed. This architecture is called BRNN [Baldi et al., 1999; Schuster and Paliwal, 1997]. BRNNs are based on RNNs' architecture theory which efficiently incorporates temporal dynamics [Elman, 1990]. Moreover, many ML models have been used to handle sequential data. Such models are LSTM-RNNs, CNNs, CW-RNNs, RC and simple FFNNs. The main questions here are:

1. How can we improve existing BRNN architectures?
2. How can we include these architectures in ML methodologies to improve their results?

3. How can we improve the training procedures of these architectures through efficient and effective learning algorithms in terms of accuracy, computational time and quality of results?
4. Are these methods good enough for the class of sequential data they have been designed for?
5. Which and how other ML methods can be evolved to handle this category of data?
6. Can we use RC methods or CW-RNNs to handle these problems?
7. How can we use CNNs in such problems? Can they show results which are comparable to the results of other methods?
8. Can we use for the same class of data simple FFNNs trained with second order learning algorithms (i.e., SCG or HFO)?

For the purposes of our work we have chosen the PSSP [Agathocleous et al., 2010, 2016; Baldi et al., 1999; Kountouris et al., 2012] problem, where the SS of a protein sequence must be predicted only through the consecutive amino acids appearing in its PS in order for its functionality to be specified. This problem belongs to the specific class of sequential data where both upstream and downstream information are needed to make predictions at a specific time-step of the sequence. The PSSP problem is an important problem in the biological sciences. The PS of a growing number of proteins is known. However, there is missing information with respect to their 3D structure. Besides being important for a basic understanding of life itself, knowledge of protein structure may assist in research against disease, providing the means for improving the general quality

of life. One of the goals of this thesis is to examine how the existing results for the PSSP problem can be improved either in terms of accuracy or training time of ML models.

The main purpose of this research work is to investigate current ML methods, design and develop new modeling methods and novel optimization algorithms that can make predictions on sequential data where important information for a specific time-step of the sequence is located upstream and downstream, and more specifically for the PSSP problem. This research has its highest impact in the field of ML in general by allowing cross-fertilisation of ideas between the research areas of analytical learning optimisation algorithms and effective implementation of practical applications for the PSSP problem, which are based on human-technology interaction and in the longer term, improve people's quality of life.

6.2 Overview of the approaches, results and conclusions

Based on the problems and questions which have been raised in *Section 6.1*, we have chosen our approach for this thesis. Firstly, we have chosen to study the BRNN architecture of [Baldi et al. \[1999\]](#), which has been utilized to handle the PSSP problem. Based on this method, we have also investigated a general methodology to improve the results of this and other ML models based on ensemble methods and filtering techniques. Then, we have investigated new optimization algorithms for BRNN architectures. Based on the advantages and drawbacks of the famous BPTT learning algorithm, which has been used to train BRNNs, we have chosen to investigate how we can apply second order learning algorithms on these architectures in an effort to improve learning for BRNNs. More

specifically, we have studied the SCG and HFO learning algorithms. Finally, we have chosen other powerful ML models, algorithms and methods which have never been used for this class of problems and we have investigated how can they be manipulated to work for these difficult sequential problems. More specifically, we have used methods which are based on CNNs, RC methods, CW-RNNs and simple FFNNs trained with the HFO learning algorithm.

In *Chapter 3*, we wanted to investigate and improve existing BRNN architectures for the PSSP problem. We also investigated how other ML methodologies like ensemble methods and filtering techniques as post-processing and NN methods can be used to improve the results of the PSSP problem. The initiation of this work has been inspired by the work of [Baldi et al. \[1999\]](#). The BRNN architecture of [Baldi et al. \[1999\]](#) is considered as one of the most optimal computational neural network type architectures for addressing the PSSP problem and in general the specific category of sequential data. Firstly, for the purposes of this work, we have redeveloped the work of [Baldi et al. \[1999\]](#). Through this process we have sorted out limitations of the model based on the learning algorithms and the lack of capturing long range dependencies. Based on these observations, we have developed a variation of this methodology. We have implemented the same BRNN architecture, but we have used a modified training procedure. More specifically, our aim was to identify the effect of the contribution of local versus global information on the PSSP problem, by varying the length of the segment on which the RNNs operate for each residue position considered. Our results with a single BRNN are better than the results of [Baldi et al. \[1999\]](#) by three percentage points (Q3) and comparable to ensembles of

BRNN models' results which appear in Baldi et al. [1999]; Pollastri et al. [2002]; Cheng et al. [2005]; Magnan and Baldi [2014]. This outcome shows that the original BRNN architecture trained with the standard BPTT learning algorithm finds difficulties in capturing the long range dependencies.

Furthermore, in *Chapter 3*, we wanted to investigate and conclude in post-processing methods which can improve the results of BRNNs for the PSSP problem. More specifically, we have been inspired again by the work of Baldi et al. [1999] to investigate how ensemble methods can improve the outcome of BRNNs. Ensemble methods is a category of well known methods which are used in ML to improve the performance of a learning model [Dietterich, 2000; Zhou et al., 2002, 2010; Li et al., 2018a; Zheng et al., 2019]. More specifically, through ensemble methods, instead of training just one model and get a single prediction, we train multiple instances of same or different methods and we combine the results. Then, the "winner takes all" method is used to take the results of each model and the class with the most representations is the final class of a specific input. Ensembles reduce the mis-predicted residues by combining the results of multiple classifiers. We have investigated 4 different ensemble methods: Voting, Borda Function, Average and Weighted Average. We have used ensembles of 6 BRNNs to enhance our results and based on the results we have concluded that the best ensemble method is the average ensemble method. This method has significantly improved the accuracy of BRNNs by 2-3%. In addition, and trying to find the best post-processing methodology for BRNNs, our results improved even further when sequence-to-structure output is filtered

in a post-processing step, with a novel Hidden Markov Model-based approach. Filtering of protein secondary structure prediction aims to provide physicochemically realistic results, while it usually improves the predictive performance.

The improvement we have noticed in our results based on Hidden Markov Model-based filtering method has motivated us to perform a comparative study on the challenging problem of filtering PSSP, utilising both widely used empirical smoothing rules and ML techniques. Using the average ensemble of 6 BRNNs with per-residue weight updating [Agathocleous et al., 2010], we have predicted the SS on two non-redundant, non-homologous datasets and, subsequently, we have applied a number of filtering techniques to smooth the predictions. We have used many different ML techniques (*Section 3.4*). Importantly, the SOV increases significantly in most cases. On the other hand, some classifiers increase the per-residue accuracy, whereas others decrease it. The Logistic function, the MLP and the SVMs were found to be superior to the tested methods in terms of both Q3 and SOV score. Notably, the results improve even further when we use combinations of ML algorithms and empirical filtering rules. This work has been published in Kountouris et al. [2012]. Based on the results of *Chapter 3*, we have concluded that an average ensemble method combined with a SVM filtering method can improve significantly both the accuracy and quality of results for the PSSP problem.

Training BRNNs with BPTT learning algorithm has many difficulties and drawbacks. Firstly, during training the network parameters gradually change in a way that the network dynamics are driven through bifurcations [Doya, 1992]. This leads to the degeneration of gradient information which cannot guarantee the algorithm convergence. Secondly,

long training times are shown for each single parameter update. This situation is computationally expensive and creates a bound on the network's size. Another problem is the well known vanishing gradient problem. The necessary information exponentially diffuses over time [Bengio et al., 2004] which destroys the long-range memory of the network. Finally, global control parameters of complex learning algorithms are not easily optimized which may lead to the creation of a chaotic system. Based on these difficulties and drawbacks, we have studied new learning algorithms which have never been used for BRNNs and how we can develop them for these architectures. Chang and Mak [1999] and Martens and Sutskever [2011] have shown that second order learning algorithms can be very efficient and effective for training RNN models to increase the performance for several problems. Second order learning algorithms have been found to be superior to the conventional BPTT algorithm in terms of accuracy, convergence rate and the vanishing-gradient problem [Hochreiter and Schmidhuber, 1997]. After this initial investigation, in *Chapter 4* we have chosen to analytically design and develop the SCG [Møller, 1993b] learning algorithm for the BRNN architecture. Unfortunately, after several tries, this attempt has suffered from stability issues and poor results. Based on the analysis of this behavior, we developed a variation of SCG learning algorithms which is applied for the first time on BRNN architecture. In particular, we present the development and implementation of the Hybrid Rectified-Scaled Conjugate Gradient (HR-SCG) learning algorithm for BRNN architectures which is based on the SCG. In contrast to the conventional Gradient Descent learning algorithm, the HR-SCG exploits both gradient and curvature information for convergence. Moreover, the HR-SCG methodology has also been enriched with

a version of GD and Dropout algorithms for faster, better and more accurate results. The model has been tested on the PSSP problem and achieved 77.6% per residue accuracy on a specific PSSP dataset. Moreover, it has been shown that the HR-SCG outperforms the BPTT learning algorithm for BRNNs in terms of convergence time, needing approximately 75% less time on PSSP datasets. The final results on the PSSP problem have demonstrated that a BRNN trained with our version of the SCG learning algorithm can capture patterns and make predictions on complicated sequences where the information in both upstream and downstream direction is important. Furthermore, the SCG learning algorithm needs much less training iterations and parameters to tune up than the conventional BPTT learning algorithm. This is very important if we take into account the latest developments in the field which demand very big datasets and network architectures, which consequently increase exponentially the amount of training time. In addition, many of these methods are used in ensemble methods where the amount of training time is even more increased. Furthermore, in *Chapter 4*, we have chosen to apply the HFO learning algorithm to the BRNN architecture. Through the work of [Martens and Sutskever \[2011\]](#), HFO is thought to outperform other learning algorithms. The final results of this model have shown comparable results to BPTT and HR-SCG learning algorithms. A general conclusion from this chapter is that powerful learning algorithms can improve significantly the training time needed for BRNNs but they all have comparable results in terms of the model's accuracy.

Finally, in *Chapter 5*, we wanted to investigate if the simple BRNN method is the most appropriate or not to handle the PSSP problem and if not, to examine which ML

methods can be evolved to handle this category of data used by the PSSP problem. This stemmed from the fact that many powerful recent NN models, their variations and specific learning algorithms have never been used to handle sequential data where both the upstream and downstream information is important to process a specific time-step of the time-series. Consequently, many ML methods have never been used for the PSSP problem. Hence, there is an open field to be investigated related to the PSSP problem and CW-RNNs (*Section 2.1.4*), CNNs (*Section 2.1.5*), RC methods (*Section 2.1.6*) and simple FFNNs with powerful learning algorithms, such as HFO have been used on this problem. Although these techniques have their advantages and disadvantages compared to each other, each one has specific characteristics related to accuracy, execution time, whether they take into account short and long term dependencies, etc. Hence, in *Chapter 5*, we have presented results on a novel image-like input representation method for the PSSP problem which is used by CNNs, results of the same problem on CW-RNNs, results on a novel Bidirectional ESN architecture and results on simple FFNNs trained with the powerful HFO learning algorithm. Also, based on the work of [Heffernan et al. \[2017\]](#), we have presented results of LSTM-BRNNs on our datasets.

