Neural Networks for Gene-Gene Interactions

Gul Kamisli
Wolfson College
University of Oxford

A thesis submitted for the degree of
Master of Science
Trinity 2019

This thesis is dedicated
to my sister.

Acknowledgements

I would like to express my immense gratitude to Dr Alejo J. Nevado-Holgado for his enthusiastic, excellent supervision and his constant encouragement throughout the project. His insightful comments and ideas during our one-to-one meetings helped me a lot in finishing my project. I also would like to express my sincere appreciation to Prof. Paul Goldberg for his guidance and support throughout the thesis process.

I would like to profoundly thank my friend, K. Tuncay Tekle, who has always given me a valuable support to shape my career and hand to deal with my problems. I am also grateful to dear Nese Hasanoglu and my dear college friend and his family for their support before my masters' journey.

I am deeply grateful to my parents for the encouragement that they gave me when I was in need. I have to thank love of my life, Tugberk Hasanoglu, for keeping things going, motivating me to get things done and for always showing how proud he is of me.

The last words goes for my sister, who has been an inseparable part of my life for the last two years and who has shown me how to be strong, and resilient.

Abstract

Gene-gene interaction is crucial for gene regulation, a process by which a cell controls what genes are active for transcription process to make a functional protein. In the recent past, several techniques have been applied to gene-gene interaction problem [1, 2, 3]. Due to the increasing number of complex diseases, there exists a need to develop accurate tools so as to identify underlying genetic factors while incorporating Single Nucleotide Polymorphisms and gene expression levels. In this work, we study and construct such tools using Recurrent Neural Networks for the gene-gene interaction problem. Moreover, we also reproduce well-known statistical approaches implemented for gene-gene interactions, namely Expressive Quantitative Trait Loci and Linkage Disequilibrium, and evaluate their performance on our data sets.

The main contributions of the work include: i) new task-based variants of the Long Short-Term Memory [4] architecture which was designed for gene-gene interactions, ii) experiments of variants on AddNeuroMed and Harvard Brain Tissue Resource Centre data sets separately to adjust parameters, iii) experiments of variants so as to ensure that the behaviour of a trained model on one data set is as expected when tested on another data set, and iv) development of MatrixEQTL and Linkage Disequilibrium which can be used to compare our variants' performance with traditional statistical approaches.

Contents

List of Figures

List of Tables

List of Codes

List of Abbreviations

AD Alzheimer's Disease.

ANOVA Analysis of Variance.

DNA Deoxyribonucleic Acid.

EOS End-of-input Symbol.

eQTL Expression Quantitative Trait Loci.

FC Fully Connected.

FDR False Discovery Rate.

GENN Grammatical Evolution Neural Networks.

GWAS Genome-Wide Association Studies.

HBTRC Harvard Brain Tissue Resource Centre.

LD Linkage Disequilibrium.

LOAD Late-On Alzheimer's Disease.

LR Likelihood Ratio.

LSTM Long Short-Term Memory.

MAF Minor Allele Frequency.

MCI Mild Cognitive Impairment.

MLP Multilayer Perceptron.

NCBI National Centre for Biotechnology Information.

NN Neural Network.

RefSeq Reference Sequence.

RNA Ribonucleid Acid.

RNN Recurrent Neural Network.

ROSMAP Religious Orders Study and Memory and Aging Project.

SNP Single Nucleotide Polymorphism.

TF Transcription Factor.

Chapter 1

Introduction: Gene - Gene Interactions

1.1 Background

Deoxyribonucleic Acid (DNA) is a thread-like chain of nucleotides (adenine, thymine, cytosine, and guanine) carrying the genetic information used in building, maintaining, and altering living cells. The information that is encoded in DNA is first transcribed into Ribonucleid Acid (RNA) and then translated into proteins, functional molecules that act as chemo-mechanical nano-machines and form the back-bone of most biological functions in all organisms. Although 25,000 proteins are known in the human organism, which of these are specifically built in each cell type and in each situation is controlled by a few hundreds of Transcription Factor (TF), also proteins themselves which activate or deactivate hundreds of other proteins. TFs are, therefore, vital for the normal development such as cell division, cell growth, and cell death throughout life; otherwise, any abnormal signal by TFs may lead to anomalous cell behaviours, then to diseases.

The investigation of complex diseases such as cancer, or Alzheimer's Disease (AD) is of great importance in genetic epidemiology, the field dedicated to identifying the underlying genetic factors of health and diseases. Some diseases, such as cystic fibrosis, are caused by a single genetic mutation[1], while the vast majority of diseases are rather caused by the interaction of multiple genetic and environmental factors [8]. Genetic architecture (factor) of a disease refers to the number of genetic variants that influence risk of a disease, distribution of their allele[2] frequencies, distribution of their effect sizes and their genetic modes of action (additive, dominant, and/or epistatic[3]). Hence, underlying genetic architecture determines how possible it is to predict genetic risks because it becomes difficult to explain effects of variants as the number of causal variants rises [9].

Research has shown that genes do not function alone; rather, they constantly interact with one another, TF being the main example of gene-gene interactions. These interactions are crucial for gene regulation, numerous physiological and developmental pathways [10, 11]. Therefore, gene-gene interactions play a significant role in the route to complex diseases.

[1] An alteration in the DNA sequence
[2] A variant mode of a gene
[3] A genetic mode that determines if a trait (phenotype) will be expressed

Over the last few decades, a wide range of statistical methods have been developed to identify gene-gene interactions; however, the reliability of these methods is severely limited due to the complexity with which changes in DNA sequence, RNA transcription and protein concentration of source (e.g., a TF) and target genes relate to each other [2]. Large-scale Expression Quantitative Trait Loci (eQTL) analysis and Genome-Wide Association Studies (GWAS) have detected plentiful variants that associate with diverse gene expression[4] levels and human diseases [12, 13]. These so-called variants, or Single Nucleotide Polymorphisms[5] (SNPs), are small mutations in the DNA sequence, which in turn may produce changes in RNA transcription, protein translation or how proteins interact with each other. Any given gene generally has from tens to thousands of SNPs, and methods often analyse how SNPs correlate with phenotypes (e.g., in GWAS studies) or RNA expression (e.g., eQTL) to infer gene function. Most of these variants are located in the non-coding regions of the human genome, indicating that non-coding variants are significant in human disorders by disrupting cis-regulation[6] [14].

One of the well-known successful statistical approaches analysing how genetic variants (e.g., SNPs) within the *cis*-regulatory regions are associated with gene expression levels is eQTL [1]. The authors created all pair-wise combinations of variants, and tested the interactions for significance after identifying variants based on *cis*-regulatory regions; in other words, variants on nearby genes. The method uses a model similar to linear, or logistic regression from Cordell's study [15]. These methods, based on statistical and traditional machine learning approaches, provide a better understanding of gene-gene interactions due to their power in detecting associations described next. In this thesis, we study and improve these methods to better detect the association between genetic loci.

Besides GWAS and eQTLs, which are based on linear models, other more complex methods based on machine learning have been proposed to identify gene-gene interactions, e.g., support-vector machines [16], random forests [17, 18], multi-factor dimensionality reduction [19, 20], combinatorial partitioning methods [21], focused interaction testing framework [22], classification and regression trees [23], logistic regression [24], and lasso regression [25]. Even though these methods are simple to implement and are frequently used in gene-gene interactions, they have not attained significant accuracy gains to predict gene-gene interactions. Additionally, because of

[4] A process by which a genetic information is used to synthesise a functional gene product such as proteins or types of RNA

[5] A substitution of a single nucleotide with another base at a specific position in the genome

[6] Non-coding DNA regions which regulate the transcription of nearby genes

the sheer number of genes, transcripts, proteins and SNPs involved, traditional exhaustive searches are computationally infeasible. Therefore, more sophisticated methods that can cope with the problems listed above and predict gene-gene interactions with higher accuracy are needed. Neural Network (NN) based approaches have been proposed to deal with these issues that are significantly more powerful in prediction ability than aforementioned works based on machine learning. Importantly, Motsinger et al. [3] implemented Grammatical Evolution Neural Networks (GENN) to simulate genome wide association data with 500,000 SNPs. The authors propose a NN that is optimal for each data set to predict the disease-risk loci; however, the method only incorporates SNPs of genes and phenotypes, not gene expression levels. In other words, GENN misses situations where any change in a SNP of a gene alters the expression of another gene, and it is this correlation between SNPs, RNA and proteins that arguably would present the best evidence on how genes regulate other genes. Therefore, we base our methods in this thesis on the statistical methods described above.

1.2 Problem Definition

Gene regulation is the process by which a cell controls what genes in DNA are expressed, "turned on", during transcription in order to make a functional protein. While every cell in our body holds exactly same DNA, each of them has a different set of "active" genes because TFs cause specific genes to turn "on" or "off" by binding to nearby DNA. Gene expression is the process where a gene is used to make a protein when it is "turned on". Simply, gene expression level specifies how much a gene in DNA is active. Proposing a method detecting how genes interact and which genes regulate one another will not only reveal information about protein folding, but also identify disease-susceptibility genes[7].

Producing such a model that aims to find gene-gene interactions is a particularly challenging task. Unlike many methods, we do not have a strong power to detect genetic effects because there are large number of predictors to handle number of samples and correlated predictors in the data set. Hence, we judge model's quality on how reliable the model is for various inputs as well as how accurate in finding pairs. In particular, we use the following two features when developing a robust model:

- **Feature 1 (Reliability):** The model should ideally test the performance on an uncorrelated data.

[7]Increased chance of having a specific disease due to genetic mutations

- **Feature 2 (Accuracy):** Unlike the baseline method, the model aims to detect genes in both nearby and spatially distant locations.

Given this background, we can define the problem which we tackle in this thesis. The problem is essentially three-fold:

1. First, we want to develop a comprehensive Recurrent Neural Network (RNN) that scans SNPs and gene expression levels of gene pairs.

2. Second, we want to evaluate the model's performance on data sets collected from brain tissues and blood samples; then, to compare resulting metric with a state-of-the-art statistical method (i.e., eQTL).

3. Finally, we want to create an attention layer in order to understand how the network performs very well as opposed to eQTL approach and on which information the network put emphasis.

1.3 Approach

The purpose of this thesis is to address the gene-gene interaction problem, and to develop a comprehensive model with accurate predictions. We implement various approaches for this purpose. The steps of our methodology are as follows:

1. **Data gathering and input preparation:** We collect data from two data sets, namely AddNeuroMed and Harvard Brain Tissue Resource Centre (HBTRC), which have SNPs and gene expression levels collected from blood samples and brain tissues. Then, we perform the following significant tasks required before developing the model:

   (a) **Data wrangling:** In this step, SNPs from raw data are first mapped to genes, then classified as either TFs or targets to create gene pairs. We generate genotypes and gene expression levels as NumPy arrays.

   (b) **Example creation:** By using gene pairs, we divide pairs into training and testing data such that there is no direct or indirect correlation between them. Then, we generate +1 (i.e., gene A $\xrightarrow{regulates}$ gene B), -1 (i.e., gene A $\xleftarrow{is\ regulated\ by}$ gene B), and 0 (i.e., no interaction between gene A and gene B) examples from training and testing data.

2. **Developing the model and measuring accuracy:** The primary technique of the model is RNN [26] which involves Long Short-Term Memory (LSTM)

cells [4] to produce a robust network handling the vanishing gradient problem. There are three versions of the network: static, encoder, and attention which are defined elaborately in Chapter 3. To prevent overfitting problems, we add dropout wrappers in every layer, and ℓ_2 regularisation to network weights. Then, we perform all experiments implemented by variants on both data sets. Our experiments have two parts: (1) train and test models on data sets separately, and (2) train on one data set and test on the other. Finally, we measure the accuracy of our models over example data generated from known TFs and targets.

3. **Comparison with existing work:** In order to evaluate our model's performance against existing models, first, we design a baseline method for the task of gene-gene interaction. There has been a lot of research on gene-gene interactions, and some of the notable approaches are mentioned in Section 1.1. We choose [1]'s work as it is one of the state-of-the-art and most sensible approach using both genotypes and gene expressions as inputs. The authors propose a statistical approach based on eQTL analysis by which a number of significant gene - SNP pairs are identified based on spatial distance. Then, with Linkage Disequilibrium (LD) which measures pairwise linkage disequilibrium (i.e., correlation) between variants, some of identified pairs are eliminated.

4. **Understanding networks:** We develop an attention mechanism, based on [6]'s work, as a layer to comprehend how the network does its learning (e.g., to which time steps it assigns more weights). Then, we provide a brief analysis of its output with respect to inputs we use.

1.4 Thesis Structure

The thesis has been organised as follows. Chapter 2 provides an explanation of the preliminary concepts on statistical analysis as well as the NN methods to be used for gene-gene interactions. The architecture of the GeneInteraction DeepRNN along with its modifications has been discussed in details in Chapter 3. In Chapter 4, we describe the data sets of which will be used to test the performance and explain how they are wrangled and set for the network. Furthermore, it presents an account of the experiments which we undertake to compare the model on different data sets. We also discuss alternative performance checking techniques, such as attention, which has been applied on the GeneInteraction DeepRNN to interpret to which neurons (nodes) the network assign more value. Finally, we present our conclusions about the work in Chapter 5.

Chapter 2

Preliminaries: Background on Expression Quantitative Trait Loci and Recurrent Neural Networks

In order to explore the application of the GeneInteraction DeepRNN, and its performance as opposed to statistical models, an in-depth discussion of the theoretical aspects of RNNs as well as the current research on detecting gene-gene interactions with statistical analysis are necessary. This chapter aims to provide the readers an insight into aforementioned topics.

2.1 Expression Quantitative Trait Loci

An eQTL is a genomic locus on which there is a variant (i.e., SNP) that influences the expression level of RNA. In other words, eQTL is a region of DNA that alters how much a gene is transcribed (i.e., "active"). There are two types of eQTL: i) when eQTLs affect genes in nearby locations, they are referred as *local* or *cis*-eQTLs. ii) if eQTLs cause changes in genes in distant locations, possibly on different chromosomes, eQTLs are referred as *distant* or *trans*-eQTLs. Hence, eQTL analysis aims to detect associations between variations in gene expression levels and genotypes (SNPs). There are different types of eQTL analysis such as linear regression approach or Bayesian regression [27].

The analysis is known to be computationally expensive due to large data sets which comprise genotypes measured over thousands or millions of SNPs, and gene expression levels over tens of thousands of transcripts (genes). However, [28] suggested a new technique that is computationally efficient, at least 2-3 times faster than existing eQTL analysis; hence, we take advantage of this algorithm for our baseline statistical analysis. In the next section, we will be looking at the details of the new method, MatrixEQTL, applied on gene-gene interaction problem.

2.1.1 MatrixEQTL Approach

In [28], the author has described a practical method, called MatrixEQTL, to implement eQTL analysis with faster computation time as opposed to other eQTL approaches. MatrixEQTL seeks a linkage between each SNP and transcript by modelling the effect of genotypes as either additive linear (least squares method) or categorical. It also performs a separate test, called multi-correction procedure, for each gene-SNP

pair in order to identify and check for false discoveries, and corrects for multiple comparisons by measuring False Discovery Rate[8] FDR [29]. SNPs located near a gene are more likely to regulate the expression of that gene. Based on this fact, MatrixEQTL uses different *p-value* thresholds and FDR for nearby (*cis*-) and distant(*trans*-) eQTLs.

MatrixEQTL comprises of an algorithm based on a simple linear regression model as it is one of the most common models applied to eQTL analysis. SNPs are encoded as 0, 1, 2 where, for given allele pairs (e.g., A and G), 0 represents the minor (i.e., rarer) allele such as A/A, 1 denotes either A/G or G/A, and 2 represents major (i.e., common) allele such as G/G. The correlation between gene expression g and genotype s is assumed to be linear with respect to the Minor Allele Frequency[9] MAF:

$$g = \alpha + \beta s + \epsilon, \qquad \text{where } \epsilon \sim \text{i.i.d } N(0, \sigma^2) \tag{2.1}$$

Normally, in standard eQTL analysis, we need to calculate a number of variables: the sample mean $\overline{g}$ and $\overline{s}$, the slope coefficient $\widehat{\beta}$, the intercept $\widehat{\alpha}$, the residuals e_i, the total sum of squares *SST*, and the residual sum of squares *SSE*; then, we apply one of test statistics such as t-statistic, F-test, or Likelihood Ratio (LR). Finally, we obtain a corresponding *p-value* for the test of every gene-SNP pair as in simple linear regression. Instead, MatrixEQTL does calculate p-values only for the test statistics of gene-SNP pairs exceeding a given threshold, indicating the significance level.

The selection of the test statistic has an effect on the performance and common test statistics are: t, F, R^2 and LR, which are equivalent in the sense that they can be written as functions of sample correlation, $r = corr(g, s)$, as follows:

$$t = \sqrt{n-2}\frac{r}{\sqrt{1-r^2}} \qquad F = t^2 = (n-2)\frac{r^2}{1-r^2}$$
$$R^2 = r^2 \qquad LR = -n\log(1-r^2)$$

MatrixEQTL chooses the absolute value of the sample correlation $|r|$ as the test statistic for the simple linear regression and as the threshold while searching for which gene-SNP pairs are more significant. If the genotype and gene expression levels are normalised to have zero mean and unit sum of squares, the correlation does not alter:

$$\sum g_i = 0, \qquad \sum g_i^2 = 1, \qquad \sum s_i = 0, \qquad \sum s_i^2 = 1 \tag{2.2}$$

[8]The proportion of error caused by falsely rejecting null hypotheses
[9]The frequency of the occurrence of minor allele in a given population (data set)

The correlation between genes and SNPs can be expressed as follows:

$$r_{gs} = corr(g, s) = \frac{\sum(s_i - \overline{s})(g_i - \overline{g})}{\sqrt{\sum(s_i - \overline{s})^2 \sum(g_i - \overline{g})^2}} \tag{2.3}$$

Using Eq. 2.2, we get:

$$r_{gs} = corr(g, s) = \sum s_i g_i = \langle s, g \rangle \tag{2.4}$$

where r_{gs} represents the correlation between gene expressions and genotypes and $\langle s, g \rangle$ denotes the inner product between vectors s, and g. MatrixEQTL slices data matrices in blocks up to 10,000 variables to avoid excessive memory consumption. We denote S as the genotype matrix, a shape of (10,000 x number of subjects), and G as the gene expression matrix, a shape of (10,000 x number of subjects). Then, the matrix of all gene-SNP pair correlations can be calculated by one matrix multiplication, $G \cdot S^T$ as in Figure 2.1.

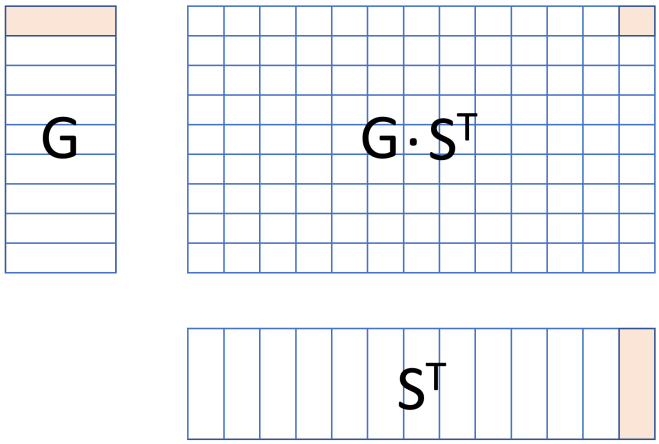


Figure 2.1: MatrixEQTL matrix multiplication

The figure shows the matrix of gene-SNP pair correlations. There are 10,000 rows in G and S matrices as the algorithm slices data matrices in blocks up to 10,000 variables. Each cell in $G \cdot S^T$ represents the correlation for a given gene and SNP pair. For instance, the highlighted cell in $G \cdot S^T$ matrix is calculated by the highlighted cells in G (i.e., first row) and S^T (i.e., last column) matrices. Note that the columns of G and S should have identical subject names.

Hence, the MatrixEQTL algorithm for the simple linear regression can be summarised as:

1. Split input matrices into blocks of up to 10,000 variables as the data sets consists

of millions of SNPs and tens of thousands of gene expressions

2. Normalise both genotypes and gene expression matrices by using Eq. 2.2
3. For each block:
   (a) Calculate the correlation matrix in one matrix multiplication, $G \cdot S^T$, by Eq. 2.4
   (b) Find correlations whose absolute values exceed a predefined threshold
   (c) For those gene-SNP pairs, calculate corresponding p-value, FDR, etc.

Although, the steps of MatrixEQTL can be carried out by a single matrix multiplication, which ensures faster computation time, the method still does not incorporate categorical genotype variables. In the next section, we will present a MatrixEQTL modification, which is based on another model, called ANOVA, introduced by Fisher [30].

2.1.2 ANOVA Model

Analysis of Variance (ANOVA) is a collection of various statistical models and their related estimation procedures. It provides an intuition about the variation among or between populations. In eQTL analysis, what ANOVA ensures is to approach each genotype variable as categorical and estimate its effect on the gene expression, which is expressed as additive and dominant effect of genotypes. The model can be expressed as a linear regression as follows:

$$g = \alpha + \beta_1 s_1 + \beta_2 s_2 + \epsilon \tag{2.5}$$

where $s_1 = I(s = 1)$ and $s_2 = I(s = 2)$ are dummy variables generated for the encodings of SNP vector, s. Let F-test and LR statistics for testing joint significance of s_1 and s_2 be equivalent. Using the fact that F and LR are monotone functions of R^2 in ANOVA model, the test statistics is R^2 for the comparison. R^2 can be calculated if s_1 and s_2 are chosen orthogonal to each other. The steps of the algorithm is as follows:

1. Centre variables g, s_1 and s_2 in order to remove the constant α
2. Orthogonalise s_2 with respect to s_1 for every gene-SNP pair as:

$$\tilde{s_2} = s_2 - \langle s_2, s_1 \rangle s_1 \tag{2.6}$$

3. Normalise s_1 and $\tilde{s_2}$

4. Calculate test statistics: $R^2 = \langle g, s\rangle^2 + \langle g, \tilde{s_2}\rangle^2$ by matrix operations

5. The threshold for R^2 and p-value to fetch gene-SNP pairs with high significance levels can be derived from a modified form of F-test as follows:

$$F = \frac{(n-k-1)R^2}{k(1-R^2)} \tag{2.7}$$

where k denotes the number of genotype categories $(s_1, s_2, ..., s_k)$; k = 2 for this model (s_1 and s_2)

Thus, ANOVA model includes both additive and dominant effect of genotypes. This means that genotypes can be treated as categorical. Given the fact that genotype data we have is composed of three categories (e.g., 0, 1, 2), we will use MatrixEQTL analysis with ANOVA model for our eQTL analysis. In the next section, we will give details of the calculation of FDR.

