American University of Sharjah

ELE494-09
Deep Networks in Machine Learning

Homework 2

Nasir Mohammad Khalid
65082
March 23, 2019

Submitted To: *Dr. Usman Tariq*

Contents

List of Figures

Listings

1 Task 1: Backpropagation

1.1 Q1.1

The goal of this part to develop a theoretical understanding of how do you get the expressions with backpropagation algorithm. Suppose you have a three layer fully connected neural network (input layer, hidden layer and output layer). Following are some further "specifications":

- The input feature vectors are d dimensional (you can think of these as MNIST images, attened as vectors)
- There are N feature vectors in you training dataset
- There are H hidden nodes and K output nodes (for K mutually exclusive classes).
- This neural network is to be trained for classification under cross-entropy loss function

Derive the equations for batch gradient updates for the input-to-hidden unit weights and hidden-to-output unit weights. Is it wise to use a learning rate parameter? Why?

1.2 Background to Answer

Based on the specifications given we can assume the architecture of the neural network to be as follows:

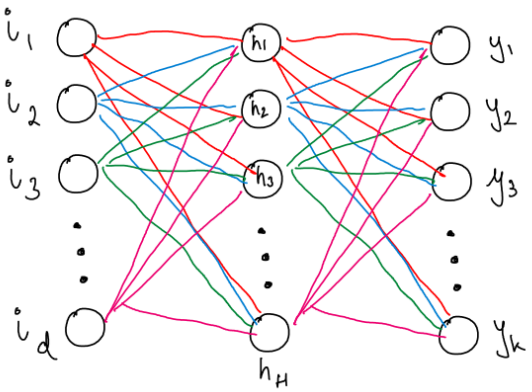

Figure 1: Representation of the neural network

The output of the node 'h' of hidden layer will be calculated using the sigmoid activation function and is given as follows:

$$Z_h = \sum_{n=1}^{d} i_n * w_{nh} + b_h \tag{1}$$

$$O_h = \frac{1}{1 + e^{Z_h}} \tag{2}$$

Z_h = Input to the sigmoid function of node 'h'
O_h = Output of the sigmoid function of the node 'h'
b_h = Bias of the input to node h
w_{nh} = Weight between input node n and the hidden node h

The output of the node 'o' of the final layer will be calculated using the softmax function and it will be given as follows:

$$Z_o = \sum_{n=1}^{h} O_n * w_{no} + b_o \tag{3}$$

$$Y_o = \frac{e^{Z_o}}{\sum_{j=1}^{k} e^{Z_j}} \tag{4}$$

Z_o = Input to the softmax function of node 'o'
Y_o = Output of the 'o'th output layer
b_o = Bias of the Output node 'o'
w_{no} = Weight between the 'n'th hidden node to the 'o'th output node

The output layer will give values between 0 and 1 only. The error function is the cross-entropy function. Since the output consists of K mutually exclusive classes then we can take the label for the feature vectors to be of a matrix of length K and it contains a 1 at the appropriate label. Then the cross entropy loss function is given as:

$$E = -\sum_{i=1}^{k} L_i * log(Y_i) \tag{5}$$

E = Error value from the cross-entropy function
L_k = Value of the label at the 'k'th index
Y_k = Output of output node 'k'

Now we want to get the derivative of the error with respect to the first set of weights between the hidden layer and output layer:

$$\frac{\partial E}{\partial w_{hk}} = \frac{\partial E}{\partial Z_k} * \frac{\partial Z_k}{\partial w_{hk}} \tag{6}$$

$\frac{\partial E}{\partial w_{hk}} = \partial$ of Error function w.r.t weights between hidden and output layer
$\frac{\partial E}{\partial Z_k} = \partial$ of the Error function with respect to the input to the softmax
$\frac{\partial Z_k}{\partial w_{hk}} = \partial$ of input to softmax w.r.t weights between hidden and output layer

The derivative of the softmax function Y_o with respect to the input of the softmax function Z_k is obtained through the quotient rule and is given by:

$$\frac{\partial Y_o}{\partial Z_k} = Y_k * (1 - Y_j) = Y_k * (1 - Y_k) \quad when \quad j = k \tag{7}$$

$$\frac{\partial Y_o}{\partial Z_k} = -Y_j * Y_k \quad when \quad j \neq k \tag{8}$$

