A NOVEL PREDICTIVE ANALYSER FOR DATA INTENSIVE APPLICATION

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

VIGNESH VAIDYANATHAN – 15BCE0076

VOLETI RAVI – 15BCE0082

SAMUDRA PRATIM BORKAKOTI – 15BCE0093

Under the guidance of

Prof. Vijayasherly V

SCOPE

VIT, Vellore.

April, 2019

DECLARATION

We hereby declare that the thesis entitled “A Novel Prediction Analyzer for Data Intensive Application" submitted by us, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by us under the supervision of Prof. Vijayasherly V.

We further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date :

Signature of Candidates

<u>**CERTIFICATE**</u>

This is to certify that the thesis entitled "A Novel Prediction Analyzer for Data Intensive Application" submitted by **Vignesh Vaidyanathan (15BCE0076), Voleti Ravi (15BCE0082), Samudra Pratim Borkakoti (15BCE0093)**, **School of Computer Science and Engineering**, VIT University, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by them under my supervision during the period, 01. 12. 2018 to 30.04.2019, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date :

Signature of the Guide

Internal Examiner

External Examiner

Head of the Department

B. Tech Computer Science and Engineering

ACKNOWLEDGEMENTS

We would like to express our utmost gratitude to our guide **Prof. Vijayasherly V** for letting us work on this project while providing her continuous guidance, support, and suggestions throughout the course of the project.

We wish to express our sincere thanks to Honorable Chancellor**, Dr. G. Viswanathan**; esteemed Vice-Presidents, **Shri. Sankar Viswanathan**; **Shri. Sekar Viswanathan** and **Shri. G. V. Selvam**; respected Vice Chancellor, **Dr. Anand A. Samuel**; respected Pro-Vice Chancellor **Dr. V. Raju**; respected ProVice Chancellor **Dr. S. Narayanan** of this prestigious institution for providing us with an excellent world class academic environment and facilities for pursuing our B.Tech Program.

We would also like to take this opportunity to thank the Dean, **Prof. Saravanan R** and HOD, **Prof. Santhi V**, School of Computer Science and Engineering for providing us the opportunity to do our project work in VIT. Our sincere gratitude to the faculties and non-teaching staff for giving us support that we needed to achieve our goals.

We would also like to thank all our peers who helped us during the course of this project. This acknowledgement would be incomplete without expressing our thanks to all our family members for their continuous support and guidance in all walks of our lives.

VIGNESH VAIDYANATHAN

VOLTI RAVI

SAMUDRA PRATIM BORKAKOTI

Executive Summary

Understanding of secondary protein structure is of high importance as it is required for research of anti-bodies, the cure for constantly evolving bacteria which has become more resistant to existing medicine. Prediction of protein structures is very data intensive and computationally complex due to large set of amino acid sequences which are unique, thereby making it difficult to achieve high accuracy.

This paper has recreated a Recurrent Neural Network (RNN) model using Long Short Term Memory (LSTM) cells for this application which we feel is the right approach to the problem. During the developmental process, multiple challenges arose such as: The manipulation of the hyper-parameters had an effect on the accuracy of the prediction, Adding and removing of filters also affected the accuracy of prediction. Moreover, addition of filters at certain sections of the network model adversely affected the accuracy of prediction.

In order to solve these issues, readjustment of hyper-parameters of the convolutional filters was done which helped to converge the data more effectively. Additionally, vertical links between the LSTM cells and batch normalization technique was established which improved the accuracy compared to the existing model.

After several training and testing sessions, an accuracy of 70.4 % was obtained which is 1.7% better than the existing base paper. This is a very positive result considering that it is over a 700 amino acid sequences per protein. Other non-neutral network model such as Support Vector Machine (SVM), Linear Regression + K-means performed poorly on the same dataset. It is suspected that it is because of the long amino acid sequence. Historically, Non-Neural Network models have proven to be accurate for shorter sequences, however, it is incorrect to do so as the real world amino acid sequence is 700 or longer and this is region where further study is required.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ACKNOWLEDGEMENT	i
	EXECUTIVE SUMMARY	ii
	LIST OF FIGURES	v
	LIST OF TABLES	v
	ABBRIVATIONS	vi
	SYMBOLS AND NOTATIONS	vi
1.	**INTRODUCTION**	1
	1.1 Theoretical Background	1
	1.2 Motivation	2
	1.3 Objectives	2
2.	**LITERATURE SURVEY**	3
3.	**TECHNICAL SPECIFICATION**	7
	3.1 Hardware requirements	7
	3.2 Software Requirements	7
4.	**DESIGN APPROACH AND DETAILS**	8
	4.1 Introduction	8
	4.2 Proposed System Design	8
	4.3 Module Wise Explanation	12
	4.4 Requirement Analysis	16
	4.4.1 Functional Requirements	16
	4.4.1.1 Product Features	16
	4.4.1.2 Assumption, Dependencies & Constraints	16
	4.4.1.3 Domain Requirements	16
	4.4.1.4 User Requirements	17
	4.4.2 Nonfunctional Requirements	17

5.	**SCHEDULE, TASKS AND MILESTONES**	18
6.	**PROJECT IMPLEMETATION**	19
7.	**RESULTS AND DISCUSSION**	23
8.	**SUMMARY**	25
9.	**REFERENCES**	26
	APPENDIX – A	27

List of Figures

Figure No.	Title	Page No.
1	System Block Diagram	9
2	Neural Network Model input layer	12
3	Gantt chart of schedules and milestones	18
4	Input Layer	19
5	Concat Layer after convolution	19
6	First Dense Layer	20
7	LSTM Layers	20
8	Concat Layer after LSTM	20
9	Second Dense Layer	21
10	Dropout Layer	21
11	Output Layer	21
12	Reshaping of output	21
13	Accuracy without CNN, vertical links and batch normalization	22
14	Dimensions of Neural Network Model	22
15	Accuracy with CNN, vertical links and batch normalization	22
16	Training & Validation accuracy of RNN model	24
17	Average precision of RNN Model	24
18	Individual Contribution	27

List of Tables

Table No.	Title	Page No.
1	Comparison of Q8 accuracy between ours and others methods	23

List of Abbreviations

LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
PDB	Protein Database
MLP	Multi-Layer Perceptron
1D	One Dimensional
SVM	Support Vector Machine
ReLU	Rectifier Linear Unit

Symbols and Notations

α	Alpha helix
β	Beta sheet
θ	Weight matrix
σ	Sigmoid function
λ	Tunable hyper parameter
Θ	Elementwise multiplication
h_t	Hidden states

1. INTRODUCTION

1.1 Theoretical Background

Our knowledge of protein structure comes predominantly from X-ray diffraction patterns of crystallized proteins. Though this method is quite accurate but is very time consuming and may involve many uncertain steps. Recently there has been an increase in the number of known protein sequences. With the development of new techniques and technologies, it is possible to predict the structure of a protein from its primary sequences. Information obtained from the secondary structure of a protein can he used to determine the structural properties of the protein which can be used for various other analysis. The protein structure primarily contains a sequence of secondary structure elements, namely, α helices and β sheets, which together constitute the overall three-dimensional configuration of the protein chain. The α helix is the most abundant sort of secondary structure in proteins. The α helix has 3.6 amino acids per turn with a H bond shaped between each fourth buildup. The most widely recognized area of α helices is at the outside of protein centers, where they give an interface the fluid condition. β sheets are made by H bonds between 5– 10 back to back amino acids in a single segment of the chain with another 5– 10 more distant down the chain. It is progressively hard to foresee the area of β sheets than of α helices. The circumstance improves to some degree when the amino acid difference in various succession arrangements is considered.

The secondary structure prediction is a set of techniques which predicts the local secondary structures of protein by knowing only their amino acid sequences. The prediction generally consists of assuming certain regions of the amino acids likely to be α helices and β sheets or coils. Modern approaches for predicting secondary protein structures makes use of various machine learning methods. New deep learning methods has surfaced and has resulted in good performance. Also, people have now moved to analyzing homologous proteins which has a better and non-repetitive protein sequences. Recent architectures use feed-forward neural networks for the predictions from the forward and backward networks in the bidirectional model [1]. This paper has incorporated a variation of the newly emerging methods using neural network to train our model with Convolutional Neural Network (CNN) and bi-directional Recurrent Neural Network (RNN) with memory cells such as, Long-Short Term Memory (LSTM) cells which is will be explained in the coming sections.

