EFFECTS OF ARCHITECTURE ON IMPUTATION OF GENE EXPRESSION USING DEEP NEURAL NETWORKS

BY

LAI WING KHANG

A PROPOSAL

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfilment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

MAY 2018

DECLARATION OF ORIGINALITY

I declare that this report entitled **"EFFECTS OF ARCHITECTURE ON IMPUTATION OF GENE EXPRESSION USING DEEP NEURAL NETWORK"** is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : _____

Name : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to express my gratitude and appreciation to my supervisor, Dr. Ng Yen Kaow for sharing his wisdom with me and giving me this opportunity to work on this project under his guidance.

ABSTRACT

Interests in using RNA sequencing (RNA-seq) data in the studies of differential gene expression analysis has been on the rise recently. Obtaining RNA-seq data at high coverages allows us to more reliably reconstruct RNA sequences, but at much higher costs. Hence it is highly beneficial to researchers if RNA sequences can be reliably inferred from low coverage RNA-seq data. One way to accomplish this is to impute low coverage RNA-seq data based on knowledge from known gene expression profiles. This is possible since studies have shown that correlations exist in the gene expression of RNA-seq data; by capturing these correlations through statistical models, the models can then be used to impute the RNA-seq data.

This project aims to study the effects of deep neural networks' architecture on imputation of gene expression. The performance of a deep neural networks on the imputation task will be evaluated by its root mean squared error. Besides, the performance of deep neural networks will be compared with a baseline K nearest neighbour imputation model as well.

We have applied 1D convolutional layers and greedy layer-wise training algorithm to achieve a good performance on our neural network.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

RNA-seq	RNA sequencing
mRNA	Messenger RNA
RNA	Ribonucleic acid
VAE	Variational Autoencoder
DAE	Denoising Autoencoder
SAM	Sequence Alignment Map

Chapter 1: Introduction

1.1 Problem Statement and Motivation

In recent years, RNA-seq data generated by next-generation sequencing technology has gained popularity in the studies of differential gene expression analysis due to its superior benefits compared to traditional microarray data (Zhao *et al.*, 2015).

Higher coverage of RNA-seq data is recommended for researchers because rarely expressed genes can be detected by increasing the coverage. (Kukurba & Montgomery, 2015) Besides, increased coverage can also improve the accuracy and precision of gene expression. However, higher coverage also comes at a higher cost. By imputing gene expression from low coverage RNA-seq, a corrected gene expression that is similar to the one derived from high coverage RNA-seq may be possible.

Many approaches have been proposed to impute RNA-seq data. These approaches use statistical models to impute the missing values in the RNA-seq data. To the best of our knowledge, there has been no attempt at using deep learning to impute RNA-seq data, although deep learning has made tremendous gains in many other areas in recent years. Promising result in data imputation by deep generative models (Duan *et al.*, 2014) has prompted us to apply deep generative model in imputation of gene expression from low coverage RNA-seq data.

1.2 Project Scope

The project scope is to study the effects of architecture on imputation of gene expression from low coverage RNA-seq data. The model should be able to correct possible errors and noises in gene expression such that the difference between corrected gene expression and gene expression from high coverage RNA-seq data is small.

1.3 Project Objectives

The project objectives are:

I. To develop different neural network model that can impute gene expression from low coverage RNA-seq data.

II. To achieve performance on par with, or above existing statistical models in the imputation of RNA-seq data.

III. To consider the difference in gene expression derived from RNA-seq data that are collected from different tissues or under different conditions.

1.4 Background Information

1.4.1 RNA-seq

A gene is a sequence of nucleotides that encodes a protein through the process of transcription and translation. Figure 1.4.1 shows the process of transcription and translation. Gene expression simply means that the gene is used in synthesizing a corresponding protein, and, this process is regulated by multiple mechanisms in our body. In transcription, an mRNA which has the complementary nucleotide sequence to the gene it was transcribed is produced. The mRNA is then used as a template to synthesize protein in the translation process.

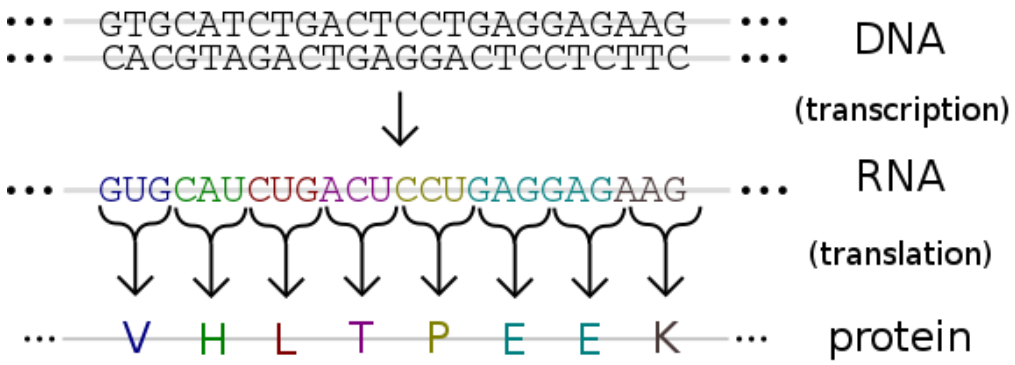


Figure 1.4.1. Illustration of the process of transcription and translation. (Madeleine, 2006)

The technology known as RNA-seq uses next-generation sequencing to generate millions of short nucleotides sequence, which is known as “read” from RNA of a sample of interest. These generated reads are then map to a reference genome, then, we can deduce which gene is being expressed from the mapped reads. This deduction is possible because RNA is used in both transcription and translation.

The number of reads mapped to a specific gene is called the read depth. In this project, read depth will also be referred to as coverage. Hence, higher coverage RNA-seq data means higher number of mapped reads. Higher coverage allows researcher to conclude that a gene is being expressed with higher confidence level. Furthermore,

higher coverage RNA-seq data is also able to reveal rarely expressed genes which low coverage RNA-seq data is unable to do so. The following figure shows the difference between high coverage and low coverage RNA-seq data.

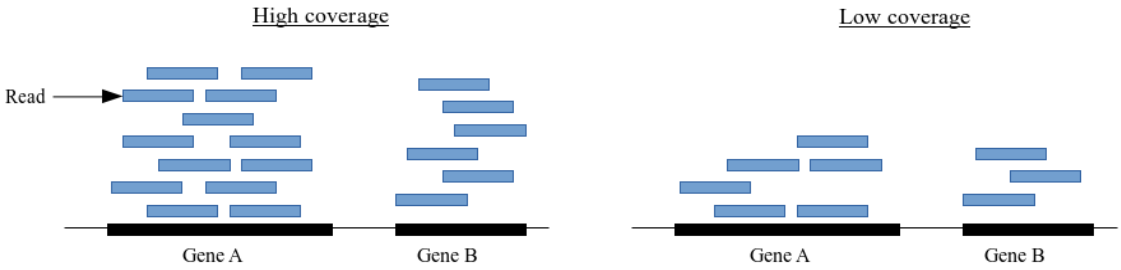


Figure 1.4.2. Illustration of high coverage and low coverage RNA-seq data.

