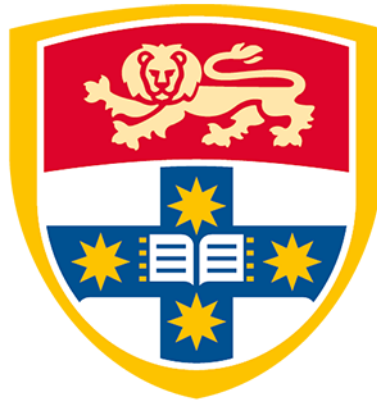


# Deep Learning Assignment 1

Lecturer: Chang Xu

Tutor: Kwon Nung Choi

*Classification Task performed on provided Dataset: A  
Latex report.*



THE UNIVERSITY OF  
SYDNEY

*Anshu Kumar (490517666)*  
*Sanna Nazir (490517677)*  
*Samarth Sehgal (490528857)*

April 2020

# Contents

<b>1</b>	<b>ABSTRACT</b>	<b>4</b>
<b>2</b>	<b>INTRODUCTION</b>	<b>4</b>
2.1	Aim (Goals and motivation) . . . . .	4
2.2	Dataset overview . . . . .	5
2.2.1	Dataset Attribute/Feature Distributions . . . . .	5
2.2.2	Visualize data in 2/3D (using PCA) . . . . .	5
<b>3</b>	<b>METHODS</b>	<b>6</b>
3.1	Data Pre-processing Techniques . . . . .	6
3.1.1	Standardization . . . . .	6
3.1.2	Encoding labels: One Hot Encoder . . . . .	7
3.2	Neural Network Modules (Required modules) . . . . .	7
3.2.1	Hidden Layers . . . . .	7
3.2.2	ReLU Activation . . . . .	7
3.2.3	Weight decay . . . . .	8
3.2.4	Momentum in SGD . . . . .	8
3.2.5	Dropout . . . . .	9
3.2.6	Softmax and cross-entropy loss . . . . .	9
3.2.7	Mini-batch training . . . . .	10
3.2.8	Batch Normalization . . . . .	10
<b>4</b>	<b>EXPERIMENTATION AND RESULTS</b>	<b>11</b>
4.1	Experiment setting . . . . .	11
4.1.1	Outline . . . . .	11
4.1.2	Performance Metrics . . . . .	11
4.1.3	Experiment steps . . . . .	11
4.2	Experiment Observations . . . . .	12
4.2.1	Hidden Layers and Neurons . . . . .	12
4.3	Fine Tuning the base network . . . . .	14
4.3.1	Learning Rate . . . . .	14
4.3.2	Weight Decay . . . . .	14
4.3.3	Momentum in SGD . . . . .	15
4.3.4	Dropout . . . . .	16
4.3.5	Mini-Batching . . . . .	17
4.3.6	Batch Normalization . . . . .	18
4.4	Analysis of results observed . . . . .	19
4.4.1	Comparative analysis of Results . . . . .	19
4.4.2	Ablation analysis of Results . . . . .	21
<b>5</b>	<b>DISCUSSION</b>	<b>22</b>
5.1	Final choice of algorithm and parameters . . . . .	22
5.2	Personal Reflection . . . . .	23
<b>6</b>	<b>CONCLUSION</b>	<b>24</b>
<b>7</b>	<b>BIBLIOGRAPHY</b>	<b>25</b>

<b>8</b>	<b>APPENDIX</b>	<b>26</b>
8.1	Hardware and Software specifications . . . . .	26
8.2	Instructions for running the code . . . . .	26

# 1 ABSTRACT

*Neural Networks have been gaining popularity and significance for a long time now. Since the 90's, global research interest in the application of neural networks restarted after an AI winter. Consequently, industry interest in the same increased as well. To the extent that in today's time, neural networks are a pervasive technology that has found application in virtually every sphere of human life. From education to healthcare, from manufacturing to energy, from automotive industry to the stock markets, from farming to space exploration: everything is touched by the impact of Neural Networks. This report is an exploration into the basics of a neural network. It explores the building blocks of a simple perceptron and extends to various improvement techniques for a multi-layered perceptron. Different modules and methods are experimented upon, analyzed and their results documented. The work is concluded with a self-reflection on the significance of the work and consequently, the significance of neural networks as well as potential future work for the researchers.*

## 2 INTRODUCTION

The purpose of this report is to analyze the building blocks of a neural network. To evaluate the performance of the neural network, two parameters: Accuracy and Loss, are defined and compared across different modules. The data for performing this task has been provided by the unit coordinator for COMP5329, taught at the University of Sydney.

### 2.1 Aim (Goals and motivation)

The motivation and goals underlying this report/project can be categorized into two major brackets:

At its core i.e. microscopically, the project is aimed at enhancing the students technical capabilities with respect to building neural networks from scratch. It also focuses on building the student's understanding of how a neural network works, and how that mathematical working can be translated into a coded application for implementation. Furthermore, by enforcing implementation of various modules such as dropout, batch normalization, weight decay, SGD momentum etc., this project encourages the student to understand the significance of each of these modules, their impacts on the neural network's performance metrics as well as the process required to integrate these modules into the code. Thus, at the student/individual level, the aim of this project is to build the technical and analytical skills, pertinent to neural networks, of the student.

Looking at the bigger picture i.e. the macroscopic level, the project is designed to enhance the understanding and importance of neural networks to current human life. It encourages the student to research and inculcate the broad range of applications for deep neural networks. By building the network from scratch, it also encourages the student to dive deeper into the foundational knowledge of these networks and to build the skill of tailoring neural networks to the application/problem at hand. At a macroscopic level, the project is aimed at understanding the design processes, evaluation metrics and significance of deep learning networks to real-world problems.

Microscopically, the key tasks of this project are: Redefining the provided Multi-Layer Perceptron code, enhance the MLP using prescribed modules to be implemented, analyze and compare the impact of these modules on the network and finally, self-reflect on the project and consequent future work.

## 2.2 Dataset overview

The data for performing this task has been provided by the unit coordinator for COMP5329, taught at the University of Sydney. The dataset is divided into three sub-files:

1. The training data: is a file with 60,000 data instance. Each data instance has 128 attributes. Thus, the file converts into an array of shape (60000, 128).
2. The training labels: is a file with 60,000 data points each representing the class of the corresponding data instance in the training data file. These labels help identify the class of each instance in the training data.
3. The testing data: is a file with 10,000 data instances that are to be classified using the neural network we build. The labels generated for these files will be submitted in a separate predicted output file.

### 2.2.1 Dataset Attribute/Feature Distributions

The attributes of the dataset are not contextually defined for the purposes of our assignment. The provided data is simply an array of numbers and has 128 features to be learnt by the neural network. While the attributes are just numbers without any contextual information, we can still visualize and understand their distributions within their attribute domain. Since the given number of 128 features is large, we have visualized only some of the feature distributions in this report.