The derivative of the error with respect to the input to the softmax can will consist of two parts. The first is when $i = k$ (in summation) and the second is when they are not the same:

$$\frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{k} L_i * \frac{\partial log(Y_i)}{\partial Y_i} * \frac{\partial Y_i}{\partial Z_k} \tag{9}$$

$$\frac{\partial E}{\partial Z_k} = -L_k * \frac{\partial log(Y_k)}{\partial Y_k} * \frac{\partial Y_k}{\partial Z_k} = -L_k * (1 - Y_j) \quad when \quad i = k \tag{10}$$

$$\frac{\partial E}{\partial Z_k} = -\sum_{i \neq k} L_i * \frac{\partial log(Y_i)}{\partial Y_i} * \frac{\partial Y_i}{\partial Z_k} = \sum_{i \neq k} L_i * Y_j \quad when \quad i \neq k \tag{11}$$

We take the summation of both cases to get the final derivative of error with respect to the input to the softmax function:

$$\frac{\partial E}{\partial Z_k} = -L_k * (1 - Y_j) + \sum_{i \neq k} L_i * Y_j \tag{12}$$

$$\frac{\partial E}{\partial Z_k} = Y_j * (L_k + \sum_{i \neq k} L_i) - L_k \tag{13}$$

but $(L_k + \sum_{i \neq k} L_i)$ is equal to 1 as it is all the intended outputs added up. Due to the fact that the outputs are always probabilities their sum will give 1 and therefore we get the final expression:

$$\frac{\partial E}{\partial Z_k} = Y_j - L_k = Y_k - L_k \tag{14}$$

$\frac{\partial E}{\partial Z_k} = \partial$ of Error function w.r.t input to the softmax function

Y_k = It is the final output of the output neuron

L_k = It is the intended output of the output neuron

Similarly we the derivative of the input to the softmax with respect to the weights. Here O_h is the output of the hidden layer:

$$\frac{\partial Z_k}{\partial w_{hk}} = O_h \tag{15}$$

1.3 A1: Gradient Update for hidden-to-output weights

The final answer is given by:

$$\frac{\partial E}{\partial w_{hk}} = \frac{\partial E}{\partial Z_k} * \frac{\partial Z_k}{\partial w_{hk}} = (Y_k - L_k) * O_h \tag{16}$$

$\frac{\partial E}{\partial w_{hk}} = \partial$ of Error function w.r.t weights between hidden-output

Y_k = It is the final output of the output neuron

L_k = It is the intended output of the output neuron

O_h = It is the final output of the hidden neuron

1.4 A2: Gradient Update for input-to-hidden weights

Here we further extend the partial derivatives until we get the change in error with respect to the input-hidden weights. Since the output of one hidden layer neuron is sent all output neurons we take a summation over all output neurons.

$$\frac{\partial E}{\partial w_{ih}} = \sum_{b=1}^{k} \frac{\partial E}{\partial Z_b} * \frac{\partial Z_b}{\partial O_h} * \frac{\partial O_h}{\partial Z_h} * \frac{\partial Z_h}{\partial w_{ih}} \tag{17}$$

$\frac{\partial E}{\partial w_{ih}} = \partial$ of Error function w.r.t weights between input-hidden

$\frac{\partial E}{\partial Z_b} = \partial$ of Error function w.r.t input to softmax for neuron b

$\frac{\partial Z_b}{\partial O_h} = \partial$ of Input to softmax for neuron b w.r.t output of hidden

$\frac{\partial O_h}{\partial Z_h} = \partial$ of Output of hidden w.r.t input to sigmoid

$\frac{\partial Z_h}{\partial w_{ih}} = \partial$ of Input to sigmoid w.r.t weight between input and hidden

We already obtained the first term $\frac{\partial E}{\partial Z_b}$ in section 1.4 and the rest are derived below. The first one (18) where w_{hb} is the weight between hidden neuron h and output neuron b

$$\frac{\partial Z_b}{\partial O_h} = w_{hb} \tag{18}$$

(19) is derivative of the sigmoid function

$$\frac{\partial O_h}{\partial Z_h} = O_h * (1 - O_h) \tag{19}$$

Here i_n is the input to the network

$$\frac{\partial Z_h}{\partial w_{ih}} = i_n \tag{20}$$

Combining all of these together gives the final equation for the change in error with respect to the weights between input-hidden layer.

