University of Cape Town

Applied Mathematics Honours Thesis

MAM4001W

Protein Annotation using Structured State Space models

Author
Qiulin Li

Supervisor
Associate Professor Jonathan Shock

Declaration of Authorship

I Qiulin Li hereby declare that this report is my original work. (except where acknowledgements indicate otherwise). I authorise the University to reproduce this for the purpose of research either the whole or any portion of the contents in any manner whatsoever.

Signature:

Date: 25-09-2023

Acknowledgements

Many of the computations in this report were done on the High Performance Computing Cluster (HPC) at the University of Cape Town. My heartfelt thanks to the engineers who maintain and support the cluster.
I would also like to thank my Supervisor, Associate Professor Jonathan Shock for his advice and patience with me throughout this project. Without his invaluable knowledge, this project would not have been possible.

Contents

Abstract

Proteins are vital for life, as they perform critical functions ranging from catalysis to structural support in all living organisms. Oftentimes diseases in organisms manifest by causing proteins to misfunction. Understanding how proteins function is essential in preventing their misfunction and thus disease. Many proteins have been sequences, however few have been annotated with their function since this is often a long and labourious process. Machine learning has been applied to the task of protein annotation to increase the number of annotate proteins available, however these models are not perfect. In this paper, we apply a new Sequence model known as the Structured State Space model to annotate proteins. We train and test S4 on the set of all Human proteins that has been annotated by hand to investigate whether it is a potential SOTA model for protein annotation.

Chapter 1

Introduction

Approximately 100 000 unique proteins exist in the human body alone [1] and many more exist in other organisms. This makes proteins incredibly important to any living organisms. Proteins perform various functions in living organism from acting as hormones to transporting nutrients [2]. Proteins are responsible for many biological processes. Many diseases manifest by changing and influencing protein functions [3]. It is thus essential to understanding how proteins function in various biological processes, since this will allow us to better understand how diseases function, which can aid in drug development.

Proteins have been classified according to their function by numerous organisations, however, due to the multi-faceted nature of proteins, they often perform several different and similar functions, leading to confusion when developing classification systems. The Gene Ontology consortium is a popular protein classification systems produced in 1998 [10] that groups proteins according to their biological process, cellular component and molecular function.

Annotating protein function experimentally is expensive and time consuming, thus many protein sequences remain without annotation. Recently, machine learning algorithms have been applied to the protein annotation problem with varying degrees of success in order to annotate proteins according to their biological, cellular and molecular function.This report introduces a new sequence model known as the Structured State Space Model (S4), which can account for long-range dependencies in sequential inputs. S4 is also computationally efficient and accurate [54] compared to Transformer-based models when given long sequences. Transformer models have recently achieved state-of-the-art performance on protein function prediction. Since S4 is designed to handle long sequences and implied to perform better than transformers, this report aims to assess its performance in protein annotation.

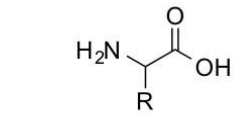
Chapter 2

Literature Review

2.1 What are Proteins?

The term protein has been used widely as a label for many organic macromolecules. In general, an organic compound that is made up of smaller molecules known as amino acids that are chemically bonded using peptide chains are known as proteins [2]. There are 22 different amino acids, 20 of which are encoded by DNA and 2 of which are encoded by tRNA [1]. Approximately 100,000 unique proteins exist in the human body alone and many more exist in other organisms [1]. Proteins can be described using their primary, secondary, tertiary and Quaternary structure. The primary structure consists primarily of amino acids which are linked together to forms higher-order structures [4].

All amino acids are made up of an amino group, a carboxylic acid functional group [1] and a unique side chain (R). The general form of an amino acid is shown in figure 2.1.



1. Amino Acid Generic
es and differentiates one

Figure 2.1: Generic Form of an amino acid[1]

The R side-chain can take on different forms depending on the amino acid. The Primary structure of a protein is the sequence of amino acid also known as a polypeptide chain. Each amino acid has unique chemical properties. The amino acids in a polypeptide chain determine how proteins' fold and bonds with other molecules [4]. The secondary structure of these proteins is determined by the local folding of the polypeptide chain [2] due to hydrogen bonds between the *-OH* group of one amino acid and a hydrogen molecule of another amino acid. The most common forms of the secondary structure is α- helices and β- pleated sheets [2]. The primary and secondary protein structures are shown in 2.2

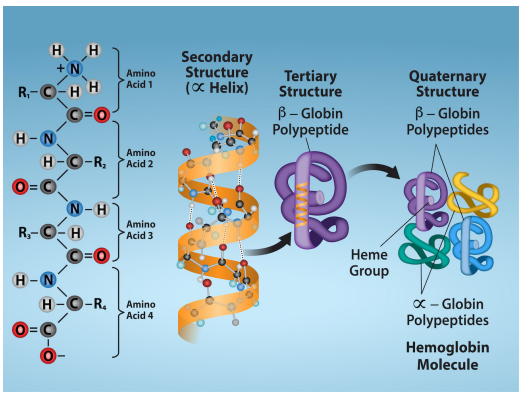


Figure 2.2: Protein Structure [4]

The tertiary structure is the 3D structure of proteins after folding. These foldings are driven by chemical interactions between the R side-chains of the amino acids in the polypeptide chain [4]. The chemical interactions between the R sidechains are more powerful than H-bonds in the secondary structure of a protein. This structure contains the binding sites where the protein acts on other molecules [2] thus if a protein loses its' tertiary (3D) structure, the binding site is lost and the protein will no longer be functional. The quaternary structure of a protein is formed when several polypeptide chains interact. This protein structure is shaped by chemical bonds between the R side-chains of different amino acids [4] in the polypeptide chains. The protein in this form is fully functional and will perform its' designated role in a cell. The tertiary and Quaternary structure is shown in 2.2.

Proteins perform many essential functions in living organisms that vary based on their structure and environment (c). Each cell in an organic system has thousands of proteins with a unique structure and function. We will outline some of the more important functions that proteins perform in an organism.

2.2 Function of Proteins

2.2.1 Enzymes

Enzymes are proteins that facilitate chemical reactions in a system. They do so by acting as catalysts and reducing the activation energy of reactions [4]. The enzymes will bind to the reactants, the region of the enzyme where this occurs is known as the activation site. The enzyme will either slightly modify its' shape to better bind to the protein in a process known as induced fit, or the reactant will fit perfectly into the enzyme, in a lock and key model. The enzyme holds the reactant molecules in such a way that the chemical bonds readily break and form [4]. This process is shown in Figure 2.3 where the enzyme is catalysing the breakdown of some molecule.

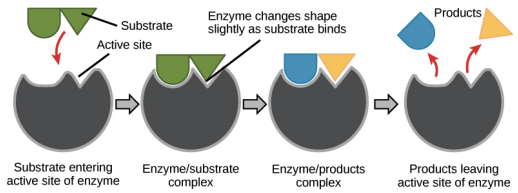


Figure 2.3: Generic Form of an amino acid [2]

Enzymes can catalyse reactions in many ways. It can breakdown reactants, take part in reactions or form part of a reaction intermediary. Enzymes can be classified as either **Catabolic reactions** that build new molecules or **Anabolic reactions** that break molecules down to their constituents [2].

2.2.2 Editing DNA

Proteins are critical to the DNA editing processes. They contribute to the maintenance and modification of genetic material through processes such as repairing DNA [2]. There are many types of DNA repairation mechanisms such as nucleotide excision repair, base excision repair, and mismatch repairs. Proteins can also edit DNA via DNA recombination which allows DNA sequences to be rearranged, resulting in genetic diversity [4]. This is usually done by proteins known as recombinases that break and rejoin DNA strands during recombination [4].

Proteins can modify DNA through chemical reactions such as DNA methylation. DNA methylation is the process in which a methyl group is added to specific DNA nucleotides, known as cytosine [6]. This process is typically studied in relation to epigenetics. The addition of methyl groups to DNA can influence gene activity by silencing or activating specific genes which can impact various cellular processes. An example of how nucleotide can be methylated is shown below in figure 2.4

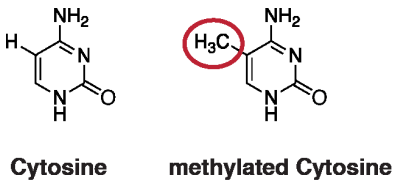


Figure 2.4: Cytosine Methylated vs unmethylated [6]

2.2.3 Transport Medium

Proteins often play important roles as transport mediums in cells. The two main types of Transport proteins are carrier and channel proteins that are responsible for moving essential substances across cellular membranes [5]. These proteins are shown below in Figure 2.5

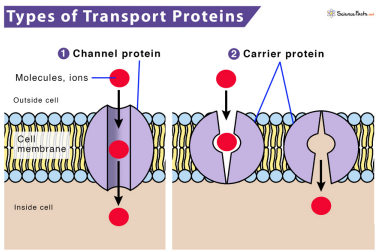


Figure 2.5: Transport Proteins[5]

2.2.4 Structural support

Proteins can also be used in intracellular transport processes. Motor proteins, such as kinesins and dyneins [8], are responsible for the movement of vesicles, organelles, and other cargo along the microtubule tracks within cells. These proteins utilize ATP hydrolysis to generate the necessary energy for movement [8], allowing for the precise positioning of cellular components and the delivery of cargo to specific cellular locations.

In summary, proteins serve as versatile transport mediums in biological systems. They facilitate the transport of molecules across cell membranes, act as carriers for gases and other substances in the bloodstream, and enable intracellular transport processes [2]. The diverse functions of transport proteins highlight their crucial role in maintaining cellular homeostasis and facilitating the proper functioning of living organisms.

A single protein can perform many different functions based on their environment, for example, the same protein sequence of a duck's eye lens is identical to active lactate dehydrogenase [10]. Protein sequences can

thus perform several functions and be labelled according to these functions. The proteins are then classified with respect to these function, however, there have been difficulties in doing so since proteins perform a wide array of unique functions that cannot necessarily be grouped (c).

2.3 Protein Seqeunces

Protein can usually be denoted as sequences of amino acids. The 20 common amino acid and their single-letter encoding as shown in the table 2.1 below [1].

Amino Acid	**Code**
Alanine	A
Arginine	R
Asparagine	N
Aspartic acid	D
Cysteine	C
Glutamine	Q
Glutamic acid	E
Glycine	G
Histidine	H
Isoleucine	I
Leucine	L
Lysine	K
Methionine	M
Phenylalanine	F
Proline	P
Serine	S
Threonine	T
Tryptophan	W
Tyrosine	Y
Valine	V

Table 2.1: Amino Acids and Single-Letter Codes

The two rare amino acids, selenocysteine and pyrrolysine, are encoded using U and O. These letters hardly occur in common protein sequences since pyrrolysine is only found in archaic bacteria and selenocysteine only occurs at special stop codons [9].

Amino acids that are far away from each other in a protein sequence can still interact introducing long-range dependencies in a protein sequence [4]. These interactions occur due to protein folding, where chains of amino acids become three-dimensional structures. Amino acids that are far apart in a sequence are brought into close proximity in the quarternary structure [4]. Amino acids that are far apart form connections that contribute to the stability and functionality of the protein via chemical reactions. These long-range interactions are important in determining the protein's structures and functions. Proteins can only reach their complex structures due to the ability of amino acids that are far away from one another in the protein sequence to interact [2], making the long-range dependencies of amino acids incredibly important.

Amino acids are fundamental in determining a protein's functions. Each amino acid has unique properties due to their unique side chains and functional groups [1]. Amino acids also contribute to both a protein's structure and function. Amino acid residues within a protein sequence can become active sites that allow proteins

to interact with one another. The distinct chemical properties of amino acids allows proteins to carry out their specific biological functions mentioned above [7]. Understanding the function of proteins in isolation as well as in relation to one another is important to understanding how living orgnisms function. There has thus been many attempts to classify proteins or alternative, annotate protein sequences.

2.4 Protein Annotation

There have been attempts to group proteins according to their function since the first protein sequences were identified, however there is currently no consensus on which method is the best. Initial attempts focused on grouping proteins into functional classes such as energy, information, communication and regulation [10]. Each class had further subclasses under which proteins where grouped. This method of classification was considered too broad since terms such as 'Biosynthesis' were too general and almost any protein could be assigned this class given the correct context [10]. Many attempts were made to overcome this issue, however only one of these methods are still used today.
This method is known as **Protein family classification**, where Proteins are classified manually according to their structure, activity and function within a cell. It is only recently that machine learning has been using to classify these proteins, however there has been little success in this regard since many protein family classifications must be verified manually [11]. These protein families are usually related through evolution, have similar motifs and domains which result in similar functions. There are many protein family groups, however, the most popular protein family classification group is *Pfam* which is maintained by Oxford and updated continuously [12]. Although this method is data intensive, Pfam has successfully classified 199,841 protein sequences into 19,804 families [12]. Proteins in a family catalyse the same reactions and can be assigned to a Superfamily. All members of a superfamily perform the same mechanism in a reaction, hence their closer relation.

Proteins that are classified using this system are subject to **Rule-based annotation**[11]. Several rules are used to classify proteins, for example, finding a certain domain at a specific location in a sequence will automatically allocate the protein to a specific family. This system soon also proved to be problematic when biologists, biochemists and molecular biologists could not agree on what the function of a protein is [10]. Biologists defined function as the biological process that the protein takes part in. Biochemists focus primarily on the chemical processes the protein functions in. Molecular biologists, similar to biochemists, focus on the function of proteins in chemical process, however, they focus more on the molecular mechanisms that govern these chemical processes [10]. Biochemists and molecular biologists also differ in the experimental methods used to verify protein functions. This often results in different functions being found since proteins vary their functions based on their environment. Since Protein families are based on functions, the lack of consensus resulted in ambiguity when describing the functions of families [11]. The functional relationship that exists between proteins is also lost in this classification system. The Gene Ontology Consortium was invented in 1999 and largely considered the best classification system that has been produced thus far. [10],[13]

2.4.1 Gene Ontology Consortium

The Gene Ontology consortium created a classification system that unified biologists, biochemists and molecular biologists when annotating proteins. A protein sequence can be labelled according to either its biological process (BP), cellular component (CC) or its' molecular function (MF), which is used by biologists, biochemists and molecular biologists respectively [13]. The functional relationships are also included in the labels. The labels of a protein sequence, also known as GOs, are used to form a directed acyclic graph where each label is a node and each edge can be denoted relations between the labels.

Although the GO System overcame many of the issues present in previous classification systems, it is not

perfect [10]. Many GO labels often only occur once without any relation to other GO labels or other proteins. The GO labels are IDs such as "GO:01135" with a corresponding function definition. These IDs have no relation to one another since the number of GO labels is large and growing. The problem of over-generalisation in previous systems has now been replaces with over-specification with the growing number of labels and function [10]. The labels are also difficult to interpret since they exist in the form of a graph. It should be noted that although the GO system has problems, it is widely accepted as the best classification system to date, with many institutions working with and on the data to continuously improve the quality and interpretability of the GO labels [13].

Only 20% of all Protein sequences in the GO consortium have been labelled and reviewed [13]. This means that almost 80% of Protein Sequences have miscellaneous or no labels at all. UniProt (Universal Protein Resource) is an open-source database with over 200 million protein sequences [14]. SwissProt is a subset of the UniProt dataset that consists of Protein sequences that have been curated by the Swiss Institute of Bioinformatics. An example of a sequence that has been curated is shown in Figure 2.6 [15]. Basic information about the protein, such as which species it is found in, which gene encodes for it and whether the protein exists (has it been found experimentally) is shown. The papers in which the protein is mentioned as well as its' GO annotations are also found on this page but not shown in Figure 2.6.

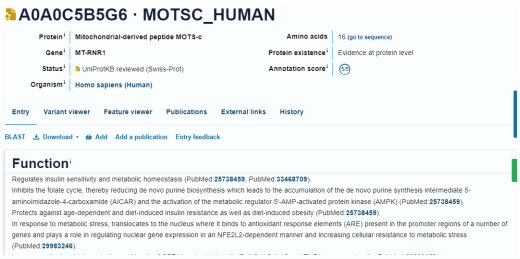


Figure 2.6: An example of a SwissProt protein Sequence

SwissProt Proteins contain large amounts of data about a protein that has been verified to be true. The SwissProt dataset, however, only has 569 793 protein sequences, leaving 248 million protein sequences unreviewed [15]. Unreviewed proteins can be thoroughly annotated and assigned various GO labels, however the accuracy of the labels cannot be verified. Figure 2.6 shows that the annotation score of that specific protein is 5/5 which indicates that there is range of information about the proteins' structure, features and functions but does not reflect the correctness of the GO labels. Unreviewed proteins can have a annotation score of 5/5 but the GO labels are more likely to be erroneous due to the lack of curation [14].

The only way to verify with certainty that a protein performs its' function is to do so experimentally, however, other methods, such as sequence alignment, have been used to label proteins. BLAST (Basic Local Alignment Search Tool) is a sequence alignment tool that is used to search for similar motifs and domains between sequences [10]. BLAST assumes that homologuous protein sequences have similar functions since the chemical properties of motifs do not change. Literature mining is another popular method that uses Natural language processing to scrape information about protein sequences and their functions from current literature that were not inserted into the GO database [11]. There have also been attempts to use various features of proteins (pH, structure etc) to label these proteins, however, due to the uncertainty of the assigned features, the assigned GO labels based on these features is uncertain.

These alternative methods are shown to produce labels with varying degrees of uncertainty, however, if enough uncertain methods can verify the label of a GO, the certainty that the Protein can be assigned that label increases. Recently, machine learning algorithms have been applied to this problem [16]. All machine learning models possess a degree of uncertainty, however if enough ML algorithms can verify a label, more proteins can be labelled with some degree of certainty without undergoing long experimental processes.

2.5 Machine learning Methods

2.5.1 A note on Proteins and Natural Language processing

The composition of proteins as sequences of amino acids. is similar to how sentences are composed of words [17]. NLP techniques that have been used to study text and languages can be applied to protein sequences and functions. Machine and Deep learning algorithms that have historically been used in NLP have been shown to perform well in many genomic tasks [17]. The similarity between NLP data and Genomic data is reflected by the fact that we can use the same preprocessing steps for both types of data.

Tokenization and Vectorization are fundamental preprocessing steps in natural language processing (NLP) to transform data into machine-readable vectors [18]. These methods can similarly be applied to protein sequences. Tokenization (in terms of NLP) consists of breaking down text into smaller units called tokens. Protein sequences can similarly be broken down into tokens that represent amino acids. This allows us to transform a protein sequence into discrete machine-readable units which can be further processed.

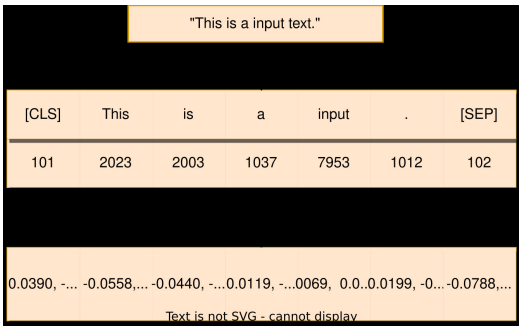


Figure 2.7: An example of a Tokenization and Embedding in NLP[18]

Tokenization is usually followed by Vectorization. Vectorization is the process of converting tokens into a vector known as the embedding [18]. These embeddings typically capture underlying semantics and patterns in the data. Popular embedding techniques include using Large Language models such as Word2Vec, GloVe, or BERT, or using vectorization algorithms such as TF-IDF [20]. An example of tokenization and vectorization is shown in Figure 2.7.

