AMATH 482 - Winter Quarter
Homework 5: MNIST Handwriting Recognition

Vicky Norman

March 11, 2018

Abstract

This project explores the implementation of neural nets to classify data from the MNIST handwritten dataset. Initially a one layer network is created and evaluated. Next a multi-layer network is implemented using MATLAB's neural net package functions, several different transfer functions are compared and the results are presented and analysed. The project concludes that the Elliot sigmoid transfer function produces the best results.

1 Introduction and Overview

The aim of this project is to produce a neural net that can classify images of handwritten digits from 0-9. We will discuss the accuracy of the method with several different training functions, as well as the advantages of one vs two layers of perception. The algorithm has been executed using MATLAB's neural network functions.

The data set consists of 25,723 images with 784 pixels each, the data is imported from a spreadsheet that contains the numeric value of each pixel in the image as well as a label which indicated which digit from 0-9 the image represents. Since we are dealing with such a large data set the neural net method should work well and is advantageous over other classification methods such as k-nearest neighbour or Naive Bayes methods.

This kind of classification is very important for many practical applications, handwriting recognition is used by the postal service to ensure handwritten cards reach the correct destination as well as for processing cheques. Neural coding and recognition is still a pioneering area of research, since as humans we have an uncanny ability to recognise practically any object even by seeing an example just once. This is something that experts are still trying to replicate with computer programs.

2 Theoretical Background

Deep neural nets consist of layers of perceptrons, perceptrons are used for supervised learning, they are binary classifiers which take an input and return an output based on the Heaviside step function. Neural nets vary in complexity depending on how many layers they have. Simple neural nets have linear relationships through the layers but in more complex networks the relationships are represented by functions. The following equations represent what is happening in a neural net that has a linear relationship. The input is represented by the matrix $\mathbf{X}$, the output after each layer is then represented by the matrices $\mathbf{X}^{(1)}$ etc.

$$\begin{aligned}\mathbf{X}^{(1)} &= \mathbf{A}_1\mathbf{X}\\ \mathbf{X}^{(2)} &= \mathbf{A}_2\mathbf{X}^{(1)}\\ &\vdots\\ \mathbf{Y} &= \mathbf{A}_n\mathbf{X}^{(n-1)}\end{aligned}$$

This network can then be represented as $\mathbf{Y} = \mathbf{A}_n\mathbf{A}_{n-1}....\mathbf{A}_2\mathbf{A}_1\mathbf{X}$ and we can generalize the matrices for each layer to get $\mathbf{Y} = \mathbf{A}\mathbf{X}$. Sine this has now become a generalised $\mathbf{A}\mathbf{x} = \mathbf{b}$ problem it can be easily solved in MATLAB

in many ways.

For a more complex neural network the layers will be non-linear and can be represented in the following way $\mathbf{Y} = f_n(\mathbf{A}_n, ...f_2(\mathbf{A}_2, f_1(\mathbf{A}_2, \mathbf{X})))$, solving this type of problem is a much more difficult task as the complexity is greatly increased. The Backpropagation method can be implemented to optimize this kind of network, represented by $y = g(z, b) = g(f(x, a), b)$, to minimize the error $E = \frac{1}{2}(y_0 - y)^2$. With correct output y_0 and the approximation of the output y the chain rule can be used to minimize the error as follows:

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{dy}{dz}\frac{dy}{dz} = 0.$$

The backprop method results in an iterative, gradient descent update rule:

$$a_{k+1} = a_k + \tau \frac{\partial E}{\partial a_k}$$
$$b_{k+1} = b_k + \tau \frac{\partial E}{\partial b_k}$$

Where τ is the so-called constant learning rate of the algorithm and $\frac{\partial E}{\partial a}$ and $\frac{\partial E}{\partial b}$ can now be explicitly calculated. The iteration algorithm is executed to convergence. As with all iterative optimization procedures, a good initial guess is critical for achieving a good solution in a reasonable amount of computational time.

3 Algorithm Implementation and Development

The implementation of the learning algorithms is done using MATLAB since it has built in functions that can be used for neural nets. For the simple one layer, linear network the stages of implementing the algorithm are as follows:

- Load data from spreadsheet,
- Separate labels from training and test data,
- Convert labels vector where each label is represented by a 10 digit vector of zeros with a 1 in the corresponding index representing the digit in the image.
- Randomly assign most of the data to the training set, leaving 1000 entries for test data.
- Compute matrix $\mathbf{A}$ as $\mathbf{A} = \mathbf{b}\mathbf{x}_{train}^{-1}$,
- Classify the test data by computing $\mathbf{A} * \mathbf{x}_{test}$.

In order to check the validity of the results the algorithm is cross validated several times by choosing a random subset of the data to be the training data and the rest to be the test data. Doing this reduces the chance of over fitting and ensures that the accuracy score produced is consistent.

For the neural net of more than one layer the first 4 steps of data processing are the same as above. Then the inbuilt MATLAB functions are used to train the network. The first layer of the network was specified by a transfer function, each additional layer was hidden and the system operated with a total of 10 layers. The algorithm was run with each transfer function available, a list of the functions used and the accuracy score for each one is shown in table 1.

The output of both networks gives a matrix where each column represents the classification of each image. The maximum entry in each column is in the index corresponding to the image that the system has classified the image to be. The accuracy for both methods is therefore calculated by comparing the index of the maximum entry to the index of the 1 in the label for that particular image. If the index is the same then that is counted as a correct result giving a final accuracy score out of the number of images classified.