$$\frac{\partial E}{\partial w_{ih}} = \sum_{b=1}^{k} (Y_b - L_b) * w_{hb} * O_h * (1 - O_h) * i_n \tag{21}$$

$\frac{\partial E}{\partial w_{ih}}$ = ∂ of Error function w.r.t weights between input-hidden

Y_b = Output of the final neuron 'b'

L_b = Intended output of final neuron 'b'

w_{hb} = Weights between hidden neuron 'h' and output neuron 'b'

O_h = Output of hidden neuron

i_n = Input of first neuron

1.5 A3: Learning Parameter

In this situation using a learning parameter would be useful because for the backpropagation in equation (21) if the input i_n is a large value then the weights would not update by a small value. Since we are using the MNIST database the values could range from 0-255

2 Task 2: Implementation

2.1 Q2

Import the Fashion mnist dataset then divide the training images further, into 40,000 training and 20,000 validation images. Make sure you do this at random. So now you will have 40,000 training images, 20,000 validation images and 10,000 testing images.

- Read about the fashion mnist dataset and report the highlights (e.g. number of images, number of classes etc) along with some example images.
- What will be reasonable loss function to use for classification if the last layer activation functions are soft-max functions? Explain your rationale.
- Train four neural networks each with one hidden layer; one with 16 nodes, one with 64 nodes, one with 128 node and one with 512 nodes. Train each one of them for 15 epochs.
  - Which one of them gives you the best performance on the validation data?

- Does increasing epoch to 30 improve the validation performance?
- Experiment with using sigmoid and relu as activation functions for the hidden layer. Plot the validation loss or validation accuracy for both kinds of activation functions for all the four network models against the number of epochs (30). This will give you 8 plots. Comment on the plots and comparison across the choice of activation functions
- Pick the model parameters that give you the best performance, then combine your validation data with training data, retrain your network with optimum parameters and report the classification accuracy as well as the confusion matrix on the test data

2.2 A1: Fashion-MNIST Details

Fashion-MNIST is a dataset consisting of different articles of clothing. It has a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Below is an image of the entire Fasion MNIST dataset

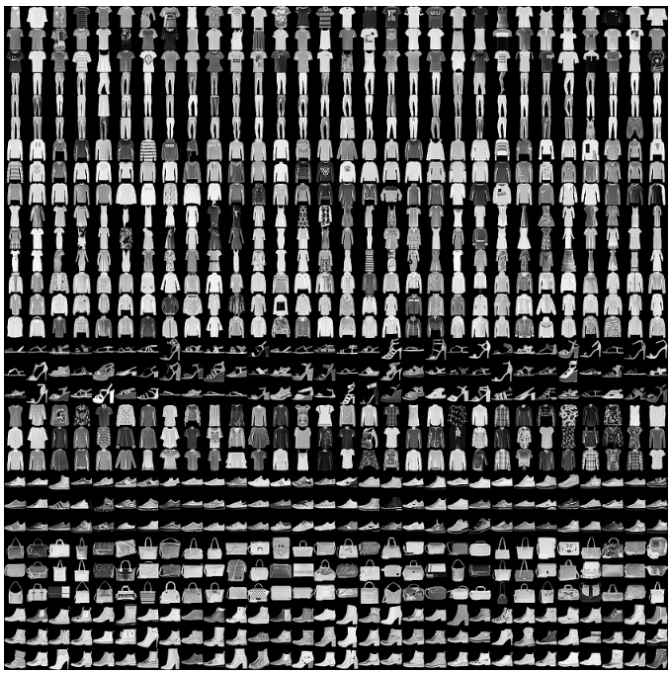

Figure 2: Fashion MNIST Dataset Images

2.3 A2: Reasonable Loss Function

If the last layer has a softmax output, this means that each neuron would output a probability between 0 and 1. The sum of the last layer activations would be 1. The reasonable loss function would be the cross-entropy loss because it has a large gradient when the output value is far from 1 and as the output becomes 1 the gradient changes slowly therefore preventing overshooting the minima. The cross entropy works best for probabilities and helps speed up learning.

Furthermore, the softmax ensures that no two classes are returned simultaneously since it returns the probability for each class.

