GalvanizeU Capstone Project

Prediction of Protein Secondary Structure

Lucas Ramadan
Masters Candidate, Data Science
GalvanizeU & The University of New Haven
San Francisco, CA 94105
LucasRamadan@gmail.com

Mike Bowles
Capstone Advisor
GalvanizeU
San Francisco, CA 94105
mike@mbowles.com

***Abstract* —** In this work, modern Machine Learning techniques were applied to the existing Protein Data Bank (PDB), in an attempt to improve protein secondary structure prediction accuracy. Neural Networks were selected as the most likely model to succeed, based on their proclivity to sequence data. Many different architectures and activations were examined, as well as regularizers and normalizers, including Recurrent and Convolutional layers. The best model produced thus far currently out-performs the state-of-the-art, achieving testing prediction accuracies of up to 90%.

Keywords — secondary structure prediction, neural networks

I. Motivation

The prediction of protein secondary structure has been long of interest, due to its importance in medical applications, such as Drug Design. The principle is relatively intuitive: In order to fix a problem, one must understand the problem itself.

Suppose there is a new disease that has entered the world. A microorganism produces a certain protein, which enters human cells and wreaks havoc upon its host. If such a virulent protein could be modeled, it's functioning could then be ascertained. Eventually, small molecules could be designed to target the proteins' modeled active sites, and potentially render it inactive in human cells.

Further impetus for the prediction of Protein Secondary Structure, is that it has lucrative potential: Pharmaceuticals in the United States alone are a 100+ billion dollar industry. This industry directly relies on the modeling of target proteins for designing their drugs. Thus the study of protein secondary structure is of both academic and economic interest.

II. Overview

A. Introduction

A remarkable amount of human effort has been spent attempting to predict protein structure. Since as far back as the 1960s[1-4] and into the 1970s[5-7], biochemists and biophysicists have been designing methodologies to predict protein secondary structure, with techniques such as X-Ray Crystallography (XRD) and Nuclear Magnetic Resonance (NMR). As the resolution and power of these scientific techniques has increased over time, the art's understanding of the relationship between a protein's amino acid sequence and its structure has greatly improved.

In an effort to consolidate the work of the industry, the Protein Data Bank (PDB) was created in 1971[8], which now holds over 100,000 characterized proteins. These data are found in a variety of formats, the most common being the PDB format, which encode the three-dimensional atomic coordinates of the protein in space. These data come primarily from XRD and NMR analyses.

Following the growing popularity of the PDB, in 1983[9], Sander and Kabsch et al. released a Pascal program, called DSSP, which produced relevant protein features, given atomic-resolution coordinates of a protein. The basis of the program is an energetic calculation, which compares bond angles between hydrogen atoms in the protein's backbone. These features include the amino acid sequence itself, the secondary structure, as well as other physical or chemical properties, such as bond angles, solubility calculations, and more. An alternative to DSSP was made in 1995[10], called STRIDE, which built upon DSSPs hydrogen-bond calculations and took into account more complex structural and electronic potentials, producing more accurate depictions of protein structure. Most recently, in 2002[11] an improvement was added on top of DSSP, which adds more complexity to the classes predicted by the original DSSP program.

B. Publishing

The number of articles published on the topic of "Protein Secondary Structure" has been increasing exponentially since the 1960s, yet has plateaued in recent years and is now slowly decreasing.

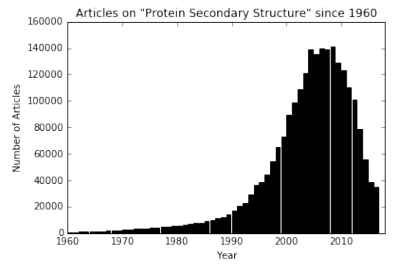

Fig. 1. Articles published on "Protein Secondary Structure" since 1960[X]

Yet interestingly enough, the number of articles about bioinformatics that mentions machine learning has been exponentially increasing (data runs out in 2010).