1.5 Report Organisation

This report consists of 5 chapters. The first chapter introduces the problem statement and motivation of the project. This chapter explains the project background as well. Chapter 2 is reviews of past researches on imputation of gene expression using deep neural networks and statistical models.

Chapter 3 explains the process of data gathering and processing. Methodology for this project is included in this chapter too. Chapter 4 shows different architectures being experimented on and its results.

The final chapter concludes this project with discussion and future improvement that can be made to this project.

Chapter 2: Literature Review

2.1 Gene Expression Imputation

2.1.1 Statistical Model

Over the years, different approaches have been proposed to impute missing values in RNA-seq data. These imputation approaches are mainly based on regression, clustering and nearest neighbour method. In 2013, Bagjfalaki *et al.* conducted a comparative study on imputation of missing value in RNA-seq data using statistical models. The notations used in the paper are defined as follow:

Let Y_{gir} = read count of the *g*th gene in *r*th iteration of condition *i*.

Y_g^{mis}= missing read count of *g*th gene.

Figure 2.1.1 shows the gene expression matrix going to be used.

		Group 1				Group 2			
The gene code number	1	Y_{111}	Y_{112}	⋯	Y_{11n_1}	Y_{121}	Y_{122}	⋯	Y_{12n_2}
	2	Y_{211}	Y_{212}	⋯	Y_{21n_1}	Y_{221}	Y_{212}	⋯	Y_{22n_2}
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	g	Y_{g11}	Y_{g12}	⋯	Y_{g1n_1}	Y_{g21}	Y_{g22}	⋯	Y_{g2n_2}
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	G	Y_{G11}	Y_{G12}	⋯	Y_{G1n_1}	Y_{G21}	Y_{G22}	⋯	Y_{G2n_2}

Figure 2.1.1. Gene expression matrix and missing values are highlighted (Baghfalak et al., 2016).

In regression imputation approach, Spearman correlation is first used to find the K most correlated genes that is not a missing read count to the Y_g^{mis}. Let

$$x_{g*} = \left(x_{g*1}, x_{g*2}, \ldots, x_{g*(n_1+n_2)}\right), \quad x_{g*r} = \begin{cases} 0 \text{ if } i = 0 \\ 1 \text{ if } i = 1 \end{cases}$$

And let $X_g = \left(Y_{g*}^1, Y_{g*}^2, \ldots, Y_{g*}^K, x_g\right)$ be the covariate matrix for *g*th gene.

Then, the regression model will be

$$Y_{g*} \vee \beta_{g*} \sim Poisson(\mu_{g*}), \text{ where } \log(\mu_{g*}) = X'_{g*}\beta_{g*}$$

The regression model will then be fitted to the data, and the missing values are replaced by the predicted values from the regression equation. Other than this, another approach is using Expectation-Maximization algorithm to estimate the parameters β_{g*} of the regression model.

A model known as Bayesian Poisson regression is discussed in the paper as well. Here, a Poisson model is first fitted to the data, and then the posterior mean $\hat{\beta}_g$ and posterior variance $\hat{V}(\hat{\beta}_g)$ of the parameters β_g, g = 1, 2, ..., G are computed. Next, new parameters β_g^* are drawn from $\left(\hat{\beta}_g, \hat{V}(\hat{\beta}_g)\right)$. Lastly, the missing values are imputed as $Y_{g*}^{mis} \sim Poisson\left(\mu_{g*}^{mis}\right)$, where $\log(\mu_{g*}^{mis}) = X_{g*}^{mis}\beta_g^*$. In the case of overdispered count data, the missing values are imputed from negative binomial distribution instead of Poisson distribution.

The approach that yields the best results in missing value imputation among all the approaches discussed is Poisson mixture model. This is a clustering method that assumes all RNA read counts follow Poisson distribution. Parameters estimation and clustering of genes are done by Expectation-Maximization algorithm, whereas the number of clusters is estimated by Bayesian information criterion and Integrated Complete Likelihood criterion. After the genes have been clustered, the missing values are imputed as the mean of available genes in the respective cluster.

A review on imputation algorithms of gene expression data was conducted in 2010 (Liew *et al.*, 2010). During that time, microarray data was the norm in studies of differential gene expression analysis. Therefore, the gene expression values discussed in the paper were derived from microarray data instead of RNA-seq data. Despite this difference, this paper is still important as it provides prior knowledges in imputation of gene expression.

Chapter 2: Literature Review

In the following discussion, the term gene expression matrix refers to the matrix shown in Figure 2.1.1. One major insight from the paper is the existence of correlations in the rows and columns of the gene expression matrix. In the case of rows, multiple genes will be expressed at the same time under a certain condition, therefore, there is a correlation between these expressed genes. In the case of columns, it is to be expected that the same set of genes will be expressed under similar condition. Thus, there exists a correlation between these columns.

The algorithms reviewed were categorized into four different categories based on the type of information used by the algorithm, namely global approach, local approach, hybrid approach and knowledge assisted approach.

For algorithms in global approach, an assumption that there exists a global covariance structure in the gene expression matrix was made. Then, imputation algorithms such as SVD imputation and Bayesian principal analysis perform imputation based on this assumption. However, when the gene expression matrix is heterogeneous (and hence the local correlation is stronger) these algorithms will not perform well. On the other hand, algorithms in local approach work better when the gene expression matrix is homogeneous. These algorithms assume that only a subset of genes has a strong correlation with the gene that contains missing values. Missing values of the target gene will then be imputed based on the selected subset of genes known as reference genes. Notable algorithms in local approach include K nearest-neighbour imputation, least square imputation and linear regression imputation. In hybrid approach, they capture both local and global correlation information by combining algorithms from global approach and local approach. Hence, the method in hybrid approach is simply a combination of algorithms from global approach and local approach. Integration of domain knowledge such as regulatory mechanism and underlying biomolecular process into imputation is known as knowledge assisted approach. In case of small datasets that contains many missing values, this approach works better compared to the other three approaches discussed.

2.1.2 Deep Learning

Recent successes of deep learning have prompted researchers to use deep learning for tackling genomic problem. In 2015, Chen *et al.* proposed a deep learning model known as D-GEX to infer gene expression from the expression of selected landmark genes. Their study was motivated by an observation discovered by researchers from LINCS program. The observation is that ~1000 genes out of ~22000 human coding genes can capture around 80% of the gene expression information. These ~1000 genes are known as landmark genes. However, LINCS program uses linear regression to infer the gene expression from the expression of landmark genes. Therefore, Chen *et al.* proposed using deep learning instead of linear regression to infer the gene expression based on landmark genes, because there exists nonlinearity in gene expression data.

The D-GEX with the best performance is a feedforward neural network that consists of three fully connected layers. The architecture of the network is shown in Figure 2.1.2. Since the network has three fully connected layers with 9000 hidden units in each layer, the memory usage of this network is relatively large. So, they separate the target genes layer into two different GPU. D-GEX also uses dropout technique which is known to be useful in regularization, and they use tanh as activation function to capture the nonlinearity in gene expression. Two important observation were discovered when Chen et al. were trying to interpret the learned D-GEX. One of it is that they discovered the connections of units in hidden layers are sparse. Another observation is that neural network is able to capture the nonlinearity in gene expression.