2.4 A3: Training 4 Neural Networks

The code used to train the 4 networks is included in the python notebook. Outlined here are the results of the 4 training cases.

2.4.1 16 nodes and 15 epochs

This was the best performing network. The final training data accuracy was 0.8648

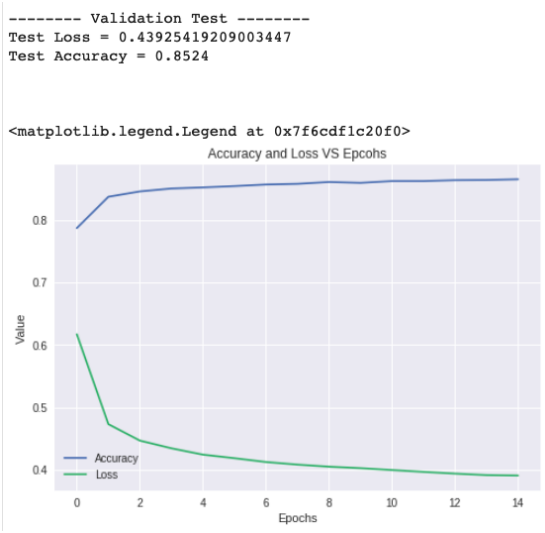

Figure 3: Trained with 16 hidden nodes and 15 epochs

2.4.2 64 nodes and 15 epochs

The final training data accuracy was 0.8620

```
-------- Validation Test --------
Test Loss = 0.44952310588359834
Test Accuracy = 0.84795


<matplotlib.legend.Legend at 0x7f6cdeea3128>
```

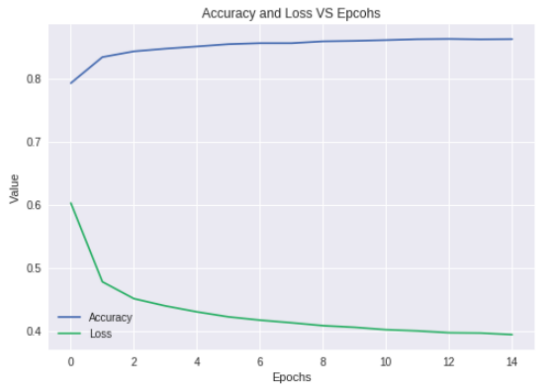

Figure 4: Trained with 64 hidden nodes and 15 epochs

2.4.3 128 nodes and 15 epochs

The final training data accuracy was 0.8603

```
-------- Validation Test --------
Test Loss = 0.4585471803426743
Test Accuracy = 0.8432


<matplotlib.legend.Legend at 0x7f6cde8aaa58>
```

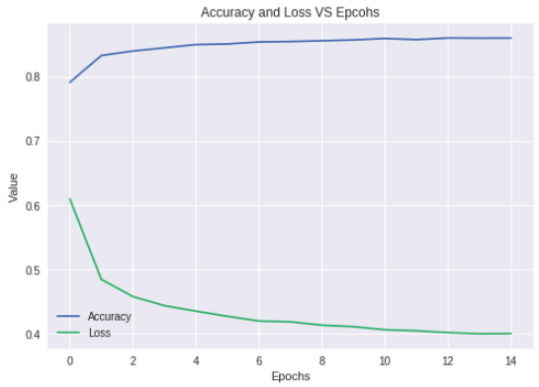

Figure 5: Trained with 128 hidden nodes and 15 epochs

2.4.4 512 nodes and 15 epochs

The final training data accuracy was 0.8567

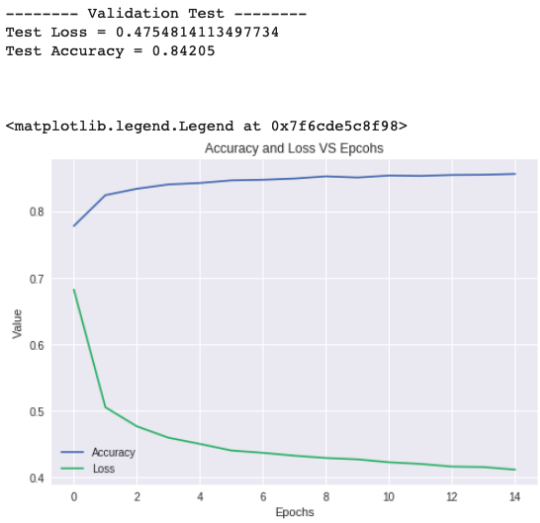

Figure 6: Trained with 512 hidden nodes and 15 epochs

2.5 A4: Increasing the epochs to 30

The jupyter notebook shows the code used and the changes from moving to 30 epochs. Overall, if there is a large number of neurons then the validation performance increases at 30 epochs. For a small number of neurons, increasing the epochs made it worse.