2.1.3 False Discovery Rate

FDR is a method of computing the fraction of Type-I errors[10] in null hypothesis testing[11] when having multiple test comparisons. The FDR method was officially introduced by Benjamini and Hochberg [29] to identify the proportion of "discoveries" (rejected null hypotheses), which are false (incorrect rejections). MatrixEQTL algorithm calculates FDR only for the gene-SNP pairs with exceeding significance levels - based on the results of test statistics.
Let $p_1 < p_2 < \cdots < p_K$ be the p-values of gene-SNP pairs that pass the user-defined threshold and let N be total number of tests. The FDR for the corresponding pairs can be expressed as follows:

$$\begin{aligned} q_K &= \frac{N}{K} p_K \\ q_i &= \min\Big(\frac{N}{i} p_i, q_{i+1}\Big), \qquad \text{for } i = 1, ..., K \end{aligned} \tag{2.8}$$

2.2 Linkage Disequilibrium

LD means a nonrandom association of alleles at two or more loci [31]. It basically indicates the correlation between two loci. In our study, we consider an interaction of

[10] The rejection of a true null hypothesis ("false positive")
[11] A general statement or a hypothesis to be tested

one pair loci; hence, we will present LD between alleles at two loci as follows:

$$D_{AB} = P_{AB} - P_A P_B \tag{2.9}$$

where, for given alleles A and B at two loci, P_A and P_B denote frequency of alleles A and B, respectively. Hence, D_{AB} is the difference between the frequency of carrying a pair of A and B at two loci, P_{AB}, and the product of P_A and P_B. Simply, D_{AB} means how frequent A and B alleles occur together and determine a phenotype. D_{AB}, the coefficient of LD, is calculated by alleles and is the frequency of determinant pairs (alleles) in a sample.

The coefficient D defines the extent to which two alleles, A and B, are non-randomly associated. When seeking for an association at different pairs of loci, D is not preferred due to the fact that D is constrained by allele frequencies. Instead, another coefficient composed of D and allele frequencies is used to assess LD:

$$R^2 = \frac{D^2}{P_A(1-P_A)P_B(1-P_B)} \tag{2.10}$$

where $R^2 = 1$ is *all* and $R^2 = 0$ is *none* indicator variable showing the presence of A and B. For genotypes, LD means the association between genotypes at a pair of polymorphic loci. $R^2 = 1$ for two SNPs indicates they provide exactly the same information; hence, we cannot make a causal inference from these SNPs; on the other hand, $R^2 = 0$ implies perfect equilibrium between those SNPs showing no correlation.

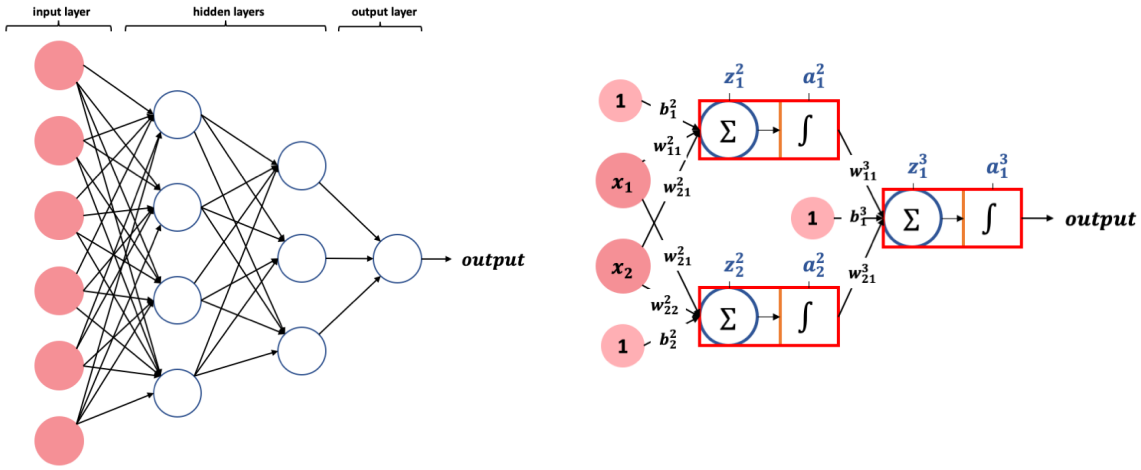
In the next section, we will be looking at RNNs and some variants which we will be applying to our problem. First, we would like to start with sharing details of vanilla NNs.

2.3 Neural Networks

NNs emerged from an important machine learning algorithm called perceptron. A perceptron, introduced by Rosenblatt [32], takes as inputs $x_1, ..., x_D$, and outputs $f = sign(b + w_1x_1 + \cdots + w_Dx_D)$, where $w_1, ..., w_D$ are weight parameters of the perceptron and b is a bias term. Artificial neurons; on the other hand, are more general models of perceptrons such that any activation function can be applied. Let $f : \mathbb{R} \to \mathbb{R}$ be an activation function, $x_1, ..., x_D$ be inputs, $w_1, ..., w_D$ be weights, and b be a bias term. Then, an artificial neuron, known as unit, outputs the following:

$$f(b + w_1x_1 + \cdots + w_Dx_D) \tag{2.11}$$

NNs composed of more than one "layer" - input, hidden, and output - are called Multilayer Perceptrons (MLPs) as in Figure 2.2a. Typically, every unit has a linear function followed by a non-linear activation function [33]. Hence, every unit has a *pre-activation* value and an *activated* output, referred as *activation*. For unit i in layer l, we use z_i^l to denote its pre-activation, and a_i^l to denote its activation function of the unit. As some activation functions apply to the entire layer, we consider activation as a function directly operating on vectors.



(a) MLP with two hidden layers (b) MLP with one hidden layer and two hidden units

Figure 2.2: MLPs

Training and learning process in a MLP is an iterative process composed of two parts:

- Forward Propagation
- Backpropagation

The first part, forward propagation, occurs when the network is exposed to the training data. The input data is passed through the network in a way that all neurons apply transformations based on activation functions on the hidden and output layers in order to use the information they received from the neurons in the previous layers and send it to the neurons in the next layers. Once the data crosses entire layers and complete their transformations, the output obtained by transformations is the label prediction for each input example.

The input $\mathbf{x}$ is thought as layer 1. As there is no linear combination and activation at the input layer, we set $\mathbf{x} = \mathbf{z}^1 = \mathbf{a}^1$. As in Figure 2.2b, every layer besides the input layer first computes a linear function of activations from the previous layer to

get $\mathbf{z}^l$ and applies a non-linear function, f_l, to obtain activation $\mathbf{a}^l$ by applying Eq. 2.11. The weight parameters between the layer $l-1$ and l are denoted as $\mathbf{W}^l$. The pre-activations and activations are computed as follows:

$$\begin{aligned} \mathbf{z}^l &= \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l \\ \mathbf{a}^l &= f(\mathbf{z}^l) \end{aligned} \tag{2.12}$$

As an intermediary step, we use a *loss function* for the estimation of *error* to measure the difference between prediction and actual label. Then, this information is sent for the second part of the process. This part, backpropagation, starts from the output layer and propagates the loss information to all neurons in each layer to calculate each neuron's relative contribution to the overall error. The process goes forward and backward until the weights are updated in such a way that the overall loss is minimised. The *gradient descent* algorithm is used to obtain optimal weights.

Suppose we have a loss function, $\ell(\mathbf{x}, y; \mathbf{W}^{2:L}, \mathbf{b}^{2:L})$, where L is the number of layers. Given training data $\mathcal{D} = \langle (\mathbf{x}_i, y_i) \rangle_{i=1}^N$, the objective function to be minimised for the training is:

$$\mathcal{L}(\mathbf{W}^{2:L}, \mathbf{b}^{2:L}; \mathcal{D}) = \sum_{i=1}^{N} \ell(\mathbf{x}, y; \mathbf{W}^{2:L}, \mathbf{b}^{2:L}) \tag{2.13}$$

For the optimisation, we need to compute the gradient of $\mathcal{L}$ with respect to the parameters $(\mathbf{W}^{2:L}, \mathbf{b}^{2:L})$ as follows:

$$\nabla_{(\mathbf{W}^{2:L}, \mathbf{b}^{2:L})} \mathcal{L} = \sum_{i=1}^{N} \nabla_{(\mathbf{W}^{2:L}, \mathbf{b}^{2:L})} \ell(\mathbf{x}, y; \mathbf{W}^{2:L}, \mathbf{b}^{2:L}) \tag{2.14}$$

Having provided brief information about NNs, we will explain why RNNs are more effective than NNs on sequences and how RNNs work in the next section.

2.4 Recurrent Neural Networks

Traditional NNs take a fixed-sized vector as input $x_1, ..., x_D$ (Figure 2.2b depicts detailed architecture) which limits its usage in situations that include a "series" type input with no predetermined size. RNNs take inputs in sequences such as text, genomes, and handwriting. These networks consider time and sequence into account one-by-one such that the decision made at time step $t-1$ influences the outcome that is determined at time step t. Briefly, the network determines how to respond to a new data based on the present and the previous time steps.

RNNs, based on Rumelhard's work [26], are different from traditional NNs in that a loop connects information from previous steps to the next step. In Figure 2.3, rolled network represents how the network, A, composed of loops, looks at input x_t and outputs h_t, where t represents time steps. Unrolled one, on the other hand, is not only a chain structure, but a multiple copies of the same network, each propagating the information to a successor network.

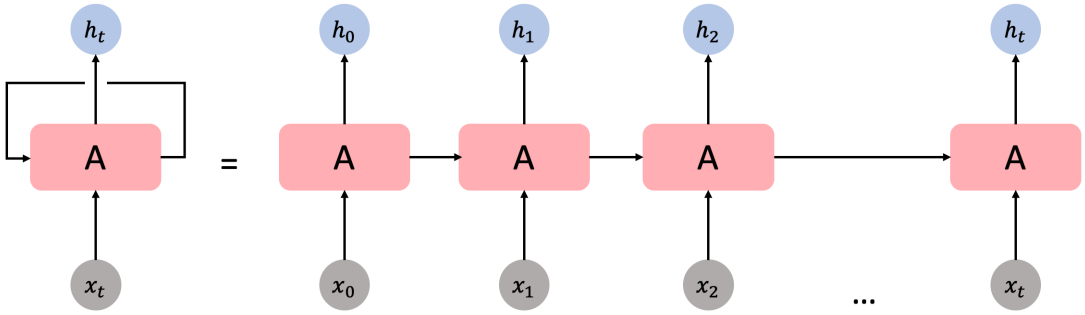


Figure 2.3: RNN unrolling

The looping architecture of RNNs enables us to take advantage of *parameter sharing* because we use same weights over and over again to different parts in the input sequence. We obtain the value of each input by introducing the "hidden state" that connects one input to the next one. The hidden state retrieves the relationship that adjacent parts might have with each other in a sequential input and keeps altering in every time step. RNNs have ability to connect the information from previous steps in order to determine current output as long as there is a small gap between them. In such cases where the gap between the related information from past and the current time step is very large, RNNs cannot learn how to link the information because gradients carrying the information to update parameters get smaller and smaller; hence, the *vanishing gradient* problem occurs.

2.4.1 Backpropagation and Vanishing Gradient Problem

RNNs inherently have the vanishing gradient problem, which inhibits learning in long sequences. The gradient, that passes the information and is used in weight parameter updates, converges to 0 over time. Hence, the parameter updates becomes insignificant and no learning can be done. As our problem is a classification of gene-gene interactions at the last time step of the sequence, it can be considered as a many-to-one

model. The many-to-one model is as follows:

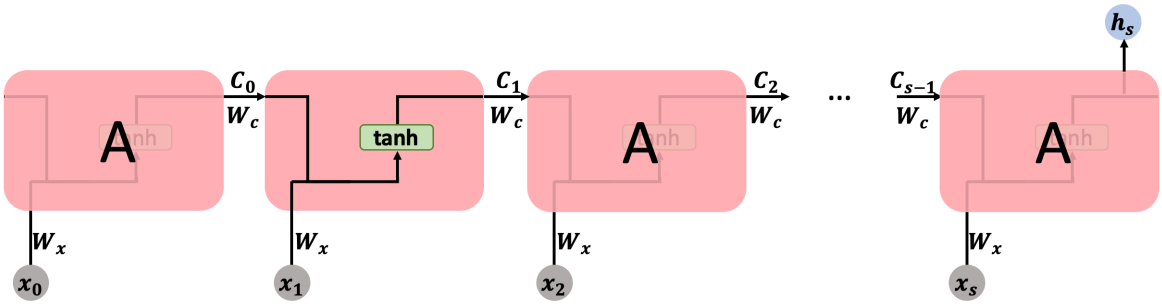


Figure 2.4: Standard RNN

Suppose the input sequence is composed of $x_0x_1x_2...x_s$, where s represents the sequence length. The network has an input sequence of vectors $[x_0, x_1, ..., x_t]$ at time step t. Cell state vector, $[C_0, C_1, ..., C_{t-1}]$, keeps the past information that is necessary for the cell state at t. The network's unit at time step t has an input cell state, C_{t-1}. The input vector, x_t, and cell state vector, C_{t-1}, are concatenated in order to obtain the input vector, $[C_{t-1}, x_t]$, at time t. Parameter sharing is a significant advantage of RNNs; hence, there are only two weight parameters, W_C and W_x, connecting the input vector to the next cell state. The weight vector is denoted as $W = [W_C, W_x]$. Then, after passing the input vector and weight vector through hyperbolic tangent (tanh) activation function, we get:

$$C_t = \tanh(W_C \cdot C_{t-1} + W_x \cdot x_t) \tag{2.15}$$

C_t updates at every step according to Eq. 2.15 and the network outputs the prediction vector, h_s, at the last time step. We compute the prediction error (loss), $\mathcal{L}$, and use the backpropagation algorithm to calculate the gradient with respect to the weight vector as:

$$\nabla_W \mathcal{L} = \frac{\partial \mathcal{L}}{\partial W} \tag{2.16}$$

The gradient is used to update weights as follows:

$$W \leftarrow W - \eta \frac{\partial \mathcal{L}}{\partial W} \tag{2.17}$$

where η denotes the learning rate.

When the gradient in Eq. 2.16 is rewritten by using chain-rule, we get:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}}{\partial h_s}\frac{\partial h_s}{\partial C_s}\frac{\partial C_s}{\partial C_{s-1}}\cdots\frac{\partial C_1}{\partial C_0}\frac{\partial C_0}{\partial W} \\ &= \frac{\partial \mathcal{L}}{\partial h_s}\frac{\partial h_s}{\partial C_s}\Big(\prod_{t=1}^{s}\frac{\partial C_t}{\partial C_{t-1}}\Big)\frac{\partial C_0}{\partial W}\end{aligned} \tag{2.18}$$

We can compute the partial derivative of C_t by using Eq. 2.15:

$$\begin{aligned}\frac{\partial C_t}{\partial C_{t-1}} &= \tanh'(W_C \cdot C_{t-1} + W_x \cdot x_t) \cdot \frac{d}{dC_{t-1}}[W_C \cdot C_{t-1} + W_x \cdot x_t] \\ &= \tanh'(W_C \cdot C_{t-1} + W_x \cdot x_t) W_C\end{aligned} \tag{2.19}$$

Using Eq. 2.18 and 2.19, we get:

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial h_s}\frac{\partial h_s}{\partial C_s}\Big(\prod_{t=1}^{s}\tanh'(W_C \cdot C_{t-1} + W_x \cdot x_t) W_C\Big)\frac{\partial C_0}{\partial W} \tag{2.20}$$

Because $\tanh(x)$ function is flat at both ends, $\tanh'(x) \approx 0$ at very small and very large values of x. Therefore, when s is very large, Eq. 2.19 $\rightarrow 0$, so as Eq. 2.16 $\rightarrow 0$. The network's weight update is as follows:

$$W \leftarrow W - \eta\frac{\partial \mathcal{L}}{\partial W} \approx W \tag{2.21}$$

Hence, there is no significant learning occurs when the gap grows due to the vanishing gradient. This brings us to the long term dependency problem and corresponding solution, called LSTM networks.

2.4.2 Long Short-Term Memory Networks

Long term dependency problem was explored in depth by Hochreiter [34] and Bengio, et al. [35]. LSTM networks, introduced by Hochreiter and Schmidhuber [4], are designed to overcome this problem by remembering information for long periods of time.

All RNNs have repeating chain structures. In standard RNN, the repeating unit has a very simple chain-like structure, including a single tanh layer as in Figure 2.4. LSTMs also have chain-like structures; however, the repeating unit in LSTMs is more complex than that of standard RNNs. Instead of having a single tanh layer, there are four layers composed of point-wise operations, vector transfers, and concatenations as in Figure 2.5. The key component of LSTMs is the cell state, C_t, remembering

important information that should pass through each time step and regulated by sigmoid gates.

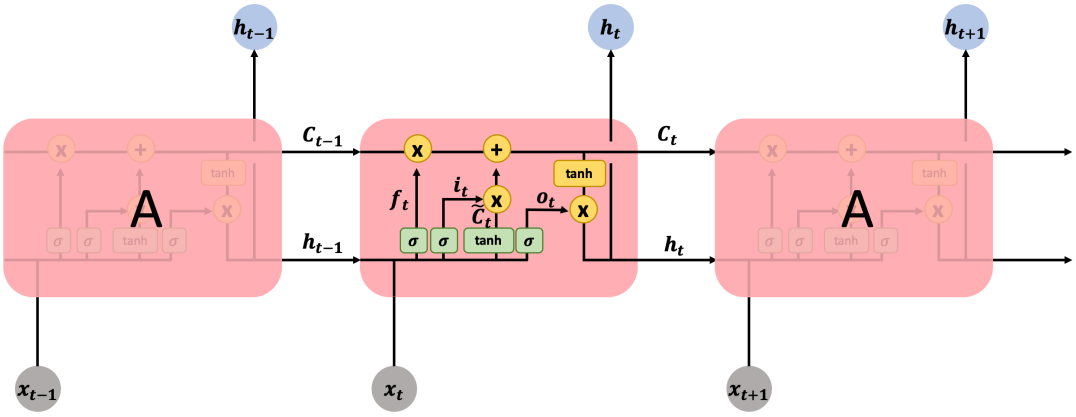


Figure 2.5: LSTM networks in [4]

The steps of obtaining an output value from each unit are as follows:

1. To decide what information we remove from the previous cell state, regulated by *forget gate*, f_t. f_t looks at h_{t-1} and x_t and outputs a number between 0 and 1, where 1 represents remembering all information and 0 denotes forgetting every information.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.22}$$

2. To decide what information we store in the cell state, regulated by *input gate*, i_t. $\tilde{C}_t$ represents a vector of new candidate values for C_t.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{2.23}$$

3. To update the new cell state by using f_t, i_t, C_{t-1}, and $\tilde{C}_t$.

$$C_t = C_{t-1} * f_t + i_t * \tilde{C}_t \tag{2.24}$$

4. To decide what to output, regulated by *output gate*, o_t. In this part, the repeating unit determines which information it should propagate from C_{t-1}.

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= \tanh(C_t) * o_t \end{aligned} \tag{2.25}$$

In our work, we base our GeneInteraction DeepRNN model variants on LSTM cells since our inputs has 101 time steps. We also use other frameworks such as *dropout wrappers*, *encoder modules*, and *attention mechanism* that will be discussed in detail in the next chapter.

Chapter 3

GeneInteraction DeepRNN

In the previous chapter, we have discussed some preliminaries including an explanation of RNNs as well as the state-of-the-art statistical approaches such as eQTL analysis and LD. This chapter describes MatrixEQTL [28] which we extend into a comprehensive method, MatrixEQTL - LD, using LD technique [31] in order to compare our model. The chapter especially presents the GeneInteraction DeepRNN developed to detect gene-gene interactions by using variants of RNN architecture. Finally, the chapter ends with explaining the performance measuring metrics which we use to assess models.

3.1 MatrixEQTL - Linkage Disequilibrium

MatrixEQTL is one of eQTL analysis tools specifically designed for the task of gene-gene interactions. There have been different tools for eQTL analysis [36, 37, 38] over the years. However, all of these tools require intensive computations over large data sets. Firstly, MatrixEQTL is computationally much efficient as opposed to other methods by means of normalising genotypes and gene expressions which gives a great advantage on the eQTL analysis. The second idea is to have a tool to approach genotypes as categories that ANOVA model ensures. MatrixEQTL has an argument *useModel* to specify which model we would like to implement such as *modelLINEAR,* *modelANOVA,* and *modelLINEAR_CROSS*. MatrixEQTL is particularly appealing as it gains its computational power by taking advantage of efficient matrix operations. In eQTL analysis, correcting p-value after multiple comparisons is crucial part of the step to avoid false positives appearing as significant eQTLs. MatrixEQTL implements a separate test for each gene-SNP pair and corrects p-value for multiple comparisons via Benjamini and Hochberg's [29] FDR calculation. However, due to highly variable nature of genomic regions, we use another testing method to assess the significance of any discovered eQTL candidate by using LD, mentioned by [1].

MatrixEQTL - LD model is composed of two parts: identification of significant eQTLs and LD depicted in Figure 3.1. In the original paper [1], gene-SNP pair identification is based on four criteria, which are p-value ≤ 0.05, MAF ≥ 0.05, being in *cis*-regulatory region in addition to FDR = 5%. In our work, we set the conditions to p-value $\leq 10^{-5}$ to be more certain about pairs, and FDR = 10% to incorporate

more pairs to be tested in the next step, LD. We have not investigated interactions between variants in distant regions (i.e., outside of the *cis*-regulatory regions) because there is an evidence from eQTL studies in human proposes *trans*-eQTL effects are less common and have smaller effects [39, 40].