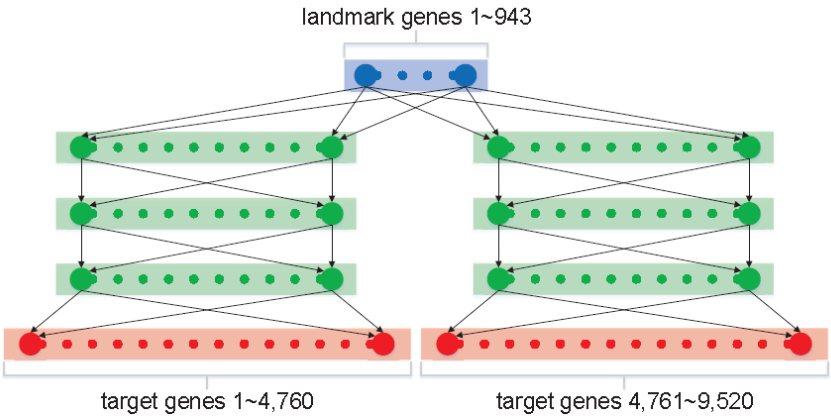


Figure 2.1.2. Architecture of D-GEX (Chen et al., 2015)

A deep autoencoder network is proposed for inferring gene expression from single nucleotide polymorphisms (SNP) genotypes (Xie *et al.*, 2017). Autoencoder (Bengio et al., 2007) is a type of neural network that made up of two components, encoder and decoder. Both encoder and decoder are feed forward neural networks. Given an input x, encoder will encode it into a compressed features vector, in other words, the input is being encoded into a low dimensional representation. Then, the compressed features vector is feed into decoder, and an output which is similar to the input x is generated by the decoder. The architecture of the proposed network is shown in Figure 2.1.3.

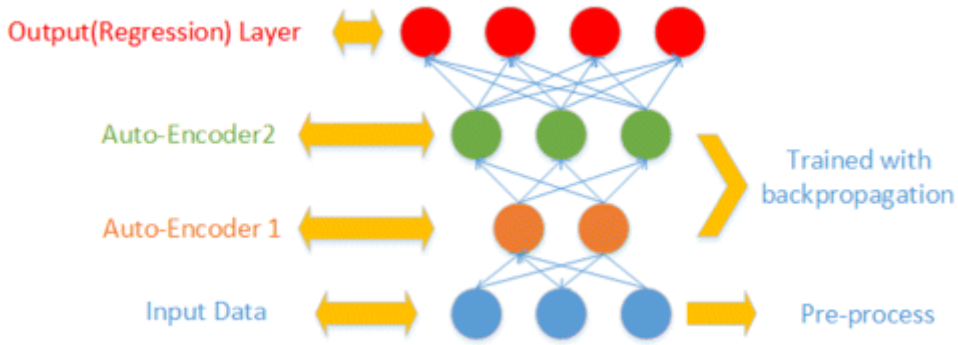


Figure 2.1.3. Architecture of the deep autoencoder network. (Xie et al., 2017)

The network consists of two hidden layers which are autoencoders, and a final layer of linear regression model. One thing to note is that they use denoising autoencoder (DAE) instead of autoencoder in the network. DAE is just another variant of autoencoder (Vincent et al., 2008). The difference between autoencoder and DAE is the input data, noises are added into the input of DAE as the name suggested. Although it uses noisy input, but the original input without noises is used to compare with the generated output. This forces the DAE to learn the features from the noisy input, instead of learning an identify function which simply recreates the input. Dropout was used in the network to improve the performance as well. Unfortunately, details of the two hidden autoencoder layers such as number of layers in autoencoder and activation function used were not stated by the authors.

2.2 Vanishing Gradient Problem

By adding more hidden layers to a neural network, it is able to model a much more complex function, and hence it can perform well on complicated tasks. However, simply adding more hidden layers does not necessarily improve the performance of a neural network, as it might suffer from vanishing gradient problem.

Training a neural network is generally done with backpropagation and gradient descent, where each layer in the neural network is updated based on the gradients of loss with respect to the layer's weights. Vanishing gradient happens when the gradients being backpropagated are getting smaller. As a result, early layers are not updated as the gradient is too small.

Over the years, multiple approaches have been proposed to deal with vanishing gradient problem.

2.2.1 Layer-wise training

One of the earlier methods used to overcome vanishing gradient is greedy layer-wise training of neural network. In 2006, Hinton *et al.* introduced Deep Belief Networks (DBN), which is a stacked of Restricted Boltzmann Machines (RBM) layers. The training of DBN involves two stages, pre-training stage and fine-tuning stage.

During the pre-training stage, each layer is trained sequentially as an RBM with Contrastive Divergence algorithm. Unlike gradient descent, each layer is trained without a cost function, instead it is trained to learn a higher-level representation of its input. Once a layer has been trained, another layer will be trained using trained layer's output as its input, and they are stacked together at the end. Hence, this is known as a greedy layer-wise unsupervised learning algorithm. In the fine-tuning stage, a wake-sleep algorithm is used to fine-tune the network. Besides, gradient descent has been used to fine-tune the DBN as well (Bengio *et al.*, 2007). Although DBN does not suffer from vanishing gradient problem, it does not address the root cause of the problem, but merely avoided the problem by not using gradient descent and backpropagation to update the network's weights.

In 2007, Bengio *et al.* attempted a greedy layer-wise supervised training algorithm. In this case, each layer is trained using gradient descent sequentially. However, their

experiments show that networks trained using this algorithm performs worse than networks trained using greedy layer-wise unsupervised learning algorithm.

Greedy layer-wise supervised training algorithm was also used to train convolutional neural network in an attempt to deal with small and specialized dataset (Rueda-Plata et al., 2015). The networks that they trained is shown as followed.

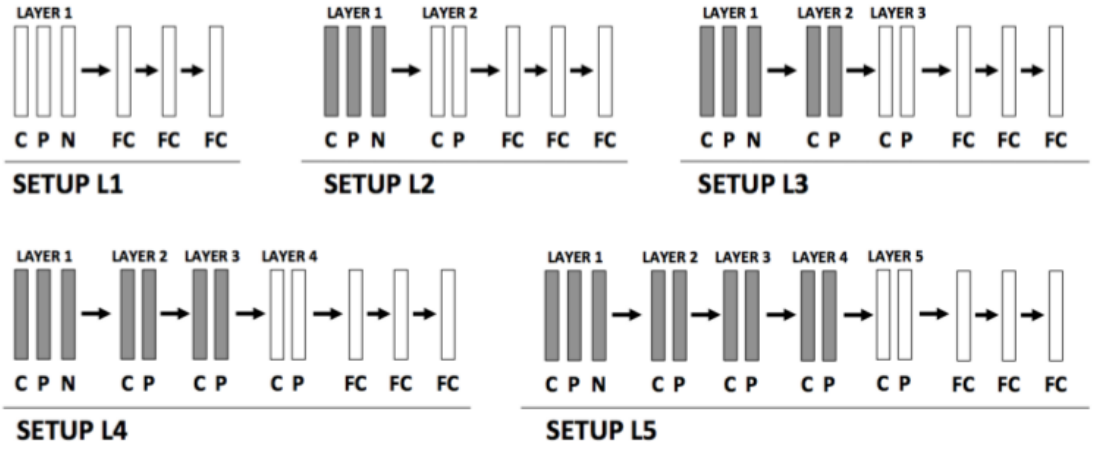


Figure 2. Different CNN trained (Rueda-Plata et al., 2015), 'C' represents convolutional layer, 'P' represents pooling, 'N' represents normalization, 'FC' represents fully connected layer.