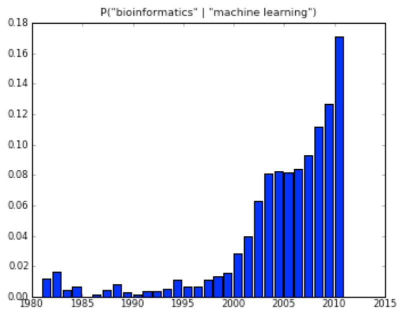

Fig. 2. Publishings on "Bioinformatics" & "Machine Learning" over time

This could suggest that while traditional bioinformatic approaches to Protein Secondary Structure prediction have been slowing down in recent years, the growing power and importance of Machine Learning, as a discipline, has been continuing to fuel study.

C. *Accuracies*

Early work into Protein Secondary Structure prediction yielded low accuracies, between 50-60%, relative to those achieved in modern Machine Learning settings. With the existence of the PDB and the release of DSSP and STRIDE, availability and automatic tagging of data allowed accuracies to increase into the 70-80% range[12]. However, as with most Machine Learning tasks, every additional percentage of accuracy is exponentially harder to obtain, and thus accuracies have stalled in the 80%s. One such paper suggests an upper limit of 90% in accuracy[13].

A number of Machine Learning approaches to this problem exist today, employing techniques such as Support Vector Machines (SVMs)[14], Feed-Forward Neural Networks[15-16] and Bayesian methods[17-18]. Yet currently, there do not appear to be many Recurrent or Convolutional Neural Networks utilized for this task, thus laying the foundation for this Capstone paper.

III. Theory

A. *Neural Network Applicability*

Of the numerous Machine Learning approaches that could be applied to the prediction of protein secondary structure, Neural Networks were selected as the most likely model to succeed, based on their effectiveness with sequence data. Furthermore, the ability to add recurrency to the network allows a form of the previous fold-state of the protein to help predict the next fold-state of the protein. This is at the core of a protein's structure, as there are often large continuous sections of amino acids all in the same fold-state. A Hidden Markov Model (HMM) is also an effective way to model a protein, as you can capture these continuous sections using a state-transition matrix, where is it most likely that the next state remains in the same current state. However, HMMs were not included in this work, as they were outside of the scope of the project.

Neural Networks are also a relevant choice for this project, because the more data they are trained on, usually the better they perform. The dataset of the entire Protein Data Bank contains over 100,000 proteins, each of which is a sequence of amino acids with an average length of 1,360 residues, which provides ample data for training.

At its core, Neural Networks rely on a series of matrix multiplications and non-linear functions, known as activations, to produce predictions. The inputs to the network are multiplied by a matrix (or tensor) of float values, known as weights, and then activated, and passed to the next layer of the network. An illustration of the process can be seen below in the following equations.

First, suppose the activation function of the network is the quintessential sigmoid function:

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

And we have our data input (X) and input weight (W) matrices below:

$$X = \begin{bmatrix} 0 & 2 \\ 1 & 1 \end{bmatrix}, \qquad W = \begin{bmatrix} 0.2 & 0.5 \\ 0.3 & 0.8 \end{bmatrix} \tag{2}$$

We then multiply X by W, and produce the input to the activation function σ:

$$XW = \begin{bmatrix} 0 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0.2 & 0.5 \\ 0.3 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.6 & 1.6 \\ 0.5 & 1.3 \end{bmatrix} \tag{3}$$

We then apply the sigmoid function element-wise, to every entry in the new input matrix:

$$\sigma(WX) = \begin{bmatrix} \dfrac{1}{1+e^{-0.6}} & \dfrac{1}{1+e^{-1.6}} \\ \dfrac{1}{1+e^{-0.5}} & \dfrac{1}{1+e^{-1.3}} \end{bmatrix} \tag{4}$$

It is worth mentioning that this is a purely linear transformation, and most Neural Networks add an affine transformation, by adding a bias term, which has been excluded for simplicity of notation. It is performed as a simple element-wise addition.