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a common vectorization technique used in Natural Language Processing [19]. It measures the relevance of one token to a vector in the context of the entire dataset, or in an NLP context, how relevant a term is to the document given a collection of documents [21]. TF-IDF takes two factors known as term frequency (TF) and inverse document frequency (IDF) into account during the vectorization process.

Term frequency (TF) is the frequency of a term (token) within a document (vector). It is a ratio of the number of occurrences of the term to the total number of terms in the document shown in equation 2.1. It quantifies how often a token appears in a vector and its' importance in that vector [20].

$$TF = \frac{\text{no. occurences of term in document}}{\text{total no. terms in document}} \tag{2.1}$$

The inverse document frequency (IDF) computes the importance of a term relative to the whole collection of documents [20]. It is the inverse of the Document Frequency (DF) term, which measures how frequent a term occurs across the whole collection. Document Frequency is calculated using equation 2.2. It is inverted and a logarithm is then applied to it in order to obtain the IDF value. The Inverse Document Frequency value can obtained via equation 2.3. It is a logarithm of the ratio of the total number of documents in a collection to the number of documents containing the term. IDF penalizes terms that appear in many documents since this diminishes the importance of the term [21].

$$DF = \frac{\text{no. documents containing term}}{\text{total no. documents}} \tag{2.2}$$

$$IDF = log(\frac{\text{total no. documents}}{\text{no. documents containing term}}) \tag{2.3}$$

The TF-IDF score of a term in a document is computed by using both its term frequency (TF) and its inverse document frequency (IDF). This score reflects the importance of the term within both document and the collection. Higher TF-IDF scores indicate that a term is more relevant to a document [21]. The function used to calculate the TF-IDF score is shown in equation 2.4. If we replace the term with token, document with vector and collection with dataset, the TF-IDF algorithm can be used to vectorise any sequential dataset, including protein sequences.

$$TF - IDF(t, d) = TF(t) \times IDF(t, d) \tag{2.4}$$

bioBERT

BioBERT is a specialized language representation model designed for biomedical text processing that is trained on PubMed Literature[22]. The architecture and training dataset of BERT is shown below in Figure 2.8. It is based on the popular BERT (Bidirectional Encoder Representations from Transformers) architecture and thus has similar performance to BERT. BioBERT was first produced in 2020 by Lee et al in Pytorch but has since been converted to Tensorflow and shifted to the HuggingFace Model repository.

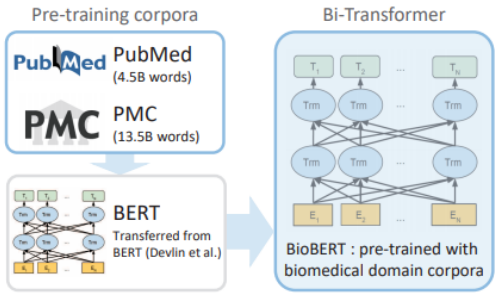


Figure 2.8: A summary of BioBERT [23]

BioBERT is used to for vectorisation in the context of protein sequences since it can handle domain-specific biomedical terminology, which is difficult for general-purpose language models [23]. Pre-training bioBERT on biomedical corpus allows it to capture biomedical entities, and relationships present only in biomedical texts . This allows bioBERT to better understand and represent biomedical concepts and improve performance on tasks such as biomedical text classification and vectorisation [22]. The vectorised protein sequences can be then be used in various machine learning algorithms. Labelling proteins is often considered a classification task. We can thus feed the vectorised protein sequences and their corresponding labels into classifier algorithms

2.6 Machine Learning Classifiers

2.6.1 Clustering Algorithms

Clustering algorithms are unsupervised machine learning algorithms [24] that aims to group data points together based on their similarities. The goal of clustering is to divide a dataset into distinct clusters, with data in the same cluster being more similar to one another than to those in other clusters. Clustering algorithms have been used to complete various tasks including pattern recognition and data analysis [25].

One of the most popular clustering algorithm is k-means clustering, which divides the data into k clusters through optimizing cluster centroids iteratively. The algorithm randomly initializing k centroids and assigns data points to their nearest centroid [25]. The centroids are then updated by calculating the mean of the data points assigned to each cluster. K-means is efficient and can be utilized on many applications, however, it has its flaws. The algorithm is extremely dependent on the initial centroid positions and is sensitive to outliers. It is therefore important to consider alternative clustering algorithms such as Hierarchical clustering alongside K-means when choosing clustering algorithms.

Hierarchical clustering creates clusters with a hierarchical structure. Each data point begins as its' own cluster and is later merged with other clusters based on similarities. This process continues until the number of desired clusters is produced [25]. Hierarchical clustering can be visualised through dendrograms shown in figure 2.9 below, which allowing the clustering results to become easily interpretable.

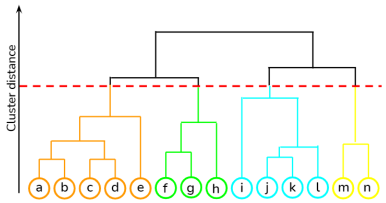


Figure 2.9: Dendrogram of Hierarchical Clusters [26]

2.6.2 Support Vector Machines

Support vector machines (SVM) are supervised classifier algorithms. SVMs attempt find an optimal hyperplane that separates different classes in a dataset. The hyperplane can be linear or non-linear, depending on the problem [27]. The hyperplane is usually chosen to have the maximum distance from the nearest data points of each class. A Figure illustrating how the Hyperplane can be used to separate two classes of data points is shown in Figure 2.10 below. In this figure, any new points in the area left of the hyperplane are considered part of the 'Green' class, whereas those on the right of the hyperplane are part of the 'Blue' class

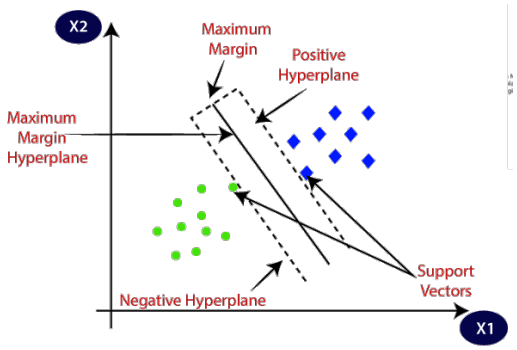


Figure 2.10: A simplified SVM [27]

Figure 2.10 provides a simple example in which the points can be easily separated and classified, however this is often not the case. Many datasets are filled with outliers. The Support vectors are thus used to make the model robust to these outliers [28]. Support vectors, as shown in Figure 2.10, are the datapoints closest to the hyperplane. They are used to construct and shape the hyperplane itself making them incredibly important to the SVM model. The goal of an SVM is to maximise the margin, or the distance from the Support vector to the hyperplane. This makes SVMs robust since it is mainly influenced by the Support vectors and not the outliers. This makes SVMs a good algorithm to use when given data with large amounts of variance.

SVMs, like many other algorithms, are not without flaws. SVMs are computationally expensive since the model retains its' entire dataset in memory while training [25]. SVM performance also varies based on hyperparameters such as type of kernel used. The kernel function is an inner product between two data points that provides a measure of similarity between data points [26]. There are various kernel functions that vary based on the nature of a given dataset. Datasets with too much noise will also not be classified well by SVMs since the algorithm will not be able to find the Decision boundary(Hyperplane) between classes.

SVMs have been applied to many tasks such as text classification and image recognition due to their ability to generalize, their interpretability, and their ability to handle datasets with large vectors. SVMs are sensitive to hyperparameters values which must be fine tuned to for optimal performance. SVMs are also computationally heavy making them unideal for large datasets, deep Learning models, however, are considered data hungry models [25] that benefit from large training datasets. Deep Learning Models, such as Neural Networks, often outperform standard Machine learning Classifiers due to their universal approximator property [29].

2.7 Neural Networks

Neural Networks are a type of deep learning model architecture that can be used to perform many tasks. The most common tasks are regression and classification[30]. The structure of neural networks is inspired by the networks of neurons that facilitate learning in the human brain. Neural Networks are made up of artificial neurons, known as **perceptron**. Perceptrons were first invented by Rosenblatt in 1957 [29] where they were used as binary classifiers. Now they are more commonly used as building blocks to construct artificial neural networks. The structure of a perceptron is shown below.

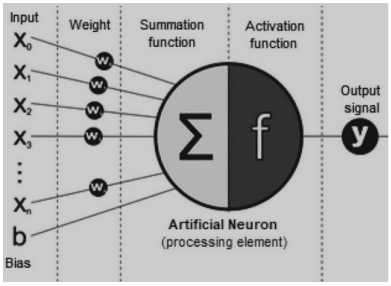


Figure 2.11: Perceptron [31]

Perceptrons are primarily made up of two functions. Given a series of inputs, the perceptron will firstly sum all the weighted inputs via the **Summation function**. The summation also consists of a bias value (b_i) which gives the perceptron an additional degree of freedom. This degree of freedom allows perceptrons and the consequent neural network to perform better as a function approximator [31]. The second function is the nonlinear **Activation function**. There are many types of Activation functions that differ according to the use case. Different activation functions have different advantages and disadvantages, hence the decision of which activation function to use in a perceptron often depends on the use case.

The architecture of a simple Multi-layer feed forward Neural network, also known as an artificial neural network (ANN), is shown below in Figure 2.12. These Neural Networks have three layer types, output, hidden and input layer. These layers can be organised in a variety of ways resulting in a variation of neural networks.

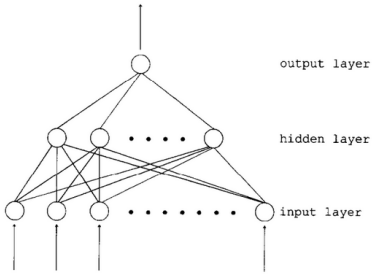


Figure 2.12: Multi-layer feedforward Neural Network [29]

All neural networks are trained to perform some task (either classification or regression). The training of various different Neural networks generally comprises of the same steps.

1. Initialise Weights and Biases
2. Forward Propagation
3. Compute Loss
4. Back-Propagation
5. Update weights

The Weights and Biases are the numerical values that are updated during training to allow the model to perform better at a specific task. Forward propagation passes the input through the ANN to get an output depending on the task [30]. We compute the difference between the desired output and the predicted output using a loss function. The difference, also known as loss, is passed back through the ANN in a process known as back propogation [30]. The loss is passed through each perceptron in the model. The weights and biases of the perceptron are then updated in such a way as to minimise the loss. This is done iteratively during training to improve the performance of an ANN on a specified task. There are two popular variations of ANNs known as Convolutional Neural Networks and Recurrent Neural Networks that make use of unique perceptrons and layers.

2.7.1 Convolutional Neural Network

Convolutional neural networks are a variation of a standard neural network most commonly used to analyse images, however they can also be used to model sequences [32]. We begin by describing CNNs in terms of images and later show how these methods can be applied to Sequences instead of images. Convolutional neural networks are commonly used to determine important features in an image and classify it accordingly [33]. Images are translated into tensors as shown in the figure below.

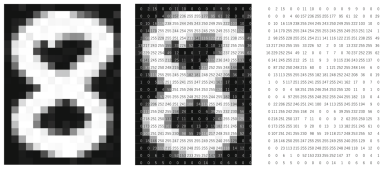


Figure 2.13: Vector Representation of '8' from MNIST dataset [34]

The grid of numbers can be interpreted as a tensor, which is fed into the CNN as the input. Figure 2.13 illustrate how CNNs use Grid data as inputs (CNNRadiology), as oppossed to ANNs that often use tabular data.
Modern CNN architecture was first established in 1990 by LeCun [32]. They were inspired by the structure of the visual cortex in animal brains[33]. Since then, many variants of the standard CNN has been proposed. CNNS are similar to standard ANNs in that they both are made up of perceptrons. [36]. CNNs are made up of three layers of perceptrons namely, the **convolutional**,**pooling** and **fully-Connected** layer.

Convolutional Layer

The Convolutional layer is the first layer in a standard CNN [33], that does most of the work in a CNN via a convolution operator. This layer acts as a filter and extracts the important features in an image [34] . The operator is the mathematical function that blends the two functions [35], which shows how grid date g is modified by filter $f \in \mathbf{R}^{N\times N}$ and returns a single value. The convolution of two functions f and g is shown in equation 2.5.

$$[f * g](t) = \int_0^t f(\tau)g(t-\tau)d\tau \tag{2.5}$$

Although equation 2.5 is the original form of the convolution operator, it is not useful when applying it to the tensor input of CNNs. We can consider tensors f and g where f smaller than g. We can rewrite the convolution operator using the summation form of an integral as follows [35]

$$G[f * g] = \sum_j \sum_k f(j,k)g(m-j, n-k) \tag{2.6}$$

The indices j, k represent the row and column numbers of vectors in the f. Indices m, n are the largest row and column number of g.

The convolution operator is used to extract important features in small parts of the grid of input data. This is done through the construction of a filter/ kernel. The filter is a small matrix of numbers [1] that is applied to the grid of input data [34]. Filters with different structures will detect different features, thus CNNs will often use many different filters in a single convolutional layer [37]. The number of filters in a convolutional layer is known as the *depth* of the layer [35]. In general, models with deeper layers have been shown to perform better according to [37]. The convolution is done many times by applying the filter over small portions of the input grid as shown in figure 2.14 [2] to produce a grid of values. The distance between two filter positions is knows as the **stride**, which is typically kept at 1 [36] but may vary. The mathematical representation of the convolution in equation 2.6 denotes f to be the filter/kernel and g to be the portion of input grid that the filter is applied to.

[1] Small relative to the size of the input. If the input is a 10 x 10 matrix of numbers the filter is a 3 x 3 matrix
[2] In this case the Filter and the grid portion are the same size, this is not always the case.

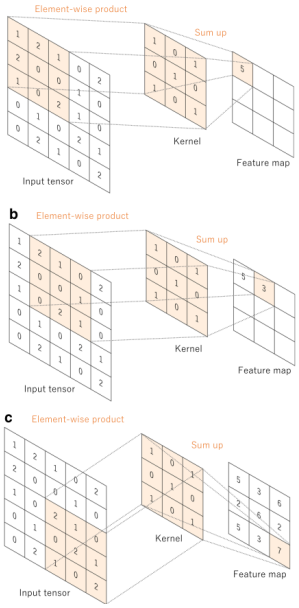


Figure 2.14: Visualisation of the Convolution Operation [34]

Convolutions are linear operation [33], which means nonlinearity is introduced by passing each output of a convolution through an activation function. Common activation functions in CNNs include tanh, sigmoid and ReLU functions [37]. The grid is known as a feature map after it has passed through the Activation function. Feature maps represents specific features in an input image such as visual patterns in computer vision tasks. [34]. Each convolutional layer will produce several feature maps as an output if it consists of several filters. These outputs are then passed into another convolutional layer as inputs. The structure of a CNN can become hierarchical, this occurs in[33], with feature maps from convolutional layers further in a CNN representing more abstract patterns in an image [35]. Nevertheless, the feature maps that are produced after several convolutional layers are fed into the pooling layer for dimensionality reduction.

Pooling Layer

Pooling layers typically occur at the end of all the convolution layers, however there are cases where it is inserted between convolutional layers [36]. The Pooling layer reduces the dimensionality of feature maps which can result in information about patterns in the data being lost [33], however this reduces the amount of parameters in the model [36] and speeds up computations [35]. The two most common pooling methods are Max Pooling and Average Pooling.

1. **Max Pooling**
   Max Pooling is a popular pooling operation applied to feature maps (CNNRadiology). It operates by scanning small sections of the feature map and selecting the largest value in each section.

2. **Average pooling** Average pooling calculates the average value of a section of the feature map and uses that to represent the whole section.

The sections of the feature map that are pooled can overlap, (this is known as **Overlap pooling**), however if the sections overlap, the resulting feature map will be distorted [36]. Stacking several pooling and convolutional layers can allow increasingly abstract features to be extracted, resulting in a feature map that captures hierarchies of features from the input data. The layers tend to focus on higher-level features and patterns, as the layers progress, allowing the model to learn complex structures in an image.

Fully-Connected Layer

Each feature map can be considered a matrix of vectors. Each feature map is fed into a single layer known as the fully connected layer [37]. This layer typically occurs before the output, but can occur elsewhere in the CNN. The Fully-Connected layer is characterized by each neuron in this layer being connected to all the neurons in the previous layers, which creates a network of connections that allows the FC layer to capture relationships between the various feature maps extracted by the previous convolutional and pooling layers [34].

Modifications from Images to Sequences

CNNs as explained above are primarily used on images, however, we can easily convert this to be used on a Sequence. This is done by using 1D filters in the Convolutional layer, instead of using $N \times N$ size filters, we use $1 \times N$ size filters [38] as shown in figure 2.15. This allows us to extract different features from different sections of the Sequence. We can similarly Pool the feature maps together and produce an output after utilizing a fully-connected layer. Although CNNs can be modified for sequences, it is slow to train and often assumes independence between elements in a sequence. Recurrent Neural Networks were thus invented specifically to work with Sequential data[37].

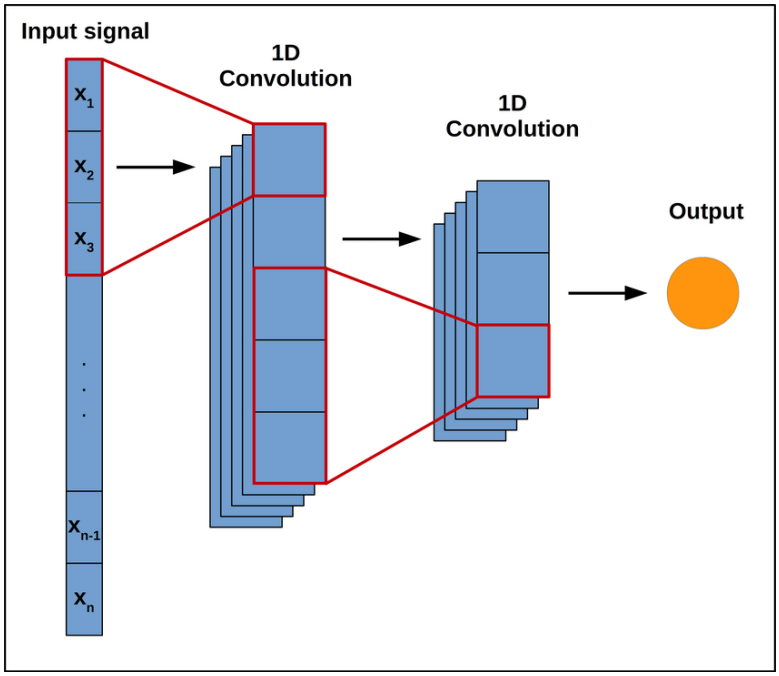


Figure 2.15: CNN for Sequences [38]

2.7.2 Recurrent Neural Network

Recurrent Neural Networks is another variation of an artificial neural network that takes in sequential or time-series data. A sequence of time series data can be rewritten as vector $X = (x_1, x_2, ..., x_t)$. Recurrent Neural networks (RNNs) differ from standard ANNs since the neurons of an RNN share parameters and are able to memorise previous inputs [39]. RNNs are made up of three layers, input, hidden and output [40]] as shown below in Figure 2.16. The left side of the Figure shows the RNN general form and the right side of the figure shows the unrolled form of a RNN. The input is fed into the RNN sequentially in the input layer as shown in Figure 2.16 below. Figure 2.16 shows how sequential data is passed into the network, once sequence point at a The input is transformed by first multiplying it by weight W_X and fed into the hidden layer.

