Protein Interaction Sites Prediction using Deep Learning

Sourajit Basak, *The University of Western Ontario*

Supervisor: Ilie, Lucian, *The University of Western Ontario*

Abstract

The accurate prediction of protein-protein interaction (PPI) binding sites is a fundamental problem in bioinformatics, since most of the time proteins perform their functions by interacting with some other proteins. Experimental methods are slow, expensive and not very accurate, hence the need for efficient computational methods.

In this thesis, we perform a study aiming to improve the performance of the currently best program for binding site prediction, DELPHI. We have employed some of the currently best techniques from machine learning, including attention and various embedding techniques, such as BERT and ELMo. This is the first time such tools are being tested for this problem. We have tested many architectures on a large dataset and analyzed our findings. While we succeeded to improve the performance, it is interesting to notice that some of the best machine learning techniques failed to provide the expected improvement, a fact that will require further investigation.

Keywords: Protein-Protein Interaction, Deep Learning

Lay Summary

Proteins are an essential component to any organism which takes part in every process within living cells. The cell functioning often happens due to the interaction of proteins with other proteins. This mechanism is called protein-protein interaction (PPI) in biology. In this event, predicting those amino acids which helps in binding, also referred as PPI binding sites becomes a fundamental problem in computational biology.

We present, a thorough study which aims to improve the performance of DELPHI, the current best program for predicting binding sites. In order to achieve this, we have implemented some of the recent best techniques from machine learning. Many architectures on a large dataset have been tested to analyze our findings. It is very interesting to note that even if we succeeded to improve the performance, some of the best machine learning techniques failed to provide expected improvement, which definitely leads us towards further investigation.

Acknowlegements

I would like to express my most sincere gratitude to my supervisor Dr. Lucian Ilie for giving me this opportunity to do research in his lab. His encouragement and feedback led me to complete this dissertation. It was an absolute honour and privilege to work with such a wise person. I would also like to show my heartiest gratitude to all the professors who taught me and helped me build the background for this dissertation.

I would like to thank my father Sanjib Basak and my mother Soma Basak for their unconditional love and support throughout this arduous journey.

I acknowledge all my friends and lab mates for always motivating me and helping me overcome all hurdle. Last but not the least, I thank the Department of Computer Science at Western University for funding my graduate studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The main focus of bioinformatics is to interpret and analyze biological data using mathematical, statistical and computational models. In most of the cases these data are represented by nucleotide sequences also referred as DNA, RNA or protein sequences. In biology proteins are essential to organisms and participate in every process within cells. They perform their functions by interacting with other proteins which is often termed as Protein-Protein Interaction (PPI). When two proteins interact with each other the bonding amino acid residues are called PPI binding sites. Identifying PPI binding sites is a fundamental problem in computational biology as many applications like understanding cell regulatory mechanism, locating drug targets or predicting protein functions are dependent on this.

1.1 Motivation

Experimental methods for solving this problem are slow, expensive and not very accurate. To overcome this, several computational methods have been developed for improvement purpose. One of such leading method is PIPE-sites [1] which is an algorithm based approach to predict PPI binding sites, and discussed later. In recent years with the advancement of technology and computational power, machine learning based models are rapidly increasing to solve this problem.

Programs like DLPred [39], DeepPPISP [38], SCRIBER [40] and DELPHI [19] which are using machine learning techniques are performing very well. Since, this is a growing research area we aim to improve the prediction performance further by using the latest techniques.

1.2 Problem Statement

There are three problems concerning prediction of protein interactions. The first predicts whether two proteins interact, as a whole. The second, which we study, is whether each individual amino acid also called residue of a given protein is a binding site or not, that is, interacting with some other residue in another, or the same protein. The third problem concerns the interaction between pairs of residues.

In recent years, machine learning has become one of the most important research areas in computer science. The major contributions of it are in areas like computer vision and natural language processing. However, these techniques can also be implemented in computational biology more specifically in problems like PPI binding site prediction. We present a comprehensive study aiming to improve the performance of the currently best program for binding site prediction, DELPHI. Considering the nature of biological data which comes in a sequence based format, we focus mainly on those methods involving NLP problems such as attention, embedding etc. This study attempts to include all the possible ways for improving the prediction performance of the program to the best of our knowledge.

1.3 Objective

The main objective is to try out some of the modern machine learning tools in a binding site prediction problem for the first time. We chose DELPHI as it is currently the best program to solve this problem. We mainly focus on ideas like embedding, attention and positional encoding, as it has been proven by many researchers that, these techniques made a significant improvement in NLP problems. Our goal is to demonstrate how biological problems can also be improved by applying these methods in the right way.

1.4 Thesis Contribution

The major contribution of this thesis is that we compared several state-of-the-art machine learning methods which are very useful in NLP to explain how the performance of a binding site prediction problem can be improved by placing them into the program, DELPHI.

Many researchers are working on this problem and most of them are using machine learning as well. Our study will definitely help them to understand the right way for selecting the appropriate methods for their programs in similar problems and also to improve the performance by a significant amount.

1.5 Thesis Outline

In Chapter 2, we discussed about the background of our thesis, starting with some basics of biology, then how deep learning has been introduced into bioinformatics in recent years, followed by some basic definition and notation.

In Chapter 3, we mainly focused on the methodologies we used. Here, first we described DELPHI [19] and its working principle then we illustrate all the techniques used for improvement.

In Chapter 4, the experimental results obtained from all the methods we use and compare the performance of the best model from each approach.

In Chapter 5, we summarize our work and present some possible future directions.

Chapter 2

Background

2.1 DNA

In biology, DNA(Deoxyribonucleic acid) [36] is a molecule which is composed of two polynucleotide chains that loop around each other so that, it can form a double helix structure. DNA carries genetic instructions for several purpose such as development, functioning, growth and reproduction of any organisms that we know and many viruses. There are four bases that can be found in any DNA, they are adenine (A), cytosine (C), guanine (G) and thymine (T). These bases are coupled with the sugar-phosphate that completes the nucleotide formation. The general rule states that adenine pairs with thymine and guanine pairs with cytosine to form A-T and G-C base pairs as shown in Figure 2.1.

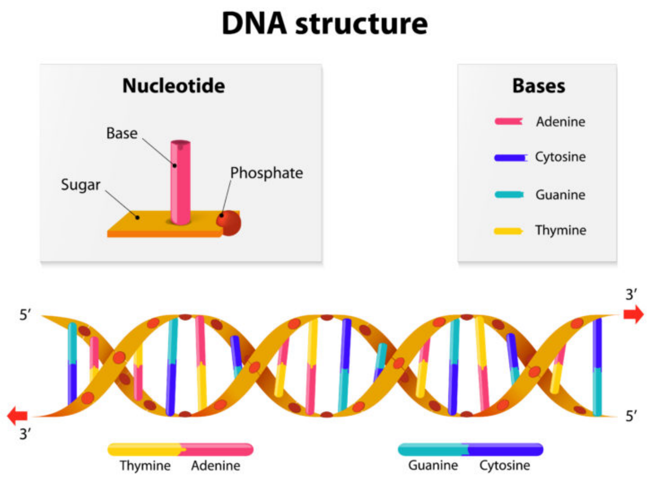


Figure 2.1: The structure of DNA
From geneonline.news

Transcription is the process by which the information contained in a DNA is replicated in the form of messenger RNA (mRNA). In translation, messenger RNA (mRNA) is decoded outside the nucleus into ribosome, to produce a specific amino acid chain, or polypeptide. The polypeptide later folds into an active protein and performs its functions in the cell.

2.2 Protein

Proteins [36] are another bio-molecules which consist a larger set of amino acids. It has one or more long chains of amino acid residues also called a polypeptide. Amino acid sequences are actually gene sequences which are further encoded into genetic code for easier reference. There are in total 20 amino acids which can be found in a protein and their genetic codes are described in Table 2.1

Category	Amino Acid	Three-letter Code	One-letter Code
Charged	Arginine	Arg	R
	Lysine	Lys	K
	Aspartic acid	Asp	D
	Glutamic acid	Glu	E
Polar	Glutamine	Gln	Q
	Asparagine	Asn	N
	Histidine	His	H
	Serine	Ser	S
	Threonine	Thr	T
	Tyrosine	Tyr	Y
	Cysteine	Cys	C
Amphipathic	Tryptophan	Trp	W
	Tyrosine	Tyr	Y
	Methionine	Met	M
Hydrophobic	Alanine	Ala	A
	Isoleucine	Ile	I
	Leucine	Leu	L
	Methionine	Met	M
	Phenylalanine	Phe	F
	Valine	Val	V
	Proline	Pro	P
	Glycine	Gly	G

Table 2.1: The 20 Amino Acids and Their Role in Protein Structures

Proteins usually fold into a unique 3D structure in most cases. There are mainly four types of protein structures. The primary structure refers to the sequence of amino acids that forms a polypeptide chain. The secondary structure is formed when the chain is locally folded into sheets or helices. The tertiary structure is the three-dimensional folding of the helices or sheets due to side chain interaction. Sometimes, proteins may consist of multiple amino acid chains and these are more complex structure formations, also called as the quaternary structure. Figure 2.2 shows different protein structures in detail.

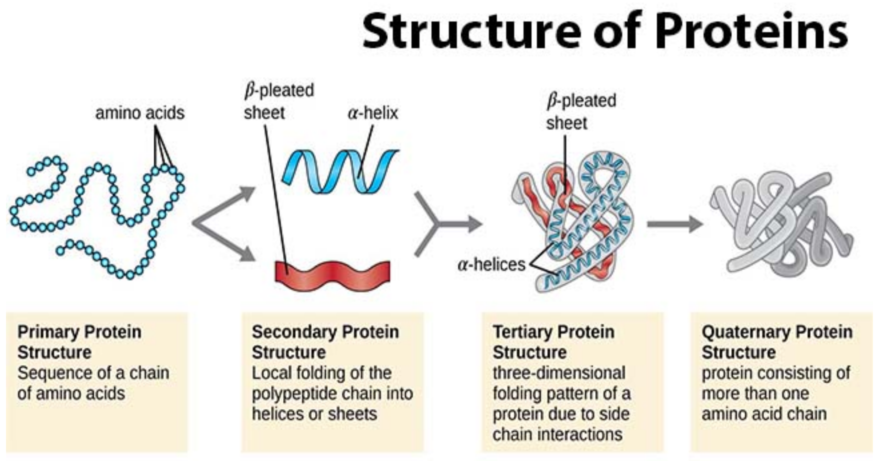
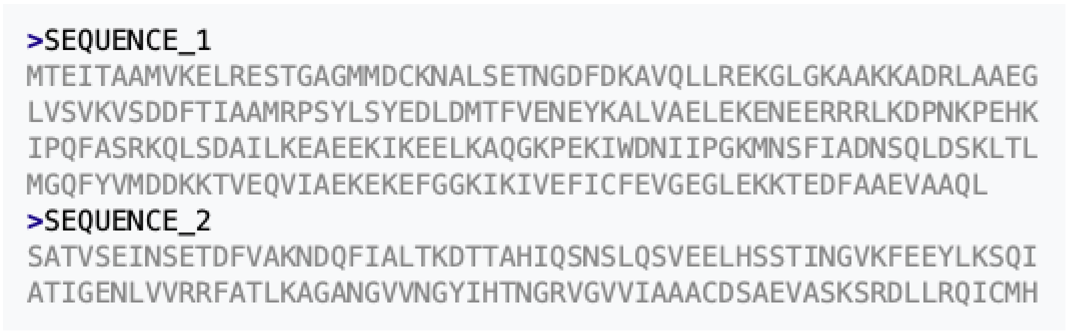

Structure of Proteins
amino acids
β-pleated sheet
α-helix
β-pleated sheet
α-helices
Primary Protein Structure
Sequence of a chain of amino acids
Secondary Protein Structure
Local folding of the polypeptide chain into helices or sheets
Tertiary Protein Structure
three-dimensional folding pattern of a protein due to side chain interactions
Quaternary Protein Structure
protein consisting of more than one amino acid chain

Figure 2.2: Types of Protein Structure
The primary, secondary, tertiary, and quaternary structure of proteins From microbenotes.com

2.3 A FASTA file

In bioinformtics, a neucleotide sequence or amino acid (protein) sequences can be represented in a text based format called the FASTA format [36]. Here, nucleotides or amino acids are represented using single-letter codes. It is a very simple format so that it can be easily parsed or manipulated using any text processing tools or scripting languages. The first line in a FASTA file starts with a ">" (greater-than) symbol, used for a unique description of the sequence. Following the initial line is the actual sequence itself in standard one-letter character string. This two line pattern is repeated as necessary.



```
>SEQUENCE_1
MTEITAAMVKELRESTGAGMMDCKNALSETNGDFDKAVQLLREKGLGKAAKKADRLAAEG
LVSVKVSDDFTIAAMRPSYLSYEDLDMTFVENEYKALVAELEKENEERRRLKDPNKPEHK
IPQFASRKQLSDAILKEAEEKIKEELKAQGKPEKIWDNIIPGKMNSFIADNSQLDSKLTL
MGQFYVMDDKKTVEQVIAEKEKEFGGKIKIVEFICFEVGEGLEKKTEDFAAEVAAQL
>SEQUENCE_2
SATVSEINSETDFVAKNDQFIALTKDTTAHIQSNSLQSVEELHSSTINGVKFEEYLKSQI
ATIGENLVVRRFATLKAGANGVVNGYIHTNGRVGVVIAAACDSAEVASKSRDLLRQICMH
```

Figure 2.3: An example of a multiple sequence FASTA file

2.4 Deep Learning in Bioinformatics

Deep Learning [17] is a branch of Machine Learning, it is powerful learning algorithms inspired by how the brain works. Traditional computational programs follows handcrafted mathematical computations whereas machine learning algorithms actually learn them. This is more like a transition from algorithm centric approach to data centric approach. The machine learns these mathematical functions from known samples which can be either labeled(supervised learning) or unlabeled(unsupervised learning). This learning mechanism is very helpful in order to solve any prediction problem like classification or regression. Classification is the process of identifying a model which helps in separating the data into multiple categorical classes i.e. discrete values. In regression we try to find a model for measuring the data into continuous real values instead of using classes or discrete values.