With the increase in the amount of data being generated, there is a need to handle and process such big data. Given that this application is very computationally intensive and complex, it would be best suited to distribute the work involved using data parallelization.

1.2 Motivation

The need to take on a problem which was complex to solve and data intensive at the same time. Today, Data Science seems to be the next step towards technological advancement where everything is fully automated with the techniques such as machine learning, deep learning and neural networks working in their core. The need to take on an application that would require such techniques to be used. Studying of Protein structures is one of most complicated work in science and its understanding will help in many ways such as, creating anti-bodies, understanding of DNA, how mutation occurs and many more.

Prediction of the secondary protein structure has many rules and sequence alterations which needs to be followed which would mean that certain predicate rules would need to be trained into our model which would require us to incorporate the technique of Neural Networks. The other aspect of choosing this application is that, it is very data intensive. The primary sequence in the dataset is very large and also the computation involved in determining the structures is complex and would take a lot of time.

1.3 Objectives

- Establishment of a fully connected Neural Network and Convolutional Neural Network with Batch normalization and vertical links.
- Training a bi-directional Recurrent Neural Network (RNN) with memory cells (long-short term memory)
- Prediction of Secondary Structure
- Compering the results with other types of models.

2. LITERATURE SURVEY

The approach by Protein Secondary Structure Prediction with Long Short Term Memory Networks [1] uses feed-forward neural networks for concatenation of predictions from the forward and backward networks in the bidirectional model and the model also includes feed-forward neural networks between hidden states in the recurrent network.

In this Model, one trains two separate RNN's, the forward RNN starts the recursion from x1 and goes forwards, the backwards model starts at x_n and goes backwards. The results from the forward and backward networks are combined and standardized. This paper extends the standard stacked bidirectional LSTM approach by presenting a feed-forward system which links the yield from the forward and backward systems into a SoftMax prediction. Also, the model further grows by embedding a feed-forward system between hidden states.

The LSTM network has a classification rate of 0.674, which is an improvement compared to anything current cutting edge execution accomplished by a Generative Stochastic Network (GSN) and Contingent Neural Field (CNF). Besides, the LSTM arrange performs fundamentally superior to the bidirectional RNN utilized in SSpro8 having a right characterization rate of 0.511.

Future work includes investigation of different architectures for the feedforwards networks.

Artificial Intelligence in Prediction of Secondary Protein Structure Using CB513 Database [2] uses the modified API_EPE to evaluate the protein set to predict the secondary protein structure using non - homologous data set. This paper describes CB513 a non-redundant dataset, suitable for development of algorithms for prediction of secondary protein structure. There was an implementation made in Borland Delphi for manipulating information from the dataset to make it appropriate for learning of neural system for predicting of protein structure executed in MATLAB Neural-System Tool kit. Learning of neural network is studied with various sizes of windows, varying number of neurons in the hidden layer and a number of training epochs, while using dataset CB513.

The methodology is described in the following: 1. Extraction, Preparing, And Encoding of Data Samples, 2. Determination of neural network training sets batch matrices, 3. Prediction of secondary structure implementation.

The evaluation of the neural network is done by finally applying the CB513 non-homologous protein data set.

- 413 training amino acid sequences
- 100 test amino acid sequences
- design of the neural network based on 5 neurons in hidden layer with a window size of 19,
- training of designed neural network in 4000 epochs,

Accuracy of a neural network prediction with non-homologous test set is 62.7253.

Protein and Secondary Structure Prediction with Convolutions and Vertical-Bi-Directional RNNS' [3] approach to the problem rose as the challenge of identifying secondary protein structures is often predictable from structures along the sequence of the amino acid and sequence profiles (Ole cite). A recurrent neural network with memory cells, such as the long-short term memory (LSTM) cell, can inform the model about previous data in a sequence, thus utilizing the sequential structure. Further the local sequence network structure has a high importance for predicting secondary protein structures (Ole cite).
A Convolutional Neural Network (CNN) is a way to deal with parameterized filters for recognizing characterized spaces of information. Accordingly, convolutions can be incorporated by using the local structures along the grouping of amino acids and sequence profiles. Since the problem of predicting the protein structures given a sequence of amino acid and sequence profiles isn't direction-dependent and the whole sequence of amino acids is given at one time, bi-directional RNNs can be used to study the sequences of the protein structures from the two directions. The contribution of this paper is an extension to the deep learning model by combining the convolutions ability to model local typology in the input while allowing a bi-directional recurrent neural network to model long term dependencies.
The Modal for this neural network, which has 2.1 million parameters, consists of three convolutional layers with batch normalization; followed by a fully connected layer with batch normalization; bi-directional recurrent layer with long-short term memory cells where the hidden state from the forward pass is used as input in the backwards pass; then feed into a fully connected layer before the final softmax layer. The network is regularized by dropout and the L^2 normalization. Further, gradients where normalized and output prediction clipped to stabilize training.

Another approach which uses logic-based machine learning for prediction of protein secondary structure from primary structure [4] is an active research area in hierarchical approach to the protein folding problem; by examining the Brookhaven database of known protein structures; to find general rules relating primary and secondary structure. The accuracy usually achieved is 60-70%. The reason for this is, the predictions are done by utilizing just the local data, so the long-range communications are not considered. Long-range interactions are essential when a protein overlap up, parts of the sequence which are widely separated turned out to be close spatially. Other methodologies have included hand-made rules by specialists and Bayesian statistical techniques. As of late different machine learning strategies have been attempted such as neural systems and emblematic enlistment. Comparison between these methods is difficult -- different workers have used different types of proteins in their datasets. Golem is an ILP program. It takes positive examples, negative examples and background knowledge (Prolog facts) as input. It produces Prolog rules which are generalizations of the examples as output. The method of generalization is based on the logical idea of Relative Least General Generalization (RLLG).

Prediction of Protein Secondary Structure using Logistic Regression, K-Means Clustering, and Naïve Bayes Variation [12] discusses the use of machine learning applied to the prediction of secondary structure for a protein. The probabilistic capabilities of the logistic regression sigmoid was used, and then the clustering of k-means to obtain secondary structure information. This paper primarily introduces the method as a way to use commonly known and efficient algorithms to compute and determine the secondary structure of the protein given its amino acid sequence. First the amino acids were passed through the logistic regression sigmoid, then combined with the k-means clustering algorithm. Overall, the final results looked promising for less proteins with similar secondary structures. However, once variety in size, sequence, and structure was introduced, the method’s accuracy drastically decreased from 84% to about 42%.
The first step is to create features of each protein sequence. The features are the amino acids in the protein sequence, as this is the determining factor when predicting the secondary structure of a protein. (1) Logistic Regression: After getting a protein sequence, the first step is to pass each amino acid through a logistic regression function; (2) K-Means Clustering: The next algorithm used on the amino acids after the probabilities were predicted was the k-means clustering algorithm. Overall, this method seemed to have work intuitively and in theory.

However, due to the choice of clustering methods, as well as the choice of computation with the amino acids, the implementation resulted in a not so good method. The method proposed using logistic regression combined with k-means clustering is a good method, but can still be refined further to push the accuracy up and possibly be used in further bioinformatics studies.

Challenging Issues Identified and addressed

- The manipulation of the hyper-parameters involved seems to have an effect in the accuracy of the prediction.
    - Therefore, it is necessary to try out different parameters in order to find the best fit for highest possible accuracy.

- Adding and removing of filters also seems to have an effect in the accuracy of the prediction. With the addition of certain filters, the accuracy seems to drop. Therefore, it is suspected that the filters are removing some necessary data along with the unnecessary ones.
    - This can be solved by analyzing the effects of each filter size on the physical data and manipulating them accordingly.

3. TECHNICAL SPECIFICATION

The Dataset used was: Princeton's ICML 2014 CB513 + CullPDB.

3.1 Hardware Requirements

Minimum computer requirements: Any multi-core CPU, GPU – GTX 860M

3.2 Software Requirements

UBUNTU 18.04 LTS

IDE: PyCharm

NVIDIA CUDA compiler driver Version 10.0.130

GPU drivers: Version 410.79

Python Library Packages:

Machine learning libraries: Lasagne, Theano, batch normalization

Mathematical calculation: NumPy, TensorFlow,

File Handling: Pickle, gedit.