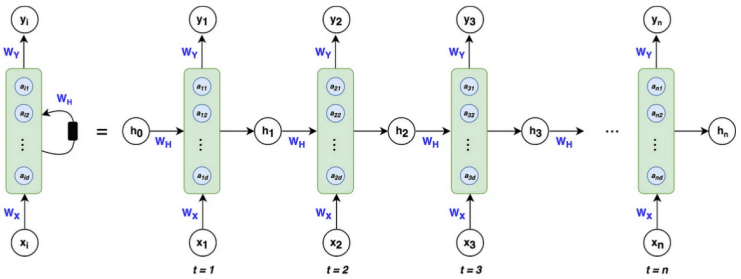


Figure 2.16: RNN architecture [39]

Hidden layers, also known as recurrent layers, is shown in green blocks which is made up the hidden nodes $a_1, a_2 ... a_j$ [40]. In general a single hidden layer has a single hidden unit. These hidden units and hidden layers consequently, can be stacked [41]. The hidden layer is the most important part of an RNN since the recurrent connections W_H in Figure 2.16 allows the model to remember past information about the sequence [42]. W_H, W_X and W_Y are the weight matrices that remain constant throughout the RNN [39]. These weights are used to produce weighted forms of the input, hidden and output vectors. Each layer can be represented by an equation. An RNN can thus be represented by a system of equations The equations of an RNN is shown below in equation 2.7. a_t represents the hidden unit(s), h_t shows the output of the hidden layer and y_t represents the output at time t.

$$\begin{aligned} a_t &= W_H h_{t-1} + W_X X_t \\ h_t &= tanh(a_t) \\ y_t &= f(h_t) \end{aligned} \tag{2.7}$$

Figure 2.16 illustrates that we can have several hidden layers and hidden units. Stacking several hidden layers on top of each other results in a deep RNN[41] that can predict and model more complex sequences. The unfolded form of a Deep RNN has a lattice structure show in Figure 2.17. Deep RNNs can predict more complex patterns, however they are trained using the exact same algorithm as a standard RNN.

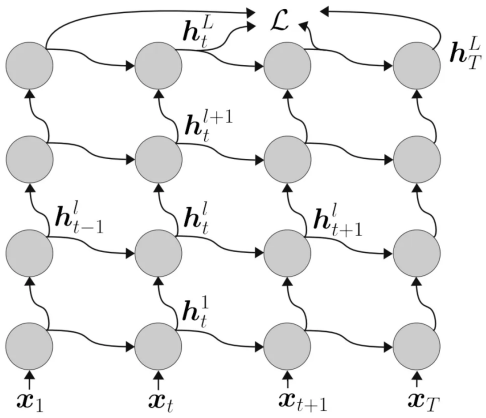


Figure 2.17: Deep RNN architecture [41]

Training

Training an RNN is similar to training an ANN. We begin by initialising the weight matrices W_H, W_X, W_Y. The input data is then propagated through the RNN to get output y_t. The predicted output y_t is then compared to the actual output z_t. The different between these two values is often computed by a Loss functions such as Euclidian Distance [40] or the Cross-Entropy Loss function [39]]. The loss function is then optimised via Backpropogation, which tells us how to vary the weight matrices to obtain y_t closer to z_t. The backpropogation

method used in RNNs is known as Backpropagation through time [39]. Given some loss function $L = g(y_t, z_t)$, the weight gradient wrt to W_Y can be computed by taking its' partial derivative with respect to W_H. This is done via the Chain rule as shown below. The gradient for W_X is also similarly computed using the Chain rule as shown in equation 2.8 where L is the loss, y is the output and h is the hidden state. Once we have obtained Gradients for W_H,W_X, and W_Y, we can simply update the weights as we would've done in a standard ANN.

$$\frac{\partial L}{\partial W_x} = \sum_{t}^{T} \sum_{k=1}^{t+1} \frac{\partial L_{t+1}}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_k} \frac{\partial h_k}{\partial W_{xh}} \tag{2.8}$$

Vanishing and Exploding Gradient Problem

RNNs have been largely replaced by LSTMs in many use cases due to the problems associated with RNNs. RNNs suffer from something known as the "Vanishing/Exploding Gradient Problem". If we look back at equation 2.8, we can see that W_X is computed via the chain rule. Each partial derivative in Equation 2.8 is also computed using the chain rule. These derivatives will vanish (become zero) if the previous gradients are less than 1. Equation 2.8 also illustrates why RNNs struggle to model long sequences. Input x_2 hsa a tiny influence on the prediction at x_{15}, since the dependence is based on $\frac{\partial h_{15}}{\partial h_2}$ which is a product of small gradients, resulting in an even smaller value[39]

Exploding gradients also occur in RNNS due to the chain rule. Large initial weights are large, will result in large partial derivatives. Since each gradient is made up of these partial derivatives, the result is an increasingly large gradient for the weights W [42]. This issue is largely negated by truncating gradients so it is often the Vanishing gradient that is the primary concern for RNNs.

2.7.3 LSTMS

LSTMs, otherwise known as Long-Short term memory models, were produced to solve the problem of vanishing gradients in RNNs [43]. LSTMs are a variation of an RNN that are capable of modelling long-term dependencies using an additional layer known as the cell state. The cell state is a continuous belt of that collects data as it is produced. The cell state is considered the global memory of the model, which allows it to model long-term dependencies [44]. The cell state C is shown in Figure 2.18 where it is represented as a horizontal line. Similarly to how Perceptrons are building blocks of ANNs, Figure 2.18 is an LSTM unit which is used to build a whole LSTM model.

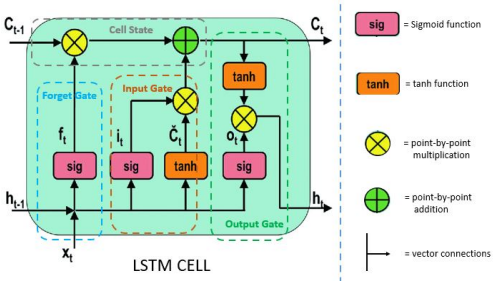


Figure 2.18: LSTM Neuron [43]

The cell state is edited using structures called gates. There are three primary gates in an LSTM, the forget, input and output gate. The forget gate is arguably the most important gate, since is used to decide how much information from the past should be stored (and how much to forget). The forget gate shown in Figure 2.18 takes the previous hidden state h_{t-1} as an input and passes it through a nonlinear activation function after some linear transformations [43]. It is common to use the sigmoid function since the output will range between

0 to 1 [46]. If the output is 0, none of the previous information is kept, if the output is 1, all the previous information is kept. The forget gate can be written mathematically as shown in equation 2.9 where σ is the sigmoid activation function. W_f is the weighted matrix of the forget gate and b_f is the connection bias. The forget gate allows the model to focus on how much attention is placed on an input as well as how much of the previous inputs to remember [46].

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ \sigma(t) &= \frac{1}{1 + e^{-t}} \end{aligned} \tag{2.9}$$

The next gate is the input gate. It decides how much of the information from the current time step should be kept [45]. The input gate returns two outputs, namely i_t and $\hat{C}_t$. i_t quantifies the importance of new information, $\hat{C}_t$ determines how the new information should be passed onto the cell state. The new information is also transformed using a tanh activation function to produce an output between -1 and 1 (c). If the value of $\hat{C}_t$ is negative i_t is subtracted from the cell state, if it is positive, i_t is added to the cell state. The equations for i_t and $\hat{C}_t$ are shown below in equation 2.10.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \hat{C}_t &= tanh(W_C \cdot [h_{t-1}, x_t] + b_c) \end{aligned} \tag{2.10}$$

The Cell can update the cell state at time t using only data from the forget and input gate. Given the previous cell state C_{t-1}, we can choose what to remember from it via the forget function f_t. Current data entering the Cell state is determined by $i_t \cdot \hat{C}_t$. The equation that iteratively updates the cell state is thus:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t \tag{2.11}$$

The current cell state C_t can now be used to generate the next hidden state. This is done via an output gate. C_t and h_{t-1} are both passed through the an activation function and then multiplied together to generate the next state (c). This is shown in equation 2.12 below [46]. The new cell state C_t and hidden state h_t are carried onto the next time step and LSTM unit until the end of the LSTM model.

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= tanh(o_t \cdot tanh(C_t) \end{aligned} \tag{2.12}$$

LSTMS are trained using the same methods in an RNN. The model still uses backpropogation through time to update the weights. Gradient descent is still used, however LSTMS's do not suffer from the vanishing gradient problem. The use of specialized gating mechanisms regulate the flow of information through the cell state, allowing gradients to flow more freely [37] during training which reduces the occurences of extremely small gradients. The forget gate is essential in preventing the vanishing gradients by selecting which data points to retain or which to forget [46].

A variation of LSTMs is a Bidirectional LSTM known as Bi-LSTM. These models are designed to capture dependencies from both directions in sequential data [47]. Unlike traditional unidirectional LSTMs, Bi-LSTMs operate in both forward and backward directions simultaneously. This means the model has two separate hidden states, one for processing data in the forward direction and one for processing it in reverse. Bi-LSTMs can capture contextual information from both past and future data for each element in the sequence [47]. The general architecture of an Bi-LSTM model is shown in figure 2.19 below.

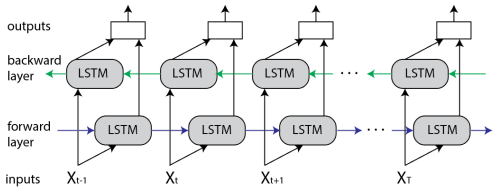


Figure 2.19: Bi-LSTM [47]

These hidden states of the forward and backward directions is combined in some way to produce the final output for each time step. Bi-directional LSTMS capture dependencies in the sequence from both forward and backwards data, allowing their outputs to be more accurate since it can take context into account. Bi-LSTMs have been used successfully in various NLP tasks such as sentiment analysis, speech recognition, and named entity recognition due to the important of context in these tasks[19]. Although Bi-LSTMs are well-performing sequence models, they have recently been overshadowed by Transformers, a sequence model described in Vaswani et al (2017).

2.8 Transformers and attention

One of the most prominent deep learning architectures that have emerged in recent years are Transformers. The model was introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017 and has since revolutionised the field of Natural Language Processing. Attention in the context of machine learning, allow models to correlate and relate different parts of its' input data [48]. This allows models to capture the structure and relations within datasets fed to models more accurately. Transformers similarly make use of attention in its architecture, in fact, it uses a mechanism known as "Self-attention" which allows it to process long sequences efficiently [49].

2.8.1 Self-Attention

Self-attention, also known as intra-attention, is a central mechanism to transformer models. It allows systems to weigh the importance of different parts of an input sequence.

Given a input sequence, $\mathbf{X} = (x_1, x_2, x_3 ... x_L)$, each element x_i can be represented by a d dimensional vector. The sequence is projected to three Weight matrices (W_K, W_Q, W_V) known as the Key weight, Query weight and Value weight matrices (c). The dimensions of the matrices are shown in equation 2.13 where d_K, d_Q, d_V are dimensions of the weight matrices that can be determined arbitrarily.

$$\begin{aligned} \mathbf{X} &\in \mathbf{R}^{L \times d} \\ W_K &\in \mathbf{R}^{d \times d_K} \\ W_Q &\in \mathbf{R}^{d \times d_Q} \\ W_V &\in \mathbf{R}^{d \times d_V} \end{aligned} \tag{2.13}$$

These weight matrices are multiplied with the input sequence $\mathbf{X}$ to get the following matrices according to [49]:

1. Key Matrix ($\mathbf{K} = W_K \mathbf{X}$): Represents the keys that elements use to provide information to other elements. Each row of the key matrix represents a key vector that is derived from $\mathbf{X}$.

2. Query Matrix ($\mathbf{Q} = W_Q \mathbf{X}$): Represents queries element x_i makes about other elements x_j. $\mathbf{Q}$ measures similarities between element within the same and other sequences.

3. Value Matrix ($\mathbf{V} = W_V \mathbf{X}$): This matrix represents the value of each element in $\mathbf{X}$.

self-attention computes attention scores for all pairs of elements in $\mathbf{X}$ by using $\mathbf{K}, \mathbf{Q}, \mathbf{V}$. The scores will tell each element how much attention should be given to other elements (c). The attention score is shown in the equation 2.14 where $\mathbf{K^T}$ is the transpose of $\mathbf{K}$

$$Self - Attention = softmax\left[\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_q}}\right]V \tag{2.14}$$

The Softmax function[3] in equation 2.14 is a row-wise normalisation function which means the activation function is applied to each row of a matrix. Although this self-attention mechanism is powerful, the row-wise computations often require a large amount of computational memory. Nevertheless, self-attention is a mechanism that allowed models to contextualise its' inputs which allows them to perform extremely well on various tasks [49]. Self-Attention is extended to Multi-Head attention in Transformers, where it employs several self-attention mechanisms at once in order to capture different relations in the data.

Multi-Head attention utilises the same Key, Query and Value vectors. Each vector is then split into smaller vectors $\mathbf{K_i}, \mathbf{Q_i}, \mathbf{V_i}$ which is collectively known as a head [48]. Each head applies self-attention to its' vectors. The outputs of the heads are then concatenated using some method to get an attention score. We will now study how this mechanism is utilised in Transformers

2.8.2 Transformer Architecture

The Transformer architecture described in Vaswani et al (2017) is shown in Figure 2.20 describes an encoder-decoder model. It consists of two structures known as the encoder on the left of the figure and the decoder on the right.

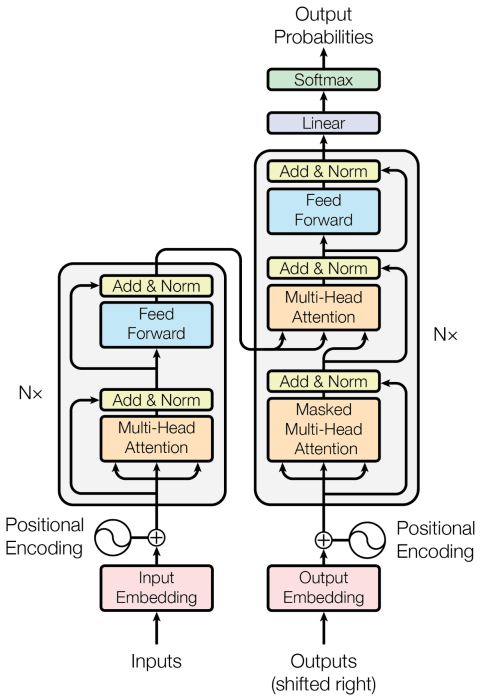


Figure 2.20: Transformer Architecture [48]

The encoder on the left, consists of n arbitrary encoding layers. Each encode layer consists of a Multi-Head attention layer and a Feed forward Neural Network. The Multi-head attention layer outputs several weighted sum values based on attention scores [48]. These values are fed into the neural network and a sequence of values

[3] activation function that transforms values into probabilities

is returned. Encoders are Seq2Seq models that outputs a high dimensional vector that represents information about the input sequence, such as relations between elements.
The positional encoding vectors have the same dimension as the input embeddings and are generated using sine and cosine functions [48]. These encodings are added to the input embeddings before they are fed into the encoder as to inject positional information into the model.

The decoder on the right will consist of n arbitrary decoding layers. Each decoding layer has three components a masked multi-head attention layer, a multi-head attention layer and a feed-forward neural network layer [49]. During training, the decoder is fed both the target data in the form of an output embedding with positional encodings as well as the encoder outputs. The output embedding is passed through a masked attention layer so the model can only look at generated parts of the encoding sequence, preventing it from "looking ahead" in the target sequence [49]. The next multi-head attention layer maps each tokens from the encoder output to an elements from the target sequence. This layer finds relations between the encoder outputs and output embeddings, allowing the model to contextualise its' inputs. The attention vector, after going through the unmasked multi-attention layer, can be fed into a feed-forward Neural network. This will transform the output vectors into a form that can be fed into another decoder block or a linear layer.

Transformers have been shown to perform extremely well on sequential data, however it possesses several issues. Transformers are computationally heavy models with quadratic time complexities [48]. The models also require a large amount of computational space, which increases with the model size and the length of the sequences. Transformers, most importantly, do not capture the positional dependencies between elements in a sequence well. Given datasets of short sequences and sufficient computational resources, Transformers can perform extremely well at its' given task, however, more often than not, models must run with limited computational resources on large datasets with long sequences. Other models such as Structured State Space Models, have thus been proposed as an alternative to Transformers as a sequence model.

2.9 Structured State Space Models

2.9.1 State Space Models

Any system can be represented by differential equations, however, increasingly complex systems often have increasingly complicated mathematical representations that are difficult to work with [50]. State space models simplify the mathematical representation of discrete or continuous dynamical systems by representing them using two linear equations [51]. The continuous representation of a State space model (SSM) is shown below:

$$\dot{x} = Ax(t) + Bu(t) \tag{2.15}$$

$$y(t) = Cx(t) + Du(t) \tag{2.16}$$

Equation 2.15 is known as the State Equation. $x(t)$ and $u(t)$ are $n \times 1$ vector that represent the state and input of the system respectively, where n is the dimensions of the system. A, B are $n \times n$ matrices [55] that are learned via gradient descent [52]. A is the state transition matrix that describes how the state evolves over time [51]. The state equation maps the input $u(t)$ to the state $x(t)$ of the system.

Equation 2.16 is known as the Output equation [50]. This equation maps the state of the system $x(t)$ to its' output $y(t)$. C, D are $p \times n$ matrices where p is the dimensions of the system output [55]. SSMs allow us to map inputs to the state of system and then directly to the system output making them useful when studying large complex systems [50]. Equation 2.15 and 2.16 reflect an SSM applied to a continuous system, however, SSMs can also be applied to discrete systems, but not directly. Matrices A, B, C, D must be discretized before

the SSM can be applied to discrete systems.

Matrices can be discretized via the Bilinear method [52], a common discretization transform used in digital signal processing. The relations between the discretized and continuous matrices is shown in equation 2.17.

$$\begin{aligned}\bar{A} &= (I - \frac{A}{2})^{-1}(I + \frac{A}{2}) \\ \bar{B} &= (I - \frac{A}{2})^{-1}B \\ \bar{C} &= C \\ \bar{D} &= D\end{aligned} \tag{2.17}$$

We can now use these discretized matrices to create an SSM that functions as a sequence-to-sequence map [53] from input u_k to output y_k. The discretized SSM is shown in equation 2.18 and 2.19. $\bar{D}u(t)$ is omitted since it can be viewed as a Skip connection[4].

$$x_k = \bar{A}x_{k-1} + \bar{B}u_k \tag{2.18}$$

$$y_k = \bar{C}x_k \tag{2.19}$$

SSMs have been applied to various field such as engineering, computer science and economics due to its' ability to solve a broad range of dynamical systems [51]. State Space model based deep learning architectures have been shown to perform better than standard deep learning models such as RNNs and CNNs [52]. Equation 2.18 and 2.19 represent a single SSM. Several SSMs can be stacked together to produce an SSM neural Network [54].