2.5.1 For 16 Neurons

The test accuracy decreased by 0.00225. The test loss increased by 0.004325. For 16 neurons it did not really improve the validation performance.

2.5.2 For 64 Neurons

The test accuracy decreased by 0.0086. The test loss increased by 0.033835. For 64 neurons it did not really improve the validation performance.

2.5.3 For 128 Neurons

The test accuracy increased by 0.00475. The test loss decreased by 0.006488. For 32 neurons it improved the validation performance.

2.5.4 For 512 Neurons

The test accuracy increased by 0.0073. The test loss decreased by 0.025331. For 512 neurons it improved the validation performance.

2.6 A5: Hidden Layer Activation Function

The code for different neurons and activation functions is included in the jupyter notebook. Below are the results for different neurons and activations each with 30 epochs

2.6.1 For 16 Neurons

Here the sigmoid activation performed better even though it started with a very high loss function value. Below is the RELU activation

```
-------- Validation Test --------
Test Loss = 0.43207713651657104
Test Accuracy = 0.86345

<matplotlib.legend.Legend at 0x7f6cdd6b8ac8>
```

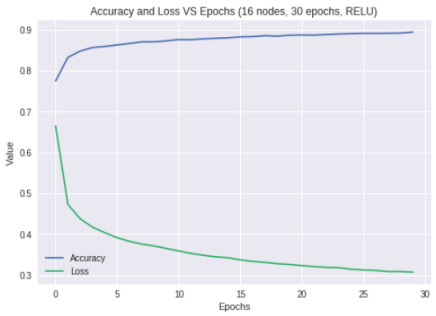

Figure 7: 16 Neurons and RELU

Below is the Sigmoid activation

```
-------- Validation Test --------
Test Loss = 0.3887210489630699
Test Accuracy = 0.8652

<matplotlib.legend.Legend at 0x7f6cdd0bf0b8>
```

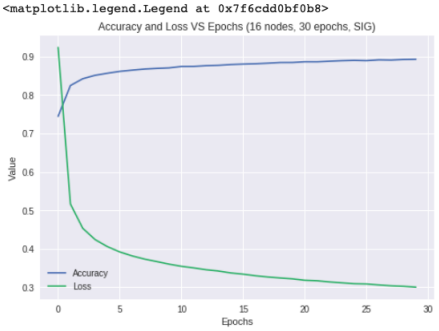

Figure 8: 16 Neurons and Sigmoid

2.6.2 For 64 Neurons

Here the sigmoid activation performed better. Below is the RELU activation

```
-------- Validation Test --------
Test Loss = 0.4500553294956684
Test Accuracy = 0.87095

<matplotlib.legend.Legend at 0x7f6cdcd86b70>
```

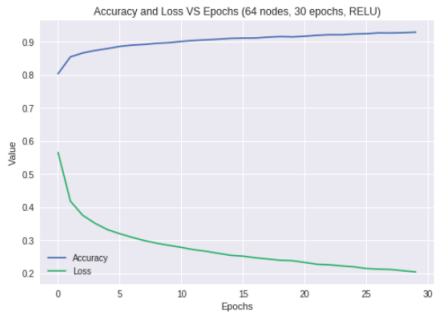

Figure 9: 64 Neurons and RELU

Below is the Sigmoid activation

```
-------- Validation Test --------
Test Loss = 0.3488295847892761
Test Accuracy = 0.88455

<matplotlib.legend.Legend at 0x7f6cdca8db70>
```