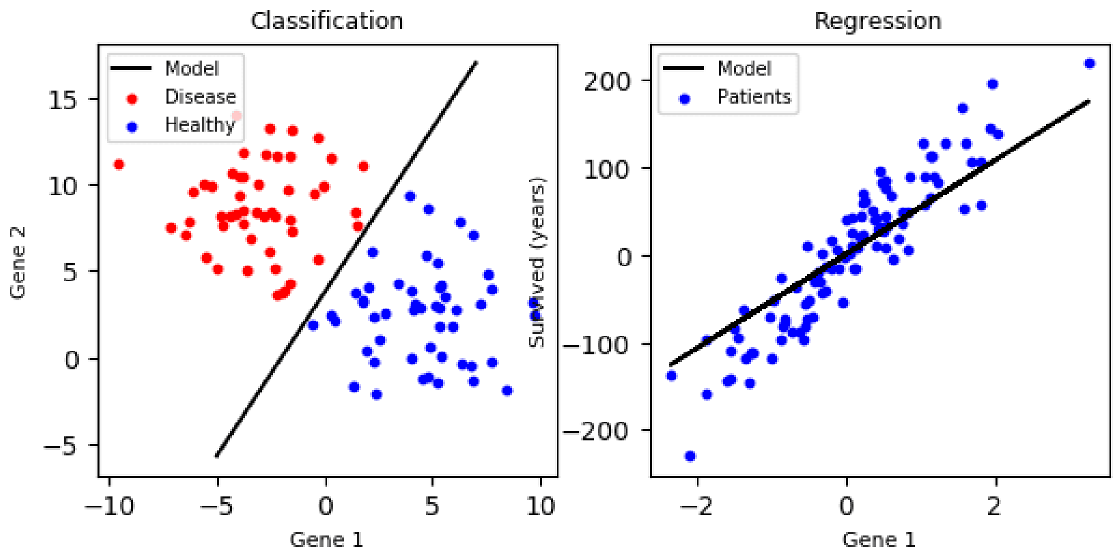


Figure 2.4: An example of Classification vs Regression
From mc.ai

Many bioinformatics problems have been transformed into well defined classification or regression problems. Several biologically relevant prediction tasks have been well defined([29]) to serve as benchmarks in terms of understanding the need of machine learning in bioinformatics as well as the advancement in computational biology. They are as follows:

Secondary Structure Prediction

Secondary structure prediction is a sequence-to-sequence task, where for a given protein sequence each input amino acid x_i is mapped to a label $y_i \in$ {Helix, Strand, Other}

The most important feature of secondary structure is it is used for understanding the function of a protein. It is very helpful for proteins with unknown structure. These prediction tools are very commonly used to create high quality input features for higher-level models.

Secondary structure prediction tests the degree to which models learn local structure.

Contact Prediction

Contact prediction is a pairwise amino acid task, where for a given sequence x each pair of input amino acids (x_i, x_j) is mapped to a label $y_{ij} \in \{0,1\}$ and the label represents whether the amino acids are in contact or not.

The major advantage of contact prediction is that contact maps provide powerful global information i.e. they are very useful in cases like modeling of full 3D protein structure. In practice, medium or long-range contact maps are most useful but they are not very often found, may be as few as twelve sequence positions apart, or as many as hundreds apart.

A model can be well interpreted in context of global protein, where medium and long-range contacts are in higher quantity. This makes contact prediction an ideal task.

Remote Homology Detection

Remote Homology Detection is a sequence classification task where each input protein x is mapped to a label $y \in \{1,....,1195\}$. Here, each class represents different possible protein folds.

Detection of remote homologs is of great interest in microbiology and medicine; e.g., for detection of emerging antibiotic resistant genes and discovery of new CAS enzymes.

In microbiology and medicine, detection of remote homologs is a very important task. There are several applications such as detection of emerging antibiotic resistant genes, discovery of new CAS enzymes, etc.

Remote homology detection measures a model's ability to detect structural similarity across distantly related inputs.

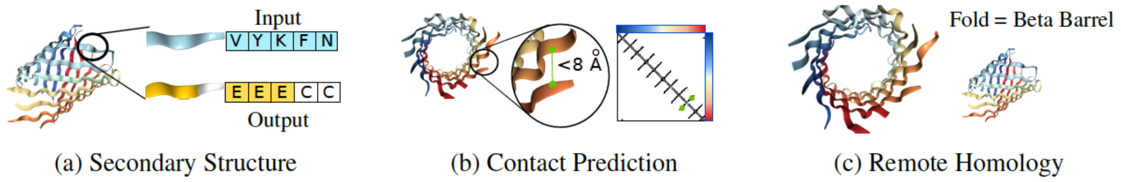


Figure 2.5: Protein Structure and Annotation Tasks.
From [29]

Fluorescence Landscape Prediction

Fluorescence Landscape Prediction is a regression task where each input protein x is mapped to a label $y \in \mathbb{R}$, that corresponds to the fluorescence intensity of the input protein.

For a given protein, the number of possible sequences is $O(L^m)$, where L is the length of the protein and m is mutations. This is not recommended as it takes large space for exhaustive search via experiment, even if m is modest. However, this greedy optimization approach is unlikely to succeed due to second and higher-order interactions between mutations at different positions. In recent years machine learning methods have already succeeded in protein engineering tasks with more accurate predictions and more efficient exploration of the landscape.

This task is very useful to test how a model can distinguish between very similar inputs, also the way it generalizes unseen combinations of mutations.

Stability Landscape Prediction

Stability Landscape Prediction is a regression task where each input protein x is mapped to a label $y \in \mathbb{R}$ denoting a threshold for measuring the most extreme circumstances in which the protein will maintains its fold.

It is very important to ensure stable protein designs, for example, while delivering any drugs we must know at what temperature it must be preserved. During expensive protein engineering experiments it is useful to maximize the yield of top candidates by refining the protein measurements in a large sample set.

This helps to understand a model's ability to generalize relevant sequences from a broad sample of proteins and to localize this information in a neighborhood of a few sequences.

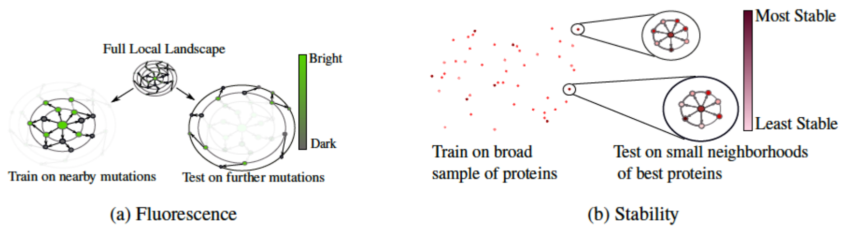

Full Local Landscape
Bright
Dark
Train on nearby mutations
Test on further mutations
(a) Fluorescence
Most Stable
Least Stable
Train on broad sample of proteins
Test on small neighborhoods of best proteins
(b) Stability

Figure 2.6: Protein Engineering Tasks.
From [29]

In conventional machine learning, model performance often depends upon the data representations or input features. These features are designed and developed by experts in related fields depending upon their relevance on a certain task. In recent years deep learning has emerged due to its parallel computing power and sophisticated algorithms. It has overcome previous limitations, and academic interest has increased rapidly since the early 2000s [23]

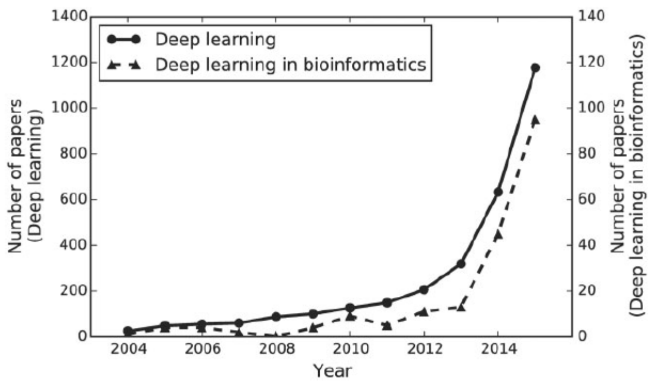


Figure 2.7: Approximate number of published deep learning articles by year
The number of articles is based on the search results on http://www.scopus.com with the two queries: 'Deep learning,' 'Deep learning' AND 'bio*'. [23]

2.4.1 Basic Notions and Definitions

Deep Neural Networks

A neural network learns complex mathematical functions [17] using its layered architecture. These functions are often highly non linear due to the network structure. A standard deep neural network has at least three layers which are: input, output and hidden. The number of the hidden layers defines the depth of the network and often used to obtain better performance. The term "deep learning" means many hidden layers.

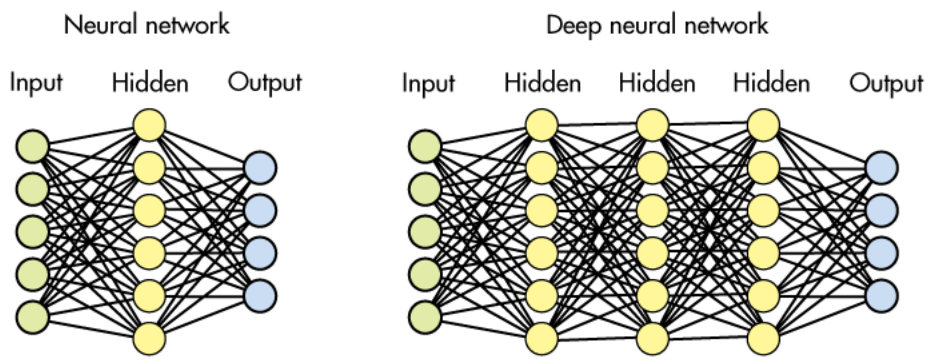


Figure 2.8: An example of Deep Neural Network
From electronicdesign.com

Activation Functions

The input layer takes the raw data $(x^{(i)}, y^{(i)})$, where $x^{(i)}$ is the feature vector of the i^{th} training sample and $y^{(i)}$ is the corresponding label. In bioinformatics, embedding, one-hot encoding, PSSM are most widely used featurization techniques. In NLP, one-hot encoding is a $1 \times N$ matrix consisting of 0's in all cells except a single 1 in a cell used to uniquely identify the word. Each neuron applies a mathematical function h_θ, also called hypothesis to obtain some output. A weight value θ is multiplied with every input before applying the hypothesis function, with the intuition of weighing it's importance. These outputs are further passed into non-linear functions also termed as "activation functions". The idea here is, if an output surpasses a certain threshold value then only that neuron will be activated and it will pass the information to the next neuron. Table 2.2 shows a list of popular activation functions which are used often.

Training and Testing

At the input layer of a deep neural network we assign some weight to each input sample which refers to its importance, these weights are also applied to each hidden units within the network using the same principle. In order to achieve a better prediction performance we need to adjust these weights several times till we find the optimal one. This entire process is called training in machine learning terminology.

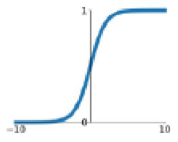
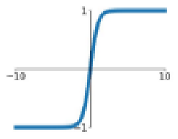
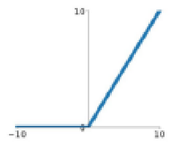
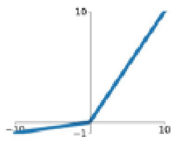
Function	Plot	Equation
Sigmoid		$\sigma(x) = \frac{1}{1+e^{-x}}$
tanh		$\tanh(x)$
ReLU		$\max(0, x)$
Leaky ReLU		$\max(0.1x, x)$

Table 2.2: A list of activation functions
From towardsdatascience.com

First, we initialize our network with some random weights and measure its performance. The difference between the predicted value and actual value is called loss. To make our model perform better we need to minimize this loss function and update the weights in every unit within the network. We will repeat this process until we find the optimal point. Two of the most commonly used loss functions are: *the mean squared error*, shown in the Equation 2.1, where $J(\theta)$ is the cost function, $h_\theta(x^{(i)})$ is the hypothesis function, $y^{(i)}$ is the ground truth labels and n is the number of training samples, and *the cross-entropy error*; see Equation 2.2

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 \tag{2.1}$$

$$J(\theta) = \sum_{i=1}^{n} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)})) \tag{2.2}$$

Gradient descent is the most commonly used algorithm for loss optimization, where we update the current weight θ_i by subtracting from it the gradient of the cost function times the learning rate α in each step, as shown in Equation 2.3

$$\theta_i = \theta_i - \alpha \frac{\delta}{\delta \theta_i} J(\theta) \tag{2.3}$$

Figure 2.9 shows how gradient descent works in a 2 dimensional space. Here, we start with some θ_0, θ_1 and keep changing it using the above mentioned formula to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum.

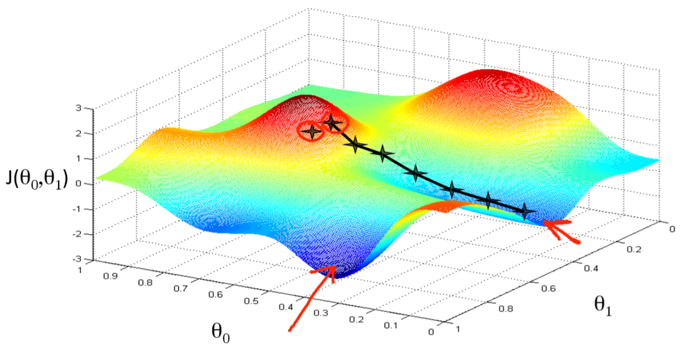


Figure 2.9: Gradient Descent in 2D space
From towardsdatascience.com

Once we optimized the loss function and set the weights in our network, the model is ready to perform prediction tasks. In prediction, we test some new data samples which our model has not seen before to know how well our model is performing. It is often observed that the testing loss is higher than the training loss for some obvious reason, i.e. the test samples are new and the model weights are not adjusted based on that. In the end the lower the test loss better the model performance.

In machine learning there are ways to understand how well our model fits the data, the two main parameters are called bias and variance which is useful for this purpose. In overfitting the training loss is much lower than the test loss and this happens due to high variance, on the other hand in underfitting the model does not perform during training itself which also results a bad testing performance because of high bias. A good fit is observed only when both the bias and variance are moderate. See Figure 2.10 for illustration.

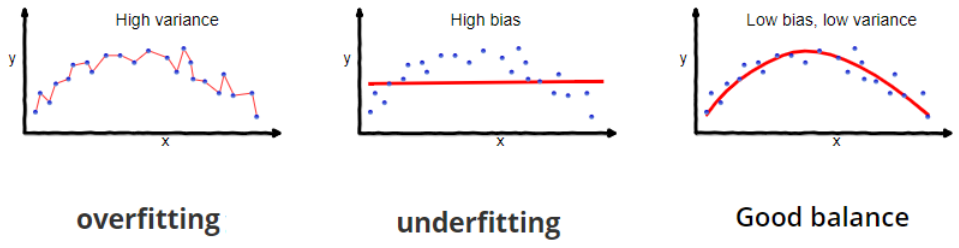


Figure 2.10: Bias Variance tradeoff
From towardsdatascience.com

2.4.2 Convolutional Neural Network

In deep learning convolutional neural network (CNN) is one of the most widely neural network architectures in recent decades. The network employs a mathematical operation called convolution which is a specialized kind of linear operation. In computer vision most of the recent advancements are based on this principle. The first CNN was built in 1998 called LeNet [18] for handwritten digit classification. Now, with the progress of time there are several other major CNN architectures like VGG-16 [31] GooLeNet [32] ResNet [12] and AlexNet [16] considered as the state-of-the-art. The main components of a CNN are convolutional layer, pooling layer and fully connected layer as shown in Figure 2.11; their details are discussed in the later section.

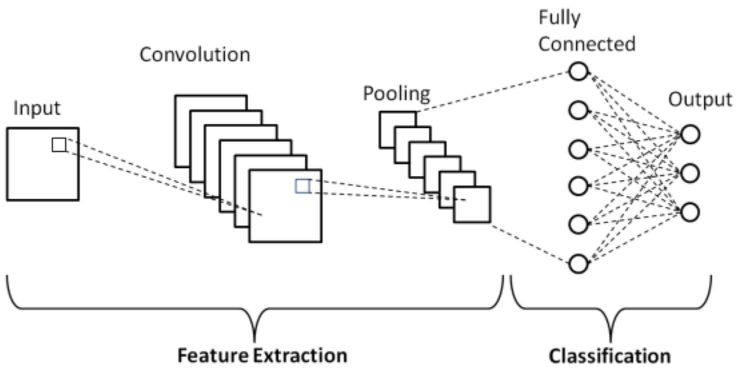


Figure 2.11: The main components of a Convolutional Neural Network
From researchgate.net

Convolution Layer

This is the most important layer in a CNN model which performs a linear mathematical operation. In this process a kernel or filter of fixed size is being traversed over an input image like a sliding window, doing an element-wise multiplication with the input image in each stride and resulting an output image as shown in Figure 2.12. The main purpose of this is to extract useful features from the input image. The pixels on the edges are used less than the pixels in the middle of the image results shrinking output, which can be avoided using boundary padding.