The SSM Neural network resembles the RNN in structure and thus suffers from the same problems when training. The 'RNN' form of the SSM can be unrolled and trained as a CNN to overcome these issues [52]. This is only possible since SSMs are linear time-invariant (LTI) systems[5] which means they can be rewritten as convolutions [55]. Recall that equation 2.5 shows how two functions can be convolved. The convolution of an LTI system with input $u(t)$ and ouput $y(t)$ can be written as shown in equation 2.20 [55].

$$y_k(t) = h(t) * u_k(t) \tag{2.20}$$

Equation 2.20 can be interpreted as the weighted sum of past and present input values. The unrolled version of this convolution is shown in equation 2.21. LTI systems can be characterized according to $h(t)$, the impulse response function. $h(t)$ meaures the response of a system to a unit pulse input (Berkeley). The unit pulse is often represented by the dirac delta function $\delta(t)$.

$$y_k(t) = h(0)u_k(t) + h(1)u_k(t-1) + h(2)u_k(t-2)... \tag{2.21}$$

Equation 2.18 and 2.19 can similarly be unrolled. The impulse function $h(t)$ of the discrete SSM is shown in equation 2.22. $\bar{K}$ also represents the kernel or filter of the convolutional SSM.

$$\bar{K} = (\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, ...\bar{C}\bar{A}^{L-1}\bar{B}) \tag{2.22}$$

The convoluted form of the SSM that utilises equation 2.22 as the impluse function is shown in equation 2.23. The convolution of an SSM can be computed using Fast Fourier transforms (FFT) rather than direct convolution shown in equation 2.5.

$$\begin{aligned}y_k &= \bar{C}\bar{A}^k\bar{B}u_0 + \bar{C}\bar{A}^{k-1}\bar{B}u_1 + \cdots\bar{C}\bar{A}\bar{B}u_{k-1} + \bar{C}\bar{B}u_k \\ y &= \bar{K} * u\end{aligned} \tag{2.23}$$

[4]Skip Connection: A shortcut connecting the output of one layer to the input of another non-adjacent layer

[5]r, time-invariant system if it obeys the laws of superposition and scaling over time. That is, if you observe an output signal $y_1(t)$ in response to an input signal $x_1(t)$, and later observe an output $y_2(t)$ in response to input $x_2(t)$,

The Fast Fourier transform shifts function in the time domain to functions in the frequency domain (Washington). The convolution theorem also states that multiplication of two functions in frequency space is equal to their convolution in time [55]. Computing convolutions using this method is quicker and more efficient [53]. The Discrete Fourier transform, which is a type of fast fourier transform is shown in equation 2.24.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i\frac{2\pi kn}{N}} \tag{2.24}$$

Each SSM can be considered as a single layer in a Neural network. Several SSMs can be stacked together to form an RNN-like architecture as shown in Figure 2.21.These SSM neural networks can now be used like RNNs or CNNs for classification tasks.

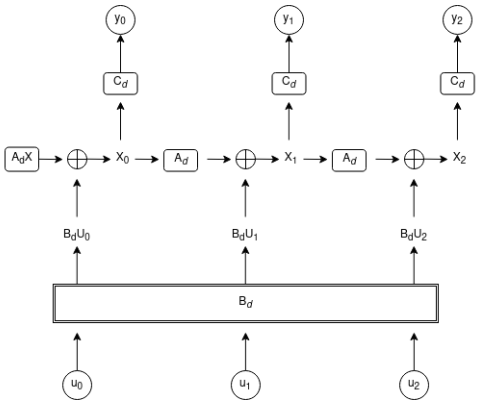


Figure 2.21: Structure of SSM Neural Network

2.9.2 Contributions of S4

HiPPO for long range dependencies

The HiPPO matrix is a class of transition matrices that allow the state $x(t)$ of a system to memorize the history of input $u(t)$. The HiPPO Matrix is shown in equation 2.25 below. The HiPPO matrix memorises its' previous inputs by keeping track of the coefficients of a Legendre polynomial [52] which can be used to approximate previous inputs.

$$A_{nk} = \begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & n > k \\ n+1 & n = k \\ 0 & n < k \end{cases} \tag{2.25}$$

Computing the Kernel quickly

The second contribution of S4 aims to compute the kernel $\bar{K}$ quickly by exploiting the form of Matrix A. The kernel constructed in equation 2.22 computes powers of $\bar{A}$ which is a time consuming process. This is done in three steps.

1. SSM Generating Function:
   We introduce a generating function that converts the SSM convolution filter fom the time domain to the frequency domain. [54]. This transofmration is known as the z-transform and commonly used in signal processing theory. The powers of $\bar{A}$ can be replaced by its' inverse. The truncated generating function is shown below.

$$\hat{K_L}(z) = \sum_{i=0}^{L-1} \bar{C}\bar{A}^i\bar{B}z^i = \bar{C}(I - \bar{A}^L z^L)(I - \bar{A}z)^{-1}\bar{B} \tag{2.26}$$

Since $z^L = 1$, we can produce a constant term $\tilde{C} = \bar{C}(I - \bar{A}^L z^L)$.

2. Diagonal Case:
   We next assume that matrix A has a special structure which allows us to compute its' inverse quicker than usual. If we expand $\bar{C}(I - \bar{A}^L z^L)(I - \bar{A}z)^{-1}\bar{B}$ using the discretised forms of matrices A and B:

$$\bar{C}(I - \bar{A}^L z^L)(I - \bar{A}z)^{-1}\bar{B} = \frac{2}{1+z}\tilde{C}\left[2\frac{1-z}{1+z} - A\right]^{-1} B \tag{2.27}$$

   If we assume that A is a diagonal matrix Λ then we can rewrite the generating function as shown in equation 2.28. This function is also known as the Cauchy kernel and used in many fast implementations.

$$\hat{K_\Lambda}(z) = c(z)\sum \cdot \frac{\tilde{C}_i B_i}{g(z) - \Lambda_i} = c(z)0 \cdot k_{z,\Lambda}(\tilde{C}, B) \tag{2.28}$$

3. Diagonal plus Low Rank (DPLR) Matrices:
   The assumption that matrix A will be a diagonal matrix is quite strict. We can therefore relax the restriction by allowing lower rank components. This is done by allowing Matrix A to be

$$A = \Lambda - PQ \tag{2.29}$$

   where $P, Q \in$. We can use the Woodbury Identity[6] to rewrite the generating function where A is a DPLR matrix.

$$\hat{K}(z) = c(z)\left[k_{z,\Lambda}(\tilde{C}, B) - k_{z,\Lambda}(\tilde{C}, P)(1 + k_{k,\Lambda}(Q, P)^{-1} k_{z,\Lambda}(Q, B)\right] \tag{2.30}$$

   Another benefit of using DPLR A matrix is that we can compute the SSM RNN without using its' inverse at all. This is done by substituting equation 2.29 into the equations that discretize the matrices of the SSM, namely equation 2.17. We can define A_0 and A_1 using the Woodbury Identity along with other algebraic methods. Their expressions are shown in 2.31.

$$\begin{aligned} A_0 &= 2I + (\Lambda - PQ) \\ A_1 &= D - DP(1 + QDP)^{-1}QD \\ D &= (2 - \Lambda)^{-1} \end{aligned} \tag{2.31}$$

   The discrete SSM can therefore be rewritten as:

$$\begin{aligned} x_k &= \bar{A}x_{k-1} + \bar{B}u_k \\ &= A_1 A_0 x_{k-1} + 2A_1 B u_k \\ y_k &= Cx_k \end{aligned} \tag{2.32}$$

4. Transforming HiPPO to DPLR
   The HiPPO matrix is a Normal plus Lower Rank matrix (NPLR). Since normal matrices are diagonizable, NPLR matrices can be transformed into DPLR matrices[54], which can be used, as shown above, to generate an SSM.

We have now shown how S4 can be used to memorise long sequences and train efficiently on large data by utilising the form of the HiPPO matrix. The S4 Sequence model has been shown to outperform Transformers and many other sequence modelling architectures on the Long range Arena benchmark tests [54] indicating that performs well given sequences with +16 000 sequences. Since Protein sequences can go up to 35 000 amino acids that interact when folding [2], S4 is the ideal model to use to predict the Biological Process labels of a protein sequence. However, before S4 was formulated, many other machine learning algorithms were used to classify proteins.

[6] Insert woodbury identity here

2.10 Protein classifications previously

Protein classification based on the its' labels assigned by the Gene Ontology Consortium has been a topic of interest in recent years. There have been attempts to classify proteins using machine and deep learning methods with varying degrees of success. The CAFA challenge is one of the most well known bioinformatic challenges [16] that has been used to discover and verify the labels assigned to protein sequences using various machine learning methods.This section aims to outline some notable and baseline models that have been used to classify proteins and their degree of success.

2.10.1 Support Vector Machines

Support Vector machines (SVMs) were one of the first machine learning algorithms that were used to classify protein functions according to their ontology. Cai et al (2003) [56] was one of the first papers that attempted to classify protein sequences according to their functional classes or Pfam superfamilies. Proteins in the same superfamily often performed similar functions and thus have similar structures and physiochemical properties. The SVM in [56] constructed nine feature vectors representing these physiochemical properties in 20 or 21 dimensions. These physiochemical properties include amino acid composition, charge and solvent accessibility amongst other properties.

Amino acid sequences from seven Superfamilies, namely, RNA-Binding proteins, homodimer proteins, drug absorption proteins, Class I and Class II drug metabolizing enzymes are gathered from the Pfam database. The amino acid sequences are then transformed into feature vectors which can be used to train the SVM. The SVM utilised consists of a nonlinear kernel and trained on some amount of proteins. The results of the SVM are shown in Figure 2.22.

Test results for different protein functional classes

Protein class	Training set		Testing set				Independent evaluation set				Sensitivity (%)	Specificity (%)	Q (%)
	Positive	Negative	Positive		Negative		Positive		Negative				
			TP	FN	TN	FP	TP	FN	TN	FP			
RNA-binding protein	871	1120	610	2	1153	4	613	127	898	80	82.8	91.8	88.0
Homodimer	349	368	88	0	97	0	88	3	82	9	96.7	90.1	93.4
Drug absorption	228	1651	53	0	2219	3	83	41	1377	59	66.9	95.8	93.6
Drug delivery	916	2037	404	5	1827	16	431	198	1360	81	68.5	94.3	86.5
Drug excretion	262	2631	59	11	2900	2	66	84	2102	161	44.0	92.8	89.8
Class-I drug metabolism	117	3032	82	15	2606	1	76	8	2190	123	90.4	94.6	94.5
Class-II drug metabolism	84	3031	53	10	2600	4	42	7	2315	7	85.7	99.6	99.4

Figure 2.22: Results of SVM from [56]

The results are shown in terms of true and false negatives as well as true and false positives. Sensitivity, specificity and accuracy Q are defined in equation 2.33. The results show that preliminarily, SVMS are a good candidate for predicting protein function, since the overall accuracy of the predictions during testing only varied from 86.5% to 99.4%. This SVM performs well given the task of using features vectors to classify protein structures into one of seven functional classes, however, realistically we need an algorithm that can classify sequences into more than seven classes.

$$\begin{aligned} Sensitivity &= \frac{TP}{TP+TN} \\ Specifiity &= \frac{TP}{TN+TN} \\ Q &= \frac{TP+TN}{TP+TN+FP+FN} \end{aligned} \tag{2.33}$$

Deen & Gayanchandi (2019) [57] thus attempted to similarly use SVMs to classify proteins according to their Gene Ontology. The dataset used in this paper consisted of a subset of the SwissProt data. The dataset is made up of 45 518 proteins from species Human, Mouse and Rat. Each function in this dataset is also assigned to 30-100 proteins. The functions in this dataset are represented by Biological process Gene Ontology labels.

K-fold cross validation is applied to the SVM while training. The kernel type in the SVM also varied from Linear to nonlinear types. The accuracy Q of these SVMs when used to classify protein sequences according to their function is shown in Figure 2.23. The SVM with an RBF[7] kernel performed the best at the classification task with an accuracy of 97.09% [57].

Protein datasets [Human, Mouse, Rat]	Overall Accuracy%
SVC with Linear Kernel	58.13
Linear SVC with Linear Kernel	63.03
SVC with RBF kernel	97.09
SVC with Polynomial Kernel	68.89

Figure 2.23: Accuracy of SVMs with different Kernels [57]

Although SVMs have been shown to perform well when classifying Proteins into functional classes, SVMs are often not used in protein function annotation. SVMS can only classify sequences into groups that are already predefined and consist of many members. Many protein functional classes (GO) and superfamilies (Pfam) consist of very few proteins, with some classes consisting of only a single protein resulting in bias data. SVMs are not robust to extreme bias in datasets [28]. Since single sequence functional classes make up a substantial amount of annotated protein databases, SVMs cannot be applied to large parts of protein databases. Since Protein databases such as Swissprot consist of large amounts of data, we can use data-hungry deep learning models to annotate proteins.

2.10.2 CNN-Based: DeepGO and DeepGOplus

One of the earliest Deep learning methods applied to the task of protein annotation is Convolutional Neural Networks. The CNN was trained on a subset of the Swissprot dataset that consisted of 60 710 proteins annotated with 19 181 BP labels, 6221 MF labels and 2358 CC labels [58]. Three CNN models are trained to predict labels in each subontology (BP, CC, MP). The output of these models is fed into another Deep Model that relates the GOs of the different sub-ontologies. Since many labels only occur once, we only considered the terms that appeared the most in the dataset. This means we only considered the 932 most frequent terms in BP, 589 most frequent terms in MF and 436 most frequent terms in CC. Each sub-dataset is then split in an 80/20 train test split before it is fed into its' respective model.

Protein sequences and their annotations must be translated into machine readable vectors before they can be fed into a CNN. The labels are converted into one-hot embeddings where the presence of a label is reflected by a 1 and its absence reflected by a 0 [58]. The Protein sequences are translated used AA trigrams where every three consecutive Amino acids is reflected by a numeric value (c). This allows sequences to be translated into dense vectors which have been found to perform better than one-hot embeddings. The trigrams are mapped to a specific numeric value using a lookup table. The mapping between the trigram and numerical values is finetuned as part of the first layer of the CNN model. The CNN model used is a 1D Convolution layer with 32 filters of size 128. These filters are applied to the embedded sequence. The use of multiple filters allows us to learn of different aspects of the data. The output of the 1D max-pooling layer is a vector with length of 832. This vector was combined with vectorised data on a sequence's protein–protein interaction (PPI)[8] networks via a fully connected layer. The resulting vector is then passed through a Hierarchical Neural Network Model, that

[7]Radial Basis Function
[8]PPI data is obtained from an external database

relates the sub-ontology labels to one another. The architecture and details of the Hierarchical Neural network, although important, is beyond the scope of this paper due to our primary focus on sequence modelling. The general model architecture of DeepGO is shown below in Figure 2.24

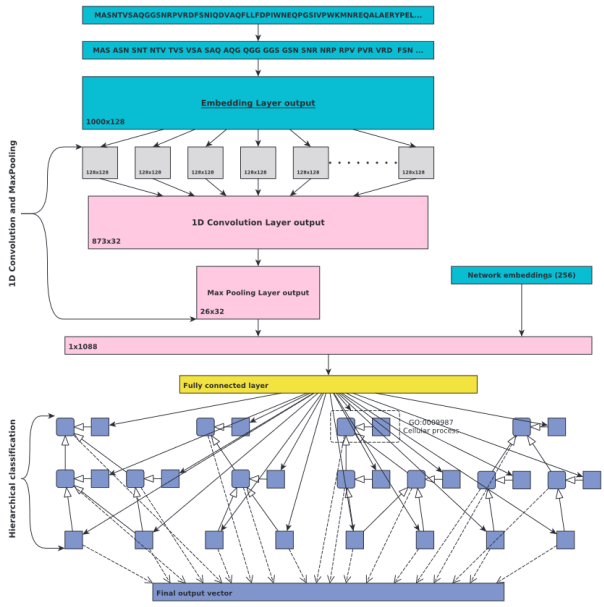


Figure 2.24: DeepGO Architecture [58]

DeepGO, although performing well, had several shortcomings. These shortcomings were fixed by its' successor, DeepGOPlus [59]. This new model had a similar architecture to its' predecessor, however, it could take in protein sequences that consisted of up to 2000 amino acids, compared to the 1002 amino acid limit of DeepGO. It also no longer required PPI data, which was sparse for most sequences. DeepGOPLus does retain the CNN models architecture [59], however, instead of three models, it only uses one that is now trained on a larger sequences and sub-ontology dataset that consists of 10,693 molecular function (MF) classes, 29,264 biological process (BP) classes and 4,034 cellular component (CC) classes. The CNN architecture is shown below in Figure 2.25. It should be noted that the sequences are now translated into one hot embeddings instead of dense vectors as before to reduce overfitting but the Labels are translated using the same method as before.

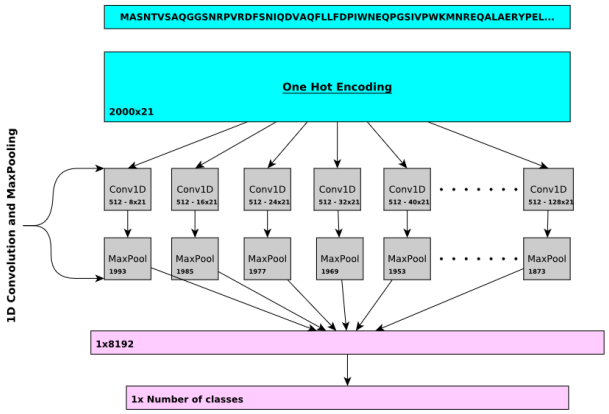


Figure 2.25: CNN model architecture in DeepGOPlus [59]

The final GO predictions made by DeepGOPlus is a weighted sum of the predictions made by the CNN models as well as Diamond BLAST, a quicker and more accurate version of BLAST. The performance of

DeepGO, DeepGOPlus and two SOTA performing models (at the time) on the CAFA 3 Dataset is shown below in Figure 2.26. The Metrics used in this case are F1 scores, S_{min} and Area under precision Recall Curve (AuPRC). S_{min} is a metric designed specifically for protein annotation, however its' robustness and use is not well known [60]. A F1 score > 0.7 and a AuPRC > 0.8 is considered good [61]. Although none of the models shown below are considered "good" by the metrics, DeepGoPLus still performs better than its' counterparts and predecessors. In Figure 2.26 below, DeepGOCNN represents the prediction made by just the CNN model without the input from Diomand BLAST [59].

Method	F_{max}			S_{min}			**AUPR**		
	MFO	BPO	CCO	MFO	BPO	CCO	MFO	BPO	CCO
Naive	0.290	0.357	0.562	10.733	25.028	8.465	0.130	0.254	0.456
DiamondBLAST	0.431	0.399	0.506	10.233	25.320	8.800	0.178	0.116	0.142
DiamondScore	0.509	0.427	0.557	9.031	22.860	8.198	0.340	0.267	0.335
DeepGO	0.393	0.435	0.565	9.635	24.181	9.199	0.303	0.385	0.579
DeepGOCNN	0.420	0.378	0.607	9.711	24.234	8.153	0.355	0.323	0.616
DeepGOPlus	**0.547**	**0.470**	**0.625**	**8.660**	**22.514**	**7.804**	**0.489**	**0.405**	**0.631**

Figure 2.26: Results of DeepGOPlus [59]

DeepGO and DeepGOplus is a baseline Deep Learning models for the task of Protein annotation to which all subsequent models are compared. One of the largest issues with these CNN backbone models is their computational costs [34]. RNN models are Deep Learning models specifically designed for sequential data that are not computationally expensive.