4. DESIGN APPROACH AND DETAILS

4.1 Introduction

Looking at the number of protein sequences in UniProt to the number of known structures in the Protein Data Base (PDB), there are around 400 times more sequences than there are structures of the same amino acid. Discovering alternative approaches to predict protein structures turns out to be increasingly more imperative as this issue increases. Given the rate at which new amino acid sequences are being found, it is getting computationally very hard to analyze structures for all the new proteins that are found and sequenced. Current methodology for three-dimensional protein structure prediction depends on utilizing proteins sequences which are unknown and overlaying them with homologous proteins with known structure. Such practice has a mistake in it, as it does not consider the proteins which don't have a homologous protein in their database.

4.2 Proposed System Design

- The neural network divides the sequenced data into three convolutional layers with different filter sizes using batch normalization.
- This data condensed into a dense layer acts as an input for RNN with LSTM cells working in forward direction.
- The output from this layer is concatenated and passed down as input for RNN with LSTM cells working in backward direction.
- Use batch normalization again with different filter sizes to generate only the data that fits out model.
- Output of the last layer generates many unwanted productions of protein structure that are dropped in the Dropout layer.
- The last layer is the Softmax layer that will be used to provide the probability of secondary structure generated in the sequence.

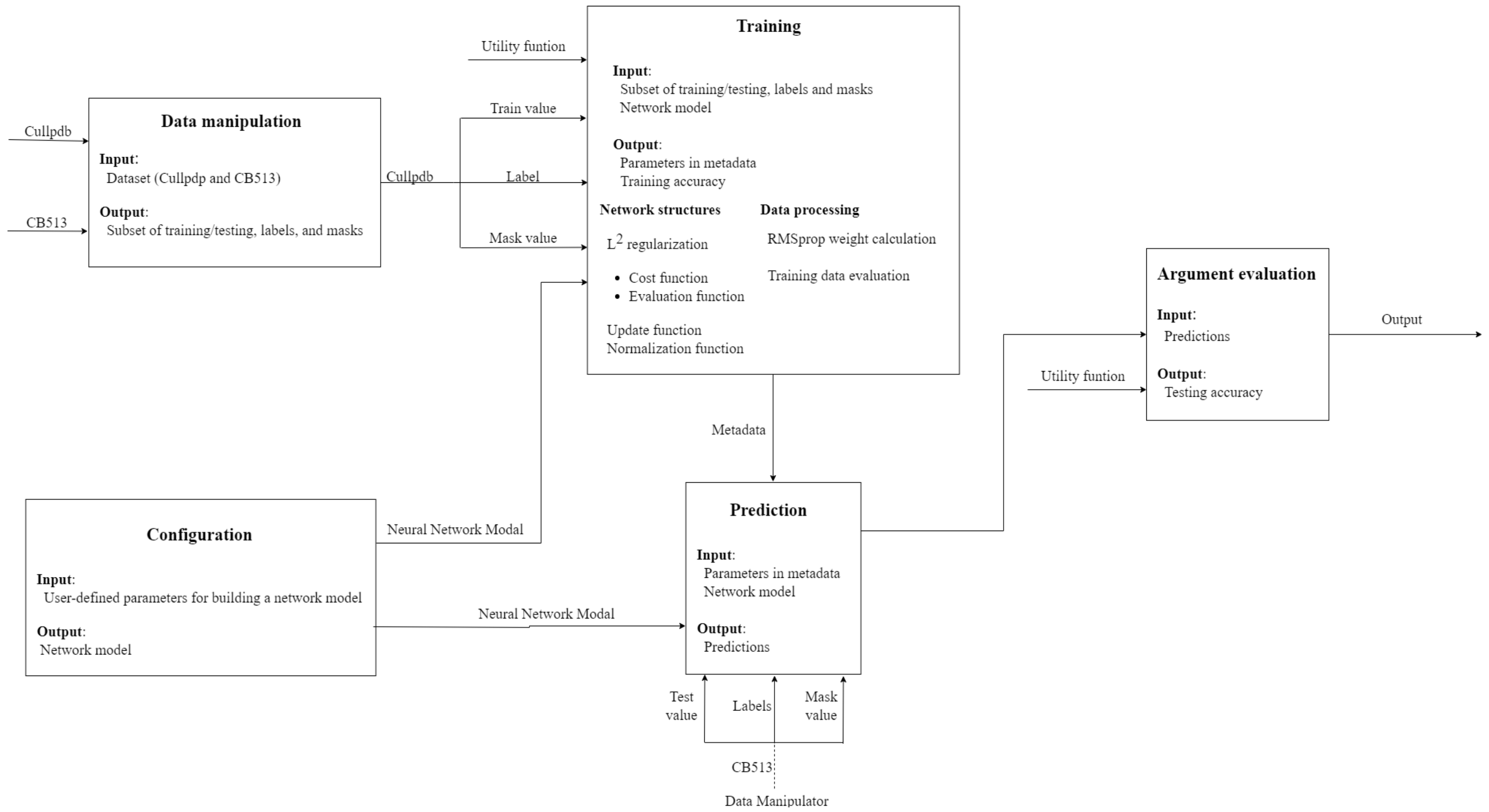


Fig 1. System Block Diagram

Data Manipulator

This is the module where data is extracted from the raw data file cullpdb+CB513 dataset. The dataset is packaged in the form of K-proteins * n features, which is reshaped using numpy to form K-proteins * 700 amino acid sequences * 57 features. Among those 57 features only 42 features are necessary for prediction. Additionally, we also extract the 8 secondary structure labels present among the 57 features on which the prediction occurs. The datasets have unique K values and have been filtered out such that there are no redundant values present among both the datasets. The lack of redundant data means that the test dataset is completely alien to the trained model and therefore more accurately represents the real world solution. This module takes no input but outputs two sets of training data, labels and mask values one for each training/validation and testing.

Configuration

This module contains blueprint for python libraries Theano and Lasagne to create the Multilayer Perceptron (MLP). It mainly consists of user defined hyper parameters such as convolution filter size, number of hidden layers between LSTM cells, learning rate and technique used for weight updates. The build model function specifies the order of building the network model. During training process when data is passed through this structure, a Q8 softmax prediction output is created. This module does not require any other inputs.

Training

This module is where the Multi-Layer Perceptron is built. The training process occurs in two stages: For the first stage, using Theano Tensor we create 3-D matrices to handle the data intensive calculations. A Structure is built to calculate the weight updates, costs and normal values. The second stage is where actual data from Data Manipulator model is imported and final output is calculated. During the stage one process we define regularization and batch normalization methods to converge on the desired result more efficiently. Also, overfitting of data is minimized using Bernoulli's droplets method. The second stage of training module runs the train process for a specified number of epochs on training values, labels and mask from Data manipulator module as directed by the Configurations module. During each epoch weight are calculated and updated using the method of Stochastic Gradient Descent (SGD). Lasagne Library allows us to directly implement SGD method with an optimizer called RMSprop. At the end of each epoch a small set of training dataset is used for validation of trained network model. Finally, at every 20 epochs we store the parameter values, accuracy for training dataset and other important metadata values in a python pickle file. This module requires training data from Data manipulator and Network model from configuration module.

Prediction

Prediction module imports the parameter values from metadata file created by the Training module. These parameters are fit over a newly created Multilayer Perceptron specifically created for testing CB513 data. Predictions are made and stored in along with other metadata.

Argument Evaluation

Argument evaluation module is to test if the predictions are correct for every 20 epochs. Predictions are compared with label values of CB513 dataset and final testing accuracy is determined.