The post-activation matrix is the output of the first layer of our network, which is then fed into subsequent layers of the network, following the same pattern of matrix multiplication and activation, until the output layer is reached, where the predictions are finally produced. Recently, it has been proposed that one should scale the input to the activation, which is a process known as Batch Normalization (BN). Other activation functions utilized in this project were softmax, tanh, ReLU, ELU, PReLU and SReLU.

A convolutional layer applies a matrix of weights across the input, sliding it along like a window. Thus a convolution can incorporate some contextual information about the observation, and is often used in Image Analysis. An example of a convolutional matrix is below in Figure 3:

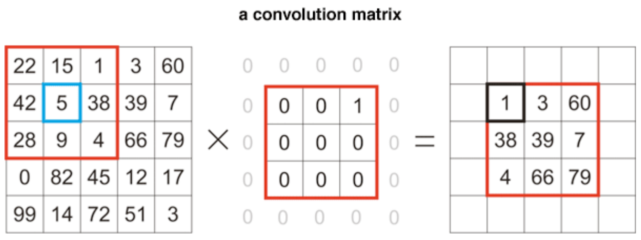

Fig. 3. An example of applying a convolution matrix to data

In order to perform convolution on the data, each amino acid of the protein needed to be encoded like an image, so that the convolution could be applied correctly. Implementations of this encoding are described later.

Adding recurrency to a layer in a Neural Network merely means feeding in the previous output of the layer as part of the input to the layer at the next timestep. This can be written as below:

$$h_t = \sigma(W_x X_t + W_h h_{t\text{-}1}) \tag{5}$$

As you progress through each timestep, or observation in your data, the current hidden state incorporates more and more past hidden states. As a result of the multiple matrix multiplication operations, the influence of long in the past observations becomes more and more diminished, eventually to zero. Many improvements have been offered to the simple recurrent layer, including Long-Short Term Memory (LSTM) and Gated-Recurrent Unit (GRU) layers, which were experimented with in the project.

As with any Machine Learning methodology, a Cost function needs to be defined in order to run any intelligent optimization. Common cost functions for Neural Networks are Sum Squared Error (SSE), Mean Squared Error (MSE) and Binary/Categorical Cross Entropy (BCE, CCE). The equation for MSE has been included below:

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(y\text{-}\hat{y})^2 \tag{6}$$

Where $\hat{y}$ is the final output of the network.

B. *Gradient Descent*

The true power of Neural Networks is the ability to utilize Gradient Descent to converge upon optima. This will not guarantee capturing the global optimum solution, but it does allow for us to move towards optimum solutions in an educated way. Gradient Descent is implemented with Neural Networks in a process known as Backpropagation.

In backpropagation, the partial derivative of each weight in the respective weight matrix is calculated with respect to the Cost function (J), in order to produce an effective incrementation of the weight. The calculation of each partial derivative results in a matrix, known as the gradient. A calculation for a single weight would look like:

$$\frac{\partial J}{\partial W} = \frac{\partial}{\partial W}\left(\sum_{i=0}^{n}(y\text{-}\hat{y})^2\right) = \sum_{i=0}^{n}\frac{\partial}{\partial W}(y\text{-}\hat{y})^2 = 2\sum_{i=0}^{n}(y\text{-}\hat{y})\frac{\partial\hat{y}}{\partial W} \tag{7}$$

Since $\hat{y}$ is a result of XW, we can solve for $\frac{\partial J}{\partial w}$. Once we have, we then increment our weight matrix W. This incrementation depends on the Optimization technique chosen, however with any Gradient Descent based methods, the incrementation is simply the subtracting of a scaled gradient from the current weight matrix. Some such optimizers utilized were AdaGrad, AdaDelta, Adam, and RMSprop, the exact details of which are unnecessary for this paper.