2.10.3 RNN-Based: ProLanGO

RNN models such as LSTMs are designed to take in sequential inputs and are thus ideal for protein annotation. RNNs have been extensively used in linguistic Machine Translation in the form of Neural Machine Translation Models, which are made up of two RNN (LSTM) models [62]. The architecture of a Neural Machine Translation Model is shown below in Figure 2.27 where both the encoder and decoder are LSTMs.

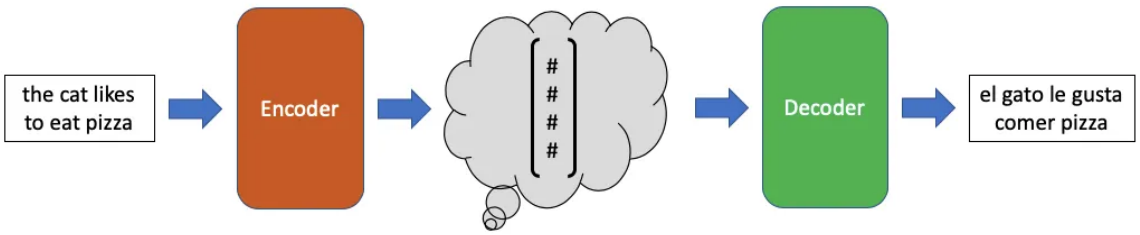


Figure 2.27: Neural Machine Translation Models [62]

Cao et al (2016) utilises the similarities between Sentences and Protein sequences to construct a NMT model for the task of Protein Annotation [63]. Protein Sequences and GO labels are first converted into two languages dubbed "ProLan" and "GOLan". Protein sequences are converted into a language by interpreting the most frequent 1000 k-mers in the dataset as words. Each protein sequence is divided into protein "words" based on this database, with preference for longer words. Any unmatchable consecutive amino acids forming new "words" that are added to the database. IN this way the Protein sequences become its' own language with a datasbase of 420 12 words[63]. Gene Ontology terms are also converted into a language using unique "Alphabet IDs" through depth-first searches on three separate hierarchies since GOs exist in a Directed Acyclic graph. These IDs are derived based on the order of encountering GO terms and are represented as four-letter codes using a base-26 alphabet. This efficiently organizes the GO labels of all sequences and converts the GO labels into a language with 25 160 words[63]. The conversion of protein sequences and GO labels into languages transforms

the protein annotation task into a language translation one.

ProLanGO is a neural machine translation model that can efficiently translate "ProLan" inputs into "GOLan" outputs which are then decoded to produce the corresponding GO Labels [63]. This is done through two RNN models. The first encodes the ProLan sequence into a machine readable vector. The second RNN decodes the vector into a GO Lan sequence that can be converted to GO Labels. Both RNNs are trained together so the model can better understand relations between ProLan and GOLan. Protein annotation is a multi-label classification problem. ProLanGO breaks this down into several single-label tasks that are addressed by seperate modules of the RNNs. The architecture of the NMT model of ProLanGO is shown below in Figure 2.28

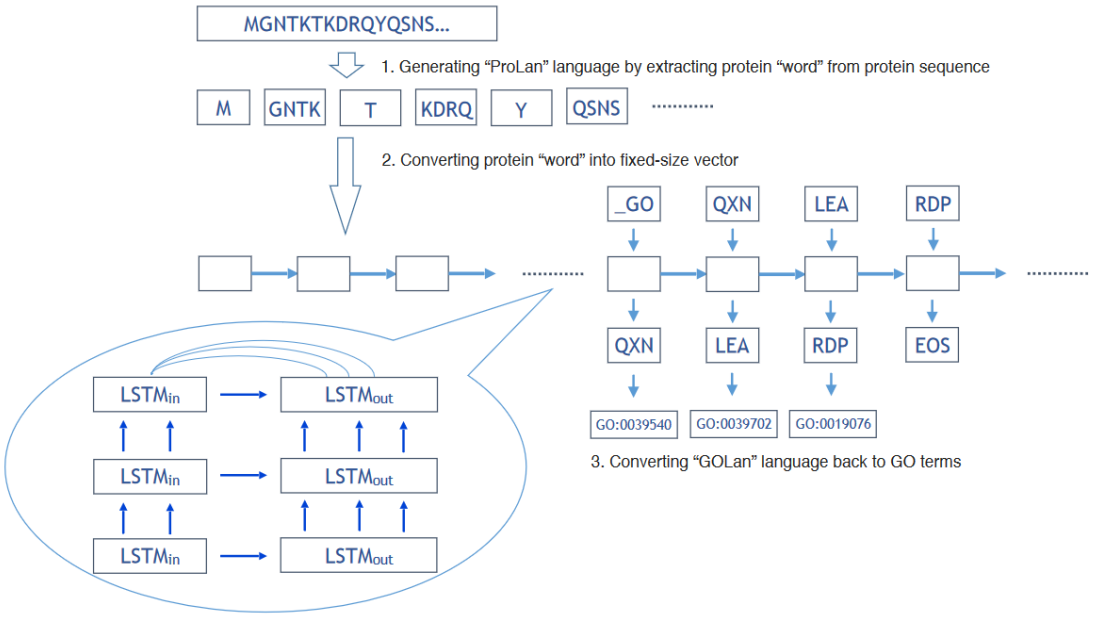


Figure 2.28: Architecture of NMT model used in ProLanGO [63]

The two RNNs used in this NMT model shown in Figure 2.28 are LSTMs since LSTMs do not suffer from gradient problems. Due to the nature of the Decoding RNN, the maximum number of GO labels that can be predicted by the model is 10 per protein sequence. Cao et al (2016) extends the maximum number of GO labels to 60 by creating buckets shown in equation 2.34 where each bucket is desribed as (no. words ,maximum length). An Extended NMT Model can be produced using these buckets than can predict up to 60 GO Labels per sequence. The Extended NMT model as well as the primary NMT model, is combined to create ProLanGO[63].

$$buckets = [(64, 25), (164, 50), (250, 60)] \tag{2.34}$$

Each NMT model is trained on 419 192 protein sequences and tested on 104 798 sequences from the Uniprot dataset. Both models are trained for 300 000 timesteps and evaluated using Precision and Recall. The Precision Recall curve for the NMT model, the Extended version and ProLanGO is shown in Figure 2.29 below. The exact AuC values for the NMT model, its extended version and ProLanGO is 0.286, 0.305, 0.333.

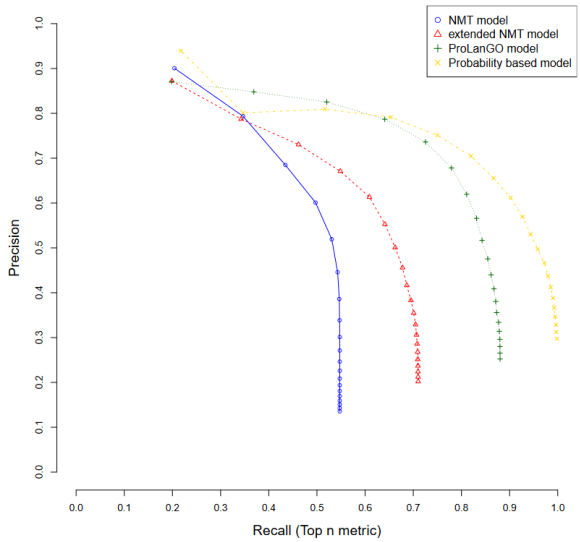


Figure 2.29: Precision Recall Curve of ProLanGO and its related models on CAFA 3[63]

ProLanGO is a novel approach to the protein annotation task, based on RNNs. This model explicitly utilises links between Proteins and Languages to apply an NMT model that is commonly used in NLP to Protein Sequences. This leads to the idea that other models commonly used in NLP can be applied to Protein Sequences [17]. One of the most common Models used in NLP are Transformers.

2.10.4 ProteinBERT and TemPROT

Transformers are SOTA models in Natural Language processing that can capture relationships between elements in a sequence. Due to the similarities between sentences and amino acid sequences, as described in Section 2.5.1 we can apply transformers to protein sequences to annotate the sequences using their functional classes. This was firstly done using Bidirectional Encoder Representations from Transformers (BERT) as described in [64].

BERT is a variation of the standard transformers models that were described by Google in 2018. Since then they have been used to perform many different tasks. ProteinBERT developed by Brandes et al (2022) has been trained on 106 Million Protein sequences from UniProtKB, Only the 8934 GOs that occurred most frequently were considered [64]. Only 46 Million protein sequences had GOs labels after this reduction. Each sequence is tokenized using 26 tokens. 22 of these tokens represent the 22 amino acids, 3 are [START], [END], [PAD] tokens and one additional token represents any miscellaneous amino acids.The GO elements are all encoded as binary vectors of size 8934 with zero entries unless it corresponds to a GO of that particular protein sequence.

ProteinBERT was developed to accomplish many protein related tasks, however it is primarily pretrained to predict protein seqeunces and GO annotations. This is done by by feeding in corrupted sequence or GO data into the model and instructing it to recover the uncorrupted form of the data [64]. In the task of protein annotation, the model is fed uncorrupted protein sequences and their corresponding GO annotations that are slightly corrupted. This data is corrupted by adding false GO annotations with a probability of 0.01 and removing GO annotations with a probability of 0.25, 50% of all annotated proteins without their corresponding GO annotations were similarly fed into the model, however in this case, the model is told to predict the GO annotations, instead of recovering "uncorrupted" data. After training, the model is test benchmark set, where each protein sequence has less than 40% similarity to any proteins in the training dataset. The model is further fine-tuned on the Test Benchmark datasets to improve its' performance at specific tasks.

Although ProteinBERT is a BERT based model, it contains features that are not often prevalent in BERT

Models. Figure 2.30 shows the structure of ProteinBERT. The model consists of two parallel paths, one that considers local representations and the other global. These only interact at a single layer as shown below. More details about the Model architecture can be found in [64].

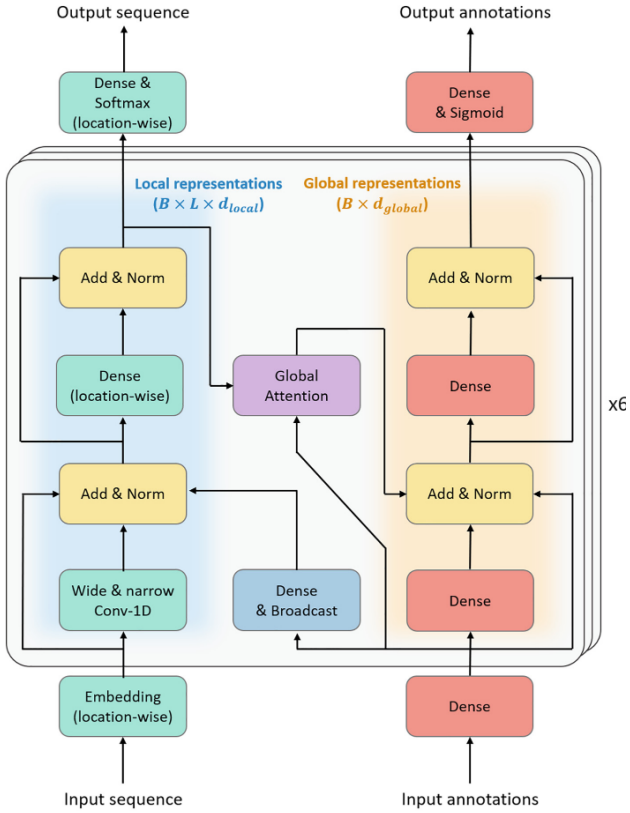


Figure 2.30: Architecture of ProteinBERT[64]

An important note is that ProteinBERT does not employ a Self-attention mechanism as described in section 2.8.2, but rather a Global attention mechanism. The Global attention layer takes in both local and global representenatations and outputs a global fixed-size vector. This attention mechanisms essentially utilises the Global input to determine the level of important of tokens in the local input.

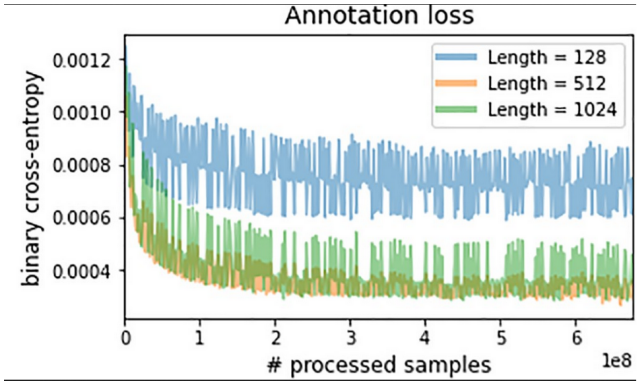


Figure 2.31: Loss function of ProteinBERT during Training[64]

ProteinBERT was trained on a single GPU for 28 days and finetuned for 14 days. This is in direct contrast to other protein sequence models that have been run on multiple GPUs. ProteinBERT has been shown to do as well as any other State of the Art (SOTA) model on many Protein Related tasks such as structure and homology prediction, all whilst being trained on smaller datasets with less computationally power [66]. Brandes

et al (2022) does not, however, evaluate ProteinBERT on GO annotation tasks however the loss achieved on this task during training is shown in Figure 2.31 where each sequence was tokenised usign different token lengths.

ProteinBERT is a transformer-based Model that performs well on a variety of Protein Related tasks. [64] point out that ProteinBERT is a computationally efficient version of the model that performs just as well as the heavy SOTA models. This indicates that Transformer-based models can be a good candidate for GO annotation prediction. Oliveira, Pedrini & Dias (2023) recently released TEMPROT [65]which similarly uses a BERT-based model to predict GO annotations. TEMPROT is uses ProtBERT-BFD [66] to produce embeddings for the Protein sequences. ProtBERT is a version of BERT that is further finetuned on BFD-100 datasets with more hidden layers. ProtBERT was first trained for 800k steps on sequences with a max length of 512 and then later trained form another 200k steps on sequences with a maximum length of 2k tokens[66].

TEMPROT is trained on the CAFA3 dataset which is their most recent and available dataset that reports on both the methods and results. TEMPROT was then trained on 47691 sequences with BP labels, 45309 sequences with CC labels and 32421 sequences with MF labels[65]. ProtBERT is a BERT-based model that utilises a self-attention mechanism that has a quadratic space complexity. Due to this constraint, ProtBERT cannot cope with sequences longer than 512 amino acids[66]. TEMPROT thus uses a sliding window technique to mitigate this problem. The window slices sequences into chunks of 500 amino acids. Additional slices are created from these chunks by joining the last 250 amino acids of the first slide and the first 250 amino acids of the next. This is illustrated in figure 2.32 below. Each slice is then passed through ProtBERT-BFD to produce a 1×1034 feature vector. Slices belonging to the same protein are then aggregated using a mean operator.

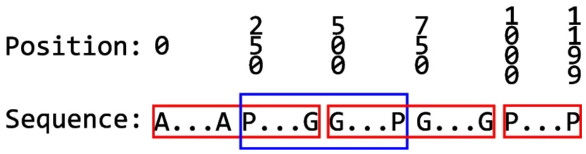


Figure 2.32: The Sliding window technique applied to a 1200 amino acid protein. The red squares represent the standard slices whilst the blue represent the additional slice [66]

The Feature vector is then fed into a meta classifier that is made up of a neural network with a single hidden layer and 1000 perceptrons. The neural network was trained using Binary cross entropy loss and an Adam Optimizer for 100 epochs with early stopping and learning rate decay. TEMPROT+ was also developed by Oliveira, Pedrini & Dias (2023)[65], which is just TEMPROT combined with BLAST using ensemble methods to produce better predictions [66]. The results of TEMPROT and TEMPROT+ compared to previous baseline models is shown in Figure 2.33.

Method	F_{max}			AuPRC			IAuPRC			S_{min}		
	BP	CC	MF	BP	CC	MF	BP	CC	MF	BP	CC	MF
Naive	0.402	0.611	0.446	0.266	0.521	0.228	0.345	0.634	0.370	25.423	10.268	9.349
CAFA-BLASTp	0.468	0.469	0.551	0.208	0.215	0.287	0.215	0.216	0.296	38.083	18.755	9.124
DeepGO	0.337	0.379	0.489	0.247	0.257	0.309	0.304	0.382	0.465	27.414	11.880	8.821
DeepGOPlusCNN	0.498	0.664	0.531	0.444	0.637	0.460	0.465	0.634	0.528	23.799	9.783	8.240
TALE+Transformers	0.491	0.661	0.550	0.477	0.613	0.444	0.469	0.706	0.549	23.929	9.682	8.115
ATGO	**0.547**	0.684	0.616	**0.506**	**0.667**	**0.623**	**0.524**	**0.724**	0.632	**22.228**	9.437	7.228
TEMPROT	0.499	**0.689**	**0.643**	0.459	0.639	0.561	0.483	0.719	**0.664**	23.652	**9.209**	**6.973**
DIAMOND	0.519	0.593	0.572	0.286	0.237	0.320	0.417	0.483	0.462	23.066	9.957	7.164
BLASTp	0.561	0.637	0.620	0.402	0.380	0.360	0.502	0.586	0.562	22.183	9.795	6.805
DeepGOPlus	0.553	0.677	0.619	0.514	0.638	0.559	0.536	0.717	0.635	22.648	9.515	7.090
TALE+	0.555	0.681	0.631	0.547	0.643	0.621	0.540	0.724	0.643	22.615	9.363	6.949
ATGO+	**0.589**	0.690	0.652	**0.550**	**0.660**	**0.650**	**0.571**	**0.731**	**0.689**	**21.233**	9.286	**6.617**
TEMPROT+	0.581	**0.692**	**0.662**	0.529	0.641	0.595	0.558	0.728	**0.689**	21.892	**9.169**	6.662

Figure 2.33: Performance metrics of each protein annotation model [66]

ATGO is a deep learning model comprised of three neural networks embedded with an LLM (c) that performed almost as well, if not better, than TEMPROT as shown above. However, TEMPROT and TEMPROT+ are shown to have outperformed many SOTA deep learning models that were designed to predict GOs. This is particularly true when considering the F_{Max} metric. This is particularly true for GOs relating to Molecular Function ($F_{Max} = 0.643$) and Cell Component ($F_{Max} = 0.689$). TEMPROT performed the best when predicting MF ontologies, however struggled the most with BF labels [66].

Transformers have shown to be a powerful and useful tool that can be used in Protein Annotation tasks, however their architecture has many weaknesses that also make them unsuitable for the task. Protein sequences are amino acid chains that often consist of more than 500 amino acids. Due to the quadratic space complexity of attention mechanisms in Transformers [65], they cannot be used for long sequences. The computational time and memory used to train Transformer models is also longer than other Deep Learning models [53].

Transformer are the most recent models that have been used to accomplish the protein annotation task. However, as shown in this section, several other models have also been used. This section is non-exhuastive since there are several models that use a combination or variation of the models described above. Google, for instance, has developed a CNN-based model known as ProtENN that utilises ensemble methods and several CNN models. The models described in this section all highlight two primary problems in protein annotation. The long sequences, and their dependencies, are not easily understood by models and any models that can understand these dependencies are computationally heavy. Structured State Space Models have been shown to be able to both accomodate long sequences as well as reduce the computational complexity of its' training. This makes Structured State Space Models an ideal model architecture to use in Protein function annotation.