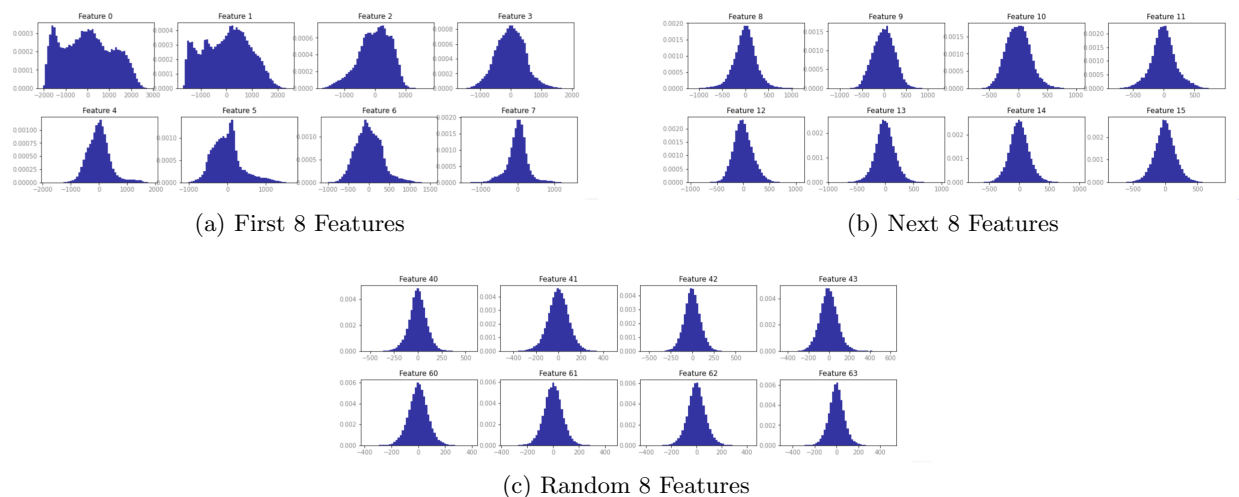


Figure 1: Data attributes distribution visualized

### 2.2.2 Visualize data in 2/3D (using PCA)

The Principal Component Analysis method has been used to visualize the data in 2 and 3 dimensions. The 2 dimensional visualization captures about 50% of the data's variance along the two principal axes. The 3 dimensional visualization captures 56% of the same. This method has just been employed for data exploration purposes and does not add any significance to the neural networks functionality.

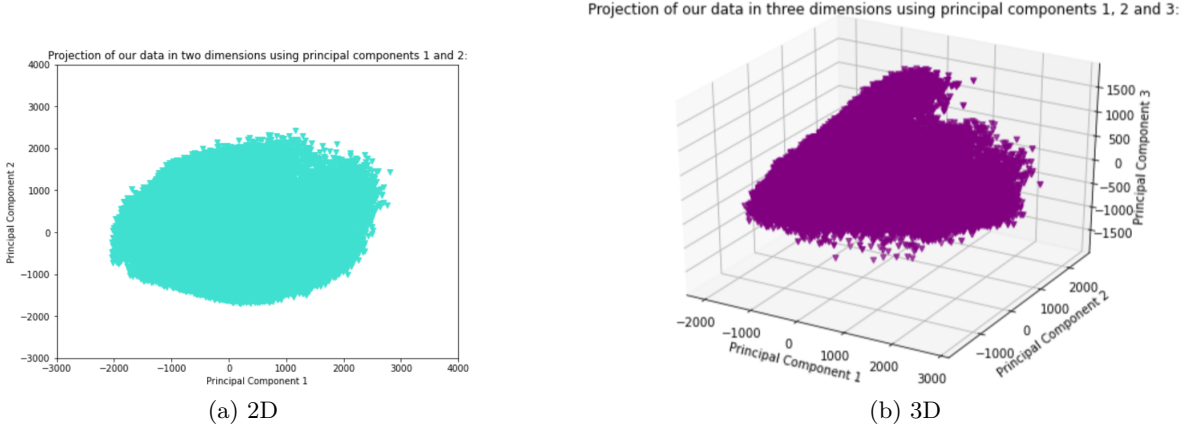


Figure 2: Data visualization in two and three dimensions

### 3 METHODS

#### 3.1 Data Pre-processing Techniques

“Pre-processing data is an essential step to enhance data efficiency [1]”. In machine learning, deep learning and essentially all data based technologies, data pre-processing is a pre-requisite. Data in its raw form may be biased, largely varying, unscaled, incomplete, inconsistent, filled with outliers etc. This raw data with flaws can result in a flawed algorithm that gives garbage results. Thus, this step is the first and most important step towards building any data-based application.

Pre-processing can involve any methods that assist in filling missing data, reducing the deviations within the data, standardizing the data, encoding categorical variables etc. For the purposes of our project, we have performed the following preliminary data pre-processing steps:

##### 3.1.1 Standardization

The standardization technique is used to control the range of values that an attribute can take. It reduces the bounds of the data and standardizes all the attributes to a similar scale. This can be achieved in many ways such as Min-Max scaling, Max Absolute scaling, Z-score Normalizing etc. For the purposes of our report, we have used Z-score normalization.

Our standardization technique (z-score method) can be depicted mathematically as:

$$z = \frac{x_i - x_{mean}}{\sigma}$$

where:

$x_i$  is a data point

$x_{mean}$  is the mean of the data points across a feature

$\sigma$  is the variance of that feature.

Attribute values can range from simple single digit values to large values in thousands to even smaller decimal values. This creates a variation across different attributes that can cause the algorithm/network to assign different weights to attributes based on their size. Thus, distorting

the weight-age of our features merely because of size of the values in the domain. Which is why, standardizing is an important step.

### 3.1.2 Encoding labels: One Hot Encoder

For a multi-class classification, each class may be represented by a natural number (as is the case with our data). However, when we feed the classes to the algorithm/network as is, it may interpret the categorical labels incorrectly as continuous data points. This means that the network may assume the labels (which are just categories) to have some dependencies on each other (say it assumes them to be in an increasing order). This will impact our end result and give flawed outputs. Thus, we perform one-hot encoding of the labels to ensure that the network treats them as separate, independent categories (classes).

```
Original labels:
[9 0 0 3 0 2 7 2 5 5]

Corresponding labels after one hot encoding:

[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
```

Figure 3: One Hot encoding

## 3.2 Neural Network Modules (Required modules)

### 3.2.1 Hidden Layers

For the purposes of this project, a layer is defined as the combination of weights and activation function connecting two neurons. Thus, a 3-layered network would have four sets of neurons. Hidden layers provide the network with its weight for each feature that maps between two neurons along with a bias term. It also provides non-linearity to the network by using an activation function to arrive at results.

### 3.2.2 ReLU Activation

ReLU stands for Rectified Linear Unit and is one of the most commonly used type of activation function. It is dead if the input is zero or negative and is equal to the input value if input is positive [7]. Mathematically, it can be defined as  $y = \max(0, x)$  and is represented graphically as follows:

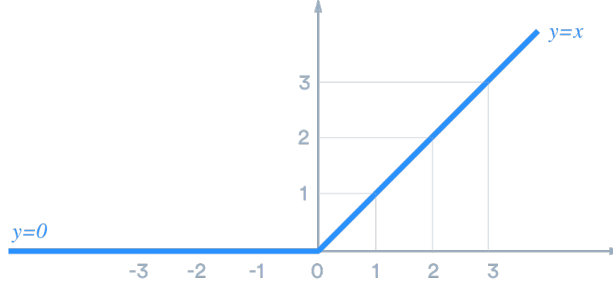


Figure 4: ReLU Activation function

The advantages of Relu are that it is cheap to compute, converges faster and is sparsely activated. It has a problem called “Dying ReLu” (derivative for  $x \leq 0$  is zero). To counter that we use Leaky ReLu or Parametric ReLu activation functions [7], both of which are as displayed in the graph below:

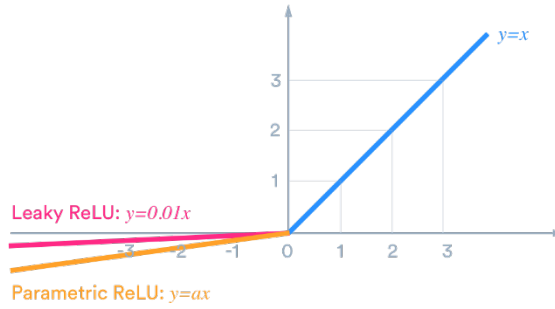


Figure 5: Leaky and Parametric ReLU Activation functions

### 3.2.3 Weight decay

To prevent over-fitting of our model, we try to introduce an extra term to the loss function to reduce the model’s complexity [8], i.e., giving less importance to terms of higher complexity. This ensures weights are not increased by a great amount and is hence called “Weight Decay”. It is implemented using different methods, most common of it being L2 Regularization. L2 Regularisation adds the square of the weights (multiplied by a certain regularisation term) to the loss function in order to reduce the complexity.

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2 + \lambda \sum_{i=1}^n |\theta_i|$$

Figure 6: Weight Decay Mathematical representation

### 3.2.4 Momentum in SGD

In neural networks, momentum with SGD is a method to accelerate the gradient vectors in the right directions. It helps in navigating ravines easily and dampens oscillations [5].