The training of their networks can be summarized as follow:

1) Start with a base network which consists of one convolutional layer and three fully connected layers.
2) Initialize the base network with random weights and train the base network.
3) Add an additional convolutional layer to the trained network. The new layer is added in between the trained convolutional layer and fully connected layers.
4) Initialize the newly added convolutional layer and fully connected layers with random weights, while keeping the weights of trained convolutional layer same. Then, train the resulting network.
5) Repeatedly add new convolutional layer as described in Step 3 and 4.

Their results show that CNN trained layer-wise achieved higher accuracy on multiple datasets compared to CNN trained from scratch.

2.2.2 Activation Function

Sigmoid and tanh non-linearity were the norm as activation function for neural network in the early days. However, both activation functions can saturate and cause the gradient to vanish. This happens on sigmoid non-linearity when the output is near 0 or 1, where the gradient at these two regions is close to 0. On the other hand, this happens to tanh non-linearity when the output is near -1 or 1.

ReLU has been the most common activation function used in neural network nowadays, because it does not suffer from vanishing gradient problem as much as other activation functions. ReLU only saturates at negative region, which is relatively better than sigmoid and tanh which saturate on much larger region. However, since the gradient of ReLU at negative region is completely 0, ReLU might "die" and never be able to be updated again. This is known as the "dying ReLU" problem. Variations of ReLU such as Leaky ReLU and Parametric ReLU have been proposed to fix the "dying ReLU" problem. These ReLU variations introduced a tiny negative slope on the negative region to deal with the problem.

2.2.3 Weight Initialization

The output of a neural network layer is generally in the form as followed:

$$output = f(weights \cdot input + bias),$$

where f is a non-linearity function. If the weights of a layer are initialized with values that are too small, the output will be getting closer to 0 at deeper layers. On the other hand, if the weights of a layer are initialized with values that are too large, and tanh is used as the non-linearity function, then the output will be saturated on -1 or 1. This will lead to vanishing gradient problem as the gradient is 0 when the output is saturated.

Therefore, it is important to ensure the variance of the output remains approximately the same across layers in neural network. In 2010, Glorot and Bengio shows that in order for the variance of output remains the same, the variance of the weights need to be

$$Var(weights) = 2 / (n\_in + n\_out),$$

where *n_in* is number of units in previous layer and *n_out* is number of units in next layer. However, while this works well for sigmoid or tanh, it does not work well for ReLU. So, in 2015, He *et al.* suggested the variance of the weights to be

$$Var(weights) = 2 / n\_in,$$

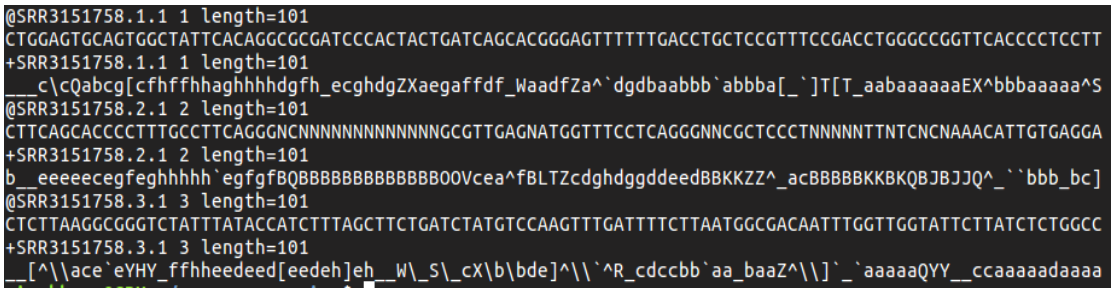
where *n_in* is number of units in previous layer. In 2015, batch normalization (Ioffe *et al.*, 2015) was introduced to ease the trouble of initializing neural network's weights properly. The idea of batch normalization is that, if we want the variance of the output to be similar across layers, we can just zero-centre and normalize the output at each layer. This is similar to normalizing the data for the first layer of neural network.

Chapter 3: System Design

3.1 Dataset

The dataset was downloaded from National Center for Biotechnology Information, it consists of 81 RNA-seq data, which were collected from skeletal muscle tissues of 23 participants. The skeletal muscle biopsies were performed on the participants before and after a training period, hence, these RNA-seq data can be categorized based on the exercise status, which is trained or untrained. 26 out of the 81 RNA-seq data have the exercise status of trained, while the rest are untrained. All the RNA-seq data are in FASTQ format and were produced by Illumina HiSeq 2000 Sequencing System. Number of bases of the dataset range from 0.7 Gbases to 8.1 Gbases. And, the file size of the dataset ranges from 0.5 GB to 5.9 GB. In total, the dataset contains around 195 GB of raw RNA-seq data.

For the RNA-seq data in FASTQ format, every read is represented by four lines. Line 1 and 3 consist of a unique sequence identifier and a description: the length of the read in our case. Line 2 is the nucleotide sequence of the read. It is made up of five different letters, A, T, C, G and N, which represents Adenine, Thymine, Cytosine, Guanine and ambiguous base respectively. Each character in line 4 indicates the quality value for the respective base in line 2. Figure 3.1.1 shows some examples of



RNA-seq data.

Figure 3.1.1. First three sequences of one sample in FASTQ format.

Since these are human RNA-seq data, Genome Reference Consortium Human Build 38 patch release 10 (GRCh38.p10) is used as the reference genome for read alignment.

3.2 Data pre-processing

3.2.1 Alignment

Alignment is first performed on the raw sequences in FASTQ format using HISAT2 (Kim et al., 2015) with GRCh38.p10 used as the gene annotations. The aligned reads are then stored in the format of SAM. In order to conduct further data preprocessing and reduce the file size of the aligned reads, SAMtools (Li et al., 2009) is used to sort the aligned reads. Next, the sorted aligned reads are converted into a binary format known as BAM. 85% of the original file size is reduced by converting the aligned reads to BAM format. Figure 3.2.1 shows first two reads in BAM format.

```
SRR3151758.31112856.1   69      chr1    10535   0       *       =       10535   0
CCCAGCATTGGCGGGTTCTGGGACTGGTGGCTGTGGTGGCTCGTTGGGCTTTGTCTCTTAGAAGGGGGGGGATAATCATC
ATCTGAGGAGAACTCTGCTCC   bbbeeeeeggggfih^ecX^aedghdfWae_be\_cHZ`Q^ZSKZU\FKT\bTKGT
KT]R]T]GXETV_aEOTX_]`b`bbb]bYR]aSQJWY_bb[Y`]_   YT:Z:UP
SRR3151758.31112856.2   153     chr1    10535   0       99M2S   =       10535   0
GTACCACCGAAATCTGTGCAGAGGAGAACGCAGCTCCGCCCTCGCGGTGCTCTCCGGGTCTGTGCTGAGGAGAACGCAAC
TCCGCCGTCGCAAAGGCGCCG   bcaa_^_ccccccccccbbb`bcca_Tabccaa]aaaccccccccb_^Wcaccc
aacbca`cccbbdcbbc^eefgghfiihfhhgfgcggceeee__b   AS:i:-14        ZS:i:-14        X
N:i:0   XM:i:2  XO:i:0  XG:i:0  NM:i:2  MD:Z:25C62T10   YT:Z:UP NH:i:2
```

Figure 3.2.1 First two lines of the aligned and sorted reads in BAM format.