Chapter 3

Data and Methods

3.1 Exploratory Data Analysis

We use the Swissprot Human Protein dataset, which contains 20422 protein sequences in total and 17232 that have biological process labels that have been verified experimentally. The average protein sequence length 1114. The longest protein sequence, known as titin, consists of 34 350 amino acids. Titin is the largest protein structure that is currently known and it is responsible for maintaining the elasticity in muscles. It is comprised of 244 individually folded domains that are connected by unstructured peptide sequences (cite). The InterQuartile Range of the the protein sequence length is 842. Figure 3.1 shows the frequency distribution of protein sequence lengths in the dataset without outliers. Any sequences with more than the upper fence of 2602 amino acids are considered outliers. The upper and lower fence of the data is determined using equation 3.1.

$$\begin{aligned} Upper &= Q_3 + 1.5 \cdot IQR \\ Lower &= Q_1 - 1.5 \cdot IQR \end{aligned} \tag{3.1}$$

The lower fence in this case is not considered since $Lower = -766$. Removing outliers removes 1320 sequences, resulting in the frequency distribution of the lengths of the remaining 19102 sequences shown in the Figure below. Outliers increase the variability of the dataset resulting in distorted standard deviation and mean values. Figure 3.1 shows that without outliers, the data is shown to exhibit a Poisson distribution.

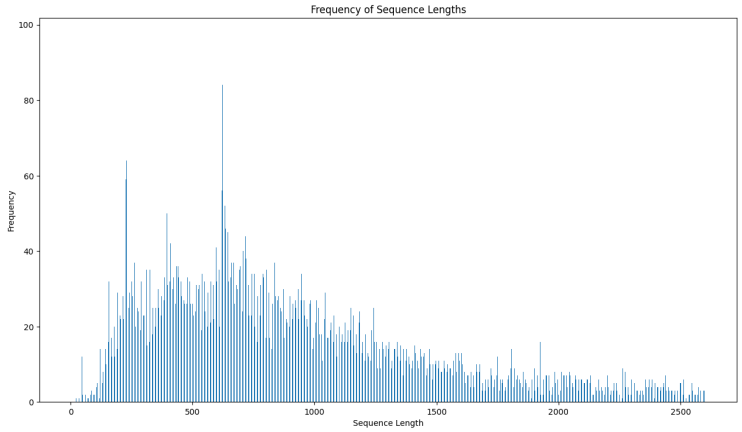


Figure 3.1: Sequence length Frequency distribution

The median of the data before outliers is 829 but after outliers it becomes 779 after the outliers are removed. As shown in Figure 3.1, the most common protein sequence length is 233. The dataset has a total of 12448 BP

Labels. The protein sequence with the most BP labels is the tumor necrosis factor with 185 BP Labels. The dataset is extremely bias with 3313 biological process labels only occurring once as shown in figure 3.2.

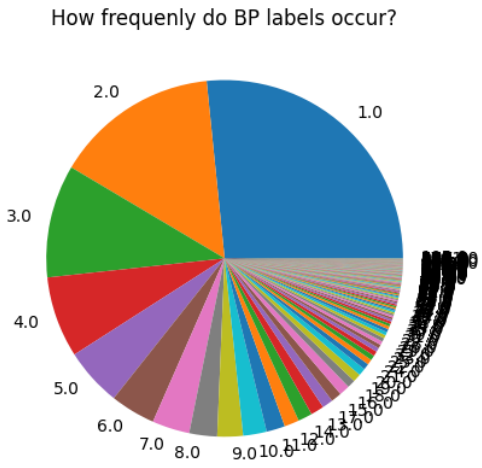


Figure 3.2: Pie Chart showing how frequent BP Labels occur

The BP labels that occur the most are GO:0006357, GO:0045944, GO:0007165 which occur 1174, 1040 and 1001 times respectively. Labels GO:0006357 and GO:0045944 are associated with regulatory processes in transcription by RNA polymerase. Label GO:0007165 is, however, associated with signal transduction. Figure 3.2 illustrates that over 70% of the labels occur less than 11 times, however the most common BP label occurs 1174 times. One way to reduce the bias of the dataset is to cluster the labels to reduce the number of labels that occur once. The functions that each label represents is tokenized and vectorized using TF-IDF and bioBERT. These embeddings are then clustered using either Hierarchical or K-means clustering.

We use silhouette scores to evaluate the quality of the clusters. The score quantifies how well each sample in a cluster is separated from samples in other clusters. The silhouette score ranges from -1 to 1, with higher values indicating better clustering results. We calculate the silhouette score using equation 3.2 and 3.3, where $a(i)$ be the average distance between sample i and all other samples in the same cluster, $b(i)$ is the average distance between sample i and all the samples in the closest neighboring cluster. The silhouette score for sample i is denoted $s(i)$ and averaged with all samples in the cluster to generate the Silhouette score. The averaging function is shown in equation 3.3, with n denoting the number of samples in the dataset.

$$s(i) = a(i) - b(i) \tag{3.2}$$

$$SilhouetteScore = \frac{1}{n} \cdot \sum s(i) \tag{3.3}$$

We cluster the BP labels to 1000 clusters using TF-IDF and bioBERT embeddings and hierarchical and K-means clustering. The quality of the clusters is thus assessed using Silhouette scores. The results of the assessment is shown in Table 3.1. In general, a Silhouette score ¿0.5 indicates good clusters, therefore the bioBERT vectorised GO Labels that is Clustered using K-Means Clustering is the only Cluster that is good

	Hierarchical Clustering	K-Means Clusterng
TF-IDF	0.41844	0.40723
bioBERT	0.47426	0.50429

Table 3.1: Table showing the Silhouette scores of various Clusters

The Cluster that performs the best is used to reduce the number of unique labels present in the dataset. Each GO term in the dataset is mapped to its respective cluster, so the protein sequences are labelled according to the cluster that their BP labels belongs to instead of the BP Labels themselves.

3.2 Data Preprocessing

In order to feed the protein sequence data into various models, the sequences must be transformed into a machine-readable format. We transform it into a machine readable embedding by transforming each amino acid into a vector. Each vector has 21 elements. The position of each element corresponds to a specific amino acid. Each amino acid ($eg. MRWQEMGYIFYPRKLR$) must first be tokenised before it can be vectorised. Tokenisation in terms of proteins is simply breaking down sequences into their amino acid constituents as shown in equation 3.4 below.

$$aminoacid = [M, R, W, Q, E, M, G, Y, I, F, Y, P, R, K, L, R] \tag{3.4}$$

Each amino acid is then transformed into a vector with 20 zero elements and a single nonzero 1 element. The position of the 1 element determines the amino acid encoded for by the vector. Equation 3.5 shows which amino acid is represented based on the location the non-zero element. An example is shown in equation 3.6 where the amino acid Cysteine, abbrieviated C, is represented by a 1×21 vector.

$$[' A',' R',' N',' D',' C',' Q',' E',' G',' H',' I',' L',' K',' M',' F',' P',' S',' T',' W',' Y',' V'] \tag{3.5}$$

$$C = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,] \tag{3.6}$$

Each protein sequence can thus be represented by a 2 dimensional tensor with shape (1, n, 21) where n is the length of the protein sequence. Many models cannot accomodate tensors of varying shapes, thus the protein tensors are all padded so every tensor has the same shape as the largest tensor. This will transform protein sequences into machine readable tensors that can thus be fed into various machine learning models. The BP Labels must similarly be transformed into a machine readable format.

The BP Labels are transformed into cluster numbers to reduce the amount of labels that only occur once. This means that Protein sequences can be labelled according to the various clusters to which their labels belong[1]. This can also reduce the number of labels each sequence has if it has two or more BP that are in the same cluster. These clusters, although in numeric format, cannot be fed into a model as is. The clusters are transformed into one hot encodings to ensure the vectorised labels are all the same size.
Consider a table, where the column headings are all the possible BP labels in the dataset. If a sequence has a specific BP Label, then the corresponding column with that BP Label and the row of the sequence will have a 1. If there is no 1 in the column then it is automatically 0. This is better illustrated in the example 3.2 where we have 3 sequences and the dataset consists of 5 GO labels.

The Cluster labels for sequence 1 according to table 3.2 are Cluster2 and Cluster4. The one-hot encoding representation of these labels will thus be $[0, 1, 0, 1, 0]$. We apply the same One-hot encoding method to our data, with more Columns and rows to represent more Cluster labels and more protein sequences. The one-hot encoding allows the vectorised set of labels to all the same shape, regardless of the number of labels in each set.

[1]Example: Rather than having labels GO1, GO2, GO4, the sequence will have labels 3, 2 3 where GO1, GO3 are in cluster 3 and GO2 is in cluster 4

Sequences	Cluster1	Cluster2	Cluster3	Cluster4	Cluster5
seq 1	0	1	0	1	0
seq 2	0	0	1	0	1
seq 3	0	0	0	0	1

Table 3.2: Gene Ontology Table

3.3 Experiment Pipeline

3.3.1 Data Preparation

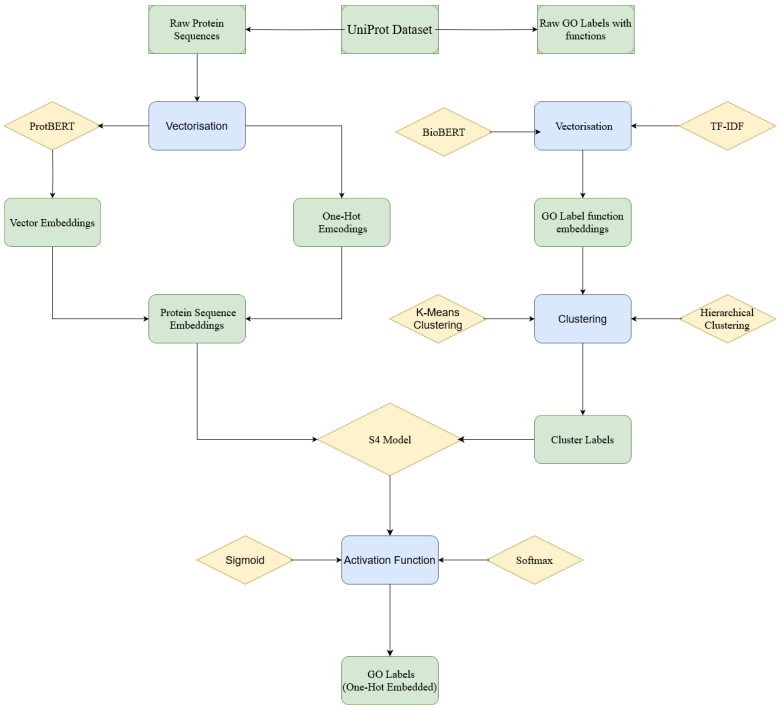


Figure 3.3: Data-Preprocessing Pipeline

The data pipeline is shown in Figure 3.3 illustrates the methods and processes applied to the UniProt Dataset. Blue blocks reflect the methods applied to the data. The yellow diamonds reflect the models utilised in the correlated methods. The Green Blocks represent the processed data. The final process data consists of the protein sequences transformed into machine readable vectors that both represent the protein sequences and can be fed into models, as well as the GO labels that have been similarly translated to numeric vectors. This data is then fed into the Structured State Space Model (S4) for training, validition and testing. The output of the S4 Model outputs probabilities that can vary from negative to positive values, however the task it to predict GO labels that is either 0 or 1. The output of the model is thus fed into an activation function that transforms the original output value to a value between 0 and 1.

3.3.2 Model HyperParameters

The Structured State Space model is trained like any Deep Learning model using the Adams[2] Optimizer algorithm to update weights in the model. The Adam Optimiser, introduced in 2015, is a combination of the Adaptive Gradient Algorithm and Root Mean Square Propagation. The Adaptive Gradient Algorithm is shown

[2]adaptive moment estimation

in equation 3.7 below where w_t represents the weights and m_t the exponentially weighted average of gradients. Equation 3.8 illustrates how this algorithm can be used to optimise weights (w_t) and the exponential moving average v_t. The hyperparameters defined in the equations below is ϵ is a small positive constant uses to avoid errors from division by 0, α the learning rate and β_1, β_2 is the rate of decay of average gradient.

$$\begin{cases} m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\partial L}{\partial w_t}\right] \\ w_{t+1} = w_t - \alpha m_t \end{cases} \tag{3.7}$$

$$\begin{cases} v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2 \\ w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\epsilon)^{\frac{1}{2}}}\left[\frac{\partial L}{\partial w_t}\right] \end{cases} \tag{3.8}$$

The Adams optimiser is thus shown below in equation 3.9 where m_t, v_t are the average gradient variations that is 'bias corrected'. This optimiser is an extension of Stochastic Gradient often used to train more complex neural networks due to its ability to adapt learning rates for individual model parameters. Adams is also capable of handling noisy gradients and non-stationary objectives making it the ideal optimiser recommended to train S4.

$$\begin{cases} w_{t+1} = w_t - \hat{m}_t(\frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon}) \\ \hat{m}_t = \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t = \frac{v_t}{1-\beta_2^t} \end{cases} \tag{3.9}$$

The optimiser finetunes weights based on the the differences between the labels predicted by a model and the ground truth labels. The difference is computed using a loss function. Two different functions are used in S4 mainly Standard and weighted Cross Entropy Loss.

Cross-entropy loss is a loss function widely used for classification tasks that quantifies the differences between the predicted probabilities and actual target labels. Entropy quantifies uncertainties associated with a random variable. It is a fundamental concept in Information theory, a subfield of mathematics concerned with signal transmission that is beyond the scope of this report. Cross entropy loss is defined as shown in equation 3.10. where t_i is the truth label, p_i is the Softmax Probability of class i and n is the number of classes.

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i) \tag{3.10}$$

Weighted cross-entropy loss is an extension of cross-entropy that assigns different weights to different classes. It assigns more importance to underrepresented or crucial classes. This modification penalizes the model more for misclassifying important classes, making it especially useful when dealing with imbalanced datasets. Weighted Cross entropy is defined the same as normal Cross entropy, with the exception of the Weight factor (w_i). This is shown in equation 3.11 below.

$$L_{WCE} = -\sum_{i=1}^{n} t_i w_i log(p_i) \tag{3.11}$$

The code used to build and train an S4 model is forked from the Hazy Research Github Repo here. The modified version used to train S4 is here along with the trained model that can be tested.

3.3.3 Metrics

Metrics are the measures used to evaluate the performance of models. It can provide insights into how models perform in terms of correctly predicted labels (True Positive and False Negatives) as well as incorrect labels (True negatives and False Positives) The choice of metrics often depends on the nature of the task and data. Metrics are crucial in assessing the quality of predictions and will guide hyperparameter tuning. Three common

metrics used in this classification task is Accuracy, F1-Scores and Area under Precision Recall Curve (AuPRC). Accuracy is defined as shown below in equation 3.12. Accuracy although often used, is not robust to class imbalance, thus we will consider other metrics.

$$Accuracy = \frac{TP + FN}{TP + FN + FP + TN} \tag{3.12}$$

Both F1-scores and AuPRC are defined in tersm of Precision and Recall. The Precision score which quantifies the accuracy of positive predictions is calculated via Equation 3.13. Recall is defined as shown in equation 3.14, measures how well a model can identify all relevant instances of a positive class. It answers the question, "Of all the actual positive instances, how many did the model correctly predict?". The F1 score helps us evaluate a model's precision in identifying classes as well as its robustness in ensuring minimal missed instances. The F1 score is defined as shown in equation 3.15

$$PS = \frac{TP}{TP + FP} \tag{3.13}$$

$$RS = \frac{TP}{TP + FN} \tag{3.14}$$

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{3.15}$$

The Area Under the Precision-Recall Curve (AuPRC) is another metric that uses Precision and Recall to evalute model performance. This metric measures the area under the curve created by plotting precision against recall at various decision thresholds and is often robust to class imbalances. AuPRC quantifies both the ability of the model to correctly classify positive instances (precision) and measures its capability to identify all positive instances (Recall). A higher AUC-PR score indicates better model performance, with values closer to 1.0 being ideal. AUC-PR is extremely useful when dealing with imbalanced datasets with sparce labels with many negative instances, as will occur in our case. Due to the class imbalance and sparse label vectors, F1-scores and AuPRC metrics are ideal to evaulate the performance of S4 when annotating proteins.

Chapter 4

Results of S4

The performance of S4 is described according to its training and validation losses as well as F1-Scores and AuPRC metrics during testing. The data from the various runs in this experiment is logged on Weights and Biases. The Steps in the x-axis reflect the steps taken by the experiment overall, hence the inconsistencies in the x-axis in the testing metrics as well as the gaps in the validation losses. The loss and metrics across all runs is averaged and smoothed so the models performance is generalised and interpretable. The values highlighted in this section references the smoothed and averaged values unless otherwise stated. This section thus presents the averaged and smoothed figures and values of the model's performance, however, the raw data from each run for each model can be found in appendix A. The lightly shaded area represents the minimum and maximum values that occurred during the various runs.

4.1 Binary S4: One-Hot embedded Proteins

The Binary S4 model is trained on a version of the data where each of the 17232 sequences is labelled as either 1 if it is in Cluster 0 or 0 if it does not belong to Cluster 0. The model is trained using for 7583 steps and 5 epochs as shown in Figure 4.1 below reflecting stochastic loss values during the total 37915 training steps. The model is trained using a batch size of 2. The averaged training loss takes on a maximum value of 0.740 and a minimum value for 0.659. The loss is extremely variable with a maximum Standard deviation of 0.110. The lowest training loss that occurred throughout all the runs is 0.4816.

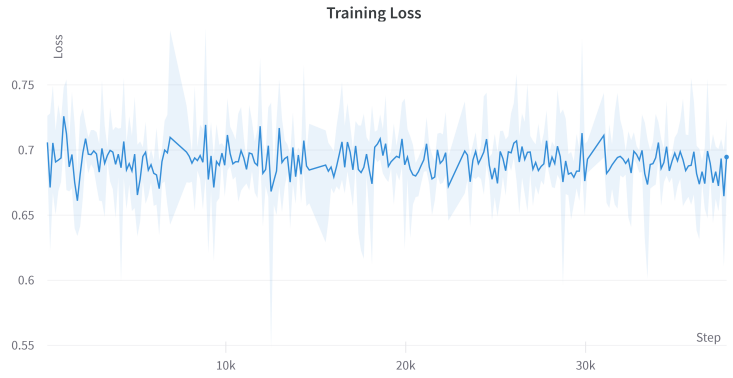


Figure 4.1: Loss during Training

The Validation Loss varies from a maximum average value of 0.704 to to 0.680 which is similar to the training loss values. The large spaces in 4.2 with straight lines illustrates the spaces where training occurs and validation does not occur. The standard deviation of the averaged loss is less than 0.001 as the steps progress indicating that the model's performance is consistent throughout the various runs.