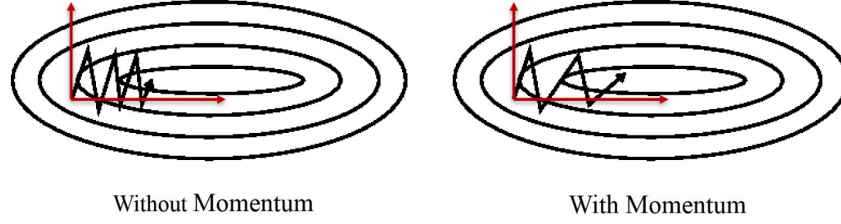


Figure 7: Graphical representation for momentum

The momentum term (usually set to 0.9 or similar value) fits in the SGD equations as follows:

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

$$\theta_t = \theta_{t-1} - v_t$$

Figure 8: Mathematical representation for momentum

### 3.2.5 Dropout

Dropout is a regularisation method that helps in approximation of a large number of neural networks with different architectures in parallel [3]. It helps in alleviating over-fitting in training phase. In this technique, nodes with all their incoming and outgoing connections are temporarily dropped from a network layer to get a configured layer for training purposes.

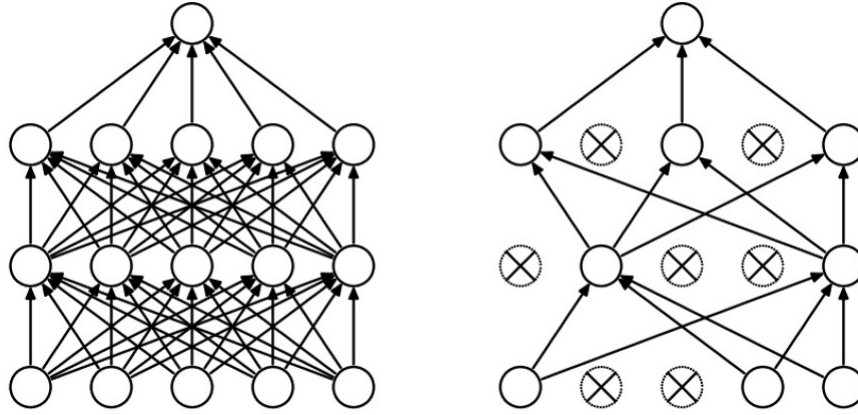


Figure 9: Graphical representation for dropout

### 3.2.6 Softmax and cross-entropy loss

In a multi-class neural network problem, the softmax function assigns a decimal probability to each class [6]. It is implemented on the last layer of the neural network just before the output layer and has number of nodes equal to the output layer. The equation for the softmax function is as follows:

$$p(y = j|\mathbf{x}) = \frac{e^{(\mathbf{w}_j^T \mathbf{x} + b_j)}}{\sum_{k \in K} e^{(\mathbf{w}_k^T \mathbf{x} + b_k)}}$$

Figure 10: Softmax Equation

Cross Entropy loss is used to measure how well the softmax function is performing. It measures the difference between the true probability distribution,  $p$ , and the estimated probability distribution,  $q$ . It is represented mathematically as:

$$H(p, q) = - \sum_{\forall x} p(x) \log(q(x))$$

Figure 11: Cross Entropy Equation

### 3.2.7 Mini-batch training

Mini Batch Training is a variation of the gradient descent algorithm in which the training dataset is divided into small groups or batches that are used to find the error and model coefficients [2]. Mini batch gradient descent tends to find the middle ground between the efficiency of the batch gradient descent and the robustness of the stochastic gradient descent.

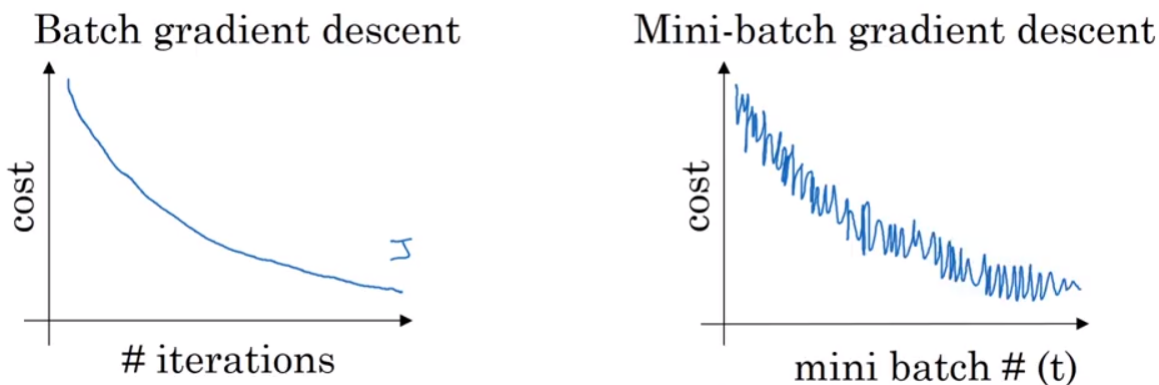


Figure 12: How mini-batch changes the gradient descent [4]

### 3.2.8 Batch Normalization

Batch Normalisation is a technique very commonly used in neural networks to improve the speed, the performance and the stability of the artificial neural networks. It essentially performs the whitening to inner layers of the multiple neural network model and helps in reducing co-variance shift and demand for regularization.

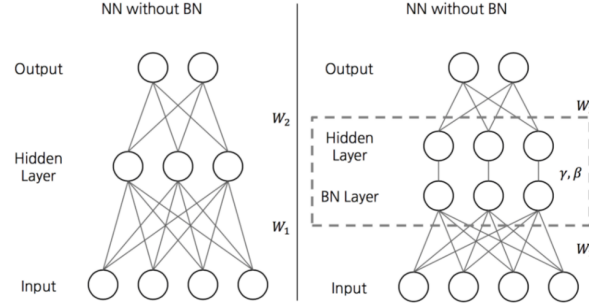


Figure 13: Graphical Representation of a batch normalization layer

## 4 EXPERIMENTATION AND RESULTS

### 4.1 Experiment setting

#### 4.1.1 Outline

For this classification task, dataset has been provided by the lecturer. Along with some base code and modules including the structure of the neural network, optimization methods for the gradient, regularization techniques etc. Further experiment is to be performed by fine tuning the base structure of the neural network and achieving optimal performance. Additionally, some more modules such as dropout, batch normalization, mini-batching etc. are required to be integrated into the neural network.

#### 4.1.2 Performance Metrics

The two key performance metrics used to measure the performance of each regression algorithm are:

##### 1. Accuracy:

Accuracy is measured in terms of the correct classifications done by our network. That is, what percentage of labels generated by the network are actually correct. Accuracy can be defined as:

$$Accuracy = \frac{\text{number of correct classifications}}{\text{total number of test examples}} * 100$$

##### 2. Loss:

In lay man's terms, loss may be defined as the difference between our expected results and our achieved results. Lesser the loss value, better our network is at predicting correct labels for the data. For the purposes of this project, we have used the "Cross Entropy Loss" Function.

#### 4.1.3 Experiment steps

The steps performed within our project are as follows:

1. Pre-processing of the dataset i.e. standardization as well as label encoding.
2. Splitting of the dataset (60,000 images) into the training and validation sets (80:20 division).

3. Start by experimenting with the hidden layers and neurons. Analyze the accuracy, loss and time taken for each combination. (Detailed analysis and observations for this can be found in section 4.2.1.)
4. Select the optimal number of hidden layers and neurons to be added.
5. Perform hyper-parameter tuning on this base network structure to analyze the impact of such changes.
6. Document all observations and derive meaningful results and conclusion.

## 4.2 Experiment Observations

Following the above experiment steps, observations are recorded as below. These observations have been recorded after the parameter tuning of the base neural network that is selected to possess 3 hidden layers with 128 input neurons followed by (1024, 64) hidden neurons and concluded with 10 output neurons.