4.3 Module Wise Explanation

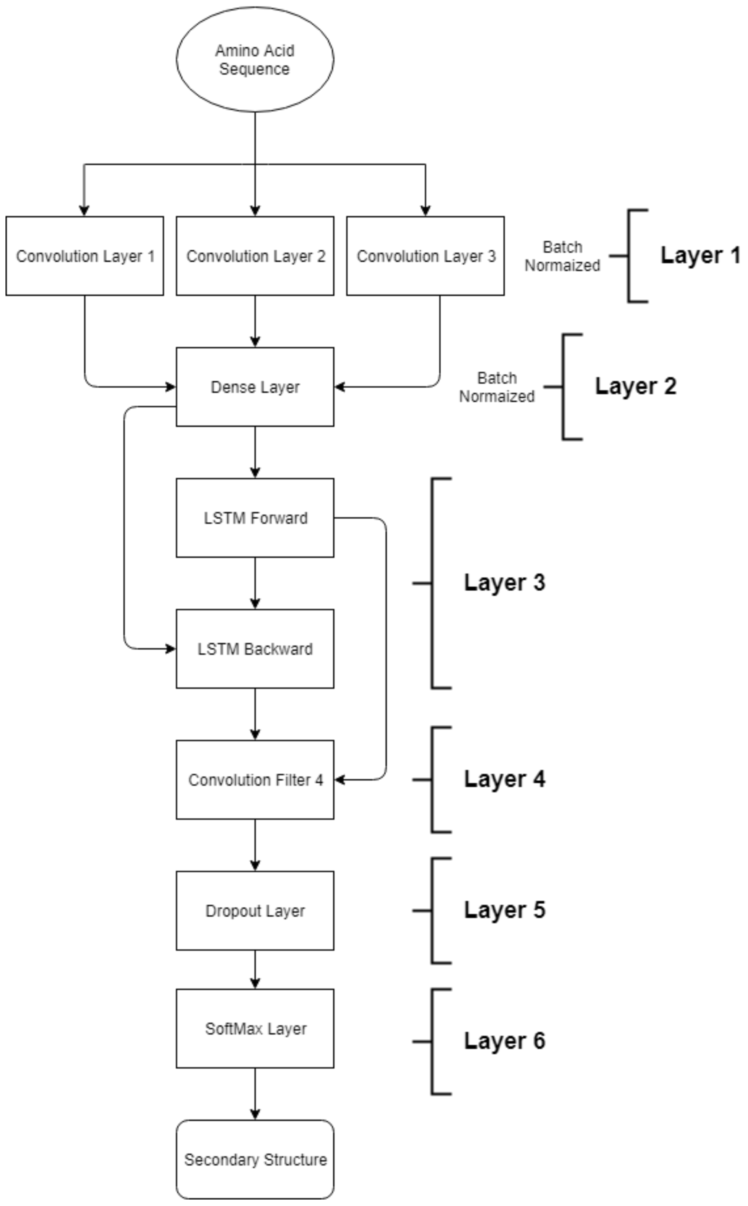


Fig 2. Neural Network Model

Neural Network

The overall architecture of this project is divided up into multiple layers for easier understanding and implementation purposes.

Module 1

Layer 1 and Layer 2

This is multi-layer perceptron (MLP) which is responsible for performing non-linear transformations on the previous layers. Input for this layer is the primary structure of amino acid and protein sequence profiles. The following standard linear algebraic equation defines the MLP.

$$z_{\ell+1} = h_l\theta_{\ell+1} + b_{\ell+1},$$
$$h_{\ell+1} = a(z_{\ell+1})$$

$$z_{l+1} = h_l\theta_{l+1} + b_{l+1}$$

$$zl+1 = hl\theta l+1+bl+1$$

h implies the current layer, θ is the weight matrix and b are the bias for the input z_{l+1}. The purpose of activation function, Rectifier Linear Unit (ReLU) $a(z) = max(0, z)$ where $a(z) = 1/(1 + e^{-z})$ and the Hyperbolic Tangent $a(z) = (e^z - e^{-z})/(e^z + e^{-z})$ is that their gradients do not vanish and it is easier to compute, which makes optimizing the network easier and quicker using stochastic gradient descent [10]. However, it suffers from “dead” gradients causing large neurons to completely miss activation. Therefore, a modified function called Leaky (slope where $a(z) < 0$) ReLU $a(z) = max(\alpha z, z)$ with $\alpha = 0.0$ is used [7].

To avoid overfitting of data, use Bernoulli’s dropouts as a regularizing layer defined by

$$h_{\ell+1} = h_l \odot p$$

Here *p* belongs to [0, 1] is a random sampled number with a hyper parameter determining bias towards 0 or 1 and Θ is elementwise multiplication.

Additionally, the data can be narrowed down by reducing the overall internal co-variate shifts by applying batch normalization [9] defined as:

$$\hat{\ell}^{(i)} = \frac{l^{(i)} - E[l^{(i)}]}{\sqrt{\mathrm{Var}[l^{(i)}]}},$$
$$u^{(i)} = \gamma^{(i)}\hat{\ell}^{(i)} + \beta^{(i)}$$

Batch normalization is incorporated after using the linear transformation and before using the nonlinear transformation. Giving us the final form:

$$
\begin{aligned}
z_{\ell+1} &= h_l \theta_{\ell+1}, \\
u_{\ell+1} &= \text{batch-norm}(z_{\ell+1}), \\
h_{\ell+1} &= a(u_{\ell+1})
\end{aligned}
$$

1D convolution filters are defined as following:

$$
\begin{aligned}
z^{id}_{\ell+1} &= x^{i}_{\ell} \theta^{d}_{\ell+1} + \beta^{d}_{\ell+1}, \\
h^{id}_{\ell+1} &= a(z^{id}_{\ell+1})
\end{aligned}
$$

Here, the input $x^i_l \in R^{kc \times 1}$ represents a co-located vector of length *k* in *c* channels and *i* refers to the specific co-located vector. The weight matrix $\theta^d_l \in R^{1 \times kc}$ corresponds to a spatial weight filter with *kc* connections, in the *d*th output channel, between the input layer and each neuron in the output layer and β^d_l is a scalar. As a result, convolutional layers are configurable to their filter size *k* and number of filters *d* [10].

This allows us to make filters that are configurable, allowing us to stack multiple convolutional filters on top of another allowing only a certain type of parameters to converge form the full parameter space. Additionally, this also allows us to vary the size of filters to improve validation.

Module 2

Layer 3

This layer provides Bidirectional Long-Short Term Memory (LSTM) with vertical links using Recurrent Neural Network (RNN). LSTM uses memory cells for modelling dependencies and improving convergence. It is defined as:

$$\begin{aligned}
i_t &= \sigma(x_t W_{xi} + h_{t-1} W_{hi} + b_i), \\
f_t &= \sigma(x_t W_{xf} + h_{t-1} W_{hf} + b_f), \\
o_t &= \sigma(x_t W_{xo} + h_{t-1} W_{ho} + b_o), \\
g_t &= tanh(x_t W_{xg} + h_{t-1} W_{hg} + b_g), \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t, \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}$$

Where $\sigma(z)$ is the sigmoid function $1/(1 + e^{-z})$ and x_t is the input, h_t^{l-1} is the previous layer's hidden state and c_t^{l-1} is the previous layer's hidden cell.

The LSTM works only in one direction, however, the layer can be implemented twice to cover the sequential information from both directions (forwards and backwards). Their hidden states, h_t for t = [1, 2,…, T] where T is sequence length, are concatenated such as suggested by Schuster & Paliwal, 1997 [11], such that:

$$h_t = \begin{bmatrix} \overrightarrow{h_t} \\ \overleftarrow{h_t} \end{bmatrix}$$

where $\overrightarrow{h_t}$ is the forward pass and $\overleftarrow{h}$ the backwards pass. Further sharing of hidden states, is possible with "Vertical Links" which is stacking the input, x_t, to the forwards pass hidden state, $\overrightarrow{h_t}$, of the previous pass. This operation allows the backwards pass to have a hidden state from both directions while computing. The concept could be extended to stacking several passes on top of one another with "Vertical Links".

Module 3

Layer 4 and Layer 5 and Layer 6

After passing the data from the LSTM layers, convolution filters will again be applied to converge on the parameter space followed by batch normalization. Unnecessary parameters will be dropped in the dropout layers and finally a specific set of parameters will be realized which will be sent to the 8-way SoftMax layer to draw out vectors with probabilities [1].

$$L(x, y) = -\sum_c y_c \ln(f_c(x))$$

Where x is the input vector, y the true label, $f(x)$ the neural networks probability prediction and c the class. The probability of prediction is set between τ and 1-τ to stabilize training and avoid gradient exploding.

