Understanding Protein Function Prediction using Deep Learning

Submitted by
Fathy Shalaby

Submitted at
Institute for Machine Learning

Supervisor
Univ.Prof.Sepp Hochreiter

Co-Supervisor
Msc.Michael Widrich

March 2021

Bachelor Thesis

to obtain the academic degree of

Bachelor of Science

in the Bioinformatic Program

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Citation

Shalaby, F. (2021). Understanding Protein Function Prediction using Deep Learning. Bachelor Thesis, in English. – 43 p.Institute for Machine Learning, Faculty of Engineering Natural Sciences, Johannes Kepler Univeristy, Linz Austria.

Annotation

Protein function prediction is a crucial task in the area of Bioinformatics especially since the recent advances in Next Generation Sequencing. The most popularized approach used is the utilization of Machine learning or more specifically Deep Learning. There are many methods that have been published which produce good results however are based on large complex models. In this thesis it was attempted to try to solve such an task by using a basic model with simple annotation however try to utilize some of the relationships between classes in the space of auxiliary tasks and their effect on the performance of the model regarding the prediction of a specified main class.

Affirmation

II hereby declare that I have worked on my bachelor's thesis independently and used only the sources listed in the bibliography. I hereby declare that, in accordance with Article 47b of Act No. 111/1998 in the valid wording, I agree with the publication of my bachelor thesis, in full. Further, I agree to the electronic publication of the comments of my supervisor and thesis opponents and the record of the proceedings and results of the thesis defence in accordance with aforementioned Act No. 111/1998. I also agree to the comparison of the text of my thesis with the Theses.cz thesis database operated by the National Registry of University Theses and a plagiarism detection system.

Wien, 12.03.2021 [illegible]

Place, Date Fathy Shalaby

Abstract

Due to the recent advancement in Next Generation Sequencing (NGS), sequence sequencing has greatly increased since it was first applied in 2009. However, a well-known bottleneck is a low scalability and high costs of classifying protein functions. In recent years, finding a solution to this problem has been an active field of research. The most promising was the current utilization of Machine Learning (ML), specifically Deep Learning (DL), which has shown promise in addressing this bottleneck. However, protein sequences have multiple functions, making annotating these sequences a large-scale, multi-scale, multi-label problem. In this thesis, I attempt to use a DL architecture known as Long Short Term Memory, which has shown much promise in the area of sequence-based data. I aim to observe the effect of auxiliary classes on the performance on a chosen main class. This is done to fully evaluate the performance of this DL architecture on such a task. Two experiments were performed. The first is a binary task where I created the baseline through an extensive hyper-parameter search. The second experiment aimed to see the effect of adding auxiliary classes, in order to observe if it would either increase or decrease the model's performance. The experiments both take the sequences as input, as output the corresponding Gene Ontology Identification (GO-ID). Which can be used to determine the protein's possible function, which could be utilized in many research areas, for example, phylogenetics. For the binary task, the model achieved an area under the receiver operating characteristic curve score of 0.81, a binary logistic loos of 0.165, and balanced accuracy of 0.775. For the auxiliary task, the model achieved an area under the receiver operating characteristic curve score of 0.8238, a binary logistic loss of 0.0481, and balanced accuracy of 0.7231 on the main class. Furthermore, when using a Mean Squared Error (MSE) loss, the model achieved an area under the receiver operating characteristic curve score of 0.8698, a loss of 0.0152, and a balanced accuracy of 0.5328 when with auxiliary tasks. Indicating that the usage of auxiliary classes to predict the main class do improve performance slightly in general.

Affirmation **iii**

Abstract **iv**

Introduction **1**

1 Background & Foundation **2**

1.1 Biology 2

1.1.1 Proteins 2

1.1.2 Gene Ontology 3

1.2 Machine Learning 4

1.3 Deep Learning 4

1.3.1 Neural Networks 5

1.3.2 Feedforward Neural Network 6

1.3.3 Recurrent Neural Networks 6

1.3.4 Vanishing and Exploding gradient problem 7

1.3.5 Long short term memory architecture 7

2 Related Work **9**

3 Method **10**

3.1 Dataset 10

3.1.1 Data Representation 11

3.2 Model Structure 12

3.3 Experiment and setup 13

3.3.1 Binary experiment 13

3.3.2 Auxiliary experiment 15

3.4 Resources 15

3.5 Tools 16

3.6 Hyper-parameter 16

3.7 Limitations 16

4 Results & Discussion **18**

4.1 Initial 18

4.2 Linear vs Tanh output function 22

4.3 Binary experiments 26

4.3.1 Hyper-parameter search result 26
4.3.2 Baseline 27
4.4 Auxiliary experiments 27

5 Conclusion 29

6 Future Work 30

7 Acknowledgement 31

Bibliography 32

Appendix 34

Introduction

Machine Learning (ML), especially Deep Learning (DL), has significantly advanced since introducing the first recorded DL method. Since then, the application of deep learning has increased drastically, ranging from weather prediction to protein sequencing and image classification. One of the areas which adapted to using DL methods is Bioinformatics. Bioinformatics is an interdisciplinary study that combines the area of Biochemistry and Computer science to attempt solving tasks ranging from toxicity prediction [May16] to protein homology detection [HHO]. A topic that has been studied well at the time of writing this thesis is protein function prediction by using Gene Ontologies (GO). It is an active field that is being researched by many [Alm17; KKH18; Jur17; Ran20] Gene Ontology is a formal representation of the knowledge in the domain of biology [The18]. This topic is attractive due to the recent advancement in Next-Generation Sequencing. Due to this advancement, the number of sequences that can be sequenced has significantly increased. However, a well-known bottleneck is a low scaleability and high costs of classifying protein functions. However, as stated previously, DL has shown promise in addressing this bottleneck. However, the main drawback is that protein sequences have multiple functions, making annotating these sequences a large-scale, multi-scale, multi-label problem. Therefore to better understand how this problem has been addressed and increase the understanding of the capabilities of DL on this task, the goal of this thesis is to first evaluate the performance of such DL method, at the same time provide a general introduction to the topic and what first steps can be done and second also evaluate the benefit of auxiliary classes on the performance of the model.

This thesis is structured to be understood by a reader with little background in this area. Therefore, first, I introduce the foundation and knowledge needed, which quickly covers the thesis's biological aspect, and then dives into machine learning and deep learning. Secondly, I briefly discuss related work that attempts to perform the similar tasks using other methods. Next, the model structure and the experimental setups are explained in some detail. Following that, the results of the different experiments are presented, which include a hyper-parameter search to find the best parameters and an exploratory investigation to get a quick idea regarding the possible outcome of performing an auxiliary task. Finally, all results are discussed, the conclusion is presented, and future work is stated.

1 Background & Foundation

1.1 Biology

In this section, the biological background for this thesis is presented.

1.1.1 Proteins

Proteins are macromolecules that are made up of one or many long amino acid chains. It is essential to study proteins due to the variety of functions in organisms, which ranges from DNA replication, responding to stimuli, catalysing metabolic reactions, providing structure to cells, and organisms, and transporting molecules from one location to another. Proteins differ mainly in their amino acid sequences which usually results in protein folding into a specific three-dimensional structure that determines its activity. Thus meaning two proteins with similar amino acid sequences may have two completely different functions, which makes their annotation not an easy task. There are four different protein structures. The primary structure which is a long amino-acid chain with no folds Figure1.1(a). Secondary structure folding and coiling occur, and we get two types of structures known as alpha-helix and beta-sheets Figure1.1(b). In tertiary structure, the protein becomes more complicated by the introduction of intramolecular forces like hydrophobic interaction and hydrogen bonding Figure1.1(c). Quaternary structure refers to the structure of a protein macromolecule formed by interactions between multiple polypeptide chains Figure1.1(d). Proteins with quaternary structure may consist of more than one of the same type of protein subunit or also different subunits. Hemoglobin is an example of a protein with a quaternary structure.

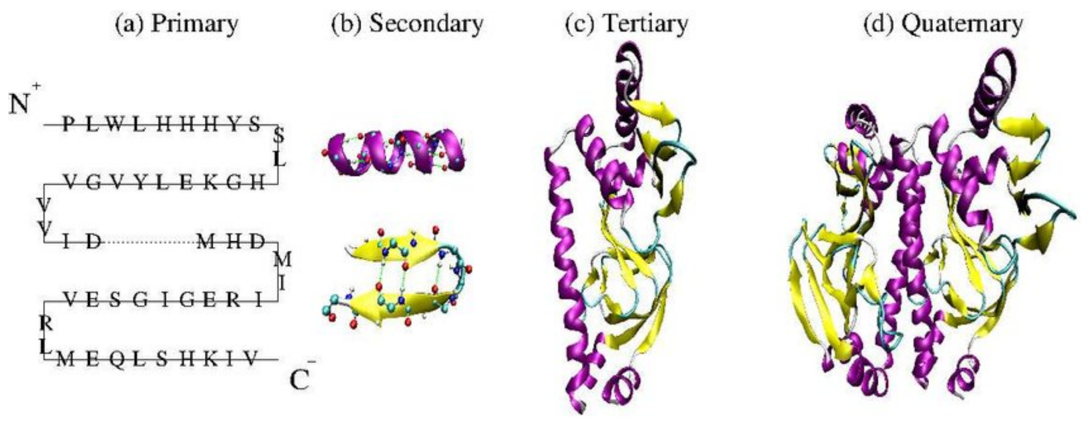


Figure 1.1: a diagram of the four main structures of a protein[Kou13]

1.1.2 Gene Ontology

A Gene Ontology (GO) is a formal representation of the knowledge in the domain of biology. This knowledge is represented as three aspects, which are biological processes, molecular functions, and cellular component. Each one of these aspects is essential to be able to represent the knowledge correctly. This representation is structured as a graph where each GO is a node and the relationships between the terms are edges between the nodes. A GO class is mainly composed of a definition, a label, a unique identifier and relations terms. There are also optional elements like obsolete tag or database cross-reference. There are three main relations in GO, is a (forms the basic structure of GO. If we say A is a B, we mean that node A is a subtype of node B.), part of (wherever B exists, it is as part of A, and the presence of the B implies the presence of A.), has part (where A always has B as a part, i.e. where A necessarily has part B.) and regulates (if both A and B are present, B always regulates A, but A may not always be regulated by B), these relations allow for better understanding of the domain of biology as shown in Figure.1.2.

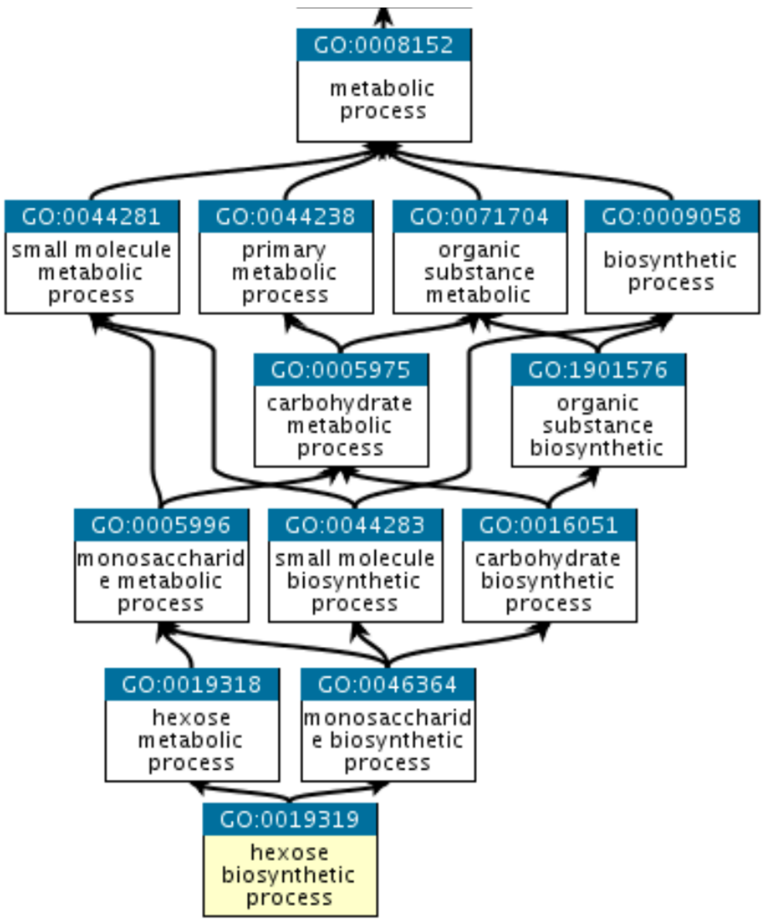

GO:0008152
metabolic process
GO:0044281
small molecule metabolic process
GO:0044238
primary metabolic process
GO:0071704
organic substance metabolic
GO:0009058
biosynthetic process
GO:0005975
carbohydrate metabolic process
GO:1901576
organic substance biosynthetic
GO:0005996
monosaccharide metabolic process
GO:0044283
small molecule biosynthetic process
GO:0016051
carbohydrate biosynthetic process
GO:0019318
hexose metabolic process
GO:0046364
monosaccharide biosynthetic process
GO:0019319
hexose biosynthetic process

Figure 1.2: A GO Graph for hexose synthetic process showing the structure of GO

1.2 Machine Learning

Machine Learning (ML) is the area of study where one aims to teach or train a machine to do specific tasks without needing to give exact instructions.
For example, if we look at the typical image classification case using the MNIST Dataset[1] we see, if hard coded, this would require one to write a long program that takes every possible variation in mind. However, if we use a machine learning algorithm like a linear regression model, which uses a linear model to recognize the patterns in the data by looking at different features and separating them, one can easily perform well on this task with unseen data without needing to write more code. Machine learning can be split into three sub-topics Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Supervised learning concerns itself with learning tasks where the model is given samples that are already labeled. A common task is classification[2] or regression[3], which are normally applied on data similar to MNIST mentioned previously. Unsupervised learning concerns itself with learning tasks where the model is given samples, however, no labels,only samples and mainly attempt to group or cluster the data based on common features. A standard method which is used in unsupervised learning is Principle Component Analysis (PCA) [4]. Reinforcement Learning concerns itself with how software agents ought to take actions in an environment to maximize the notion of cumulative reward. For this thesis Supervised learning will be utilized.

1.3 Deep Learning

Unlike the three branches of machine learning presented, which use approaches like Random Forest[5], PCA, and Q-learning[6], Deep Learning utilize Neural Networks. By utilizing neural networks, the Deep Learning approaches like Feedforward Neural Network (FNN), Recurrent Neural Network (RNN), and Long Short Term Memory Neural Network (LSTM) can, without the need of much prepossessing, perform great on complex data where as feature selection would not be quickly done. This characteristic resulted in the recent boom in applications that

[1] It is a common baseline dataset. The dataset contains handwritten numbers, and the task is to recognize the numbers written.

[2] The model tries to predict the class a sample is from based on features that the data shares

[3] Models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting

[4] A dimensionality-reduction method that which allows to down project the data into a more distinct and more comfortable to learn dimension while keeping the nearly same information as the original data.

[5] an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees

[6] a model-free reinforcement learning algorithm to learn quality of actions telling an agent what action to take under what circumstances

utilize Deep Learning for tasks ranging from image classification to drug discovery.