### 4.2.1 Hidden Layers and Neurons

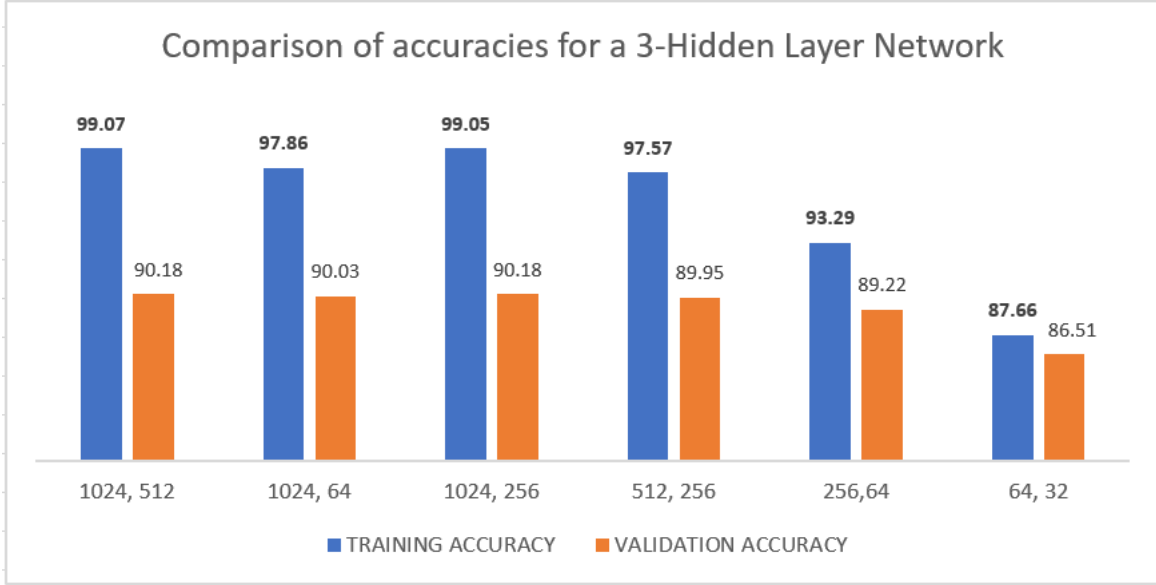
To start the experiment, we first need to initialize the structure of the neural network. This includes choosing the number of hidden layers we would like to have in our network as well as the number of neurons at each end of those layers. The first neuron stage would obviously possess 128 neurons as we have a dataset with 128 features. Similarly, since we are performing multi-class classification with cross-entropy loss, we have mapped the output layer to 10 neurons for each class.

Following this, we have tested various combinations of number of neurons and number of hidden layers as can be seen in the figure below.

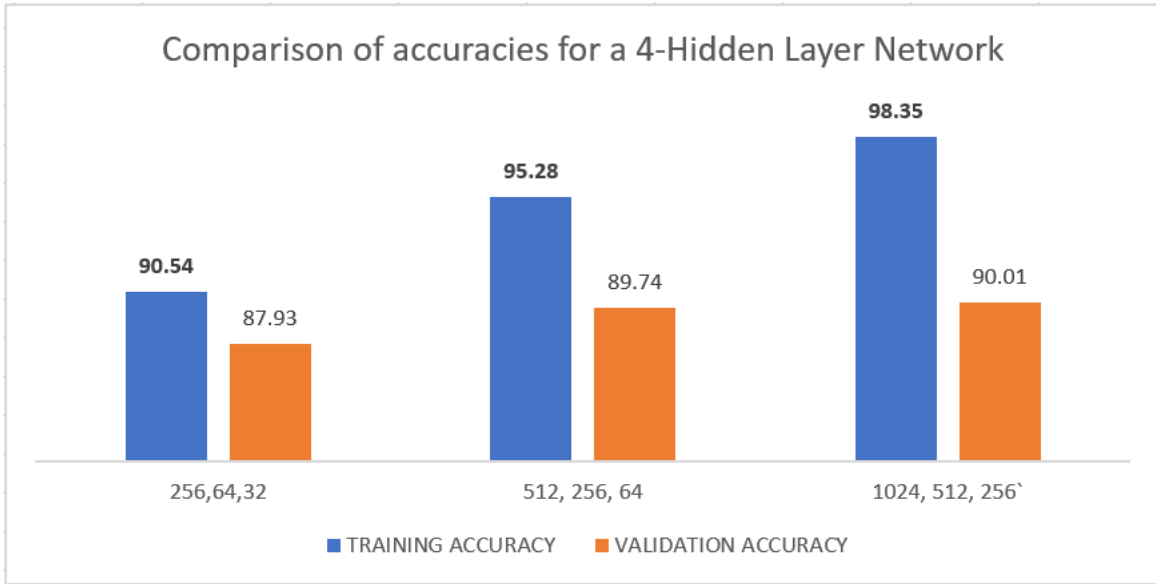
HIDDEN LAYERS AND NEURONS					
NN LAYERS	NEURONS ADDED BETWEEN I/P AND O/P	TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
3	1024, 512	72 minutes, 20.8 seconds	99.07	90.18	0.09
	1024, 64	28 minutes, 22.9 seconds	97.86	90.03	0.13
	1024, 256	61 minutes, 49.3 seconds	99.05	90.18	0.08
	512, 256	42 minutes, 54.3 seconds	97.57	89.95	0.12
	256, 64	9 minutes, 2.9 seconds	93.29	89.22	0.25
	64, 32	1 minutes, 11.0 seconds	87.66	86.51	0.46
4	256, 64, 32	7 minutes, 41.2 seconds	90.54	87.93	0.33
	512, 256, 64	34 minutes, 46.0 seconds	95.28	89.74	0.17
	1024, 512, 256	92 minutes, 49.9 seconds	98.35	90.01	0.09

Figure 14: Observations for choosing number of Hidden Layers and associated neurons

We stopped the experimentation at 4 hidden layers as sufficient accuracy (99% in best case) was already achieved in the training data.



(a) Accuracy comparisons for 3 layer network



(b) Accuracy comparisons for 4 layer network

Figure 15: Accuracy across tested hidden layers and neurons

After careful analysis of the observations, we concluded that for the purposes of this project, we would choose the **3-Hidden Layered Network with 1024, 64 neurons** between the input and output neurons. All further testing and observations are based on this base structure of the network.

*NOTE: The base structure is incorporated with the modules of dropout (35%), ReLU activation on layers except the output layer that has softmax activation, momentum parameter of 0.7, no batch normalization, L2 regularization parameter of 0.003 and a batch size of 500 for training within an epoch.*

### 4.3 Fine Tuning the base network

Using the above base network, we have performed experimentation across various parameters. This underlines the basis of our ablation and comparison analysis detailed in sections 4.4 followed by the discussion in section 5.

#### 4.3.1 Learning Rate

Tuning the learning rate on the base neural network, we observed the following data:

LEARNING RATE				
LEARNING RATE	TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
0.01	30 minutes, 56.9 seconds	96.05	89.6	0.19
0.02	28 minutes, 22.9 seconds	97.86	90.03	0.13
0.03	29 minutes, 28.9 seconds	98.5	90.03	0.1
0.04	21 minutes, 6.8 seconds	98.48	90.08	0.1
0.05	21 minutes, 13.5 seconds	98.56	89.98	0.09