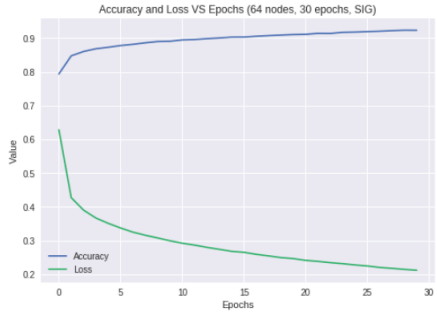

Figure 10: 64 Neurons and Sigmoid

2.6.3 For 128 Neurons

Here the sigmoid activation performed better. Below is the RELU activation

```
-------- Validation Test --------
Test Loss = 0.4898193704277277
Test Accuracy = 0.88365

<matplotlib.legend.Legend at 0x7f6cdc474f28>
```

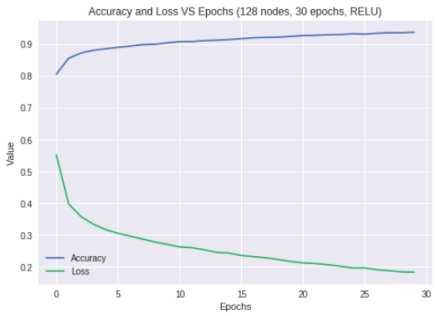

Figure 11: 128 Neurons and RELU

Below is the Sigmoid activation

```
-------- Validation Test --------
Test Loss = 0.35383276596963403
Test Accuracy = 0.88975

<matplotlib.legend.Legend at 0x7f6cdc123dd8>
```

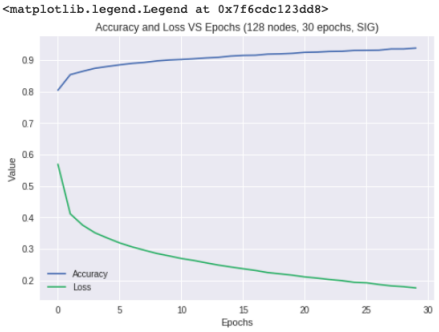

Figure 12: 128 Neurons and Sigmoid

2.6.4 For 512 Neurons

Here the sigmoid activation performed better. Below is the RELU activation

```
-------- Validation Test --------
Test Loss = 0.5486640579611063
Test Accuracy = 0.8884

<matplotlib.legend.Legend at 0x7f6cdbe856a0>
```

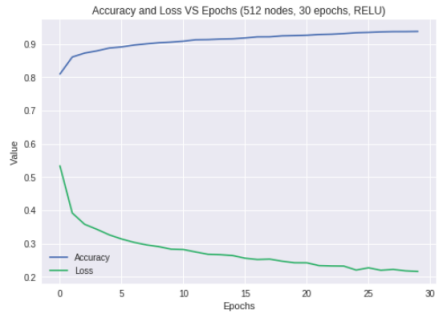

Figure 13: 512 Neurons and RELU

Below is the Sigmoid activation

```
-------- Validation Test --------
Test Loss = 0.39151631913036106
Test Accuracy = 0.89315

<matplotlib.legend.Legend at 0x7f6cdbb297b8>
```

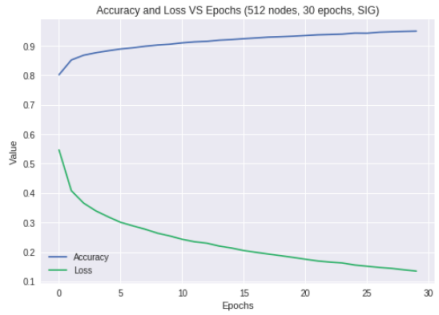

Figure 14: 512 Neurons and Sigmoid

Overall the sigmoid function performed best as the activation function of the hidden layer. It was much faster at reaching the lowest error value.

2.7 A6: Retraining Network

Here we retrain the network with 512 nodes, 30 epochs and the sigmoid activation for hidden layer. The network was trained for 60000 images and the tested on the 20000 test data. The code is in the jupyter notebooks final cell.

```
-------- Validation Test --------
Test Loss = 0.39151631913036106
Test Accuracy = 0.89315

<matplotlib.legend.Legend at 0x7f6ce74b7cf8>
```