1.3.1 Neural Networks

Neural Networks are set of algorithms modeled loosely after the human brain called perceptrons. They are superficial nodes that can take multiple inputs x_i, apply a weight w_i to each of them, and produce one output y. By then, placing many of these perceptions in a row creates a Neural Network that can take input and generate an output based on the different connections' weights. Those have one input layer, one hidden layer, and one output layer. If more hidden layers are added, it is now called a Deep Neural Network (DNN). The output of the first hidden layer becomes an input in the second hidden layer, and so on. This allows for more complex hidden representations, which in turn allows for solving more complex tasks.

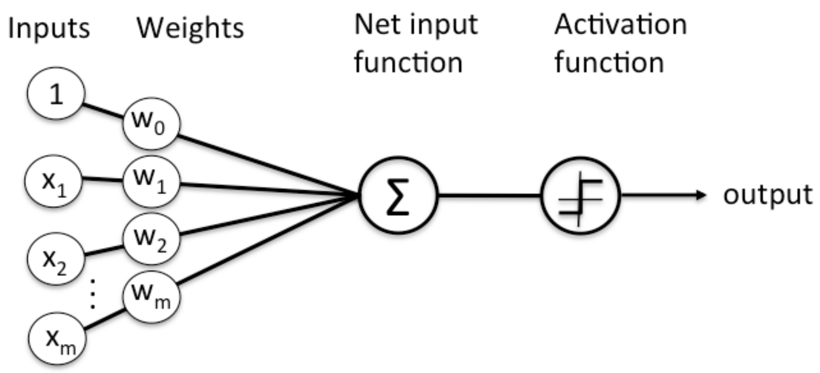


Figure 1.3: A diagram of a Feedforward Neural Network.

These Neural Networks are designed to recognize patterns in the data, through this patterns be able to learn and perform inference on the data. They interpret sensory data through a kind of machine perception, labeling (supervised Learning), or clustering raw input (unsupervised Learning). The patterns they recognize are in numerical form and normally contained in vectors, into which all real-world data, be it images, sound, text, or time series, must be translated to.

A Neural Network consists of layers which are made of nodes (neurons), and each node is connected to each other. This structure allows for information to flow from one node to the other, where at the same time, patterns and features are detected from the inputs.

The most basic and commonly known neural network is a Feedforward Neural Network, which utilizes linear activation function to control the information flow and thus detect features in the data.

1.3.2 Feedforward Neural Network

An Neural Network's basic example is a simple Feedforward Neural Network (FNN), where the information only moves in one direction. The Network is normally made up of an input layer (X) and sometimes a hidden layer, and finally an output layer. Each of these layers comprises nodes or neurons, and each node has weights[7], a net input function[8], and an activation function that activates the node, by calculating a weighted sum of the inputs and calculates the output. A feed-forward network can be considered as a function: $\hat{y} = g(x; w)$ that maps an input vector x to an output (or prediction) vector using network parameters w. That means the forward pass activates the network depending on the input variables only and produces output values [Arr19].

1.3.3 Recurrent Neural Networks

The main difference between a normal FNN and a RNN is that RNN utilize loopback connections to allow them to look at previous hidden states and thus simulate memory. Hidden state is the representation of the previous input which is used to simulate memory in the network. The structural difference can be seen in Figure 1.4.
In general RNNs are based on the idea of utilizing both the sequence and the previous hidden state ($h_{t-1}l$) where l is the layer and t is the time step. In Figure. 1.4,the left graph shows a

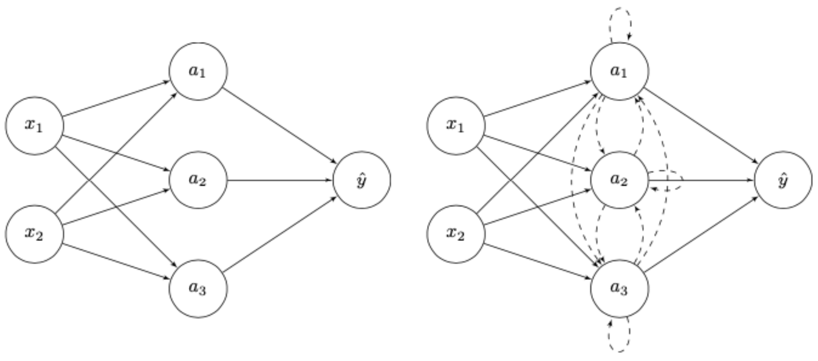


Figure 1.4: Fully connected vs. recurrent network.

simple FNN with input layer, hidden layer, and output layer. The right graph a recurrent network. It has the same basic architecture, but with recurrent connections (loops) in the hidden layer. The dashed lines indicate time lag, i.e. the transformation takes values at time t − 1 and

[7]The strength of the connection, how does changing this input affect the output of the network. Normally weights are set to a number near zero first, meaning that changing the input will not affect the output, which simplifies the model.

[8]Calculates the layers net input by matrix multiplication between the weighted inputs and biases.

feeds them back into the network at time t. As you can see, the loop connections make the difference between these two architectures [Arr19].
An forward pass would be as follows:

$$\mathbf{s}(t) = \mathbf{W}^\top x(t) + \mathbf{R}^\top \alpha(t-1) \tag{1.1}$$

$$a(t) = f(s(t)) \tag{1.2}$$

$$\hat{y} = \varphi(\mathbf{V}^\top a(t)) \tag{1.3}$$

Where $\mathbf{W}$ is the input weight matrix, $\mathbf{R}$ is the recurrent weight matrix, $\mathbf{V}$ is the output weight matrix, $s(t)$ is the pre-activations at time t, a(t) is the hidden activation at time t.

1.3.4 Vanishing and Exploding gradient problem

One of the most significant problems or difficulties of training an RNN are the two problems known as the vanishing and exploding gradient problem. The vanishing gradient occurs during Back Propagation Through Time (BPTT)[9] as the learning algorithm cannot carry the error signal over more than a few steps backward in time. This results in catastrophic learning as the weight updates become infinitesimally small, resulting in no change in weights and learning stalls. Similar to exploding gradients, the absolute values of the weights quickly become more extensive than what a computer can represent, and we run into overflows. These problems are discussed in more detail in both [BSF94] and also [Hoc91].

1.3.5 Long short term memory architecture

LSTMs are designed to address the Vanishing, and Exploding gradient problem [BSF94]. During backpropagation, gradients would either vanish or explode, greatly affecting the network's performance on long sequences, which is known as the vanishing gradient problem discussed in the previous subsection. The traditional approach to mitigating this problem was gradient clipping at a maximum value. However LSTM utilizes it ability to store both a hidden state vector (h_t^l) and memory vector (c_tl). This ability allows the LSTM to decide at each time step to either read from (input gate), write to (output gate), or reset the cell (forget gate[10]). It protects jamming hidden units, other cells, and the output layer with information that may be currently

[9]A training method that has become the norm for training feed-forward models.
[10]The original LSTM developed by [HS97] did not have a forget gate which required a lot of cell memory.[GSC99] implemented the forget gate to solve that problem.

irrelevant but important later. A memory cell consists of:

$$\mathbf{i}(t) = \sigma(\mathbf{W}_i^\top x(t) + \mathbf{R}_i^\top y(t-1)) \tag{1.4}$$

$$\mathbf{o}(t) = \sigma(\mathbf{W}_o^\top x(t) + \mathbf{R}_o^\top y(t-1) \tag{1.5}$$

$$\mathbf{z}(t) = \sigma(\mathbf{W}_z^\top x(t) + \mathbf{R}_z^\top y(t-1) \tag{1.6}$$

$$\mathbf{c}(t) = c(t-1) + \mathbf{i}(t) \odot z(t) \tag{1.7}$$

$$\mathbf{y}(t) = o(t) \odot h(c(t)) \tag{1.8}$$

Where vector x(t) is the external input and y(t-1) are the memory cell activation's of the last time step.The vector i(t) is the input gate activation, o(t) is the output activation, z(t) is the cell input activation, c(t) is the cell state, and y(t) is the current memory cell activation vector. The function g is the cell input activation function and the function h is the memory cell activation function. h(c) is the memory cell state activation. The operator $\odot$ is Hadamard's product, that is a point-wise vector product [Arr19]. An example of the LSTM structure can be seen in Figure.1.5.

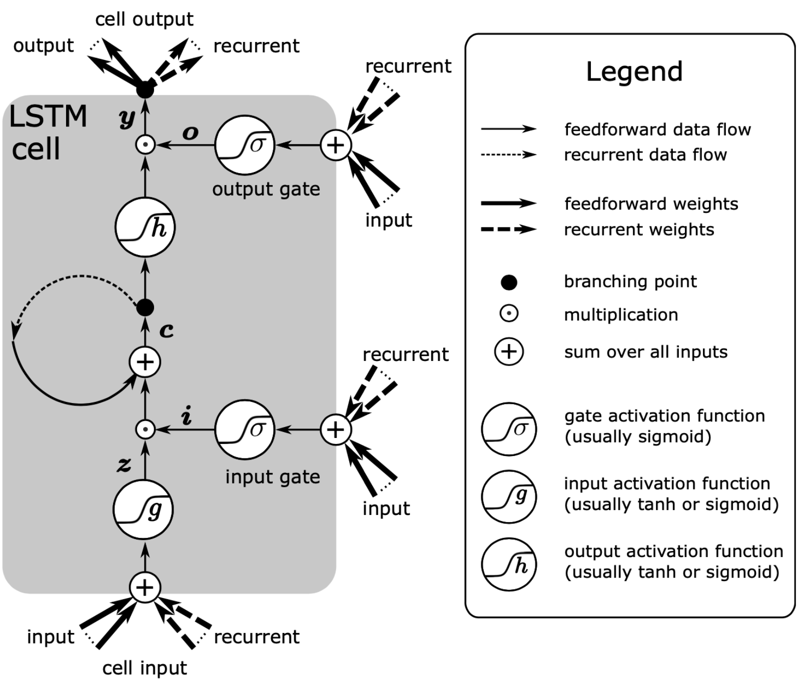


Figure 1.5: A LSTM memory cell [Arr19]

2 Related Work

The topic of using deep learning for the prediction or classification of protein sequences by ontology class and thus determine their function has been explored previously by [Sha19] and [KKH18].

In [Sha19], a similar task was performed where the goal was also to use a deep learning method to be able to predict the protein function by classification through Gene ontology tags or id. There a three-layer convolutional neural network with a max-pooling layer was used to train a model that attempts to predict the correct class in multi-class classification space. Maxpooling was used to create a abstracted representation of the sequences and reduce overfitting. The results were not as expected; however it did display a simple model's performance on a complex task.

[KKH18] observed that proteins have multiple functions, making function prediction a large-scale, multi-class, multi-label problem. Therefore they developed a novel method to predict protein function from sequence. They used deep learning to learn features from protein sequences as well as a cross-species protein-protein interaction network. Their approach specifically outputs information in the GO structure and utilizes the dependencies between GO classes as background information to construct a deep learning model. They evaluated their method using the standards established by the Computational Assessment of Function Annotation (CAFA) and demonstrated a significant improvement over baseline methods such as BLAST, with significant improvement for predicting cellular locations.

In the paper [Alm17], they presented a prediction algorithm using deep neural networks to predict protein subcellular localization relying only on sequence information. The prediction model uses a recurrent neural network that processes the entire protein sequence and an attention mechanism identifying protein regions important for the subcellular localization. The model was trained and tested on a protein dataset extracted from one of the latest UniProt releases at that time (2016 version 4), in which experimentally annotated proteins follow more stringent criteria than previously. they demonstrated that their model achieves a good accuracy (78 percent for ten categories; 92 percent for membrane-bound or soluble), outperforming current state-of-the-art algorithms, including those relying on homology information.

3 Method

In this section the different resources, data processing methods and hyper-parameter searches used for this thesis are discussed. The goal was to determine the predictive capability of the model in the space of protein function predict through the assigning of the correct Gene-Ontology tags when given a specific protein sequence.

3.1 Dataset

In this thesis, the Gene-Ontology (GO) dataset[11] GO has an acyclic graph structure and has three major domains, which are, biological processes, molecular functions, and cellular component. Each of these domains contain 30,821, 12,133 and 4,407 classes, respectively. As the aim of this thesis was to attempt to predict protein function, biological processes is chosen as the domain best fits for this task. [Gen04]
Furthermore I used SwissProt's [Bou16] reviewed and manually annotated protein sequences from Homo Sapiens with GO annotations. The data set contained 20,394 samples. The data was filtered by first removing samples with ambiguous amino acid codes (B, O, J, U, X, Z) in their sequence. Next a maximum sequence length of 1000 was set based on sequence length distribution visually observed Figure.3.1 where the majority of the sequence seem to be in the 0-1000 range. The final size of the dataset becomes 19,257 samples.

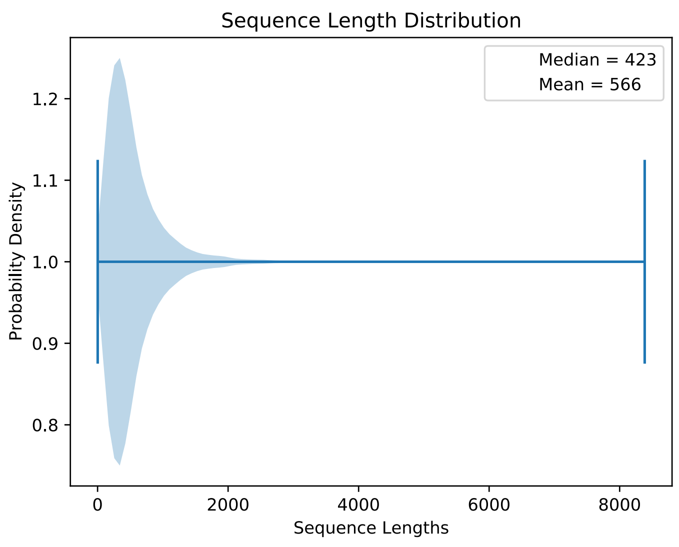


Figure 3.1: The sequence lengths from the Swissprot file of the species human

[11] http://purl.obolibrary.org/obo/go/go-basic.obo (format version 1.2)

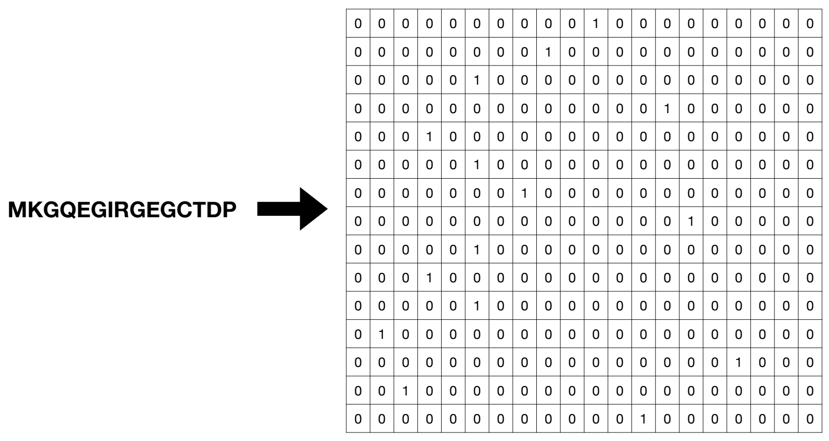
3.1.1 Data Representation

The Dataset is created from three sources:

- Gene-Ontology
- Swissprot
- GAF

Gene-Ontology discussed in section 1.1.2. Which is used as the labels.Swissprot, is a variant of the Uniprot sequences where the sequences are filtered and annotated manually by experts. Due to that this source is preferred due to it allowing for a better data quality. Gene Association Files (GAF) are tab-delimited plain text files, where each line in the file represents a single association between a gene product and a GO term, with an evidence code, the reference to support the link between them, and other information, which is comprised of 17 tab-delimited fields. This data is used to correctly assign the corresponding sequences to the correct sequences and acts as the ground truth.