3.2.2 Subsampling

To simulate low coverage RNA-seq data, we use a function in SAMtools to subsample 10% from the original aligned BAM file. We generate five low coverage data for each of the sample with different seed value. The original aligned BAM file will be used as high coverage data.

3.2.3 Read count

Another tool known as featureCounts (Liao et al., 2013) is used to count the mapped reads for each gene in GRCh38.p10. The output of featureCounts consists of seven columns, but, we only require three out of the seven columns. Therefore, a basic UNIX command cut is used to extract the relevant fields from the output of featureCounts. The required columns are unique identifier of the gene, length of the gene and count of mapped reads to the gene. Figure 3.2.2 shows read counts of first three genes.

```
Geneid  Length  SRR3151758_sorted.bam
ENSG00000223972 1735    2
ENSG00000227232 1351    32
ENSG00000278267 68      4
```

Figure 3.2.2. First 4 lines of the extracted output from featureCounts.

3.2.4 Gene expression value

Since we are using paired-end RNA-seq data, we use FPKM to measure the gene expression value. The equation of FPKM is:

$$FPKM_i = \frac{n_i \times 10^9}{N \times L_i}$$

where the number of reads mapped a gene *i* is denoted as *n*i, the total number of reads mapped to all genes is denoted as N and the length of gene *i* is denoted as *L*i. A simple Python script is written to compute the gene expression value.

Figure 3.2.3. shows the flowchart of the whole data pre-processing.

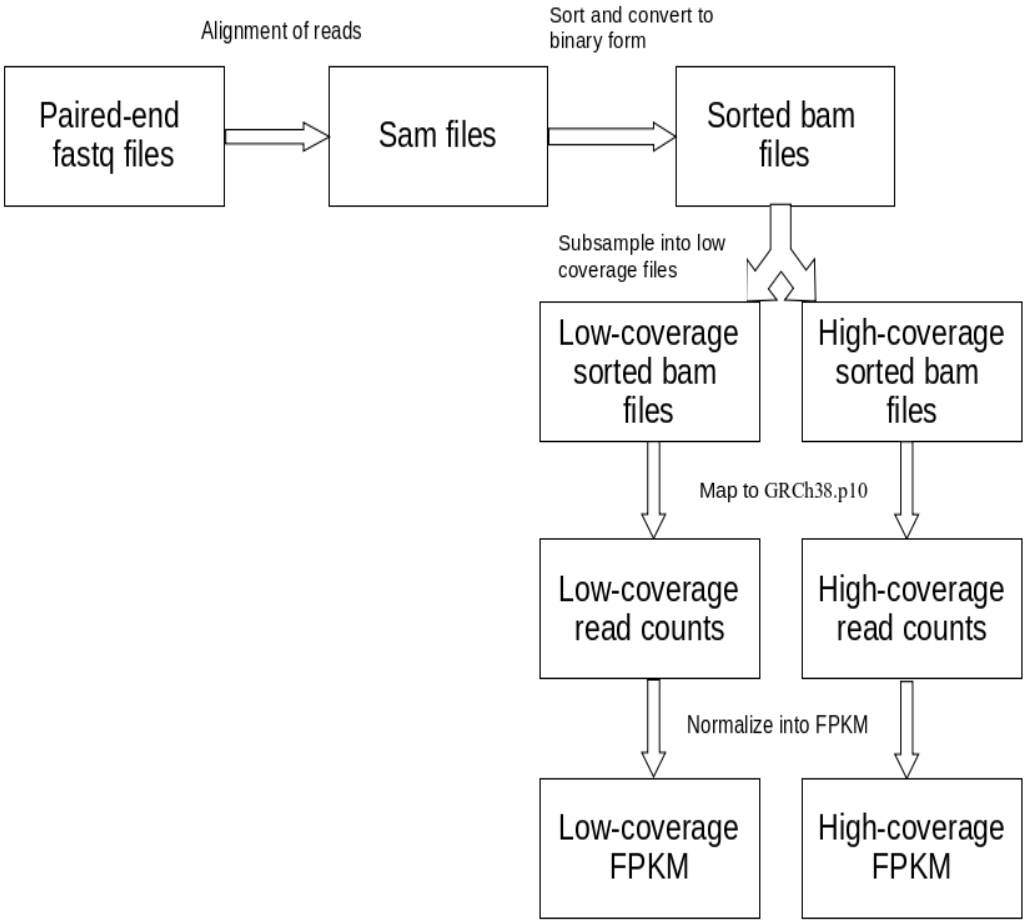


Figure 3.2.3. Flowchart of data pre-processing.

3.2.5 Gene expression matrix

After the gene expression value of the 81 high coverage bam files and the 405 low coverage bam files have been computed, the results are concatenated into two different gene expression matrices. Figure 3.2.4. shows the gene expression matrix of high coverage bam files.

Geneid	ENSG00000223972	ENSG00000227232	ENSG00000278267	ENSG00000243485	
SRR1560164	0.0	101.68057346785957	577.186784685203	0.0	
SRR1560165	0.0	189.04916076321769	234.74762517564992	0.0	
SRR1560166	0.0	107.50210097360039	363.5427745064684	0.0	
SRR1560167	0.0	60.21612048333892	239.27052580291436	3.18	
SRR1560168	0.0	186.34697364076925	361.1976490511898	0.0	
SRR1560169	0.0	60.37552171530735	224.90991683101169	0.0	
SRR1560170	0.0	111.86572802530071	231.5113335817728	0.0	

Figure 3.2.4. Gene expression matrix of high coverage bam files.

However, the range of the values in the gene expression matrix is too large. Therefore, we $\log_2$ transformed the values in the matrix.

3.3 Implementation

3.3.1 Methodology and experiment procedure

In order to impute a gene expression value from low coverage data, it is important that the neural network model is able to capture the correlations between genes in high coverage data.

However, there are 58243genes in the gene expression matrix. Therefore, we will experiment with different deep learning models to determine the most suitable model that is able to capture these correlations.

To ensure that the independent variable in our experiment is the architecture, we will keep the following hyperparameters constant across different architectures.

1) number of epochs trained: 100
2) batch size: 16
3) optimization algorithm: Adam
4) learning rate: 0.0001

Agile methodology will be adopted throughout this project, since it allows us to quickly experiment with different architecture based on one's result.

3.3.2 Models implementation

All the proposed models will be implemented in PyTorch and run on a system with two GPUs (Nvidia GTX1060 and Nvidia GTX1080). Since we will be experimenting with different models, a single system is not enough. Therefore, we will also be using Colaboratory, a free cloud service with GPU (Nvidia Tesla K80) from Google.