In *Chapter 5*, we have developed and implemented a hybrid machine learning method based on the application of CNNs in combination with SVMs, for complex sequential data classification and prediction [[Dionysiou et al., 2018](#)]. For the purposes of this work, many attempts have been made to preprocess and encode the data in a way the CNN can extract features and useful information. Unfortunately, many of them have failed.

Finally, we have designed and implemented a novel two dimensional (2D) input representation method for sequential data which was able to encode the sequential information of the PS of a protein. Then we have tested it on the PSSP problem for 3-state secondary structure (SS) prediction. The results of this method were much better than the results we have shown in *Chapters 3-4*. More specifically, improved results have been obtained compared to BRNN methods. Clearly, the correct encoding of sequential data in an image-like representation has enclosed all the necessary and important features of the data. Moreover, impressive was the fact that these results were comparable to the results of LSTM-BRNN method which was shown to be the best method of tackling the PSSP in terms of accuracy [Heffernan et al., 2017]. Under the same scope, we have used CW-RNNs and our BESN architecture for the PSSP problem. These methods did not outperform CNNs or LSTM-BRNNs but they have achieved good results in significantly less training, which is very important in case of ensemble methods where many parallel models have to be trained. Finally, through this chapter, we have concluded that very complicated NN models trained with the simple BPTT learning algorithm can give the best results for the PSSP problem in terms of accuracy and quality of results. One drawback of these models is that they are very complicated and need too much time to be trained. Under this scope, we have indicated the need to build efficient methods in terms of accuracy, convergence time and simplicity to handle sequential data where both the upstream and downstream information of a sequence is important for a specific time-step. Consequently, we have chosen to use simple FFNNs but trained with the very powerful HFO learning algorithm. We have designed and implemented a ML method based on the

application of ensembles of FFNNs trained with the HFO learning algorithm. The output of the ensemble of FFNNs is then filtered with an SVM model. Despite the simplicity of the models which have been used in this latest experiment of ours, the results of this method in terms of accuracy and the quality of results were comparable to CNNs and LSTM-BRNN methods and much better than the methods used in *Chapters 3-4*.

If we assume that the PSSP problem can represent equally well the category of problems where both upstream and downstream information is important for processing a specific time-step of a sequence, the simple BRNN architectures can make predictions and extract useful information from this category of sequential data but might not be the best option to solve this problem. Even if we use more sophisticated learning algorithms than the state of the art BPTT, such as HR-SCG and HFO, we can slightly improve the accuracy of the network and the quality of the predicted data, but still these complicated architectures are suffering from the vanishing gradient problem which discards long range dependencies. Nevertheless, the HR-SCG and HFO second order learning algorithms can significantly decrease the convergence time of the model compared to BPTT, which is very important in ML techniques where ensemble methods, filtering techniques and network architectures are used to improve the quality of results. The drawbacks of simple BRNNs can be overcome by LSTM-BRNNs. Based on their capability to forget gates, the LSTM blocks are making the cells able to learn when and what to forget. Consequently, they can handle the vanishing gradient problem and improve the results of the PSSP problem by taking into account any long range dependencies of a protein PS. Further to this variation of BRNN architectures which is based on LSTM cells, similar results can be

retrieved from very powerful models such as CNNs trained with very simple learning algorithms (i.e., BP) or very simple models such as FFNNs trained with very powerful learning algorithms (i.e., HFO). BRNNs and CNNs are large NN architectures which sometimes take days to be trained. On the other hand FFNNs trained with the HFO learning algorithm, even if they cannot handle well the long range dependencies, they can be trained easily and in much less computational time.

In this thesis, we have managed to present Q3 accuracy results for the PSSP problem which fall in the range of 76%-81%. Through our research, we did not show any statistical significance test results because they do not contribute on the accuracy results of the PSSP problem. Nevertheless, the accuracy confidence interval in the context of the size of the datasets used is robust enough due to the relatively large size of our datasets. The CB513 dataset consists of 84104 input vectors while the PISCES dataset consists of 1886698 input vectors. Both datasets are large enough to avoid any over-fitted or biased results. The Q3 accuracy of 100% for the PSSP problem will never be achieved and should not be expected [Magnan and Baldi, 2014]. This is due to the presence of disordered regions in protein sequences, the ambiguities inherent in the definitions of SS as reflected by the imperfect correlation between several programs for determining features from PDB files, the errors and uncertainties contained in PDB database and the role of the solvent and other molecules in determining structure which are not taken into consideration by most present methods. Based on these uncertainties, a theoretical upper limit for the PSSP problem with models where the input is the MSA sequence profiles is a Q3 accuracy of 88% [Rost and Sander, 1993]. Nevertheless, according to the work of Magnan and Baldi

[2014], an acceptable accuracy when the input to a model is MSA sequence profiles is somewhere between 80%-85%. Finally, through an extensive study of related problems, which are well described in *Section 6.3*, we concluded that these inherent limitations are not the same in other sequence to structure prediction problems.

However, besides using MSA sequence profiles, there is potentially an additional way of using sequence similarity to improve the predictive accuracy of the models. The Q3 accuracy can be further improved by providing additional protein domain information to models. More specifically, if a portion of a protein sequence is similar to another sequence in the PDB, we can use the annotation of the PDB sequence to improve the results of the predictive model. This method is called sequence-based structural similarity [Magnan and Baldi, 2014]. This method works smoothly because it is well known that two domains with similar sequences will in general have similar structures [Kaczanowski and Zielenkiewicz, 2010]. The effectiveness of this method can be shown in the works of Magnan and Baldi [2014] and Wang et al. [2016], where the Q3 accuracy for the PSSP problem has been increased to approximately 93%. Obviously, these results are higher compared to the theoretical limitation of 88% Q3, which is based only on MSA encoding [Rost and Sander, 1993]. Nevertheless, because of the limitations which have been analyzed in the previous paragraph, a theoretical accuracy of 100% is still not feasible.

Nowadays there is a debate on how large a dataset needs to be for the efficient training of an ANN model. With larger datasets, the ANN generalization capabilities are increased, the over-fitting issues are decreased and rare patterns can be identified. The same theory can be inferred for the PSSP problem. As we increase the size of a dataset

then we avoid over-fitting in specific patterns which are common in a protein sequence and effectively we increase the generalization capabilities of our models. A general problem in sequential data is that patterns which are related to long range dependencies are difficult to be identified. As we increase a dataset size, these patterns become more common and can be identified and learned more accurately by a network. Finally, the protein structures in worldwide databases are daily updated. In these databases new rare patterns which are related to SS can be identified and must be used to improve an ANN's results. In this thesis, we have used the CB513 and the PISCES datasets which consist of approximately 500 and 8000 protein sequences, respectively. As we can see from our own results in *Chapters 3-4*, if we train the same methodology on the CB513 and on the PISCES datasets then the latter results are higher. This indicates the need for new larger datasets which will include new more informative MSA sequences to improve further our methodologies' results. Today only the PDB contains more than 140000 protein structures. These sequences can be used for the development of new larger datasets to train any one of our models. In the same direction, some recent works, like the one by [AlQuraishi \[2019b\]](#), have published datasets in specific structures which can be used by other researchers to train and compare their models.

A question which has been raised through this PhD work is how imbalanced all these datasets are and how such an imbalance may affect the final results of each methodology. For example, the CB513 dataset has data which correspond to 34% H, 22% E and 42% C classes. On the other hand, the PISCES dataset consists of data which correspond to 39% H, 22% E and 38% C classes. Obviously, the datasets are imbalanced; therefore, we

assume that new larger datasets would also be imbalanced. This issue might be related to the final results of this thesis. As we can see from the confusion matrices of all our methodologies (*Figures 23, 30, 35, 41, 42 and 47*), the predictive accuracy for each SS class of all our models is analogous to the quantity of each class in our datasets. This can be addressed through simple ML techniques where fragments from classes which have less data can be used during the training procedure of a model more than once, so that the models will be trained on more balanced datasets.

Finally, another question which must be answered, is the possibility of identical protein sequence fragments having different SS. This possibility would negatively affect our results. Nevertheless, this possibility is very low. Although, identical protein sequence fragments can be identified with different SS, this does not affect our results given that each one of these proteins belong to a different family of evolutionary information. Consequently, the MSA encoding profiles of all the identical protein sequence fragments are different. Furthermore, most of our models use both short and long term interactions to predict the SS of a specific fragment. Therefore, even if some short range interactions which can be identified in protein sequence fragments are identical, the corresponding long range interactions are different.

6.3 Presented approaches to other problems

Through this thesis we have chosen to analyze and evaluate our methods on the PSSP problem. In particular, we investigated the approach of multiple ML algorithms and methods on processing sequential data where both the upstream and downstream information

is important to make predictions, classification or in general extract useful information for a specific time-step on the sequence. Nevertheless, these methods can be applied on any other problem which belongs in the same class of sequential data. More specifically, relevant problems to our methods are:

1. Transmembrane Protein Topology Prediction (TMPTP): Knowledge of the structure and topology of Transmembrane (TM) proteins is important since they are involved in a wide range of important biological processes and more than half of all drugs on the market target membrane proteins [Nugent and Jones, 2009]. However, due to experimental difficulties, this class of proteins is under-represented in structural databases. Similarly to the PSSP problem, a TMPTP dataset consists of the proteins' PS and each amino acid can be assigned to a topology class: inside a cell (I), outside a cell (O) and inside a cell's membrane (T). Such a dataset has been introduced by Nugent and Jones [2009] which contained 131 sequences (TM131) with all available crystal structures, verifiable topology and N-terminal locations. As in the PSSP problem, MSA profiles can be used to represent a sequence's PS. Preliminary results for this dataset on training BRNNs with the SCG learning algorithm can be found in our work of Agathocleous et al. [2016].
2. Other problems related to Protein Sequence Processing (PSP) : Transmembrane Protein Residue Contacts [Yang and Shen, 2018], Folding Membrane Proteins [Wang et al., 2017], Signal Peptide Detection in Proteins [Savojardo et al., 2018], Prediction of Ubiquitination Sites in Proteins [Hongli et al., 2019], Signal Peptide Predictions [Armenteros et al., 2019].

3. Phoneme Speech Recognition (PSR): The speech processing and more specifically the continuous speech recognition is a very difficult problem to be solved because it refers to the asynchronous mapping of acoustic frames to sequences of linguistic symbols [Graves and Schmidhuber, 2005; Wollmer et al., 2009; Graves, 2013]. Unfortunately, NNs can learn only synchronous mapping between sequences of input-output pairs, therefore the datasets that we will use consist of sequences of acoustic frames-phonemes pairs. Those datasets can be taken from the TIMIT Acoustic-Phonetic Continuous Speech Corpus (URL: www.ldc.upenn.edu/, accessed 29/04/2019).
4. Other problems related to Natural Language Processing (NLP): Translation [Sundermeyer et al., 2014], Handwriting Recognition [Liwicki et al., 2007], Part-of-speech tagging [Plank et al., 2016], Dependency Parsing [Grella and Cangialosi, 2018] and Entity Extraction [Dernoncourt et al., 2015].