The input of the model is the Amino Acid (AA) sequence of a protein. Each protein is a character sequence composed of 20 unique AA codes. Inputs are represented through a matrix where the columns correspond to the amino acid and the rows to the position of the amino acid in the protein sequence, also known as a feature vector. As presented in Figure 3.3. This



0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 3.2: An example of the data representation

representation is chosen as it allows for a naive representation of the data which also represents relations between the bases.

3.2 Model Structure

In this thesis, I'm using a variant of the LSTM to be able to predict the GO classes, which would allow us to predict the protein function. Unlike the standard LSTM as visualised in Figure1.5, a modified version is applied where there are forward connections to cell input and recurrent connections to input and output gate only. Furthermore for the output a linear activation is used and then a tanh to see the performance based on this change in the output activation. The goal is to further reiterate the importance of the tanh output to the LSTM structure. The structure also does not have a forget gate as the model is not complex enough where the usage of a forget gate would be needed.

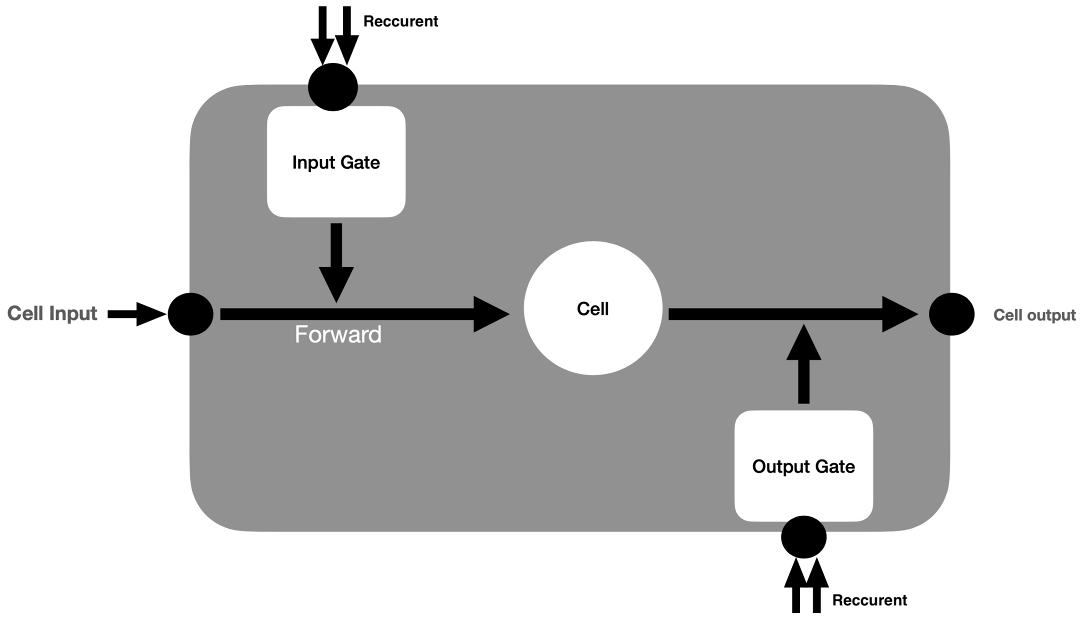


Figure 3.3: General model structure

Furthermore the specific structure was chosen as the input to the model is "naive" where the input is very simple. Next the forward connection for the cell input would allow for the model to look at each sample separately. When each sample is seen separately it would reduce the noise in the data, stopping the errors to flow through. Which would make the predictions better and in theory improve the performance.

3.3 Experiment and setup

In this section, the two main experiments of this thesis are discussed. First, the binary experiment is presented where a quick overview of the classes is shown and the training procedure is discussed. Second, the auxiliary experiment is discussed which takes the outcome of the first experiment and attempts to utilize the relationships between classes to improve the model performance.

3.3.1 Binary experiment

I ran five binary tasks (for each class separately to determine a baseline) using the model. As previously mentioned the data is very imbalanced where the most number of positive samples a single class has is 1150 compared to the 12000 samples in the data-set. First, I take the class with the most samples and try to find the hyper-parameters which will give the best scores. Second, apply this hyper-parameter on the rest of the classes to determine the base line.

3.3.1.1 Classes

The five classes selected for this experiment are the following: GO:122 (negative regulation of transcription by RNA polymerase II), GO:6357 (regulation of transcription by RNA polymerase II), GO:7165 (signal transduction), GO:7186 (G protein-coupled receptor signaling pathway) and GO:45944 (positive regulation of transcription by RNA polymerase II).

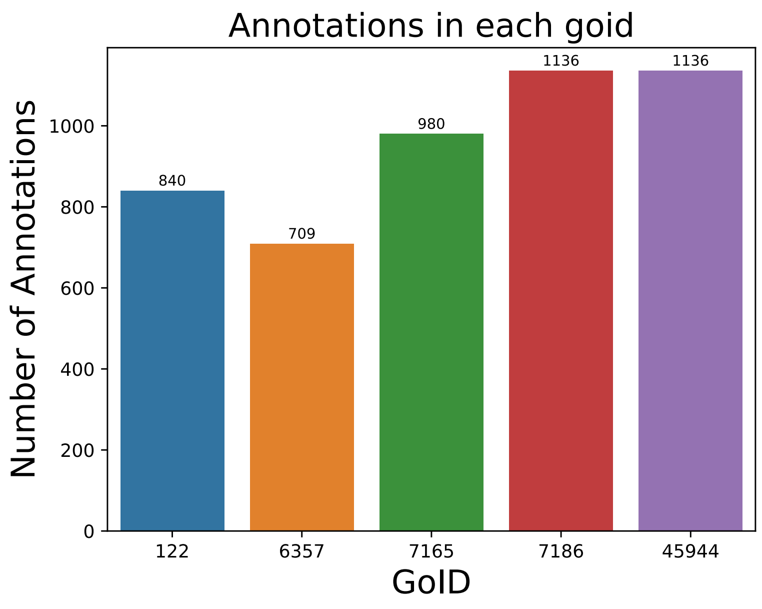


Figure 3.4: Number of positive samples in the data-set for the five classes respectively.

In Figure.3.4, the distribution of unique annotations per class is shown. Where there is equal amount of annotations for class ”45944” and ”7186” followed by ”7165”, ”122” and finally ”6357” with 709 unique annotations.

3.3.1.2 Training procedure

During training, binary cross-entropy with logistic loss was used, which is a combination of sigmoid and binary cross-entropy, with a learning rate of 0.001. This rate is one of the recommended learning rates when using ADAM optimizer [Din15]. I also applied both a linear and tanh activation function separately for the output to observe the effect applying tanh. Furthermore, to minimize the chance of over-fitting, a weight decay of 1e-5 was used. The loss can be described mathematically as:

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right] \tag{3.9}$$

where N is the batch size. This is used for measuring the error of a reconstruction in for example an autoencoder. Note that the labels need to be between 0 and 1. It is possible to trade off recall and precision by adding weights to positive examples. In this case the loss can be described as:

$$\begin{gathered} \ell_c(x, y) = L_c = \{l_{1,c}, \ldots, l_{N,c}\}^\top, \\ l_{n,c} = -w_{n,c} \left[p_c y_{n,c} \cdot \log \sigma(x_{n,c}) + (1 - y_{n,c}) \cdot \log(1 - \sigma(x_{n,c})) \right] \end{gathered} \tag{3.10}$$

where c is the class number ($c > 1$ for multi-label binary classification, $c = 1$ for single-label binary classification),n is the number of the sample in the batch, w_n is the weight, p_c is the weight of the positive sample for the class c .

For example, if a data set contains 100 positive and 300 negative examples of a single class, then the positional weight[12] for the class should be equal to $\frac{300}{100} = 3$.

The loss would act as if the data set contains $3 \times 100 = 300$ positive examples.To reduce the possibility of the model over-fitting, I have included two regularization methods, early stopping[13] and weight decay (method used to measure the model complexity) to reduce the possibility of over-fitting in the model as the aim is to have a model that should work well on unknown data.

[12]The weight assigned to each class, commonly also seen as the probability of the class in the data set.

[13]one continually evaluates the model performance on the validation set. As soon as the validation error becomes significantly worse than the training error, we stop training.

3.3.2 Auxiliary experiment

I ran an auxiliary task where the main class (GO:45944) is being predicted while using the remaining classes as auxiliary classes[14] using the model. As previously mentioned, the data is very imbalanced, where the most number of positive samples a single class has is 1150 compared to the 19,257 samples in the data-set. I selected the GO-ID 45944 as the main class and then used the remaining classes as auxiliary classes, which should boost the Area Under the Receiver Operating characteristic Curve (AUROC).

3.3.2.1 Training procedure

Both mean squared error (MSE) and binary cross entropy with logistic loss (BCELL) are being applied separately, to see their effect on the model performance. Mean Squared error is one of the first loss function normally used which is simply the following:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{3.11}$$

where MSE is the mean squared error, n is the number of data points Y_i are the observed values and $\hat{Y}_i$ are the predicted values.
To make sure that the model prioritizes the main class (GO-ID 45944), the following custom loss function is implemented:

$$L_{total} = l_{main} + w_a * \sum_{i}^{N} \frac{1}{N} * l_{aux_i} \tag{3.12}$$

where l_{main} is the loss for the main class, w_a is a coefficient between 0 and 1 and l_{aux_i} is the loss for the given auxiliary class.

3.4 Resources

For this thesis, I used Intel i7 7th generation CPU for pre-processing and training of the model. For heavier processes including a hyper-parameter search to be able to train the large models, I used both the Cobra and Raptor servers 20 Cores (Xeon(R) CPU E5-2660 v3 @ 2.60GHz) provided by the Institute for Machine Learning located in Johannes Kepler Universitat Linz.

[14]These classes should in theory allow us to improve the performance as they would add more information which the model could use to improve its classification capabilities .

3.5 Tools

The model was implemented in the popular ML framework Pytorch [Pas19]. This framework was chosen due to its ability of creating prototypes easily. Furthermore other libraries like scikit-learn [Ped11] and numpy [Har20] were used during the pre-processing and post-post-processing steps of the experiments.

3.6 Hyper-parameter

Table 3.1 shows the different hyper-parameter sets which were chosen. Each set was chosen based on preliminary experiments and observations from other papers and lectures in the area of ML and DL. The three main parameters that get changed are the batch size, number of LSTM blocks and the optimzer. Batch size determines the amount of samples that the model sees during training per iteration. Number LSTM Blocks is the number of LSTM that will be used. Optimizer is the optimization algorithm used while training, which was either Adaptive moment estimation(ADAM)][KB14] or Stochastic Gradient Descent(SGD)[Ama93].

Learning rate	Batch size	LSTM Blocks	Optimizer	Weight decay	Final Layer
0.001	32	64	ADAM	0.001	No
0.0001	32	64	SGD	0.000001	No
0.0001	8	32	ADAM	0.01	Yes
0.01	64	64	SGD	0.0001	No
0.001	32	32	ADAM	0.00001	Yes
0.001	32	16	ADAM	0.00001	Yes
0.001	32	32	ADAM	0.00001	No
0.1	64	16	ADAM	0.00001	Yes
0.01	32	32	SGD	0.00001	No
0.01	32	32	SGD	0.00001	No
0.001	32	32	SGD	0.00001	No
0.01	32	32	ADAM	0.00001	No
0.001	64	32	ADAM	0.0001	No
0.01	32	8	ADAM	0.00001	No
0.001	32	8	ADAM	0.00001	No

Table 3.1: Hyper-parameters for the hyper-parameter search.

3.7 Limitations

This thesis does not present a novel method using the LSTM architecture to predict protein functions but rather explore the different variations of the original LSTM and how it performs on this particular task. To my knowledge, there has been extensive studies regarding protein function prediction or prediction of the biological process using DNA sequence however not many using only amino acid sequences[KKH18]. Furthermore the sample data is limited to one species due to time and resource constrains. Finally the hyper-parameter search is limited to a specific range to keep the size of the model small enough, to be trained more easily.

4 Results & Discussion

In this section, three groups of results are presented. First, initial results are presented, which are running the model with a randomly selected hyper-parameter set. The goal was to assess the general performance of the model and determine the maximum iterations. Also, the effect of the two different activation functions is presented. Second, the results of the hyper-parameter search and the baseline of the auxiliary classes are shown. Finally, the auxiliary experiment results are presented to show the effect of auxiliary classes on the model performance.

4.1 Initial

The model was run with a basic hyper-parameter set, which are normally associated with LSTMs. The goal was to determine if the model is already better than the previous model discussed in [Sha19] with this hyper-parameter set. The model was run for 200000 iterations, eight LSTM Blocks, batch size of 32, a learning rate of 0.001, ADAM optimizer with a 60, 20, 20 split of the data and using the two top classes to simulate a multi-task and label task presented in [Sha19].

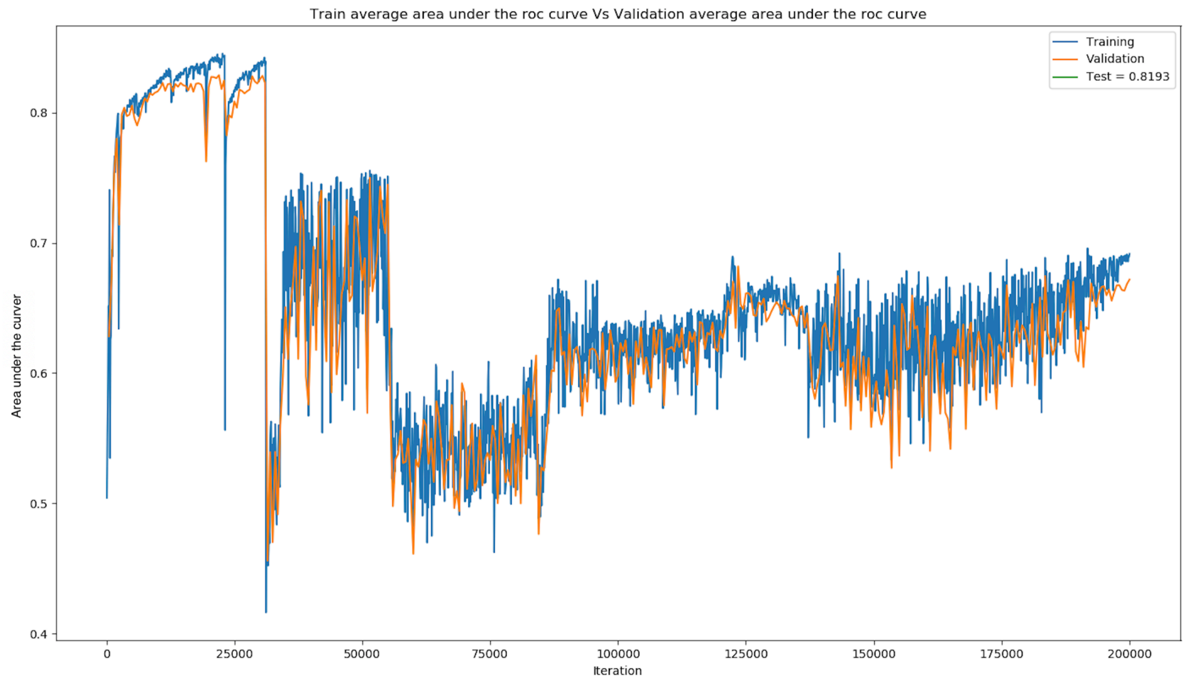


Figure 4.1: The progression of area under the curve as the model learns.