IV. Methodology

A. *Data*

Two datasets were utilized for this project, a small dataset of 347 proteins, and a large dataset of all of the PDB proteins in DSSP format. The full dataset was gathered from The Radboud University Nijmegen in the Netherlands, and contains 116,190 proteins. A brief overview of the datasets is included below in Table 1.

TABLE I. Dataset Overview

Dataset	**Total Proteins**	**Mean Length**	**Total Amino Acids**
Small	347	776	269,272
Large	116,190	1,360	158,023,256

For each DSSP file in each dataset, multiple data formats were produced, specific to a modeling architecture.. The general flow of data processing can be illustrated as in Figure 4 below:

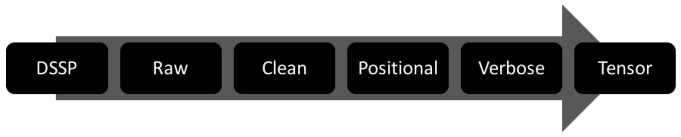

Fig. 4. Data Processing Flow

DSSP files were first parsed, such that only the relevant sequence data was captured. These raw files were then cleaned, which involved replacing blank Structure assignments with the 'C' class, as per the literature. A further idiosyncrasy of the DSSP program is the underassignment of the 'I' class, which represents a π helix, which was remedied by a replacing a structure assignment of 'HHTHH' with 'IIIII'.

From the clean DSSP files, a positional file was constructed, which encodes context around each amino acid. For example, take the following amino acid and structure sequence in Figure 5, below:

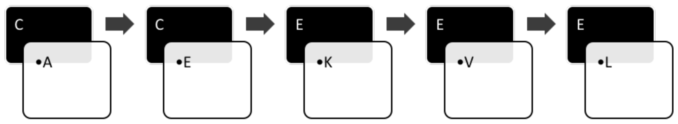

Fig. 5. An Amino Acid Sequence (white), with Structure Labels (black)

Suppose we wanted to encode the context around amino acid 3, K, with two amino acids on either side. The corresponding observation of the third amino acid would be as in Table II:

TABLE II. POSITIONAL AMINO ACID

Amino -2	Amino -1	Amino 0	Amino 1	Amino 2
A	E	K	V	L

This method of encoding is like sliding a window of a fixed length across the sequence, in this case a window of width 5, and combining the context of the window into a single observation. Different window sizes were experimented with, and the largest window size of 19, which encodes 9 amino acids to the left and right of the central, was the most successful. For this window size of 19, the size of the feature space was 912.

From the positional DSSP files, verbose DSSP files were constructed, which binarize the entire feature space of amino acids and positions. Continuing with the previous example, we would have a feature space for each amino acid at each possible position, which would look like:

TABLE III. VERBOSE AMINO ACID

A-2	A-1	…	L1	L2
1	0	…	0	1

Finally, to encode this as a tensor, each residue of the sequence gets encoded as a matrix, with rows as amino acids and columns as positions. Using our initial example, the amino image for the third residue is below in Table IV:

TABLE IV. AMINO IMAGE

Positions → Amino Acids ↓	-2	-1	0	1	2
A	1	0	0	0	0
E	0	1	0	0	0
K	0	0	1	0	0
V	0	0	0	1	0
L	0	0	0	0	1

We then perform this construction of "amino images" for each residue in the sequence, to construct a 3D tensor. Thus each protein can be conceived of as a movie of black and white images. The dataset then is 4D, of shape (n_proteins, n_residues, amino_acids, positions).

Further processing involved experimentation with scaling of the data, and especially with standardizing all proteins to have the same length, by padding with rows of zeros, and then utilizing masking layers; The latter standardizing being important for use with recurrent and convolutional layers.

The distribution of protein lengths for the small datasets is included below in Figure 6. It is worth noting that the majority of proteins are very short, with a single protein of large length. The result is that the short proteins end up with a large percentage of zero rows when standardizing their lengths, which is a challenge to overcome.