The methods which have been developed in this thesis can be directly applied or modified in order to be used on the above sequential problems. More specifically, all the methods which have been developed in *Chapters 3-5* can be directly applied to the TMPTP or any other PSPr problem. The input to each one of our methodologies would remain the MSA profile vectors of each protein sequence. The output of each one of our models would change by the desired output classes of those problem. Our models would still be trained to identify short and long range dependencies between the amino acids of the respective protein sequences. On the other hand, most of our methods can be modified in order to be able to be applied

to PSR or any other NLP problem. Particularly, the input and the output of each one of our models would be changed based on the sequential data which would be processed. Again, our models would still be trained to identify short and long range dependencies between phonemes, words or any other data which would be presented to the algorithms. In this case, the our image-like representation method and any empirical rule used as filtering technique in this thesis must be investigated and redeveloped for the latter problems.

6.4 Contributions

A list of contributions resulted from this work is presented below.

1. The proof that the upper limit of a window size which can be captured from a simple BRNN trained with the BPTT algorithm for the PSSP problem is 31 amino acids.
2. The proof that in a BRNN trained with BPTT, the local information of a window around a specific residue is sufficient, compared to a method where the whole sequence is used for each residue. The results are comparable because in the second case the information of long range dependencies is lost due to the vanishing gradient problem.
3. The introduction of the γ parameter, a modified shift operator, which in effect adds a constant weight based on the importance given to the outputs of the FRNN and BwRNN.

4. The comparison of PSSP filtering methods where it was shown that the SVM, the Logistic function and the MLP are the most suitable learning techniques to tackle this problem.
5. The MLP, which is faster than SVM, can lead to reliable filtering of PSSP predictions despite achieving slightly lower predictive accuracy than SVM.
6. Combinations of machine learning techniques and empirical smoothing rules can improve the quality of the predictions, particularly the SOV score.
7. The SCG, a second order method, is applied for the first time for training BRNNs for the PSSP problem.
8. The introduction of a hybrid learning algorithm where the GD method is used to optimize the weights of a BRNN model updated with the SCG learning algorithm.
9. The introduction of an Adaptive Step Size Scaling Parameter which combined with the SCG learning algorithm can train a simple BRNN in 75% less training time.
10. The HFO, a second order method, is applied for the first time for training BRNNs for the PSSP problem.
11. The introduction of an image-like input representation of the protein MSA profiles for the CNN networks.
12. The achievement of approximately 80.4% Q3 accuracy for the PSSP problem with CNNs.

13. The introduction of a new BESN architecture to handle bidirectional sequences. This architecture can be trained with a simple linear regression learning algorithm.
14. The application of the CW-RNN model to the PSSP problem with comparable results to BRNN but much faster training.
15. The achievement of approximately 80.4% Q3 accuracy for the PSSP problem with a novel method of simple Feed-forward Neural Networks trained with the HFO learning algorithm.

6.5 Dissemination of PhD work

The research carried out during this PhD has been widely disseminated, both by publications in refereed archival journals, compiled volumes and full conference proceedings, and by presentations in various conferences/workshops/meetings. The publications in refereed archival journals are as follows:

1. K. Charalambous, M. Agathocleous, C. Christodoulou, and V. Promponas, “Solving the Protein Secondary Structure Prediction problem with the Hessian Free Optimization algorithm”, **IEEE Access** (manuscript currently being revised after reviewers requested minor corrections).
2. M. Agathocleous, C. Christodoulou, V. Promponas, P. Kountouris, and V. Vassiliades, “A hybrid learning scheme for efficiently training bidirectional recurrent neural networks for protein secondary structure prediction”(in preparation for submission to *IEEE Transactions on Cybernetics*).

3. P. Kountouris, M. Agathocleous, V. Promponas, G. Christodoulou, S. Hadjicostas, V. Vassiliades, and C. Christodoulou, "A comparative study on filtering protein secondary structure prediction" **IEEE/ACM Transactions on Computational Biology and Bioinformatics**, vol. 9, pp. 731-739, 2012.

The publications in compiled volumes and full conference proceedings are:

1. A. Dionysiou, M. Agathocleous, C. Christodoulou, and V. Promponas, "Convolutional neural networks in combination with support vector machines for complex sequential data classification" 27th Artificial Neural Networks and Machine Learning - ICANN 2018, Lecture Notes in Computer Science, ed. by V. Kurkova, Y. Manolopoulos, B. Hammer, L. Iliadis and I. Maglogiannis, Springer-Verlag, vol. 11139, pp. 444-455, 2018.
2. M. Agathocleous, C. Christodoulou, V. Promponas, P. Kountouris, and V. Vassiliades, "Training bidirectional recurrent neural network architectures with the scaled conjugate gradient algorithm", Artificial Neural Networks and Machine Learning - ICANN 2016, Lecture Notes in Computer Science, ed. by A. E. P. Villa, P. Masulli and A. J. P. Rivero, Springer-Verlag, vol. 9886, pp. 123-131, 2016.
3. M. Agathocleous, G. Christodoulou, V. Promponas, C. Christodoulou, V. Vassiliades, and A. Antoniou, "Protein secondary structure prediction with bidirectional recurrent neural nets: can weight updating for each residue enhance performance?" In Artificial Intelligence Applications and Innovations ed. by H. Papadopoulos, A. Andreou, M. Bramer, IFIP Adv Info & Comm Tech, IFIP International Federation

for Information Processing AICT, Berlin: Springer-Verlag, vol. 339, pp. 128-137, 2010.

The presentations of this work by the candidate or the collaborator in conferences/workshops/ meetings are as follows:

1. Dimitriou, P., Agathocleous, M., Promponas, V. J. and Christodoulou, C. (2018).
Fast and accurate protein secondary structure prediction using Clockwork Recurrent Neural Networks. Proceedings of the 17th European Conference on Computational Biology, Athens, Greece, Sept. 2018, Abstract for Poster #P_Pr043.
2. Dionysiou, A., Agathocleous, M., Christodoulou, C. and Promponas, V. J. (2018).
A Hybrid Machine Learning Algorithm for Complex Sequential Data Classification, Using a Novel Data Representation Method. Proc. of the 11th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2018, p. 8.
3. Dimitriou, P., Agathocleous, M., Promponas, V. J. and Christodoulou, C. (2018).
Clockwork Recurrent Neural Networks for fast and accurate protein secondary structure prediction. Proc. of the 11th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2018, p. 16.
4. Maslioukova, M. I., Agathocleous, M., Christodoulou, C. and Promponas, V. (2017).
A novel Bidirectional Echo State Network for Protein Secondary Structure Prediction. Proc. of the 10th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2017, p. 9.

5. Agathocleous, M., Christodoulou, C., Promponas, V., Kountouris, P. and Vassiliades, V. (2016). Using Second-order learning algorithms to train Bidirectional Recurrent Neural Networks. Proceedings of the 9th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, June 2016, p. 19.
6. Agathocleous, M., Hadjicostas, S., Kountouris, P., Promponas, V., Vassiliades, V. and Christodoulou, C. (2011). Improving protein secondary structure prediction using evolutionary strategies and RBF networks. Proceedings of the 6th conference of the Hellenic Society for Computational Biology and Bioinformatics, HSCBB11, Patras, Greece, October 2011, p.34.
7. Kountouris, P., Agathocleous, M., Promponas, V., Christodoulou, G., Hadjicostas, S., Vassiliades, V. and Christodoulou, C. (2011). A comparative study on filtering protein secondary structure prediction. 19th Annual International Conference on Intelligent Systems for Molecular Biology and 10th European Conference on Computational Biology (ISMB/ECCB), Vienna, Austria, July 2011, Abstract for Poster W39.
8. Agathocleous, M., Kountouris, P., Promponas, V., Christodoulou, G., Vassiliades, V. and Christodoulou, C. (2011). Training bidirectional recurrent neural networks with Conjugate gradient-type algorithms for protein secondary structure prediction. 19th Annual International Conference on Intelligent Systems for Molecular Biology and 10th European Conference on Computational Biology (ISMB/ECCB), Vienna, Austria, July 2011, Abstract for Poster W67.

9. Kountouris, P., Agathocleous, M., Promponas, V., Christodoulou, G., Hadjicostas, S., Vassiliades, V. and Christodoulou, C. (2011). A comparative study on filtering protein secondary structure prediction. Proceedings of the 4th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2011, p. 13.
10. Agathocleous, M., Christodoulou, G., Promponas, V., Christodoulou, C., Vassiliades, V. and Antoniou, A. (2010). Per residue weight updating procedure for Protein Secondary Structure Prediction with Bidirectional Recurrent Neural Networks. Proceedings of the 3rd Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2010, p. 23.
11. Agathocleous, M. Antoniou, A., Christodoulou, C. and Promponas, V. (2009). Genetic Algorithm Optimisation of a Bidirectional Recurrent Neural Network for Protein Secondary Structure Prediction. Proceedings of the 2nd Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2009, p. 17.

The Technical Reports are:

1. Agathocleous, M., Kountouris, P., Promponas, V. and Christodoulou, C. (2011). A general Neural Network Library for the Protein Secondary Structure Prediction. Technical Report Number TR-11-10, November 2011, Department of Computer Science, University of Cyprus (25 pages).
2. Agathocleous, M., Christodoulou, C. and Promponas, V. (2009). Protein secondary structure prediction with bidirectional recurrent neural networks. Technical Report

Number TR-09-01, December 2009, Department of Computer Science, University of Cyprus (in Greek, 114 pages).

Michalis Agathocleous

Chapter 7

Future work

The primary objective of this thesis was the study of processing sequential data where both the upstream and downstream information is important to make predictions at a specific time-step of a sequence and more specifically the study of the PSSP problem. This has been achieved through ML models, learning algorithms and methodologies. This has been done through the investigation and study of well known ML algorithms which have been utilized for this class of sequential data. More specifically, we have worked with BRNNs and new novel learning algorithms for these architectures. Furthermore, we have applied new novel ML methodologies on this class of sequential data and more specifically on the PSSP problem. Based on our work, more questions and potential approaches have been raised. The next major steps that have to be done in the near future related to this thesis will include the combination of new learning algorithms and ANN models for the PSSP problem and other related problems to this class of sequential data.