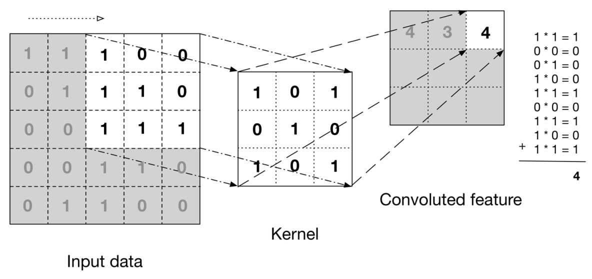


Figure 2.12: The computation mechanism of a convolution operation
From medium.com

Pooling Layer

The next important layer in a CNN model is pooling layer which is often lead by a convolutional layer. This is responsible for reducing the extracted features from the previous layer keeping the dominant features that are required to represent the original image. Pooling operation is mainly of two types, in max pooling the maximum value is pooled from the portion of the image and in average pooling the average value is captured from the portion. Figure 2.13 shows how the pooling operation works.

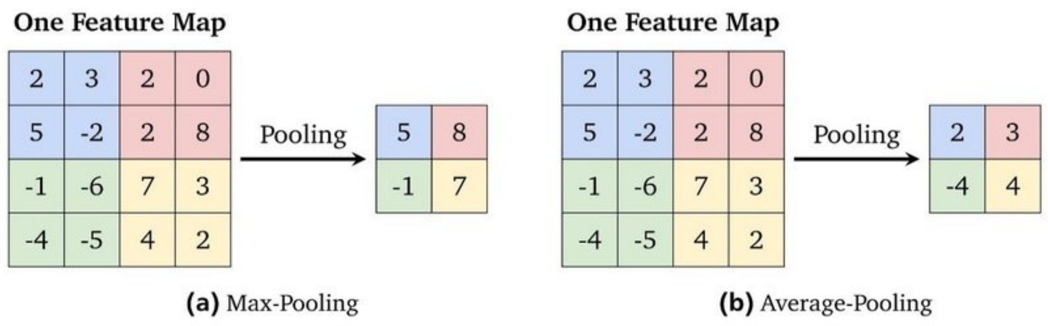


Figure 2.13: A detailed mechanism of pooling layer
From researchgate.net

Fully Connected Layer

This is usually the last layer of a convolutional neural network which transforms the result obtained from the previous layers into fewer values. The term "fully connected" means each node of this layer is connected to all the nodes of its previous layer by doing a weighted sum of all the input. Each node computes a mathematical function shown in Equation 2.4 where $h_\theta(x)$ is the hypothesis on the given input x, θ^T and b are the weight and bias term. Figure 2.14 shows a network design with many fully connected layers.

$$h_\theta(x) = \theta^T x + b \tag{2.4}$$

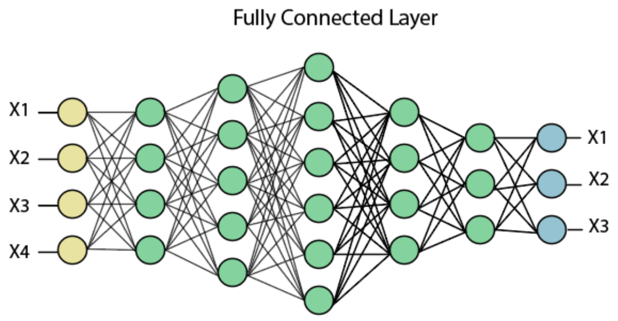


Figure 2.14: An illustration of the fully connected layer
From javatpoint.com

2.4.3 Recurrent Neural Network

Another invention in deep learning is recurrent neural network(RNN), mainly used for modelling sequential data. In recent years most of the natural language processing(NLP) applications use this architecture. A standard RNN is designed as a sequence of blocks also called "timestamps", where each block takes a certain chunk from an input sequence. Several featurization techniques like one hot encoding, embeddings are used to vectorize the input of a RNN. In this network architecture the hidden states are stacked on top of each other and combining it all together forms the complete network. Figure 2.15 shows the detailed design of a RNN.

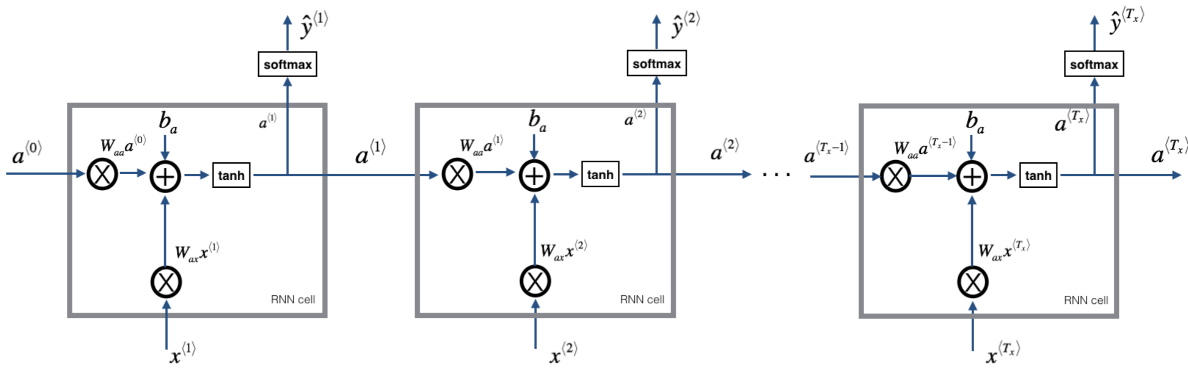


Figure 2.15: The basics of recurrent neural network
From datascience-enthusiast.com

Each RNN cell takes the input $x^{<t>}$ for it's t^{th} position and the output $a^{<t-1>}$ from its previous cell. Both the input parameters are multiplied with some weight which is learned during the training process. The output $a^{<t>}$ and $\hat{y}^{<t>}$ are produced by Equation 2.5 and Equation 2.6. The softmax function takes a vector of n real numbers as input, and normalizes it into a probability distribution consisting of n probabilities, proportional to the exponentials of the input numbers.

$$a^{<t>} = tanh(W_{ax}x^{<t>} + W_{aa}a^{<t-1>} + b_a) \tag{2.5}$$

$$\hat{y}^{<t>} = softmax(W_{ya}a^{<t>} + b_y) \tag{2.6}$$

LSTM and GRU

The main challenge with a recurrent neural network is vanishing gradient. During back propagation, each node in a certain layer computes their gradient with respect to the previous layer and as a result it becomes much smaller such that it is treated as vanished. To overcome this problem long short term memory(LSTM) [14] and gated recurrent unit(GRU) [6] have been introduced by replacing a standard RNN cell. The main components of LSTM are input gate, output gate, forget gate and a cell state whereas in GRU update gate and reset gate serves the purpose of memorization. These gates also include activation functions which decide whether to remember a certain input or not. For illustration of LSTM and GRU please see Figure 2.16

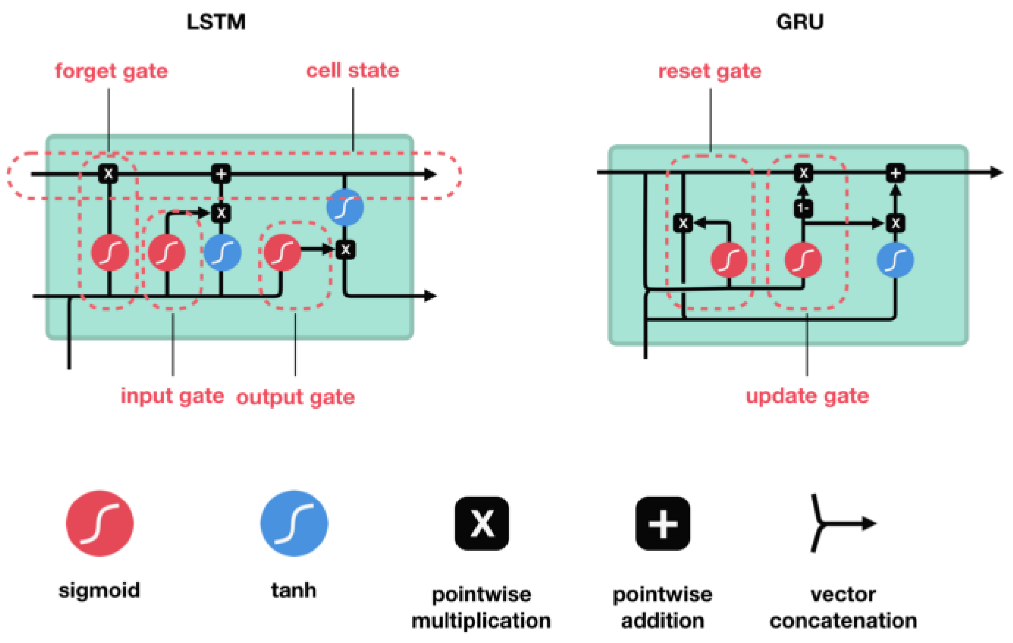


Figure 2.16: A single LSTM and GRU unit
From towardsdatascience.com

2.4.4 Tensorflow and Keras

TensorFlow [33] is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Keras [36] is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. A list of popular keras function is shown in Table 2.3

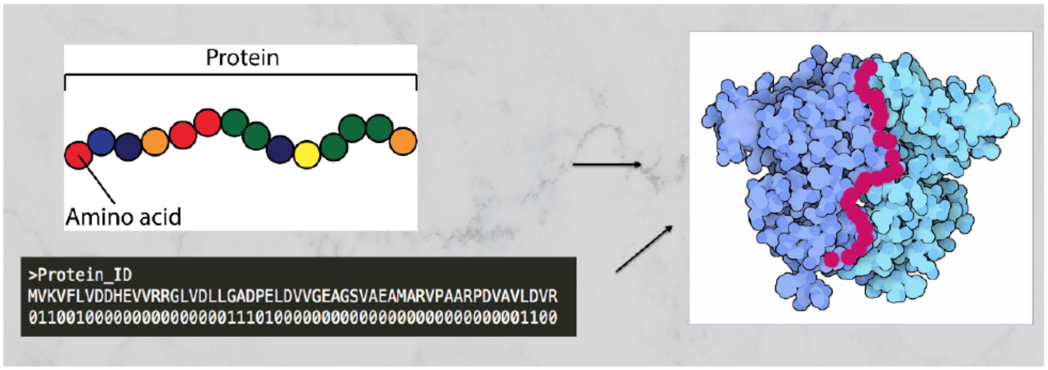
Operation	Syntax for implementation
Training	model.fit(x = input_data, y = labels, epochs = 10, ...)
Prediction	model.predict(x = input_data, ...)
Convolution	keras.layers.Conv2D(filters, kernel_size, ...)
Max Pooling	keras.layers.MaxPooling2D(pool_size = (2, 2), ...)
Average Pooling	keras.layers.AveragePooling2D(pool_size = (2, 2), ...)
Fully Connected Layer	keras.layers.Dense(units = 1, activation = ‘sigmoid’, ...)
LSTM	keras.layers.LSTM(units = 32, activation = ‘tanh’, ...)
GRU	keras.layers.GRU(units = 32, activation = ‘tanh’, ...)

Table 2.3: A list of Keras functions

2.5 Protein Binding Site Prediction

Proteins often perform their functions by interacting with other proteins. During this process they bind with each other and those bonding amino acid residues are protein-protein interaction binding sites. Many biological problems such as understanding cell regulatory mechanisms, locating drug targets, predicting protein functions are dependent on the accurate detection of PPI binding sites. There are many industries working on this area of research to accelerate their drug design process. The protein data bank (PDB) stores information of protein binding sites derived from several 3D protein structures but they are limited. Over the past decades many researchers have attempted to solve this prediction problem but the performance is still far from satisfactory.

In our work we focus on sequence based method, where the program is trained with primary sequences stored in FASTA format as shown in Figure 2.17(left) along with its binding site information (0,1) where 1 indicates binding sites and 0 means non-binding sites for the corresponding location. These binding sites are same as the red circles marked on the right, assuming that the protein interacts with another protein. Our program will predict the binding sites for any new protein sequence.



```
>Protein_ID
MVKVFLVDDHEVVRRGLVDLLGADPELDVVGEAGSVAEAMARVPAARPDVAVLDVR
01100100000000000000001110100000000000000000000000001100
```

Figure 2.17: Illustration of the input and output in the protein-protein interaction binding sites prediction task. From customequinenutrition.com

In recent years machine learning made major contributions in PPI binding site prediction, and in other bioinformatics problems. Before these computational methods were developed experimental methods were used which was time consuming and expensive. In this section we will first talk about one algorithmic based program and then will come into recent machine learning solutions for predicting binding sites. It has been observed that at many of the research uses position-specific scoring matrix (PSSM), evolutionary conservation (ECO), putative relative solvent accessibility (RSA) [41], high-scoring segment pair (HSP) [20], one-hot encoding [38] or embedding [2] as feature group to represent a protein sequence. There are several learning structures like random forest, support vector machine [35], logistic regression [40] or neural network [38] that have been used to solve PPI binding sites classification problem.

2.5.1 PIPE-sites

PIPE-sites [1] is a tool for paired residue prediction, where it predicts the binding sites of a protein with another given protein. This program is based on the PPI prediction program PIPE [27] and also called a partner-specific sites predictor. The input to this program is the protein sequence and it's known interactions. The program follows a walk algorithm which is shown in Figure 2.18. First it generates a scoring matrix for each protein pair using the PIPE algorithm, and finds the peak location within that matrix. Next it extends from the peak towards all four directions until the value drops below a predefined threshold, also called percent peak. Finally, all the residues within the window are considered as binding sites. For multiple peaks the program will generate a ranked list of potential binding sites.