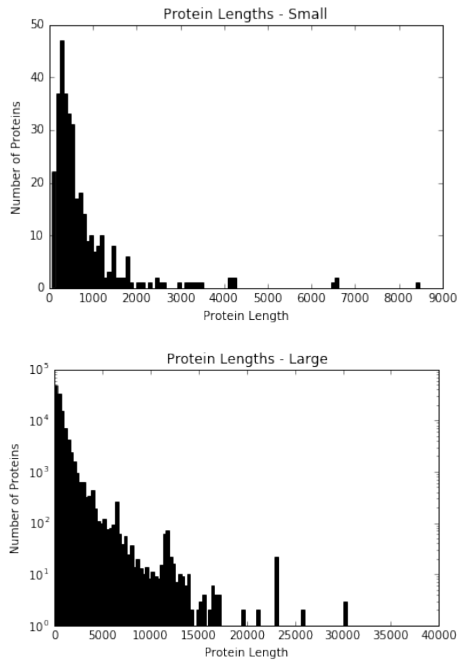

Fig. 6. Distribution of Protein Lengths for the Datasets

Datasets were stored on separate external hard drives, the small on a 500GB and the large on a 1TB. The small dataset, being converted into all the different data formats described above, ended up taking up approximately 100GB of space. The large dataset was then expected to be over 500 times more bytes to store, which would have required roughly 50TB of storage space. Thus, only non-standardized verbose proteins were produced, due to space restrictions.

B. Training

Due to the complexity of the Neural Networks deployed for this project and the size of the data, computationally powerful instances were utilized via Amazon Web Services (AWS). Data was processed on a c3 instance, to handle the large dataset. All models were trained on g2 instances, which were GPU optimized, running Theano and CUDA for performance. For simplicity of code, Keras was used as the package of choice for Neural Network architecture.

To handle the large amount of data, a data generator object was written, which passed data as training and testing splits straight into Keras models for training and validating. This was crucial, as even the small dataset of 347 proteins was too large to load into memory, even for the largest g2 instance type available from AWS (g2.8xlarge), let alone the large dataset, which is three orders of magnitude larger.

Data was primarily stored in CSV format, until data generation was realized as a necessity. In this case, pairs of data and corresponding labels were stored in NumPy's NPZ format. Model training data was saved in JSON format, and model weights were saved as HDF5 files.

Training times varied by model and dataset, as expected, and ranged from less than an hour, up to multiple weeks. These multi-day trainings were terminated, due to time restrictions of the project. Some trainings were lost, due to vanishing gradients and missing data in certain DSSP files from the large dataset. These lost trainings were regrettable, to say the least.

C. Models

Below is a table that summarizes the models trained during the course of this Capstone. The nomenclature was created for ease of identifying the models, since many different architectures were tried. Other models were attempted, but omitted because their training times were intractable.

TABLE V. MODEL NOMENCLATURE AND ARCHITECTURE

Model Name	Layers	Nodes	Activation	Augmentation
FFNN-2L-500-SIG	Input Output	500	Sigmoid Softmax	None
FFNN-2L-500-ELU-BN-DO	Input Output	500	ELU Softmax	Dropout, Batch Normalization
FFNN-3L-1000-ELU-BN-DO	Input Hidden Ouput	1000	ELU Softmax	Dropout, Batch Normalization
FFNN-3L-2000-BN-SReLU-DO	Input Hidden Output	2000	SReLU Softmax	Dropout Batch Normalization
FFNN-4L-2000-BN-SReLU-DO	Input Hidden Hidden Output	2000	SReLU Softmax	Dropout Batch Normalization
CNN-2L-8462-3-3-ELU	Input Output	8462	ELU Softmax	None
CNN-2L-8462-3-3-ELU-DO	Input Output	8462	ELU Softmax	Dropout
RNN-1L-1000-MASK	Output	1000	Softmax	Masking
RNN-2L-1000-ELU-BN-DO	Input Output	1000	ELU Softmax	Dropout Batch Normalization

V. RESULTS

Below is a table that summarizes the results achieved for each model. Training times were included to illustrate the increase in time required for added complexity. Data utilized for training was the small dataset, unless otherwise noted.