We then apply the L^2 normalization such that:

$$regterm(\lambda, \theta) = \lambda\left(\sum_{n=1}^{N} \theta_n^2\right)$$

Where λ is a tunable hyper-parameter, N is the number of non-bias weights and θ is the weights in the neural network model. This is then added to the cost function [1].

$$L_{reg}(x, y) = L(y, f(x)) + regterm(\lambda, \theta)$$

4.4 Requirement Analysis

4.4.1 Functional Requirements

4.4.1.1 Product features

The main feature would be to produce the expected output out of a protein secondary structure prediction system. The product should give accuracy of the prediction so that the users may make decision accordingly. The system should handle the intense data computation and handling. The system should have an inbuilt error mechanism to avoid and discrepancy in the prediction process.

4.4.1.2 Assumption, Dependencies & Constraints

The dataset has no flaw and used as it is. There is a dependency between different epochs as after each iteration, the previously computed values improve the accuracy of the next. There are also inter-dependencies in calculation between tasks within a single epoch. The processing power of a single system is definite, which imposes constraints in the training of the system. The existence of dependencies in the software imposes constraints in the distribution of the tasks that are independent.

4.4.1.3 Domain Requirements

Given the system is application specific, it is important that it satisfy the domain requirement, which could be the field of medicine for cancer research, biology for understanding the human DNA. For such areas, it is essential that the structural features that can be analyzed from the secondary structure is as accurate as possible.

4.4.1.4 User Requirements

Since researchers in various fields use this type of application, the users should be well versed in their field as the inputs and outputs are very scientific in nature. The users are expected to know how to use the product, and analyze the generated output. They must also know what type of output the program generates.

4.4.2 Non-Functional Requirements

Robustness

It is very important for the system to give the output as correctly as possible and manage any error it may generate. An error handling mechanism can manage errors.

Performance

Due to the complexity of the program and large amount of dataset, the performance will be affected in terms of computation time.

Recoverability

If there is a system failure, there needs to be a backup of the software and data that can be easily handled in a distributed system.

Usability

The users must know how to use the application, what input is necessary for the output to be generated. This will not be much of a problem as the users of the secondary protein structure prediction software are predominantly professionals of their respective fields and are usually aware what inputs are necessary and what outputs are to be expected.

5. SCHEDULE, TASKS AND MILESTONES

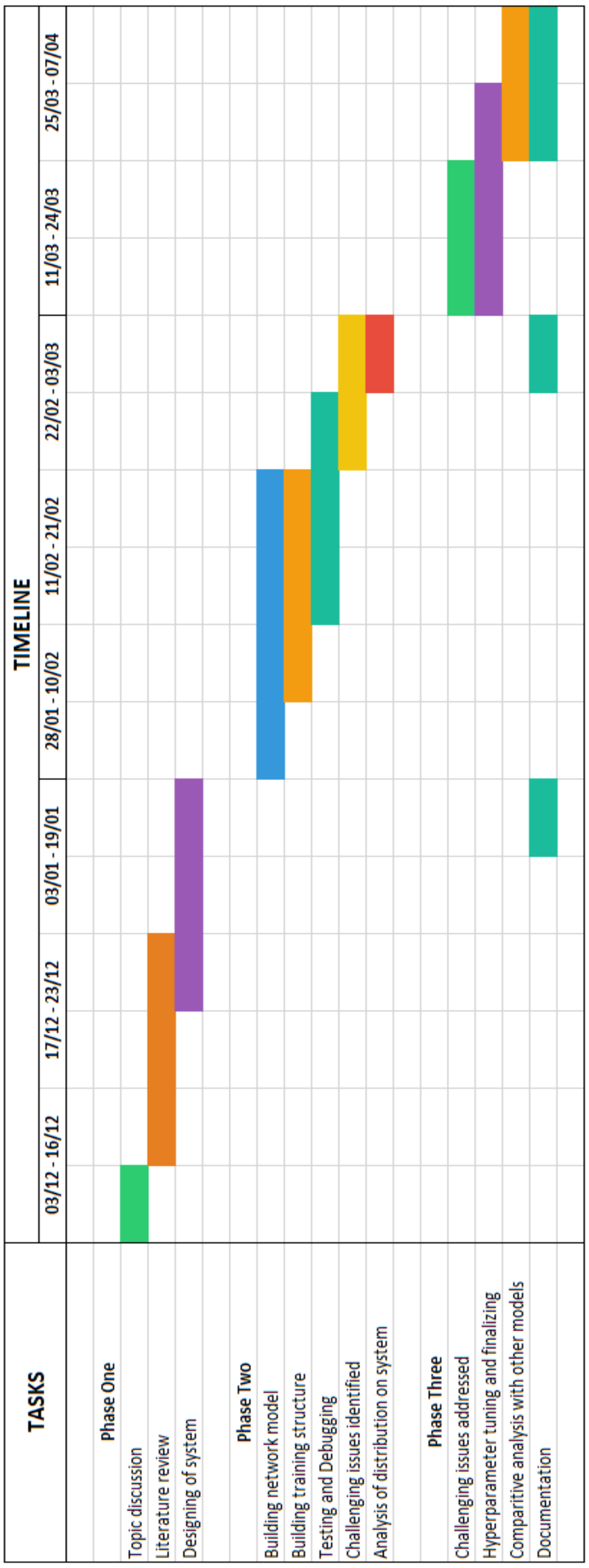
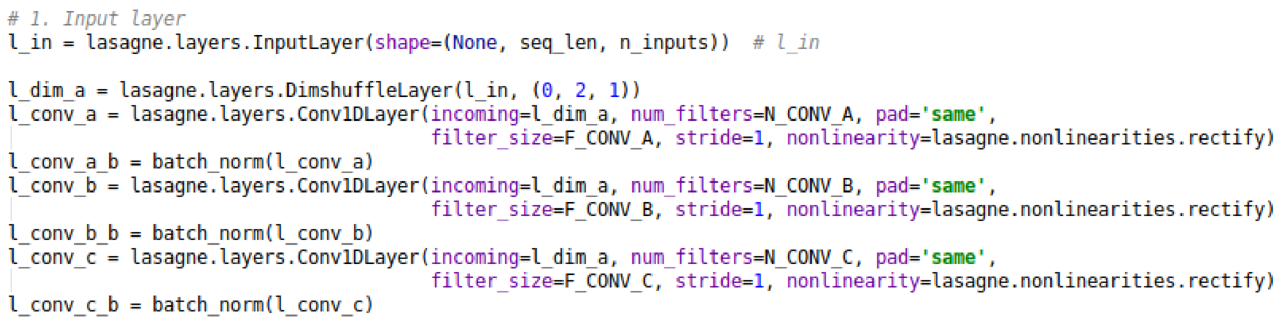


Fig 3. Gantt chart of schedule and milestones

6. PROJECT IMPLEMENTATION

Module 1

Layer 1

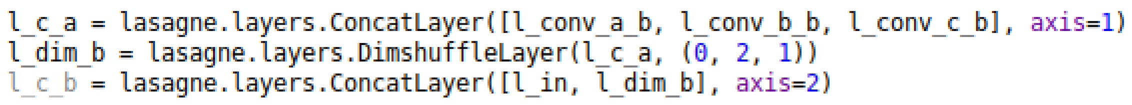


```
# 1. Input layer
l_in = lasagne.layers.InputLayer(shape=(None, seq_len, n_inputs))  # l_in

l_dim_a = lasagne.layers.DimshuffleLayer(l_in, (0, 2, 1))
l_conv_a = lasagne.layers.Conv1DLayer(incoming=l_dim_a, num_filters=N_CONV_A, pad='same',
                                      filter_size=F_CONV_A, stride=1, nonlinearity=lasagne.nonlinearities.rectify)
l_conv_a_b = batch_norm(l_conv_a)
l_conv_b = lasagne.layers.Conv1DLayer(incoming=l_dim_a, num_filters=N_CONV_B, pad='same',
                                      filter_size=F_CONV_B, stride=1, nonlinearity=lasagne.nonlinearities.rectify)
l_conv_b_b = batch_norm(l_conv_b)
l_conv_c = lasagne.layers.Conv1DLayer(incoming=l_dim_a, num_filters=N_CONV_C, pad='same',
                                      filter_size=F_CONV_C, stride=1, nonlinearity=lasagne.nonlinearities.rectify)
l_conv_c_b = batch_norm(l_conv_c)
```

Fig 4. Input Layer

To start the process of training we first create a 3D tensor based on the list of inputs from the dataset (700 sequences and 42 features). **InputLayer** holds a symbolic variable that represents a network input. **DimshuffleLayer** readjusts the dimension of its input tensor but still maintains the same total number of elements. This is necessary for convolution filter. **Conv1DLayer** performs a 1D convolution on its input. The features are processed through 3 convolution filters of sizes 3, 5 and 7 to compute the weighted averages, and optionally add a bias and apply an elementwise nonlinearity. The output of this module creates a tensor of 3 dimensions based on user-defined filters.