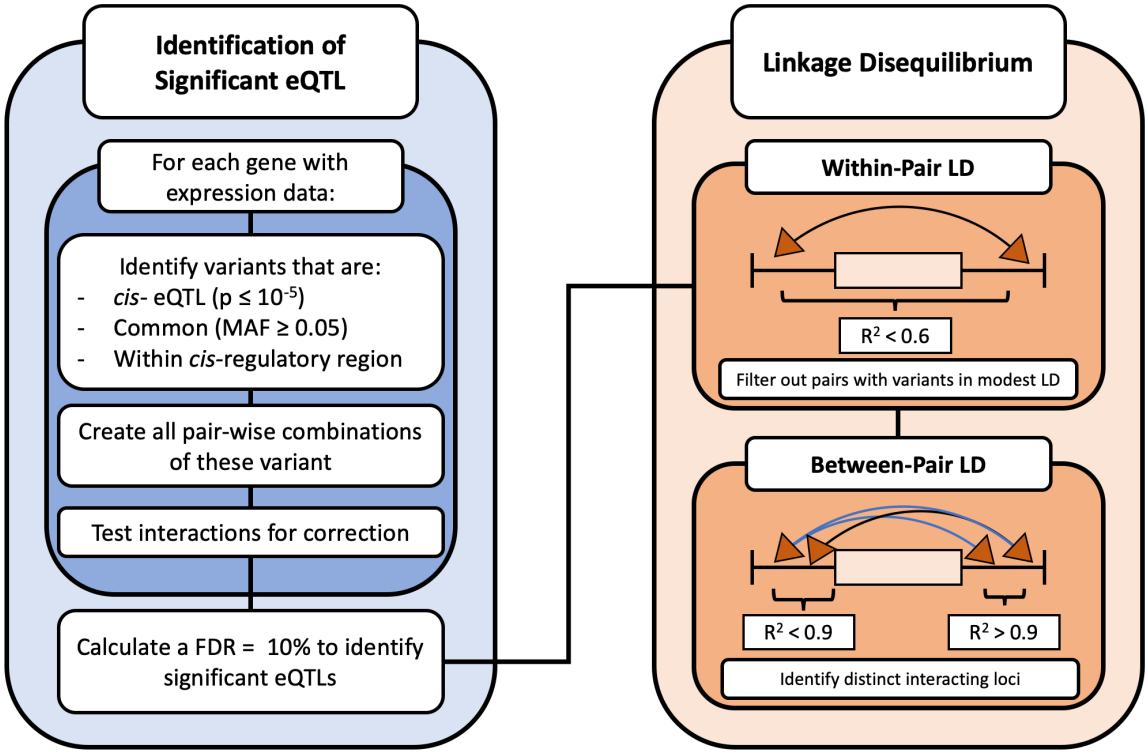


Figure 3.1: Work flow used to identify and group eQTL [1]

Locally significant *cis*-eQTL (denoted by triangles) are paired together and tested for interactions associated with gene expression levels (denoted by arcs). Within-pair LD is calculated and pairs with variants in modest LD ($R^2 > 0.6$) are filtered out. For pairs having interactions indicating the same pair of interacting genomic loci are partitioned into distinct groups (denoted by the arc colour). Grouping two interactions together is determined by whether each SNP within one significant eQTL model is in high LD ($R^2 > 0.9$) with a SNP in the second eQTL model.

Once we have significant gene-SNP pairs from MatrixEQTL, we proceed with LD step. LD between SNPs complicates the interpretation of the association models. There are two types of LD in significant eQTL models as:

1. Within-pair: LD between variants in the same cis-eQTL models
2. Between-pair: LD between variants in different cis-eQTL models

In the first step, we remove all gene-SNP pairs that have variants in modest LD with one another ($R^2 > 0.6$) because modest within-pair LD ($R^2 \leq 0.6$) indicates

that variants can actually cause an association with gene expression. Secondly, we calculate between-pair LD, which means the correlation of variants in distinct eQTL models. Variants with high LD, in other words highly correlated ones, are grouped together because they most likely indicate the same pair of genomic loci.

The forementioned methodology can be implemented on different data sets with genotypes including allele counts and allele types, gene expression levels, chromosome numbers and variant locations, in addition to genes' start and end points. MatrixEQTL [28] with LD [41] will be our baseline comparison framework in this thesis. The next section is dedicated to discussing frameworks that will be used for GeneInteraction DeepRNN model.

3.2 Recurrent Neural Network Frameworks

3.2.1 Encoder Module

One of the requirements for good performance in gene-gene interactions is being able to read genotypes at variable lengths. In general, RNNs have proven to be good at learning sequences at varying lengths in such cases where sequences are padded to have fixed sized input vectors. However, the performance can be improved further if an effort is made to incorporate variable sequence length within the network. There are a few ways of sequence length processing in RNNs, but we focus on encoder module introduced in sequence-to-sequence models.

The concept is widely used in machine-to-machine translation with an encoder and a decoder, introduced by [5]. In this case, the encoder obtains a fixed-length vector representation from a variable length input sequence, and the decoder makes accurate predictions for variable length target translations that are conditioned on the input sequence. The main idea behind the model is to use one LSTM cell to process the input sequence, one time step at a time, and to attain a fixed-sized vector representation. Then, the network uses another LSTM for the output sequence based on the vector representation as shown in Figure 3.2.

In this framework, the encoder converts a sequence of vectors $\mathbf{x} = (x_0, x_1, ..., x_{T_x})$ into a context vector, c. Next, the decoder is trained to predict the next output y_{t+1}(at time step $t + 1$) given the vector c and previous outputs $(y_1, y_2, ..., y_t)$. Briefly, the decoder represents a probability over the whole output vector $\mathbf{y}$ by a conditional joint probability as:

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t|(y_1, y_2, ..., y_{t-1}), c) \tag{3.1}$$

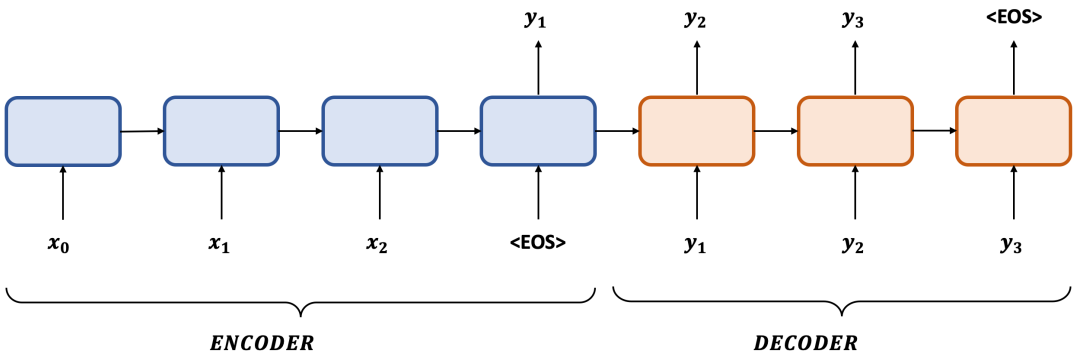


Figure 3.2: Encoder-decoder RNN model for sequence-to-sequence learning [5]

This technique does work on sequence-to-sequence learning. Since input sequences generated from genotypes and gene expressions have variable lengths and output is a classification, one should modify the encoder-decoder module. Hence, in our work, the encoder extracts a variable length vector from a fixed-length input sequence. Note that we require that each example sequence also ends with a special end-of-input symbol similar to "<EOS >" in Figure 3.2. That symbol is used to fill the rest of the sequence and the decoder is only applied to generate a correct classification conditioned on the output of the variable length vector. An illustration has been provided in Figure 3.3 for the reader's convenience.

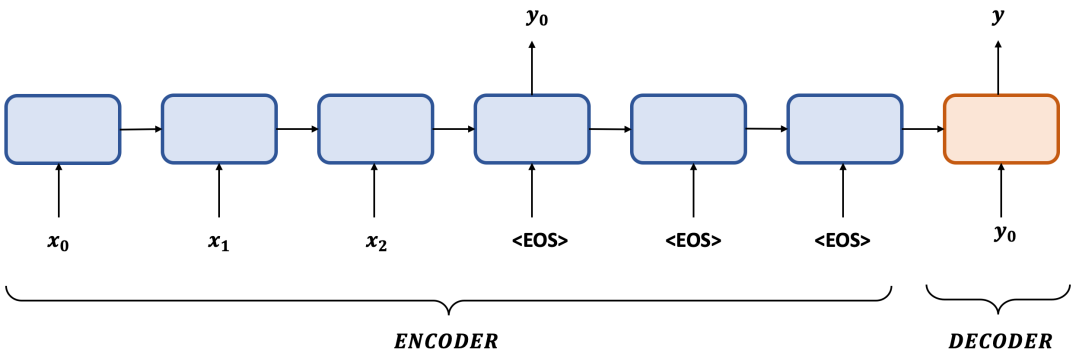


Figure 3.3: Encoder-decoder RNN many-to-one model used for gene-gene interaction

3.2.2 Attention Mechanism

A standard RNN either with regular RNN or LSTM cells, learns parameters in different tasks such as machine translation, time-series prediction, and image captioning where the input include long sequences of data and the output is another sequence. LSTMs overcome the vanishing gradient problem and adapt to work well on long sequences with no information loss, whereas advanced models have limitations developing robust models working with long sequences. Hence, one needs to change the architecture adapted to better handle such tasks.

Attention, introduced by [6], is a mechanism which enables the network to concentrate on specific parts of input sequence and predicts an output with easier and more accurate learning. The main idea behind the concept is to utilise all states in order to create the vector, c, as mentioned in the previous section. The attention mechanism, located between the encoder and decoder, takes as inputs from encoder's output vector, $\mathbf{h} = (h_1, h_2, ..., h_{T_x})$ to generate a context vector c. c makes the decoder to focus on specific points of the input in order to capture relevant parts. The context vector is computed as the weighted sum of the output vectors as follows:

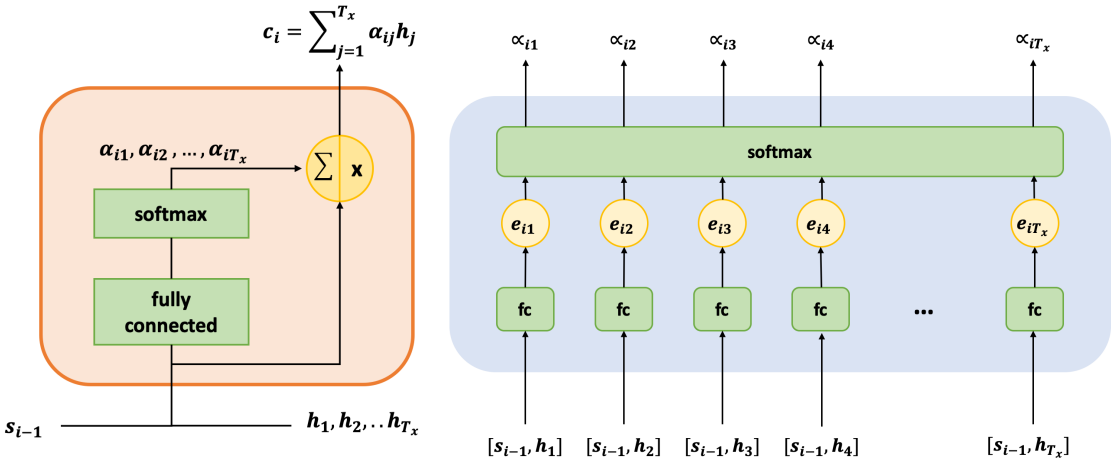
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{3.2}$$

where T_x denotes the input sequence length, i and j represent the time step of the output and input sequence, respectively. The attention weight of each output, h_j, is calculated as:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp_{ik}} \tag{3.3}$$

where $e_{ij} = \text{FC}(s_{i-1}, h_j)$(FC: Fully Connected) is a model which shows how much the inputs around time step j and the output at time step i are associated. In short, the attention layer is a *fully connected* network receiving the concatenation of vectors $[s_{i-1}, h_j]$(s: states) as an input at time step i. The network consists of a FC layer. The output of the layer, called as e_{ij}, is passed through a softmax function in order to calculate the attention weights, which is in $[0, 1]$. An illustration of attention mechanism has been provided in Figure 3.4.

Notice that the attention weights denote the significance of h_j with respect to the previous hidden state s_{i-1} to decide s_i and predict y_i. The attention weight, α_{ij}, makes the network to concentrate on the input x_j in deciding the output y_i. The fully-connected layer in the mechanism is trained along with encoder and decoder



(a) Components of attention mechanism (b) Computation of attention weights

Figure 3.4: Attention mechanism in [6]

At time step i in output sequence, the mechanism has $h_1, h_2, ..., h_{T_x}$ and s_{i-1} as inputs. It uses *fully connected* layer and softmax function to compute attention weights, $\alpha_{i1}, \alpha_{i2}, ..., \alpha_{iT_x}$ to be used for context vector, c

modules using backpropagation. Hence, we obtain another set of weights for learning and fully-connected network is also trained. The weight vector, W_A, having a shape of T_x x T_y, represents the mapping from every input to every output.
Note that the typical attention mechanism, provided in Figure 3.4, is applied at every time step of output sequence inasmuch as what the network predicts and the output is determined by attention weights computed specifically at each time step. In our work, we use an attention mechanism that incorporates states and outputs of input sequence for predictions. The mechanism weights help us comprehend on what the network focuses.

3.2.3 Dropout for Recurrent Neural Networks

Another requirement for high performance for NNs is having a regularisation. Due to a large number of parameters which increase the capacity, the networks are prone to overfitting in the absence of large amounts of training data. Therefore, the regularisation techniques are necessarily used to prevent overfitting problem.

Dropout, introduced by [42] as a regularisation method for feed-forward NNs, does not perform well on RNN. Bayer et al. [43] suggest that the recurrence in RNN proliferates noise. In the work [7], authors have described a method to apply dropouts to a particular subset of RNNs, especially works well with LSTM cells. Their approach

involves applying dropouts to non-recurrent connections such that LSTM cells do not lose its valuable information. Their method is as follows: the dropout twists the information carried by units and forces units to make their intermediate computations more powerfully. The dropout operation can be applied to LSTM cell at time step t in layer ℓ as follows:

$$\begin{aligned} i_t^\ell &= \sigma(W_i \cdot [h_{t-1}^\ell, \mathbf{D}(h_t^{\ell-1})] + b_i) \\ f_t^\ell &= \sigma(W_f \cdot [h_{t-1}^\ell, \mathbf{D}(h_t^{\ell-1})] + b_f) \\ o_t^\ell &= \sigma(W_o \cdot [h_{t-1}^\ell, \mathbf{D}(h_t^{\ell-1})] + b_o) \\ \tilde{C}_t^\ell &= \tanh(W_C \cdot [h_{t-1}^\ell, \mathbf{D}(h_t^{\ell-1})] + b_C) \\ C_t^\ell &= f_t^\ell * C_{t-1}^\ell + i_t^\ell * \tilde{C}_t^\ell \\ h_t^\ell &= \tanh(C_{t-1}^\ell) * o_t^\ell \end{aligned} \tag{3.4}$$

where $\mathbf{D}$ is the dropout operator which assigns zero to a random subset. The information carried over units is not lost because dropout operations occur between layers, not within the same layer. Figure 3.5 depicts the flow of how information is influenced when a dropout is implemented between time step $t-2$ and time step $t+2$. Note that the dropout alters the information flow $L+1$ times, where L is the number of layers.

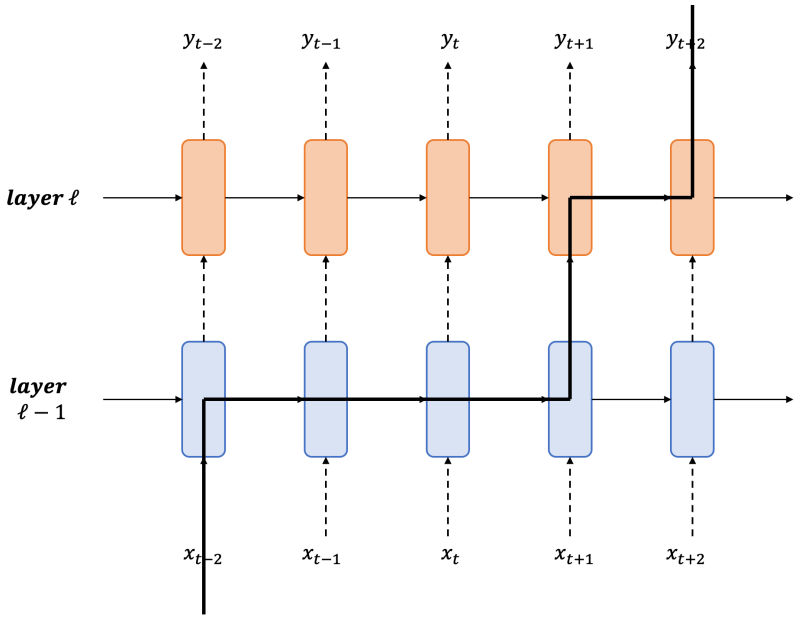


Figure 3.5: Flow of LSTM regularisation with dropout proposed in [7]

The thick line represents the flow of information in LSTM.

3.3 GeneInteraction DeepRNN

In the previous sections, we have discussed the components of the MatrixEQTL - LD model as our baseline model, as well as the fundamental frameworks necessary for our network. Embraced by this knowledge, in this section, we present GeneInteraction DeepRNN, a variant of sequence model, which is able to classify whether two genes regulate one another. As explained in Chapter 2, the main idea behind creating an exhaustive network that bypasses vanishing gradients and long short-term dependency problem is to use LSTM cells.

In our work, we use ℓ_2 regularisation applied to network weights and insert dropout wrappers between layers of the network. These wrappers place an added level determining the rate of outputs to be passed from one layer to the next. As in optimal networks, we have chosen to insert dropout wrappers after each layer because a network like GeneInteraction DeepRNN which contains only 1500 training examples will adapt itself to the training data over iterations and result in a weak performance in the absence of dropout layers. Furthermore, we also create three variants of the GeneInteraction DeepRNN: *static* model, *encoder* model, and *attention* model. The purpose behind the static model is essentially to enable training on sequences with padding values as well as to assess its performance with different architectures. These versions will be discussed in detail in the next section.

3.3.1 Model Variants

In this section, we discuss details of the GeneInteraction DeepRNN architecture and its versions. Before we explicate the variants, a quick note on the structure of the model should be given. Mainly, the fundamental framework is similar to LSTMs (i.e., the network takes inputs in sequences and generates LSTM cells for each time step; then, the network allows information pass through these cells even for long sequences without losing relevant information). It is worth noting that there are architectural differences between regular LSTM networks [4] and the one which we use. The differences are as follows:

1. Instead of combining inputs for gene A and gene B, we construct independent networks for both genes. We use separate LSTM networks with dropout wrappers for each layer. This helps in obtaining different representations for gene A and gene B.

2. An element-wise addition is applied to outputs before passed through a dense

layer. This is slightly different from the original many-to-one networks where the output of the last time step is the determinant of classification.

In our work, we have developed three variants of the model. The *static* model includes LSTM architectures comprising of 3 layers with 32 neurons as well as dropout wrappers and this variant takes fixed size sequences. Though sequence lengths differ for each example, they are padded with 0s to achieve same lengths. The *encoder* model has a similar architecture, but the input sequences, this time, are read until paddings. Paddings can be considered as <EOS> in Figure 3.3. Rather than using last hidden output for predictions, with this architecture, we use last relevant output. Not only does this dynamic architecture enable the model to make more accurate predictions, but also saves computation time. For instance, the maximum sequence length is 101 and an example has a sequence of 40 frames; hence, the encoder sends the output to the decoder when it reaches 41^{st} time step. Finally, the *attention* model has an additional attention layer before output layer. Because number of attention weights is fixed to the maximum sequence length, we implement an attention mechanism to the encoder variant adapted to fixed sequence length feature. The purpose of the attention layer is to comprehend on which points of the sequence the network focuses. This allows us to explain why our network works better than traditional statistical analysis for gene-gene interaction problem. An illustration of *encoder* and *attention* model variants is given in Figure 3.6 and Figure 3.7. Parameters and hyper parameters of both models (e.g., dropout rate, batch size, sequence length, regularisation constants) are described in Table 4.1. Though the parameters for the encoder are very similar to those of attention model, the architecture is slightly different. The full architecture of static variant has been provided in Appendix C. In the next section, we discuss the evaluation metrics to understand performance in GeneInteraction DeepRNN.

3.4 Performance Evaluation Metric

Having described the frameworks and features of variants of the GeneInteraction DeepRNN, in this section, we are ready to address the evaluation metric which we use for assessing the performance of our models and MatrixEQTL - LD analysis.