One of the most important steps, which must be done in the near future, is the generalization of our methods to other related problems which belong to the same class of sequential data. More specifically, as described in *Section 6.3*, other related to the PSSP problems from the field of Bioinformatics and NLP can be processed with our ML methods and algorithms. Some of the most important problems where the BRNNs have been used are TMPTP [Nugent and Jones, 2009], PSR [Graves and Schmidhuber, 2005; Wollmer et al., 2009; Graves, 2013], Translation [Sundermeyer et al., 2014], Handwriting Recognition [Liwicki et al., 2007], Part-of-speech tagging [Plank et al., 2016], Dependency Parsing [Grella and Cangioli, 2018] and Entity Extraction [Dernoncourt et al., 2015]. Although, multiple BRNN architectures have been utilized to handle these problems, the latest developments in GD algorithms (i.e., Adam) and second order learning algorithms combined with specific ANN models or ML methodologies shown in this thesis can be used to improve the results in terms of accuracy, quality and convergence time. More specifically, training BRNNs with second order learning algorithms or CW-RNNs have the potentials to work for these problems. Moreover, architectures of ensemble methods combined with filtering techniques, stacked networks of CNNs and SVMs or simple FFNNs trained with HFO learning algorithms could show interesting results for these problems. Therefore, most of the methods which have been designed and developed in this thesis can be directly or with minor changes applied to the majority of these problems.

Based on our methods, higher results in the PSSP problem can be achieved. In *Chapter 5*, we have shown results of LSTM-BRNNs trained on our datasets. These results have a deviation of approximately 3% from the work of Heffernan et al. [2017]. This

indicates the need for new larger datasets which will include new more informative MSA sequences. These datasets will be developed based on the methods described in *Section 2.3.6* and will directly applied on the methods developed in this thesis.

As we have discussed in *Chapter 4*, the original BRNN architecture by [Baldi et al. \[1999\]](#) was trained using the BPTT learning algorithm. Although it resulted in good predictions, it suffers from problems like overfitting, slow convergence and getting stuck in local minima. Using second order optimization algorithms, additional information is used by the network which improves the optimization process in terms of both speed and accuracy. However, calculating and using the Hessian Matrix (HM) is often prohibiting due to its extremely large memory requirements. As we have seen in *Chapter 4*, the HR-SCG learning algorithm is addressing this problem by replacing the HM calculation with an approximation vector based on conjugate vectors theory. Furthermore, the HFO [[Martens and Sutskever, 2011](#)] learning algorithm [[Martens, 2010](#); [Martens and Sutskever, 2011](#)] is addressing the memory issues by replacing the HM by the product of specific arbitrary vectors which costs just as much as a gradient calculation. On the other hand, LSTM-BRNNs have reported some of the most accurate results in the PSSP problem. Hence, we have to investigate how we can accelerate the training procedure and improve the prediction accuracy of LSTM-BRNN models based on the outcome of this thesis. As we have seen, the HR-SCG and HFO learning algorithms cannot improve the results of a single BRNN but they can significantly decrease the training time needed for these architectures. Furthermore, ensemble methods and filtering techniques can improve significantly the accuracy of BRNNs. Training LSTM-BRNNs faster with the HR-SCG

or HFO learning algorithms can lead to bigger ensemble architectures and consequently to better results for problems related to sequential data. Consequently, we have to investigate the application of the HFO learning algorithm on the LSTM-BRNN architecture: we have to design and develop the HFO learning algorithm for the LSTM-BRNN architecture and then we have to extract and compare results of these models for the PSSP problem, in terms of training speed and accuracy. The same strategy can be followed for the HR-SCG learning algorithm.

Then, we have to investigate how we can improve the performance of methods which have been designed or used in this thesis on the PSSP and other problems as follows: (1) we have to use the image-like input representation of PSSP sequences in deeper CNNs and (2) we have to train the CW-RNN and BESN models on bigger datasets. After these are done, we have to investigate how reservoir computing can be used on the PSSP and other problems to improve the results in terms of accuracy and training time: (1) by applying the Conceptors on the PSSP and other problems and (2) a new activation function which can combine linear and non-linear information on the BESN model.

In *Chapter 5*, we have shown results from an image-like input representation of protein sequences which have been used to train shallow CNNs [Dionysiou et al., 2018]. This method has achieved very good results in terms of accuracy rate. Trying to extract features from complex sequential data for classification and prediction problems is an extremely difficult task. Deep ML techniques, such as CNNs, have been exclusively designed to face this class of problems. Deep structures of CNNs have successfully been applied for analyzing visual imagery [Krizhevsky et al., 2012; Rawat and Wang, 2017].

Our work in [Dionysiou et al. \[2018\]](#) has shown very good results which fall in the accepted range for this problem of > 80 and < 85 [[Magnan and Baldi, 2014](#)]. Nevertheless, to achieve these results, we have used a shallow CNN. Consequently, based on the same image-like input representation of our data, we have to design and execute deeper CNN models in an attempt to improve further our accuracy rate for the PSSP problem. Furthermore, images with bigger window size and more sophisticated CNN architectures must be designed to investigate more the capabilities of these models. Under the same scope, the latest development in deep learning has shown that Deep Residual CNNs [[He et al., 2015](#)] and Dilated CNNs [[Li et al., 2018b](#)] have shown very good results in many problems. Interestingly, we have to replace the CNNs with these models to investigate if they can improve even more the accuracy and quality of CNN results on the PSSP problem.

RC models overcome the most common problems that appear generally in RNNs. The general RC architectures and learning avoid the gradient decent training algorithms and their problems [[Jaeger and Haas, 2004](#); [Ozturk et al., 2007](#)]. The RC methods can be trained very fast with a simple linear regression learning algorithm. Thus, multiple RC models can be designed in ensemble methods to improve the accuracy rate of the PSSP problem. Conceptors [[Jaeger, 2014](#)] have been developed as an extension of the well known ESN to create filters on a reservoir of neurons to extract more informative features and consequently achieve better results on sequential problems. Furthermore, our BESN's results, which are shown in *Chapter 5*, are far from the state of the art results but there is still work to be done. Through the literature is shown that memorization skills of RC models are generally optimized in the case of reservoir units with linear activation

functions [Jaeger, 2001b]. Furthermore, using reservoir units with non-linear activation functions has the effect of degrading the memorization ability of the ESN but they are very important in practical non-linear applications of ESNs [Inubushi and Yoshimura, 2017]. Those two observations have been combined by Di Gregorio et al. [2018] in simple ESNs and have shown significant improvement in toy problems. Consequently, we have to investigate if we can improve the results of the PSSP problem with these methods by increasing the accuracy rate in very fast training time. To this effect, we could (1) apply the Conceptors on the PSSP problem, (2) apply on the BESN the hybrid activation function of Di Gregorio et al. [2018] and (3) design and develop a Bidirectional Conceptor model with the hybrid activation function for the PSSP problem.

In *Chapter 5*, we have shown that simple FFNNs trained with the powerful HFO learning algorithm can achieve results comparable to CNNs and LSTM-BRNN. These models can achieve these results based on the local information of the sequence. Interestingly, we have to investigate how can we combine these models to get information from multiple points of a protein sequence. More specifically, we can get feature vectors from FFNNs trained with the HFO learning algorithm and feed a LSTM-BRNN to combine strong local and strong long range dependencies.

Moreover, CNNs trained with BP learning algorithm and combined with SVMs have shown very good results. A combination of this methodology with the proven powerful HFO learning algorithm can improve even better the results of these models for the PSSP problem or at least decrease the high training time needed for these models.

Finally, ensemble methods can improve the performance of a learning model [Dietrich, 2000; Zhou et al., 2002, 2010] by combining multiple instances of the same or different models to improve results. In this thesis, we present results from multiple models which can capture feature of the protein sequences from different angles based on short or long range dependencies. A detailed analysis of this thesis results can show where each one of these model is stronger, i.e. short vs long range dependencies. A weighted combination of these models and their results might increase the results of the PSSP problem significantly. Consequently, we have to use a sophisticated weighted ensemble method to combine the results of the models have been used or developed in this thesis.

References

M. M. Abo-Elkhier, "Similarity/dissimilarity analysis of protein sequences using the spatial median as a descriptor," *Journal of Biophysical Chemistry*, vol. 3, pp. 142–148, 2012.

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, vol. 9, no. 1, pp. 147–169, 1985.

M. Agathocleous, G. Christodoulou, V. Promponas, C. Christodoulou, V. Vassiliades, and A. Antoniou, "Protein secondary structure prediction with bidirectional recurrent neural nets: can weight updating for each residue enhance performance?" *In Artificial Intelligence Applications and Innovations* ed. by H. Papadopoulos, A. Andreou, M. Bramer, IFIP Adv Info & Comm Tech, *IFIP International Federation for Information Processing AICT, Berlin: Springer-Verlag*, vol. 339, pp. 128–137, 2010.

M. Agathocleous, C. Christodoulou, V. Promponas, P. Kountouris, and V. Vassiliades, "Training bidirectional recurrent neural network architectures with the scaled

conjugate gradient algorithm,” *Artificial Neural Networks and Machine Learning - ICANN 2016, Lecture Notes in Computer Science*, ed. by A. E. P. Villa, P. Masulli and A. J. P. Rivero, Springer-Verlag, vol. 9886, pp. 123–131, 2016.

S. Alkatib, T. T. Fleischmann, L. B. Scharff, and R. Bock, “Evolutionary constraints on the plastid trnaset decoding methionine and isoleucine,” *Nucleic Acids Research*, vol. 40, pp. 6713–6724, 2012.

M. AlQuraishi, “End-to-end differentiable learning of protein structure,” *Cell Systems*, vol. 8, pp. 292–301, 2019a.

———, “Proteinnet: a standardized data set for machine learning of protein structure,” *BMC Bioinformatics*, vol. 20, 311, 2019b.

S. Altschul, T. Madden, A. Schäffer, A. Zhang, Z. Zhang, W. Miller, and D. Lipman, “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs,” *Nucleic Acids Res*, vol. 25, pp. 3389–3402, 1997.

N. Andrei, “Scaled conjugate gradient algorithms for unconstrained optimization,” *Computational Optimization and Applications*, vol. 38, pp. 401–416, 2007.

J. J. A. Armenteros, K. D. Tsirigos, C. K. Sønderby, T. N. Petersen, O. Winther, S. Brunak, G. Heijne, and N. H., “Signalp 5.0 improves signal peptide predictions using deep neural networks,” *Nature Biotechnology*, vol. 37, pp. 420–423, 2019.

A. F. Atiya and A. G. Parlos, "New results on Recurrent Network training: Unifying the algorithms and accelerating convergence," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 697–709, 2000.

P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri, "Exploiting the past and the future in protein secondary structure prediction," *Bioinformatics*, vol. 15, pp. 937–946, 1999.

D. Baram and A. Yonath, "From peptide-bond formation to cotranslational folding: dynamic, regulatory and evolutionary aspects," *FEBS Letter*, vol. 4, pp. 948–954, 2005.

R. Battiti, "Accelerated backpropagation learning: two optimization methods," *Complex Systems*, vol. 3, pp. 331–342, 1989.