TABLE VI. MODEL PERFORMANCES ON SMALL DATASET

Model Name	Epochs	Training Time	Training Accuracy	Validation Accuracy
FFNN-2L-500-SIG	5	8m 13s	64%	65%
FFNN-2L-500-ELU-BN-DO	5	8m 11s	65%	-
FFNN-3L-1000-ELU-BN-DO	20	1h 23m	81%	83%
FFNN-3L-2000-BN-SReLU-DO	50	12h 5m	95%	90%
FFNN-4L-2000-BN-SReLU-DO	20	8h 20m	93%	90%
CNN-2L-8462-3-3-ELU	10	6h	99%*	95%*
CNN-2L-8462-3-3-ELU-DO	10	6h 23m	95%*	95%*
RNN-1L-1000-MASK	10	10m	95%*	95%*
RNN-2L-1000-ELU-BN-DO	10	9h	87%*	94%*

It is worth noting that for the Convolutional and Recurrent Networks, the accuracies reported are deceiving. Since the input to these networks must be in the 4D (samples, timesteps, width, height) or 3D tensor (samples, timesteps, features) forms, respectively, we had to standardize the length of every protein. This was done by searching the dataset for the longest length protein, and then padding each protein that was shorter with observations of all zeroes. The corresponding label for these padded rows was a one-hot encoding of all zeroes. Thus, when accuracies are calculated, the network will inherently get most of the observations correct, since no matter what the weights are, when it multiplies by the zero-valued observation, it will always produce zeroes (when no bias term is added).

The largest improvements in performance seemed to come from changes in activation functions, with SReLU giving substantial returns over ELU, at least with simple Feed Forward networks. Batch Normalization also made an impact, especially as it relates to converging upon maxima in a timely fashion.

VI. FUTURE WORK

Much is still left to do for this project, especially in exploring the realm of Convolutional and Recurrent Neural Networks. Different convolution filter sizes should be experimented with, to find sizes that can capture patterns in amino images for prediction. In attempting to complete this project within a three-month timeframe, data processing for the large dataset and training times proved prohibitive.

The large dataset's 50+TB space requirement also proved difficult to handle. Future effort should go into storing data with AWS S3, so that all forms of the data can be stored and the interfacing between AWS EC2s can be more streamlined.

Marrying this project with Keras was initially helpful, to get simple Feed-Forward Neural Networks off the ground and training. However, Keras' recurrent layers seem to have much more overhead than Theano. It is also currently impossible to use Batch Normalization at the same time as Masking layers in Keras, which makes more training epochs necessary for an already lengthy training. For more tractable training times, Theano could be utilized, at least for future Recurrent work.

An additional complexity of adding bi-directionality to the Neural Network would have likely proved useful, and is the ultimate, yet far away, goal of future work.

Metrics will need to be designed to correctly calculate accuracy with standardized proteins, skipping the padding rows, since they deceivingly inflate accuracy.

Finally, other models could be explored, especially Probabilistic Graphical Models, such as Conditional Random Fields and Hidden Markov Models, as they incorporate the influence of previous (or future) structural designations on the current, which is a useful property for the problem at hand.

VII. Conclusion

Overall, the utilized methodology of producing verbose proteins, with a sliding context window, proved very successful as a way to encode context into a single amino acid residue, as indicated by the achieved 90% accuracy. Batch Normalization and SReLU layers were very effective in decreasing training time and accuracy, respectively, while additional hidden nodes and deeper networks provided minimal returns. Recurrent and Convolutional layers likely hold much potential, but require more time for development.

VIII. Acknowledgments

I would like to thank my friends, family and teachers, without all of whom I would not have been able to complete this work. I would also like to explicitly thank Mike Bowles and his collegue Ron Shigeta, who graciously provided his curated dataset of 347 proteins, which was indispensible for experimental training, as well as The Radboud University Nijmegen of the Netherlands for their well-managed full DSSP dataset.