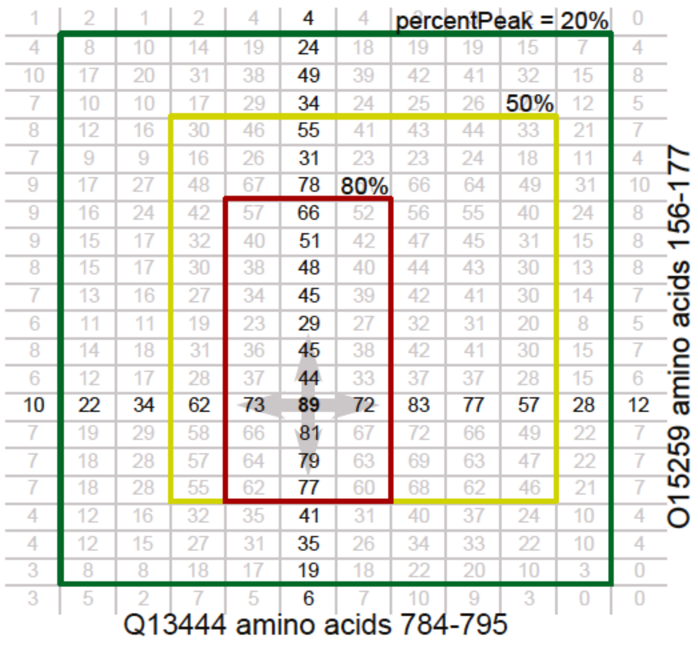


Figure 2.18: The walk algorithm of PIPE-Sites
From [1]

2.5.2 DLPred

One of the major machine learning based protein binding site prediction programs in recent years is DLPred [39]. The main architecture includes LSTM which are simplified by the authors for better time complexity and fully connected layers. In this program PSSM, physical properties, hydropathy index, physio-chemical characteristics, conservation scores and one-hot encoding is used as the input feature group. During training process it also uses a filter which picks up the sequences with high binding residue percentage helping the program to perform better with imbalanced datasets. The architecture of this program is shown in Figure 2.19

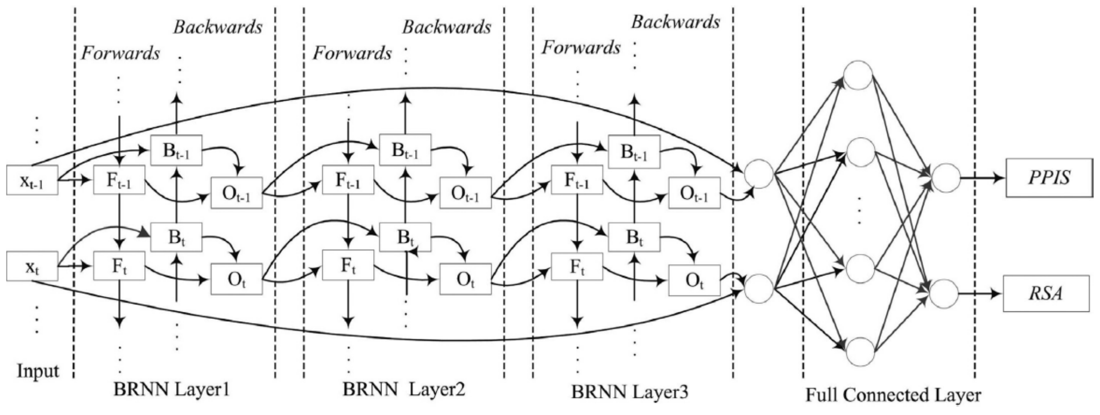


Figure 2.19: The architecture of DLPred
From [39]

2.5.3 SCRIBER

SCRIBER [40] is another top program which uses putative relative solvent accessibility (RSA), evolutionary conservation (ECO), relative amino acid propensity (RAAP), selected relevant physiochemical properties, putative secondary structure (SS) as the feature group. The main layer of this model uses a simple logistic regression which is trained on five different tasks they are: DNA-binding, RNA-binding, ligand-binding, other-binding and protein-binding. In the next layer the output of these five are combined together to make the final prediction of protein-protein binding sites. The complete architecture is provided in Figure 2.20

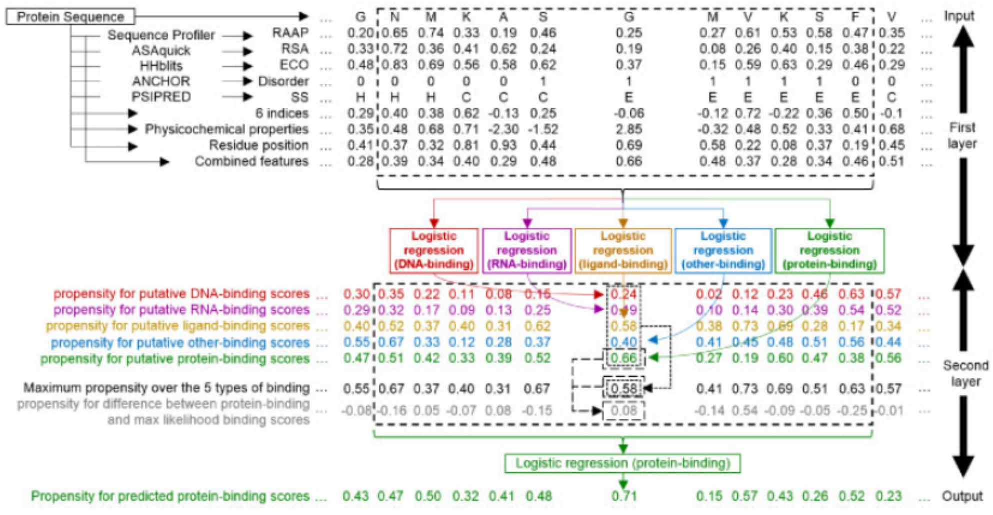

Protein Sequence
Sequence Profiler
ASAquick
HHblits
ANCHOR
PSIPRED
RAAP
RSA
ECO
Disorder
SS
6 indices
Physicochemical properties
Residue position
Combined features
Input
First layer
Logistic regression (DNA-binding)
Logistic regression (RNA-binding)
Logistic regression (ligand-binding)
Logistic regression (other-binding)
Logistic regression (protein-binding)
propensity for putative DNA-binding scores
propensity for putative RNA-binding scores
propensity for putative ligand-binding scores
propensity for putative other-binding scores
propensity for putative protein-binding scores
Maximum propensity over the 5 types of binding
propensity for difference between protein-binding and max likelihood binding scores
Second layer
Logistic regression (protein-binding)
Propensity for predicted protein-binding scores
Output

Figure 2.20: The architecture of SCRIBER
From [40]

2.5.4 DeepPPISP

Another famous ML based approach to predict PPI binding sites in recent years is DeepPPISP [38]. This also uses secondary structure information. The architecture shown in Figure 2.21 includes a Text CNN, a variation of convolution layer used for feature extraction and fully connected layer for predicting binding residues.

The interesting fact about this program is it uses both local and global information. For each amino acid in the protein sequence, local information is the neighbouring 7 amino acids and global information consist of 500 neighbouring residues, which has been chosen empirically and shown that it helps the prediction largely.

In this program one hot encoding, PSSM and secondary structure information are combined together as input feature. The local features simply consist of the input features and in global features the input vectors are further passed to a Text-CNN layer to extract information and finally both local and global information are combined together to make the final prediction.

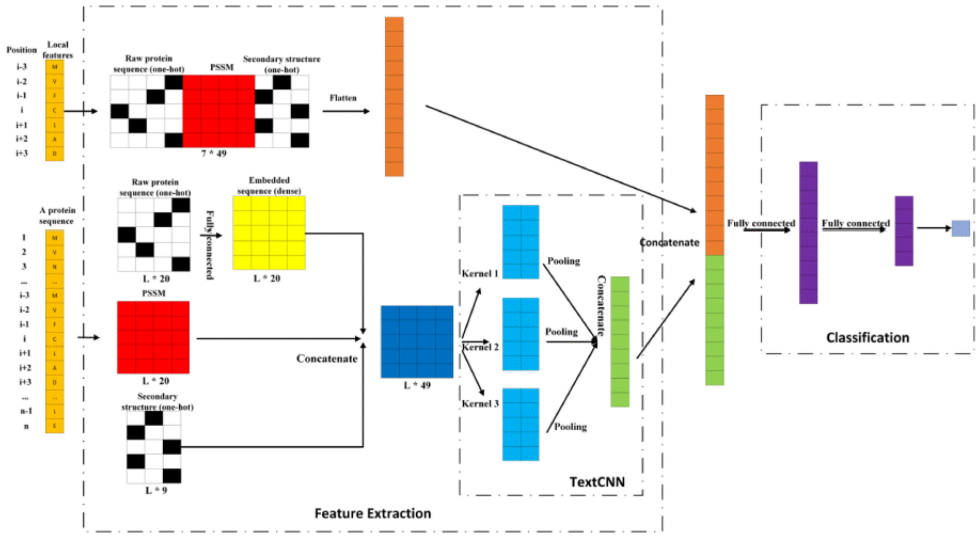


Figure 2.21: The architecture of DeepPPISP
From [38]

Chapter 3

Methodology

In this chapter we present our techniques for improving the prediction of binding sites of a protein sequence. Our main work focuses on several techniques we applied on top of the currently best program DELPHI [19] aiming for improvement. So, first we will describe DELPHI in detail and then move on to our improvements.

3.1 DELPHI

Protein-protein interaction and binding site prediction is one of the key problems in bioinformatics which has been explored by many researchers in the last few years. In machine learning perspective it can be defined as a classification problem where each amino acid in a protein sequence comes under two classes i.e binding and non-binding. DELPHI is a neural network based model which outperforms the state-of-the-art methods for binding sites prediction. The most interesting fact about the DELPHI architecture is that it uses ensemble learning which is a combination of multiple classifiers(in this case CNN and RNN) for prediction. In the real world most of the protein binding datasets are imbalanced as the binding residue percentage is much lower than non-binding residues so, this can also be called as an imbalanced dataset problem.

3.1.1 Data Preparation

Datasets are one of the most important parts of any machine learning problem and there are many efficient ways to prepare this. Data can be found in many formats such as text, images or other raw formats. In order to feed them into a machine they need to be transformed in certain manner. It is often very likely that machines will not understand raw formatted data, and one common technique used for this purpose is featurization.

In this problem we use fasta format data to represent each protein sequence, a discussion on this specific format has already been done in the previous chapter. DELPHI accepts multi sequence fasta format for its input, where each proteins has three lines of information, they are: protein ID, protein sequence and known interactions(1: interacting; 0: non-interacting).

3.1.2 Features

DELPHI uses 12 feature groups to represent each amino acid within a protein sequence, a list of which is shown in Table 3.1. Features are the most powerful tool in any data modeling problem, a good feature can easily improve the performance of a model. Keeping this in mind DELPHI introduces three new features high-scoring segment pair (HSP) [20], 3-mer amino acid embedding(ProtVec1D) [2] and position information which was never used before in binding site prediction problems. The rest 9 features have already been used by other researchers in similar problems. In DELPHI we compute all 12 features having a total dimension of size 39 for each amino acid within a protein sequence. So, a protein of length L will have a $L \times 39$ shape 2D matrix as a input sample. A brief definition of each feature is as follows:

Feature	Program	Dimension
High-scoring segment pair (HSP)	SPRINT and compute	1
3-mer amino acid embedding (ProtVec1D)	Load and compute	1
Position information	Compute	1
Position-specific scoring matrix (PSSM)	Psi-Blast	20
Evolutionary conservation (ECO)	Hhblits	1
Putative relative solvent accessibility (RSA)	ASAquick	1
Relative amino acid propensity (RAA)	Load	1
Putative protein-binding disorder	ANCHOR	1
Hydropathy index	Load	1
Physicochemical characteristics	Load	3
Physical properties	Load	7
PKx	Load	1

Table 3.1: The feature groups used by DELPHI
From [19]

HSP(High Scoring Pair) is a pair of similar sub-sequences between two proteins. The similarities between two sub-sequences of the same length are measured by scoring matrices such as PAM and BLOSUM. SPRINT [20] is used for computing all HSPs as it detects similarities fast and accurately among all proteins in training and testing.

ProtVec is a feature developed based on [2]. It uses word2vec [22] to construct a one hundred dimensional embedding for each amino acid 3-mer; we discuss word embeddings in the next section. ProtVec can be applied to problems such as protein family classification, protein visualization, structure prediction, disordered protein identification, and protein-protein interaction prediction.

Position information is shown to be useful in natural language processing tasks. The position information of each amino acid is used as an input feature hoping that it provides certain global information of a protein. The position of an amino acid in a protein is in the range of 1 to the length of the protein. Then the position is divided by protein's length so that the value is between 0 to 1.

PSSM (Position Specific Scoring Matrices) are widely used in protein interaction related problems. They contain the evolutionary conservation of each amino acid position by aligning an input sequence with protein databases. Each row of a PSSM represents an amino acid symbol and each column represents a position in the sequence. In order to construct a PSSM first we need to create a Position Frequency Matrix (PFM), counting the occurrences of each nucleotide at each position. For example, given the following DNA sequences:

```
GAGGTAAAC
TCCGTAAGT
CAGGTTGGA
ACAGTCAGT
TAGGTCATT
TAGGTACTG
ATGGTAACT
CAGGTATAC
TGTGTGAGT
AAGGTAAGT
```

The corresponding PFM is:

$$M = \begin{array}{c} A \\ C \\ G \\ T \end{array} \begin{bmatrix} 3 & 6 & 1 & 0 & 0 & 6 & 7 & 2 & 1 \\ 2 & 2 & 1 & 0 & 0 & 2 & 1 & 1 & 2 \\ 1 & 1 & 7 & 10 & 0 & 1 & 1 & 5 & 1 \\ 4 & 1 & 1 & 0 & 10 & 1 & 1 & 2 & 6 \end{bmatrix}$$

Next, we need to compute a Position Probability Matrix (PPM) by dividing the nucleotide count at each position of the PFM by the number of sequences. Given a set X of N aligned sequences of length l, the elements of the PPM are calculated as:

$$M_{k,j} = \frac{1}{N} \sum_{i=1}^{N} X_{i,j}$$

Where $i \in (1, ..., N)$, $j \in (1, ..., l)$ and k is the set of amino acids. Therefore, the resulting PPM is:

$$M = \begin{array}{c} A \\ C \\ G \\ T \end{array} \begin{bmatrix} 0.3 & 0.6 & 0.1 & 0.0 & 0.0 & 0.6 & 0.7 & 0.2 & 0.1 \\ 0.2 & 0.2 & 0.1 & 0.0 & 0.0 & 0.2 & 0.1 & 0.1 & 0.2 \\ 0.1 & 0.1 & 0.7 & 1.0 & 0.0 & 0.1 & 0.1 & 0.5 & 0.1 \\ 0.4 & 0.1 & 0.1 & 0.0 & 1.0 & 0.1 & 0.1 & 0.2 & 0.6 \end{bmatrix}$$

The elements in PSSMs are calculated as log likelihoods using a background model b, where $b_k = 1/|k|$ i.e. the elements of a PPM are transformed as:

$$M_{k,j} = log_2(M_{k,j}/b_k)$$

Applying this transformation to the above PPM produces the PSSM:

$$M = \begin{array}{c} A \\ C \\ G \\ T \end{array} \begin{bmatrix} 0.26 & 1.26 & -1.32 & -\infty & -\infty & 1.26 & 1.49 & -0.32 & -1.32 \\ -0.32 & -0.32 & -1.32 & -\infty & -\infty & -0.32 & -1.32 & -1.32 & -0.32 \\ -1.32 & -1.32 & 1.49 & 2.0 & -\infty & -1.32 & -1.32 & 1.0 & -1.32 \\ 0.68 & -1.32 & -1.32 & -\infty & 2.0 & -1.32 & -1.32 & -0.32 & 1.26 \end{bmatrix}$$

ECO also contains evolutionary conservation, but in a more compact way. To compute this score, the faster multiple alignment tool HHBlits [30] is run against the non-redundant Uniprot20 database with default parameters.

Putative relative solvent accessibility(RSA) is predicted using ASAquick [11]. The values are obtained in the from rasaq.pred file in each output directory.

Relative amino acid propensity(RAA) for binding is quantified as relative difference in abundance of a given amino acid type between binding residues and the corresponding non-binding residues located on the protein surface.

The putative protein-binding disorder is computed using the ANCHOR program [10].

Hydrophobicity scales is experimentally determined transfer free energies for each amino acid. It contains energetic information of protein bi-layer interactions [37].

Physicochemical characteristics includes three features for each protein: the number of atoms, electrostatic charges and potential hydrogen bonds for each amino acid.