——, "First-and second-order methods for learning: between steepest descent and newton's method," *Neural Computation*, vol. 4, pp. 141–166, 1992.

Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, 2004.

Y. Bengio, "A connectionist approach to speech recognition," *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 7, no. 4, pp. 647–667, 1993.

—, “Learning Deep Architectures for AI,” *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

Y. Bengio, A. C. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

J.-M. Blanc and P. F. Dominey, “Identification of prosodic attitudes by a temporal recurrent network,” *Cognitive Brain Research*, vol. 17, no. 3, pp. 693–699, 2003.

T. Bluche, H. Ney, and C. Kermorvant, “Feature extraction with convolutional neural networks for handwritten word recognition,” *In Proceedings of the 12th IEEE International Conference on Document Analysis and Recognition*, pp. 285–289, 2013.

S. R. Bolsover, J. S. Hyams, E. A. Shephard, H. A. White, and C. G. Wiedemann, *Cell Biology*. Hoboken, New Jersey, USA: John Wiley and Sons, 2004.

P. Borkar, M. Sarode, and L. Malik, “Employing speeded scaled conjugate gradient algorithm for multiple contiguous feature vector frames: an approach for traffic density state estimation,” *Procedia Computer Science*, vol. 78, pp. 740–747, 2016.

H. Bourlard and Y. Kamp, “Auto-association by multilayer perceptrons and singular value decomposition,” *Biological Cybernetics*, vol. 59, no. 4, pp. 291–294, 1988.

S. Burney, T. Jilani, and C. Ardil, "A comparison of first and second order training algorithms for artificial neural networks," *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 7, pp. 145–151, 2007.

A. Carnell and D. Richardson, "Linear algebra for time series of spikes," in *Proceedings of the European Symposium on Artificial Neural Networks*. Evere, Belgium, 2005, pp. 363–368.

L. Cayton, "Algorithms for manifold learning," (Technical Report No. CS2008-0923), University of California, San Diego, Tech. Rep., 2005.

B. Cetişli and A. Barkana, "Speeding up the scaled conjugate gradient algorithm and its application in neuro-fuzzy classifier training," *Soft Computing*, vol. 14, pp. 365–378, 2010.

C. C. Chang and C. J. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, p. 27, 2011.

W. Chang and M. Mak, "A conjugate gradient learning algorithm for recurrent neural networks," *Neurocomputing*, vol. 24, pp. 173–189, 1999.

C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks," *IEE Proceedings G: Circuits, Devices and Systems*, vol. 139, pp. 301–310, 1992.

J. Chen and N. Chaudhari, “Cascaded bidirectional recurrent neural networks for protein secondary structure prediction,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, pp. 572–582, 2007.

J. Cheng, A. Z. Randall, M. J. Sweredoski, and P. Baldi, “Scratch: a protein structure and structural feature prediction server,” *Nucleic Acids Research*, vol. 33, pp. W72–W76, 2005.

C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, pp. 273–297, 1995.

———, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

J. Cuff and G. Barton, “Evaluation and improvement of multiple sequence methods for protein secondary structure prediction,” *Proteins*, vol. 34, pp. 508–519, 1999.

J. Cuff, M. Clamp, A. Siddiqui, M. Finlay, and G. Barton, “JPred: a consensus secondary structure prediction server,” *Bioinformatics*, vol. 14(10), pp. 892–893, 1998.

F. Dernoncourt, J. Y. Lee, and P. Szolovits, “NeuroNER: an easy-to-use program for named-entity recognition based on neural networks,” *arXiv preprint:1705.05487*, 2015.

C. Di Gregorio, C. Gallicchio, and A. Micheli, “Combining memory and non-linearity in echo state networks.” *Artificial Neural Networks and Machine Learning - ICANN 2018*, Lecture Notes in Computer Science, ed. by V. Kurkova, Y. Manolopoulos, B. Hammer, L. Iliadis and I. Maglogiannis, Springer-Verlag, vol. 11139, pp. 556–566, 2018.

P. Diederik and L. Jimmy, “Adam: a method for stochastic optimization,” *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, Leuven, Belgium, D. Suthers, K. Verbert, E. Duval and X. Ochoa, Eds. New York, NY, USA: ACM, pp. 1–13, 2015.

T. Dietterich, “Machine learning for sequential data: A review,” *In Structural, Syntactic, and Statistical Pattern Recognition, Lecture Notes in Computer Science*, ed. by T. Caelli, Vol. 2396, Springer-Verlag, vol. 2396, pp. 15–30, 2002.

T. G. Dietterich, “Ensemble methods in machine learning,” *Proceedings of the the First International Workshop on Multiple Classifier Systems, MCS 00, London, UK*, vol. 15, pp. 1–15, 2000.

P. Dimitriou, M. Agathocleous, V. J. Promponas, and C. Christodoulou, “Fast and accurate protein secondary structure prediction using clockwork recurrent neural networks,” *Proceedings of the 17th European Conference on Computational Biology, Athens, Greece, Sept. 2018, Abstract for Poster #P_Pr043.*, 2018.

A. Dionysiou, M. Agathocleous, C. Christodoulou, and V. Promponas, "Convolutional neural networks in combination with support vector machines for complex sequential data classification," *Artificial Neural Networks and Machine Learning - ICANN 2018*, Lecture Notes in Computer Science, ed. by V. Kurkova, Y. Manolopoulos, B. Hammer, L. Iliadis and I. Maglogiannis, Springer-Verlag, vol. 11139, pp. 444–455, 2018.

P. F. Dominey and F. Ramus, "Neural network processing of natural language: sensitivity to serial, temporal and abstract structure of language in the infant," *Language and Cognitive Processes*, vol. 15, no. 1, pp. 87–127, 2000.

P. F. Dominey, M. Hoen, and T. Inui, "A neurolinguistic model of grammatical construction processing," *Journal of Cognitive Neuroscience*, vol. 18, no. 12, pp. 2088–2107, 2006.

K. Doya, "Bifurcations in the learning of Recurrent Neural Networks," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 92)*, San Diego, Calif, vol. 6. IEEE Press, 1992, pp. 2777–2780.

J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

R. Dunne and N. Campbell, "On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function,"

Proceedings of the 8th Australian Conference on Neural Networks, Melb., pp. 181–185, 1997.

J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, “Why does unsupervised pre-training help Deep Learning?” *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

C. Fang, Y. Shang, and D. Xu, “Mufold-ss: New deep inception-inside-inception networks for protein secondary structure prediction,” *Proteins*, vol. 86, pp. 592–598, 2018.

M. Fatemi, “A scaled conjugate gradient method for nonlinear unconstrained optimization,” *Optimization Methods and Software*, vol. 32, pp. 1095–1112, 2017.

P. Frasconi, M. Gori, and A. Sperduti, “A general framework for adaptive processing of data structures,” *IEEE Transactions on Neural Networks*, vol. 9, pp. 768–786, 1998.

T. Fukai and S. Tanaka, “A simple neural network exhibiting selective activation of neuronal ensembles: from winner-take-all to winners-share-all,” *Neural Computation*, vol. 9, pp. 77–97, 1997.

K.-i. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time Recurrent Neural Networks," *Neural Networks*, vol. 6, no. 6, pp. 801–806, 1993.

T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Hausler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data," *Bioinformatics*, vol. 16, pp. 906–914, 2000.

K. Gayathri and N. Kumarappan, "Double circuit ehv transmission lines fault location with rbf based support vector machine and reconstructed input scaled conjugate gradient based neural network," *International Journal of Computational Intelligence Systems*, vol. 8, pp. 95–105, 2015.

F. Gers, N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, pp. 115–143, 2002.

X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feed forward neural networks," *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy*, vol. 9, pp. 249–256, 2010.

I. Goodfellow, Q. Le, A. Saxe, H. Lee, and A. Ng, "Measuring invariances in deep networks," in *Proceedings of the Advances in Neural Information Processing Systems 22 (NIPS 2009), Vancouver, BC, Canada*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., vol. 22, 2009, pp. 646–654.

I. J. Goodfellow, A. C. Courville, and Y. Bengio, “Large-Scale Feature Learning With Spike-and-Slab Sparse Coding,” in *Proceedings of the 29th International Conference on Machine Learning (ICML 2012), Edinburgh, Scotland, UK*, J. Langford and J. Pineau, Eds. New York, NY, USA: Omnipress, July 2012, pp. 1159–1166.

P. Granitto, P. Verdes, and H. Ceccatto, “Neural network ensembles: evaluation of aggregation algorithms,” *Artificial Intelligence*, vol. 163, pp. 139–162, 2005.

———, “Neural network ensembles: evaluation of aggregation algorithms,” *Artificial Intelligence*, vol. 163, pp. 139–162, 2005.

A. Graves, “Generating sequences with recurrent neural networks,” *arXiv preprint arXiv:1308.0850*, 2013.

A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional LSTM and other neural network architectures,” *Neural Networks*, vol. 18, pp. 602–610, 2005.

A. Graves, A. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada*, pp. 6645–6649, 2013.

J. Green, M. Korenberg, and M. Aboul-Magd, “Pci-ss: Miso dynamic nonlinear protein secondary structure prediction,” *BMC Bioinformatics*, vol. 10, p. 222, 2009.

M. Grella and S. Cangialosi, “Non-Projective Dependency Parsing via Latent Heads Representation,” *arXiv preprint:1802.02116*, 2018.

R. Gupta, A. Dey, A. Vijan, and B. Gartia, “In silico structure modeling and characterization of hypothetical protein yp0045903191 present in enterobacter aerogens,” *Journal of Proteomics and Bioinformatics*, vol. 10, pp. 152–170, 2017.

S. Haeusler and W. Maass, “A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models,” *Cerebral Cortex*, vol. 17, no. 1, pp. 149–162, 2007.

K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv preprint:1604.05529*, 2015.

S. He and X. Pan, “Predicting protein secondary structure and solvent accessibility with an improved multiple linear regression method,” *Proteins*, vol. 61, pp. 473–480, 2005.

R. Heffernan, Y. Yang, K. Paliwal, and Y. Zhou, “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*, vol. 33, pp. 2842–2849, 2017.

S. Henikoff and J. Henikoff, “Amino acid substitution matrices from protein blocks.” *Proc Natl Acad Sci USA*, vol. 89, pp. 10 915–10 919, 1992.

M. Hestenes, "Conjugate direction methods in optimization." *New York: Springer-Verlag.*, 1980.

G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and Helmholtz free energy," in *Proceedings of the Advances in Neural Information Processing systems 6 (NIPS 1993), Vancouver, BC, Canada*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 1994, pp. 3–10.

G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

U. Hobohm, M. Scharf, R. Schneider, and C. Sander, "Selection of representative protein data sets." *Protein Sci*, vol. 1, pp. 409–417, 1992.