Figure 4.1 shows the progression of area under the curve as the model learns. The blue line indicates the values for the average training set and the orange line indicates the average values of the validation set. We also have the score on the test set in the legend of the diagram which is 0.819.

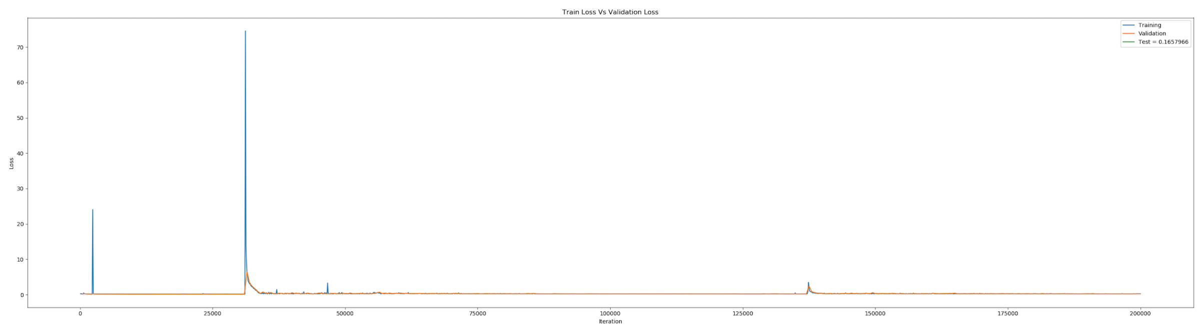


Figure 4.2: The progression of loss as the model learns.

Figure 4.2 shows the progression of loss as the model learns. The blue line indicates the values for the training set and the orange line indicates the values of the validation set. We also have the score on the test set in the legend of the diagram which is 0.12.

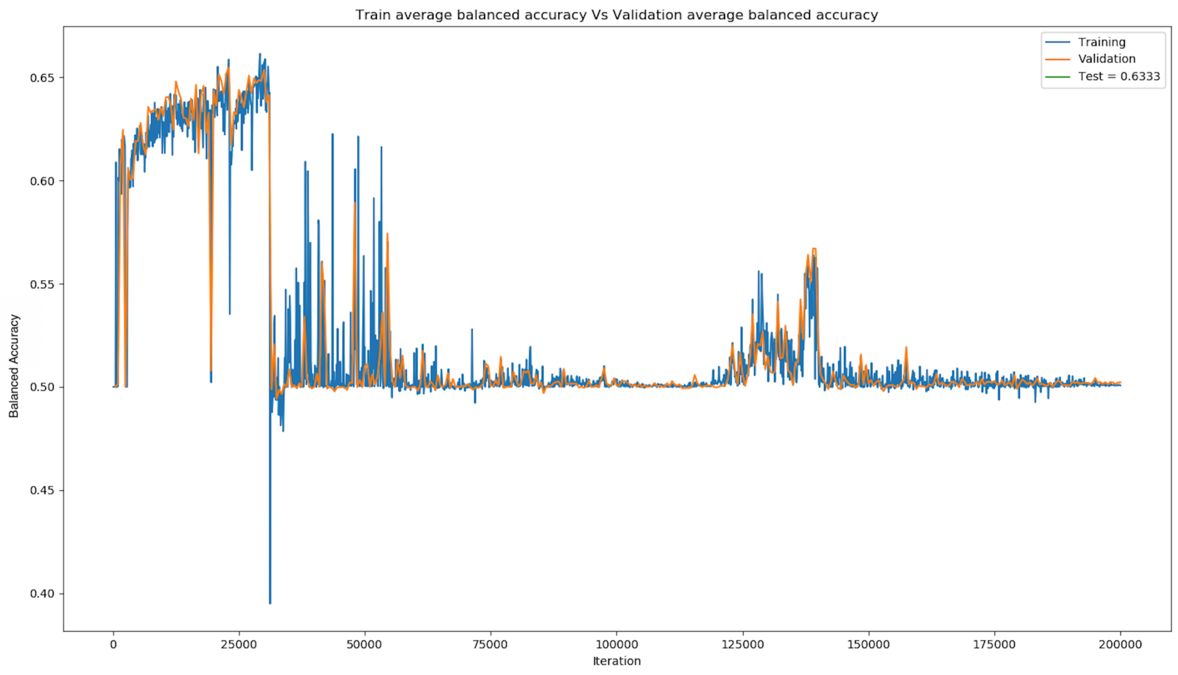


Figure 4.3: The progression of balanced accuracy as the model learns.

Figure.4.3 shows the progression of balanced accuracy as the model learns. The blue line indicates the values for the training set and the orange line indicates the values of the validation set. We also have the score on the test set in the legend of the diagram which is 0.62.

From Figures 4.1, 4.2 and 4.3, it can be observed that the model starts to overfit quickly after approximately 25000 iterations. This indicates that the model after this point stops to learn and starts to memorize the data. This is frowned upon as it may result to generalization errors. Furthermore, it can be seen that both the training and validation set exhibit a similar increasing and decreasing trend as the model iterates through the samples, which may indicate that for this run the validation and training set had samples with very similar sequences.

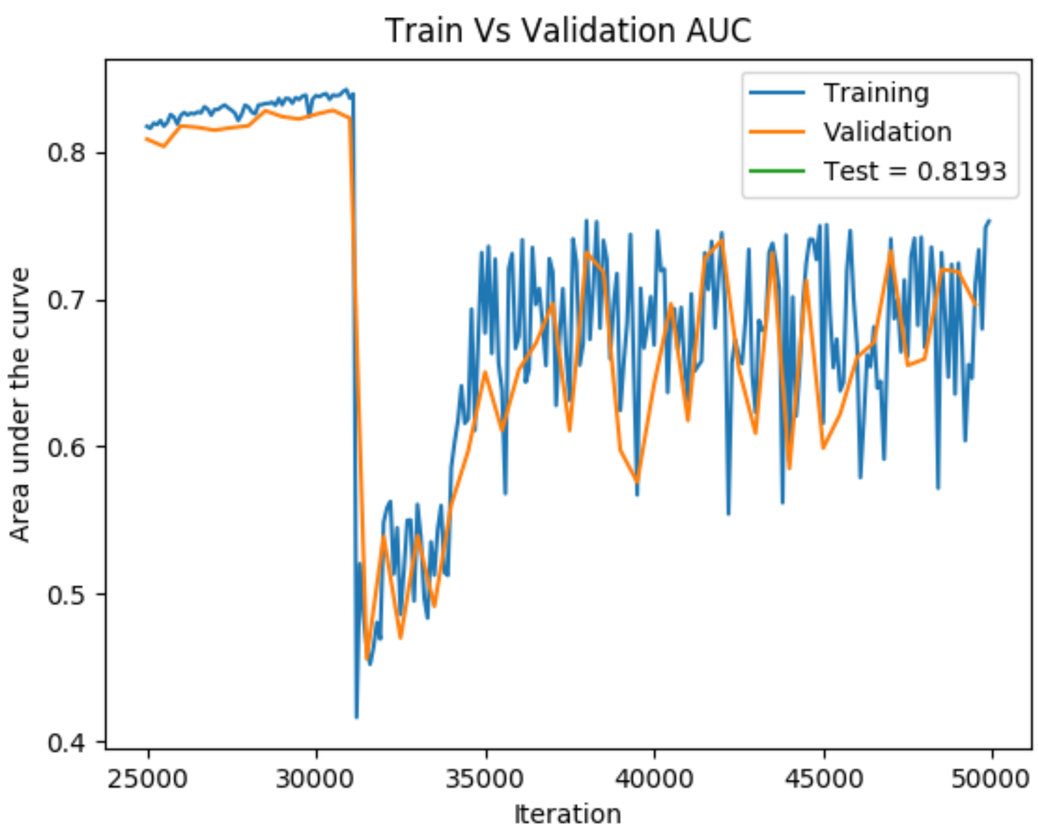


Figure 4.4: A zoomed in version of Figure 4.1, where the drop in auc is shown

As already stated above and shown in Figures 4.4, 4.5 and 4.6, which are enlarged versions of Figures 4.1, 4.3 and 4.2 the optimum number of iterations seems to be 25000, however to make sure that this is not specific to this run a max iterations of 40000 is set for the remaining experiments.

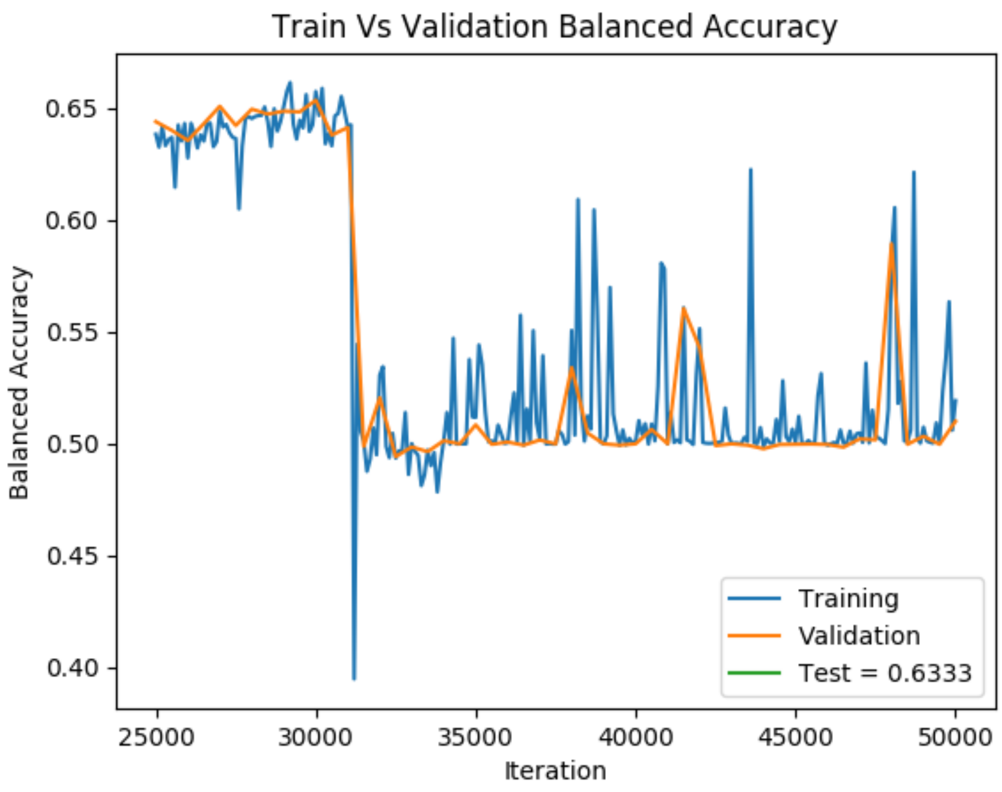


Figure 4.5: A zoomed in version of Figure 4.3, where the drop of the balanced accuracy is shown.

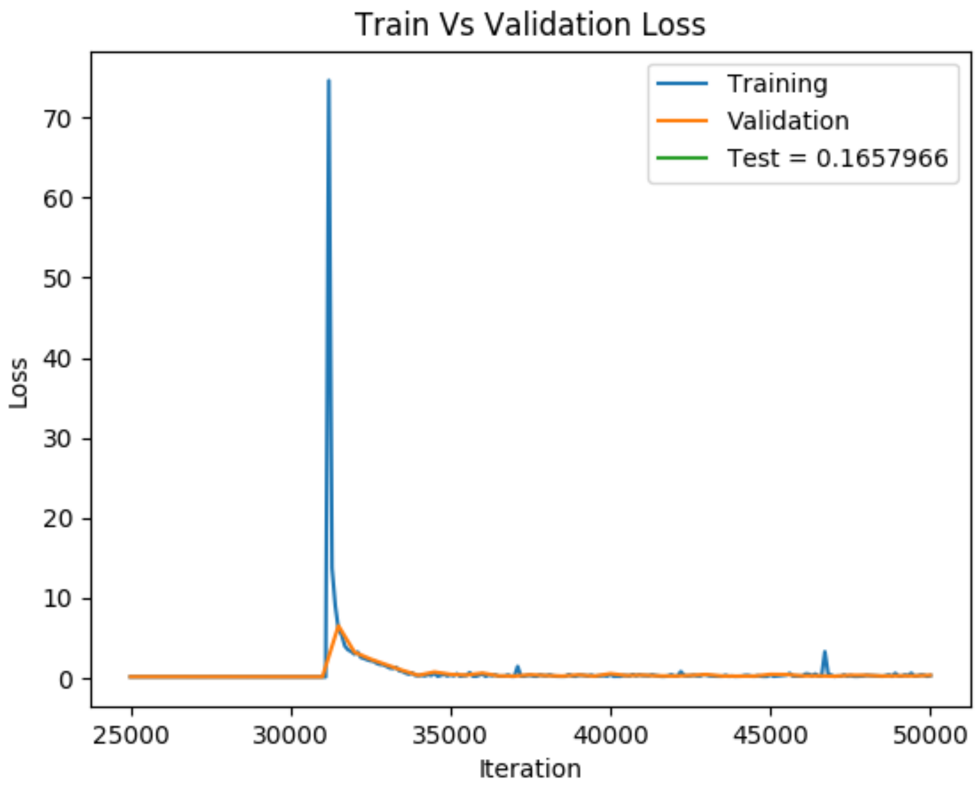


Figure 4.6: A zoomed in version of Figure 4.2, where the increase of the loss is shown.

4.2 Linear vs Tanh output function

In this subsection, a sample run with two different outputs activation functions is presented. The performance metrics from each of the activations are presented and then compared. First, the results using a linear activation function are presented, followed by the same run results, with a tanh activation function for the output.

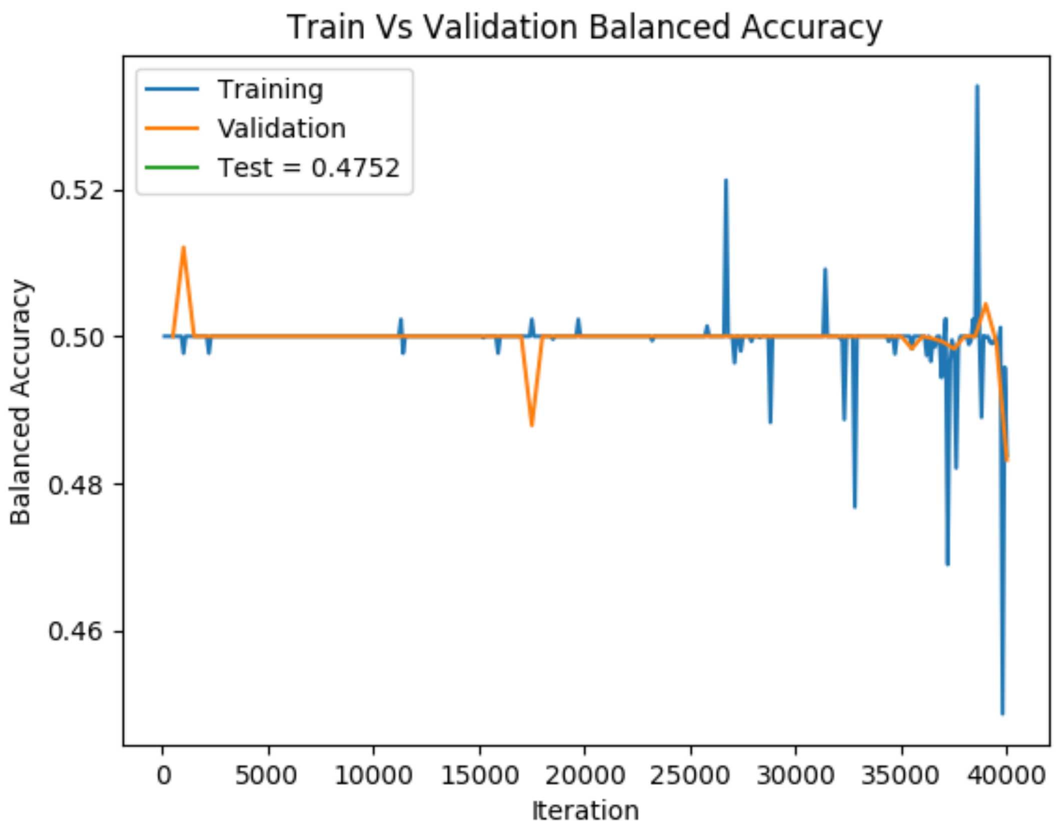


Figure 4.7: Balanced accuracy using linear.