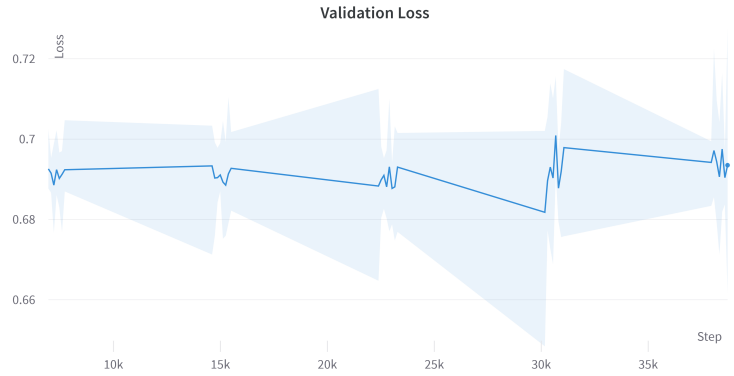


Figure 4.2: Loss during Validation

After Validation, we conduct testing. The model is tested on 6298 samples using two metrics, F1-Scores and AuPRC scores. These Metrics are shown in 4.4 and 4.3 below. The F1-Scores and AuPRC are computed each time on one sample, thus the raw values in figure 5.3 and figure 5.4 will fluctuate between 0 and 1. These metrics are continuously updated to reflect their running values which more accurately reflect the performance of the model. These running values are averaged to produce the metric graphs shown below. The F1-Score takes on a maximum value of 0.581 at step 2340 and a minimum value of 0.485 at step 1334. The F1-Score reflects how well a model identifies both True Positives and False Negatives. The AuPRC metric however, provides a more robust evaluation of the true positives identified.

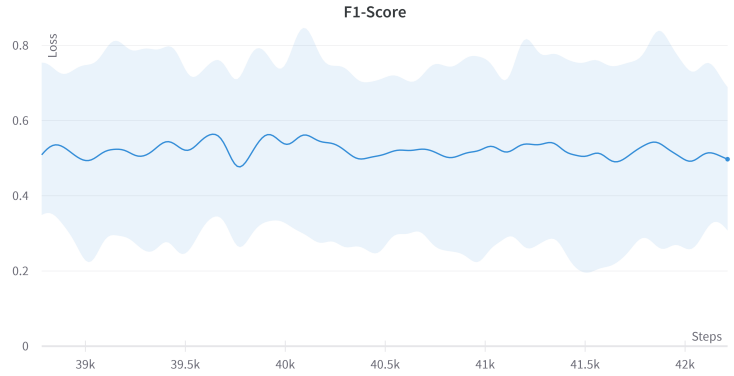


Figure 4.3: F1 during Testing

The AuPRC values, also known as the area under precision recall curve, is at a maximum at testing step 1707 with a value 0.580 and a minimum of 0.483 at step 4091. The difference between the Minimum and Maximum AuPRC at each time step remains fairly consistent indicating a low variability in the models' performance.

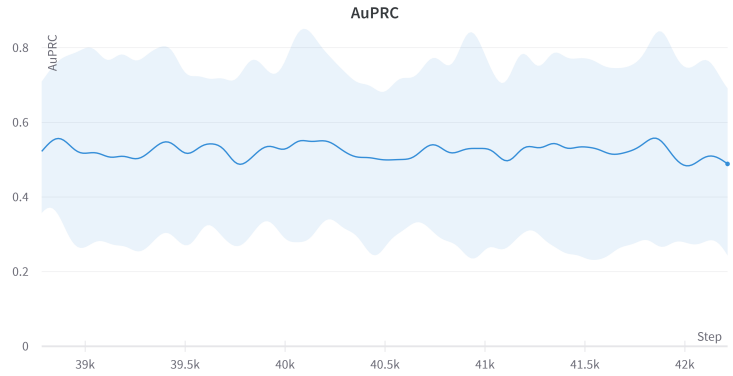


Figure 4.4: AuPRC during Testing

The metrics indicate that S4 performs adequately and consistently for the binary classification task. S4 can thus be implemented on the Multi-Label classification task.

4.2 Multi-Label S4: One-Hot embedded Proteins

The Multi-Label S4 is utilises the full dataset with 17232 sequences and the full set of one-hot encoded labels. The smoothed and averaged training loss is shown in Figure 5.5. The training loss, similar to its binary counterpart, is extremely stochastic and does not converge to a single value. S4 is trained for 1283 steps over 20 epochs with a batch size of 12 resulting in a total 25663 training steps. The spiky nature of both the smoothed graph and raw values shown in the shaded region behind it indicate that the loss tends to dramatically change at random steps but will return to a value between 4.5 and 6 afterwards. The standard deviation of the training loss stays around 1.5 unless a spike occurs, in which case, the standard deviation can jump to values close to 3.

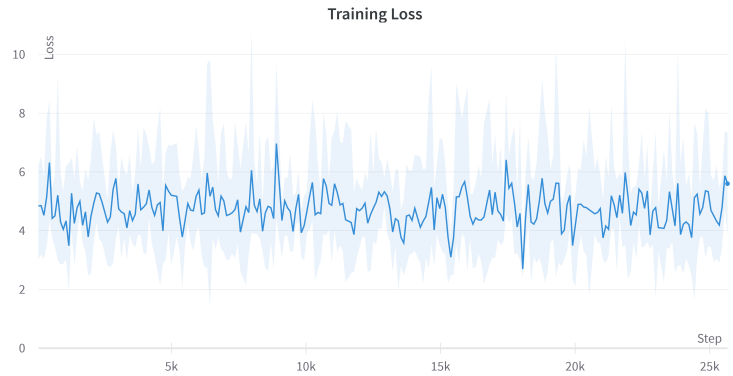


Figure 4.5: Loss during Training

The Validation loss shown below in figure 4.6, similar to training loss, has values between 4 and 6. The smoothed and averaged validation loss has a maximum value of 6.041 and a minimum value of 3614. The Standard deviation of validation loss between the various runs fluctuates around 0.6 as the model trains, however the standard deviation spikes to 1.641 when the averaged validation loss is at its' max. Consistent validation losses indicate that the model maintains stable performance even when given various inputs. This performance across varying inputs, however, should be assessed using various evaluation metrics and not just the loss values.

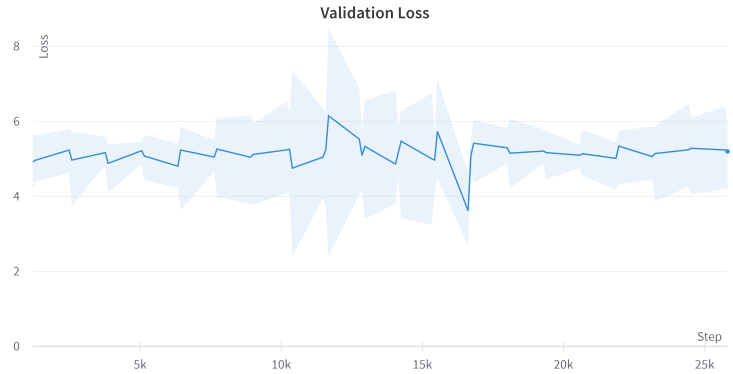


Figure 4.6: Loss during Training

The metrics used to evaluate S4 for Binary Classification can similarly be used to evaluate the performance of S4 on the Multi-Label task. Multi-Label S4 is evaluated on 3431 samples which results in an average F1 score of 0.616 and an average aAuPRC of 0.002. Figure 4.7 shows the smoothed and averaged F1 scores which reflect a maximum score of 0.654 at sample 689 and a minimum score of 0.633 at sample 3348. The maximum and minimum F1-scores that the various runs take on all lie between 0.586 and 0.731 which is a range of 0.145. This small range indicates that the F1 Scores are fairly consistent. We can further analyse the performance of Multi-Label S4 by analysing the AuPRC values.

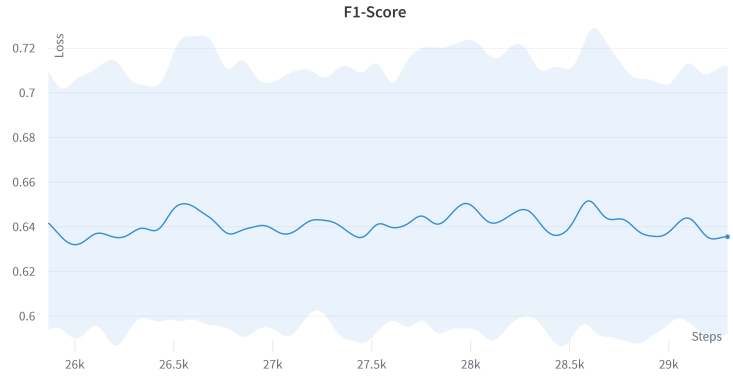


Figure 4.7: F1 during Testing

The AuPRC values of Multi-Label S4 differ from the Binary S4 AuPRC values. The smoothed and averaged AuPRC curve is shown in Figure 4.8 where its maximum value of 0.0128 occurs at the end of testing and the minimum AuPRC value of 0.009 occurs at sample 3169. The maximum AuPRC value that occurs across all the runs is 0.0151 whereas the minimum AuPRC value is 0.00749 which indicates that the range of AuPRC values produced by the runs is 0.00761, which is extremely small. This indicates that the in terms of predicting true-positives, the model is fairly consistent regardless of the input.

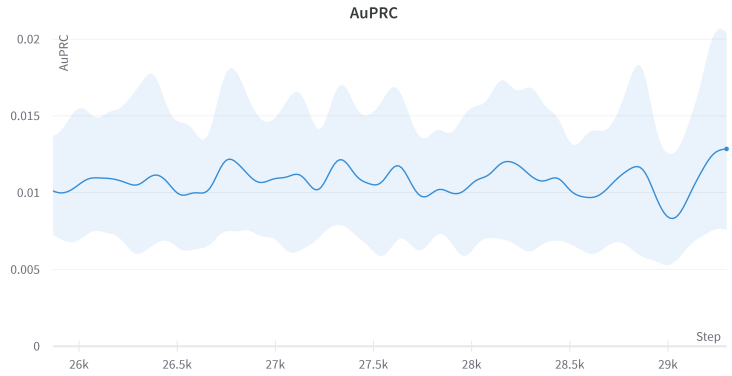


Figure 4.8: AuPRC during Testing

This section has summarised the performance of Binary and Multi-Label S4 by utilising figures and highlighting important loss and metric values. No patterns in the data are highlighted due to the lack of patterns in the metrics and the data. The data can now be analysed and interpreted in the context of its task not just as a classifier but as a model used to model proteins.

Chapter 5

Discussion

S4 is a Sequence model introduced by Gu et al (2021) that illustrates SOTA performance on tasks related to sequences with long-range dependencies, such as Proteins. This report applies S4 to the protein annotation task where the model must classify proteins to various GO labels based on their amino acid chain. Since many of the GO labels are unique, we cluster the labels according to their associated functions and label the proteins according to their cluster instead of their label which reduces the prevalence of unique labels and bias in our dataset. We constructed S4 firstly as a binary classifier where it was tasked to identify whether a protein belonged in a cluster or not. S4 was then extended to a multi-label classifier, where it must identify which of 500 clusters the protein can belong to. The protein can also belong to more than one cluster which adds a degree of complexity to the problem. The performance of Binary and Multi-Label S4 is evaluated using the performance metrics F1-Scores and AuPRC. These values are shown in table 5.1 below along with the metrics of some other SOTA methods. The best F1-Score and AuPRCs across all the models in the table are emphasised for interpretability.

	Binary-S4 (One-Hot)	**S4 (One-Hot)**	**ATGO+**	**TEMPROT+**
F1-Score	0.581	**0.654**	0.589	0.581
AuPRC	**0.580**	0.0128	0.550	0.529

Table 5.1: Table showing the Metrics of Binary and Multi-Label S4 using different protein sequence embeddings

5.0.1 Binary S4 vs Multi-Label S4

Binary and MultiLabel S4 both exhibit similar loss curves for training and validation. The loss figures in Chapter 4 and Appendix A illustrate a stochastic loss function that does not change as time (steps) progress. This indicates that the model is not learning any underlying patterns or structure in the protein sequence data. Protein Sequences are extremely complex, varying in length, structure and properties. Given the clustering and data preprocessing it is possible that the underlying structures in the data was lost. The loss values for Binary S4 is ¡1 which indicates good performance, however the loss values for multi-label S4 is between 4 and 6 which indicates that it is not performing well and failing to produce outputs similar to desired outputs. The different in loss indicates that Binary S4 is better at producing the desired output compared to Multi-Label S4. This is reflected in the different performance metric values.

The F1-Score for Binary and Multi-Label S4 is similar with values 0.581 and 0.654 respectively. Multi-Label S4 has a better F1 Score than Binary S4 , however ths does not indicate that it is performing better. F1-scores evaluate how many False Negatives (FN) and True Positives (TP) the model produces. Since Binary Classifiers cannot have both FN and TPs in the same sample, its F1 Score will be lower. The AuPRC of the Binary and Multi-Label S4, unlike the F1 scores, are extremely different. The AuPRC score values of Binary S4 and MultiLabel S4 is 0.580 and of 0.0128 respectively which have a difference of 0.567. The AuPRC of Binary S4 is

¿0.5 which indicates good performance whereas the AuPRC value of MultiLabel S4 is near 0, which indicates extremely poor performance. The difference in the two AuPRC values as well as the similarities in the F1-scores can be attributed to the vectorisation of the labels.

The AuPRC metric measures how many true positives are identified. F1-Scores measures the amount of both true positives and false negatives. S4 is trained on one hot labels of the clusters associated with each protein. Over 50% of the protein sequences in our dataset belong to only 1 of the 500 clusters which means the output labels are extremely sparse with many 0s and extremely few 1s. The F1-score¿0.5 for the MultiLabel S4 indicates that is is correctly identifying many clusters to which the protein doesn't belong which is reflected by many 0s. Multi-Label S4 is, however, not reflecting how many clusters the protein sequence does belong to with very few 1s in the output. F1-Scores, like any other metric, can be misleading if not considered alongside other factors and metrics. If Multi-Label S4 and Binary S4 was evaluated on F1-Scores alone, Multi-Label S4 would be the better performing model, however, considering the context of the AuPRC values as well as the form of the outputs, Binary S4 has the better metrics and thus better performance.

5.0.2 How does S4 compare to SOTA methods?

Binary and Multi-Label S4 is compared to State of the Art (SOTA) models in Table 5.1. Binary S4 produces a better AuPRC value compated to other the SOTA models, however, the other models are trained and evaluated as a multi-label classifier which is a more complex task. A direct comparison between the S4 models in this report and current SOTA methods may not be entirely fair since the SOTA models train on larger datasets for extended periods. Due to the scale of the datasets used to train and test SOTA methods, many sequences with unique labels are removed. This report takes a novel approach to preprocessing the labels by clustering instead of removing them. To the best of our knowledge, no existing models have been trained using clustered labels, which makes it difficult for us to evaluate the performance of S4. Given the size of our dataset and time taken to run S4, it is possible that training S4 on larger datasets, removing sequences with unique labels and longer computation times may improve S4 and allow it to match SOTA model performance.

5.0.3 Challenges and Future Work

ProtBERT is a model than can create non-sparse feature emebeddings of protein sequences, however it is not utilised in to vectorise proteins in this report. This is due primarily to memory restrictions since loading the ProtBERT model and converting 17232 sequences into vectors using this model requires more compute than is available. ProtBERT and other similar models such as ProteinBERT should be investigated in future works to evaulate whether using model-based embeddings as inputs improves the performance of S4. This, as well as experimenting with different sequence models could be investigated in future to improve the quality and state of the protein annotation problem.

S4 is a new sequence model that we apply to Protein Annotation due to its' quicker runtime and accuracy compared to Transformers which are SOTA methods in Protein annotation. One of the challenges of protein annotation is the large amount of data that is available. h Currently, we apply S4 to a subset of all available annotated protein sequence data since the full UniProt Dataset is extremely large and the amount of computational power required to use it is not available. Future experiments that have sufficient computational power can trained S4 on larger datasets which can better illustrate the performance of S4 compared to the SOTA Transformer-based models. S4, in this report, focuses primarily on predicting Biological Process labels, however, it can be extended to predict Cell component and molecular function labels using the same or multiple S4 models.

This report serves as a starting point from which many experiments can be done applying variations of S4 to Protein Sequences. As part of preprocessing, the BP labels were clustered to reduce the biasness of the dataset. This clustering method has not previously been explored and should be investigated further.

This report evaluates the performance of S4 compared to other SOTA based methods at Protein Annotation. Although S4 does not perform better than SOTA models, it is the first to be trained, validated and tested on sequences with BP labels that only occur once. This report thus presents the only model (to our knowledge) that can potentially label proteins with unique labels that only occurred once.

Appendix A:Glossary

Activation function- decides whether a neuron should be activated or not using a weighted sum and further adding bias to it.
Amino acid residues - individual amino acid units that are linked together through peptide bonds in a protein chain.
Classifiers- an algorithm that automatically sorts data
Depth-first searches- Algorithm used when searching tree or graph data structures
Directed Acyclic Graph - Type of graph with nodes that are directionally related to each other and don't form a directional closed loop
Disordered - do not have a specific shape, may be intrinsically disordered
Domain - a stable structural unit
Embedding - a learned representation of objects or features in a lower-dimensional space. It's a way of converting categorical data into continuous, lower-dimensional vectors, which can be more easily processed by machine learning algorithms.
Family - a protein region
Fence (statistics) -boundary used to identify outliers in data
K-mers- Substrings of length k that are extracted from a longer sequence, typically a DNA, RNA, or protein sequence.
Loss Function-A method to evaluate how well an algorithm models your dataset
Motifs - short unit outside globular protein domains
One-hot encodings- A binary vector representation of categorical data
Repeat - a short unit which may be unstable in isolation but forms a stable structure when multiple copies are present
semantics - the meaning of a word, phrase, or text.