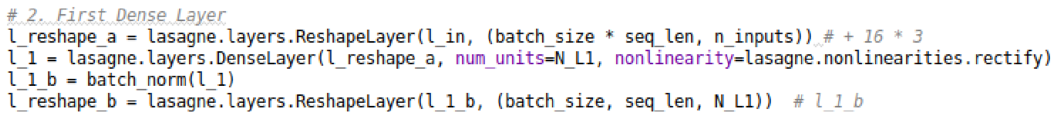


```
l_c_a = lasagne.layers.ConcatLayer([l_conv_a_b, l_conv_b_b, l_conv_c_b], axis=1)
l_dim_b = lasagne.layers.DimshuffleLayer(l_c_a, (0, 2, 1))
l_c_b = lasagne.layers.ConcatLayer([l_in, l_dim_b], axis=2)
```

Fig 5. Concat Layer after convolution

The **ConcatLayer** concatenates multiple inputs along the specified axis. The input for this layer is the filtered values from Conv1DLayer. The dimensions are reshuffled back and concatenated with initial input.

Layer 2



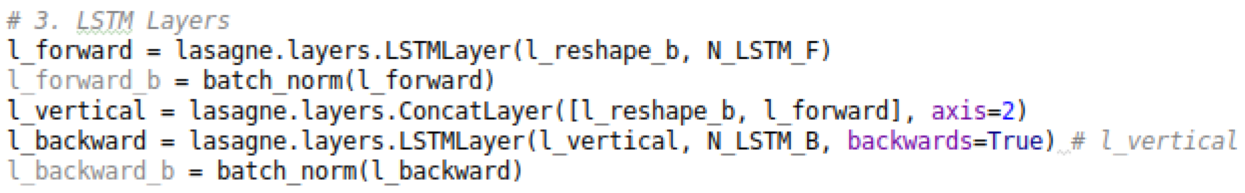
```
# 2. First Dense Layer
l_reshape_a = lasagne.layers.ReshapeLayer(l_in, (batch_size * seq_len, n_inputs)) # + 16 * 3
l_1 = lasagne.layers.DenseLayer(l_reshape_a, num_units=N_L1, nonlinearity=lasagne.nonlinearities.rectify)
l_1_b = batch_norm(l_1)
l_reshape_b = lasagne.layers.ReshapeLayer(l_1_b, (batch_size, seq_len, N_L1))  # l_1_b
```

Fig 6. First Dense Layer

A **ReshapeLayer** reshapes its input tensor to another tensor of the same total number of elements. This is necessary to increase the size of dimension for 3 convolution filters. **DenseLayer** forms a fully connected layer (number of units provides hidden states) based on the input list given.

Module 2

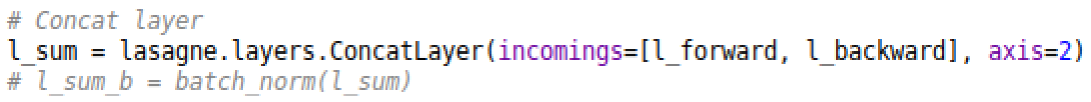
Layer 3



```
# 3. LSTM Layers
l_forward = lasagne.layers.LSTMLayer(l_reshape_b, N_LSTM_F)
l_forward_b = batch_norm(l_forward)
l_vertical = lasagne.layers.ConcatLayer([l_reshape_b, l_forward], axis=2)
l_backward = lasagne.layers.LSTMLayer(l_vertical, N_LSTM_B, backwards=True) # l_vertical
l_backward_b = batch_norm(l_backward)
```

Fig 7. LSTM Layers

To construct bi-directional recurrent layers, **LSTMLayer** is used which includes optional “peephole connections” and a forget gate. It creates forward and backward networks with vertical links. LSTM makes small modifications to the information by multiplications and additions.



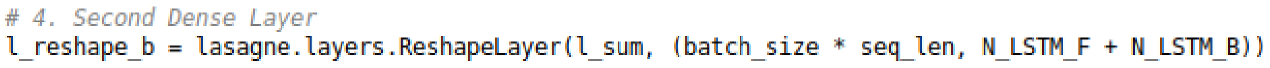
```
# Concat layer
l_sum = lasagne.layers.ConcatLayer(incomings=[l_forward, l_backward], axis=2)
# l_sum_b = batch_norm(l_sum)
```

Fig 8. Concat Layer after LSTM

The forward and backward networks are concatenated to form the overall tensor layer after LSTM.

Module 3

Layer 4

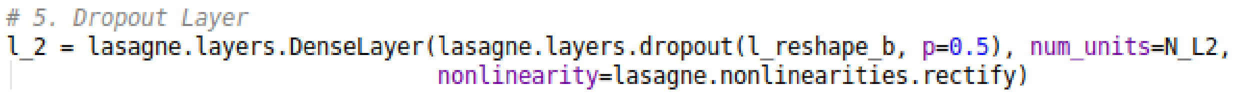


```
# 4. Second Dense Layer
l_reshape_b = lasagne.layers.ReshapeLayer(l_sum, (batch_size * seq_len, N_LSTM_F + N_LSTM_B))
```

Fig 9. Second Dense Layer

The overall tensor layer is reshaped to fit the batch size based on the required output dimensions for the dropout and Softmax layers. Dropout is a way to incorporate regularization in a neural network which helps in reducing interdependent learning among the neurons to avoid any over-fitting. Softmax is an activation function that turns numbers (logits) into probabilities that adds up to one which is also implemented in a multi-class world. It is executed via a neural network layer before the output layer. The number of nodes in the softmax layer must be equal to that of the output layer.

Layer 5

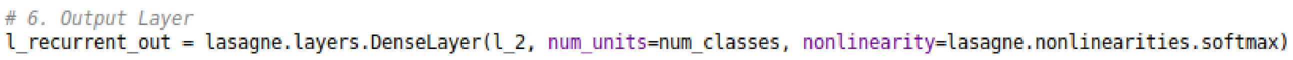


```
# 5. Dropout Layer
l_2 = lasagne.layers.DenseLayer(lasagne.layers.dropout(l_reshape_b, p=0.5), num_units=N_L2,
                                nonlinearity=lasagne.nonlinearities.rectify)
```

Fig 10. Dropout Layer

A DenseLayer is created from the **dropout** of the tensor layer which sets values to zero with probability p=0.5.

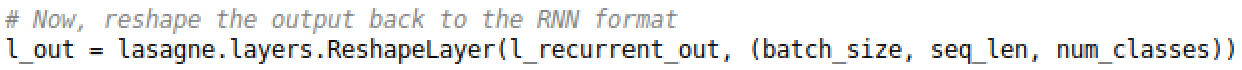
Layer 6



```
# 6. Output Layer
l_recurrent_out = lasagne.layers.DenseLayer(l_2, num_units=num_classes, nonlinearity=lasagne.nonlinearities.softmax)
```

Fig 11. Output Layer

Again, a DenseLayer is created from the tensor layer using Softmax activation function. This activation function gets applied row-wise

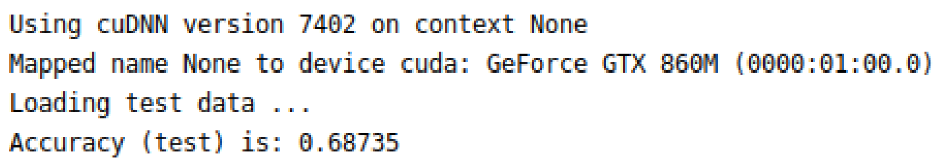


```
# Now, reshape the output back to the RNN format
l_out = lasagne.layers.ReshapeLayer(l_recurrent_out, (batch_size, seq_len, num_classes))
```