3.3.3 Evaluation of models

We adopt root mean square error (RMSE) to evaluate the performance of the imputation model in imputation of gene expression from low coverage data,

$$RMSE = \sqrt{\frac{\sum(x_i - y_i)^2}{N}}$$

where N is the total number of genes, x_i is the corrected $gene_i$ expression value of low coverage data, and y_i is the $gene_i$ expression value of high coverage data.

In order to have a better understanding on the RMSE score obtained by each model, we need a baseline model. K nearest-neighbour imputation will be used as the baseline model for evaluation.

3.4 Timeline

Figure 3.4.1 shows the timeline of this project that span across roughly eight months. The outcome of this project includes two reports.

ID	Task Name	Start	Finish	Duration
1	Understanding RNA-seq data	22/1/2018	2/2/2018	2w
2	Study existing methods	5/2/2018	16/2/2018	2w
3	Data pre-processing	9/2/2018	22/2/2018	2w
4	**Build an imputation model**	**22/2/2018**	**27/7/2018**	**22.4w**
5	Experiments with different model	22/2/2018	11/7/2018	20w
6	Training and testing of model	12/3/2018	27/7/2018	20w
7	Report 1	22/1/2018	9/3/2018	7w
8	Report 2	11/5/2018	2/8/2018	12w

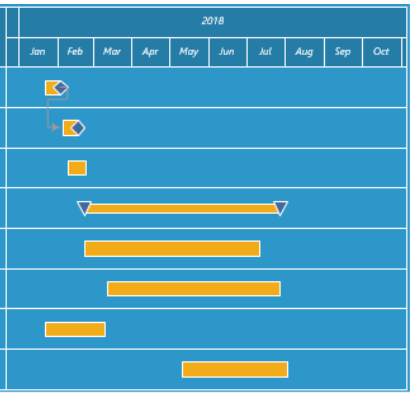


Figure 3.4.1 Project Gantt chart.

Chapter 4: Experiments and Results

4.1 Fully connected and autoencoder models

The first architecture that we experiment with is fully connected model, which is similar to D-GEX (Chen et al., 2015). The difference being our input and output layers consist of 58243 units. Ideally, we would like to experiment with hidden layer that has 3000, 6000 or 9000 hidden units. However, a fully connected layer between 58243 input units and 3000 hidden units will requires more than 8GB of memory, which is unable to run on our machine. Thus, we only experiment with 1000, 2000 or 2500 hidden units. The number of hidden layers we experiment with is 3, 4, 5, 6 or 7. Dropout was applied to all the hidden layers in the model as well, with the dropout rate set to 25%. This fully connected model will be referred to as *FC_x* model in this project, with *x* indicating number of hidden layers. Figure 4.1.1. shows an example of architecture of *FC_3* that has 3 hidden layers.

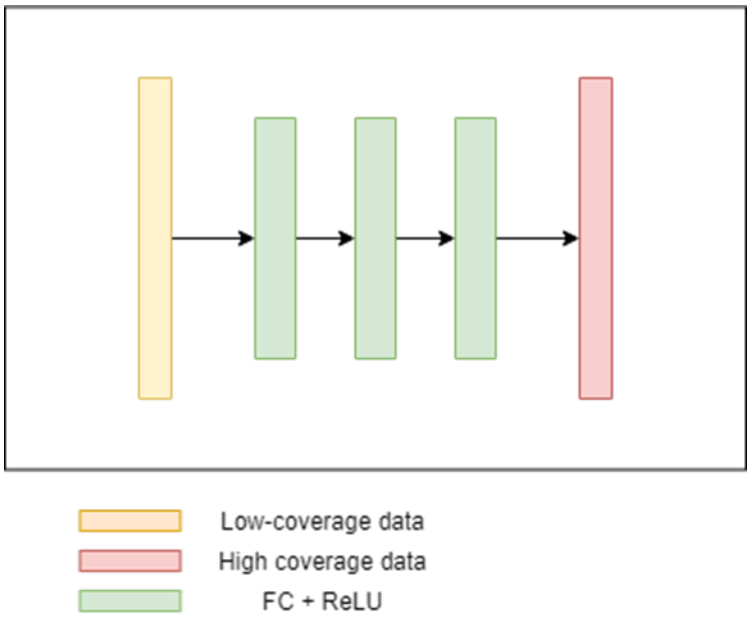


Figure 4.1.1. Architecture of *FC_3* model.

An autoencoder model that is similar to MLP-SAE (Xie et al., 2017) is being experimented on as well. The hidden layer of this model is an autoencoder as shown in the following figure.

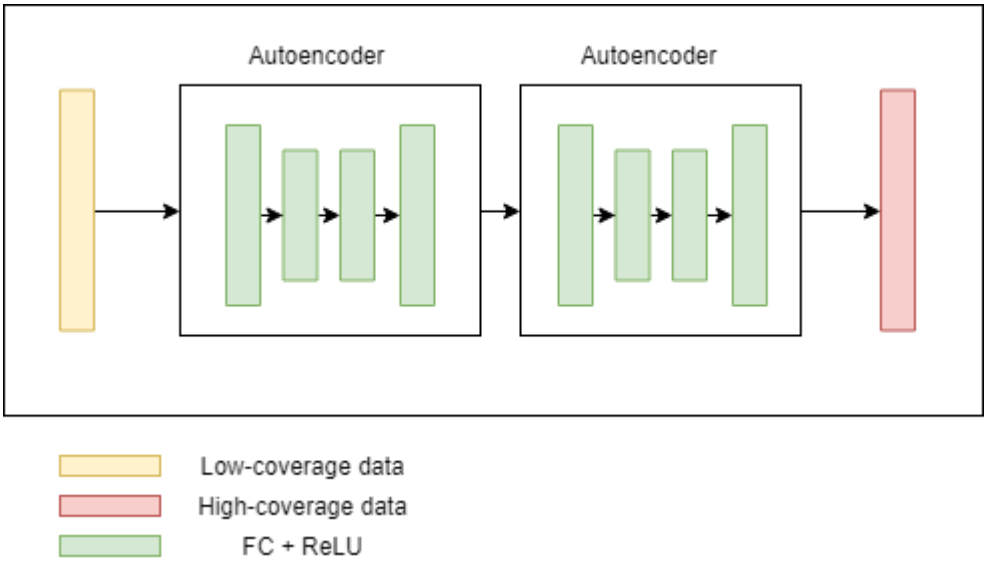


Figure 4.1.2. Architecture of autoencoder model.

We experiment with one autoencoder as hidden layer, as well as two autoencoder stacked together as hidden layer. The model with one autoencoder as hidden layer will be referred to as *AE_1*, whereas the model with two autoencoder as hidden layer will be referred to as *AE_2*.

	Number of hidden units		
Architecture	1000	2000	2500
FC_3	1.425	1.391	**1.383**
FC_4	1.444	1.435	1.426
FC_5	1.489	1.518	1.511
FC_6	5.502	2.449	1.566
FC_7	5.310	2.243	1.736
KNN imputation	**1.46**		
AE_1	9.82		
AE_2	10.24		

Table 4.1.1. Results of fully connected and autoencoder models.

Results show that *FC_3* perform better than all the other models, where three *FC_3* with different hidden units have the lowest RMSE. There are two trends that can be observed from this result. First, as the number of hidden layer in fully connected model increases, their RMSE increases too. Second, RMSE decreases as the number of hidden units increases.