S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

F. Hongli, Y. Yingxi, W. Xiaobo, W. Hui, and X. Yan, "Deepubi: a deep learning framework for prediction of ubiquitination sites in proteins," *BMC Bioinformatics*, vol. 20, p. 85, 2019.

J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, pp. 359–366, 1989.

S. Hua and Z. Sun, “A novel method of protein secondary structure prediction with high segment overlap measure: support vector machine approach,” *Journal of Molecular Biology*, vol. 308, pp. 397–407, 2001.

M. Inubushi and K. Yoshimura, “Reservoir computing beyond memory-nonlinearity trade-off,” *Sci. Rep.*, vol. 7, p. 10199, 2017.

H. Jaeger, “Short term memory in echo state networks,” German National Research Center for Information Technology, Tech. Rep., 2001b.

——, “Controlling recurrent neural networks by conceptors,” *arXiv preprint arXiv:1403.3369v3*, 2014.

H. Jaeger and H. Haas, “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication,” *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

H. Jaeger, “The echo state approach to analysing and training recurrent neural networks—with an erratum note (technical report gmd-report 148),” German National Research Center for Information Technology, Tech. Rep., 2001.

——, “Discovering multiscale dynamical features with hierarchical echo state networks (technical report no. 9),” Jacobs University Bremen, Tech. Rep., 2007.

F. Jiang, H. Berry, and M. Schoenauer, "Supervised and Evolutionary Learning of Echo State Networks," in *Proceedings of the Parallel Problem Solving from Nature (PPSN X), Dortmund, Germany*. Springer, 2008, pp. 215–224.

E. Johansson, E. Dowla, and D. Goodman, "Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method," *International Journal of Neural Systems*, vol. 2, pp. 291–302, 1991.

B. Jones, D. Stekel, J. Rowe, and C. Fernando, "Is there a Liquid State Machine in the bacterium *escherichia coli*?" in *Proceedings of the IEEE Symposium on Artificial Life (ALIFE 07), Honolulu, Hawaii, USA*. IEEE Press, 2007, pp. 187–191.

D. Jones, "Protein secondary structure prediction based on position-specific scoring matrices," *Journal of Molecular Biology*, vol. 292, pp. 195–202, 1999.

D. Jones and M. Swindells, "Getting the most from PSI-BLAST." *Trends Biochem Sci*, vol. 27, pp. 161–164, 2002.

M. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, vol. 114, no. 5, 1986, pp. 953–956.

W. Kabsch and C. Sander, "Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers*, vol. 22, pp. 2577–2637, 1983.

S. Kaczanowski and P. Zielenkiewicz, "Why similar protein sequences encode similar three-dimensional structures?" *Theoretical Chemistry Accounts*, vol. 125, pp. 643–650, 2010.

K. Karplus, C. Barrett, M. Cline, M. Diekhans, L. Grate, and R. Hughey, "Predicting protein structure using only sequence information," *Proteins*, vol. 3, pp. 121–125, 1999.

G. Karypis, "YASSPP: Better kernels and coding schemes lead to improvements in protein secondary structure prediction," *Proteins*, vol. 64(3), pp. 575–586, 2006.

C. Khadse, M. Chaudhari, and V. Borghate, "Electromagnetic compatibility estimator using scaled conjugate gradient backpropagation based artificial neural network," *IEEE Transactions on Industrial Informatics*, vol. 13, pp. 1036–1045, 2017.

C. Kieslich, J. Smadbeck, G. Khouryand, C. Floudas, and J. Chem., "conSSert: Consensus SVM model for accurate prediction of ordered secondary structure," *Journal of Chemical Information and Modeling*, vol. 56, pp. 455–461, 2016.

P.-J. Kindermans, P. Buteneers, D. Verstraeten, and B. Schrauwen, "An uncued brain-computer interface using reservoir computing," in *Proceedings of the Workshop : machine learning for assistive technologies, Advances in Neural Information Processing Systems 23 (NIPS 2010), Vancouver, BC, Canada*. Ghent University, Department of Electronics and information systems, 2010, pp. 1–8.

D. P. Kingma and J. L. Ba, "Adam: a method for stochastic optimization," *In Proceedings of the 3rd International Conference on Learning Representations (ICLR*

2015), *Leuven, Belgium*, D. Suthers, K. Verbert, E. Duval and X. Ochoa, Eds. *New York, NY, USA: ACM*, pp. 1–13, 2015.

D. Kollias, A. Tagaris, A. Stafylopatis, S. Kollias, and G. Tagaris, “Deep neural architectures for prediction in healthcare,” *Complex and Intelligent Systems*, vol. 4, pp. 119–131, 2018.

A. N. Kolmogorov, “On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition,” in *Proceedings of the Doklady Akademii Nauk*, vol. 114, no. 5, 1957, pp. 953–956.

S. Kotsiantis, I. Zaharakis, and P. Pintelas, “Supervised machine learning: A review of classification techniques,” *Informatica*, vol. 31, pp. 249–268, 2007.

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

P. Kountouris, M. Agathocleous, V. Promponas, G. Christodoulou, S. Hadjicostas, V. Vassiliades, and C. Christodoulou, “A comparative study on filtering protein secondary structure prediction,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, pp. 731–739, 2012.

J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, “A clockwork rnnl,” *arXiv preprint arXiv:1402.3511v1*, 2014.

A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *In Advances in Neural Information Processing Systems 25: Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, Nevada. F. Pereira, C.J.C. Burges, L. Bottou and K.Q. Weinberger, Eds. Red Hook, NY: Curran Associates*, pp. 1097–1105, 2012.

H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training Deep Neural Networks," *Journal of Machine Learning Research*, vol. 10, pp. 1–40, 2009.

Q. Le, J. Ngiam, A. C. A. L. B. Prochnow, and A. Ng, "On optimization methods for deep learning," *Proceedings of the 28th International Conference on Machine Learning (ICML 2011), Bellevue, Washington, USA, L. Getoor and T. Scheffer, Eds. New York, NY, USA: Omnipress*, pp. 265–272, 2011.

Q. Le, N. Jaitly, and G. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint:1504.00941*, 2015.

S. leCessie and J. vanHouwelingen, "Ridge estimators in logistic regression." *Applied Statistics*, vol. 41, pp. 191–201, 1992.

Y. LeCun and Y. Bengio, "In the handbook of brain theory and neural networks, M. A. Arbib, Ed. Cambridge, MA: MIT Press," *Convolutional networks for images, speech, and time series*, pp. 255–258, 1998.

Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

Y. LeCun, L. B. G. Orr, and K. Müller, "Efficient backprop," *Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science*, ed. by L. Bottou G.B. Orr and K.R. Müller, Springer, vol. 1524, pp. 9–50, 1998.

R. Legenstein, D. Pecevski, and W. Maass, "A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback," *PLoS Computational Biology*, vol. 4, no. 10, p. e1000180, 2008.

K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.

C. Li and L. Yan, "Mechanical system modelling using recurrent neural networks via quasi-newton learning methods," *Applied Mathematical Modelling*, vol. 19, pp. 421–428, 1995.

H. Li, X. Wang, and S. Ding, "Research and development of neural network ensembles: a survey," *Artificial Intelligence Review*, vol. 49, pp. 455–479, 2018.

W. Li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences," *Bioinformatics*, vol. 22, pp. 1658–1659, 2006.

Y. Li, X. Zhang, and D. Chen, “CSRNet: Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes,” *arXiv preprint:1802.10062*, 2018.

B. Liebald, “Exploration of effects of different network topologies on the ESN signal crosscorrelation matrix spectrum,” *Bachelor’s Thesis, International University Bremen*, 2004.

S. Liew, M. Khalil-Hani, and R. Bakhteri, “An optimized second order stochastic learning algorithm for neural network training,” *Neurocomputing*, vol. 186, pp. 74–89, 2016.

K. Lin, V. Simossis, W. Taylor, and J. Heringa, “A simple and fast secondary structure prediction method using hidden neural networks,” *Bioinformatics*, vol. 21, pp. 152–159, 2005.

M. Liwicki, A. Graves, H. Bunke, and J. Schmidhuber, “A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks,” *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR), Curitiba, Brazil*, pp. 367–371, 2007.

M. Lukoševičius and H. Jaeger, “Reservoir computing approaches to recurrent neural network training,” *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

M. Lukoševičius, H. Jaeger, and B. Schrauwen, “Reservoir computing trends,” *KI-Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

S. Ma and C. Ji, "Fast training of Recurrent Networks based on the EM algorithm," *IEEE Transactions on Neural Networks*, vol. 9, no. 1, pp. 11–26, 1998.

W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

W. Maass, T. Natschläger, and H. Markram, "A model for real-time computation in generic neural microcircuits," *Proceedings of the Advances in Neural Information Processing Systems 15 (NIPS 2002)*, Vancouver, BC, Canada, ed. Sebastian Thrun and Lawrence Saul and Bernhard Schölkopf, pp. 213–220, 2003.

W. Maass, T. Natschläger, and H. Markram, "Computational models for generic cortical microcircuits," in *Computational Neuroscience: A comprehensive approach*, J. Feng, Ed. London, UK: Chapman & Hall/CRC Press, 2004, pp. 575–605.

W. Maass, P. Joshi, and E. D. Sontag, "Principles of real-time computing with feedback applied to cortical microcircuit models," in *Proceedings of the Advances in Neural Information Processing Systems 18 (NIPS 2005)*, Vancouver, BC, Canada,

Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, pp. 835–842.

C. N. Magnan and P. Baldi, "Ssproaccpro 5: almost perfect prediction of protein secondary structure and relative solvent accessibility using pro files, machine learning and structural similarity," *Bioinformatics*, vol. 30, pp. 2592–2597, 2014.

D. Marquardt, "An algorithm for the least-squares estimation of nonlinear parameters," *SIAM Journal of Applied Mathematics*, vol. 11, pp. 431–441, 1963.

J. Martens, "Deep learning via hessian-free optimization," *Proceedings of the 27th Int. Conf. on Machine Learning, Haifa, Israel*, pp. 735–742, 2010.

J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the the 28th International Conference on Machine Learning (ICML 2011), Bellevue, Washington, USA*, ser. ICML 2011, L. Getoor and T. Scheffer, Eds., vol. 46. Omnipress, 2011, pp. 1033–104.

B. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme." *Biochim Biophys Acta*, vol. 404, pp. 442–451, 1975.

W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

L. Medsker and L. Jain, *Recurrent neural networks: Design and applications*, ser. International Series on Computational Intelligence, Taylor and Francis, Eds. London: CRC Press, Inc. Boca Raton, FL, USA 1999, 2010.

D. Meyer and F. Wien, "Support vector machines," *R Newss*, vol. 1, pp. 23–26, 2001.

M. F. Møller, “Exact calculation of the product of the hessian matrix of feed-forward network error functions and a vector in $o(n)$ time,” *Technical Report PB-432, Computer Science Department, Aarhus University, Denmark*, 1993a.

——, “A Scaled Conjugate Gradient algorithm for fast supervised learning,” *Neural Networks*, vol. 6, no. 4, pp. 525–533, 1993b.