Physical properties are a 7-dimensional property of each amino acid type is used. They are a steric parameter (graph-shape index), polarizability, volume (normalized van der Waals volume), hydrophobicity, isoelectric point, helix probability and sheet probability.

PKx is the negative of the logarithm of the dissociation constant for any other group in the molecule.

3.1.3 Feature Normalization

After computing all the feature vectors, the values in in each row vector are normalized to a number between 0 to 1 using Equation 3.1 where *v* is the original feature value, and *max* and *min* are the biggest and smallest value observed in the training dataset, respectively. This is to ensure each feature group are of the same numeric scale and help the model converge better.

$$v_{norm} = \frac{v - min}{max - min} \tag{3.1}$$

3.1.4 Model Architecture

The architecture of DELPHI is inspired by ensemble learning. The intuition behind this is by training different classifier the model will learn different information and finally combining it together helps it to perform better. DELPHI uses a CNN based classifier and a RNN based classifier placed in parallel and their output layer is combined to obtain the final prediction. At the input layer a sliding window is being used to capture the input window of each amino acid for a protein sequence. The window will slide until the end of that sequence keeping the centre element as the pointed amino acid. In DELPHI a window size of 31 is chosen for this purpose experimentally which results a 31×39 size input feature vector for each amino acid. This mechanism is shown in Figure 3.1. The CNN and the RNN module has several components and certain training parameters, which is discussed in the next section.

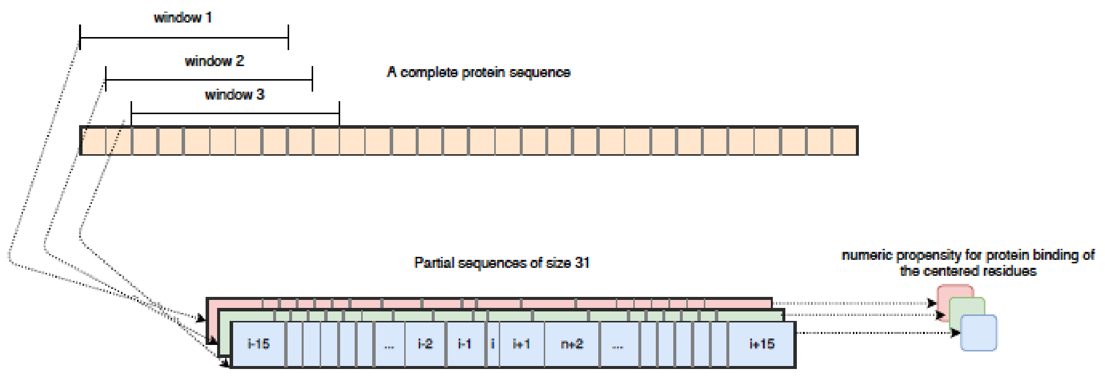

window 1
window 2
window 3
A complete protein sequence
Partial sequences of size 31
numeric propensity for protein binding of the centered residues
i-15
...
i-2
i-1
i
i+1
n+2
...
i+15

Figure 3.1: The input mechanism of DELPHI
From [19]

The CNN module

The CNN module of DELPHI is made of 4 consecutive layers. At first, the input matrix is passed to a convolution layer with a filter of size 5×5 having a stride of 1 with valid padding. Then a max pooling is done on top of that, the output of which is flatten and then passed to a fully connected layer made of 64 neurons with dropout. This model is trained for 8 epochs on the entire training data with a batch size of 1024; see Figure 3.2

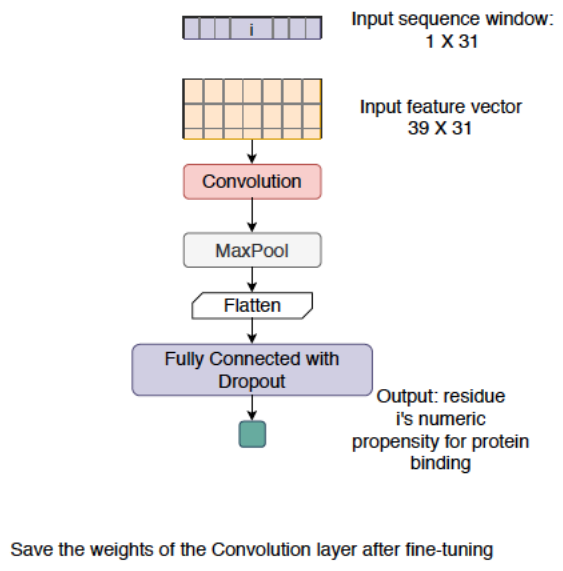

i
Input sequence window: 1 X 31
Input feature vector 39 X 31
Convolution
MaxPool
Flatten
Fully Connected with Dropout
Output: residue i's numeric propensity for protein binding
Save the weights of the Convolution layer after fine-tuning

Figure 3.2: Detailed design of the CNN module
From [19]

The RNN module

The RNN module of DELPHI is build using three layers. Each input vector is passed to a bidirectional GRU layer which is made of 32 hidden units in each direction. So, in total the bidirectional layer has $32 \times 2 = 64$ hidden units. The output from that is flatten first then passed on to a fully connected layer with 64 neurons. This model is trained for 9 epochs with the same batch size used for CNN module; see Figure 3.3

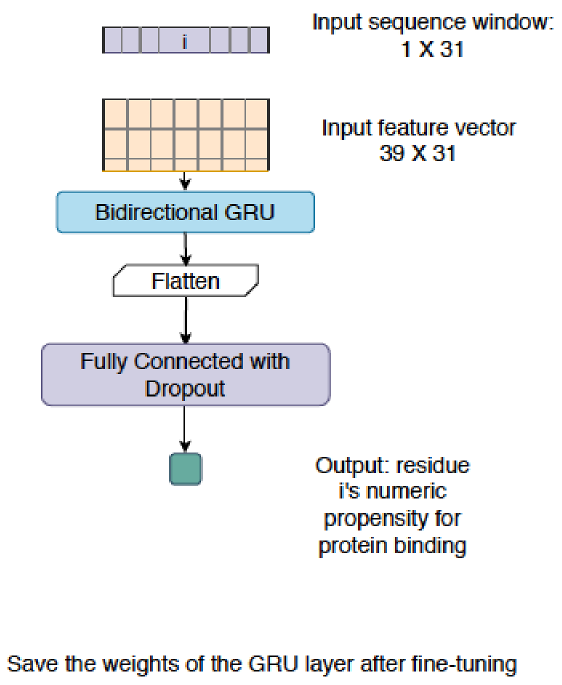

i
Input sequence window: 1 X 31
Input feature vector 39 X 31
Bidirectional GRU
Flatten
Fully Connected with Dropout
Output: residue i's numeric propensity for protein binding
Save the weights of the GRU layer after fine-tuning

Figure 3.3: Detailed design of the RNN module
From [19]

The Ensemble network

As discussed earlier the ensemble network used in DELPHI is the combination of both the CNN and RNN modules. The interesting fact is that the CNN and RNN modules are trained separately and saved their weight. After the first phase is done we load these weights in the ensemble network and only train the last fully connected layer which combines both modules. Experimentally it has been found that this approach performs better than an end-to-end training of the ensemble model. The fully connected layer is made of 96 neurons in total. During training we used only 5 epochs to train this model which was decided using early stopping; see Figure 3.4 for an illustration of the ensemble model.

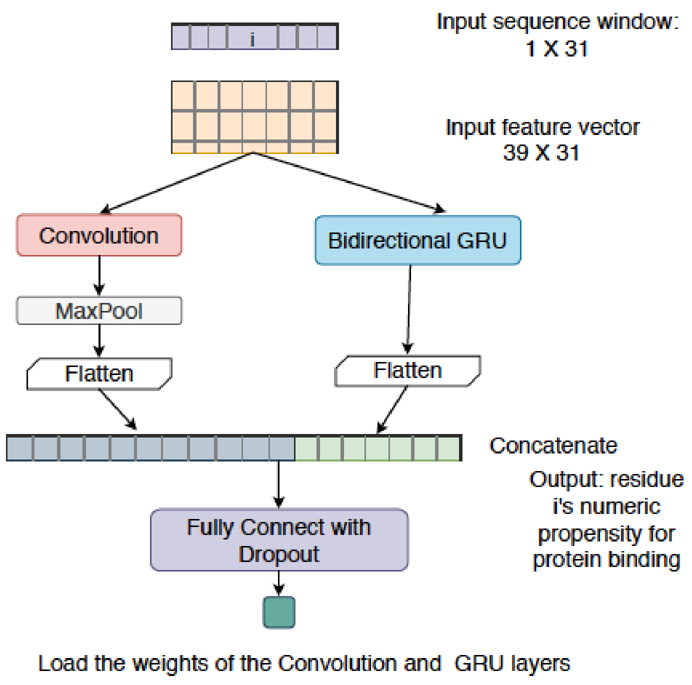

Input sequence window: 1 X 31
i
Input feature vector 39 X 31
Convolution
Bidirectional GRU
MaxPool
Flatten
Flatten
Concatenate
Output: residue i's numeric propensity for protein binding
Fully Connect with Dropout
Load the weights of the Convolution and GRU layers

Figure 3.4: An illustration of the ensemble network
From [19]

3.1.5 Parameters and Hyper-parameters

The parameters used in DELPHI are already discussed in the last section such as number of epochs, kernel size, hidden unit size etc. These are chosen in order to maximize the area under precision-recall curve (AUPRC) of the training data. The total training time for DELPHI is 4-8 hours depending on the machine configuration.

There are several hyper-parameters like batch size, learning rate, etc. used in DELPHI, which are described in Table 3.2

Hyper-parameter	Value
Batch Size	1024
Dropout Rate	0.3
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Patience in early stop	4
Loss Function	Binary Cross-entropy
Learning Rate	0.002

Table 3.2: The hyper-parameters used in DELPHI
From [19]

3.2 Methods for Improvement

In this section we will talk about the methods we use to improve the prediction performance of DELPHI. The primary target was to replace the existing protein embedding ProtVec with some pre-trained contextual embedding in order to get more accurate predictions. Embeddings are one of the most powerful techniques used in NLP, it vectorizes a word into a feature space. In ProtVec they uses word2vec [22] which is context independent, whereas in our new approach we introduced several ideas of contextual embeddings [26] [8] [21]. Later, we also tried some latest NLP techniques such as self-attention and positional encoding [34]. A detailed discussion on all these methods are presented next.

3.2.1 Word2Vec

Since words are interpreted as indices in a vocabulary, many existing NLP systems and techniques regard them as atomic units. Simple models that are trained on a large amount of data outperform complex systems that are trained on a smaller amount of data, such as the N-gram model. Many functions, such as automatic speech recognition (which is controlled by the size of high-quality transcribed speech data), and machine translation (the existing corpora for many languages contain only a few billions of words or less), are beyond the capabilities of simple techniques. With the development of machine learning techniques in recent years, neural network-based language models have made it possible to train more complex models on much larger data sets.

Word2Vec [22] describes strategies for extracting high-quality word vectors from large data sets. Not only would identical terms appear to be close to each other in vector representations, but they will also have several degrees of similarity. Word representations are similar in more ways than just syntactic regularities. e.g. vector("King") - vector("Man") + vector("Woman") results closest to vector("Queen").

Model Architecture

Most NLP researchers uses Latent Semantic Analysis (LSA) [7] and Latent Dirichlet Allocation (LDA) [5] for estimating continuous representations of words. Because of its recent advancement, the main focus of word2vec was on distributed representations of words learned by neural networks. Equation 3.2 defines the training complexity of this technique, where E denotes the number of training epochs, T denotes the number of words in the training set, and Q is defined further for each model architecture. Back-propagation and stochastic gradient descent are used to train all models.

$$O = E \times T \times Q \tag{3.2}$$

Feedforward Neural Net Language Model (NNLM)

The first probabilistic feed-forward neural network language model, consisting of an input layer, a projection layer, a hidden layer, and output layers, was demonstrated in [4]. *N* previous words are encoded using one-hot encoding at the input layer, then projected using a shared projection matrix to a projection layer P with dimensionality $N \times D$. Since the values in the projection layer are dense, this method is a little more difficult to compute. Equation 3.3 shows the computational complexity for each training example where H and V are the hidden layer and vocabulary size. The number of output units that must be evaluated can be reduced to about $log_2(V)$ with binary tree representations of the vocabulary.

$$Q = N \times D + N \times D \times H + H \times V \tag{3.3}$$

The detailed architecture of this probabilistic language model is shown in Figure 3.5

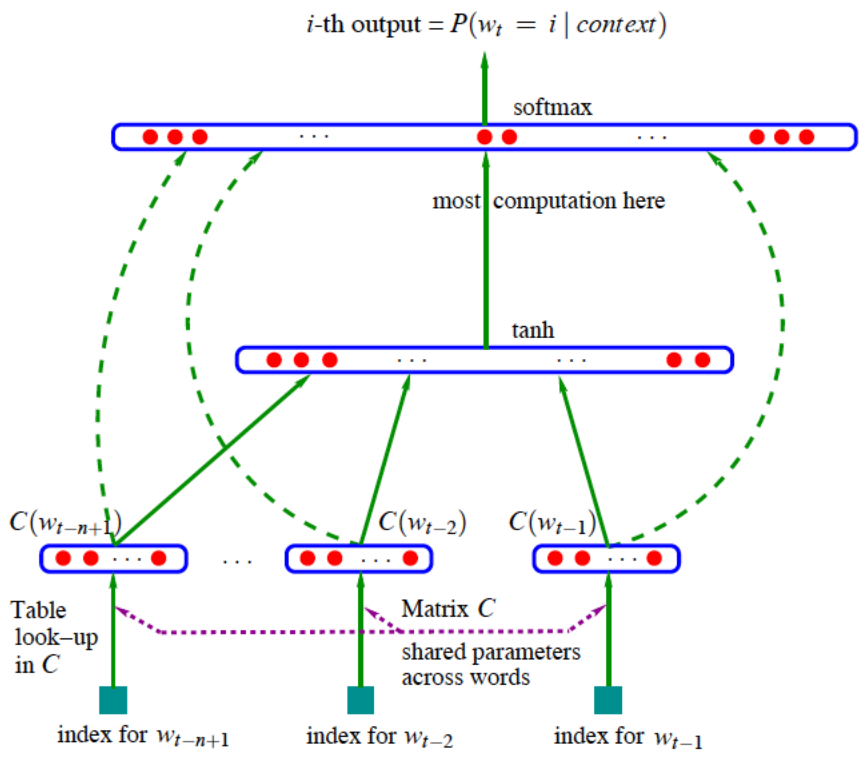


Figure 3.5: The architecture of neural probabilistic language model
From [4]

New Log-linear Models

Two new log-linear model ideas were presented in [22]. They are the continuous bag-of-words (CBOW) and skipgram models, respectively. The CBOW model is similar to the feed-forward NNLM in that the non-linear hidden layer is removed and the projection layer for all words is shared, resulting in all words being projected into the same location. Since the order of words in the background has no impact on the projection, this architecture is known as the continuous bag-of-words model. Equation 3.4 shows the training complexity for this.

$$Q = N \times D + D \times log_2(V) \tag{3.4}$$

Instead of predicting the current word based on context, the Continuous Skipgram Model seeks to optimise classification of a word based on another word in the same sentence. It predicts words within a certain range before and after the current word, using each current word as an input to a log-linear classifier with continuous projection layer. Equation 3.5 shows the training complexity of this architecture, where C is the maximum word distance.