In classification problems, in order to measure performance of the classifiers, we are particularly interested in the misclassification rates. Though the learning algorithms aim to minimise overall objective such as the addition of costs from LSTM cells and ℓ_2 regularisation, we can measure the test error by calculating the number of misclassified points. However, not all mistakes are considered as equally problematic. For instance,

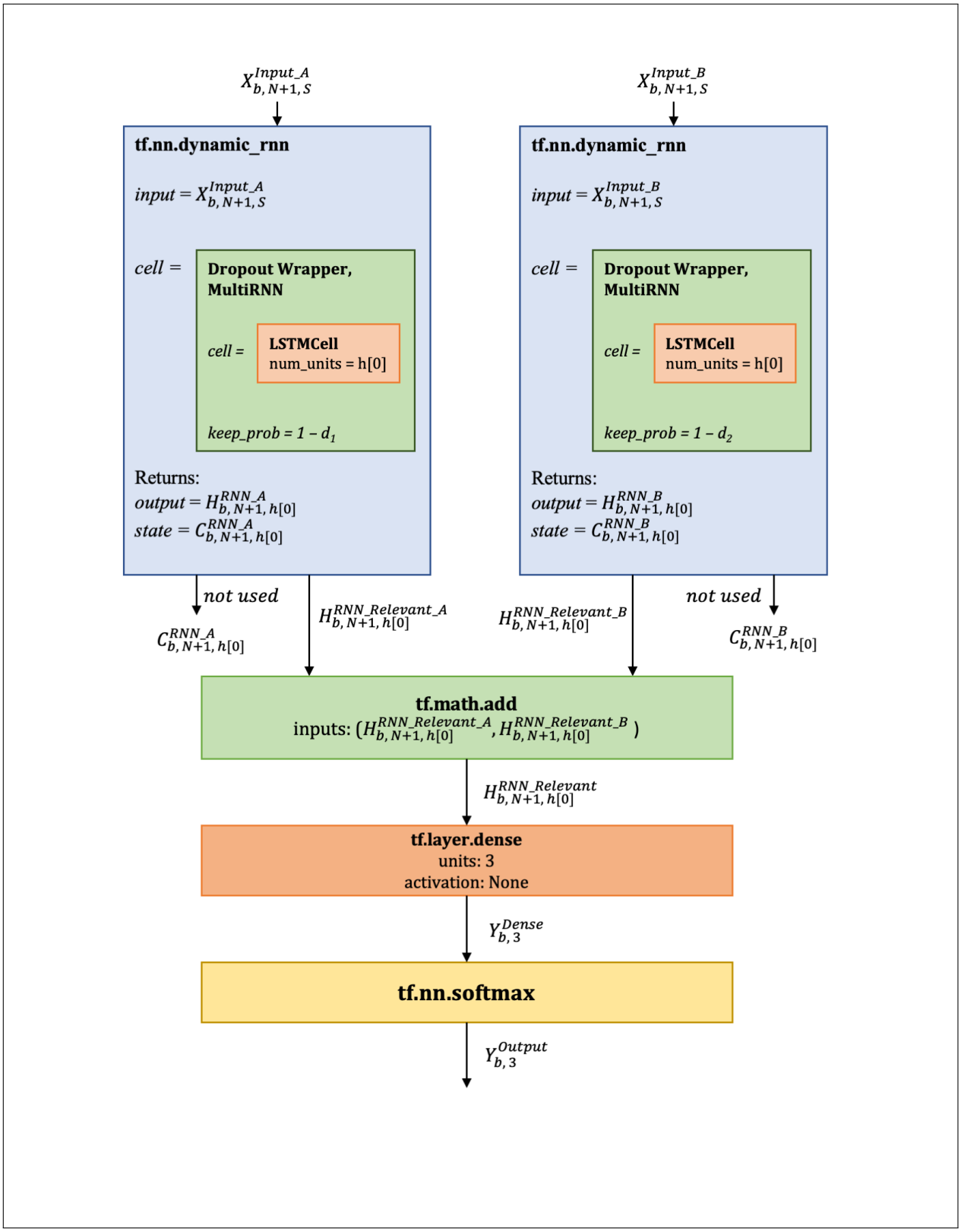


Figure 3.6: Architecture of the GeneInteraction DeepRNN encoder model

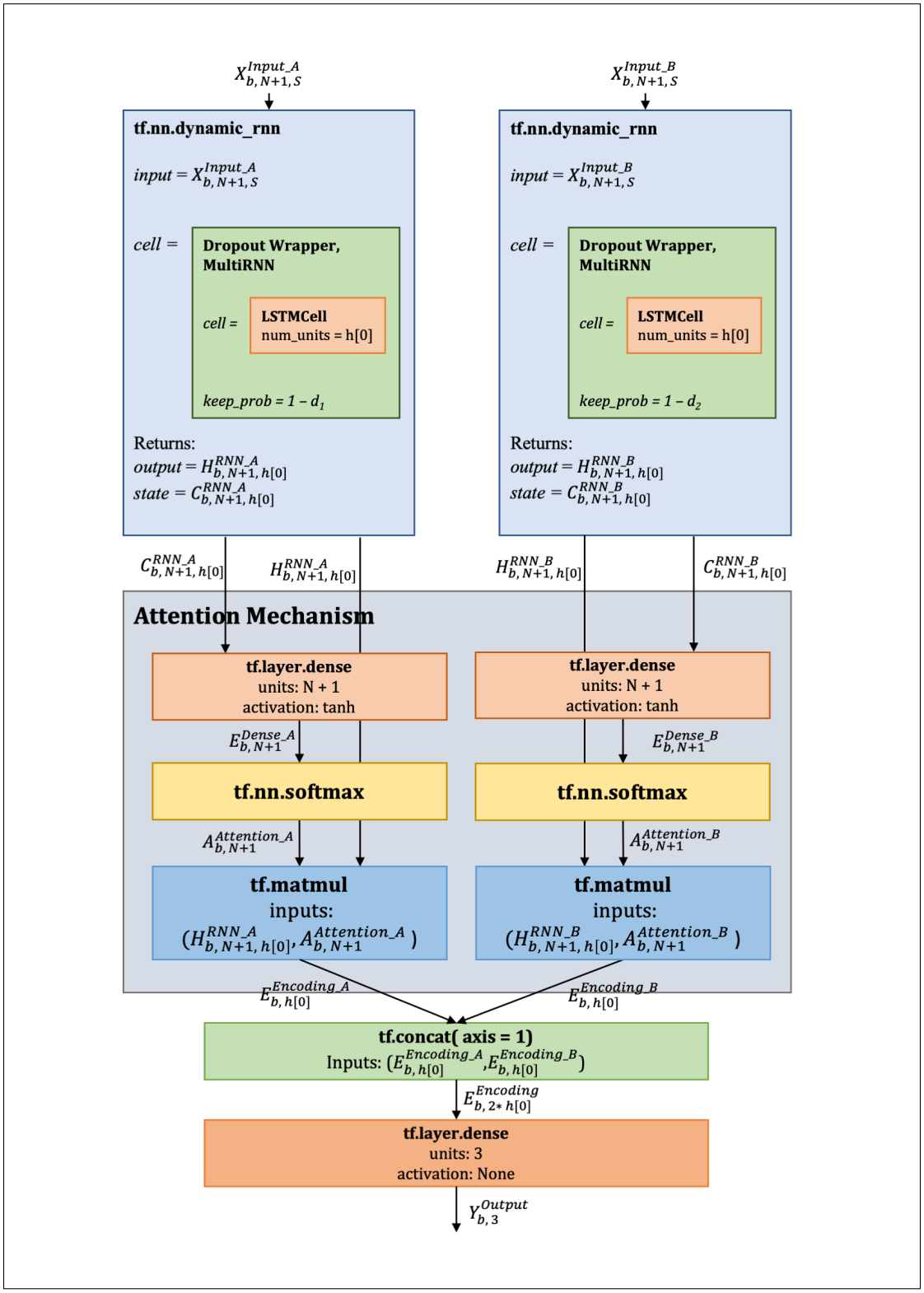


Figure 3.7: Architecture of the GeneInteraction DeepRNN attention model

in the field of epidemiology, detecting whether two genes interact with one another is an extremely challenging task; hence, the effect of classifying as "Gene A regulates gene B", vice versa (suppose it is a false positive) can not be considered equally important as predicting as "No gene regulation at all" (consider as a false negative); hence, we want to treat false positive detection of gene-gene interactions as more significant. For performance measuring, we need to incorporate *true positive*, *false positive*, *false negative*, and *true negative*. The 2 x 2 truth table for a binary classification is presented as below:

Predicted Labels	Actual Labels	
	yes	no
yes	true positive	false positive
no	false negative	true negative

Table 3.1: Truth table (confusion matrix) for binary classification

For a classifier and its predictions on a data set,

- **Accuracy**: The ratio of correctly predicted observation to the total observations
  Accuracy = $\frac{TP+TN}{TP+FP+FN+TN}$
- **Precision**: The ratio of the number of positives to the number of predicted positives
  Prediction = $\frac{TP}{FP+TP}$
- **Recall**: The ratio of correctly predicted positive observations to the all observations in actual class
  Recall = $\frac{TP}{TP+FN}$
- **F1 score**: The weighted average of Precision and Recall
  F1 Score = $\frac{2*(Recall*Precision)}{Recall+Precision}$

where **TP**: true positives, **FP**: false positives, **FN**: false negatives, and **TN**: true negatives. It is worth stating that one might think accuracy is more intuitive and meaningful interpretation to use for performance evaluation; however, accuracy is great for such cases when there is a symmetric data set. On the other hand, F1 score is more useful than accuracy for uneven classes. In our work, there are three classes: +1 (gene A regulates gene B), -1 (gene A is regulated by gene B), and 0 (no interaction between gene A and gene B) labels. The primary goal of the work is to identify an interaction among genes; hence, +1 and -1 examples have more value than

0 examples. Additional insight on precision and recall should be given. Note that precision is *exactness* measure and recall is *completeness* measure. Since F1 score is the harmonic mean of both measures, a higher F1 score can be achieved when both are balanced. Therefore, F1 score is the sensible metric in order to assess the validity of our models. In the next chapter, we mention and discuss the experiment we carry out and the results acquired from the variants.

Chapter 4

Experiments and Results

In this chapter, we present results obtained by applying the GeneInteraction DeepRNN which we develop in the previous chapter on our data sets. We conduct tests to compare accuracy of complex RNN and eQTL statistical analysis by [1] supported by LD [41]. For our tests, we use AddNeuroMed and HBTRC data sets which we describe elaborately in section 4.1. Next, we discuss the first experiment which involves training and testing the models on data sets separately to ensure that the model demonstrates satisfying accuracy. In the second experiment, we apply the networks on one data set for training and on another one for testing to cross-validate the validity of the models. Finally, we compare the performance using the GeneInteraction DeepRNN and MatrixEQTL - LD models. The codes of eQTL analysis with LD are presented in Appendix B.

4.1 Data Sets

In our work, we employ two data sets provided by AMP-AD Knowledge Portal (AMP-AD) and AddNeuroMed consortium (AddNeuroMed) for our experiments.

4.1.1 AddNeuroMed Data Set

AddNeuroMed, a cross-European study funded by private-public partnerships, aims to develop existing experimental models of biomarkers, to find biomarkers associated with AD particularly for the purpose of early diagnosis of the disease. It also intends to detect early development of dementia in patients with Mild Cognitive Impairment (MCI), and track progression of disease for clinical trials and practice. AddNeuroMed includes genomics, proteomics, neuroimaging information from blood samples and brain scans collected from 250 AD patients, 250 MCI patients, and 250 healthy control group.

Part of the data we use for experiments consists of genomic variants with genotypes and demographics as well as gene expression data from blood samples. There are a total of 163,142 SNPs, and 957 gene expression levels from 206 subjects. Using these SNPs and gene expression data, we attain 2204 gene pairs, 957 transcripts, and 163,142 SNPs to be used as input data for generating a total of 1545 examples which of these

are 1500 training set and 45 test set for the network. The details of data processing will be explained in section 4.2

4.1.2 HBTRC Study Data Set

HBTRC is a publicly available data set provided by Mt. Sinai Health System within AMP-AD which is a portal for data, analysis results, analytical methodology and research tools generated through multiple programs. HBTRC has tissues collected from three brain regions: dorsolateral prefrontal cortex, visual cortex, and cerebellum. It has 1,647 tissue samples from 549 brain samples collected from 376 Late-On Alzheimer's Disease (LOAD) patients and 153 nondemented healthy controls.

Only some part of this HBTRC data set is used for experiments. These data consists of genomic variants including alleles, demographics, and genotypes and expression data generated from prefrontal cortex as well as gene expression levels data from dorsolateral prefrontal cortex collected from 310 LOAD patients and 153 nondemented healthy controls. There are a total of 672,828 SNPs and 39,579 gene expression levels from 463 subjects. From these SNPs and gene expression levels, we obtain 2060 gene pairs, 609 transcripts, and 672,828 SNPs to be used to create a total of 1545 examples which of these are 1500 training set and 45 test set for the network. In the next section, we will explicate how raw data is turned into input data for the network.

4.2 Experimental Setup: Raw Data Set and Data Wrangling

One of the main components of our project is to process raw data to get input format for the GeneInteraction DeepRNN. In this section we present the raw data we use and procedures that are followed in order to achieve the training and test data for the network.

4.2.1 Raw Data Set

In bioinformatics, data for samples and genotypes are in formats of .bim, .fam and .bed. .bim file in Figure 4.1a includes information about SNPs such as chromosome numbers, positions on chromosomes, minor and major alleles. .fam file in Figure 4.1b has sample background including family IDs, subject IDs, genders, whether the mother or father of subjects takes part as subjects, and traits (e.g., phenotypes). In .bed file are genotypes of each variant for these subjects. Note that we, as homo sapiens, have two sets of chromosomes, one from each parent. Normally, bases at a specific locus on chromosomes are supposed to be the same. However, in situations where a SNP

occurs, one nucleotide is replaced with another base. For instance, let a part of DNA sequence be ...ATGTTGAC..., which should be identical for the second chromosome. The genotype encoding for the same sequence can be shown as follows:

- ...ATGT**T**GAC... represents usual pairs, which is encoded as 3
  ...ATGT**T**GAC...
- ...ATGT**T**GAC... indicates a substitution of T with base C for one pair, which
  ...ATGT**C**GAC... is encoded as 2
- ...ATGT**C**GAC... shows replacement with base C for both pairs, which is encoded
  ...ATGT**C**GAC... as 1 (rarer case)

In the gene expression data, rows correspond to the sequence IDs, a special ID associated with Reference Sequence (RefSeq)[12] database, of transcripts. The data has two parts where we can extract transcripts' location information such as chromosome number, corresponding genes, start and end coordinates on chromosomes as well as gene expression levels of transcripts for subjects.

	chrom	snp	cm	pos	a0	a1
0	1	rs3094315	0.0	752566	C	T
1	1	rs12562034	0.0	768448	A	G
2	1	rs3934834	0.0	1005806	T	C
3	1	rs9442372	0.0	1018704	A	G
4	1	rs3737728	0.0	1021415	T	C
5	1	rs9442398	0.0	1021695	A	G

(a) .bim file

	fid	iid	father	mother	gender	trait	i
0	15876	15876	0	0	0	-9.0	0
1	15877	15877	0	0	0	-9.0	1
2	15878	15878	0	0	0	-9.0	2
3	15879	15879	0	0	0	-9.0	3
4	15880	15880	0	0	0	-9.0	4
5	15881	15881	0	0	0	-9.0	5

(b) .fam file

Figure 4.1: Examples of SNPs and sample information

In (a), a snapshot of the .bim file is shown. Cm, a0, and a1 denote the position in morgans or centimorgans (genetic distance), minor and major alleles, respectively. In (b), fid shows family ID, and iid represents ID within family.

4.2.2 Data Wrangling and Example Creation

As the GeneInteraction DeepRNN takes input in a shape of (batch size x time steps x subjects), we transform and map raw data into ultimate data format. In this section, we present processes followed to attain inputs. The process is composed of two parts:

[12] An open access collection, built by National Center for Biotechnology Information (NCBI), of publicly available nucleotide sequences (DNA, RNA)

i) extraction of relevant information from raw data, called loadData, and ii) generation of examples from gene pairs intended to feed to the network, called preprocessData. An illustrative flowchart has been presented in Figure 4.2.

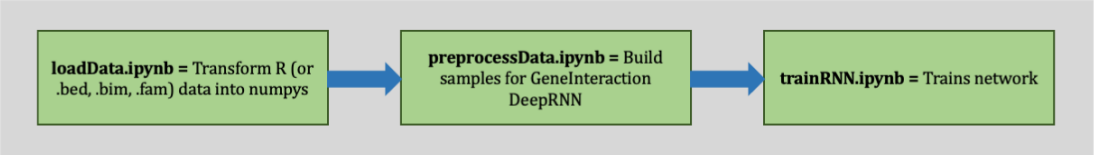


Figure 4.2: Flow for the network

It is worth noting that the input of loadData is either in (.bim, .bed, .fam) or Rdata file formats. Although, one of the data set we use, namely AddNeuroMed, includes most of the necessary data for preprocessData in Rdata, steps of part (i) have been applied to its' (.bim, .bed, .fam) files in order to obtain necessary data for eQTL analysis and LD. The baseline method requires chromosome numbers, positions, and pair of alleles information for SNPs and chromosome numbers, start and end coordinates information for transcripts in addition to genotypes and gene expression levels, respectively. The second data set, namely HBTRC, stored in (.bim, .bed, .fam) files, requires all data wrangling steps. The steps of part (i) are listed as follows:

1. **Mapping SNPs**: It is compulsory to map SNPs to corresponding genes on which SNPs are located. For this purpose, we use *elink* function of Biopython package in Python that sends request to the NCBI Entrez API and retrieves ID for links (SNP $\rightarrow$ gene) created.

2. **Classification of transcripts**: In order to create gene pairs, which have a pair of TF and target gene, transcripts should be identified as either a TF or a target gene. There are only a limited number of TFs that are known to regulate target genes. Hence, we use GWAS[13]'s known TF and target pairs including their Entrez Gene ID[14], and gene symbols (i.e. "NFE2L2" instead of gene "4780").

3. **Gene pairs generation**: After matching transcripts with known TFs and target genes, the generation of gene pairs is in order. This process is a method, for a

[13] An observational study related to genome-wide set of genetic variants occur in different subjects in order to find out if a variant is associated with a trait
[14] An identifier for a gene in NCBI database

given TF, checks and retrieves the index of a valid TF-target pair from the matrix obtained in step 2.

4. **Genotype**: The crucial thing about dealing with genomic variants and gene expressions data is to detect mutual subjects because genomic variants and gene expression are generally collected separately such influences the participation of subjects. Before retrieving genotype data, we proceed with identification of overlapping subjects by their IDs. Then, we create the genotype variable over mutual subjects by using .bed file.

5. **Gene expression**: Having overlapping subjects and classification of transcripts as TFs or targets, gene expression data is generated from raw data, where rows correspond to Entrez IDs of transcripts instead of their RefSeq IDs.

6. **eQTL analysis + LD data**: Now that we have all data necessary for part (ii), there is an additional data generation process. In addition to genotypes, subject names, transcript Entrez IDs, gene expression, SNP IDs, the remaining step is to create data for SNP and gene locations, also called as annotation and probe location, respectively.

loadData is particularly important because it maps the raw data to the data that will be used to generate samples. Note that variables are deliberately turned into "NumPy" arrays because operations on NumPys (e.g., searching for index of an element, boolean operations, array creations) are much faster. The variables obtained by loadData are as follows:

- rRna_nSR: Gene expression levels, NumPy array with a shape of (S x R)
- rSnp_nSN: Genotypes, NumPy array with a shape of (S x N)
- nNs_lRlN: SNP names each gene has, list length of (R: [N])
- nPairs_lPlC: Gene pairs, list length of (P: [C])
- nRs: Transcripts, NumPy array with a shape of (R x 1)
- nNs: SNPs, NumPy array with a shape of (N x 1)
- nSs: Subjects, NumPy array with a shape of (S x 1)

where **N**: number of SNPs, **S**: number of subjects, **R**: number of transcripts, **P**: number of pairs, **C**: number of characteristics, **n**: NumPy array, **l**: list, and **s:** string. A small portion of variables rRna_nSR and rSnp_nSN are presented in Figure 4.3. Notice that

the density of SNP encodings, (i.e., 1, 2, 3) and gene expressions differs from one data set to another. The reason of this variation mostly stems from which tissues or samples are collected. As mentioned in sections 4.1.1 and 4.1.2, AddNeuroMed data comprises of blood samples, whereas HBTRC is collected from brain tissues.

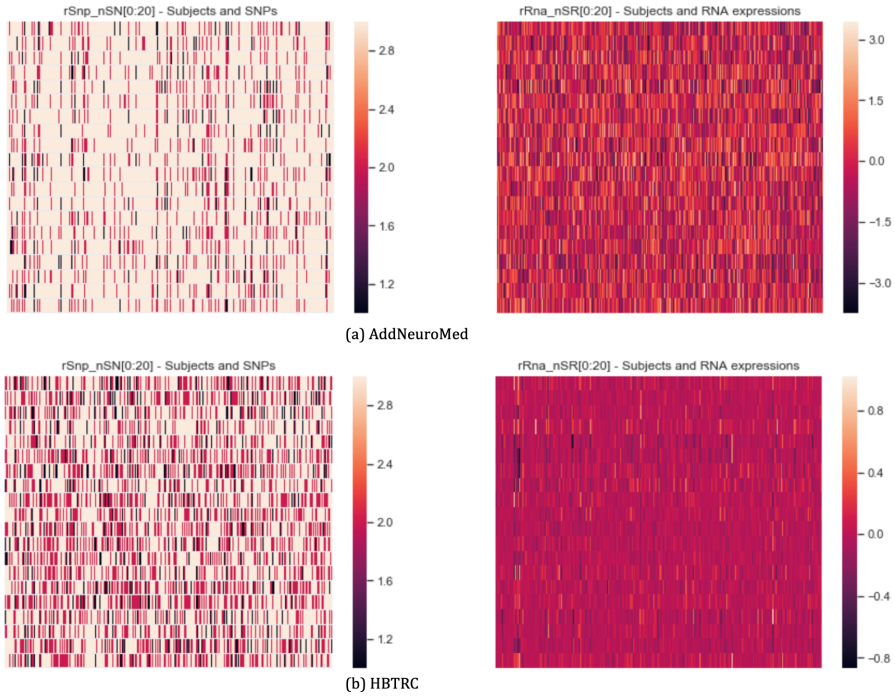


Figure 4.3: Data sets

Figures show the heatmap of SNPs and gene expression levels for 20 SNPs and transcripts, respectively. Notice that in (a), encoding 3 is dominant, whereas in (b), 1s and 2s are abundant. The reason stems from the genetics difference between brain tissues and blood samples. Also, values of gene expression levels are normalised around mean 0.

Part (ii), as an intermediary step, is the backbone of our work because variables created by loadData are passed through a division process followed by a sample building procedure and finally trainRNN takes outputs of preprocessData as inputs. The steps are listed as follows:

1. **Separating data**: It is interesting to mention that nPairslPlC is composed of gene pairs where a TF can be a target in a pair and vice versa. In other words, there is an indirect correlation between most pairs that might lead the network to tweak and learn in virtue of the correlation. Hence, one of the main points we need to ensure in our work is a perfect division of training and test data for each data set. With our method, each TF is searched through with a maximum window for associated pairs (e.g., 20 gene pairs) before moving to another TF

and those pairs are assigned to boolean values to indicate if they belong to train or test data. Then, training and test data pairs are obtained.

2. **Building samples**: GeneInteraction DeepRNN depends on training data in the shape of (number of examples x time steps x subjects) for both gene A and gene B, in addition to the corresponding labels in the shape of (number of examples x number of classes). At every iteration, the network shuffles data and selects samples in batches from the training data randomly; hence, we can run the network multiple times on the same input. Hence, providing an appropriate training and test data become very significant. Building samples process can be divided into five parts:

   (a) **Indexing SNPs**: Since AddNeuroMed and HBTRC data sets have 163,142 and 672,828 SNPs, respectively, an easy access to each SNP becomes computationally important. In this process, for a given pair of genes for each example, it fetches genotypes of all SNPs that genes have from the variable rSnp_nSN in the shape of (206 x 163,142) and (343 x 672,828) in AddNeuroMed and HBTRC, respectively.

   (b) **Generating right directional examples**: Right direction (i.e., gene A $\rightarrow$ gene B) shows that gene A regulates gene B, which is also denoted as $+1$ labels in our work. In order to generate these examples, a random pair is drawn from either training or testing gene pairs and we retrieve genotypes and gene expression information for this gene pair. Same procedure is applied as long as we reach the targeted number of examples for the work. Additional note on the testing data should be given. As we divide our gene pairs to have uncorrelated training and test data, we end up with $\approx$15 gene pairs that are assigned to test gene pairs. Hence, we can use a total of at most 45 test examples.

   (c) **Generating left directional examples**: Like the right direction examples, left direction (gene A $\leftarrow$ gene B) represents gene A is regulated by gene B, denoted as -1 labels in our work. Generating -1 examples are very similar in the sense that when a random pair is selected and corresponding genotypes and gene expressions are fetched for both genes, we store them in the opposite variables (e.g., putting information about gene A to target genes variable and gene B to TFs variable).