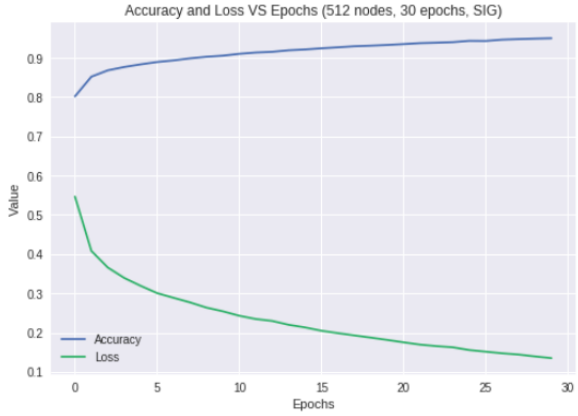

Figure 15: Network trained on all training data and then tested on testing data

This network had the best performance on all previously trained networks and the confusion matrix for it is shown below:

```
-------- Confusion Matrix --------
[[900   2  17  18  11   2  43   0   7   0]
 [  0 982   3   8   6   0   0   0   1   0]
 [ 20   1 798   8 143   1  27   0   2   0]
 [ 40  10   9 882  43   0  10   0   6   0]
 [  0   0  55  22 905   0  16   0   2   0]
 [  0   0   0   1   0 958   0  26   1  14]
 [200   2  96  25 182   0 486   0   8   1]
 [  0   0   0   0   0   8   0 975   0  17]
 [  4   1   5   4   8   3   2   3 970   0]
 [  1   0   0   0   0   8   1  39   0 951]]
```

Figure 16: Confusion Matrix

3 Task 3: Theory Questions

3.1 Q1

What is generalization in the context of machine learning. Is it better for the machine learning algorithms to generalize?

3.2 A1

Generalization is the ability for a network to figure out/adapt to the new input data. It determines how well a network will perform on data it has never seen before. Generalization is better because the better the network is at it, the better it will be with brand new data. If a system does not generalize then it will be unable to adapt.

3.3 Q2

What is overfitting and under-fitting, how is it related to generalization?

3.4 A2

Overfitting is when the network essentially 'memorizes' the data and it manages to lower it's cost function heavily however this sort of behavior means it is unable to generalize and it will do well on training data but not on real life data. Under-fitting is when the network is unsuccesfully trained and it's cost function is not as low as it can get, this means that it performs poorly on training data as well as test data. It's weights need to be adjusted further.

3.5 Q3

What is capacity of a neural network? Is it related to generalization, if so how?

3.6 A3

Capacity of a network is how much the network is able to learn. The more it is able to learn the more complex it's model will become. In relation to generalization, a network that is extremely complex can tend to overfit on a very simple problem, however, on a complex problem it will be good at picking up the subtle nuances of the input data. So a network with low capacity can underfit while a network with high capacity can often overfit. It is vital to find a middle ground

3.7 Q4

What is the relationship between number of epochs and overfitting?

3.8 A4

If the number of epochs (iterations) exceeds a certain value the model can start to overfit the data. Initially the increasing number of epochs can improve accuracy but after a limit the network starts overfitting

3.9 Q5

What is weight regularization? How does it improve generalizations? Write a code snippet that can be included in Task 2 for adding weight regularization.

3.10 A5

When the weights of a network tend to increase by a large amount we reduce them through normalization techniques. This is called weight regularization and it is used to reduce the chances of overfitting in a network. This is because in a network with overfitting there are usually large weights so by making them smaller and reducing them through normalization we tend to prevent overfitting. This in turn helps the network generalize better. Below is some code that shows how we would implement regularization on the Fashion MNIST code

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import regularizers

model = Sequential()

model.add(Dense(16,
                input_dim=784,
                kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))

model.add(Dense(10,
                activation='softmax',
                kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Listing 1: Network with Weight Regularization

3.11 Q6

What is drop out? How does it improve generalizations? Write a code snippet that can be included in Task 2 for adding drop out.

3.12 A6

Dropout is when we randomly turn some of the weights in the network to 0, thereby forcing the network to learn based on other parameters. This reduces dependency on a single weight and it also helps prevent overfitting.

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout

model = Sequential()

model.add(Dropout(0.2))

model.add(Dense(16, input_dim=784,))

model.add(Dropout(0.2))

model.add(Dense(10, activation='softmax',))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Listing 2: Network with Dropout

References

[1] Usman Tariq Notes and Slides from ELE494.

[2] Eli Bendersky's Website. *The Softmax Function and its derivative*
`https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/`

[3] Keras Documentation,
`https://keras.io/`