$$Q = C \times (D + D \times log_2(V)) \tag{3.5}$$

An illustration of CBOW and skip-gram is shown in Figure 3.6

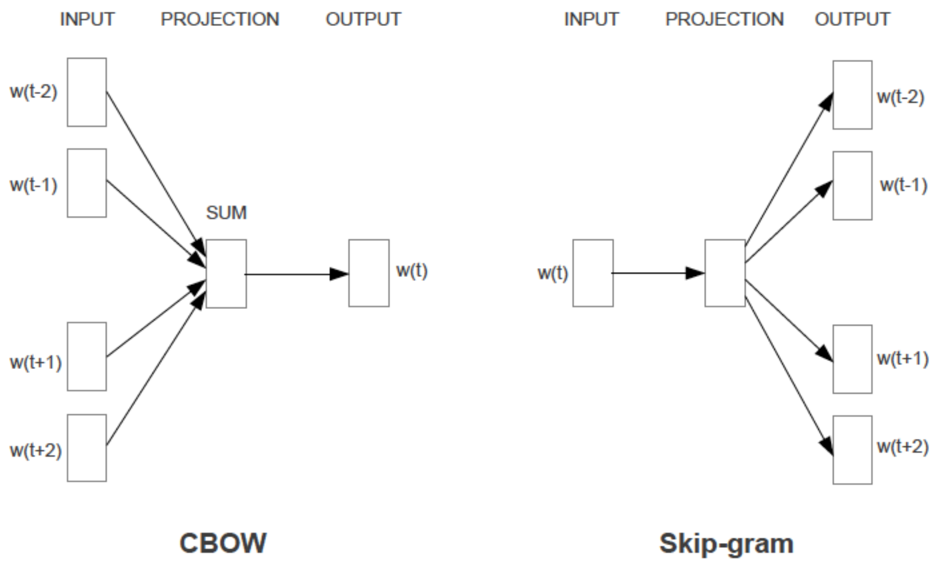


Figure 3.6: The architecture of CBOW and skip-gram model
The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. From [22]

3.2.2 ProtVec

In this section we will discuss in detail about ProtVec [2] one of the most important features used in DELPHI for the first time in any protein binding site prediction problem according to the best of our knowledge. This is a new approach for representing biological sequences in bioinformatics applications such as family classification, protein visualisation, structure prediction, etc.

ProtVec uses word2vec, which was, at the time, state-of-the-art for training word vector representations. To train the embedding vectors it considers a vector of size 100 and a context of size 25. The training data is prepared from Swiss-Prot which is a rich protein database, using the simplest and most common technique involving fixed length overlapping n-grams. This generates 3 lists of shifted non-overlapping words, as shown in Figure 3.7, each 3-gram (a "biological" word) is presented as a vector of size 100.

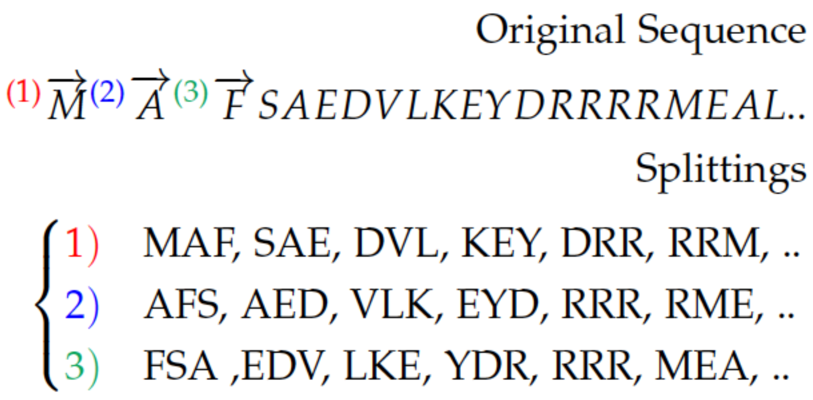

Original Sequence
(1) M (2) A (3) F SAEDVLKEYDRRRRMEAL..
Splittings
1) MAF, SAE, DVL, KEY, DRR, RRM, ..
2) AFS, AED, VLK, EYD, RRR, RME, ..
3) FSA ,EDV, LKE, YDR, RRR, MEA, ..

Figure 3.7: Training data for ProtVec
From [2]

3.2.3 ELMo

ELMo [26] is a contextualized word embedding that depends on the entire input sentence, unlike word2vec which is context independent. The model architecture consists of two-layer bidirectional language models (biLMs) with character convolutions. It allows to perform semi-supervised learning where, the biLM is pre-trained on a large dataset first, and then fine-tuned on variety of NLP tasks.

For a sequence of N tokens, $(t_1, t_2, ..., t_N)$ a forward language model computes the probability of the sequence by modelling the probability of token t_k given the history $(t_1, ..., t_{k-1})$ as in Equation 3.6

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1}) \tag{3.6}$$

Before ELMo, most of the recent language models compute a fixed context-independent token representation x_k^{LM} and projects it to a L layer forward LSTM. Each LSTM layer generates a context-dependent representation $\overrightarrow{h}_{k,j}^{LM}$ where $j = 1, ..., L$ for every position k. The next token t_{k+1} is predicted based on the final layer LSTM output, passed through a softmax layer. Similarly, for a backward language model the sequence is processed in reverse. This predicts the previous token given the future context; see Equation 3.7

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ..., t_N) \tag{3.7}$$

A backward LSTM can be implemented with a layer j in a L layer deep model producing representations $\overleftarrow{h}_{k,j}^{LM}$ of t_k given $(t_{k+1}, ..., t_N)$. A bidirectional language model combines both a forward and backward LM. This formula jointly maximizes the loglikelihood of the forward and backward directions with parameters for both the token representation and softmax layer. It also uses separate parameters for the LSTMs in each direction; see Equation 3.8

$$\sum_{k=1}^{n}(\log p(t_k|t_1,...t_{k-1};\theta_x,\overrightarrow{\theta}_{LSTM},\theta_s)+\log p(t_k|t_{k+1},...t_N;\theta_x,\overleftarrow{\theta}_{LSTM},\theta_s)) \tag{3.8}$$

ELMo is a task specific combination of the intermediate layer representations in the biLM. A L-layer biLM computes a set of $2L + 1$ representations for each token t_k, shown in Equation 3.9

$$R_k = \{x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, ..., L\} = \{h_{k,j}^{LM} | j = 0, ..., L\} \tag{3.9}$$

Where the token layer is $h_{k,0}^{LM}$ and each biLSTM layer is defined as $h_{k,j}^{LM} = [\overrightarrow{h}_{k,j}^{LM}; \overleftarrow{h}_{k,j}^{LM}]$. ELMo combines all layers in R into a single vector, to include it into downstream model, where $ELMo_k = E(R_k;\theta_e)$. In the simplest case, ELMo just selects the top layer, $E(R_k) = h_{k,L}^{LM}$. More generally, it computes a task specific weighting of all biLM layers: $ELMo_k^{task} = E(R_k;\theta^{task}) = \gamma^{task}\sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}$, here s^{task} are softmax-normalized weights, γ^{task} is a scalar parameter that allows the task model to scale the entire ELMo vector, γ helps in the optimization process. Since the activation of each biLM layer have a different distribution, in some cases it also useful to apply normalization to each biLM layer before weighting.

3.2.4 SeqVec

SeqVec [13] is a novel embedding of protein sequences derived from large, unlabeled sequence data with bi-directional language model ELMo. The hidden states of the forward and backward pass of each LSTM-layer are concatenated to a 1024-dimensional embedding vector summarizing information from the left and the right context. This has the advantage that two identical words can have different embeddings, depending on the words surrounding them.

The architecture of SeqVec is shown in Figure 3.8 and it says that, an input sequence is padded with special tokens first, then character convolutions map each word onto a fixed-length latent space. The bidirectional Long Short Term Memory introduces context-specific information by processing the sentence sequentially. The second LSTM-layer tries to predict the next word given all previous words in a sentence.

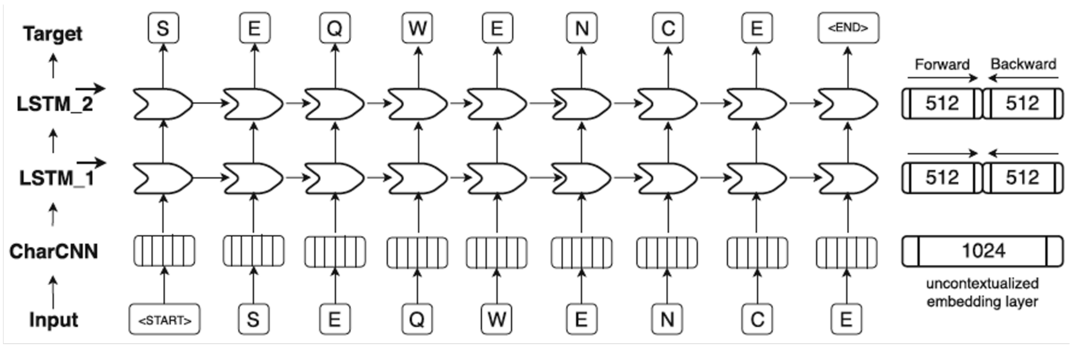


Figure 3.8: The model architecture of SeqVec
From [13]

3.2.5 BERT

Bidirectional Encoder Representations from Transformers (BERT) [8] is a new language representation model. Before BERT, there were two existing strategies for applying pre-trained language representations to downstream tasks they are, feature-based(ELMo) [26] which uses task-specific architectures and fine-tuning(GPT) [28] introduces minimal task-specific parameters. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. BERT alleviates the previously mentioned unidirectionality constraint by using a "masked language model" (MLM) pre-training objective, which enables the representation to fuse the left and the right context. During pre-training, the model is trained on unlabelled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labelled data from the downstream tasks. The overall pre-training and fine-tuning procedure of BERT is shown in Figure 3.9

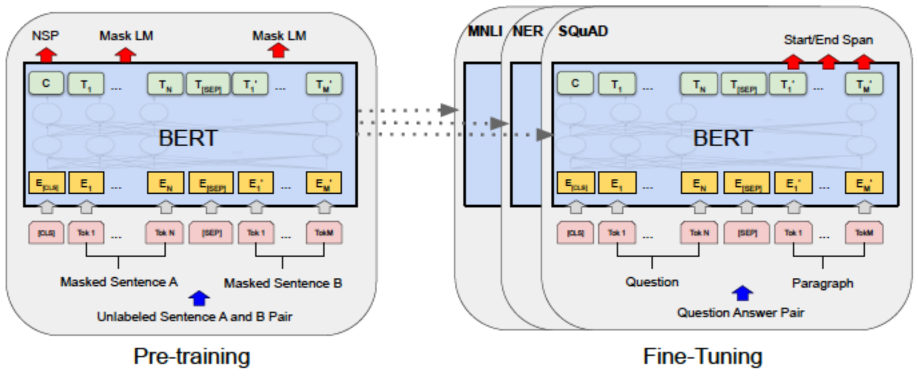


Figure 3.9: The overall pre-training and fine-tuning procedure of BERT
From [8]

Model Architecture

The model architecture of BERT is a multi-layer bidirectional Transformer encoder, which is based on the original implementation described in [34] and released in the *tensor2tensor* library. The number of layers (i.e., Transformer blocks) is denoted by L, the hidden size as H, and the number of self-attention heads as A. See Equation 3.10 and Equation 3.11

$$BERT_{BASE}(L = 12, H = 768, A = 12, TotalParameters = 110M) \tag{3.10}$$

$$BERT_{LARGE}(L = 24, H = 1024, A = 16, TotalParameters = 340M) \tag{3.11}$$

In the literature the bidirectional Transformer is often referred to as a "Transformer encoder" while the left-context-only version is referred to as a "Transformer decoder" since it can be used for text generation.

Input-Output Representations

BERT can handle a variety of down-stream tasks for which the input representation is both single sentence and a pair of sentences (e.g. Question, Answer) in one token sequence. A "sentence" can be an arbitrary span of contiguous text whereas a "sequence" may be a single sentence or two sentences packed together. It uses word-piece embedding [15] with a 30,000 token vocabulary. The first token of every sequence is always a special classification token ([CLS]) used as the aggregate sequence representation for classification tasks. A sentence pair (A,B) is packed into a single sequence separated with a special token ([SEP]) and a learned embedding is added to every token indicating whether it belongs to sentence A or sentence B. The notation for input embedding is E, the final hidden vector of the special [CLS] token is $C \in R^H$ and and the final hidden vector for the i^{th} input token is $T_i \in R^H$. For the illustration of input-output representation; see Figure 3.10

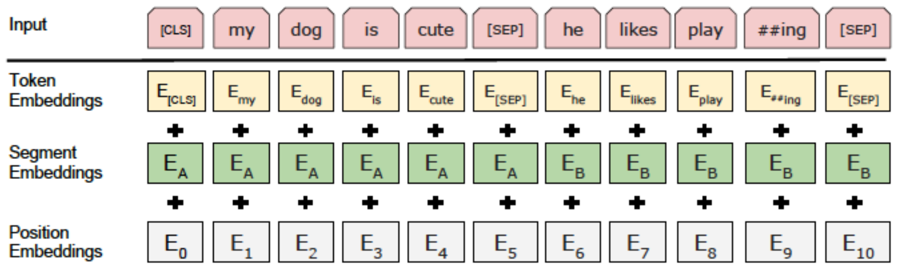


Figure 3.10: An illustration for the input-output representation of BERT
From [8]

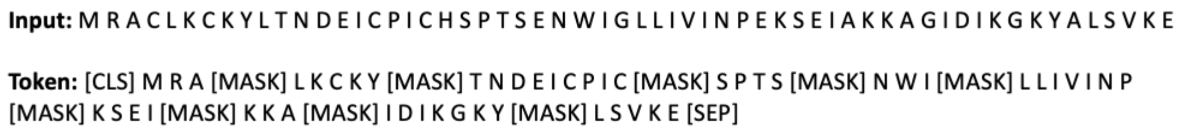
Pre-training and Fine-tuning

The pre-training of BERT includes two unsupervised tasks they are masked language model (MLM) and next sentence prediction (NSP). The MLM is a bidirectional conditioning that allows each word to indirectly "see itself", and the model could trivially predict the target word in a multi-layered context. First it masks 15% of all word-piece tokens in each sequence at random, the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard LM. Next, each token T_i is used to predict the original token with cross entropy loss. In NSP, to train a model that understands sentence relationships, a binarized next sentence prediction task is used for pre-training that can be trivially generated from any monolingual corpus.

In fine-tuning, for each task we simply plug in the task specific inputs and outputs into BERT and finetune all the parameters end-to-end. At the input, sentence A and sentence B from pre-training are analogous to sentence pairs in paraphrasing, hypothesis-premise pairs in entailment or question-passage pairs in question answering. At the output, the token representations are fed into an output layer for token level tasks. The [CLS] representation is fed into an output layer for classification.