Figure 16: Observations for different learning rates on the neural network

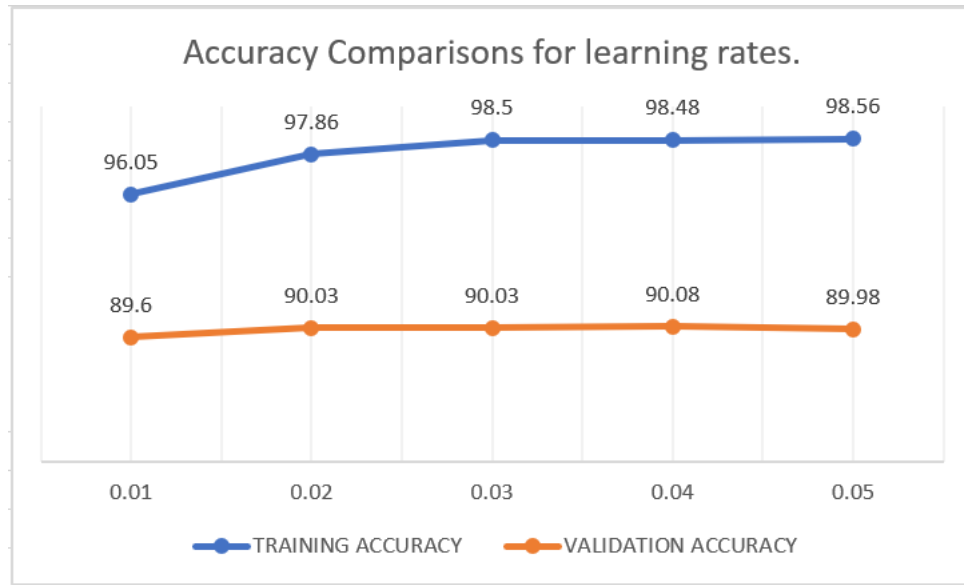


Figure 17: Accuracy comparisons for different learning rates.

#### 4.3.2 Weight Decay

Weight decay has been executed by means of L2 Regularization technique. For the same, lambda parameter is varied to understand its impact on the network. Recorded observations are:

WEIGHT DECAY (USING L2 REGULARIZATION)				
LAMBDA VALUE	TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
0.001	40 minutes, 39.6 seconds	98.52	90.03	0.1
0.002	21 minutes, 56.2 seconds	97.11	89.97	0.16
0.003	28 minutes, 22.9 seconds	97.86	90.03	0.13
0.004	32 minutes, 21.5 seconds	98.14	89.9	0.13
0.005	42 minutes, 35.2 seconds	98.55	90.18	0.1

Figure 18: Observations for varying lambda i.e. weight decay on the neural network

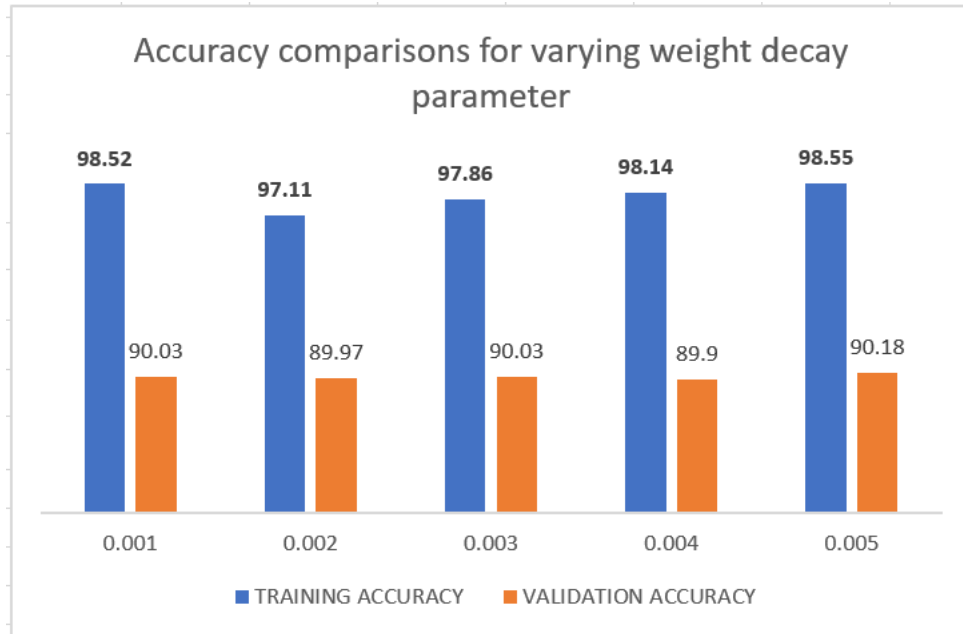


Figure 19: Accuracy comparisons for varying lambda.

### 4.3.3 Momentum in SGD

Gradient optimization for the network has been achieved using the momentum technique. To understand the significance of momentum on the network, the gamma parameter is varied. Thus, the importance of momentum while calculating the gradient varies. The following observations were discovered:

MOMENTUM				
GAMMA	TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
0.5	20 minutes, 41.4 seconds	99.46	89.73	0.06
0.7	28 minutes, 22.9 seconds	97.86	90.03	0.13
0.8	33 minutes, 28.8 seconds	99.48	89.8	0.04

Figure 20: Observations for varying gamma i.e. momentum parameter on the neural network

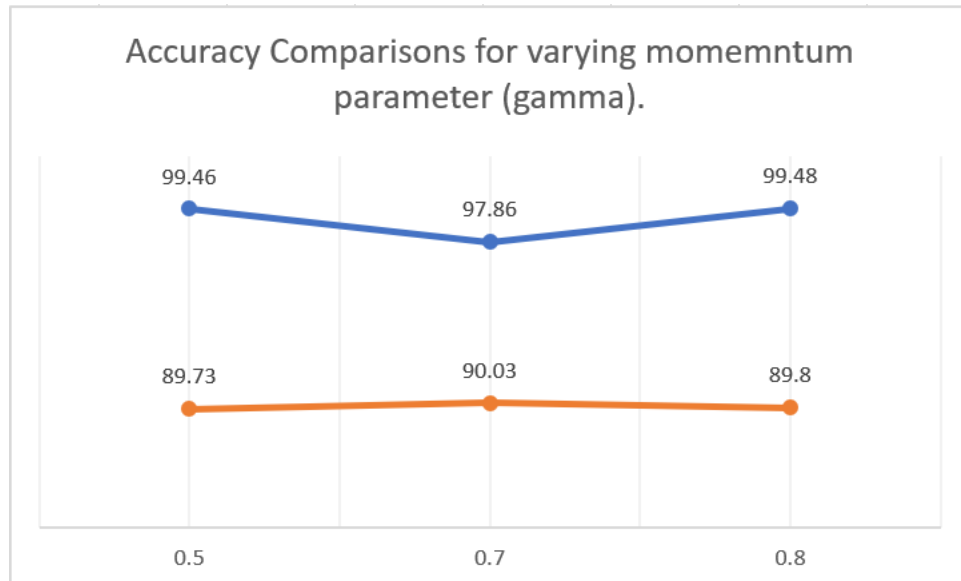


Figure 21: Accuracy comparisons for varying gamma/momentum rate.

#### 4.3.4 Dropout

For the purposes of our project, dropout percentage signifies the percentage of neurons that are turned off during the processing of the neural network. That is, there neurons do not partake in the training of our network. Recorded observations are as under:



DROPOUT				
PERCENTAGE	TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
0.25	27 minutes, 37.6 seconds	99.58	89.88	0.05
0.35	28 minutes, 22.9 seconds	97.86	90.03	0.13
0.45	23 minutes, 29.7 seconds	96.94	90.08	0.15
0.55	37 minutes, 27.6 seconds	95.6	89.98	0.18

Figure 22: Observations for different dropout percentages on the neural network

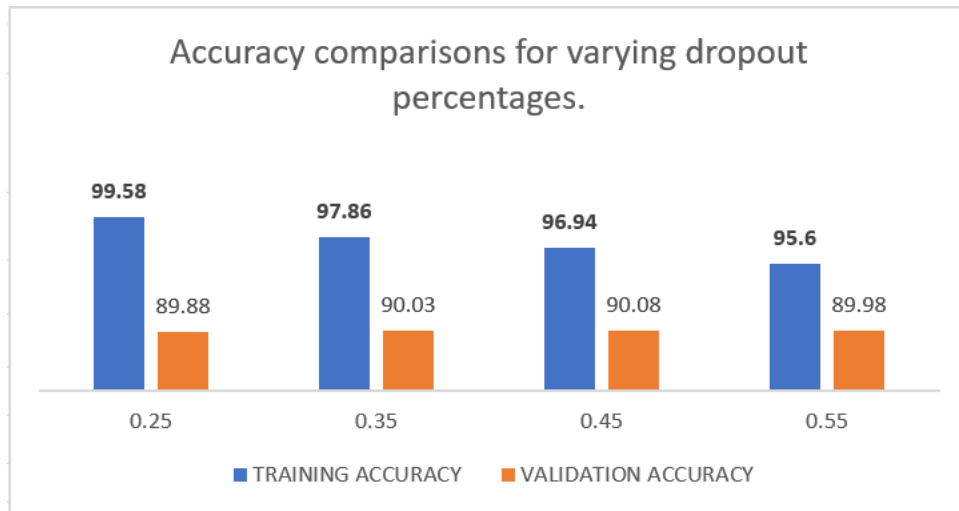


Figure 23: Accuracy comparisons for different dropout rates.