Figure 4.7 shows the balanced accuracy as the model learns using a linear activation. The model scored on the test set 0.4752 as shown in the legend. It can also be seen that the training starts to variate greatly after it reaches the 25000 iterations mark.

Figure 4.8 shows the loss as the model learns using a linear activation. The model achieved an loss of 0.6387885 on the test set as shown in the legend.

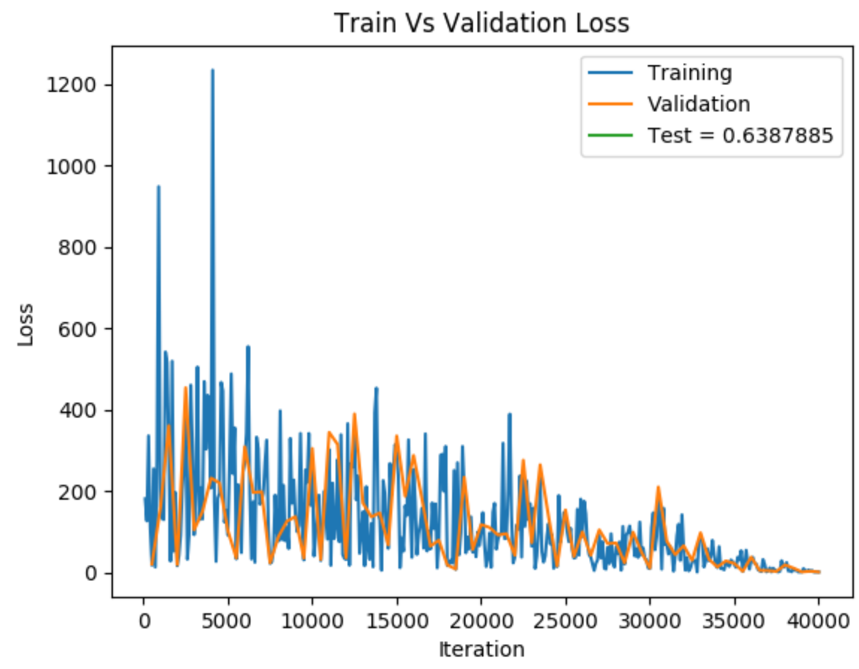


Figure 4.8: Loss using a linear output activation.

Figure 4.9 shows the area under the curve as the model learns using a linear activation. The model achieved an AUC score of 0.5549 on the test set as shown in the legen.

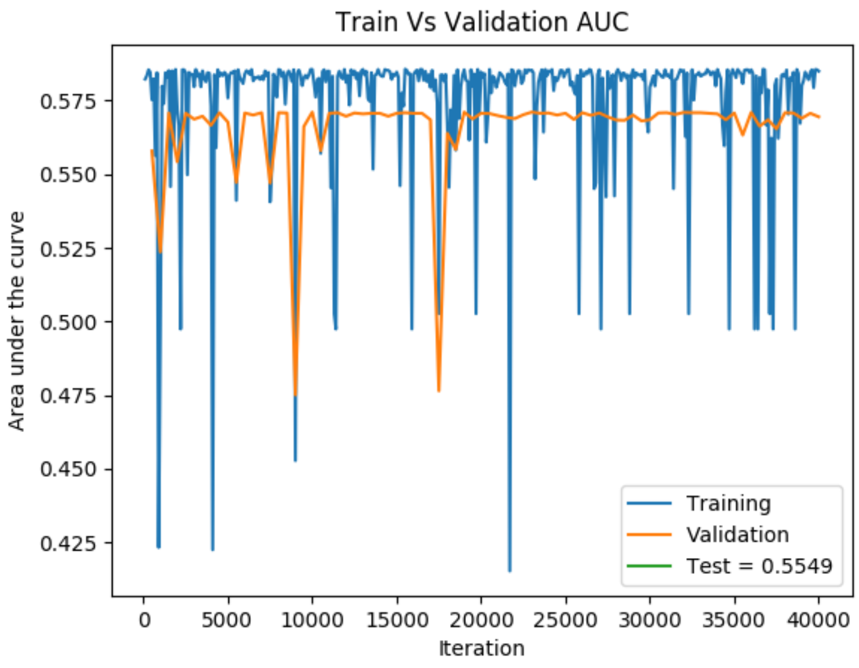


Figure 4.9: AUC using linear output activation.

Figure 4.10 shows the balanced accuracy as the model learns using a tanh activation. The model achieved an balanced accuracy score 0.727 as shown in the legend.

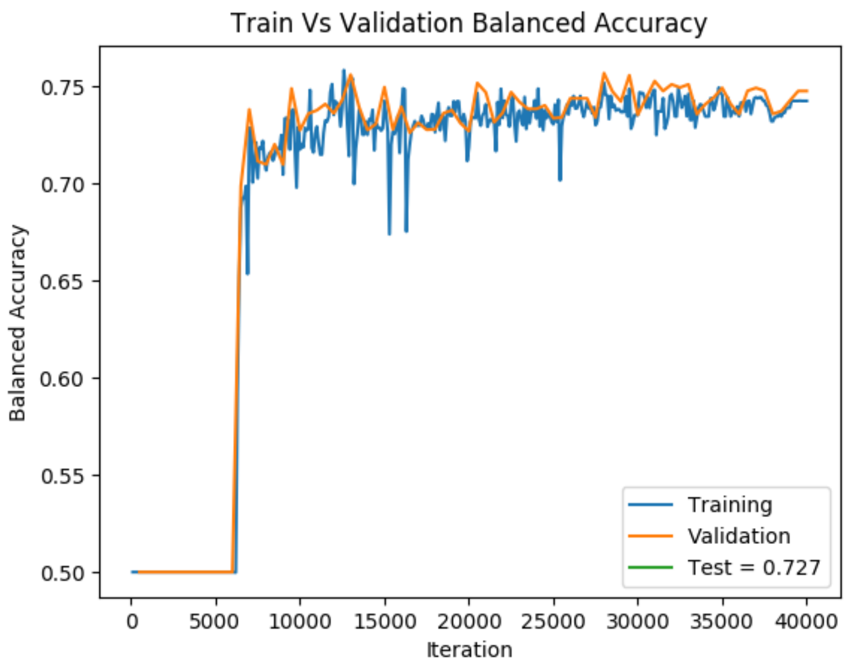


Figure 4.10: Balanced accuracy using tanh output activation

Figure 4.11 shows the loss as the model learns using a tanh activation. The model achieved a loss of 0.1503 on the test set as shown in the legend.

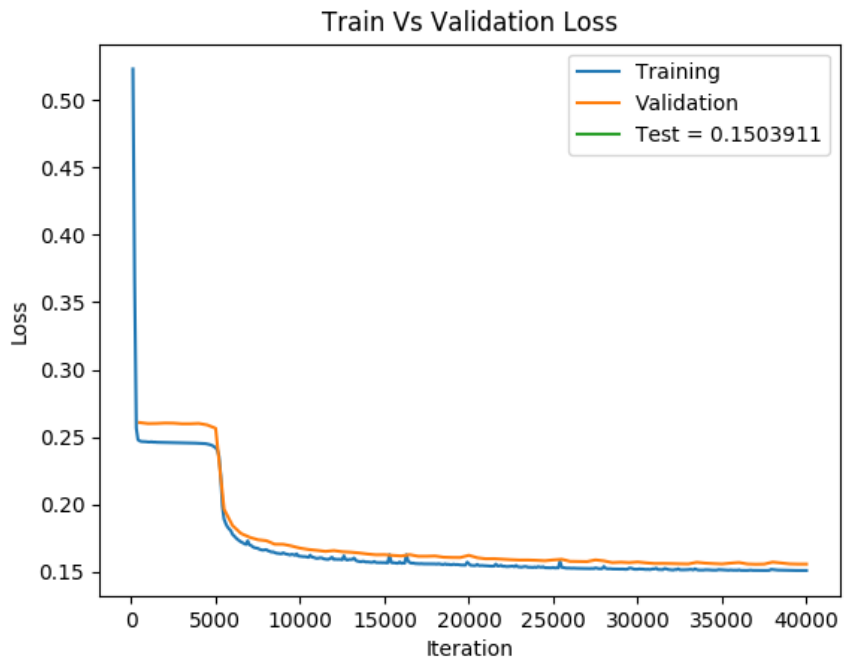


Figure 4.11: Loss with tanh using output activation

Figure 4.12 shows the area under the curve as the model learns using a tanh activation. The model achieved an AUC score of 0.1503 on the test set as shown in the legend.

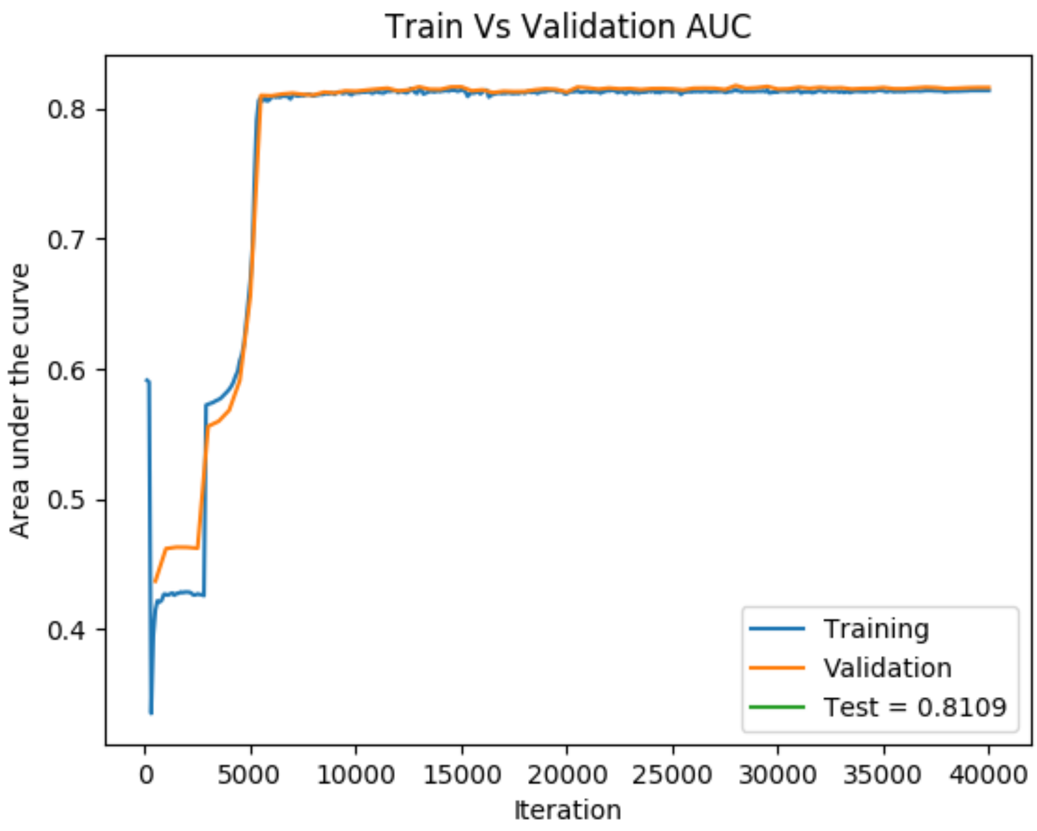


Figure 4.12: AUC with a tanh activation output activation

When replacing the output activation function from linear to tanh, the test AUC has increased greatly. Furthermore looking at the gate values illustrated in Figures 4.10, 4.11 and 4.12, the effectiveness of the tanh activation function is further illustrated. Next, it can also be stated that using the tanh also resulted in a slightly faster training. Finally, when looking at the Figures in Appendix B, it can be observed that the input gate seems to fluctuate less when the output activation is set to tanh. This may be due to the nature of the tanh activation function.

4.3 Binary experiments

4.3.1 Hyper-parameter search result

These are the results of the hyper-parameter search shown in Table 3.1. The results shown in the table are an average of the metrics achieved through running the experiment twice with two different random seeds (42,123).

Hyper-parameter set	Area under the Curve	Loss	Balanced Accuracy
1	0.772	0.197	0.5122
2	0.5089	0.358	0.5128
3	0.6214	0.236	0.499
4	0.5718	3.968	0.4984
5	0.7582	0.198	0.6012
6	0.7749	0.195	0.562
7	0.7730	0.195	0.5094
8	0.5575	0.232	0.499
9	0.5472	0.231	0.5
10	0.4273	0.240	0.5
11	0.7447	0.197	0.5324
12	0.7203	0.216	0.5198
13	0.4382	0.573164	0.5
14	0.438	0.5731	0.5
15	0.8109	0.1503	0.727

Table 4.1: These are the results of the hyper-parameter search.

As shown in the table above, changing parameters like learning rate, batch size, or optimizer affect the model's performance. For the given task with the used data set in this hyper-parameter space, it seems to be that learning rate of 0.001, batch size of 32, eight LSTM blocks, optimizer set to ADAM, an L2 or weight decay of 0.00001, and no extra final feed-forward layer, performed the best.

I should note that this hyper-parameter set was the initial parameter set during the exploration subsection presented in the results section. Furthermore, it can be observed that only one out of the 15 hyper-parameter sets was able to produce good results, which would indicate that a more extensive search is needed to understand why it performed this well fully. Unfortunately, due to lack of time and resources, it was decided not to explore the reason why this phenomenon occurs in the scope of this thesis.

4.3.2 Baseline

Table 4.2 shows the performance metrics achieved by the model when trained on the auxiliary classes separately. This was done to determine a baseline for each class to then better understand the model. Furthermore these baselines are used during the auxiliary experiment

Class	Balanced Accuracy	Area under the curve	Loss
6375	0,502	0,5329	0,552
7186	0,5	0,7479	0,320
7165	0,5	0,649	0,450
122	0,5	0,7419	0,165

Table 4.2: The base performance for each class using the best hyper-paramters for class ”45944” from the hyper-parameter search

to determine how the model learns from these auxiliary classes. Interestingly enough it can be observed that in Table 4.2 the majority of classes are only able to score a maximum of 0.5 which could be due to the imbalanced dataset and also the naive representation. On the other hand, it can be observed that class 7186 and 122 have similar area under the curve scores eventough they have a noticeable difference between the number of annotations. Finally, for loss it can be observed that class 122 seems to have the lowest loss which could be due to the sequences being more unique to other classes. The graphical representation of the data in Table 4.2 can be seen in Appendix C.