References

[1] A. Guzzo, "The Influence of Amino Acid Sequence on Protein Structure", *Biophysical Journal*, vol. 5, no. 6, pp. 809-822, 1965.

[2] J. Prothero, "Correlation between the distribution of amino acids and alpha helices", *Biophysical Journal*, vol. 6, no. 3, pp. 367-370, 1966.

[3] M. Schiffer and A. Edmundson, "Use of Helical Wheels to Represent the Structures of Proteins and to Identify Segments with Helical Potential", *Biophysical Journal*, vol. 7, no. 2, pp. 121-135, 1967.

[4] D. Kotelchuck and H. Scheraga, "THE INFLUENCE OF SHORT-RANGE INTERACTIONS ON PROTEIN CONFORMATION, II. A MODEL FOR PREDICTING THE -HELICAL REGIONS OF PROTEINS", *Proceedings of the National Academy of Sciences*, vol. 62, no. 1, pp. 14-21, 1969.

[5] P. Lewis, N. G[unk]o, M. G[unk]o, D. Kotelchuck and H. Scheraga, "Helix Probability Profiles of Denatured Proteins and Their Correlation with Native Structures", *Proceedings of the National Academy of Sciences*, vol. 65, no. 4, pp. 810-815, 1970.

[6] J. Garnier, D. Osguthorpe and B. Robson, "Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins", *Journal of Molecular Biology*, vol. 120, no. 1, pp. 97-120, 1978.

[7] P. Argos, J. Schwarz and J. Schwarz, "An assessment of protein secondary structure prediction methods based on amino acid sequence", *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 439, no. 2, pp. 261-273, 1976.

[8] H. Berman, "The Protein Data Bank: a historical perspective", *Acta Cryst Sect A*, vol. 64, no. 1, pp. 88-95, 2007.

[9] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features", *Biopolymers*, vol. 22, no. 12, pp. 2577-2637, 1983.

[10] D. Frishman and P. Argos, "Knowledge-based protein secondary structure assignment", *Proteins: Structure, Function, and Genetics*, vol. 23, no. 4, pp. 566-579, 1995.

[11] C. Andersen, A. Palmer, S. Brunak and B. Rost, "Continuum Secondary Structure Captures Protein Flexibility", *Structure*, vol. 10, no. 2, pp. 175-184, 2002.

[12] W. Pirovano and J. Heringa, "Protein Secondary Structure Prediction", *Methods in Molecular Biology*, pp. 327-348, 2009.

[13] O. Dor and Y. Zhou, "Achieving 80% ten-fold cross-validated accuracy for secondary structure prediction by large-scale training", *Proteins*, vol. 66, no. 4, pp. 838-845, 2006.

[14] G. Karypis, "YASSPP: Better kernels and coding schemes lead to improvements in protein secondary structure prediction", *Proteins*, vol. 64, no. 3, pp. 575-586, 2006.

[15] S. Mirjalili, S. Saremi and S. Mirjalili, "Designing evolutionary feedforward neural networks using social spider optimization algorithm", *Neural Comput & Applic*, vol. 26, no. 8, pp. 1919-1928, 2015.

[16] B. Petersen, T. Petersen, P. Andersen, M. Nielsen and C. Lundegaard, "A generic method for assignment of reliability scores applied to solvent accessibility predictions", *BMC Structural Biology*, vol. 9, no. 1, p. 51, 2009.

[17] J. Peng and J. Xu, "Raptorx: Exploiting structure information for protein alignment by statistical inference", *Proteins*, vol. 79, no. 10, pp. 161-171, 2011.

[18] J. Garnier, J. Gibrat and B. Robson, "[32] GOR method for predicting protein secondary structure from amino acid sequence", *Methods in Enzymology*, pp. 540-553, 1996.