S. Montgomerie, S. Sundararaj, W. Gallin, and D. Wishart, “Improving the accuracy of protein secondary structure prediction using structural alignment,” *BMC Bioinformatics*, vol. 7, p. 301, 2006.

C. Mooney and G. Pollastri, “Beyond the twilight zone: automated prediction of structural properties of proteins by recursive neural networks and remote homology information,” *Proteins*, vol. 77, pp. 181–190, 2009.

A. Murzin, S. Brenner, T. Hubbard, and C. Chothia, “SCOP: a structural classification of proteins database for the investigation of sequences and structures.” *J Mol Biol*, vol. 14, pp. 536–540, 1995.

S. G. Nash, “Newton-type minimization via the lanczos method,” *SIAM Journal on Numerical Analysis*, vol. 21, pp. 770–788, 1984.

——, “A survey of truncated-newton methods,” *Journal of Computational and Applied Mathematics*, vol. 124, pp. 45–59, 2000.

D. Nikolić, S. Haeusler, W. Singer, and W. Maass, “Temporal dynamics of information content carried by neurons in the primary visual cortex,” in *Proceedings of*

the Advances in Neural Information Processing Systems 19 (NIPS 2006), Vancouver, BC, Canada, B. Schölkopf, J. Platt, and T. Hoffman, Eds. Cambridge, MA: MIT Press, 2007, pp. 1041–1048.

J. Nocedal and S. Wright, *Numerical Optimization*. New York, NY: Springer, 2006.

T. Nugent and D. Jones, “Transmembrane protein topology prediction using support vector machines,” *BMC Bioinformatics*, vol. 10, 159, 2009.

B. A. Olshausen, “Emergence of simple-cell receptive field properties by learning a sparse code for natural images,” *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

M. Ozturk, D. Xu, and J. Príncipe, “Analysis and design of echo state networks,” *Neural Computation*, vol. 19, no. 1, pp. 111–138, 2007.

A. D. PalÁz, A. Dovier, and F. Fogolari, “Constraint logic programming approach to protein structure prediction,” *BMC Bioinformatics*, vol. 5, 2004.

X. Pan, “Multiple linear regression for protein secondary structure prediction,” *Proteins*, vol. 43, pp. 256–259, 2001.

J. Park and I. W. Sandberg, “Universal approximation using radial-basis-function networks,” *Neural Computation*, vol. 3, pp. 246–257, 1991.

B. A. Pearlmutter, “Fast exact multiplication by the hessian,” *Neural Computation*, vol. 6, pp. 147–160, 1994.

C.-C. Peng and G. D. Magoulas, "Sequence Processing with Recurrent Neural Networks," *Encyclopedia of Artificial Intelligence*, pp. 1411–1417, 2008.

B. Plank, A. Søgaard, and Y. Goldberg, "Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss," *arXiv preprint:1604.05529*, 2016.

G. . Pollastri and A. McLysaght, "Porter: a new, accurate server for protein secondary structure prediction," *Bioinformatics*, vol. 21, pp. 1719–1720, 2005.

G. Pollastri, D. Przybylski, B. Rost, and P. Baldi, "Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles," *Proteins*, vol. 47, pp. 228–235, 2002.

G. V. Puskorius and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 279–297, 1994.

N. Qian and T. Sejnowski, "Predicting the secondary structure of globular proteins using neural network models." *Journal of Molecular Biology*, vol. 202, pp. 865–884, 1988.

W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, pp. 2352–2449, 2017.

F. Richards and C. Kundrot, "Identification of structural motifs from protein coordinate data: secondary structure and first-level supersecondary structure," *Proteins*, vol. 3, pp. 71–84, 1988.

M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm," *Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, USA*, pp. 586–591, 1993.

B. Robitaille, B. Marcos, M. Veillette, and G. Payre, "Quasi-newton methods for training neural networks," *WIT Transactions on Information and Communication Technologies*, vol. 2, pp. 323–335, 1993.

J. W. Ronald and D. Zipser, "Experimental analysis of the Real-Time Recurrent Learning Algorithm," *Connection Science*, vol. 1, no. 1, pp. 87–111, 1989.

B. Rost, "PHD: Predicting one-dimensional protein structure by profile-based neural networks," *Methods in Enzymology*, vol. 266, pp. 525–539, 1996.

—, "Review: protein secondary structure prediction continues to rise." *J Struct Biol*, vol. 134, pp. 204–218, 2001.

B. Rost and V. Eyrich, "EVA: large-scale analysis of secondary structure prediction." *Proteins*, vol. 5, pp. 409–417, 2001.

B. Rost and C. Sander, "Improved prediction of protein secondary structure by use of sequence profiles and neural networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 16, pp. 7558–7562, 1993.

——, “Combining evolutionary information and neural networks to predict protein secondary structure,” *Protein*, vol. 19, pp. 55–72, 1994.

B. Rost, C. Sander, and R. Schneider, “Redefining the goals of protein secondary structure prediction,” *Journal of Molecular Biology*, vol. 235, pp. 13–26, 1994.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986a.

D. Rumelhart, G. Hinton, and R. Williams, “Learning internal representation by error propagation,” *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed. by In Rumelhart D. E. and McClelland J. L., MIT Press, Cambridge, MA, vol. 1, pp. 318–362, 1986b.

R. Salakhutdinov and G. E. Hinton, “Deep Boltzmann Machines,” in *Proceedings of the the 12th International Conference on Artificial Intelligence and Statistics (AISTATS 09), Clearwater, Florida USA*, vol. 5, no. 2. MIT Press Cambridge, MA, 2009, pp. 448–455.

A. Salamov and V. Solovyev, “Prediction of protein secondary structure by combining nearest-neighbor algorithms and multiple sequence alignments,” *Journal of Molecular Biology*, vol. 247, pp. 11–15, 1995.

C. Savojardo, P. L. Martelli, P. Fariselli, and R. Casadio, “Deepsig: deep learning improves signal peptide detection in proteins.” *Bioinformatics*, vol. 34, pp. 1690–1696, 2018.

F. Scarselli and A. Tsoi, "Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results," *Neural Networks*, vol. 11, pp. 15–37, 1998.

J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 65, pp. 85–117, 2015.

J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, "Training recurrent networks by evolution," *Neural Computation*, vol. 19, no. 3, pp. 757–779, 2007.

R. Schneider and C. Sander, "The hssp database of protein structure-sequence alignments," *Nucleic Acids Research*, vol. 24, pp. 201–205, 1996.

N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural Computation*, vol. 14, pp. 1723–1738, 2002.

B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt, "Improving reservoirs using intrinsic plasticity," *Neurocomputing*, vol. 71, no. 7, pp. 1159–1171, 2008.

F. Schürmann, K. Meier, and J. Schemmel, "Edge of chaos computation in mixed-mode vlsi - a hard liquid," in *Proceedings of the Advances in Neural Information Processing Systems 17 (NIPS 2004)*, Vancouver, BC, Canada, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 1201–1208.

M. Schuster and K. Paliwal, "Bidirectional Recurrent Neural Networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

T. Schwede, "Protein modeling: what happened to the "protein structure gap"?" *Structure*, vol. 21, pp. 1531–1540, 2013.

A. Senior, G. Heigold, M. Ranzato, and K. Yang, "An empirical study of learning rates in deep neural networks for speech recognition," *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Vancouver, BC, Canada*, pp. 6724–6728, 2013.

A. Shepherd, D. Gorse, and J. Thornton, "Prediction of the location and type of β -turns in proteins using neural networks." *J Struct Biol*, vol. 134, pp. 204–218, 2001.

K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. Kruthiventi, and R. V. Babu, "A taxonomy of deep convolutional neural nets for computer vision," *Frontiers in Robotics and AI*, vol. 36, p. 2, 2016.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

J. J. Steil, "Memory in Backpropagation-Decorrelation O(N) Efficient Online Recurrent Learning," in *Proceedings of the 15th International Conference on Artificial Neural Networks (ICANN 2005), Warsaw, Poland*, W. Duch, J. Kacprzyk, E. Oja, and S. Zadrozny, Eds., vol. 3697. Springer, 2005, pp. 649–654.

M. Sundermeyer, T. Alkhoul, J. Wuebker, and H. Ney, "Translation modeling with bidirectional recurrent neural networks," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar*, pp. 14–25, 2014.

D. Sussillo, P. Nuyujukian, J. M. Fan, J. C. Kao, S. D. Stavisky, S. Ryu, and K. Shenoy, "A recurrent neural network for closed-loop intracortical brain-machine interface decoders," *Journal of Neural Engineering*, vol. 9, no. 2, p. 026027, 2012.

I. Sutskever, G. E. Hinton, and G. Taylor, "The recurrent temporal restricted Boltzmann Machine," in *Proceedings of the Advances in Neural Information Processing Systems 21 (NIPS 2008), Vancouver, BC, Canada*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., vol. 21, 2009, pp. 1601–1608.

Y. Tang, "Deep learning using linear support vector machines." *arXiv preprint arXiv:1306.0239*, 2013.

T. Tieleman and G. Hinton, "Lecture 6.5 - rmsprop, divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, pp. 26–31, 2012.

A. C. Tsoi, "Recurrent Neural Network architectures: An overview," in *Proceedings of the Sequences and Data Structures, International Summer School on Neural Networks*, E.R. Caianiello, Vietri sul Mare, Salerno, Italy, C. L. Giles and M. Gori, Eds., vol. 1387. Springer, 1998, pp. 1–26.

K. Vandoorne, W. Dierckx, B. Schrauwen, D. Verstraeten, R. Baets, P. Bienstman, and J. Van Campenhout, "Toward optical signal processing using photonic reservoir computing," *Optics Express*, vol. 16, no. 15, pp. 11 182–11 192, 2008.

V. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, pp. 988–999, 1999.

D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated word recognition with the Liquid State Machine: a case study," *Information Processing Letters*, vol. 95, no. 6, pp. 521–528, 2005.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a Deep Network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." *IEEE Trans Inf Theory*, vol. 13, pp. 260–269, 1967.

A. Waibel, "Modular construction of time-delay neural networks for speech recognition," *Neural Computation*, vol. 1, no. 1, pp. 39–46, 1989.

I. Wallace, G. Blackshields, and D. Higgins, "Multiple sequence alignment," *Current Opinion in Structural Biology*, vol. 15, pp. 261–266, 2005.

G. Wang and R. Dunbrack Jr, "Pisces: a protein sequence culling server," *Bioinformatics*, vol. 19, pp. 1589–1591, 2003.

S. Wang, J. Peng, J. Ma, and J. Xu, "Protein secondary structure prediction using deep convolutional neural fields," *Scientific Reports*, 18962, vol. 6, 2016.

S. Wang, Z. Li, Y. Yu, and J. Xu, "Folding membrane proteins by deep transfer learning," *Cell Systems*, vol. 5, pp. 202–211, 2017.