   (d) **Generating no interaction examples**: No interaction examples (gene A x gene B), called as 0 labels, are generated from scratch by using all gene

pairs. We check the existence of genes from a randomly drawn pair within all gene pairs and store genotypes and gene expression levels of those genes.

(e) **Labels:** True labels, $+1, -1$, and 0 for the network are transformed into one-hot encoding representations, where $[1, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$ represent $+1$, -1, and 0 labelled examples, respectively.

Additionally, we use an argument to define maximum number of SNPs (e.g., 100) to scan because, for a given gene, if there are more than 100 SNPs, this might be associated with a mutation and affect the network's ability to train. Finally, the resulting variables generated by preprocessData are as follows:

- rSnpA_tr_nXSN: Genotypes of gene A in training data, NumPy array in shape of (X x S x N)
- rSnpB_tr_nXSN: Genotypes of gene B in training data, NumPy array in shape of (X x S x N)
- rRnaA_tr_nXS: Gene expression of gene A in training data, NumPy array in shape of (X x S)
- rRnaB_tr_nXS: Gene expression of gene B in training data, NumPy array in shape of (X x S)
- rRelated_tr_nXC: Labels of training data, NumPy array in shape of (X x C)
- rSnpA_tst_nXSN: Genotypes of gene A in test data, NumPy array in the shape of (X x S x N)
- rSnpB_tst_nXSN: Genotypes of gene B in test data, NumPy array in the shape of (X x S x N)
- rRnaA_tst_nXS: Gene expression of gene A in test data, NumPy array in shape of (X x S)
- rRnaB_tst_nXS: Gene expression of gene B in test data, NumPy array in shape of (X x S)
- rRelated_tst_nXC: Labels of test data, NumPy array in shape of (X x C)
- sGeneNames_tr_nX2: Entrez IDs of gene pairs in training data, NumPy array in shape of (X x 2)

- sGeneNames_tst_nX2: Entrez IDs of gene pairs in test data, NumPy array in shape of (X x 2)

where **X**: number of examples and **C**: number of classes. Furthermore, methods used in the training steps of GeneInteraction DeepRNN variants have been provided in Appendix A.

4.3 Experimental Parameters

The input parameters to train the GeneInteraction DeepRNN model include:

- the *number of neurons* for LSTM cells
- the *maximum sequence length*, which is determined by the number of SNPs each gene has, with an additional length for gene expression levels
- the *dropout rate* for the DropoutWrapper that can be assigned separately to LSTM architecture of gene A and gene B (Refer: Section 3.2.1 for the GeneInteraction DeepRNN architecture),
- an *attention size* in case the model uses an attention mechanism
- a *batch size* for training, and
- the total *number of training iterations*

For entire models, we set 32 as the number of neurons for each LSTM cell, 101 as the maximum sequence length, 101 as the attention size, 150 as the batch size, 0.001 as the learning rate, and train our model for 200 iterations in Experiment 1 and 250 in Experiment 2. The encoder models do not have an attention mechanism; hence, we use variable sequence length for inputs instead of having a fixed length sequence. The *variable length*, a list of numbers with a maximum of 101, contains sequence lengths for each example until paddings (e.g., [101, 40, 32, 19]). For encoder and attention models, there are two additional hyper parameters: a regularisation constant, which is set to 0.0001, penalises weights in both models except for bias, and another regularisation constant applied to dense layer parameters, which is set to either 0.3 or 0.4. The detailed parameter and hyper parameter assignments are given in Table 4.1. Finally, each network is trained on NVIDIA GeForce GTX 1080 Ti GPU. The time required to train for all models are similar, which is around 15 minutes on GPU and 30 minutes on CPU.

Model	Data Set	Seq. Length	# of Epochs	Dropout	λ_{ℓ_2}	λ_{Dense}
Static	AddNeuroMed	101	200	0.6	-	-
	HBTRC	101	200	0.6	-	-
	AddNeuroMed → HBTRC	101	250	0.6	-	-
	HBTRC → AddNeuroMed	101	250	0.5	-	-
Encoder	AddNeuroMed	variable	200	0.6	0.0001	0.3
	HBTRC	variable	200	0.4	0.0001	0.3
	AddNeuroMed → HBTRC	variable	250	0.5	0.0001	0.3
	HBTRC → AddNeuroMed	variable	250	0.5	0.0001	0.3
Attention	AddNeuroMed	101	200	0.4	0.0001	0.4
	HBTRC	101	200	0.4	0.0001	0.4
	AddNeuroMed → HBTRC	101	250	0.5	0.0001	0.4
	HBTRC → AddNeuroMed	101	250	0.5	0.0001	0.4

Table 4.1: Parameter assignments of static, encoder, and attention models

First and second columns indicate variant models and data sets, respectively. Other columns of the table represent the values of parameters and hyper parameters assigned for each variant with respect to data sets.

4.4 Experiments

In this section, we present the results of applying static, encoder, and attention models i) *train and test on data sets separately*, ii) *cross validation: train on one data set and test on the other*, and iii) *attention mechanism and inferences* to assure that the model behaviour is as expected.

4.4.1 Experiment 1: Train and Test on Data Sets Separately

This section shows the results of applying the encoder and attention model on both data sets separately. Optimal hyper parameters are chosen after testing a range of values on the sensitivity of the trained model. Figure 4.4 and 4.5 show the results of different values assigned to *learning rate* $\in \{0.1, 0.01, 0.001, 0.0001, 0.00001\}$ and *dropout* $\in \{0.4, 0.5, 0.6, 0.7, 0.8\}$ applied on the variants. Firstly, we should note that in all plots, apart from learning rate and dropout rates, all parameters are fixed in order to analyse the sensitivity of the model with respect to those hyper parameters. Furthermore, ℓ_2 regularisation with $\lambda = 0.0001$ is applied to network weights except for *biases* in addition to *kernel regularisation* $= 0.3$ on the dense layer to prevent overfitting problem. For the results of the encoder model, it is apparent from the plots that some of the values dominate others in terms of performance. For instance, optimal dropout and learning rates are 0.6 and 0.001 and 0.4 and 0.001 for AddNeuroMed and HBTRC data sets, respectively. Notice that training losses in all plots show a downward trend which indicates that the network learns and minimises training errors.

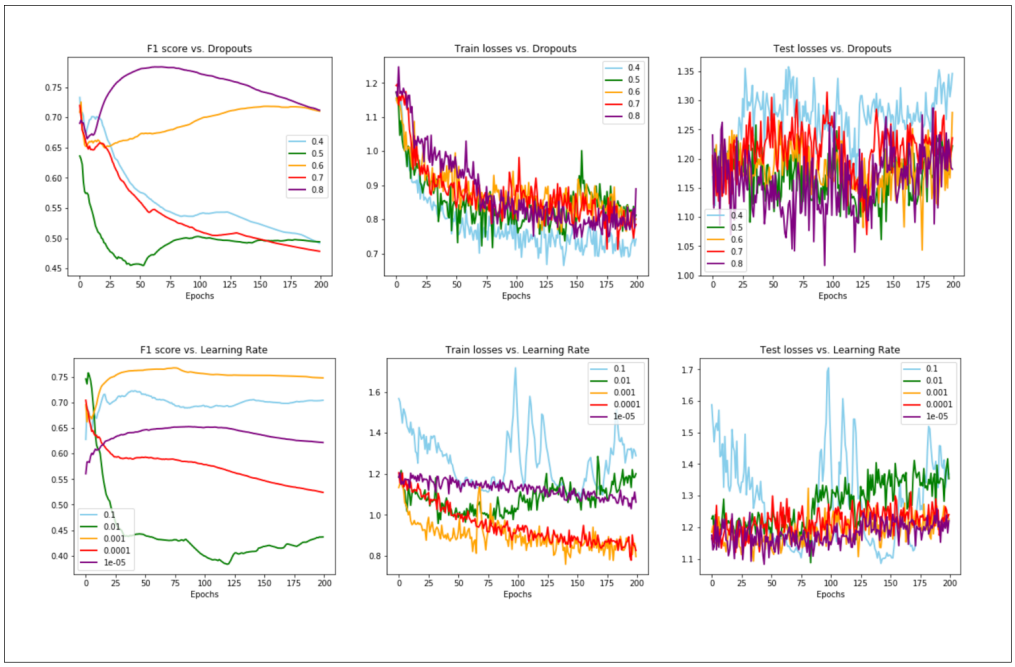
The plots obtained with the same range of *dropout* and *learning rates* by applying

attention model have been given in Figure 4.5. While the best dropout rate is same (0.4) for both data sets, the learning rate is 0.001 and 0.0001 for AddNeuroMed and HBTRC data sets, respectively. As expected, all training loss curves go down; hence, the network again adapts to the training data. We observe that the GeneInteraction DeepRNN encoder variant outperforms the attention model. The reason stems from their architectural differences. The encoder network takes sequences with variable lengths and the last relevant output is provided to the output layer; on the other hand, the attention network reads sequences with 0 paddings and all outputs as well as states are used to construct the attention layer. The results of the static model have been provided in Appendix C.

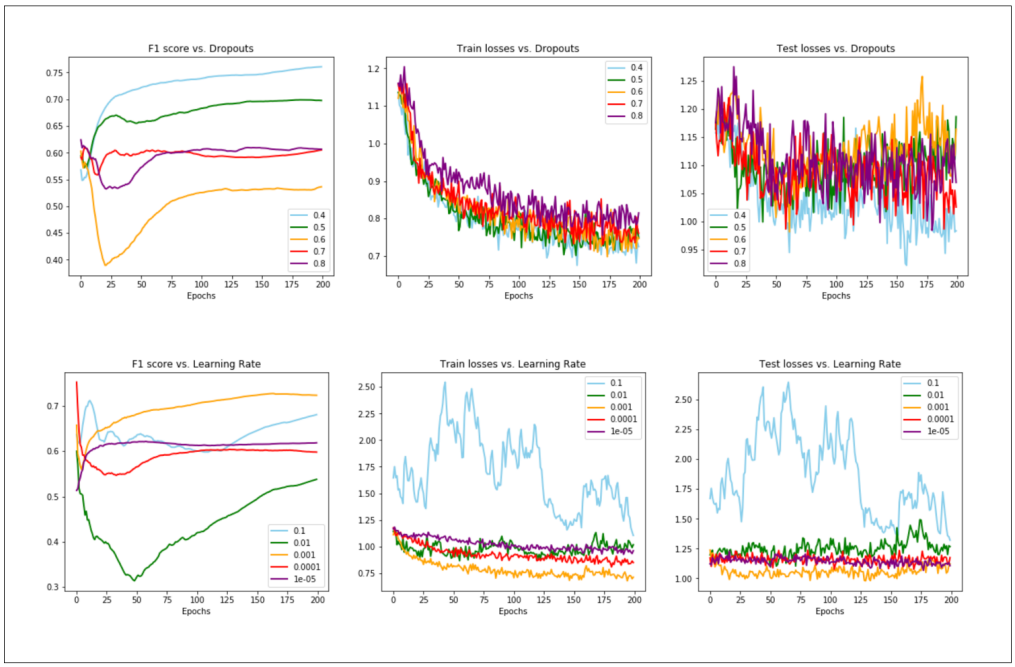
4.4.2 Experiment 2: Cross Validation

In this section, we present results obtained by training encoder and attention models on AddNeuroMed (HBTRC), and testing on HBTRC (AddNeuroMed) data sets. We aim to ensure that the GeneInteraction DeepRNN variants do not overfit and make better predictions when tested on similar data. In Figure 4.6, we should note that when networks learn on AddNeuroMed and predict on HBTRC, plots become flat smoothly after some iterations. However, for the reverse, plots go up and down before reaching the flat level. This points out that HBTRC data set has more noise than AddNeuroMed data set. The reason of having less noise in AddNeuroMed is that this data set is processed according to genetics fundamentals. For instance, when dealing with missing information in genotypes, there is a special methodology, namely *imputation*, is used. In HBTRC data set, missing data are handled by substituting with common allele encoding (i.e., 3). On the other hand, despite of the sensitivity to noise, the networks still sustain learning process and make better predictions after 50^{th} epochs.

It is worth noting that results of encoder model are still more promising than that of attention model. As mentioned above, this fact grounds on the architectural differences and cross validation on different data sets supports the idea. On the other hand, the attention model outperforms our baseline method (Refer: Table 4.5 and Appendix B for details). Besides, having an attention mechanism is crucial for researchers to interpret underlying mechanism of NNs and shed a light on gene interactions (e.g., which encodings of SNPs are more significant, whether gene lengths influence, how gene expression is associated).



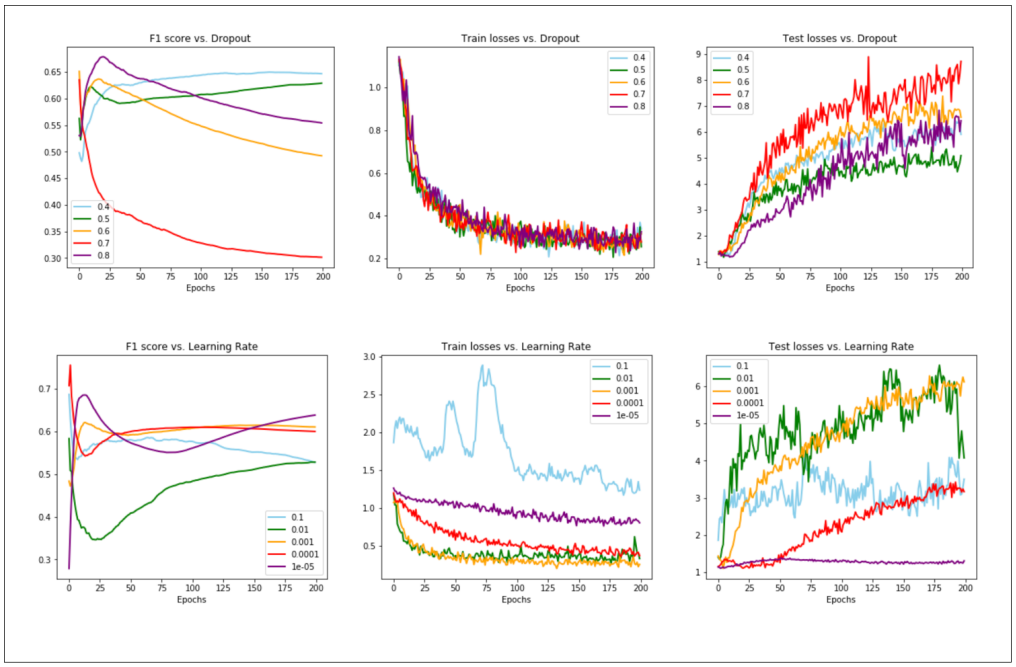
a) AddNeuroMed



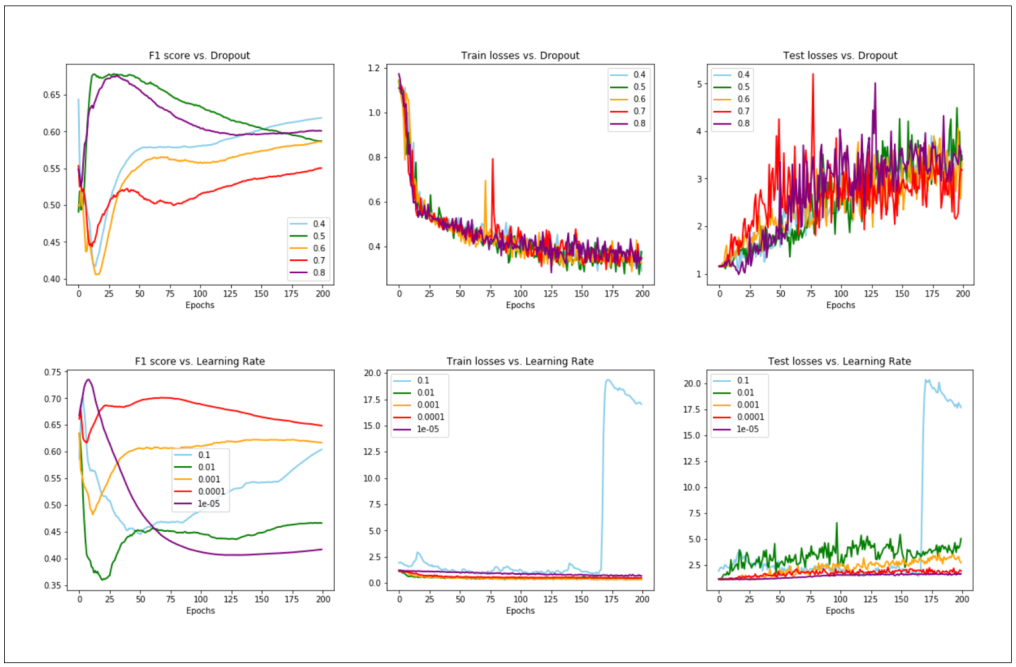
b) HBTRC

Figure 4.4: Results of encoder model with respect to different values of hyper parameters, where dropout $\in \{0.4, 0.5, 0.6, 0.7, 0.8\}$ and learning rate $\in \{0.1, 0.01, 0.001, 0.0001, 0.00001\}$

Figure represents F1 scores of the encoder model with respect to a range of dropout and learning rates. In (a), the model is tested on AddNeuroMed data set and the optimal dropout and learning rates are 0.6 and 0.001. In (b), the variant is implemented on HBTRC data set. The best dropout and learning rates are 0.4 and 0.001.



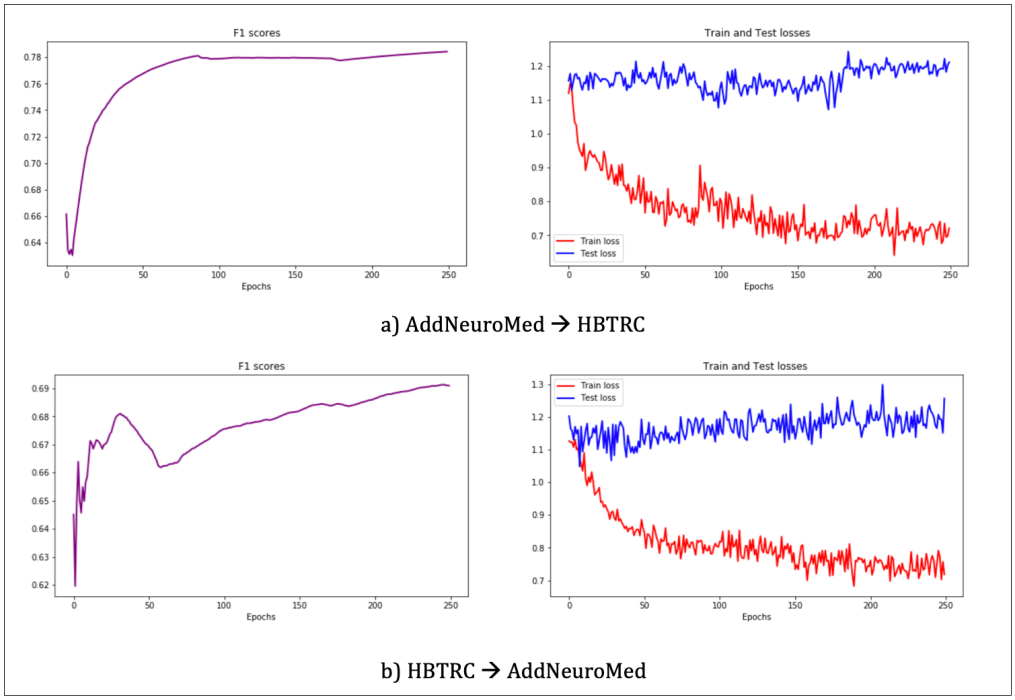
a) AddNeuroMed



b) HBTRC

Figure 4.5: Results of attention model with respect to different values of hyper parameters, where dropout $\in \{0.4, 0.5, 0.6, 0.7, 0.8\}$ and learning rate $\in \{0.1, 0.01, 0.001, 0.0001, 0.00001\}$

Figure represents F1 scores of the attention model with respect to a range of dropout and learning rates. In (a), the model is tested on AddNeuroMed data set and the optimal dropout and learning rates are 0.4 and 0.001. In (b), the variant is implemented on HBTRC data set. The best dropout and learning rates are 0.4 and 0.0001.



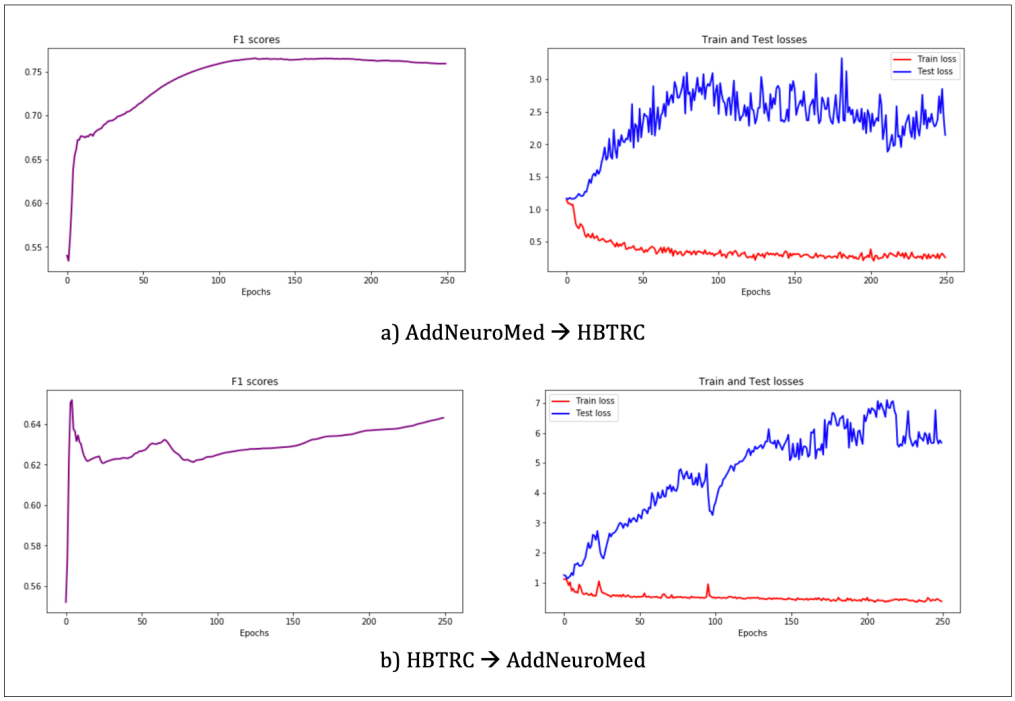


Figure 4.6: Cross validation results of encoder and attention models

First two plots show results for encoder model and plots below represent results for attention model. The models are trained on AddNeuroMed (HBTRC) and tested on HBTRC (AddNeuroMed) in (a) and (b), respectively.