#### 4.3.5 Mini-Batching

Mini-batching signifies the batch size for which gradient update will take place within an epoch. We have experimented with varying batch sizes and have recorded the observations as below:

MINI-BATCHING				
BATCH SIZE	TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
300	22 minutes, 48.2 seconds	98.3	90.1	0.09
500	28 minutes, 22.9 seconds	97.86	90.03	0.13
800	32 minutes, 10.9 seconds	98.49	90.24	0.1

Figure 24: Observations for varying batch sizes in an epoch.

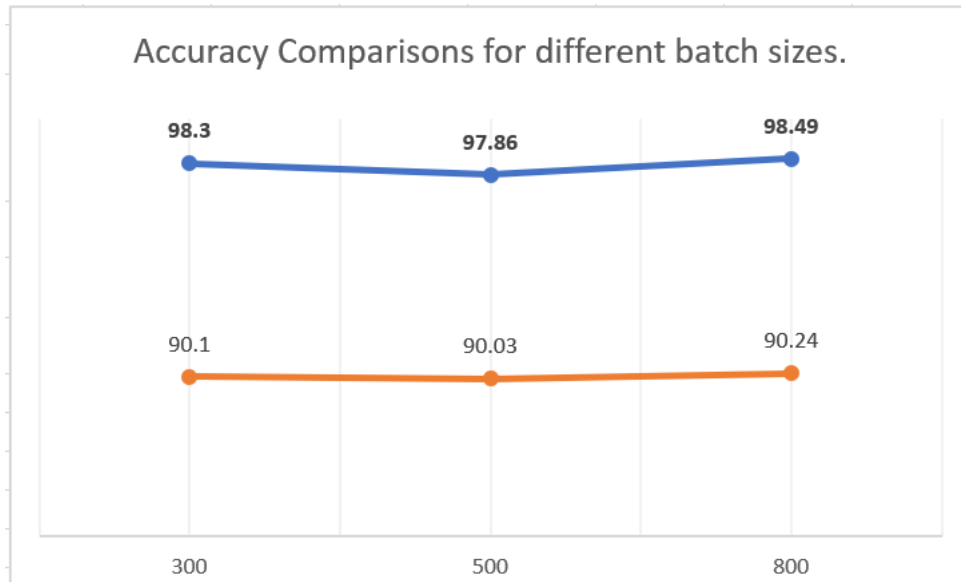


Figure 25: Accuracy comparisons for varying batch sizes.

#### 4.3.6 Batch Normalization

Batch normalization has been used as a toggle-enabled parameter. That is, it can be turned on or off for the neural network. Recording the observations for both the cases:

BATCH NORMALIZATION				
USED?	TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
TRUE	34 minutes, 34.6 seconds	95.58	89.65	0.2
FALSE	28 minutes, 22.9 seconds	97.86	90.03	0.13

Figure 26: Observations for batch normalization on the neural network

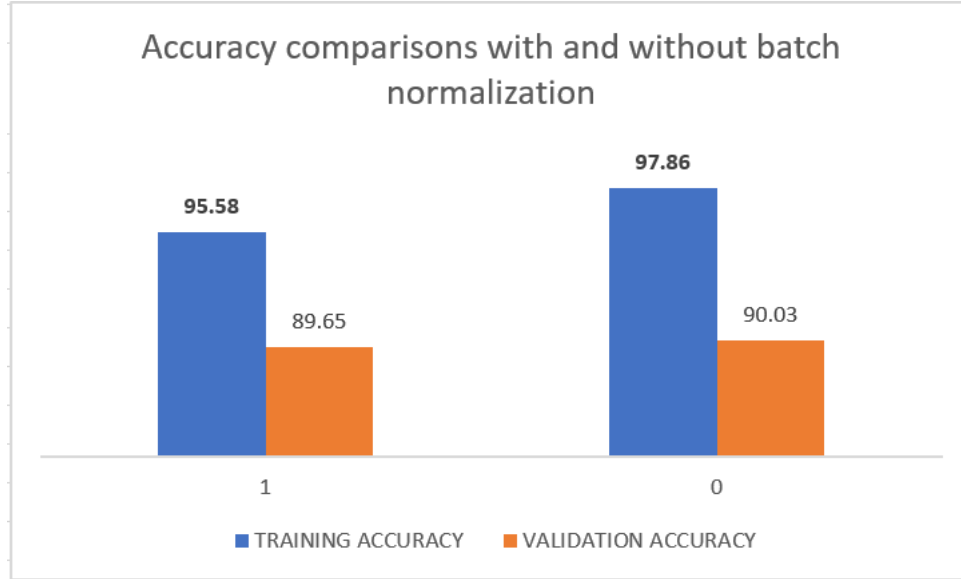


Figure 27: Accuracy comparisons for different batch normalization.

## 4.4 Analysis of results observed

### 4.4.1 Comparative analysis of Results

As can be seen in the above observations, extensive comparative experimentation has been done for various building blocks of the neural network.

- **HIDDEN LAYERS AND NEURONS**

Referring to figure 14, we can see that changing the number of layers between the input and output neurons creates significant impacts on the models performance factors. Increasing the number of layers, obviously, leads to increase in the training time of the model. A best-case accuracy of 90.2% on the validation dataset is achieved with 3 layered ([1024,256], [1024,512]) network and 4 layered ([1024,512,256]). However, all of these networks require a training time of more than an hour, ranging from 61-92 minutes. Thus, there is a trade-off between the accuracy and time.

Comparing these metrics across other observations, we can see that the 3 layered network [1024,64] has an almost comparative performance with validation accuracy at 90% while having a training time of almost half (ranging from 20-30 minutes) of the above networks.

*Thus, for the purposes of this project, we have chosen the [1024,64] network to train our network within a feasible time with the best performance possible (accuracy trade-off of only 0.2%)*

- **LEARNING RATE**

Referring to figure 16, we can see that increasing the learning rate decreases the time of training for the network. This resonates with our intuition that the net will train more quickly with a larger learning rate. This is because, the gradient takes larger steps in moving towards the minimal loss point. The accuracy for the validation set increases with the first step increase to 0.02 but decreases when we increase the rate too much to 0.05. Between 0.02 to

0.04, accuracy remains relatively constant with a decrease in the time taken for training.

*Thus, for the purposes of this project, we have chosen our learning rate to be 0.04 since it gives high accuracy with a comparatively smaller training time.*

- **WEIGHT DECAY**

Referring to figure 18, varying the lambda value for the L2 Regularization parameter (weight decay process) has a significant impact on the timing of the neural network's training as well as creates variations in accuracy.

Comparing the values of lambda, we can see that the validation accuracy for 0.001 and 0.003 values is the highest. However, the time taken in training for the 0.003 value is significantly lesser than 0.001. This effect might be observed because at the lower value, the model puts high impact on learning the training data completely. As such, we run the risk of over-fitting our data.

*Thus, for the purposes of this project, we have chosen the lambda value of 0.003 to avoid the risk of over-fitting as well as handling the time for training.*

- **MOMENTUM (GAMMA)** Referring to figure 20, we can see that varying the momentum parameter for the gradient learning process affects the timing and accuracy of the model. Keeping the parameter low (0.5), time taken is lesser but validation accuracy is decreased. On the other hand, if the parameter is increased (0.8), training time increases while accuracy is still less.