4.4 Auxiliary experiments

For this thesis, the performance of the model in an auxiliary task was tested. The results are the performance metrics for the main class. It was noted that while training class 7186 and 122 had an increase in the area under the curve and balanced accuracy, compared to other classes. In Table 4.3 the performance of the model on an auxiliary task is shown. As stated previously, to extend the experiment, two-loss functions were used. As a result, two main observations can be seen.

First, it can be observed that there seems to be a trend that MSE causes the model to achieve better AUC but lower balanced accuracy compared to BCELL, which may indicate that depending on what the primary performance measure is, a specific loss function should be used. As example for auxiliary classes ”7186”,”6357” where there was drop in AUC but a increase in balanced accuracy when using BCELL compared to MSE.

Auxiliary classes	Loss function	Area under the Curve	Loss	Balanced Accuracy
7186	MSE	0,8689	0,0416	0,5479
7186	BCELL	0,8181	0,1866	0,7264
7165	MSE	0,7525	0,1137	0,5499
7165	BCELL	0,6305	0,3835	0,5
6357	MSE	0,7643	0,0710	0,5392
6357	BCELL	0,7431	0,2648	0,5
122	MSE	0,7635	0,0472	0,5479
122	BCELL	0,7380	0,1878	0,5
7186,7165	MSE	0,8544	0,0224	0,554
7186,7165	BCELL	0,8689	0,0789	0,7381
7186,6357	MSE	0,8588	0,0152	0,54629
7186,6357	BCELL	0,8238	0,0481	0,7231
7186,122	MSE	0,8698	0,0109	0,5328
7186,122	BCELL	0,8238	0,0481	0,7231
7186,7165, 6357	MSE	0,8616	0,133	0,5546
7186,7165, 6357	BCELL	0,8172	0,4853	0,7127
7186,7165, 122	MSE	0,8468	0,1144	0,5175
7186,7165, 122	BCELL	0,8140	0,4183	0,7151
ALL	MSE	0,7547	0,1249	0,5492
ALL	BCELL	0,496	0,5051	0,5

Table 4.3: The metrics for the main class depending on the auxiliary classes and what loss was used. These results are while using the best performing parameters in the binary task shown in Table 4.1.

The second observation is that adding more auxiliary classes does not always result in better performance in the main class. As supported by the results in Table 4.3. Next, it was observed that given auxiliary class combinations did perform better than others. For example, when class ”7186” was used as an auxiliary class, the model performed the best out of the rest. Even though the remaining classes are part of the same biological process as the main class (RNA), this could be due to the overlap in samples between classes 49544 and 7186, which has significantly contributed to the results observed above. Finally, even though impressive, these results indicate that for an auxiliary task similar to the one performed in this thesis, a less naive representation could highlight the relationships as an example, protein-protein interaction [KKH18].

5 Conclusion

LSTMs were used for protein sequences and showed great potential in this task, and the method was taken further by evaluating the model's performance based on the addition of auxiliary classes.

The main difficulty of changing from DNA sequences to protein sequences is that there are more unique characters in the sequence, making it an immense task (4 vs. 22 unique symbols), which has many aspects that need to be considered. There have been papers that attempted to solve the protein function prediction from amino acid sequences. However, they use complex models and specific annotations (DeepGo, Deeploc).

In this thesis, an extensive hyper-parameter search was performed, and notable results were presented and discussed in their respective sections. The best hyperparameter set with the given parameter space was found. Mean squared error and Binary cross-entropy with logistic loss were used to investigate what metrics are optimized for auxiliary tasks.

The experiment showed that MSE achieves a better AUC in most cases, where binary cross-entropy achieves a better-balanced accuracy for auxiliary classification. The thesis brought forth many areas of improvements and parameters that can be further studied to improve the model, making the model more applicable to real-world data (the balanced accuracy is still not good enough for being used in labs).

The thesis further proved tanh is the best activation function for this architecture in this given space. It can be agreed that a more extensive data set and better annotation may allow us to improve the model performance. Finally, it was shown deep learning can be used to learn and predict protein functions from protein sequences.Most importantly it was shown that training with auxiliary classes did result to some improvements on the main class. Which shows that further work could be done regarding the application of auxiliary tasks in the area of Bioinformatics.

6 Future Work

From the results, there is still possible further work that can be done. First, an larger hyper-parameter space could improve the model performance as there are still some unexplored hyper-parameter set. Second, a larger multi-species data set may allow for a diversity of the data and improve the data. The addition of extra data similar to [KKH18] and the usage of a more complex representation could, in theory, also improve the performance. Finally, due to the recent popularity of Transformer models, which are based on attention mechanism, attention in this task could also be further explored. In addition, hybrid models have also shown promise, which could also be used for this task [Alm17].

7 Acknowledgement

I want to thank the whole Institute of Machine Learning in Linz and the University of Budweis for their generous help in studying and creating this study program. Furthermore, I would like to thank my parents and friends who helped me push through and always believed me. Finally, I would like to thank my co-supervisor Msc. Michael Widrich always helped me with my problems and provided me with his python library used in this thesis.

References

[Alm17] Almagro Armenteros, J. J. et al.: “DeepLoc: prediction of protein subcellular localization using deep learning”. In: *Bioinformatics (Oxford, England)* 33.21 (2017), pp. 3387–3395.

[Ama93] Amari, S.-i.: “Backpropagation and stochastic gradient descent method”. In: *Neurocomputing* 5.4-5 (1993), pp. 185–196.

[Arr19] Arras, L. et al.: “Explaining and Interpreting LSTMs”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11700 LNCS.2019 (2019), pp. 211–238. arXiv: 1909.12114.

[Bou16] Boutet, E. et al.: “UniProtKB/Swiss-Prot, the Manually Annotated Section of the UniProt KnowledgeBase: How to Use the Entry View”. In: *Plant Bioinformatics: Methods and Protocols*. Ed. by Edwards, D. New York, NY: Springer New York, 2016, pp. 23–54.

[BSF94] Bengio, Y. ; Simard, P., and Frasconi, P.: “Learning Long-Term Dependencies with Gradient Descent is Difficult”. In: *IEEE Transactions on Neural Networks* (1994).

[Din15] Ding, S. et al.: “Extreme learning machine: algorithm, theory and applications”. In: *Artificial Intelligence Review* 44.1 (2015), pp. 103–115. arXiv: 1311.4555.

[Gen04] Gene Ontology Consortium: “The Gene Ontology (GO) database and informatics resource”. In: *Nucleic Acids Research* 32.90001 (2004), pp. 258D–261.

[GSC99] Gers, F. A. ; Schmidhuber, J., and Cummins, F.: “Learning to forget: continual prediction with LSTM”. In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*. Vol. 2. 1999, 850–855 vol.2.

[Har20] Harris, C. R. et al.: “Array programming with NumPy”. In: *Nature* 585.7825 (2020), pp. 357–362.

[HHO] Hochreiter, S. ; Heusel, M., and Obermayer, K. In: *Bioinformatics* 14 (), pp. 1728–1736.

[Hoc91] Hochreiter, S.: “Untersuchungen zu dynamischen neuronalen Netzen. Diploma. Technische Universität München”. In: *Diploma, Technische Universität München* 91.1 (1991).

[HS97] Hochreiter, S. and Schmidhuber, J.: “Long Short-Term Memory”. In: *Neural Computation* (1997).

[Jur17] Jurtz, V. I. et al.: “An introduction to deep learning on biological sequence data: examples and solutions”. In: *Bioinformatics* 33.22 (2017), pp. 3685–3690. eprint: `https://academic.oup.com/bioinformatics/article-pdf/33/22/3685/25167669/btx531.pdf`.

[KB14] Kingma, D. P. and Ba, J.: “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[KKH18] Kulmanov, M. ; Khan, M. A., and Hoehndorf, R.: “DeepGO: Predicting protein functions from sequence and interactions using a deep ontology-aware classifier”. In: *Bioinformatics* 34.4 (2018), pp. 660–668. arXiv: `1705.05919`.

[Kou13] Kouza, M.: “Numerical Simulation of Folding and Unfolding of Proteins”. In: (2013).

[May16] Mayr, A. et al.: “DeepTox: Toxicity prediction using deep learning”. In: *Frontiers in Environmental Science* (2016).

[Pas19] Paszke, A. et al.: “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: NeurIPS (2019). arXiv: `1912.01703`.

[Ped11] Pedregosa, F. et al.: “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[Ran20] Ranjan, A. et al.: “Deep Robust Framework for Protein Function Prediction Using Variable-Length Protein Sequences”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 17.5 (2020), pp. 1648–1659.

[Sha19] Shalaby, F.: *Project report : Protein function prediction through Neural Networks*. Tech. rep. 2019, pp. 1–10.

[The18] The Gene Ontology Consortium: “The Gene Ontology Resource: 20 years and still GOing strong”. In: *Nucleic Acids Research* 47.D1 (2018), pp. D330–D338. eprint: `https://academic.oup.com/nar/article-pdf/47/D1/D330/27437640/gky1055.pdf`.

Appendix A: Code

The code base for this thesis can be found under the following GitHub repository:

https://github.com/fathyshalaby/ONTLSTM

Appendix B: Gate activation's

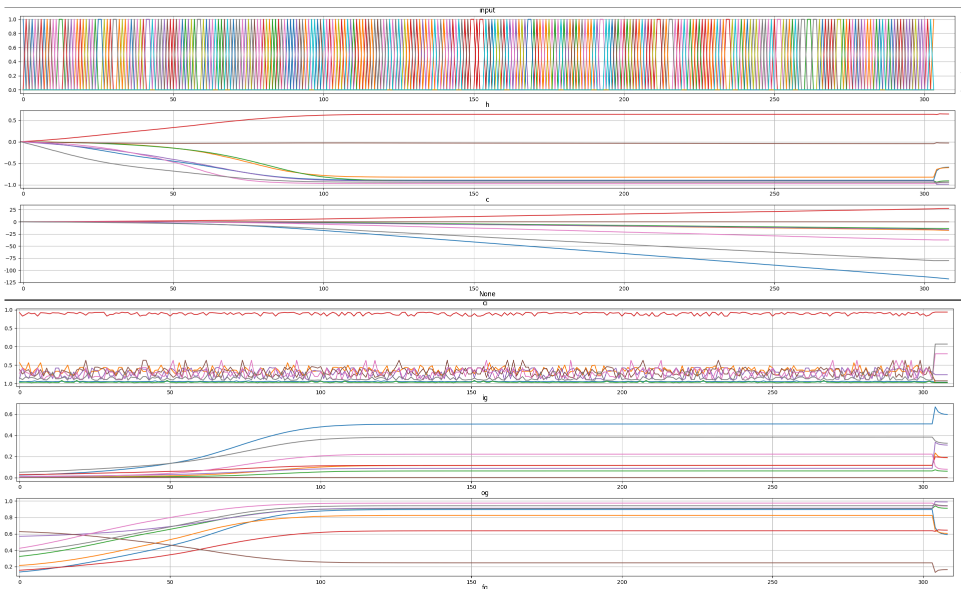


Figure .1: Gate activation at update 1000 for tanh output.

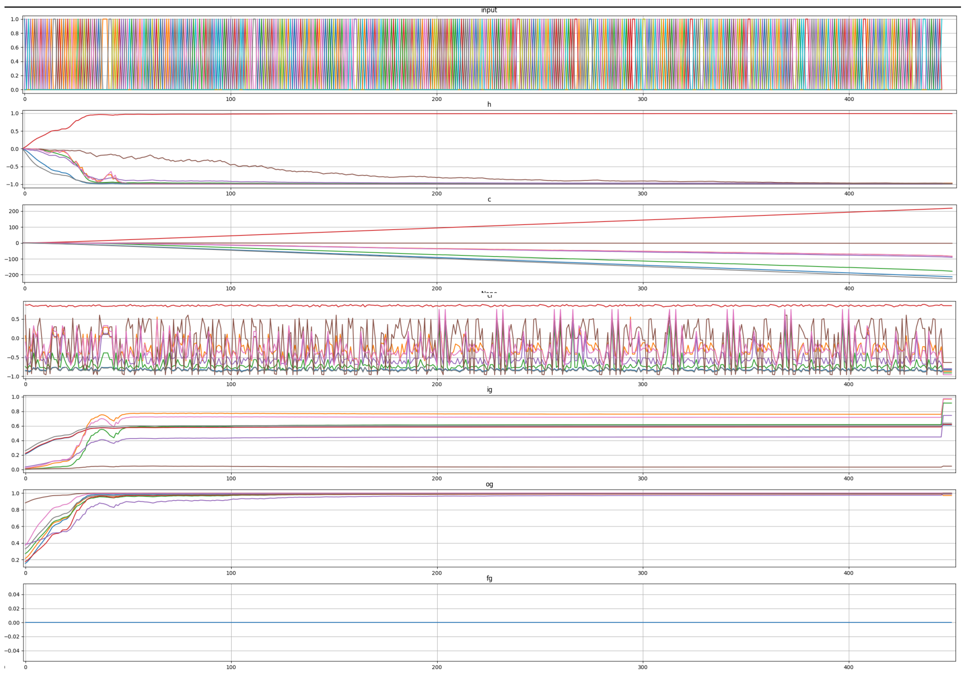


Figure .2: Gate activation at update 40000 for tanh output.

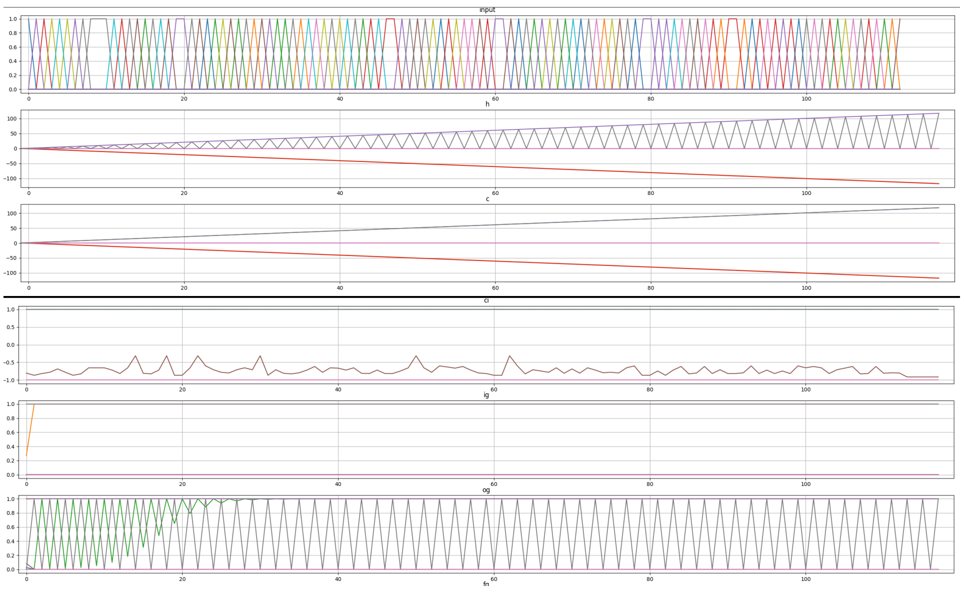


Figure .3: Gate activation at update 1000 for linear output.