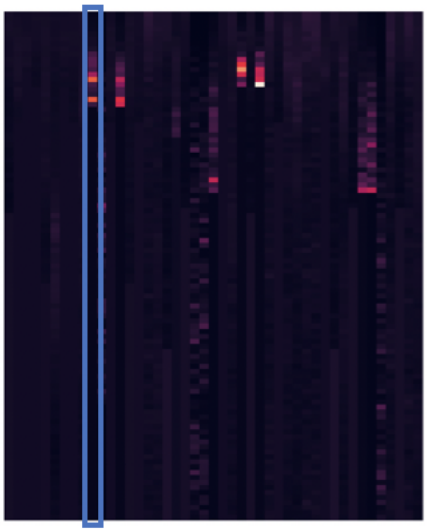
4.4.3 Experiment 3: Attention Mechanism

In this section, we examine the attention mechanism and what our network pays attention while making a classification. Figures 4.7, 4.8, and 4.9 represent the results of attention matrix from the attention model applied on AddNeuroMed data set. The tables with headings of Gene A and Gene B show the count of SNP encodings (i.e., 1, 2, 3) at every time step for given 206 subjects. The table below represents mean, maximum and minimum of gene expression levels. First, we should note that bright points on the attention matrix indicate at which points the network assigns more weight. *x-axis* denotes test examples and *y-axis* shows the number of time steps, which is also the number of attention weights assigned for every prediction. The attention weights for architecture of gene A and gene B are distinct as shown in figures. Genes have separate SNPs and gene expressions levels that affect the decision and we want to capture their distinct influence.

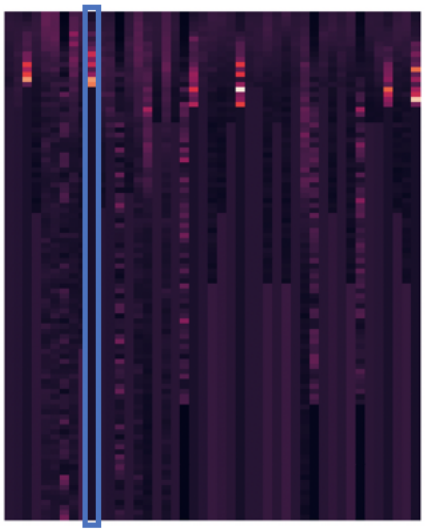
In Figure 4.7, which is the example that network predicts as +1, namely gene A regulates gene B, significant time steps are approximately between 8 and 15 for both genes. The weights are zero beyond 18^{th} and 15^{th} for gene A and gene B, which are the length of these genes, respectively. This indicates that the attention mechanism detects zero paddings and does ignore padding frames. It is also apparent from tables that attention weights at time steps with SNP encodings of 1 and 2 dominate those with SNP encoding 3 because encodings of 1 and 2 are less frequent types as opposed to encoding 3. This is true for Figure 4.8 especially for gene B where the number of time steps with encodings of 1 and 2 are abundant (i.e., 88 out of 101 time steps). Moreover, we observe that another indicative feature for classification is the length of genes. In order the network to predict no interaction between genes, we believe that encodings of 1 or 2 should be dense in both genes with high number of SNPs. Otherwise, the existence of encodings of 1 and 2 indicate a gene regulation between genes.

Output	Gene A	Gene B
+1	101	67
-1	19	101
-1	101	101

Table 4.2: Lengths of genes



a) Attention Matrix for Gene A



b) Attention Matrix for Gene B

Gene A

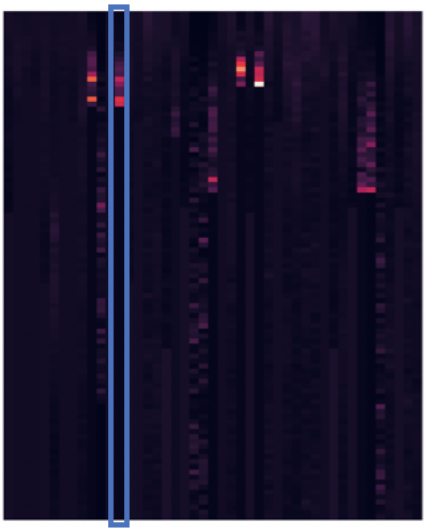
	Count 1	Count 2	Count 3
8	58	92	56
9	6	72	128
10	0	0	206
11	0	1	205
12	0	0	206
13	0	1	205
14	6	73	127
15	23	84	99

Gene B

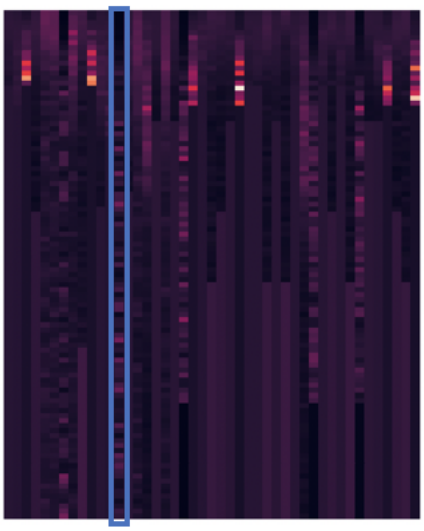
	Count 1	Count 2	Count 3
8	1	19	186
9	3	21	182
10	0	0	206
11	1	4	201
12	0	2	204
13	9	77	120
14	9	76	121
15	0	0	0

Gene A - max	Gene B - max	Gene A - mean	Gene B - mean	Gene A - min	Gene B - min
2.572461	2.606738	0.071207	-0.089090	-3.334071	-2.964612

Figure 4.7: Attention matrix for AddNeuroMed data set (+1 labelled example)



a) Attention Matrix for Gene A



b) Attention Matrix for Gene B

Gene A

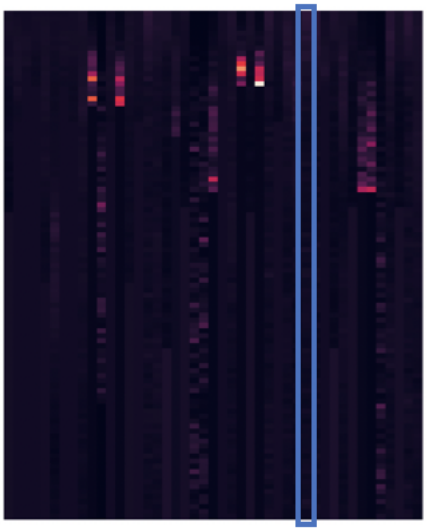
	Count 1	Count 2	Count 3
13	0	1	205
14	6	73	127
15	23	84	99
16	0	1	205
17	0	0	206
18	22	83	101

Gene B

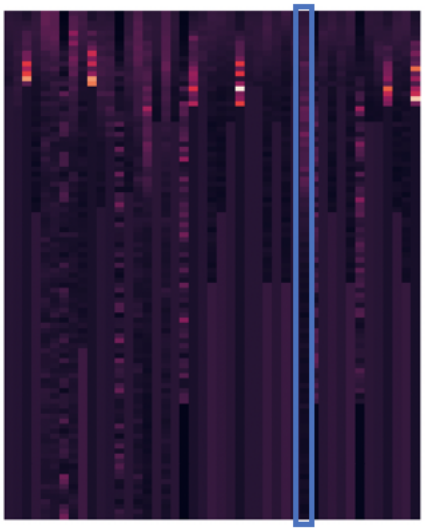
	Count 1	Count 2	Count 3
6	23	91	92
7	29	97	80
13	23	91	92
26	50	102	54
83	24	92	90
86	28	86	92
97	39	89	78

Gene A - max	Gene B - max	Gene A - mean	Gene B - mean	Gene A - min	Gene B - min
2.572461	2.515578	0.071207	-0.027390	-3.334071	-2.853252

Figure 4.8: Attention matrix for AddNeuroMed data set (-1 labelled example)



a) Attention Matrix for Gene A



b) Attention Matrix for Gene B

Gene A

	Count 1	Count 2	Count 3
6	27	105	74
7	26	93	87
9	33	105	68
10	1	21	184
14	26	94	86
23	26	93	87
84	25	98	83
100	8	64	134

Gene B

	Count 1	Count 2	Count 3
6	0	1	205
7	0	20	186
9	0	17	189
10	0	17	189
14	0	20	186
23	0	1	205
84	0	20	186
100	0	8	198

Gene A - max	Gene B - max	Gene A - mean	Gene B - mean	Gene A - min	Gene B - min
2.945235	2.399947	0.020163	0.016875	-4.114111	-2.590793

Figure 4.9: Attention matrix for AddNeuroMed data set (0 labelled example)

4.4.4 Performance Analysis

In this section, we report the performance of the GeneInteraction DeepRNN based on two parts:

- Variants' (i.e., static, encoder, attention) accuracy with respect to each other
- NNs' success in detecting against the baseline statistical model

The detailed results have been provided in Table 4.5. We can deduce following insights from the results:

1. Firstly, let us analyse performances of the GeneInteraction DeepRNN variants with respect to eQTL + LD analysis. All variants achieve F1 scores between 60% and 78%, whereas eQTL + LD analysis has 33%. This indicates that eQTL + LD analysis cannot detect any of interactions in our test data and labels them all as "no interaction". The reason is that eQTL uses a linear regression between genotypes and gene expressions. Then, LD calculates the correlation between variants to identify if we need to eliminate any pairs of variants with high LD (i.e., variants above threshold ($R^2 > 0.6$) depicted in Figure B.1 for within-pair LD). Conversely, our models are constructed based on multiple "non-linearity" principle. For instance, we have different representations for gene A and gene B by using LSTM cells. An LSTM cell is composed of *tanh* and *sigmoid* layers which are regulated by gates for different tasks. Additionally, for the attention model, we have another non-linear (i.e., *tanh*) layer for the attention mechanism. This shows that the best network (or analysis) to identify gene interactions lies at the non-linearity principle.

2. It is worth mentioning that there is a slight drop in or an increase in the performance for training on HBTRC and testing on AddNeuroMed as opposed to training and testing on HBTRC. However, there is a noticeable increase in that for AddNeuroMed. As mentioned before, the main reason behind this can be how different methodologies are applied for the missing values, which misleads the network. Furthermore, another reason can be data from blood samples (as in AddNeuroMed) is more descriptive for learning process than data from brain tissues (as in HBTRC). Therefore, a possible solution can be to let the network train on data sets with blood samples and make predictions on any type of tissue.

3. We observe that for AddNeuroMed data set the order of performance is: *Encoder* $>$ *Attention* $\approx$ *Static* $>$ *eQTL + LD*. For HBTRC data set: *Encoder* $>$ *Static* $>$

Attention > *eQTL + LD*. For AddNeuroMed → HBTRC: *Encoder* > *Attention* > *Static*. Finally, for HBTRC → AddNeuroMed: *Encoder* > *Attention* ≈ *Static*. Summary of results have been illustrated in Figure 4.10 and 4.11. We have not included any eQTL + LD results for cross validation since eQTL + LD analysis cannot be implemented on different data sets *per se*. Here, *Model A* > *Model B* means that model A is better than model B in terms of F1 score. Model A ≈ Model B denotes that model A has similar F1 score as model B. It is important to note that encoder model outperforms other models. A potential reason behind this performance can be that the network learns merely on the relevant sequence. Additionally, the network make predictions based on the relevant output (i.e., the last output obtained at the previous time step of the first padding frame) produced by LSTM. On the other hand, the attention model incorporates states and outputs attained by LSTM cells. Even though the network recognise and assigns zero weight to padding frames, states and outputs might also create a noise for the network. Therefore, a possible solution would be to construct a modified attention mechanism that is able to handle variable sequence lengths.

4. It should be noted that optimal values for hyper parameters are chosen by having a sensitivity analysis. Tables 4.3 and 4.4 represent F1 scores with respect a range of dropout and learning rates (Refer: Figure C.2, 4.4, and 4.5 for graph illustrations). Bold results show best F1 scores for given dropout and learning rates. Notice that we obtain slightly different F1 scores when we assign same values to these rates. The reason is that models shuffle data sets and learn in batches; hence, the training is done on slightly different examples. Table 4.5 shows the average of these F1 scores for each model. Moreover, some models perform better with other learning rates as well. For instance, attention model trained and tested on HBTRC achieves 64.8% and 61.7% when we assign 0.0001 and 0.001 to the learning rate, respectively. However, we fix learning rates in entire models to 0.001 since most of models obtain high accuracy and we adjust dropout rates as models are constructed by using DropoutWrappers.

5. Finally, we notice that the static model performs almost as good as the attention model in terms of F1 score. Considering that both also takes a fixed sized sequence with paddings, they make better predictions (e.g., F1 scores $\approx 60-75\%$) as compared to eQTL + LD analysis. One possible reason can be that the attention model is heavily regularised because the model has one more non-linear layer which adds more capacity to the network. Hence, we assign more ℓ_2 regularisation

constant. However, in terms of computations times, the attention model takes less time than the static model. The reason is how different the models create their networks. For instance, the static model creates an unrolled graph for a fixed sequence length. When we call $tf.nn.static\_rnn$ for inputs with 101 time steps, we construct a graph with 101 time steps in advance. On the other hand, the attention model is based on $tf.nn.dynamic\_rnn$ that uses a loop to create the graph on-the-fly. The computation time necessary for running the static and attention models 30 minutes and 20 minutes, respectively.

Briefly, we have framed architectures for the GeneInteraction DeepRNN variants, which we proposed in Chapter 3, implemented on AddNeuroMed and HBTRC data sets. These models can be used for the detection of gene-gene interactions on different data sets with various tissues and samples. In the next chapter, we present our conclusions about our work and discuss further research possibilities.

Model	Data Set	Learning Rate	d = Dropout Rate				
			0.4	0.5	0.6	0.7	0.8
Static	AddNeuroMed	0.001	43.3%	39.5%	**65.3%**	58.7%	46.8%
	HBTRC	0.001	54.4%	60.1%	**62.9%**	**64.6%**	52.4%
Encoder	AddNeuroMed	0.001	49.1%	49.4%	**71.0%**	47.8%	**71.1%**
	HBTRC	0.001	**76.8%**	69.7%	53.6%	60.5%	60.5%
Attention	AddNeuroMed	0.001	**64.7%**	62.9%	49.2%	30.1%	55.4%
	HBTRC	0.001	**61.8%**	58.7%	58.6%	55.0%	60.0%

Table 4.3: F1 scores of static, encoder, and attention models with respect to different dropout rates

First and second columns indicate variant models and data sets, respectively. Third column of the table represent assigned learning rates to test sensitivity of models with respect to a range of dropout rates $\in \{0.4, 0.5, 0.6, 0.7, 0.8\}$.

Model	Data Set	Dropout	η = Learning Rate				
			0.1	0.01	0.001	0.0001	0.00001
Static	AddNeuroMed	0.6	2.3%	55.9%	**62.3%**	47.4%	52.3%
	HBTRC	0.6	$\approx 0\%$	52.3%	**66.7%**	13.9%	44.6%
Encoder	AddNeuroMed	0.6	70.4%	43.6%	**74.8%**	52.4%	62.1%
	HBTRC	0.4	68.0%	53.7%	**72.3%**	59.8%	61.8%
Attention	AddNeuroMed	0.4	52.7%	52.7%	**61.0%**	59.9%	**63.7%**
	HBTRC	0.4	60.4%	46.6%	**61.7%**	**64.8%**	41.7%

Table 4.4: F1 scores of static, encoder, and attention models with respect to different learning rates

First and second columns indicate variant models and data sets, respectively. Third column of the table represent assigned dropout rates to test sensitivity of models with respect to a range of learning rates $\in \{0.1, 0.01, 0.001, 0.0001, 0.00001\}$.

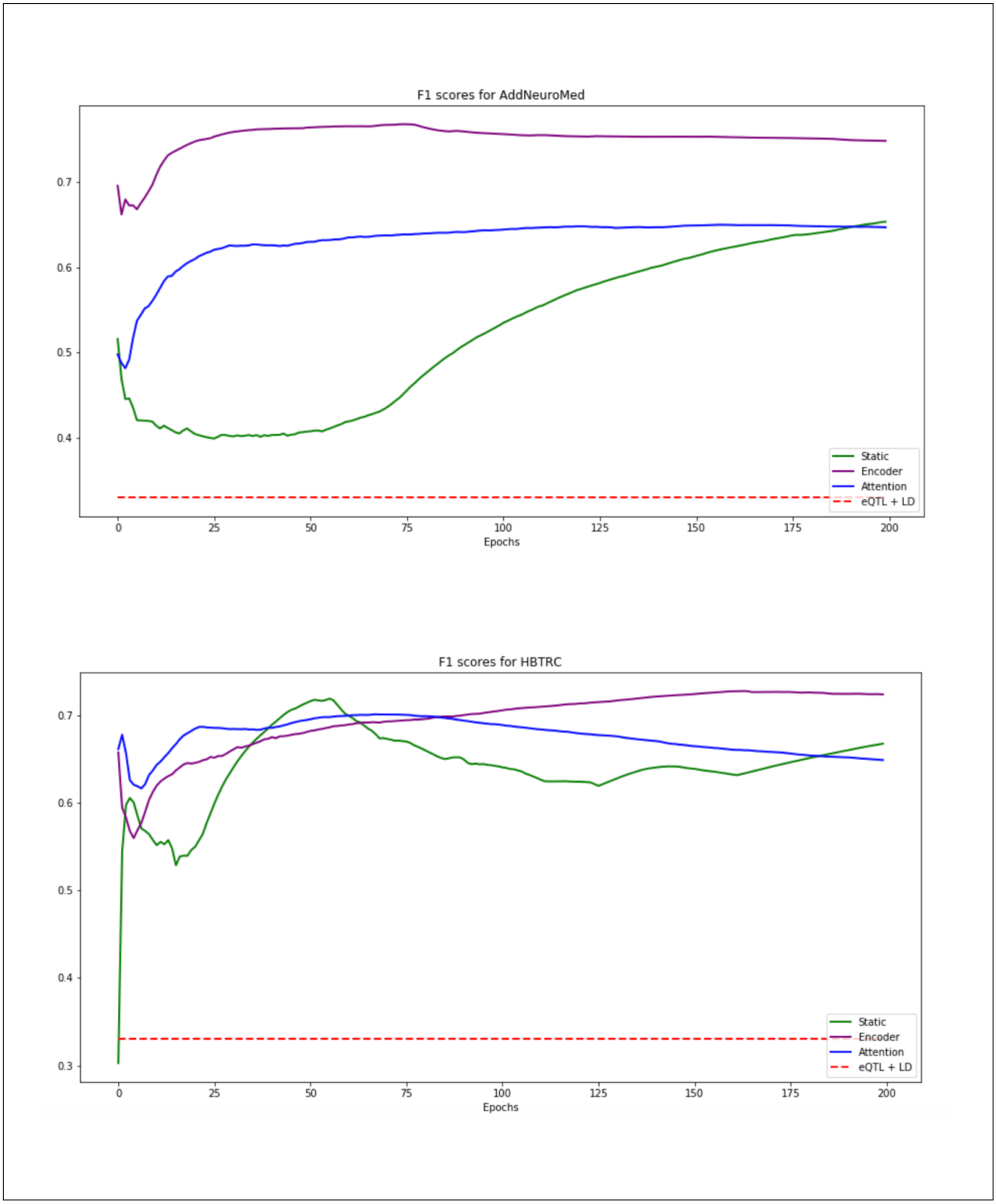


Figure 4.10: Results' summary of 1^{st} experiment on AddNeuroMed and HBTRC data sets

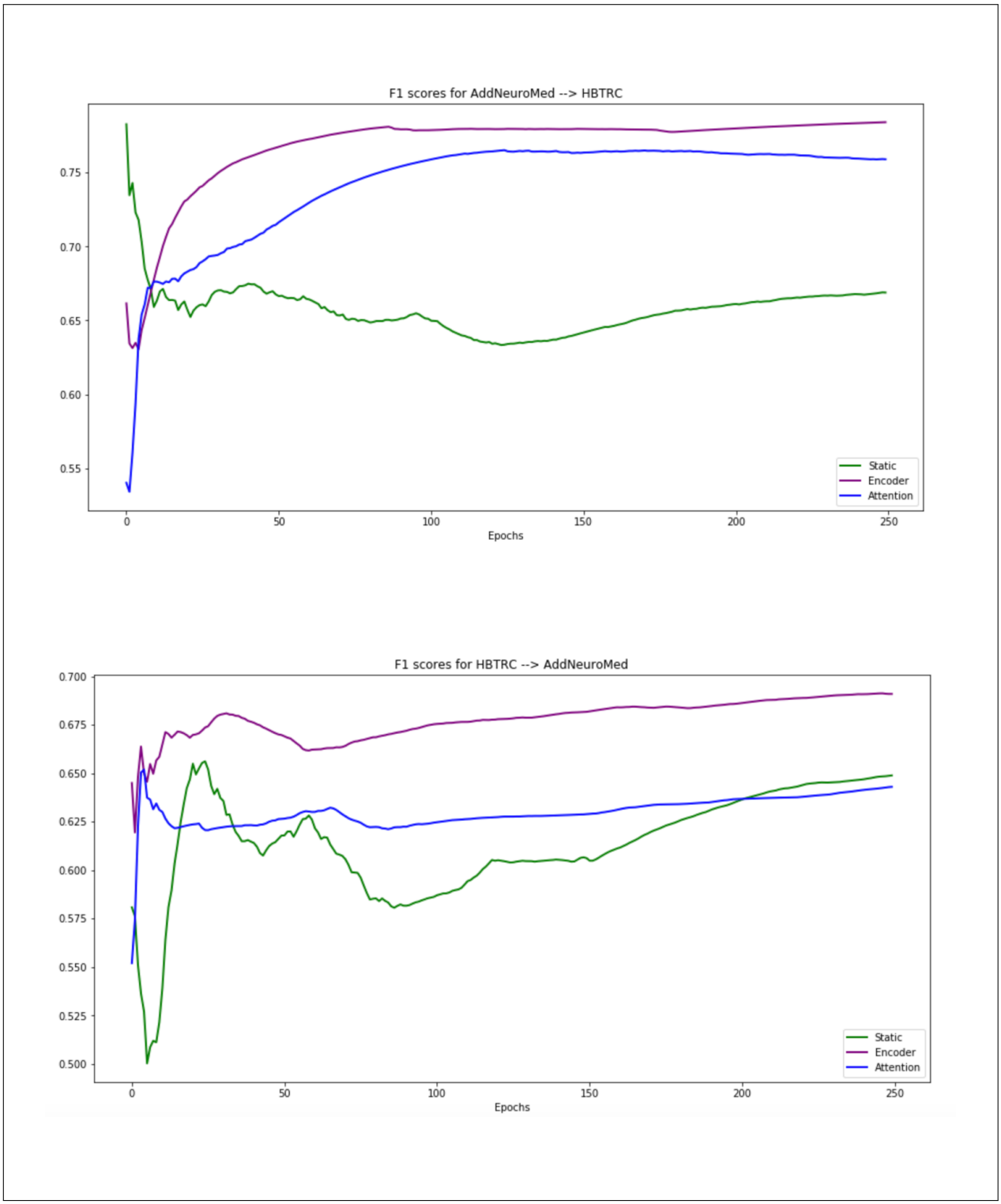


Figure 4.11: Results' summary of 2^{nd} experiment on AddNeuroMed and HBTRC data sets

Model	Data Set	F1 Score
eQTL + LD	AddNeuroMed	33.3%
	HBTRC	33.3%
Static	AddNeuroMed	63.8%
	HBTRC	64.8%
	AddNeuroMed → HBTRC	66.9%
	HBTRC → AddNeuroMed	64.9%
Encoder	AddNeuroMed	**71.7%**
	HBTRC	**74.5%**
	AddNeuroMed → HBTRC	**78.4%**
	HBTRC → AddNeuroMed	**69.1%**
Attention	AddNeuroMed	62.8%
	HBTRC	61.8%
	AddNeuroMed → HBTRC	75.9%
	HBTRC → AddNeuroMed	64.3%

Table 4.5: Evaluation results

Evaluation results obtained by the GeneInteraction DeepRNN variants (static, encoder, and attention) and our baseline statistical model (eQTL analysis with LD) using performance measure metric: F1 score

Chapter 5

Conclusions

In this chapter, we briefly define the contribution of our work and analyse how we completed the project goals. We also propose potential extensions which can be taken into account for future research.