This effect is observed as the momentum parameter gamma defines the stride that the gradient takes in the direction of the previous gradient. As such, lesser value seems to be converging quickly but larger value takes time in converging. This may be because the the gradient gets stuck and oscillates around the minimum loss point.

*Thus, for the purposes of this project we have chosen our momentum rate to be 0.7 as it gives optimum timing as well as higher accuracy.*

- **DROPOUT**

Referring to figure 22, variations in dropout can create varying impacts on our performance metrics. This impact is created because some of the neurons are turned off while training of the model. A low or high dropout rate results in decreased validation accuracy but an intermediate rate results in higher accuracy.

*For the purposes of this project, we have chosen the dropout rate of 0.25 to allow our network to learn the data patterns while also allowing it to not over-fit on the dataset.*

- **MINI-BATCHING**

Breaking the dataset into batches allows the gradient learning process to be broken into smaller batches of learning. Thus, each epoch has multiple update steps from the smaller batches of the gradient. Mini-batching is an extension of the Stochastic gradient descent and overcomes the issues of SGD while harnessing the advantages of the batch gradient descent.

Referring to figure 24, we can see that a large batch size is resulting in larger accuracy but also a larger training time. On the other hand, a smaller batch size results in lesser accuracy with lesser training time.

*For the purposes of this project, we have thus chosen a batch size of 500 to trade-off the timing and accuracy metrics.*

- **BATCH NORMALIZATION** Adding a batch normalization layer over the network layers has been enabled as a true/false feature. Referring to figure 26, we can see that validation accuracy for both cases is comparable. However, using the batch norm layers causes the network to slow down and thus, training time increases significantly. This effect is observed because the network has to normalize the incoming data at every layer and thus, an overhead computation is added.

*Thus, for the purposes of this project, we have chosen to favor time and keep the batch normalization toggled off.*

#### 4.4.2 Ablation analysis of Results

Ablation studies are carried out by removing one or many modules of a deep neural network to understand their significance on the network. To carry out the analysis, some modules of the net were removed and observations recorded as below:

ABLATION ANALYSIS					
PARAMETER	VALUE	TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
<b>Gamma (Momentum)</b>	0	23 minutes, 54.5 seconds	99.34	90	0.06
<b>Dropout</b>	0	50 minutes, 1.1 seconds	92.53	84.63	0
<b>Lambda (Weight Decay)</b>	0	29 minutes, 55.9 seconds	99.35	90.21	0.07
<b>Batch Normalization</b>	FALSE	28 minutes, 22.9 seconds	97.86	90.03	0.13
<b>Mini-Batch</b>	48000	36 minutes, 42.8 seconds	87.38	86.06	0.44

Figure 28: Observations recorded after removing various modules (each individually) within the neural network

Analyzing these observations for each parameter:

- **Gamma (Momentum):**  
By turning off the gamma parameter, loss was greatly decreased for the network and training accuracy reaches almost 100%. This follows intuition as the network trains completely with respect to the available dataset. However, this would result in over-fitting and may even result in wrong predictions on new data.

- **Dropout:**

Removing the dropout feature, the network takes a very long time in training and loss is reduced to 0. Furthermore, the validation accuracy falls even below 85%. This follows intuition as removing dropout would mean that the network trains the dataset across each neuron. As such, the noise within the data is also modelled into patterns for prediction. Consequently, resulting in over-fitting and reduced accuracy for new data.

- **Lambda (Weight Decay):**

Removing the regularization factor by setting lambda to zero increases the training time of the network while increasing the training accuracy by a large amount (almost 100%). This signifies that the model has created the prediction function purely based on the available dataset and over-fits the same. This follows intuition as regularization acts as a technique to handle over-fitting and removing it would result in an almost cent percent accuracy on the training data.

- **Batch Normalization:**

Removing the batch normalization functionality for our algorithm ensures that it trains in a lesser time than using batch normalization. For the purpose of this dataset, the network can do without batch norm. However, in case of a much complex and larger dataset, batch normalization may be required (despite the timing trade-off) to compensate for exploding or vanishing gradients.

- **Mini-Batch:**

Setting the mini-batch size to the size of the entire dataset available means that the mini-batching functionality is not used. Thus, this has been done to understand the impact of not performing mini-batching. As can be observed from the figure below, both training and validation accuracy's decrease significantly because the gradient updates performed are much lesser in number than otherwise. Because of this, the algorithm would need many more epochs to train better and as such, much more training time as well.

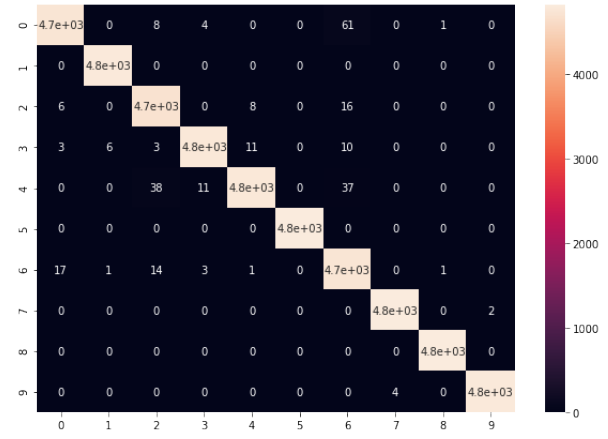
## 5 DISCUSSION

### 5.1 Final choice of algorithm and parameters

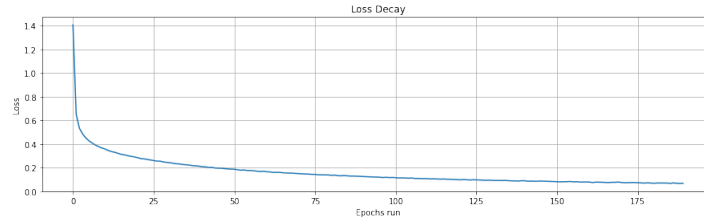
Based on the above observations and justification of choices, the final network has the following parameters and results:

FINAL ALGORITHM			
TIME TAKEN FOR TRAINING	TRAINING ACCURACY	VALIDATION ACCURACY	LOSS
20 minutes, 14.0 seconds	99.44	89.63	0.067
PARAMETERS USED			
Hidden Layers & Neurons	3-layered (1024, 64)		
Learning Rate	0.04		
Momentum (Gamma)	0.7		
Mini-Batch Size	500		
Dropout Percentage	0.25		
Weight Decay (Lambda)	0.003		
Primary Activation	ReLU		
Activation on Output layer	Softmax		
Batch Norm	FALSE		
Loss Function	Cross Entropy		

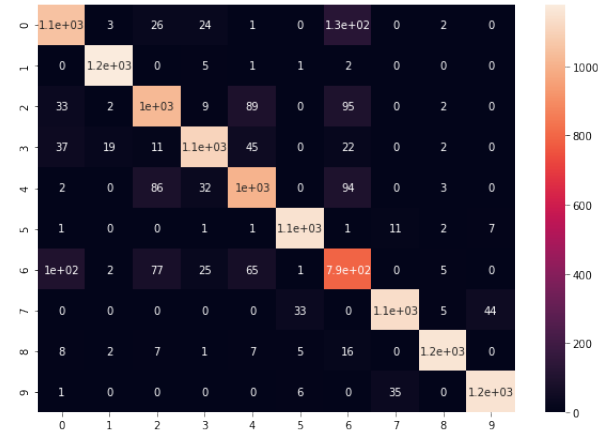
(a) Final Network Structure and outputs



(b) Confusion Matrix for the training dataset



(c) Plotting the loss against the epochs



(d) Confusion Matrix for the validation dataset

Figure 29: The Final Algorithm/Neural Network.