3.2.6 Pre-training and Feature Extraction for Proteins

In our method we introduced a new protein embedding which is obtained from a pre-trained model of BERT. As per the present implementation of DELPHI which uses 1D ProtVec for each amino acid in a protein sequence, our pre-trained BERT model also learns the embedding for each 1-mer. The pre-training dataset we used here is an updated version of the Swiss-Prot database which was used in ProtVec. First, we split each protein sequence into 1-mer representation by adding a space between each amino acid. This way each protein sequence is considered as a sentence and each amino acid is considered as a word within that sentence which further fed into the BERT model. The input representation is shown in Figure 3.11

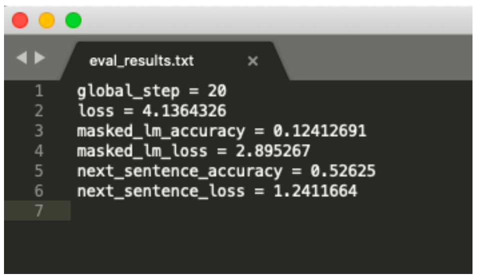


Input: M R A C L K C K Y L T N D E I C P I C H S P T S E N W I G L L I V I N P E K S E I A K K A G I D I K G K Y A L S V K E

Token: [CLS] M R A [MASK] L K C K Y [MASK] T N D E I C P I C [MASK] S P T S [MASK] N W I [MASK] L L I V I N P [MASK] K S E I [MASK] K K A [MASK] I D I K G K Y [MASK] L S V K E [SEP]

Figure 3.11: The input representation of BERT for Proteins

The pre-training takes almost 8-10 hours on 5M protein sequences in a GPU machine with 64GB of memory in our SHARCNET cluster. The output result of this is reported in Figure 3.12



```
global_step = 20
loss = 4.1364326
masked_lm_accuracy = 0.12412691
masked_lm_loss = 2.895267
next_sentence_accuracy = 0.52625
next_sentence_loss = 1.2411664
```

Figure 3.12: The evaluation of pre-training BERT on Swiss-Prot dataset

Feature Extraction

In certain cases, rather than fine-tuning the entire pre-trained model end-to-end, it can be beneficial to obtain pre-trained contextual embeddings, which are fixed contextual representations of each input token generated from the hidden layers of the pre-trained model. These vectors can be captured from each hidden layer in the model, in our purposes we only consider the last layer. Since the embedding dimension is 768 and we will be using only 1D features, we simply add all of them into 1 dimension. They are further normalized using the same principle used in DELPHI given in Equation 3.1. The original implementation of BERT produce a javascript object notation (json) formatted file to store all the embedding vectors. In our case, we had to process it further in order to compute the embeddings in our desired format.

Parameters and Hyper-Parameters

The parameters and hyper-parameters used in pre-training is defined inside "bert_config.json" file. A list of these attributes are provided in Table 3.3

Parameter	Value
Attention dropout probability	0.1
Hidden activation	gelu
Hidden dropout probability	0.1
Hidden size	768
Initializer range	0.02
Intermediate size	3072
Maximum position embedding	2048
Number of attention head	12
Number of hidden layers	4
Type vocab size	2
Vocab size	30

Table 3.3: The parameters and hyper-parameters used in pre-training

The hidden size defines the dimension of the embedding vector that will be produced by BERT and the maximum position embedding is used to specify the highest length of the sequence the model will see during training and inference.

3.2.7 RoBERTa

BERT was found to be substantially under-trained, and a new recipe for training BERT models named RoBERTa [21], was proposed, which can equal or even outperform all of the post-BERT methods. The goal was achieved by training the model for longer periods of time with larger batches of data, eliminating the next sentence prediction target, training on longer sequences, and dynamically modifying the masking pattern applied to the training data.

BERT works by masking and predicting tokens at random. Masking was done only once during data pre-processing in the original BERT implementation, resulting in a single static mask. To avoid using the same mask for each training instance in each epoch, training data was duplicated 10 times, resulting in each sequence being masked in 10 different ways over the 40 training epochs.

In the BERT pre-training process, the model is trained to predict if the observed document segments come from the same or different documents using Next Sentence Prediction (NSP) loss, in addition to the masked language modelling objective. Whereas, it has been observed that training without the NSP loss and training with blocks of text from a single document outperforms the originally published $BERT_{BASE}$ results and that removing the NSP loss matches or slightly improves downstream task performance.

Previous work in Neural Machine Translation has shown, when the learning rate is increased appropriately, training with very large mini-batches will boost both optimization speed and end-task efficiency. This technique improves the accuracy for the masked language modelling objective.

Byte-Pair Encoding (BPE) is a combination of character and word-level representations that enables natural language corpora to handle broad vocabularies. BPE uses subword units instead of complete words, which are derived by statistical analysis of the training corpus. Instead of the 30K character-level BPE vocabulary used in the original BERT implementation, a larger byte-level BPE vocabulary with 50K subword units is trained here.

3.2.8 PRoBERTa

A Transformer neural network, which is called Protein RoBERTa (PRoBERTa [25]), to pre-train task-agnostic vector representations of amino acid sequences. Then fine-tune these representations towards two protein prediction tasks: protein family classification and binary PPI prediction.

The architecture of PRoBERTa is inspired from BERT which consists of an embedding layer, followed by stacked Transformer encoder layers, and a final layer which constructs a task-specific output as shown in Figure 3.13. Following the RoBERTa procedure, this model generates a new masking pattern every time a sequence is fed to the model. The original BERT procedure also includes a Next Sentence Prediction (NSP) task. However, given that proteins are not made up of multiple sentences, the NSP task is considered as inappropriate for pre-training.

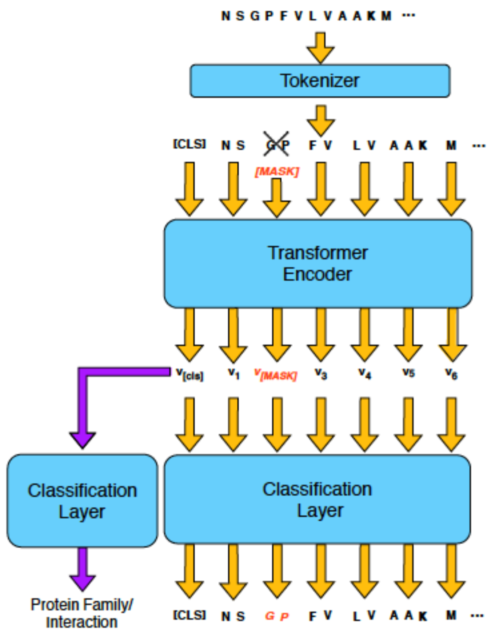

N S G P F V L V A A K M ···
Tokenizer
[CLS] N S GP F V L V A A K M ···
[MASK]
Transformer Encoder
v[cls] v1 v[MASK] v3 v4 v5 v6
Classification Layer
Classification Layer
Protein Family/ Interaction
[CLS] N S G P F V L V A A K M ···

Figure 3.13: Pre-Training and Fine-Tuning of PRoBERTa
From [25]

In our approach we generate the similar embeddings obtained from the original RoBERTa implementation as the source code of PRoBERTa was not available at that time. We took the same pre-training dataset used in BERT with the default parameters. The pre-training takes almost 14-15 hours on T4 GPU with 64GB of memory in our SHARCNET cluster.

3.2.9 Attention

A new approach [3] to compute the weighted average of all LSTM hidden states. This helps the model to learn from weighted long term dependencies i.e. the global context for the current amino acid in a sequence. The attention mechanism for processing sequential data that considers the context for each timestamp is defined by the following set of equations.

$$h_{t,t'} = tanh(x_t^T W_t + x_{t'}^T W_x + b_t) \tag{3.12}$$

$$e_{t,t'} = \sigma(W_a h_{t,t'} + b_a) \tag{3.13}$$

$$a_t = softmax(e_t) \tag{3.14}$$

$$l_t = \sum_{t'} a_{t,t'} x_{t'} \tag{3.15}$$

Here, $h_{t,t'}$ is the contribution of all the before and after recurrent units on the current unit, a_t is the attention weight and l_t is the final encoding for the time t.

In our method we used this attention layer implemented in Keras, in place of the existing Convolution or GRU layer(i.e. we compute the weighted average of the input) and created a separate branch for that. Then all three branches, CNN, RNN and attention are trained individually and their outputs are concatenated and passed through a dense layer to get the final prediction. We also compared the results obtained, by keeping either convolution or GRU along with attention branch. This experiment shows that attention works better with RNN than with CNN.

3.2.10 Positional Encoding

Positional encoding [34] is used to provide the model some information about the relative position of the words in the sentence. An embeddings is a token representation in d-dimensional space where tokens with similar meaning will be nearer to each other. However, the embeddings do not encode the relative position of words in a sentence. That is why positional encoding vector is added with an embedding vector. So once we add the positional encoding, words will be closer to each other based on the similarity of their meaning as well as their position in the sentence. The formula for calculating positional encoding is given in Equation 3.16 and Equation 3.17.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{3.16}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{3.17}$$

In our approach we computed the positional encoding for each amino acid in a protein sequence and normalized it further. We used a dimension of size 4096 and maximum sequence length of 2048. This new feature replaces the old position feature used in DELPHI aiming to improve the model performance.

3.2.11 HSP Score

The idea presented in this section is different from the above, as we do not introduce a new machine learning construct. Instead, we discuss using the HSP feature of DELPHI as a separate score, rather than a feature.

In our initial phase of experiment we used the HSP features as an input to our model. Similar to DELPHI, we use SPRINT to compute the high scoring pairs (HSP) among all proteins in training and testing set. After we get the HSPs, the score of the i^{th} residue, $P[i]$ of a test protein P, denoted by $HSP_{score}(P[i])$ is calculated. Lets say that, for P and a training protein Q we have an HSP pair (u, v), where the residue $P[i]$ is within u. Similarly let j be the position in Q that corresponds to i, such that v covers the residue $Q[j]$. Now, if $Q[j]$ is a known interacting residue, then we add the PAM120 score between $P[i]$ and $Q[j]$ to the HSP score of $P[i]$:

$$HSP_{score}(P[i]) = \sum_{Q_{label}[j]=1} max(0, PAM120(P[i], Q[j])) \tag{3.18}$$

Later, we realize that the model performance improves, if we add the HSP score with the model prediction probability. In that case, we remove the HSP feature from our model input and compute the prediction probabilities. Then add the HSP score with corresponding propensity score of an amino acid in a certain protein sequence. So, our final prediction score for the i^{th} residue, $P[i]$ will be:

$$Prediction_{final}(P[i]) = Prediction_{withoutHSP}(P[i]) + HSP_{score}(P[i]) \tag{3.19}$$

Chapter 4

Experimental Results

4.1 Testing Datasets

In order to test our program on common benchmark data, first we prepare the testing data, then training data is constructed maintaining high dissimilarity between each other.

We already have four benchmark datasets named as: Dset_72, Dset_186, Dset_164 and Dset_448 which are publicly available from previous studies [24] [9] [40] and used in DELPHI. All the four datasets have been widely used and explored by several researchers. In our study we combined all these 4 test datasets into one single test set and compared the prediction performance obtained from each of these methods. An overview of the training and testing dataset is described in Table 4.1

Dataset	Proteins	Residues	Binding	Non-Binding	Binding %
Test	870	2,04,540	29,346	1,75,194	14.34%
Train	9,982	4,254,198	427,687	3,826,511	10.05%

Table 4.1: An overview of the training and testing datasets

4.2 Evaluation Scheme

Similar to DELPHI, we also used the same set of metrics in order to evaluate our program. They are: sensitivity, specificity, precision, accuracy, F1-measure, Matthews correlation coefficient, area under receiver operating characteristic curve (AUROC) and area under precision recall curve (AUPRC). We focus more on AUROC and AUPRC because they are threshold independent and convey an overall performance measurement of a program. The formulas for calculating the other matrices are given in the below set of equations where where true positives (TP) and true negatives (TN) are the correctly predicted binding sites and non-binding sites respectively, and false positives (FP) and false negative (FN) are incorrectly predicted binding sites and non-binding sites, respectively.

The receiver operating characteristic curve plots TPR (true positive rate) = $TP/(TP + FN)$ against FPR (false positive rate) = $FP/(FP + TN)$ that are computed by binarizing the propensities using thresholds equal to all unique values of the propensities. The precision-recall curve plots precision against TPR. Given the imbalanced nature of our datasets (only about 14% of residues are protein binding), we also quantify the AULC (Area Under the Low false positive rate ROC Curve) value. AULC is the area under the receiver operating characteristic where the number of predicted PBRs <= number of native PBRs, i.e. where FPR is relatively low.

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.1}$$

$$Specificity = \frac{TN}{TN + FP} \tag{4.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{4.4}$$

$$F1 = 2 \times \frac{Sensitivity \times Precision}{Sensitivity + Precision} \tag{4.5}$$

$$MCC = \frac{TP \times TN - FN \times FP}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \tag{4.6}$$

4.3 Performance Comparison

In this section we compare all the methods we applied and discussed in the previous section. We analysed that the model performance can be improved by modifying the number of HSPs generated by SPRINT. The main three parameters of SPRINT is k_{size}, T_{hit} and T_{sim} which by default is set to 20, 15 and 35 respectively. We can compute less HSPs by increasing these parameter values and vice versa; see Table 4.2

Name	kSize	Thit	Tsim	Count
HSP-0.3M	20	15	35	3,38,700
HSP-52K	25	20	40	52,545
HSP-32K	26	21	41	35,451
HSP-2M	15	10	30	20,10,581
HSP-2.8M	14	9	29	28,47,052

Table 4.2: An overview of the HSP computation

In Table 4.3 we present the performance of DELPHI on our testing dataset, where the first line HSP stands for the baseline model performance, i.e. the default DELPHI and others with a '+' sign indicates that HSP is being used as a score and not as an input feature. This shows that, the model performs better when we reduce the HSP count.

Pro2Vec	Sens.	Spec.	Prec.	Acc.	F1	MCC	AUROC	AUPRC	AULC
HSP	0.234	0.872	0.234	0.780	0.234	0.105	0.588	0.203	0.011
HSP-2.8M+	0.275	0.879	0.275	0.792	0.275	0.154	0.657	0.238	0.013
HSP-2M+	0.291	0.881	0.291	0.797	0.291	0.173	0.669	0.255	0.015
HSP-0.3M+	0.330	**0.888**	0.330	**0.808**	0.330	0.218	0.702	0.298	0.018
HSP-35K+	**0.332**	**0.888**	**0.332**	**0.808**	**0.332**	**0.220**	**0.707**	0.300	**0.019**
HSP-52K+	**0.332**	**0.888**	**0.332**	**0.808**	**0.332**	**0.220**	**0.707**	**0.301**	**0.019**

Table 4.3: Performance of DELPHI

Next, we evaluate 5 different models that modify DELPHI to use ELMo, BERT, RoBERTa, Attention and Positional Encoding. The first three models are a small modification from the original implementation of DELPHI, where in place of ProtVec we are using different contextual embeddings. In ELMo, we replace the ProtVec feature with SeqVec embedding and keep the rest of the model architecture and parameters same as DELPHI. The pre-trained model of SeqVec is used as published by the authors. Similarly, for BERT and RoBERTa we follow the same procedure but this time the language models are pre-trained on our computer. Later, we perform the same experiment with the HSP score on these models as well. All these model performance are reported in Table 4.4, Table 4.5 and Table 4.6 respectively.