5.1 Summary and Contributions

In Chapter 3, we constructed a new architecture in order to produce different representations for gene A and gene B and we call the architecture *GeneInteraction DeepRNN*. We incorporated fundamental frameworks into the network: i) LSTMs [4], ii) Encoder module [5], iii) Attention mechanism [6] as well as regularisation frameworks iv) Dropouts [42, 7]. Furthermore, we have presented eQTL analysis [1] where we applied MatrixEQTL [28] with LD [31] as our baseline method. We decided to use F1 score to evaluate performance in order to include the number of TP, TN, FN, and FP.

In our experiments, we have applied GeneInteraction DeepRNN on two data sets (AddNeuroMed and HBTRC) and presented the results in Chapter 4. We notice that our models perform quite well in terms of F1 score when comparing to the baseline approach (with the best performing having a score of 74.5% as opposed to 33% in baseline method when applied on our data sets). In Chapter 4, we demonstrated insights which researchers can assess and compare other gene interaction models. Thus, we achieved the thesis goals which we listed in Chapter 1.

5.2 Future Research

In the field of epidemiology, the subject of gene-gene interactions and a model development aiming to detect these interactions are gaining more and more popularity with the uprising number of complex diseases. With this work, we intend to produce a thorough model that researchers will have another approach evaluating the existence of regulation. Having said that, there are multiple extensions of this work which can be implemented in future research endeavours. Some of them are as follows:

1. **Cross validation on larger data sets:** In this work, we have used AddNeuroMed and HBTRC data sets for all our experiments. We have assessed models based on F1 scores in detecting gene-gene interactions within and across these

data sets. We obtain high F1 scores for all models as opposed to statistical approaches; however, data sets we use in experiments are fairly small and various imputation methods are applied in order to fill missing genotype values. When training model variants, there is a possibility of encountering with overfit problems; hence, we use various regularisation methods (e.g., dropout, ℓ_2 for model weights except for biases, and another regularisation in dense layer) to prevent it. When the model is trained on a larger data set for longer training steps, our work would become more robust and accurate in making predictions for gene pairs. For instance, one of the significant data set used in epidemiology field is the Religious Orders Study and Memory and Aging Project (ROSMAP) study. ROSMAP, funded by Rush University, is a longitudinal clinical study carried out on AD patients (approximately 1700 people) and collects their genotypes, phenotypes, covariates, and gene expression information.

2. **Classification on regular genes:** In this work, we only consider detecting regulations between genes carrying special proteins, namely TFs and targets. We can test our model having trained on a larger data set to predict an gene-gene interactions between non-special genes and present possible gene pairs for researchers to prove the presence of their interactions.

3. **Comprehensive tool for all types of tissues:** Finally, in this work we have developed a network which aims to explore interactions on brain tissue and blood sample data sets. Having different data sets from brain, muscle, heart, lung tissues, etc., one can create a comprehensive tool trained across all data sets in order to expose APIs which can help researchers retrieve any interaction either within a specific tissue or across all tissues.

In conclusion, we hope that our work will enable researchers to find out undiscovered gene interactions.

Bibliography

[1] A.E. Fish; J.A. Capra; and W.S. Bush. Are interactions between cis-regulatory variants evidence for biological epistasis or statistical artifacts? *American Journal of Human Genetics*, 99(4):817–830, 2016.

[2] F. Günther; N. Wawro; and K. Bammann. Neural networks for modeling gene-gene interactions in association studies. *BMC Genet.*, 9:87, 2009.

[3] A.A. Motsinger-Reif; S.M. Dudek; L.W. Hahn; and M.D. Ritchie. Comparison of approaches for machine-learning optimization of neural networks for detecting gene-gene interactions in genetic epidemiology. *Genetic Epidemiology*, 32:325–340, 2008.

[4] S. Hochreiter; and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[5] I. Sutskever; O. Vinyals; and Q.V. Le. Sequence to sequence learning with neural networks. In *Proc. NIPS*, Montreal, CA, 2014.

[6] D. Bahdanau; K. Cho; and Y. Bengio. Neural machine translation by jointly learning to align and translate. ICLR, 2015.

[7] W. Zaremba; I. Sutskever; and O. Vinyals. Recurrent neural network regularization. ICLR, 2015.

[8] P.M. Sham; and S.S. Cherny. Analysis of complex disease association studies. chapter Genetic architecture of complex disease, pages 1–13. 2010.

[9] N.R. Wray; M.E. Goddard; and P.M. Visscher. Prediction of individual genetic risk of complex disease. *ScienceDirect*, 18:257–263, 2008.

[10] J.H. Moore. The ubiquitous nature of epistasis in determining susceptibility to common human diseases. *Human Heredity*, 56:73–82, 2003.

[11] R.J. Greenspan. The flexible genome. *Nature Review Genetics*, 2:383–387, 2001.

[12] T.N. Griebel; P.G. Ferreira. et al. T. Lappalainen; M. Sammeth; M.R. Friedlander; P.A. Hoen; J. Monlong; M.A. Rivas.; M. Gonzalez-Porta; Kurbatova. Transcriptome and genome sequencing uncovers functional variation in humans. *Nature*, 501:506–511, 2013.

[13] D. Welter; J. MacArthur; T. Burdett; P. Hall; H. Junkins; A. Klemm; P. Flicek; T. Manolio; L. Hindorff; et. al. The nhgri gwas catalog, a curated resource of snp-trait associations. 42:1001–1006, 2014.

[14] M.T. Maurano; R. Humbert; R.E. Thurman; E. Haugen; H. Wang; A.P. Reynols; R. Sandstrom; H. Qu; J. Brody; et.al. Systematic localization of common disease-associated variation in regulatory dna. *Science*, 337:1190–1195, 2012.

[15] H.J. Cordell. Detecting gene-gene interactions that underlie human diseases. *Genetics*, 10(6):392–404, 2009.

[16] S.H. Chen; J. Sun; L. Dimitrov; A.R. Turner; T.S. Adams; D.A. Meyers; B.L. Chang; S.L. Zheng; H. Gönberg; J. Xu and F.C. Hsu. A support vector machine approach for detecting gene-gene interaction. *Genetic Epidemiology*, 32:152–167, 2008.

[17] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588, 1997.

[18] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[19] M.D. Ritchie; L.W. Hahn; N. Roodi; L.R. Bailey; W.D. Dupont; F.F. Parl; and J.H. Moore. Multifactor-dimensionality reduction reveals high- order interactions among estrogen-metabolism genes in sporadic breast cancer. *Am J Hum Genet*, 69:138–147, 2001.

[20] L.W. Hahn; M.D. Ritchie; and J.H. Moore. Multi-factor dimensionality reduction software for detecting gene-gene and gene-environment interactions. *Bioinformatics*, 19:376–382, 2003.

[21] M.R. Nelson; S.L.R Kardia; R.E. Ferrell; and C.F. Sing. A combinatorial partitioning method to identify multilocus genotypic partitions that predict quantitative trait variation. *Genome Res.*, 11:458–470, 2001.

[22] J. Millstein; D.V. Conti; F.D. Gilliland; and W.J. Gauderman. A testing framework for identifying susceptibility genes in the presence of epistasis. *Am J Hum Genet*, 78:15–27, 2006.

[23] N.R. Cook; R.Y.L. Zee; and P.M. Ridker. Tree and spline-based association analysis of gene-gene interaction models for ischemic stroke. *Stat Med*, 23:1439–1453, 2004.

[24] I. Ruczinski; C. Kooperberg; and M. LeBlanc. Logistic regression. *Comput. Graph Stat.*, 12:475–511, 2003.

[25] R. Tibshirani. Regression shrinkage and selection via the lasso. *Royal Statistical Society*, 58:267–288, 1996.

[26] D. Rumelhart; G.E. Hinton; and R.J. Williams. Learning representations by back-propagating erros. *Nature*, 323:533–536, 1986.

[27] B. Servin and M. Stephens. Imputation-based analysis of association studies: candidate regions and quantitative traits. *PLoS Genet.*, 3:114, 2007.

[28] A.A. Shabalin. Matrix eqtl: ultra fast eqtl analysis via large matrix operations. *Bioinformatics*, 28(10):1353–1358, 2012.

[29] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Royal Statistical Society*, 57(1):289–300, 1995.

[30] R.A. Fisher. *Statistical Methods for Research Workers*. Oliver and Bond, 1925.

[31] M. Slatkin. Linkage disequilibrium understanding the evolutionary past and mapping the medical future. *Nat Rev Genet.*, 9(6):477–485, 2008.

[32] F. Rosenblatt. The perceptron: A probalistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[33] K. Murphy. *Machine Learning*. MIT Press, 2012.

[34] S. Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*. PhD thesis, Technische Universität München, Arcisstr. 21, 8000 München 2, Germany, 6 1991.

[35] Y. Bengio; P. Simard; and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks*, 5(2):157–166, 1994.

[36] L. Thomas; M.A. Ferreira; D. Bender; et. al S. Purcell; B. Neale; K. Todd-Brown. Plink: a tool set for whole-genome association and population-based linkage analyses. *Am J Hum Genet.*, 81(3):559–575, 2007.

[37] G. Li; D. Jima; F.A. Wright; and A.B. Nobel. Ht-eqtl: integrative expression quantitative trait loci analysis in a large number of human tissues. *BMC Bioinformatics*, 19(95), 2018.

[38] D.M. Gatti; A.A. Shabalin; T-G. Lam; F.A. Wright; I. Rusyn; and A.B. Nobel. Fastmap: Fast eqtl mapping in homozygous populations. *Bioinformatics*, 25(4):482–489, 2009.

[39] V. Emilsson; G. Thorleifsson; B. Zhang; A.S. Leonardson; F. Zink; J. Zhu; S. Carlson; A. Helgason; G.B. Walters; S. Gunnarsdottir; et al. Genetics of gene expression and its effect on disease. *Nature*, 452:423–428, 2008.

[40] E. Grundberg; T. Kwan; B. Ge; K.C.L. Lam; V. Koka; A. Kindmark; H. Mallmin; J. Dias; D.J. Verlaan; M. Ouimet; et al. Population genomics in a disease targeted primary cell model. *Genome Res.*, 19:1942–1952, 2009.

[41] R.R. Hudson. Linkage disequilibrium and recombination. In D.J. Balding DJ; M. Bishop; C. Cannings, editor, *Handbook of Statistical Genetics*, chapter 11, pages 309–324. Wiley, 2001. In: D.J. Balding DJ; M. Bishop; C. Cannings editors.

[42] N. Srivistava. *Improving neural networks with dropout*. PhD thesis, University of Toronto, 2013. PhD thesis.

[43] J. Bayer; C. Osendorfer; N. Chen; S. Urban; and P. van der Smagt. On fast dropout and its applicability to recurrent networks, 2013.

Appendix A

Methods for GeneInteraction DeepRNN

The code of training for this project has two sections: i) the *gene network creation* method which is responsible for creating tensors on a given Tensorflow placeholders on both data sets and ii) the *gene interaction code* which contains training code for variants of GeneInteraction DeepRNN. The steps are as follows:

1. *Load AddNeuroMed / HBTRC data sets*: The model starts by loading +1 / -1 / 0 labelled examples for training and testing data generated from these data sets. Since outputs of preprocessData are saved into *pickle* format files, it is easy to retrieve all variables.

2. *Call Gene Network Creation Methods*: The graph of the GeneInteraction DeepRNN is generated by using utility and LSTM network methods, which are different for each variant.

3. *Train model*: The static, encoder, and attention models are trained on AddNeuroMed / HBTRC data sets. The gene interaction codes can be found in *trainRNN.ipynb* specified for versions as *Static, Encoder, Attention*. Also, helper methods used in training are presented in *trainRNN_utils.py*.

4. *Evaluate*: The models use F1-score to evaluate trained model. The results are stored as lists.

5. *View results*: The method plots the metric and it is finally presented. Methods can be found in *trainRNN_plot_utils.py*.

In the next section, we present the crucial methods exposed by variants in step 2 in detail.

A.1 Gene Network Creation Methods

The main purpose of the gene network methods is to define each variant based on their differences which can help model train on data sets robustly and perform better for gene-gene interactions. Entire gene network creation methods can be found in *trainRNN_network_utils.py*. The significant ones presented in the network creation have been described below.

A.1.1 Variable Sequence Length

There are two important methods *length(sequence)* and *last_relevant(output, length)* which return the length of every input sequence (i.e., the length before 0 paddings) and the last relevant output which LSTM cells have after the last layer. In the length method, we present an input having a fixed size, and it extracts the length of each sequence. We provide a training batch as a single tensor of shape (batch size x time steps x number of subjects). Length(sequence) collapses the last dimension into scalars using maximum as each sequence is a vector of scalar having 0 paddings in the end. Then, tf.sign() converts the actual vectors from maximum values to ones as this introduces a binary mask including ones for non-zero values and zeros for paddings. By using the binary mask, the method retrieves the length by summing the number of ones. The output length from this method is fed into the sequence_length argument of tf.nn.dynamic_rnn. tf.nn.dynamic_rnn assigns zero vectors for states and outputs of corresponding padding frames. When we use the last output, it includes only zero vectors for the last time step and is irrelevant for the training. In last_relevant(output, sequence_length) method, we provide an output having also a fixed size vector as a tensor of shape (batch size x time steps x number of units). It flattens the output tensor to another tensor in shape (examples x number of units), where examples = batch size x time steps. Next, we create an index tensor that initiates the starting points for each example with tf.range(0, batch_size) * time_steps and increases the starting point by individual length using the length method. Finally, tf.gather constructs the actual index. Notice that we do not do masking for the cost functions as the network assigns zero vectors to paddings when predicting; hence, paddings are ideal for predictions. The code for the variable length sequence methods is presented below:

```
def length(sequence):
    binary_mask = tf.sign(tf.reduce_max(tf.abs(sequence),2))
    sequence_length = tf.reduce_sum(binary_mask, 1)
    sequence_length = tf.cast(sequence_length, tf.int32)
    return sequence_length

def last_relevant(output, sequence_length):
    num_examples = tf.shape(output)[0]
    time_steps = tf.shape(output)[1]
    n_hidden = int(output.get_shape()[2])
```

```
    last_index = tf.range(0, num_examples) * time_steps +
                 (sequence_length - 1)
    flat = tf.reshape(output, [-1, n_hidden])
    last_relevant_output = tf.gather(flat, last_index)
    return last_relevant_output
```

Listing 1: Variable sequence methods

A.1.2 Attention Mechanism

The attention mechanism adds another layer to calculate importance of time steps and to which points the network pays attention more. We use an attention method introduced by [6], which includes three components: calculation of scores, attention weights, and context vectors. Two inputs of this methods are hidden states and hidden outputs, tensors having shapes of (batch size x number of units) and (batch size x time steps x number of units), respectively. Firstly, as score = FC(tanh(FC(O) + FC(H))), where FC: fully connected, O: hidden output, H: hidden state, we add another dimension to the hidden state to indicate the time step. Then, the output of tanh is passed through another dense layer for the score calculation. Having a score with a shape of (batch size x sequence length x 1), attention weights are calculated by softmax function. We apply the function on the second axis (axis = 1) because there are as many attention weights as time steps and we are trying to assign a weight to inputs at every time step. Context vector is a multiplication of attention weights x hidden output, followed by tf.reduce_sum on axis = 1 since only time steps matter. It is worth noting that we construct separate LSTM cell architectures for gene A and gene B mentioned in Chapter 3 and this implies that context vectors and attention weights are distinct for both genes. Before passing those to the dense layer for the final layer, context vectors are concatenated in order to be evaluated at the same time while making a prediction. The code is provided as below:

```
def attention(state, output, n_hidden):
    state_with_time_axis = tf.expand_dims(state, 1)
    score = tf.layers.dense(
        tf.nn.tanh(tf.layers.dense(output, n_hidden) +
                   tf.layers.dense(state_with_time_axis,
                                   n_hidden)), 1)
```

```
    attention_weights = tf.nn.softmax(score, axis=1)
    context_vector = attention_weights * output
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

Listing 2: Attention method

Appendix B

MatrixEQTL and Linkage Disequilibrium

The purpose of MatrixEQTL and LD methods provided in this section is to ensure that the reader has a better understanding of the baseline model applied in our work in details. The code for the statistical analysis is composed of three parts: i) *MatrixEQTL*, ii) *Extraction of cis-eQTLs*, and iii) *Linkage Disequilibrium* that are implemented on **R/Rstudio** as it has supporting packages called "MatrixEQTL" and "genetics". The methods have been introduced below.

B.1 MatrixEQTL

The code for implementing both analysis over AddNeuroMed and HBTRC is present in *linkageDisequilibrium.R* file. The first part deals with loading necessary files which are prepared in preprocessData step defined in section 4.2.2. The file paths are generated from .csv files on our CPU machine, but we provide those .csv files; hence, one can implement analysis by own file paths. Before applying these functions, all files should be in the right format. For instance, columns of SNPs and genes expression levels should have identical subject names. Rows of gene locations and gene expression levels as well as SNP locations and SNP encodings should be the same. MatrixEQTL has the edge on computation time as it uses blocks of variables up to 10,000. Having altered shapes and features of variables in order to obtain them in the correct form for eQTL analysis, we create sliced matrices from *genesValue* and *snpsValue*. In our work, we select ANOVA model because it treats SNPs as classifications instead of continuous values (Refer: Section 2.1.2). The code for the eQTL analysis is presented below:

```
# Load gene_expression, genotype, probe_loc, and snp_loc data
# Gene names: (R x 1)
geneNames = read.csv(filepath, header=TRUE)
geneNames <- as.matrix(geneNames)

# Subject names: (S x 1)
subjectNames = read.csv(filepath, header=TRUE)
subjectNames <- as.matrix(subjectNames)

# SNP names: (N x 1)
```

```
snpNames = read.csv(filepath, header=TRUE)
snpNames <- as.matrix(snpNames)

# Genes value: Matrix with gene expressions (R x S)
genesValue = read.csv(filepath, header=TRUE)

# SNP value: Matrix with SNP encodings (N x S)
snpsValue = read.csv(filepath, header=TRUE)

# probePos: Table with gene_id, chromosome number, start and
# end locations (R x 4)
probePos = read.csv(filepath, header=TRUE)

# snpPos: Table with snp_id, chromosome number, and position
# (N x 3)
snpPos = read.csv(filepath, header=TRUE)

# snpToGene: Table with SNPs and corresponding genes
# 1st col: snpNames, 2nd col: gene_id, (N x 2)
snpToGene = read.csv(filepath, header=TRUE)

# Process data: change row names, col names and delete cols
rownames(genesValue) <- genesValue[['id']]
genesValue <- subset(genesValue, select = -c(1))
colnames(genesValue) <- subjectNames

rownames(snpsValue) <- snpNames
snpsValue <- subset(snpsValue, select = -c(1))
colnames(snpsValue) <- subjectNames

rownames(snpPos) <- snpNames
snpToGene <- subset(snpToGene, select = -c(1))

# Create sliced matrix for eQTL analysis
genes <- SlicedData$new()
```

```
genes$CreateFromMatrix(as.matrix(genesValue))
snps <- SlicedData$new()
snps$CreateFromMatrix(as.matrix(snpsValue))

# eQTL analysis
eQTL <- Matrix_eQTL_main(snps, genes,
                         useModel = modelANOVA,
                         output_file_name = NULL,
                         output_file_name.cis = NULL,
                         pvOutputThreshold.cis=1e-3,
                         snpspos=as.data.frame(snpPos[,1:3]),
                         genepos=as.data.frame(probePos[,1:4]))
```

Listing 3: MatrixEQTL analysis

B.2 Extraction of cis-eQTLs

The intermediary step of the analysis is the *extraction of cis-eQTLs* method. It allows us to retrieve SNPs in significant gene - SNP pairs to investigate further with LD analysis. The selection of SNPs can be performed by calling "eQTLciseqtls", which returns a table having columns for SNPs, genes, test statistics result, p-value, and FDR. MAF is included as an additional column for researchers that are interested in the frequency of second most common allele occurrence in given data sets. The code for *extraction of cis-eQTLs* is given as follows:

```
# Minor allele frequency (MAF)
mafs = apply(as.matrix(snpsValue), 1, mean)/2
mafs = data.frame(snps=names(mafs), maf=mafs)

# cis-eQTL analysis
cisEqtl_res = eQTL$cis$eqtls
cisEqtls = merge(cisEqtl_res, mafs, by="snps")

# Retrieve SNPs from cis- analysis
cisSnps = cisEqtl_res[['snps']]
cisSnps <- as.matrix(cisSnps)
```

```
# Retrieve corresponding genes
cisGenes <- list()
for (snp in cisSnps){
    index = which(snp == snpToGene[[1]])
    cisGenes <- c(cisGenes, snpToGene[[2]][index])
    }

# Merge with cis- analysis report
cisEqtlsAll <- cbind(cisEqtls, cisGenes)
cisEqtlsAll <- cisEqtlsAll[c(1, 7, 2, 3, 4, 5, 6])
```

Listing 4: Extraction of cis-eQTLs

B.3 Linkage Disequilibrium

The *linkage disequilibrium* step requires allele pairs for each SNP; hence, *alleles* file having minor and major alleles has been prepared and loaded. It is worth noting that LD analysis are done on SNPs with encodings of 0, 1, and 2 instead of 1, 2, and 3. Therefore, for the sake of analysis, we alter encodings and store them as *genotypes* variable. Rather than having LD analysis on every pair of SNPs, we eliminate ones we do not find in *cis*-eQTLs to refrain from unnecessary consumption of memory and time because LD matrix has a shape of N x N and values are determined after analysing **N**: number of SNPs and **R**: number of gene expression values for **S**: number of subjects. Furthermore, since it is not possible to obtain LD values for all SNP pairs, resulting LD have fewer SNPs than SNPs from significant gene-SNP pairs. Hence, we store SNPs from LD analysis in a different variable, called *ld_snps*, so as to keep track of them.

Once the process of LD is complete, it is more convenient to save the results obtained from methods in R as .Rdata and load them either in NumPy or Pandas formats in Jupyter Notebook to prevent unnecessary computation time because matrices in R are not optimised as Python for the calculation of large data sets. For instance, converting genotypes to allele pairs takes 45 minutes on AddNeuroMed and 6 hours on HBTRC data sets.