## 5.2 Personal Reflection

The results obtained from the implementation of this project are two-fold:

Microscopically, at the student level, we have learned to implement a neural network from scratch without employing any available libraries. We have performed the basic tasks of integrating the neural network, creating and modifying the hidden layers, defining the loss functions and

then training the network by updating the weights and biases. Following the basic structure, we also learnt how to handle for problems like exploding/vanishing gradients, over-/under-fitting etc. by incorporating techniques like batch normalization, regularization's etc. This project has helped us build the skills for performing a deep learning task (specifically classification) along with trouble shooting abilities within the built network. To conclude, it has strengthened our grass roots ability of design and implementing a deep neural network from scratch.

Macroscopic-ally, looking at the big picture, the project has helped us understand the significance of deep neural networks. It has driven us to understand the importance deep learning for solving complex, data-heavy problems. It taught us to model and tailor our networks to solve a problem statement represented by nothing else than simple data points. Furthermore, the project also taught us about trade-offs and sacrificing one parameter for the sake of another or building a compromise that suits your problem and provides significant average results across all performance metrics. To conclude, it has imbibed a problem-solving skill within the students and equipped us with the tool of deep learning.

## 6 CONCLUSION

After extensive experimentation, analysis and reflection, we conclude that for the purpose of this project, our designed algorithm (specifications as in figure 29a) appears to be the best fit. To satisfy the constraints of "feasible time" and "top accuracy," we have tuned various parameters within the network to achieve a quick training speed ( 20 minutes) and high validation accuracy ( 90%). The pre-processing technique of one-hot encoding/label encoding also holds a significant position in the multi-class classification problem.

Reflecting on the future work for such a project, we believe further experimentation can be carried out to understand the impact of various other modules on the dataset as well. For regularization, other techniques than L2 can be tested and analyzed. Dropout in terms of channels instead of neurons (as executed here) can be a next testing step. Furthermore, variations in the length and breadth of the network, varying activation functions at each layer, different loss functions, different gradient optimization techniques (other than Momentum e.g, RMSProp, Adam, AdaGrad etc.) are all probable next steps in the project. By further testing and experimentation along these lines, accuracy and timing may be improved along with enhancing our knowledge bases.



## 7 BIBLIOGRAPHY

### References

- [1] Suad A Alasadi and Wesam S Bhaya. Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, 12(16):4102–4107, 2017.
- [2] Jason Brownlee. A gentle introduction to mini-batch gradient descent and how to configure batch size. <https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/>, Last accessed on 2020-04-30.
- [3] Jason Brownlee. A gentle introduction to dropout for regularizing deep neural networks, 2018. <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>, Last accessed on 2020-04-30.
- [4] Imad Dabbura. Gradient descent algorithm and its variants, 2017. <https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3>, Last accessed on 2020-04-30.
- [5] Nici Schraudolph Genevieve Orr and Fred Cummins. Momentum and learning rate adaptation, 1999. <https://www.willamette.edu/~gorr/classes/cs449/momrate.html>, Last accessed on 2020-04-30.
- [6] Google. Multi-class neural networks: Softmax. <https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax>, Last accessed on 2020-04-30.
- [7] Danqing Liu. A practical guide to relu, 2017. <https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>, Last accessed on 2020-04-30.
- [8] Dipam Vasani. This thing called weight decay, 2019. <https://towardsdatascience.com/this-thing-called-weight-decay-a7cd4bcfccab>, Last accessed on 2020-04-30.

## 8 APPENDIX

### 8.1 Hardware and Software specifications

The code was implemented using Google Colabratory Open Source Computing Platform. Thus, resources provided by Google were used. This comprised of their standard CPU configuration with a RAM space of 12 GB provided.

However, we used a laptop for running the same online, which has below features:

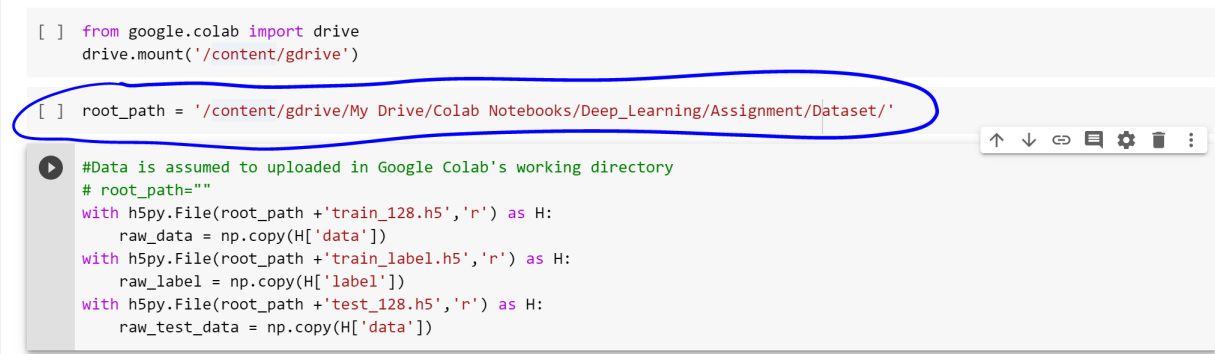
- Processor: Intel i7
- OS: Windows 10
- RAM: 8 GB
- GPU: 4 GB
- Model: HP Pavilion 360

### 8.2 Instructions for running the code

Follow the below instructions to run the ipynb file attached in the submission:

1. Upload the .ipynb file on Google Colab (<https://colab.research.google.com/>)
2. Upload the required datasets on your google drive. The default directory is set as “My Drive/Colab Notebooks/Deep.Learning/Assignment/Dataset/” in our code. Change this path to the user’s preferred location within their google drive by changing the file path inside root\_path variable declared in section 2 of the code.

#### ▼ 2. IMPORTING DATA



```
[ ] from google.colab import drive
drive.mount('/content/gdrive')

[ ] root_path = '/content/gdrive/My Drive/Colab Notebooks/Deep_Learning/Assignment/Dataset/'

#Data is assumed to be uploaded in Google Colab's working directory
# root_path=""
with h5py.File(root_path + 'train_128.h5', 'r') as H:
    raw_data = np.copy(H['data'])
with h5py.File(root_path + 'train_label.h5', 'r') as H:
    raw_label = np.copy(H['label'])
with h5py.File(root_path + 'test_128.h5', 'r') as H:
    raw_test_data = np.copy(H['data'])
```

3. The user is required to mount his/her google drive by running the cell above root\_path declaration in the above image. Upon running that cell, Colab will ask to authenticate drive access by providing a key. The user can get this key by following the link provided by Colab and signing in to their corresponding Google account (same account on which the datasets are uploaded in the drive).



4. Run all the cell blocks. The training section (section 4) of the neural network can take from 10 minutes to 150 minutes depending on the value of hyper parameters. The default values (chosen by us) are described in the report. The hyper parameters can be tuned in Section 3.4 of the report. (Note: Image shows section 4.4 but the final version for parameter tuning is in section 3.4)

#### 4.4 INITIALISING HYPERPARAMETERS FOR TUNING

```
# Define architecture
primary_activation = 'relu'          # 'leaky_relu', 'relu',
HL_neurons = [1024, 64]
|

# Define algorithm hyperparameters
#All these are tuned for best results

lrate = 0.04                        # learning rate
size_of_batch = 800                 # batch size
percent_dropout = 0.25              # percentage of neurons to turn
momentum_of_gamma = 0.7             # momentum parameter
batch_norm = False                  # Toggle Batch normalisation
lambada = 0                         # Weight decay L2 regularisation p

# Define stopping and reporting conditions

planned_epochs = 1000               # maximum planned number o
interval_reported = 10              # frequency of reporting t
convergence_threshold = 0.1         # convergence threshold pe

[ ] # Determine neural network architecture
no_of_features = raw_data.shape[1]
no_of_classes = onehot_raw_label.shape[1]

# Assign neurons and activations
final_layer_activation = 'softmax'
def_loss_function = 'cross-entropy' # loss function to be
```

5. Sections 5.1 and 5.2 give the accuracy's achieved on the given labelled dataset split into training and testing set.
6. Running section 6 will save the "Predicted\_labels.h5" file required as a submission for the assignment. This file has the predicted labels for the testing dataset. The file will be saved to the same directory on google drive where the dataset was uploaded. Right click on the file and click "Download" to download the file to your system.