Fig 12. Reshaping of output

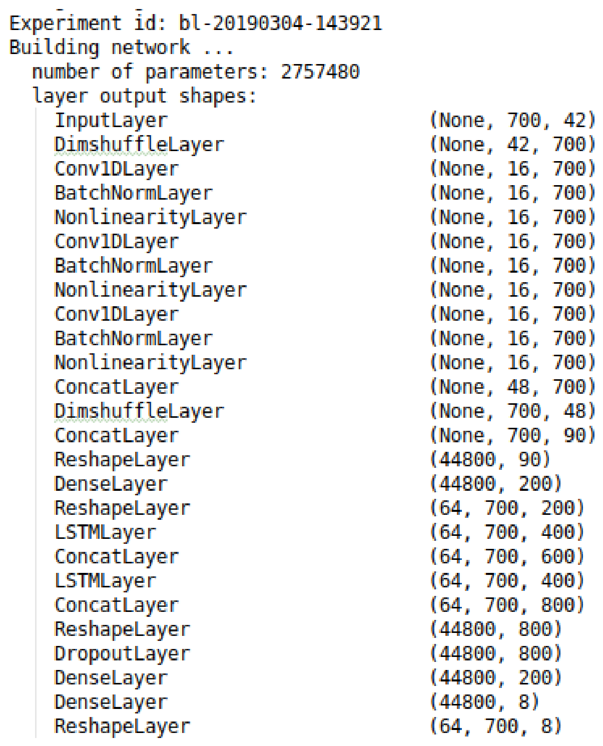
Finally, the output layer is created by reshaping the tensor layer to the required based on the batch size and number of classes.

OUTPUT:



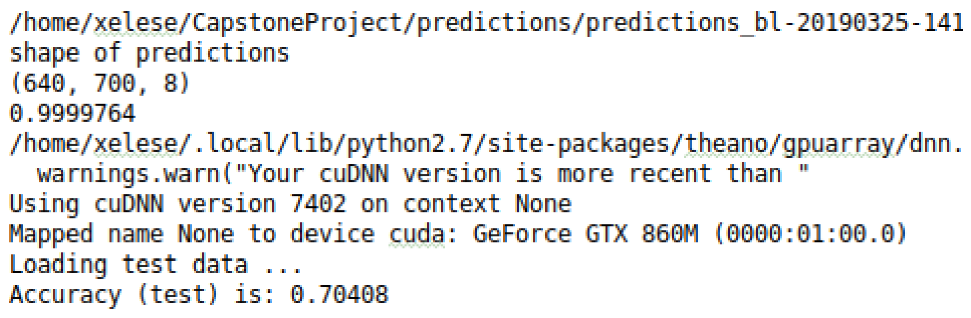
```
Using cuDNN version 7402 on context None
Mapped name None to device cuda: GeForce GTX 860M (0000:01:00.0)
Loading test data ...
Accuracy (test) is: 0.68735
```

Fig 13. Accuracy without CNN, vertical links and batch normalization



```
Experiment id: bl-20190304-143921
Building network ...
  number of parameters: 2757480
  layer output shapes:
    InputLayer                      (None, 700, 42)
    DimshuffleLayer                 (None, 42, 700)
    Conv1DLayer                     (None, 16, 700)
    BatchNormLayer                  (None, 16, 700)
    NonlinearityLayer               (None, 16, 700)
    Conv1DLayer                     (None, 16, 700)
    BatchNormLayer                  (None, 16, 700)
    NonlinearityLayer               (None, 16, 700)
    Conv1DLayer                     (None, 16, 700)
    BatchNormLayer                  (None, 16, 700)
    NonlinearityLayer               (None, 16, 700)
    ConcatLayer                     (None, 48, 700)
    DimshuffleLayer                 (None, 700, 48)
    ConcatLayer                     (None, 700, 90)
    ReshapeLayer                    (44800, 90)
    DenseLayer                      (44800, 200)
    ReshapeLayer                    (64, 700, 200)
    LSTMLayer                       (64, 700, 400)
    ConcatLayer                     (64, 700, 600)
    LSTMLayer                       (64, 700, 400)
    ConcatLayer                     (64, 700, 800)
    ReshapeLayer                    (44800, 800)
    DropoutLayer                    (44800, 800)
    DenseLayer                      (44800, 200)
    DenseLayer                      (44800, 8)
    ReshapeLayer                    (64, 700, 8)
```

Fig 14. Dimensions of Neural Network Model



```
/home/xelese/CapstoneProject/predictions/predictions_bl-20190325-141
shape of predictions
(640, 700, 8)
0.9999764
/home/xelese/.local/lib/python2.7/site-packages/theano/gpuarray/dnn.
  warnings.warn("Your cuDNN version is more recent than "
Using cuDNN version 7402 on context None
Mapped name None to device cuda: GeForce GTX 860M (0000:01:00.0)
Loading test data ...
Accuracy (test) is: 0.70408
```

Fig 15. Accuracy with CNN, vertical links and batch normalization

7. RESULTS AND DISCUSSION

After running the program for 600 epochs, which took us 36 hours to train the model and adding a batch normalization function before the LSTM layer, the evaluation function has given a score of **70.4%**. We were able to achieve this is on protein chain of 700 amino acid sequences which was a challenge found in previous models.

Compared to the base paper, our model is an improvement by 1.7%. This was possible because of better **hyper parameter tuning** and **strict control of physical data using convolutional filter** and **batch normalization**. Additionally we also implemented vertical links in our model successfully which was suggested as an improvement in the base paper.

After applying the filters suggested by different papers, we realized that some of the essential data gets filtered in 1D Convolution. This affects the accuracy of the prediction negatively. Hence, there is a need to evaluate each filter size and analyze how they affect the physical data and make changes accordingly.

Our results of ICML 2014 CB513 + CullPDB dataset has been summarised in the table 1. For comparisons, we have recreated the Bi-Directional LSTM as used by alexander et al., and also the SVM model for the aforementioned dataset.

Table 1. Comparison of Q8 accuracy between ours and others methods

Models	Q8 Accuracy
Linear Regression + K means [12]	0.412
SVM (ICML 2014 CB513+CULLPDB)	0.546
BI-LSTM (Base Paper)[3]	0.687
Our Model (ICML 2014 CB513+CULLPDB)	**0.704**

The SVM performs poorly than the neural network model, the reason being that the CULLPDB training dataset being used is suitable for RNN model than a classifier. This is because each protein sequence containing amino acid residues and sequence profiles has a uniquely labeled secondary structure which is non redundant with the CB513 testing dataset. The multi-layer perceptron is capable to handle this due to the possibility of multiple training iterations which eventually converges to the desired result.

SVM on the other hand did not perform well with the same dataset. We suspect that the dataset is not very compatible with linear classifiers including Linear Regression + K means because it has 700 amino acid sequence per protein which historically has given us a poorer classification result. While better accuracy can be achieved with shorter amino acid sequences, it is incorrect to do so as the real world amino acid sequence is 700 or longer and this is region where further study is required.