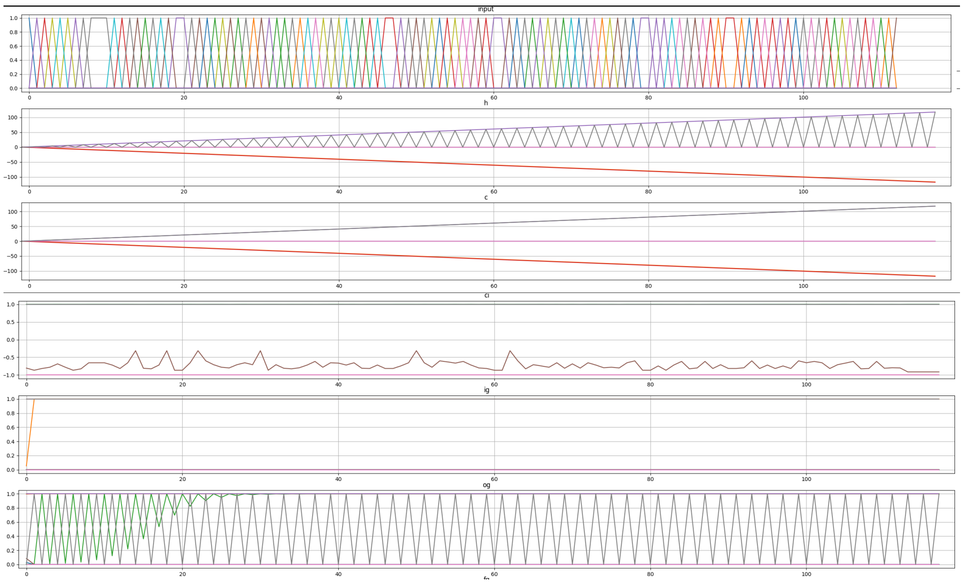


Figure .4: Gate activation at update 40000 for linear output.

Appendix C: Binary task results with tanh

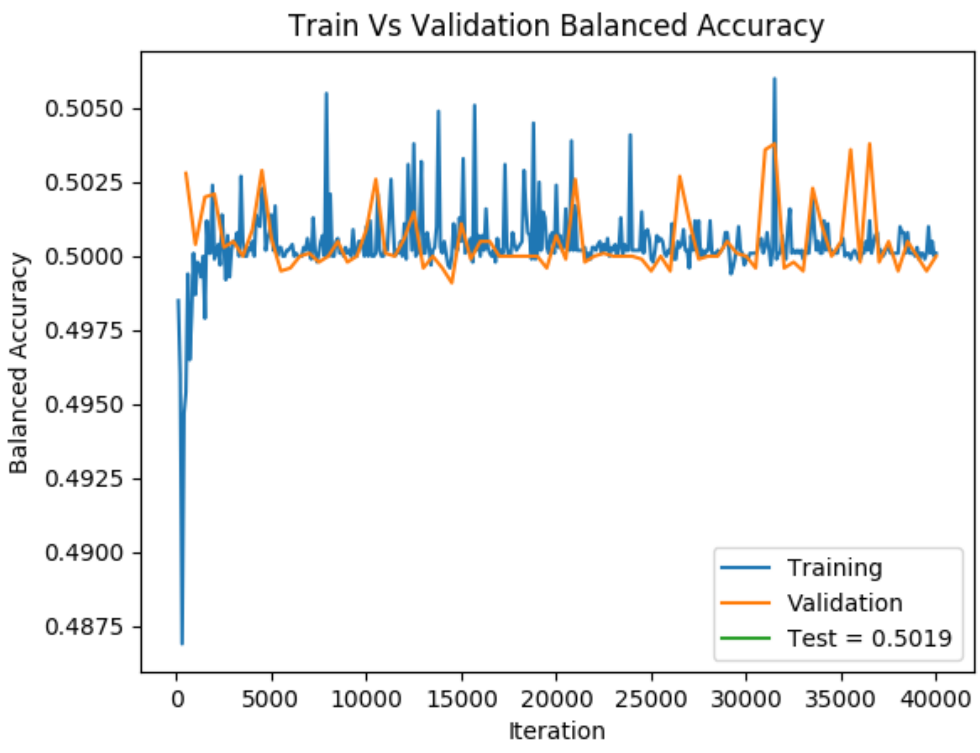


Figure .5: Balanced accuracy throughout training for class 122. Scored on test set 0.5019

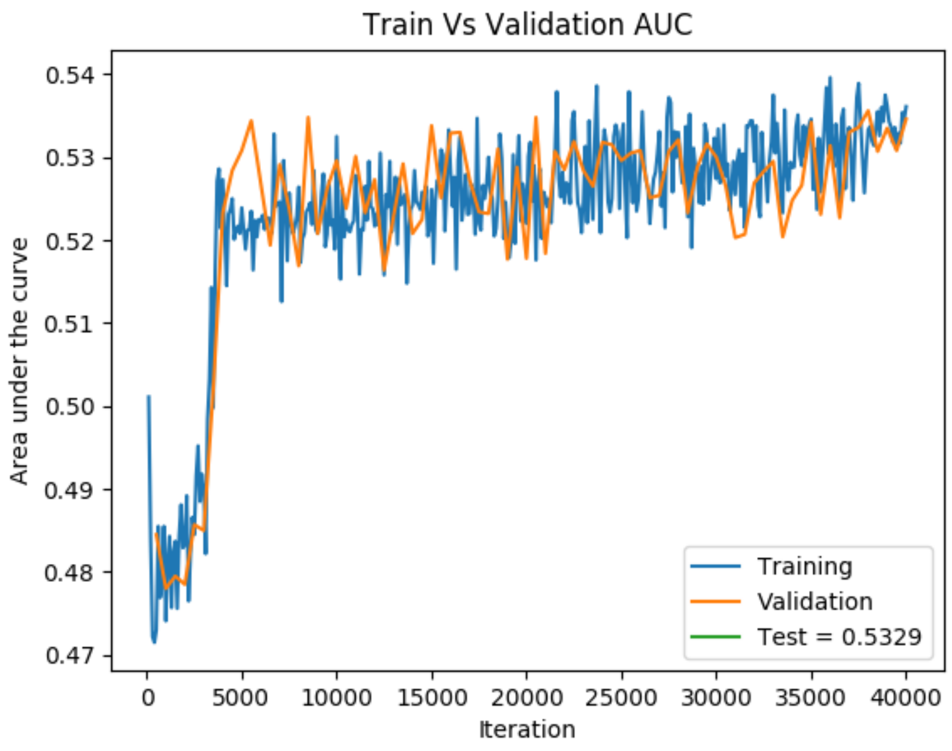


Figure .6: Area under the curve throughout training for class 6375. Scored on test set 0.5329

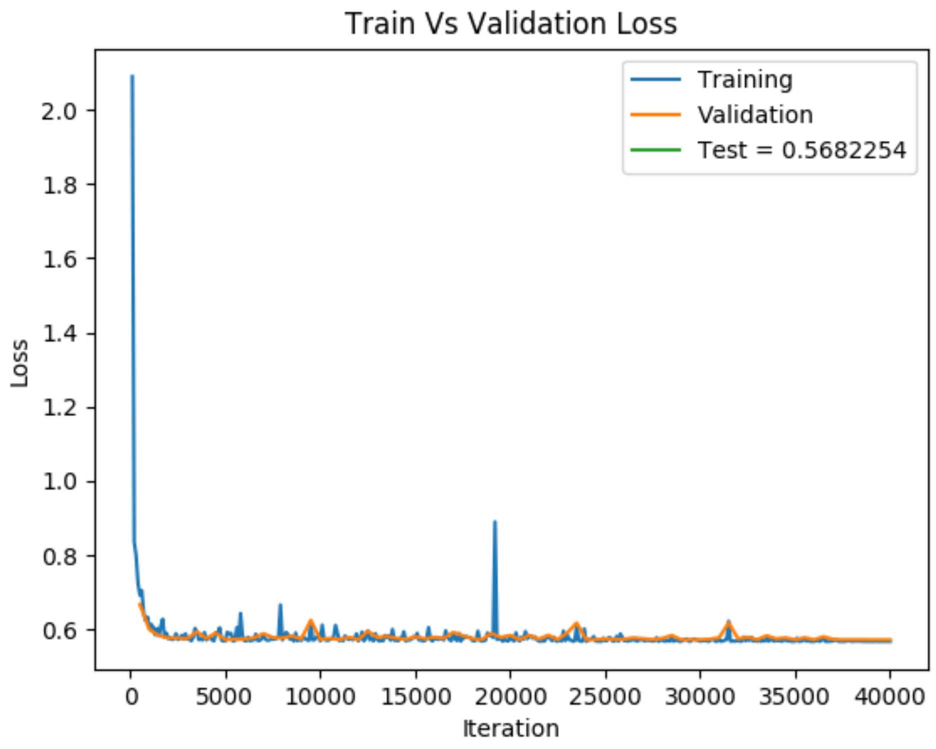


Figure .7: Loss throughout training for class 6375. Scored on test set 0.552254

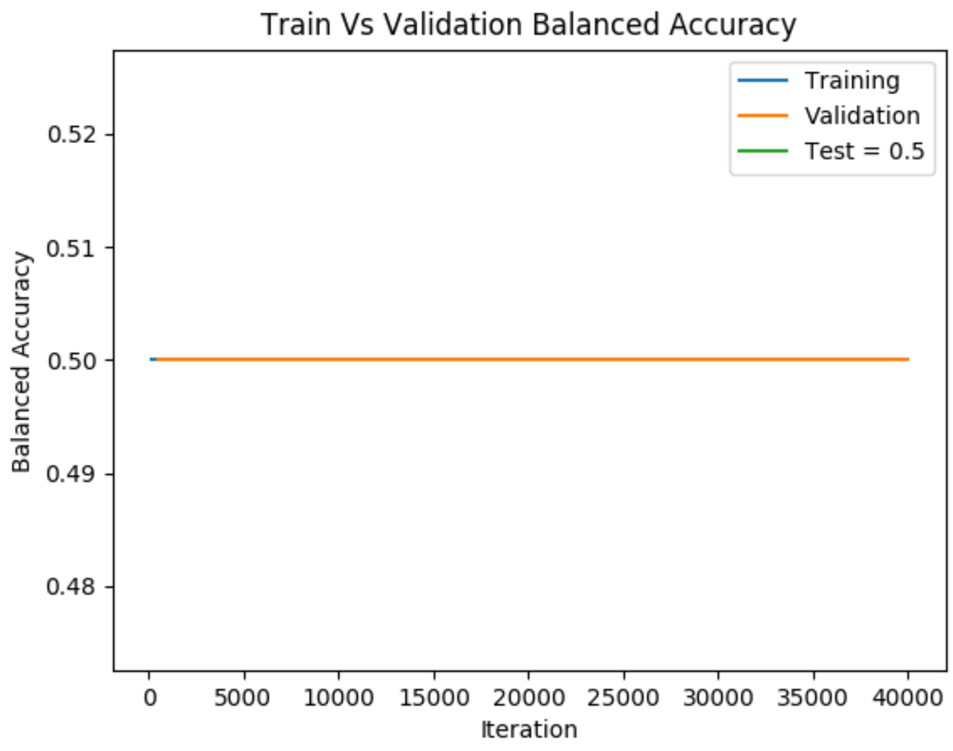


Figure .8: Balanced accuracy throughout training for class 7186. Scored on test set 0.5

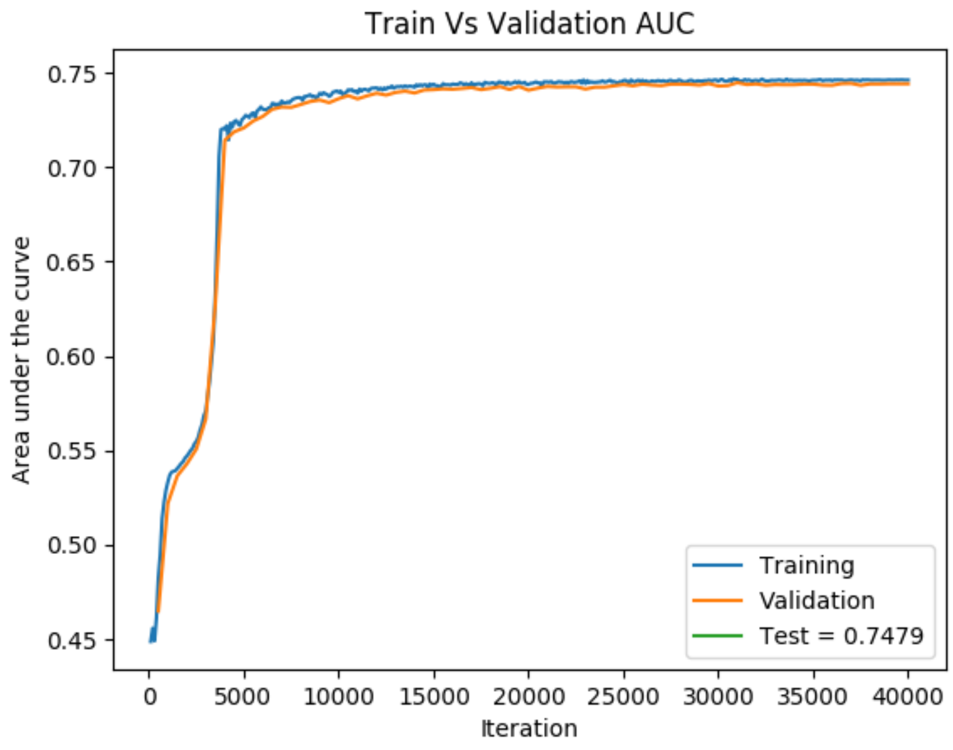


Figure .9: Area under the curve throughout training for class 7186. Scored on test set 0.7479