```
# alleles: (N x 2)
```

```
alleles = read.csv(filepath, header=TRUE)
alleles <- as.matrix(alleles)
alleles <- subset(alleles, select=-c(1))
rownames(alleles) <- snpNames

# genotypes: Matrix with genotypes (N x S)
genotypes = read.csv(filepath, header=TRUE)
rownames(genotypes) <- snpNames
genotypes <- subset(genotypes, select=-c(1))
colnames(alleles) <- subjectNames

# Store genotypes as list with each row of genotypes matrix as
# numeric vector
genotypesList <- lapply(seq_len(nrow(genotypes)),
                        function(i) as.numeric(genotypes[i,]))

# Function to convert genotypes to allele pairs (i.e. "A/T")
linkGenotypes <- lapply(seq_len(nrow(genotypes)),
                        function(i)
                        as.genotype.allele.count(
                        genotypesList[[i]],
                        alleles = c(alleles[i, "a0"],
                                    alleles[i, "a1"])))
# Change the name
names(linkGenotypes) <- snpNames

# Find eQTL SNPs before generating genotypes
cisSnpIndex <- sapply(seq_len(length(cisSnps)), function(i)
                      match(cisSnps[i], snpNames))
selectedAsGenotypes <- asGenotypes[,cisSnpIndex]

# Make genotypes from cisSnps
selectedCisMakeGenotypes <- makeGenotypes(selectedAsGenotypes)

# Linkage disequilibrium analysis
```

```
LD <- LD(selectedCisMakeGenotypes)

# Create a variable to keep track of LD snps
ld_snps <- colnames(LD_cis_ANM$`R^2`)
ld_snps <- as.matrix(ld_snps)
```

Listing 5: LD analysis

B.4 Results

In this section, we present the results of eQTL analysis with LD. Firstly, Figure B.1 shows the results of LD analysis between *cis-* variants. Notice that the density of correlation between SNPs changes with respect to data sets. Hence, the number of significant gene-SNP pairs is different for both. Next, Figure B.2 presents F1 score results for the eQTL and LD analysis on both data sets. The resulting plot is expected to have a reversed U-shaped curve since stretching the threshold for *p-value* would increase the number of detected gene pairs. However, the result indicates in Figure B.2 that the statistical analysis cannot estimate any relationship between genes on AddNeuroMed and HBTRC data sets even though the analysis suggests other gene pairs.

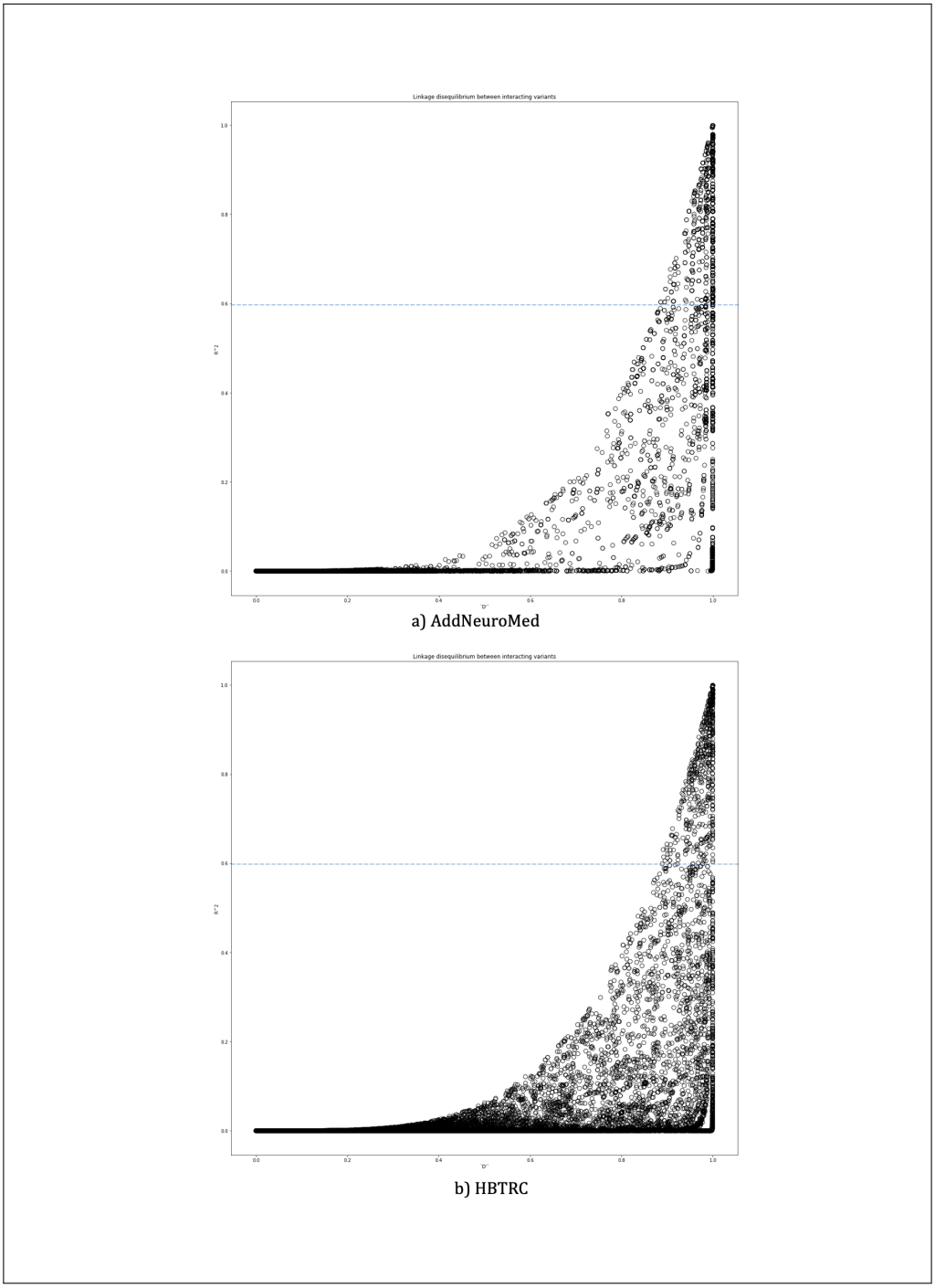
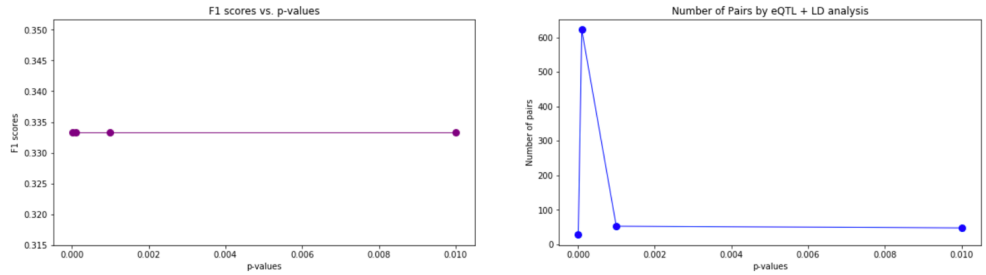
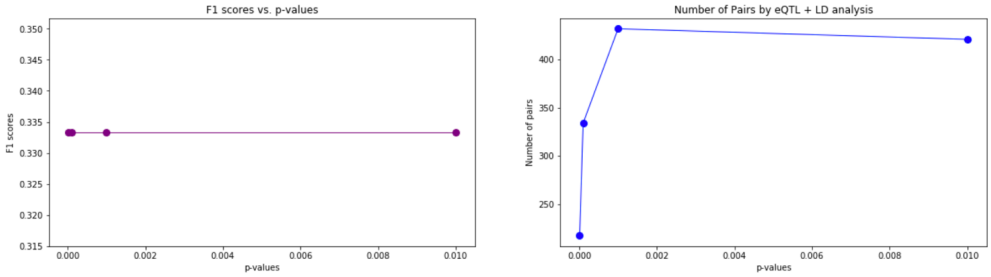


Figure B.1: Pairwise LD

Figures show the linkage disequilibrium between *cis*- variants. D' and R^2 are chosen in deciding high LD. Interactions between variants above dashed threshold ($R^2 > 0.6$) within pairs are removed.



a) AddNeuroMed



b) HBTRC

Figure B.2: F1 scores of eQTL and LD analysis and number of pairs found by eQTL + LD analysis

Figures on the left represents F1 scores with respect to a range of *p-values* $\in \{0.1, 0.01, 0.001, 0.0001\}$ for eQTL analysis and LD approaches applied on AddNeuroMed and HBTRC data sets. Figures on the right show the number of gene pairs that are found by the analysis for different p-values.

Appendix C

GeneInteraction DeepRNN Static Model

In the figures on the next pages, we present architecture and results of the static variant of the GeneInteraction DeepRNN.

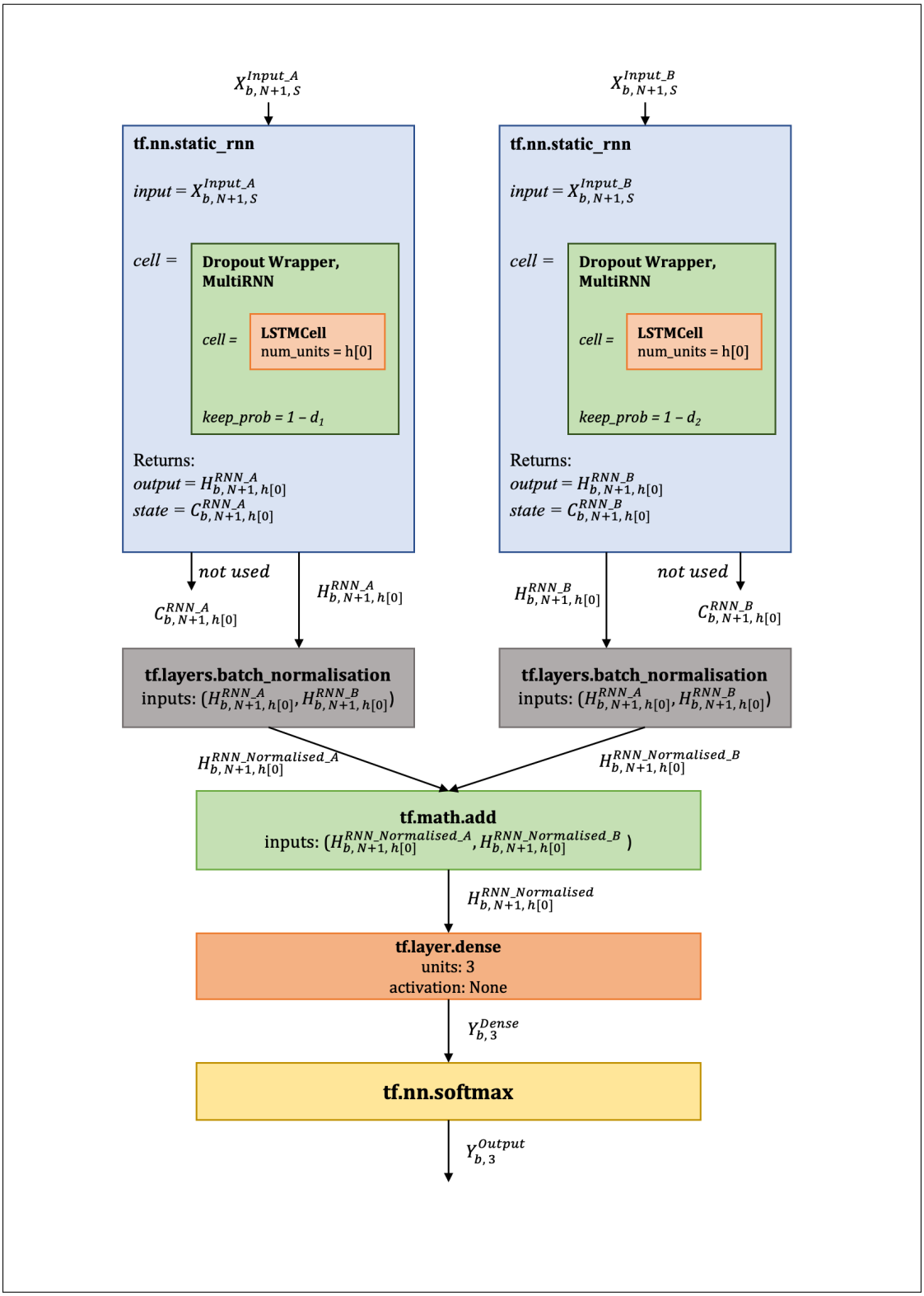
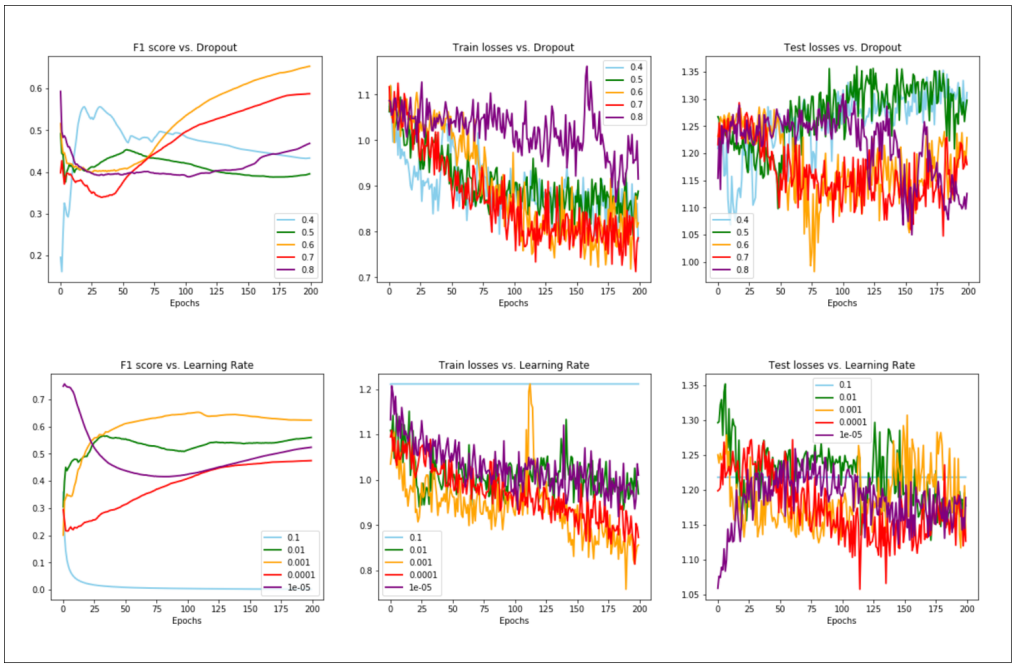
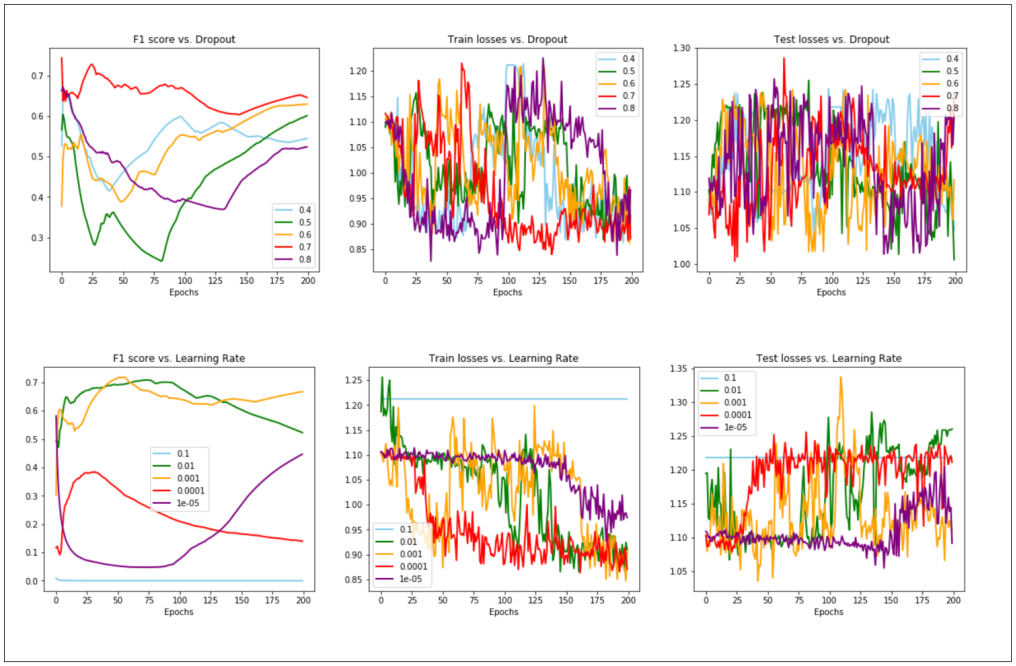


Figure C.1: Architecture of the GeneInteraction DeepRNN static model



a) AddNeuroMed



b) HBTRC

Figure C.2: Results of static model with respect to different values of hyper parameters, where dropout $\in \{0.4, 0.5, 0.6, 0.7, 0.8\}$ and learning rate $\in \{0.1, 0.01, 0.001, 0.0001, 0.00001\}$

Figure represents F1 scores of the static model with respect to a range of dropout and learning rates. In (a) and (b), the model is tested on AddNeuroMed and HBTRC data sets, respectively. The optimal dropout and learning rates are 0.6 and 1e-3 for both.

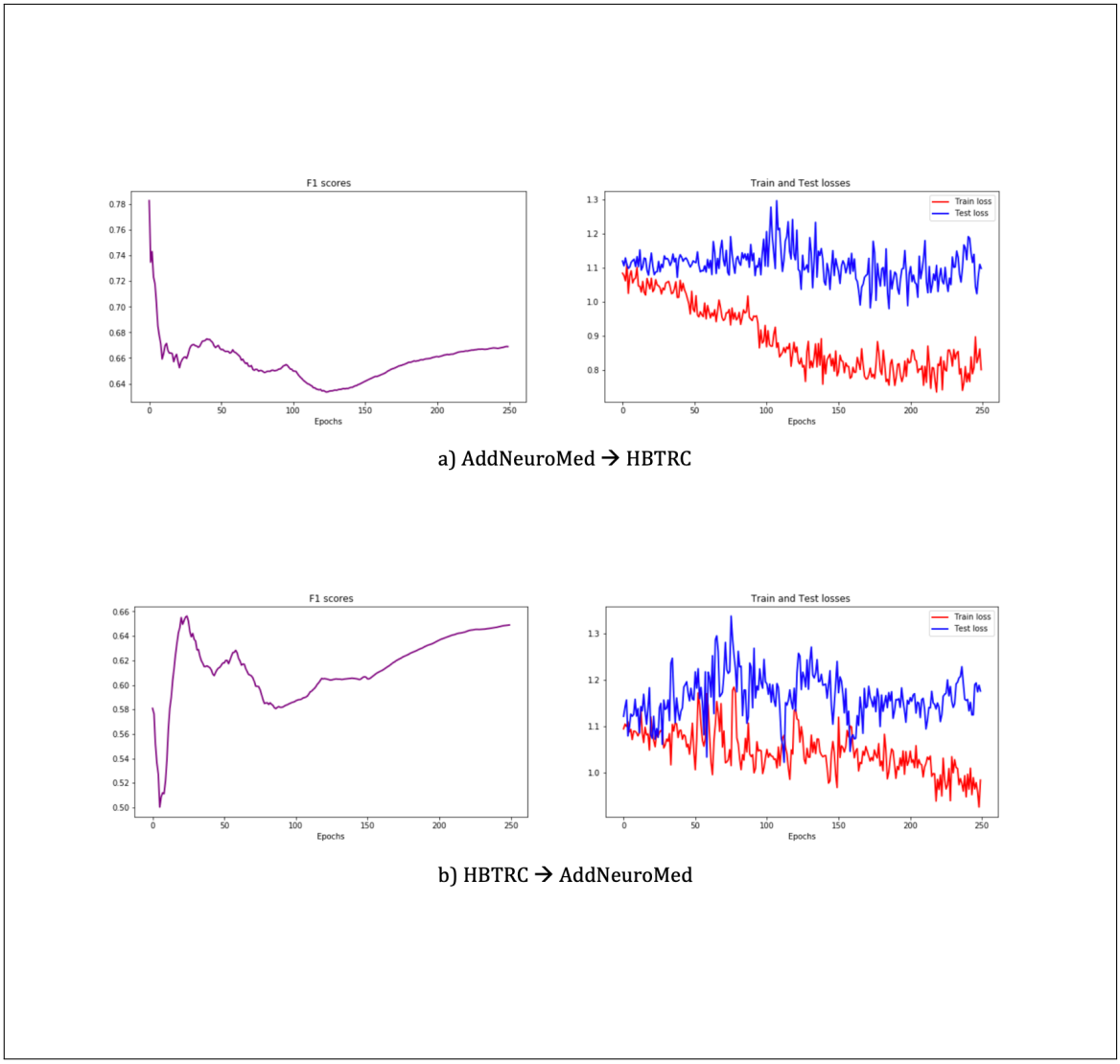


Figure C.3: Cross validation results of static model

Figures show cross validation results for static model. The model is trained on AddNeuroMed (HBTRC) and tested on HBTRC (AddNeuroMed) in (a) and (b), respectively.