Both autoencoder models performed the worst in this experiment with 9.82 and 10.24. This result is to be expected, as the autoencoder model is essentially the same as the fully connected model, with the difference being autoencoder model has less hidden units in the middle layers.

To investigate further on why fully connected model performs worse when it has more than 5 hidden layers, we examine the gradients of the hidden layers. First, we examined the magnitude of gradients for units in the hidden layers by plotting a graph. The sum of magnitude of gradients can be used as a heuristic to measure how fast the layer is learning (Socher, 2016). Intuitively, if the sum of magnitude of gradients is large, then the layer is learning faster.

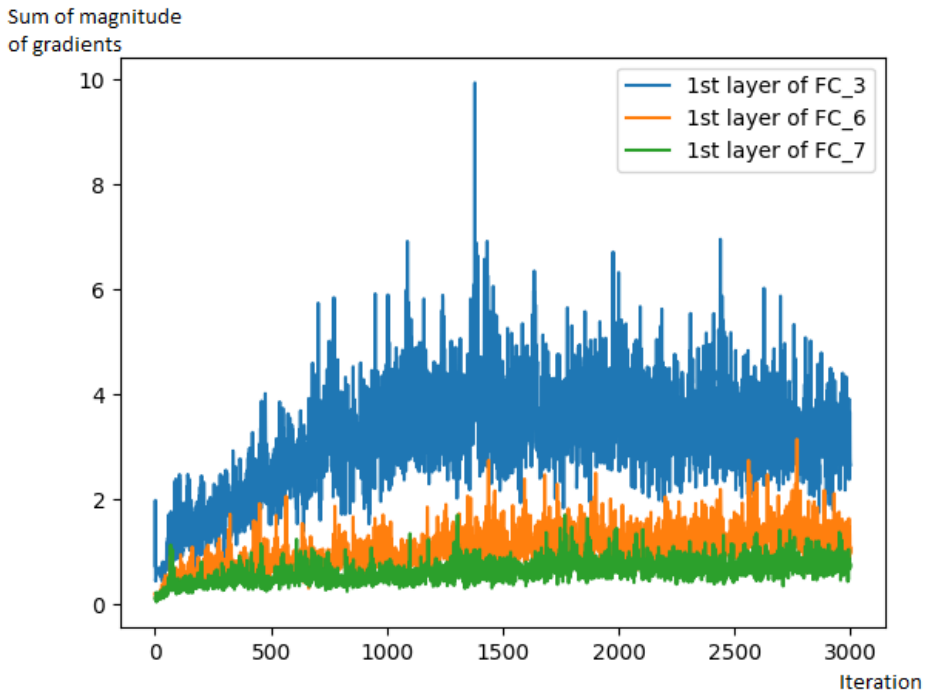


Figure 4.1.3. Plot of sum of magnitude of gradients in first layer for *FC_3*, *FC_6* and *FC_7*.

Based on the graph, we can observe that the sum of magnitude of gradients is lower for deeper fully connected model, such as *FC_6* and *FC_7*. This indicates that first layer of deeper fully connected models is learning relatively slower. This is one of the possible explanations that fully connected models have a higher RMSE when number of hidden layers increases.

4.2 Fully connected models with convolutional layer

With the input layer and output layer having 58243 units, it is difficult to increase the number of units in hidden layers to more than 2500, as it will take tremendous amount of GPU memory to train the model. Hence, we used 1D convolutional layers to reduce the number of units in the input layer.

We experiment with two different number of 1D convolutional layers added to fully connected model, these combinations of 1D convolutional layers will be referred to as *CONV_1* and *CONV_2*. The configuration of each of them is shown in the following table:

Combination	Layer	Input size	Filter size	Stride	Output size
CONV_1	1D convolution	58243	3	2	29121
	1D convolution	29121	3	2	14560
CONV_2	1D convolution	58243	3	2	29121
	1D convolution	29121	3	2	14560
	1D convolution	14560	3	2	7280

Table 4.2.1. Configuration of 1D convolutional layers.

As shown in Table 4.2.1., the input size got reduced to 29121, 14560 or 7280. These reduced size input are then feed into FC model.

FC Model with 1000 hidden units	Without Conv	With *CONV_1*	With *CONV_2*
FC_3	1.425	1.489	1.756
FC_4	1.444	1.742	1.843
FC_5	1.489	1.596	1.774
FC_6	5.502	**1.729**	**1.677**
FC_7	5.310	**1.567**	**1.647**

Table 4.2.2. Comparison of results of FC model with and without *CONV*.

Based on the results shown in Table 4.2.2., we can see that shallower fully connected models such as *FC_3*, *FC_4* and *FC_5* has a worse performance when *CONV_1* or *CONV_2* is added to the model. However, deeper fully connected models such as *FC_6* and *FC_7* are able to perform better with *CONV_1* or *CONV_2* added compared to without any convolutional layer added. Again, we examined the gradients of the hidden units in the first layer.

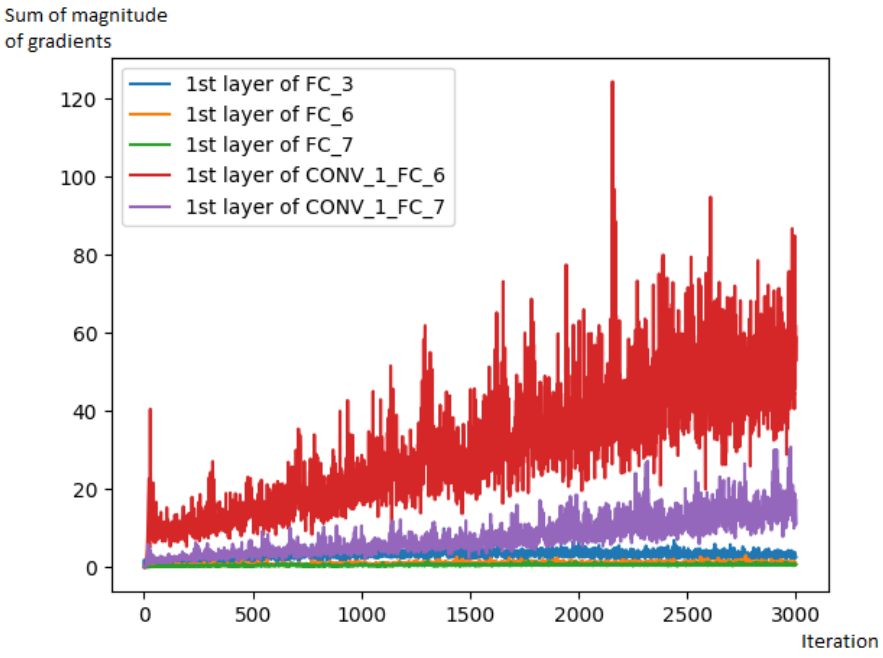


Figure 4.2.1. Plot of sum of magnitude of gradients in first layer for *FC_3*, *FC_6*, *FC_7*, *FC_6* with *CONV_1* and *FC_7* with *CONV_1*.

Based on Figure 4.2.1., sum of magnitude of gradients on first layer for *FC_6* and *FC_7* with *CONV_1* is larger than *FC_6* and *FC_7* without any convolutional layer. This might be the reason that deeper fully connected models with convolutional layers are performing better than deeper fully connected models without convolutional layers.