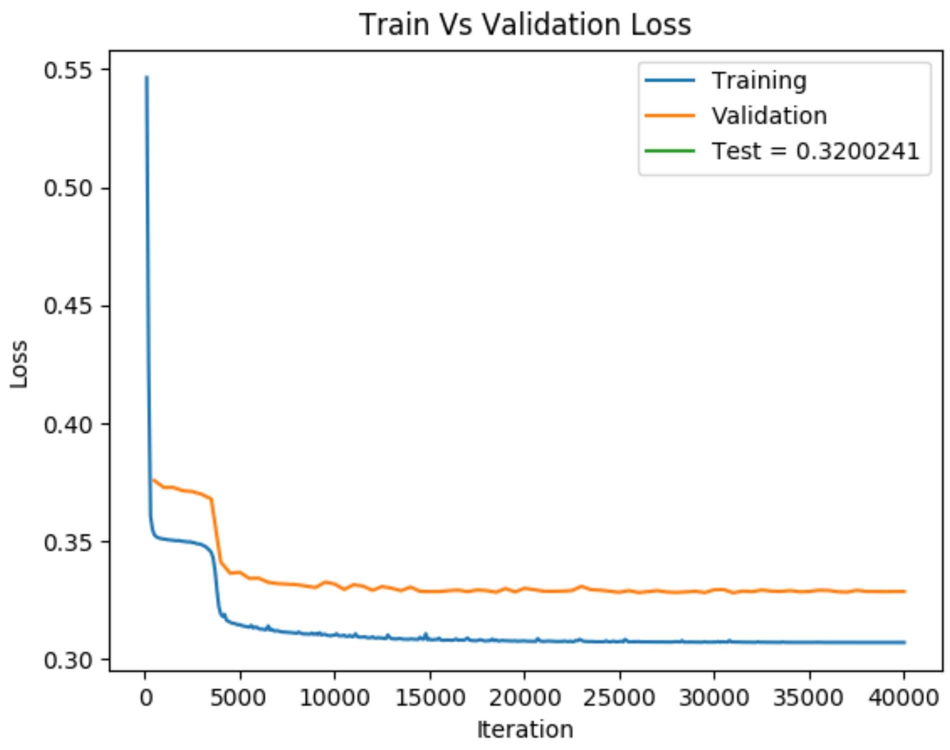


Figure .10: Loss throughout training for class 7186. Scored on test set 0.32

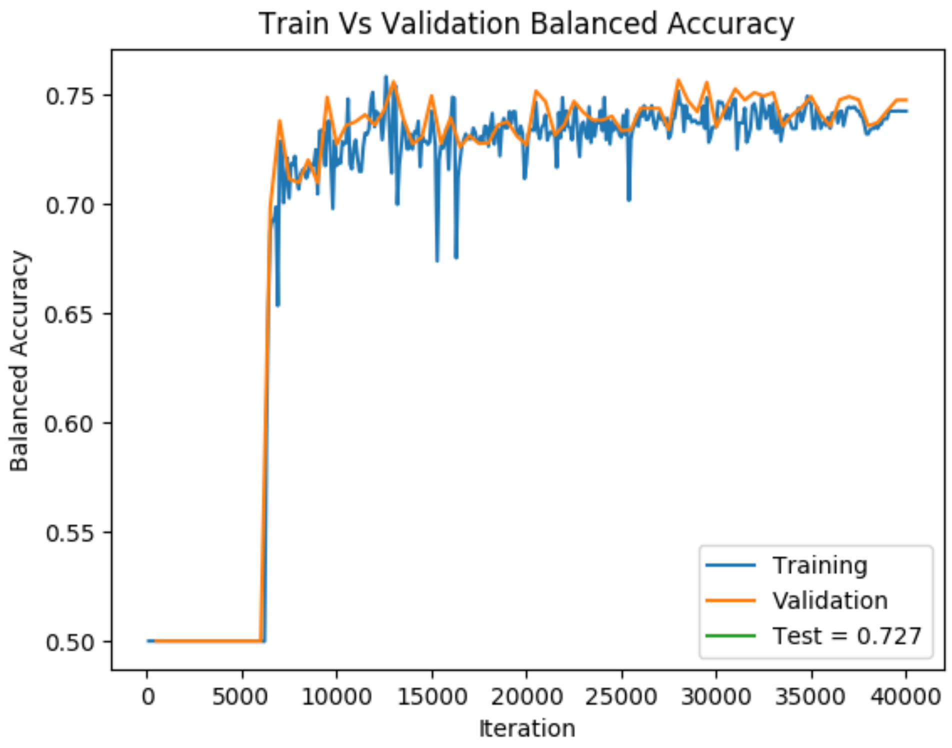


Figure .11: Balanced accuracy throughout training for class 45944. Scored on test set 0.727

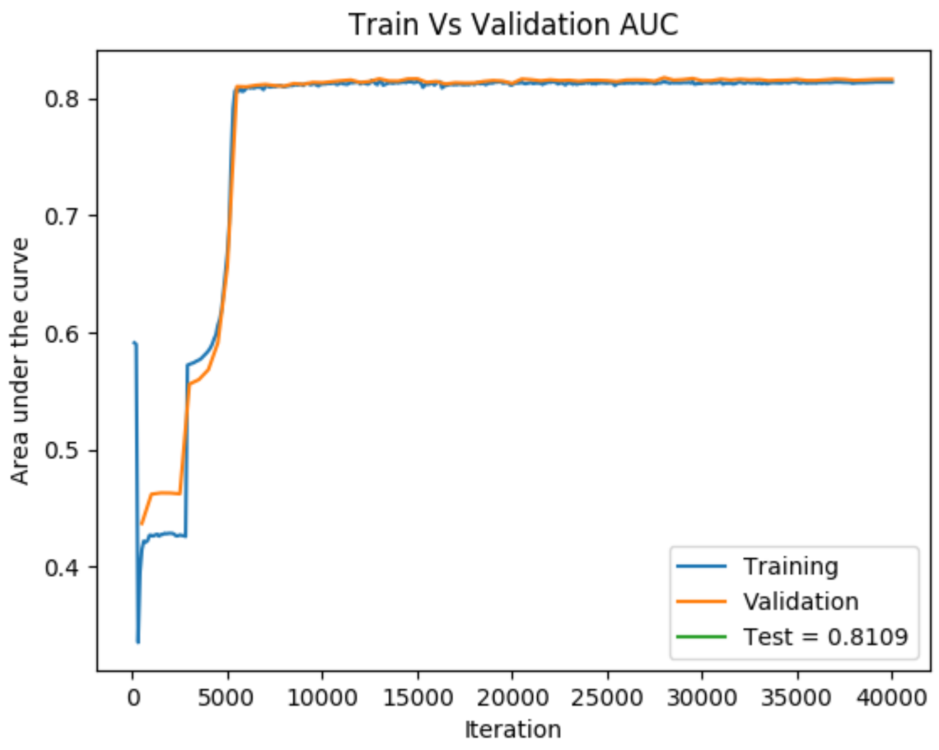


Figure .12: AUC throughout training for class 45944. Scored on test set 0.8109

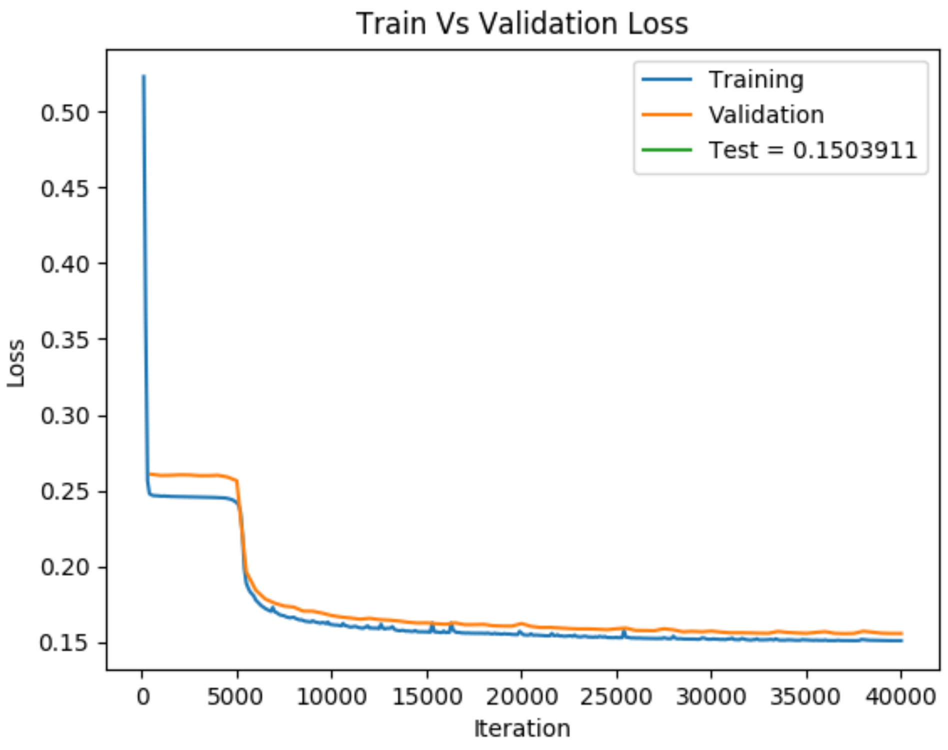


Figure .13: Loss throughout training for class 45944. Scored on test set 0.150

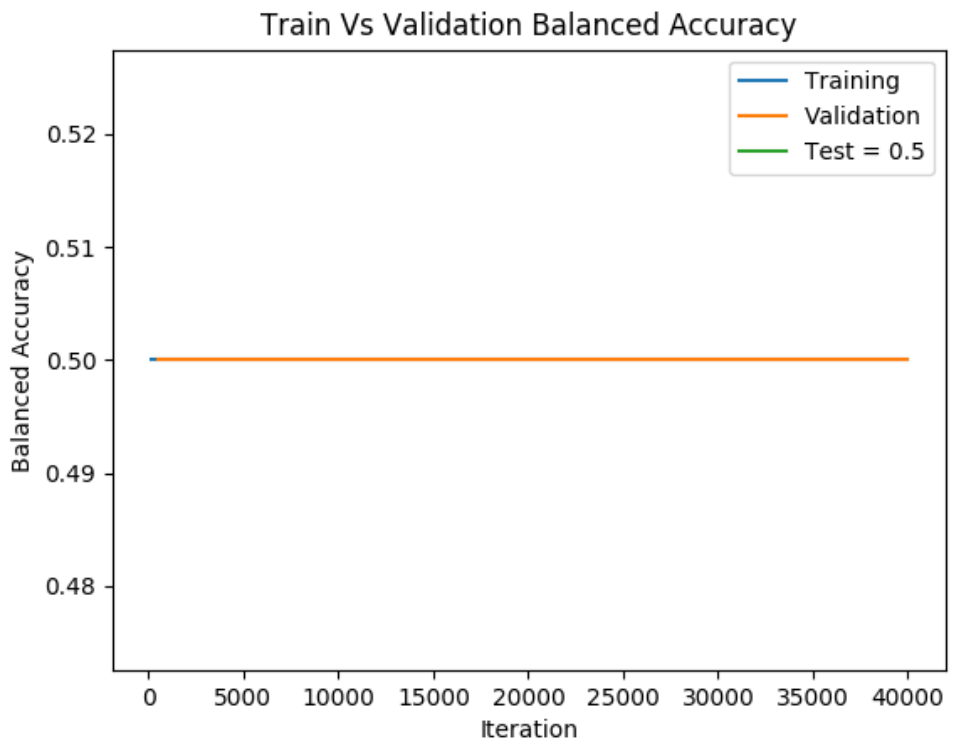


Figure .14: Balanced accuracy throughout training for class 122. Scored on test set 0.5

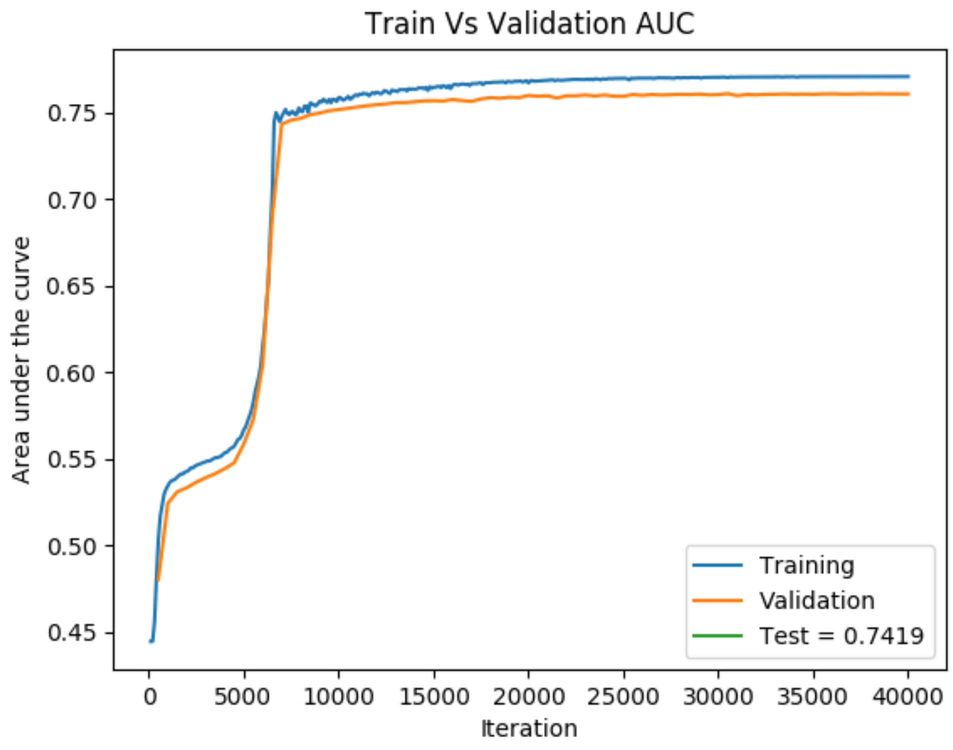


Figure .15: AUC throughout training for class 122. Scored on test set 0.7419

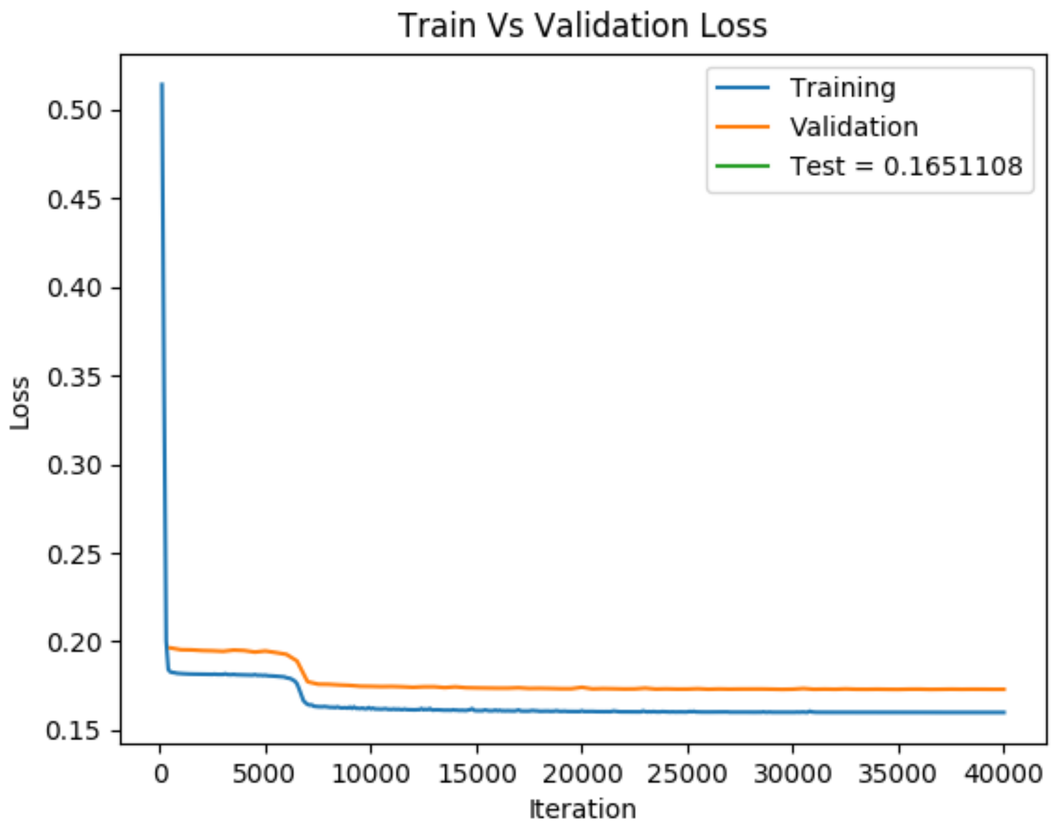


Figure .16: Loss throughout training for class 122. Scored on test set 0.165