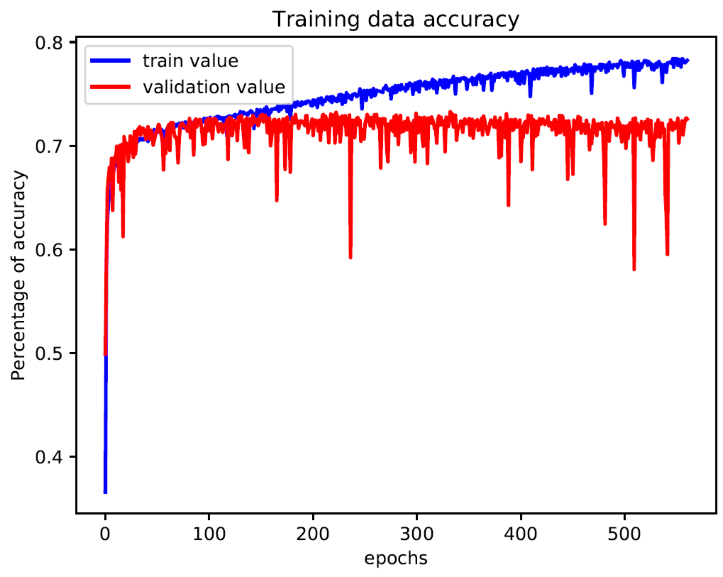


Fig 16. Training & Validation Accuracy of RNN Model

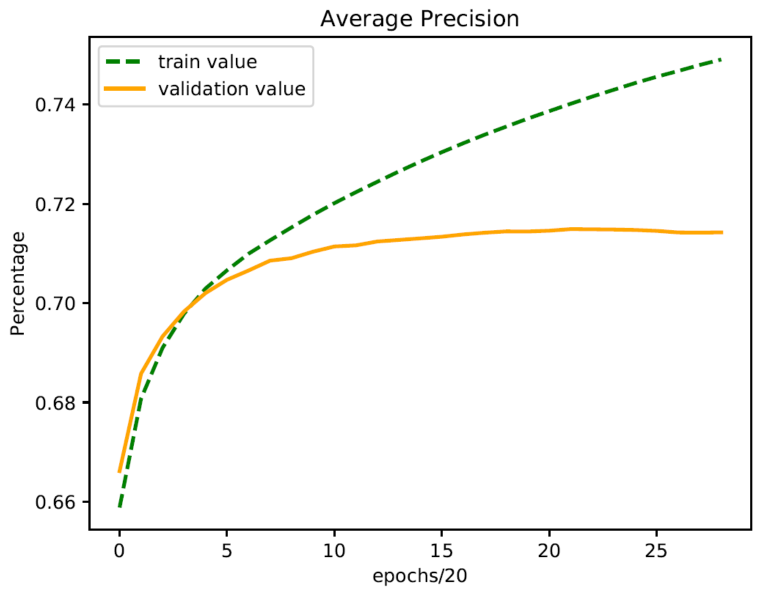


Fig 17. Average Precision of RNN Model

8. SUMMARY

Prediction of protein secondary structure from primary structure has been an active research area in hierarchical approach to the protein-folding problem.

The application works just as expected and by incorporating the features suggested in the survey and adding our own additional filters and tuned parameters and have obtained an accuracy of 70.4% in our prediction.

The amount of non-homogenous dataset available is limited. There are many task-dependent variables in the training process that makes its distribution difficult. This results in longer computation time.

Every established approaches have involved handcrafted rules by experts and Bayesian statistical methods. A variety of machine learning methods have been applied along with neural networks and symbolic induction. Comparison between different methods is difficult because different types of proteins used in their datasets.

The limitations of this project comes down to the accessibility of the dataset. The current dataset that we have used is proper for models like RNN and LSTM but not for classifiers such as SVM, Linear Regression and K means clustering. Thereby yielding a poor result.

9. REFERENCES

[1] Søren Kaae Sønderby and Ole Winther. Protein secondary structure prediction with long short term memory networks. arXiv preprint arXiv:1412.7828, 2014.

[2] Zikrija Avdagic, Prof.Dr.Sci Elvir Purisevic, Mr.Sci Samir Omanovic, Mr.Sci. Zlatan Coralic, Artificial Intelligence in Prediction of Secondary Protein Structure Using CB513 Database. 2009

[3] Alexander Rosenberg Johansen, Søren Kaae Sønderby, Ole Winther. Protein and secondary structure prediction with convolutions and Vertical-bi-directional RNNs. 2016

[4] Stephen Muggleton, Ross D.King and Michael J.E. Sternberg. Protein secondary structure prediction using logic-based machine learning. vol.5 no.7 pp.647-657, 1992

[5] Jian Zhou and Olga G Troyanskaya, "Deep supervised and convolutional generative stochastic network for protein secondary structure prediction," arXiv preprint arXiv:1403.1347, 2014.

[6] Sheng Wang, Jian Peng, Jianzhu Ma, and Jinbo Xu, "Protein secondary structure prediction using deep convolutional neural fields," Scientific reports, vol. 6, 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing humanlevel performance on imagenet classification," CoRR, vol. abs/1502.01852, 2015.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929–1958, Jan. 2014.

[9] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," CoRR, vol. abs/1502.03167, 2015.

[10]Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25, F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, Eds., pp. 1097–1105. Curran Associates, Inc., 2012.

[11]Schuster & Paliwal, "Bidirectional recurrent neural networks,"Signal Processing, 45, 1997.

[12]Abhishek Sethi, Prediction of Protein Secondary Structure using Logistic Regression, K-Means Clustering, and Naïve Bayes Variation.

APPENDIX – A

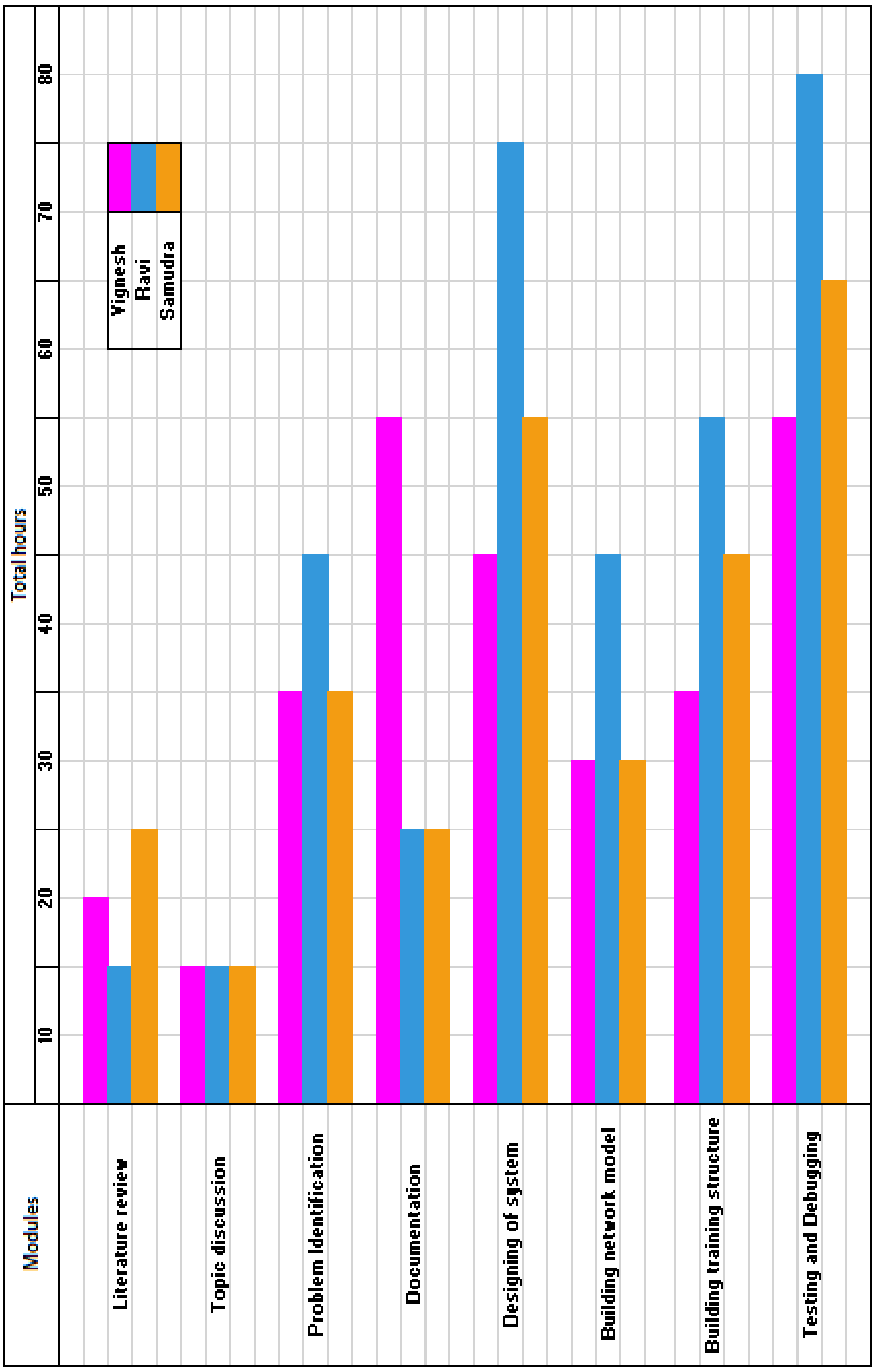


Fig 18. Individual Contributions