Appendix B:Non-Averaged Runs

5.1 Binary S4 runs

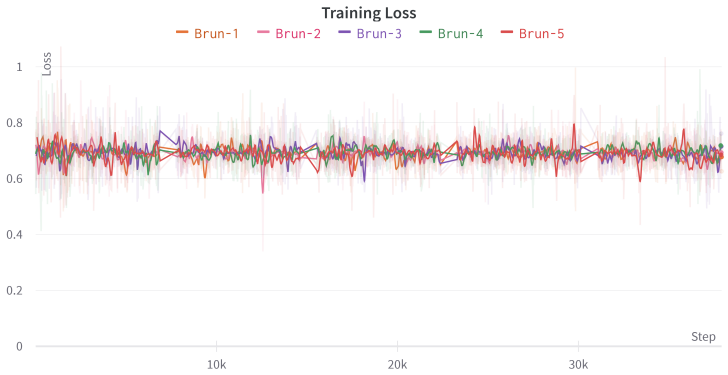


Figure 5.1: Raw Loss values during Training

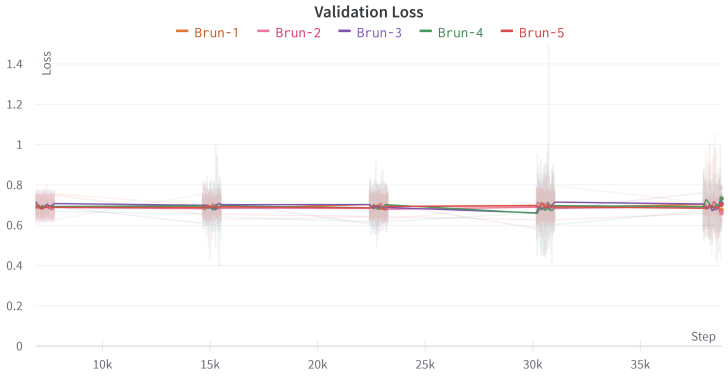


Figure 5.2: Raw Loss values during Validation

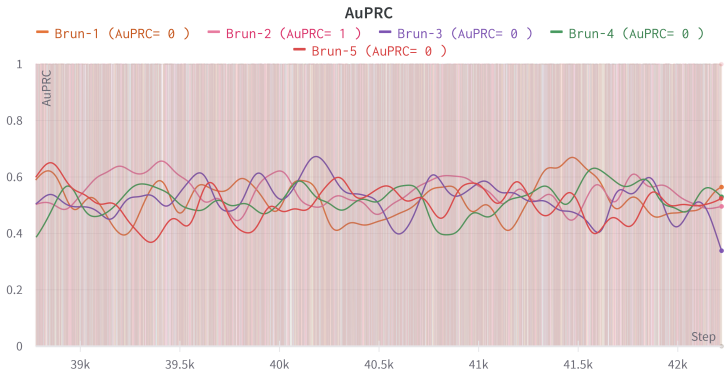


Figure 5.3: Raw AuPRC scores during Testing

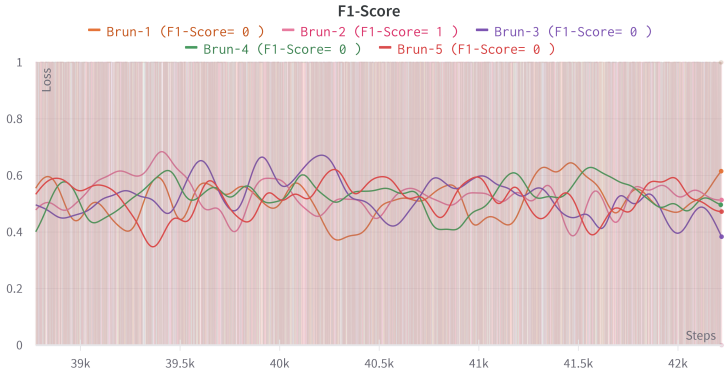


Figure 5.4: Raw F1 Scores during Testing

5.2 Multi-Label S4

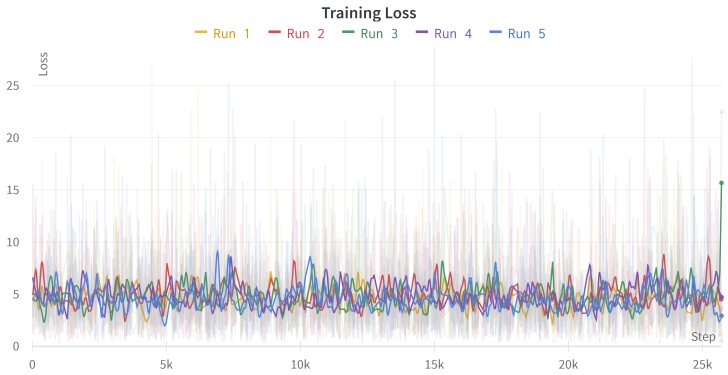


Figure 5.5: Raw Loss values during Training

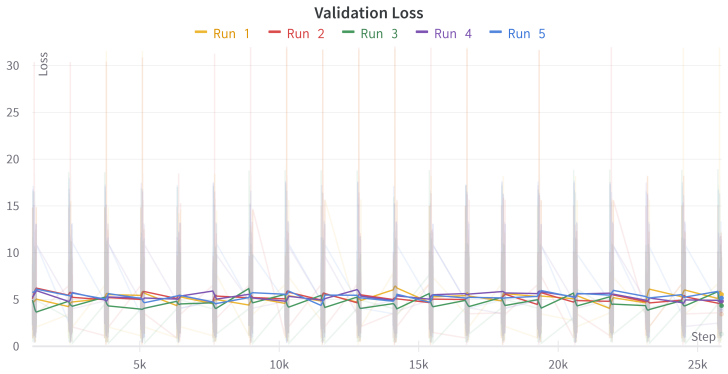


Figure 5.6: Raw Loss values during Validation

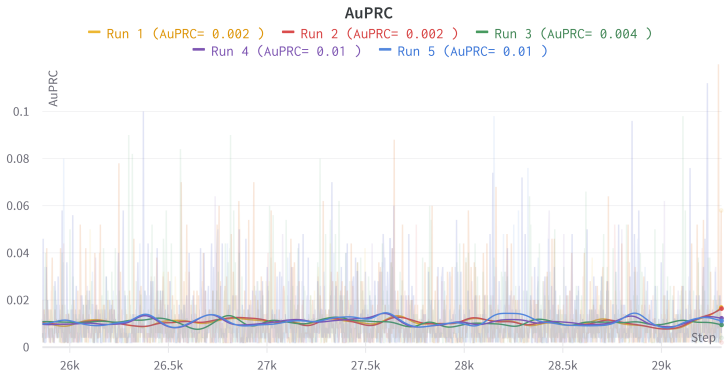


Figure 5.7: Raw AuPRC values during Testing

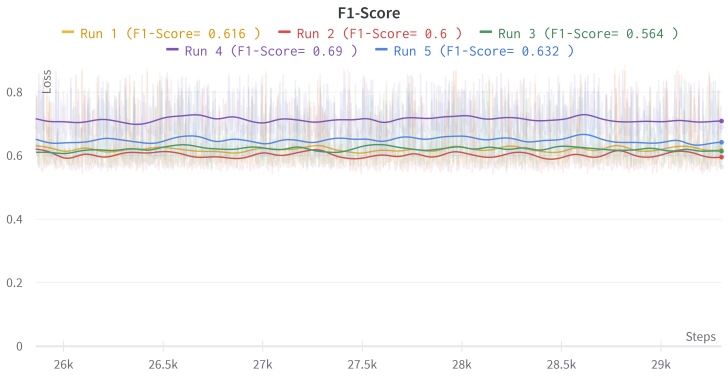


Figure 5.8: Raw F1-Scores during Testing

Bibliography

[1] Lopez MJ, Mohiuddin SS. Biochemistry, Essential Amino Acids. [Updated 2023 Mar 13]. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2023 Jan-. Available from: https://www.ncbi.nlm.nih.gov/books/NBK557845/

[2] Book: General biology (boundless) (2022) Biology LibreTexts. Available at: https://bio.libretexts.org/Bookshelves/Introductory_and_General_Biology/Book%3A_General_Biology_(Boundless) (Accessed: 24 September 2023).

[3] , L.T. et al. (2019) 'The role of protein complexes in human genetic disease', Protein Science, 28(8), pp. 1400–1411. doi:10.1002/pro.3667.

[4] Clark, M.A., Choi, J.H. and Douglas, M.M. (2020) Biology. Houston, TX: OpenStax, Rice University.4. Clark, M.A., Choi, J.H. and Douglas, M.M. (2020) Biology. Houston, TX: OpenStax, Rice University

[5] Mukherjee, S. (2023) Transport proteins: Definition, types, functions, examples, Science Facts. Available at: https://www.sciencefacts.net/transport-proteins.html (Accessed: 24 September 2023).

[6] Levy, J., Titus, A., Petersen, C., Chen, Y., Salas, L. Christensen, B. 2020. MethylNet: an automated and modular deep learning approach for DNA methylation analysis. BMC Bioinformatics. 21(1). DOI: 10.1186/s12859-020-3443-8

[7] Kimball, J.W. (1994) Biology. Dubuque, IA: Wm. C. Brown Publishers.

[8] Cooper GM. The Cell: A Molecular Approach. 2nd edition. Sunderland (MA): Sinauer Associates; 2000. Microtubule Motors and Movements. Available from: https://www.ncbi.nlm.nih.gov/books/NBK9833/

[9] Hombalkar, S. (2023) What are the two rare amino acids?, LabXchange. Available at: https://www.labxchange.org/library/items/lb:LabXchange:6b721849:html:1 (Accessed: 24 September 2023).

[10] Whisstock, J., & Lesk, A. (2003). Prediction of protein function from protein sequence and structure. *Quarterly Reviews of Biophysics, 36*(3), 307-340. doi:10.1017/S0033583503003901

[11] Wu, C.H. et al. (2003) 'Protein family classification and functional annotation', Computational Biology and Chemistry, 27(1), pp. 37–47. doi:10.1016/s1476-9271(02)00098-1.

[12] Mistry, J. et al. (2020) 'Pfam: The Protein Families Database in 2021', Nucleic Acids Research, 49(D1). doi:10.1093/nar/gkaa913.

[13] Gene Ontology Consortium, The Gene Ontology (GO) database and informatics resource, Nucleic Acids Research, Volume 32, Issue suppl_1, 1 January 2004, Pages D258–D261, https://doi.org/10.1093/nar/gkh036

[14] The UniProt Consortium UniProt: the Universal Protein Knowledgebase in 2023 Nucleic Acids Res. 51:D523–D531 (2023)

[15] Séverine Duvaud, Chiara Gabella, Frédérique Lisacek, Heinz Stockinger, Vassilios Ioannidis, Christine Durinx; Expasy, the Swiss Bioinformatics Resource Portal, as designed by its users Nucleic Acids Research, 2021. DOI: 10.1093/nar/gks225

[16] Zhou, N., Jiang, Y., Bergquist, T.R. et al. The CAFA challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens. Genome Biol 20, 244 (2019). https://doi.org/10.1186/s13059-019-1835-8

[17] Ofer, D., Brandes, N. and Linial, M. (2021) 'The language of proteins: NLP, Machine Learning Protein Sequences', Computational and Structural Biotechnology Journal, 19, pp. 1750–1758. doi:10.1016/j.csbj.2021.03.022.

[18] Kosar, V. (2022) Tokenization in Machine Learning explained, Vaclav Kosar's face photo. Available at: https://vaclavkosar.com/ml/Tokenization-in-Machine-Learning-Explained (Accessed: 24 September 2023).

[19] Chowdhary, K.R. (2020) 'Natural language processing', Fundamentals of Artificial Intelligence, pp. 603–649. doi:10.1007/978-81-322-3972-7_19.

[20] Jha, A. (2023) Vectorization techniques in NLP [guide], neptune.ai. Available at: https://neptune.ai/blog/vectorization-techniques-in-nlp-guide (Accessed: 24 September 2023).

[21] Prabhu (2019) Understanding NLP word embeddings-text vectorization, Medium. Available at: https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223 (Accessed: 24 September 2023).

[22] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So and Jaewoo Kang, BioBERT: a pre-trained biomedical language representation model for biomedical text mining Bioinformatics (2020), 36(4), 1234–1240. doi: 10.1093/bioinformatics/btz682.

[23] Raghudeep (2019) Review: BioBERT paper, Medium. Available at: https://medium.com/@raghudeep/biobert-insights-b4c66fde8fa7 (Accessed: 24 September 2023).

[24] Seif, G. (2022) *The 5 clustering algorithms data scientists need to know*, *Medium*. Available at: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68 (Accessed: 25 September 2023).

[25] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp.2825-2830, 2011.

[26] Gupta, Y. (2022) *Understanding hierarchies using dendrograms*, *Medium*. Available at: https://medium.com/dssimplified/understanding-hierarchies-using-dendrograms-e3aef7ac5ea4 (Accessed: 25 September 2023).

[27] Saini, A. (2023) *Guide on Support VectorMachine (SVM) algorithm*, *Analytics Vidhya*. Available at:https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/ (Accessed: 25 September 2023).

[28] Peixeiro, M. (2020) *The Complete Guide to support vector machine (SVM)*, *Medium*. Available at: https://towardsdatascience.com/the-complete-guide-to-support-vector-machine-svm-f1a820d8af0b (Accessed: 25 September 2023).

[29] Svozil, D., Kvasnicka, V. \ Pospichal, J. 1997. Introduction to multi-layer feed-forward neural networks. Chemometrics and Intelligent Laboratory Systems. 39(1):43-62. DOI:10.1016/s0169-7439(97)00061-0.

[30] Goodfellow, I., Bengio, Y. \ Courville, A. 2016. Deep learning. 1st ed. Cambridge (EE. UU.): MIT Press.

[31] Sarker, I. 2021. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN Computer Science. 2(6). DOI: 10.1007/s42979-021-00815-1.

[32] Khan, A. *et al.* (2020) 'A survey of the recent architectures of deep convolutional Neural Networks', *Artificial Intelligence Review*, 53(8), pp. 5455–5516. doi:10.1007/s10462-020-09825-6.

[33] *What are convolutional neural networks?* (no date) *IBM*. Available at: https://www.ibm.com/topics/convolutional-neural-networks (Accessed: 25 September 2023).

[34] Yamashita, R. *et al.* (2018) 'Convolutional Neural Networks: An overview and application in Radiology', *Insights into Imaging*, 9(4), pp. 611–629. doi:10.1007/s13244-018-0639-9.

[35] Skalski, P. (2019) *Gentle dive into math behind Convolutional Neural Networks*, *Medium*. Available at:https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9 (Accessed: 25 September 2023).

[36] Wang, Z.J. *et al.* (2021) 'CNN explainer: Learning convolutional neural networks with interactive visualization', *IEEE Transactions on Visualization and Computer Graphics*, 27(2), pp. 1396–1406. doi:10.1109/tvcg.2020.3030418.

[37] Haykin, S. 2009. Neural Networks and Learning Machines. 3rd ed. New Jersey: Pearsons.

[38] Shenfield, A. and Howarth, M. (2020) 'A novel deep learning model for the detection and identification of rolling element-bearing faults', *Sensors*, 20(18), p. 5112. doi:10.3390/s20185112.

[39] Khuong, B. (2020) *The basics of recurrent neural networks (rnns)*, *Medium*. Available at: https://pub.towardsai.net/whirlwind-tour-of-rnns-a11effb7808f (Accessed: 25 September 2023).

[40] Salehinejad, Hojjat & Sankar, Sharan & Barfett, Joseph & Colak, Errol & Valaee, Shahrokh. (2017). Recent Advances in Recurrent Neural Networks.

[41] Turkoglu, Mehmet Ozgur & D'Aronco, Stefano & Wegner, Jan. (2021). Gating Revisited: Deep Multi-Layer RNNs That can be Trained. IEEE Transactions on Pattern Analysis and Machine Intelligence. PP. 1-1. 10.1109/TPAMI.2021.3064878.

[42] Kalita, D. (2023) *A brief overview ofrecurrent neural networks (RNN)*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/ (Accessed: 25 September 2023).

[43] Smagulova, K., James, A.P. A survey on LSTM memristive neural network architectures and applications. *Eur. Phys. J. Spec. Top.* **228**, 2313–2324 (2019). https://doi.org/10.1140/epjst/e2019-900046-x

[44] Dolphin, R. (2021) *LSTM networks: A detailed explanation*, *Medium*. Available at: https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9 (Accessed: 25 September 2023).

[45] K.L. (2023) *What is LSTM - introduction to long short term memory*, *Intellipaat Blog*. Available at: https://intellipaat.com/blog/what-is-lstm/ (Accessed: 25 September 2023).

[46] Srivastava, P. (2023) *Essentials of deep learning: Introduction to long short term memory*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/ (Accessed: 25 September 2023).

[47] Zvornicanin, W. by: E. (2023) *Differences between bidirectional and Unidirectional LSTM*, *Baeldung on Computer Science*. Available at: https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm (Accessed: 25 September 2023).

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, & Illia Polosukhin (2017). Attention Is All You Need. *CoRR, abs/1706.03762.*

[49] Phi, M. (2020) *Illustrated guide to transformers- step by step explanation*, *Medium*. Available at: https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation=f74876522bc0 (Accessed: 25 September 2023).

[50] Cheever, E. (no date) *State Space Representations of Linear Physical Systems*, *State space representations of linear physical systems*. Available at: https://lpsa.swarthmore.edu/Representations/SysRepSS.html (Accessed: 25 September 2023).

[51] Chen, Z. and Brown, E.N. (no date) *State space model*, *Scholarpedia*. Available at: http://www.scholarpedia.org/article/State_space_model (Accessed: 25 September 2023).

[52] Gu, A. *et al.* (no date) *Structured State Spaces for Sequence Modeling (S4)*, *Hazy Research*. Available at: https://hazyresearch.stanford.edu/blog/2022-01-14-s4-1 (Accessed: 25 September 2023).

[53] Albert Gu, Karan Goel, & Christopher Ré. (2022). Efficiently Modeling Long Sequences with Structured State Spaces.

[54] Gu, A., Goel, K. and Ré, C. (no date) *The annotated S4 - github pages*. Available at: https://srush.github.io/annotated-s4/ (Accessed: 25 September 2023).

[55] 'SS Introduction' (2010) *CSE571*. Washington: Washington, 21 September.

[56] Cai, C.Z. *et al.* (2003) 'Protein function classification via support Vector Machine Approach', *Mathematical Biosciences*, 185(2), pp. 111–122. doi:10.1016/s0025-5564(03)00096-8.

[57] Deen, A.J. and Gyanchandani, M. (2019) 'Machine learning kernel methods for protein function prediction', *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)* [Preprint]. doi:10.1109/icssit46314.2019.8987852.

[58] Maxat Kulmanov, Mohammed Asif Khan, Robert Hoehndorf, DeepGO: predicting protein functions from sequence and interactions using a deep ontology-aware classifier, *Bioinformatics*, Volume 34, Issue 4, February 2018, Pages 660–668, https://doi.org/10.1093/bioinformatics/btx624

[59] Maxat Kulmanov, Robert Hoehndorf, DeepGOPlus: improved protein function prediction from sequence, *Bioinformatics*, Volume 36, Issue 2, January 2020, Pages 422–429, https://doi.org/10.1093/bioinformatics/btz595

[60] Plyusnin I, Holm L, To¨ro¨nen P (2019) Novel comparison of evaluation metrics for gene ontology classifiers reveals drastic performance differences. PLoS Comput Biol 15(11): e1007419. https://doi.org/10.1371/journal.pcbi.1007419

[61] M, H. and M.N, S. (2015) 'A review on evaluation metrics for Data Classification Evaluations', *International Journal of Data Mining &; Knowledge Management Process*, 5(2), pp. 01–11. doi:10.5121/ijdkp.2015.5201.

[62] Lanners, Q. (2019) *Neural machine translation*, *Medium*. Available at: https://towardsdatascience.com/neural-machine-translation-15ecf6b0b (Accessed: 25 September 2023).

[63] Cao, R.; Freitas, C.; Chan, L.; Sun, M.; Jiang, H.; Chen, Z. ProLanGO: Protein Function Prediction Using Neural Machine Translation Based on a Recurrent Neural Network. *Molecules* **2017**, *22*, 1732. https://doi.org/10.3390/molecules22101732

[64] Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, Michal Linial, ProteinBERT: a universal deep-learning model of protein sequence and function, *Bioinformatics*, Volume 38, Issue 8, March 2022, Pages 2102–2110, https://doi.org/10.1093/bioinformatics/btac020

[65] Oliveira GB, Pedrini H, Dias Z. TEMPROT: protein function annotation using transformers embeddings and homology search. BMC Bioinformatics. 2023 Jun 8;24(1):242. doi: 10.1186/s12859-023-05375-0. PMID: 37291492; PMCID: PMC10249241.

[66] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, Burkhard Rost bioRxiv 2020.07.12.199554; doi: https://doi.org/10.1101/2020.07.12.199554