ELMo	Sens.	Spec.	Prec.	Acc.	F1	MCC	AUROC	AUPRC	AULC
HSP	0.233	0.871	0.233	0.780	0.233	0.104	0.577	0.201	0.011
HSP-2.8M+	0.289	0.881	0.289	0.796	0.289	0.170	0.666	0.248	0.014
HSP-2M+	0.302	0.883	0.302	0.800	0.302	0.186	0.678	0.264	0.016
HSP-0.3M+	**0.330**	**0.888**	**0.330**	**0.808**	**0.330**	**0.218**	0.706	0.298	**0.018**
HSP-35K+	**0.330**	**0.888**	**0.330**	**0.808**	**0.330**	**0.218**	**0.709**	**0.299**	**0.018**
HSP-52K+	**0.330**	**0.888**	**0.330**	**0.808**	**0.330**	**0.218**	**0.709**	**0.299**	**0.018**

Table 4.4: ELMo model performance

BERT	Sens.	Spec.	Prec.	Acc.	F1	MCC	AUROC	AUPRC	AULC
HSP	0.230	0.871	0.230	0.779	0.230	0.101	0.581	0.200	0.011
HSP-2.8M+	0.275	0.879	0.275	0.792	0.275	0.154	0.660	0.240	0.013
HSP-2M+	0.292	0.881	0.292	0.797	0.292	0.173	0.672	0.256	0.015
HSP-0.3M+	0.330	**0.888**	0.330	0.808	0.330	0.218	0.705	0.297	**0.018**
HSP-35K+	**0.333**	**0.888**	**0.333**	**0.809**	**0.333**	**0.221**	**0.709**	0.299	**0.018**
HSP-52K+	**0.333**	**0.888**	**0.333**	**0.809**	**0.333**	**0.221**	**0.709**	**0.300**	**0.018**

Table 4.5: BERT model performance

While the embeddings are used as an input feature to our model, we use the attention mechanism in a different way. In neural networks attention can be used as a layer similar to convolution or GRU, keeping this in mind we build a separate module where an attention layer is used followed by a fully connected layer. This new module is placed in addition to the existing CNN and RNN module that DELPHI uses. Next, all three modules are trained

RoBERTa	Sens.	Spec.	Prec.	Acc.	F1	MCC	AUROC	AUPRC	AULC
HSP	0.238	0.872	0.238	0.781	0.238	0.110	0.585	0.203	0.012
HSP-2.8M+	0.280	0.879	0.280	0.794	0.280	0.160	0.662	0.242	0.013
HSP-2M+	0.295	0.882	0.295	0.798	0.295	0.177	0.674	0.258	0.015
HSP-0.3M+	0.333	0.888	0.333	**0.809**	0.333	0.221	0.704	0.296	**0.018**
HSP-35K+	**0.335**	**0.889**	**0.335**	**0.809**	**0.335**	**0.223**	**0.709**	**0.298**	**0.018**
HSP-52K+	0.334	**0.889**	0.334	**0.809**	0.334	**0.223**	**0.709**	**0.298**	**0.018**

Table 4.6: RoBERTa model performance

separately, then the ensemble network loads their weights and train the final layer. The model performance is reported in Table 4.7, where in addition to the baseline model the effect of HSP scores are also captured.

Attention	Sens.	Spec.	Prec.	Acc.	F1	MCC	AUROC	AUPRC	AULC
HSP	0.234	0.872	0.234	0.780	0.234	0.105	0.583	0.202	0.012
HSP-2.8M+	0.273	0.878	0.273	0.791	0.273	0.152	0.656	0.235	0.013
HSP-2M+	0.290	0.881	0.290	0.796	0.290	0.171	0.669	0.251	0.014
HSP-0.3M+	**0.330**	**0.888**	**0.330**	**0.808**	**0.330**	**0.218**	0.701	0.292	**0.018**
HSP-35K+	**0.330**	**0.888**	**0.330**	**0.808**	**0.330**	**0.218**	**0.706**	0.293	**0.018**
HSP-52K+	**0.330**	**0.888**	**0.330**	**0.808**	**0.330**	**0.218**	**0.706**	**0.294**	**0.018**

Table 4.7: Attention model performance

The Positional Encoding feature is used by replacing the existing Position information used by DELPHI. While using this feature we kept the original embedding ProtVec in place to understand the sole effect of this new feature on our model. Table 4.8 reports the model performance where the experiment with HSP score on this is also included.

Position	Sens.	Spec.	Prec.	Acc.	F1	MCC	AUROC	AUPRC	AULC
HSP	0.232	0.871	0.232	0.780	0.232	0.103	0.579	0.200	0.011
HSP-2.8M+	0.282	0.880	0.282	0.794	0.282	0.162	0.663	0.246	0.014
HSP-2M+	0.297	0.882	0.297	0.798	0.297	0.180	0.674	0.262	0.015
HSP-0.3M+	**0.331**	**0.888**	**0.331**	**0.808**	**0.331**	0.218	0.704	0.296	**0.018**
HSP-35K+	0.330	**0.888**	0.330	**0.808**	0.330	0.218	**0.707**	0.297	**0.018**
HSP-52K+	**0.331**	**0.888**	**0.331**	**0.808**	**0.331**	**0.219**	**0.707**	**0.298**	**0.018**

Table 4.8: Positional Encoding model performance

Since, it has been observed that for all the models we achieve the best performance by using HSP as a score and not as a feature, and that can be further improved by reducing the HSP count, we put all the best results from each of the methods in Table 4.9. This gives us a clear picture of which method works best.

Here, we also added some new tests. As discussed in the previous chapter, attention can be used separately or with either of GRU and convolution branch, we added this models as well in this comparison. First, we took the default feature group that DELPHI uses excluding HSP (as we are using it as a score), and test three different models by using only attention module, attention with RNN and attention with CNN. Next, we also modified the input feature group by replacing the Position information with the Positional Encoding and Pro2Vec with BERT embedding, and evaluate another three models.

Best Models	Sens.	Spec.	Prec.	Acc.	F1	MCC	AUROC	AUPRC	AULC
BERT-POS-Attn	0.260	0.876	0.260	0.788	0.260	0.137	0.617	0.228	0.013
ProVec-Attn	0.268	0.877	0.268	0.790	0.268	0.145	0.623	0.232	0.013
BERT-POS-CNN-Attn	0.302	0.883	0.302	0.800	0.302	0.185	0.669	0.264	0.016
Pro2Vec-CNN-Attn	0.301	0.883	0.301	0.799	0.301	0.184	0.671	0.268	0.016
Attention	0.330	0.888	0.330	0.808	0.330	0.218	0.706	0.294	0.018
BERT-POS-RNN-Attn	0.327	0.887	0.327	0.807	0.327	0.214	0.705	0.295	0.018
Pro2Vec-RNN-Attn	0.330	0.888	0.330	0.808	0.330	0.217	0.704	0.296	0.018
POSITION	0.331	0.888	0.331	0.808	0.331	0.219	0.707	0.298	0.018
RoBERTa	**0.334**	**0.889**	**0.334**	**0.809**	**0.334**	**0.223**	**0.709**	0.298	0.018
ELMo	0.330	0.888	0.330	0.808	0.330	0.218	**0.709**	0.299	0.018
BERT	0.333	0.888	0.333	**0.809**	0.333	0.221	**0.709**	0.300	0.018
Pro2Vec	0.332	0.888	0.332	0.808	0.332	0.220	0.707	**0.301**	**0.019**

Table 4.9: Best model performance comparison

4.4 Discussion

It is interesting to notice that while we succeeded to improve the performance, some of the best machine learning techniques failed to provide the expected improvement. At first, when we came up with the idea of applying contextual embedding in place of context independent Word2Vec we were influenced by their performance in several NLP tasks. Since, both BioInformatics and NLP deals with same type of data i.e. sequence based, we assumed that it might improve our PPI prediction program. The fact that it does not, make us believe that, in natural language there are more syntactic and semantic similarities between words, whereas there is no such thing between amino acid residues in protein sequences, that makes the embedding a less important feature. Similar things happen with attention and positional encoding. On the other hand, a significant improvement happens when we use the HSP score in addition to the prediction probability, which gives 48.27% improvement on the PR curve and 20.23% improvement on the ROC curve for the default model. This proves that there can be non-machine learning ways to improve a machine learning model.

However, from Table 4.9 firstly we can see that, attention works better with RNN than with CNN module. Secondly, though the machine learning techniques are not helping our model very significantly, there is a very slight improvement in some of the metrics when we replace ProtVec with the RoBERTa or BERT embedding. This definitely leads us towards further investigation.

Chapter 5

Conclusion and Future Works

In this chapter, we will discuss the summary of our work and possible future works to continue with further investigation.

5.1 Summary

In this thesis, we have considered the problem of predicting binding sites in protein protein interaction. We have discussed various PPI binding site prediction techniques including both computational and machine learning based approaches. Then we discussed the current best program, DELPHI and its architecture.

We then present our work, where we have employed some of the currently best techniques from machine learning, including attention and various embedding techniques, such as BERT and ELMo. This is the first time such tools are being tested for this problem. Finally we have tested many architecture on a large dataset and analyzed our findings.

5.2 Conclusion

It is very interesting to notice that, some of the best machine learning techniques failed to provide the expected improvement, while we succeeded to improve the performance using a non machine learning approach. The current best program, DELPHI uses the HSP score as an input feature to the model, while we found out that using it in some other way improves the performance. In that case, we remove the HSP feature from our input, then train the model an finally add the HSP scores with model's prediction to obtain the final prediction probability of the binding sites. Later, we also discovered that the performance can be improved further by generating less HSPs before computing the HSP scores. It is very surprising that, this way we managed to achieve 48.27% improvement on the PR curve and 20.23% improvement on the ROC curve for the default model. Finally, by comparing all the best models we can say that attention works better with RNN than with CNN and some contextual embedding(BERT, RoBERTa) are slightly better than ProtVec in some cases.

5.3 Future Work

There is a plenty of scope for future work. Firstly, it may be possible to make the program more accurate by tuning its parameters and probably by training it on a larger dataset, though binding site information is not easier to find.

Secondly, we need to further investigate why the state-of-the art machine learning techniques are not making the expected improvement on our model. Since, these methods have already made a significant improvement in several NLP tasks we still believe that this can also improve a bioinformatics problem, but maybe in some other way.

Bibliography

[1] Adam Amos-Binks, Catalin Patulea, Sylvain Pitre, Andrew Schoenrock, Yuan Gui, James R Green, Ashkan Golshani, and Frank Dehne. Binding site prediction for protein-protein interactions and novel motif discovery using re-occurring polypeptide sequences. *BMC bioinformatics*, 12(1):225, 2011.

[2] Ehsaneddin Asgari and Mohammad RK Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 10(11):e0141287, 2015.

[3] Ehsaneddin Asgari, Nina Poerner, Alice C McHardy, and Mohammad RK Mofrad. Deepprime2sec: Deep learning for protein secondary structure prediction from the primary sequences. *bioRxiv*, page 705426, 2019.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[7] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Kaustubh Dhole, Gurdeep Singh, Priyadarshini P Pai, and Sukanta Mondal. Sequence-based prediction of protein–protein interaction sites with l1-logreg classifier. *Journal of theoretical biology*, 348:47–54, 2014.

[10] Zsuzsanna Dosztányi, Bálint Mészáros, and István Simon. Anchor: web server for predicting protein binding regions in disordered proteins. *Bioinformatics*, 25(20):2745–2746, 2009.

[11] Eshel Faraggi, Yaoqi Zhou, and Andrzej Kloczkowski. Accurate single-sequence prediction of solvent accessible surface area using local and global features. *Proteins: Structure, Function, and Bioinformatics*, 82(11):3170–3176, 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Michael Heinzinger, Ahmed Elnaggar, Yu Wang, Christian Dallago, Dmitrii Nechaev, Florian Matthes, and Burkhard Rost. Modeling aspects of the language of life through transfer-learning protein sequences. *BMC bioinformatics*, 20(1):1–17, 2019.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Yiwei Li, G Brian Golding, and Lucian Ilie. DELPHI: accurate deep ensemble model for protein interaction sites prediction. *Bioinformatics*, 08 2020. btaa750.

[20] Yiwei Li and Lucian Ilie. Sprint: ultrafast protein-protein interaction prediction of the entire human interactome. *BMC bioinformatics*, 18(1):485, 2017.

[21] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[23] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869, 2017.

[24] Yoichi Murakami and Kenji Mizuguchi. Applying the naïve bayes classifier with kernel density estimation to the prediction of protein–protein interaction sites. *Bioinformatics*, 26(15):1841–1848, 2010.

[25] Ananthan Nambiar, Maeve Heflin, Simon Liu, Sergei Maslov, Mark Hopkins, and Anna Ritz. Transforming the language of life: Transformer neural networks for protein prediction tasks. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–8, 2020.

[26] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[27] Sylvain Pitre, Frank Dehne, Albert Chan, Jim Cheetham, Alex Duong, Andrew Emili, Marinella Gebbia, Jack Greenblatt, Mathew Jessulat, Nevan Krogan, et al. Pipe: a protein-protein interaction prediction engine based on the re-occurring short polypeptide sequences between known interacting protein pairs. *BMC bioinformatics*, 7(1):365, 2006.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[29] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Peter Chen, John Canny, Pieter Abbeel, and Yun Song. Evaluating protein transfer learning with tape. In *Advances in Neural Information Processing Systems*, pages 9689–9701, 2019.

[30] Michael Remmert, Andreas Biegert, Andreas Hauser, and Johannes Söding. Hhblits: lightning-fast iterative protein sequence searching by hmm-hmm alignment. *Nature methods*, 9(2):173–175, 2012.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[33] Tensorflow. An end-to-end open source machine learning platform. https://www.tensorflow.org.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[35] Zhi-Sen Wei, Ke Han, Jing-Yu Yang, Hong-Bin Shen, and Dong-Jun Yu. Protein–protein interaction sites prediction by ensembling svm and sample-weighted random forests. *Neurocomputing*, 193:201–212, 2016.

[36] Wikipedia. The free encyclopedia. https://www.wikipedia.org.

[37] William C Wimley and Stephen H White. Experimentally determined hydrophobicity scale for proteins at membrane interfaces. *Nature structural biology*, 3(10):842–848, 1996.

[38] Min Zeng, Fuhao Zhang, Fang-Xiang Wu, Yaohang Li, Jianxin Wang, and Min Li. Protein–protein interaction site prediction through combining local and global features with deep neural networks. *Bioinformatics*, 36(4):1114–1120, 2020.

[39] Buzhong Zhang, Jinyan Li, Lijun Quan, Yu Chen, and Qiang Lü. Sequence-based prediction of protein-protein interaction sites by simplified long short-term memory network. *Neurocomputing*, 357:86–100, 2019.

[40] Jian Zhang and Lukasz Kurgan. Scriber: accurate and partner type-specific prediction of protein-binding residues from proteins sequences. *Bioinformatics*, 35(14):i343–i353, 2019.

[41] Jian Zhang, Zhiqiang Ma, and Lukasz Kurgan. Comprehensive review and empirical analysis of hallmarks of dna-, rna-and protein-binding residues in protein chains. *Briefings in bioinformatics*, 20(4):1250–1268, 2019.

Curriculum Vitae

Name:	Sourajit Basak
Post-Secondary Education and Degrees:	West Bengal University of Technology Kolkata, India 2011 - 2015 B.Tech
	University of Western Ontario London, ON 2019 - 2020 M.Sc.
Honours and Awards:	Western Graduate Research Scholarship 2019 - 2020
Related Work Experience:	Teaching and Research Assistant The University of Western Ontario 2019 - 2020
	Data Analyst British Telecom 2018 - 2019
	Software Engineer Tech Mahindra Ltd. 2015 - 2018