X. Wang and Y. Huang, "Convergence study in extended kalman filter-based training of recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 22, pp. 588–600, 2011.

N. Weiskopf, K. Mathiak, S. W. Bock, F. Scharnowski, R. Veit, W. Grodd, R. Goebel, and N. Birbaumer, "Principles of a brain-computer interface (BCI) based on real-time functional magnetic resonance imaging (fMRI)," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 966–970, 2004.

R. Wengert, "A simple automatic derivative evaluation program," *Communications of the ACM*, vol. 7, pp. 463–464, 1964.

P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," PhD thesis, Harvard University, Tech. Rep., 1974.

P. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 10, pp. 1550 – 1560, 1990.

J. Weston, F. Ratle, and R. Collobert, "Deep learning via semi-supervised embedding," in *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, Helsinki, Finland, ser. ICML 2008, W. W. Cohen, A. McCallum, and

S. T. Roweis, Eds., vol. 307. ACM International Conference Proceeding Series, 2008, pp. 1168–1175.

B. Widrow and M. E. Hoff, “Adaptive switching circuits,” *Western Electronic Show and Convention: Convention record*, vol. 4, pp. 96–104, 1960.

R. Williams, “Training recurrent networks using the extended kalman filter,” *Proc. of International Joint Conference on Neural Networks, Baltimore, MD*, vol. 4, pp. 241–246, 1992.

R. J. Williams and D. Zipser, “Experimental analysis of the real-time recurrent learning algorithm,” *Connection Science*, vol. 1, no. 1, pp. 87–111, 1989.

I. Witten and E. Frank, “Data mining: Practical machine learning tools and techniques, second edition,” *Morgan Kaufmann*, 2005.

M. Wollmer, F. Eyben, J. Keshet, A. Graves, B. Schuller, and G. Rigoll, “Robust discriminative keyword spotting for emotionally colored spontaneous speech using bidirectional LSTM networks,” *Proceedings of the IEEE International Conference on Acoustic, Speech, and Signal Processing, Taipei, Taiwan*, pp. 740–747, 2009.

M. Wood and J. Hirst, “Protein secondary structure prediction with dihedral angles,” *Proteins*, vol. 59, pp. 476–481, 2005.

K. Wu, H. Lin, J. Chang, T. Sung, and W. Hsu, “HYPROSP: A hybrid protein secondary structure prediction algorithm—a knowledge-based approach,” *Nucleic Acids Res*, vol. 32(17), pp. 5059–5065, 2004.

X. Wu, V. Kumar, J. RossQuinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, Z. Zhou, M. Steinbach, D. Hand, and D. Steinberg, "Top 10 algorithms in data mining." *Knowl Inf Syst*, vol. 14, pp. 1–37, 2007.

J. Yang and H. Shen, "Membrain-contact 2.0: A new two-stage machine learning model for the prediction enhancement of transmembrane protein residue contacts in the full chain." *Bioinformatics*, vol. 34, pp. 230–238, 2018.

Y. Yang, J. Gao, J. Wang, R. Heffernan, J. Hanson, K. Paliwal, and Y. Zhou, "Sixty-five years of the long march in protein secondary structure prediction: the final stretch?" *Briefings in Bioinformatics*, vol. 19, pp. 482–494, 2016.

K. Yue and K. Dill, "Constraint-based assembly of tertiary protein structures from secondary structure elements," *Protein Science*, vol. 9, pp. 1935–1946, 2000.

M. D. Zeiler, "Adadelata: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

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

H. Zhang, T. Zhang, K. Chen, K. Kedarisettiand, M. Mizianty, Q. Bao, W. Stach, and L. Kurgan, "Critical assessment of high-throughput standalone methods for secondary structure prediction," *Brief Bioinform*, vol. 12(6), pp. 672–688, 2011.

J. Zheng, X. Cao, B. Zhang, X. Zhen, and X. Su, “Deep ensemble machine for video classification,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, pp. 1723–1738, 2019.

Z. Zhou, J. Wu, and W. Tang, “Ensembling neural networks: many could be better than all,” *Artificial Intelligence*, vol. 137, pp. 239–263, 2002.

———, “Corrigendum: Corrigendum to ensembling neural networks: Many could be better than all [Artificial Intelligence 137 (1-2)(2002) 239-263],” *Artificial Intelligence*, vol. 174, p. 1570, 2010.

Appendix A

Publications during PhD work

Articles in Refereed Archival Journals

1. K. Charalambous, M. Agathocleous, C. Christodoulou, V. Promponas, “Solving the Protein Secondary Structure Prediction problem with the Hessian Free Optimization algorithm”, **IEEE Access** (manuscript currently being revised after reviewers requested minor corrections).
2. M. Agathocleous, C. Christodoulou, V. Promponas, P. Kountouris, and V. Vassiliades, “A hybrid learning scheme for efficiently training bidirectional recurrent neural networks for protein secondary structure prediction”(in preparation for submission to *IEEE Transactions on Cybernetics*).

3. P. Kountouris, M. Agathocleous, V. Promponas, G. Christodoulou, S. Hadjicostas, V. Vassiliades, and C. Christodoulou, "A comparative study on filtering protein secondary structure prediction" *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, pp. 731-739, 2012.
4. M. Krambia-Kapardis, C. Christodoulou and M. Agathocleous, "Neural Networks: The panacea in fraud detection?" ***Managerial Auditing Journal***, vol. 25, pp. 659-678, 2010.

Full Conference Papers

1. A. Dionysiou, M. Agathocleous, C. Christodoulou, and V. Promponas, "Convolutional neural networks in combination with support vector machines for complex sequential data classification" 27th Artificial Neural Networks and Machine Learning - ICANN 2018, Lecture Notes in Computer Science, ed. by V. Kurkova, Y. Manolopoulos, B. Hammer, L. Iliadis and I. Maglogiannis, Springer-Verlag, vol. 11139, pp. 444-455, 2018.
2. M. Agathocleous, C. Christodoulou, V. Promponas, P. Kountouris, and V. Vassiliades, "Training bidirectional recurrent neural network architectures with the scaled conjugate gradient algorithm", Artificial Neural Networks and Machine Learning - ICANN 2016, Lecture Notes in Computer Science, ed. by A. E. P. Villa, P. Masulli and A. J. P. Rivero, Springer-Verlag, vol. 9886, pp. 123-131, 2016.
3. M. Agathocleous, G. Christodoulou, V. Promponas, C. Christodoulou, V. Vassiliades, and A. Antoniou, "Protein secondary structure prediction with bidirectional recurrent neural nets: can weight updating for each residue enhance performance?" In Artificial Intelligence Applications and Innovations ed. by H. Papadopoulos, A. Andreou, M. Bramer, IFIP Adv Info & Comm Tech, IFIP International Federation for Information Processing AICT, Berlin: Springer-Verlag, vol. 339, pp. 128-137, 2010.

Refereed Short Papers and Abstracts

1. Dimitriou, P., Agathocleous, M., Promponas, V. J. and Christodoulou, C. (2018). Fast and accurate protein secondary structure prediction using Clockwork Recurrent Neural Networks. Proceedings of the 17th European Conference on Computational Biology, Athens, Greece, Sept. 2018, Abstract for Poster #P_Pr043.
2. Dionysiou, A., Agathocleous, M., Christodoulou, C. and Promponas, V. J. (2018). A Hybrid Machine Learning Algorithm for Complex Sequential Data Classification, Using a Novel Data Representation Method. Proc. of the 11th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2018, p. 8.
3. Dimitriou, P., Agathocleous, M., Promponas, V. J. and Christodoulou, C. (2018). Clockwork Recurrent Neural Networks for fast and accurate protein secondary structure prediction. Proc. of the 11th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2018, p. 16.
4. Maslioukova, M. I., Agathocleous, M., Christodoulou, C. and Promponas, V. (2017). A novel Bidirectional Echo State Network for Protein Secondary Structure Prediction. Proc. of the 10th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2017, p. 9.
5. Agathocleous, M., Christodoulou, C., Promponas, V., Kountouris, P. and Vassiliades, V. (2016). Using Second-order learning algorithms to train Bidirectional

Recurrent Neural Networks. Proceedings of the 9th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, June 2016, p. 19.

6. Agathocleous, M., Hadjicostas, S., Kountouris, P., Promponas, V., Vassiliades, V. and Christodoulou, C. (2011). Improving protein secondary structure prediction using evolutionary strategies and RBF networks. Proceedings of the 6th conference of the Hellenic Society for Computational Biology and Bioinformatics, HSCBB11, Patras, Greece, October 2011, p.34.
7. Kountouris, P., Agathocleous, M., Promponas, V., Christodoulou, G., Hadjicostas, S., Vassiliades, V. and Christodoulou, C. (2011). A comparative study on filtering protein secondary structure prediction. 19th Annual International Conference on Intelligent Systems for Molecular Biology and 10th European Conference on Computational Biology (ISMB/ECCB), Vienna, Austria, July 2011, Abstract for Poster W39.
8. Agathocleous, M., Kountouris, P., Promponas, V., Christodoulou, G., Vassiliades, V. and Christodoulou, C. (2011). Training bidirectional recurrent neural networks with Conjugate gradient-type algorithms for protein secondary structure prediction. 19th Annual International Conference on Intelligent Systems for Molecular Biology and 10th European Conference on Computational Biology (ISMB/ECCB), Vienna, Austria, July 2011, Abstract for Poster W67.
9. Kountouris, P., Agathocleous, M., Promponas, V., Christodoulou, G., Hadjicostas, S., Vassiliades, V. and Christodoulou, C. (2011). A comparative study on filtering

protein secondary structure prediction. Proceedings of the 4th Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2011, p. 13.

10. Agathocleous, M., Christodoulou, G., Promponas, V., Christodoulou, C., Vassiliades, V. and Antoniou, A. (2010). Per residue weight updating procedure for Protein Secondary Structure Prediction with Bidirectional Recurrent Neural Networks. Proceedings of the 3rd Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2010, p. 23.
11. Agathocleous, M. Antoniou, A., Christodoulou, C. and Promponas, V. (2009). Genetic Algorithm Optimisation of a Bidirectional Recurrent Neural Network for Protein Secondary Structure Prediction. Proceedings of the 2nd Cyprus Workshop on Signal Processing and Informatics, Nicosia, Cyprus, July 2009, p. 17.

Technical Reports

1. Agathocleous, M., Kountouris, P., Promponas, V. and Christodoulou, C. (2011). A general Neural Network Library for the Protein Secondary Structure Prediction. Technical Report Number TR-11-10, November 2011, Department of Computer Science, University of Cyprus (25 pages).
2. Agathocleous, M., Christodoulou, C. and Promponas, V. (2009). Protein secondary structure prediction with bidirectional recurrent neural networks. Technical Report Number TR-09-01, December 2009, Department of Computer Science, University of Cyprus (in Greek, 114 pages).