4.3 Greedy layer-wise trained models

We used greedy layer-wise supervised training algorithm to train fully connected with convolutional layer models as well. The training of the models can be described as follow:

1) Train a base network which consists of two 1D convolutional layers and three fully connected layers.
2) Add an additional fully connected layer to the trained network. The new layer is added in between the trained second last and last fully connected layer.
3) Train the resulted network.
4) Repeatedly add new fully connected layer as described in Step 2 and 3.

The architecture of base network and resulted network with 1 fully connected layer added is shown in the following figure.

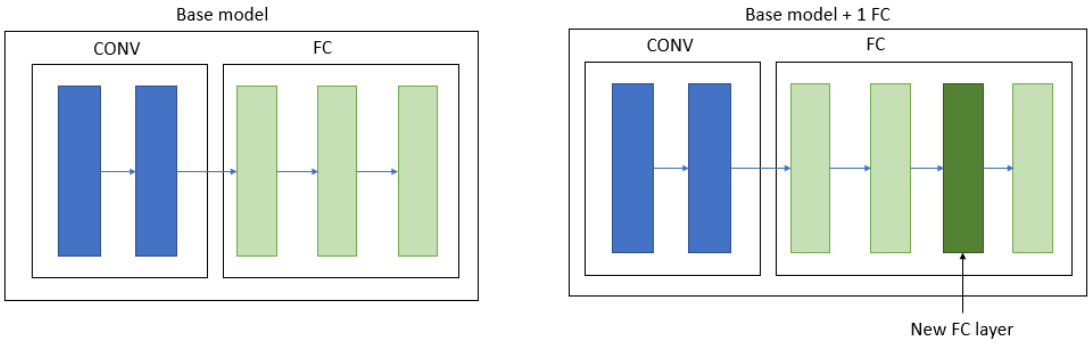


Figure 4.3.1. Architecture of base network and base network with 1 fully connected layer added.

FC Model with 1000 hidden units	Without Conv and greedy layer-wise training	With CONV_1 and greedy layer-wise training
FC_3	1.425	1.490
FC_4	1.444	1.387
FC_5	1.489	1.370
FC_6	5.502	1.371
FC_7	5.310	1.360
FC_8	-	**1.351**

Table 4.3.1. Results of greedy layer-wise trained models.

Based on the results shown in Table 4.3.1., it shows that with convolutional layer and greedy layer-wise training, we are able further reduce the RMSE to 1.351.

Chapter 5: Conclusion

5.1 Project Review

RNA-seq has gained a lot of interests from researchers lately due to its superiority compared to microarray data. The level of coverage of the RNA-seq data will affects the application of the data, as high coverage RNA-seq data is able to reveal rarely expressed genes. However, higher cost of high coverage RNA-seq data has hinder the progress of researches. Thus, it is important that we can impute low coverage RNA-seq data into high coverage RNA-seq data.

In this project, we experimented with different deep neural network architectures to impute gene expression. By applying both convolutional layer and greedy layer-wise supervised training algorithm, we are able to train deep neural networks that are able to achieve better performance than the baseline model.

5.2 Future Work

In this project, we studied the effects of different deep neural network architectures on a dataset of only 81 unique RNA-seq data. However, deep neural networks require massive amounts of data to achieve a better performance. Therefore, we expect that a larger dataset will be able to fully utilize the power of deep neural networks.

Bibliography

Baghfalaki, T., Ganjali, M. and Berridge, D., 2016. Missing Value Imputation for RNA-Sequencing Data Using Statistical Models: A Comparative Study. Journal of Statistical Theory and Applications, 15(3), pp.221-236.

Bao, J., Chen, D., Wen, F., Li, H. and Hua, G., 2017. CVAE-GAN: fine-grained image generation through asymmetric training. *arXiv preprint arXiv:1703.10155*.

Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H., 2007. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems* (pp. 153-160).

Buuren, S.V. and Groothuis-Oudshoorn, K., 2010. mice: Multivariate imputation by chained equations in R. *Journal of statistical software*, pp.1-68.

Chen, Y., Li, Y., Narayan, R., Subramanian, A. and Xie, X., 2016. Gene expression inference with deep learning. *Bioinformatics*, *32*(12), pp.1832-1839.

Duan, Y., Lv, Y., Kang, W. and Zhao, Y., 2014, October. A deep learning based approach for traffic data imputation. In Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on (pp. 912-917). IEEE.

Goodfellow, I., 2016. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

Isola, P., Zhu, J.Y., Zhou, T. and Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004.*

Karpathy, A., Abbeel, P., Brockman, G., Chen, P., Cheung, V., Duan, R., Goodfellow, I., Kingma, D., Ho, J., Houthooft, R., Salimans, T., Schulman, J., Sutskever, I. and Zaremba, W., 2016. *Generative Models*. [ONLINE] Available at: https://blog.openai.com/generative-models/. [Accessed 13 March 2018].

Kim, D., Langmead, B. and Salzberg, S.L., 2015. HISAT: a fast spliced aligner with low memory requirements. Nature methods, 12(4), p.357.

Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

Kukurba, K.R. and Montgomery, S.B., 2015. RNA sequencing and analysis. Cold Spring Harbor protocols, 2015(11), pp.pdb-top084970.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G. and Durbin, R., 2009. The sequence alignment/map format and SAMtools. Bioinformatics, 25(16), pp.2078-2079.

Li, J., Skinner, K.A., Eustice, R.M. and Johnson-Roberson, M., 2018. WaterGAN: unsupervised generative network to enable real-time color correction of monocular underwater images. *IEEE Robotics and Automation Letters*, *3*(1), pp.387-394.

Bibliography

Li, Y., Liu, S., Yang, J. and Yang, M.H., 2017, April. Generative face completion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Vol. 1, No. 3, p. 6).

Liao, Y., Smyth, G.K. and Shi, W., 2013. featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. Bioinformatics, 30(7), pp.923-930.

Liew, A.W.C., Law, N.F. and Yan, H., 2010. Missing value imputation for gene expression data: computational techniques to recover missing data from available information. *Briefings in bioinformatics*, *12*(5), pp.498-513.

Madeleine Price Ball, (2013), Genetic code [ONLINE]. Available at: https://upload.wikimedia.org/wikipedia/commons/3/37/Genetic_code.svg [Accessed 9 March 2018].

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B. and Lee, H., 2016. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*.

Shang, C., Palmer, A., Sun, J., Chen, K.S., Lu, J. and Bi, J., 2017. VIGAN: Missing View Imputation with Generative Adversarial Networks. *arXiv preprint arXiv:1708.06724*.

Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.A., 2008, July. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning* (pp. 1096-1103). ACM.

Xie, R., Wen, J., Quitadamo, A., Cheng, J. and Shi, X., 2017. A deep auto-encoder model for gene expression prediction. *BMC genomics*, *18*(9), p.845.

Zhao, S., Fung-Leung, W.P., Bittner, A., Ngo, K. and Liu, X., 2014. Comparison of RNA-Seq and microarray in transcriptome profiling of activated T cells. PloS one, 9(1), p.e78644.

Bibliography