Although the accuracy of classification is very important there are also other factors of the network that should be considered. MATLAB measures the performance of the neural networks through cross entropy, this measures

Table 1: Transfer function and corresponding accuracy rate of neural net.

MATLAB command	Function name	% accuracy
compet	Competitive transfer function	35
elliotsig	Elliot sigmoid transfer function	90
hardlim	Positive hard limit transfer function	88
hardlims	Symmetric hard limit transfer function	88
logsig	Logarithmic sigmoid transfer function	90
netinv	Inverse transfer function	13
poslin	Positive linear transfer function	90
purelin	Linear transfer function	90
radbas	Radial basis transfer function	25
radbasn	Radial basis normalized transfer function	33
satlin	Positive saturating linear transfer function	69
satlins	Symmetric saturating linear transfer function	71
softmax	Soft max transfer function	47
tansig	Symmetric sigmoid transfer function	88
tribas	Triangular basis transfer function	11

how close the outputs are to their target values, the best net performance wise is the one with the smallest cross entropy values. The formula for this is as follows:

$$\text{Cross entropy} = -\sum_i y_0^{(i)} log(y^{(i)}). \tag{1}$$

4 Computational Results

4.1 Single Layer System

After running the single layer neural net several times with cross validation the average accuracy was 85%. For a single layer, linear system this level of accuracy is reasonably good, for a system with multiple, more complex layers we would expect to reach a higher level of accuracy.

4.2 Multi-layer System

From the results in table 1 was can see that the best accuracy was achieved by the Elliot sigmoid, Logarithmic sigmoid, Positive linear and Linear transfer functions. All of these functions achieved accuracy of around 90%. In order to try and determine which of these functions performed best we will examine the cross-entropy, error histograms and time these functions took to execute.

Figures 2, 5, 8 and 11 show the error histograms for all four transfer functions, we can see that the graphs are very similar with the modal error being around -0.05, it is expected that these graphs would be very similar since the accuracy calculated for all the functions was the same.

The lowest (best) value achieved for cross entropy was from the linear transfer functions at 0.031124. This value is not very far from the cross entropy values of the other three functions, however when comparing figures 3, 6, 9 and 12 we can see that the cross entropy of the Elliot sigmoid and logarithmic sigmoid has a smaller range than that of the other two functions, specifically they both start at a lower cross entropy and achieve their lowest cross entropy much faster. The system also measures the number of Epochs which is the number of times that the weights are updated in the backpropagation algorithm. Critically the Elliot sigmoid transfer function reaches it's minimal cross entropy at just 16 Epochs and the positive linear function takes 436 to reach it's minimal cross-entropy. This value is important because it gives us an idea of how much computational power is being used.

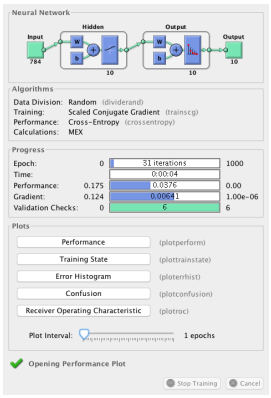

Figure 4: Summary of performance diagnostics for logarithmic sigmoid transfer function.

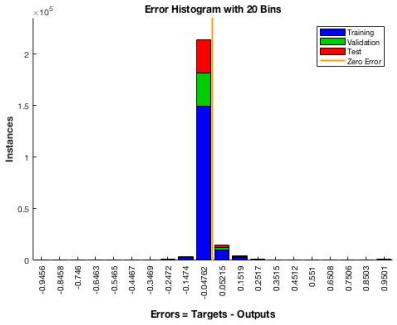

Figure 5: Error histogram for logarithmic sigmoid transfer function.

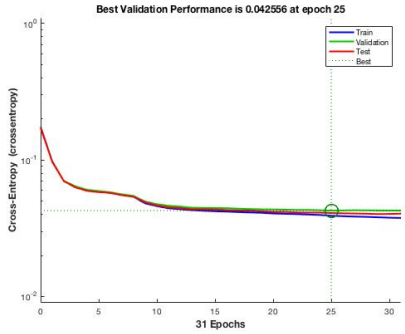

Figure 6: Cross entropy graph for logarithmic sigmoid transfer function.

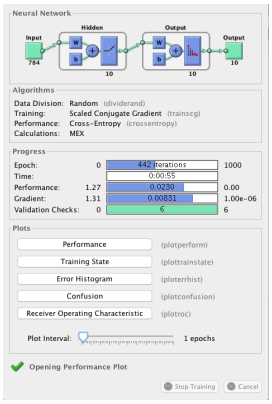

Figure 7: Summary of performance diagnostics for positive linear transfer function.

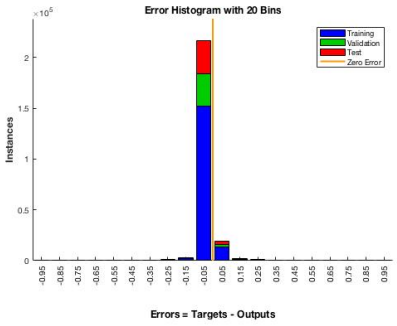

Figure 8: Error histogram for positive linear transfer function.

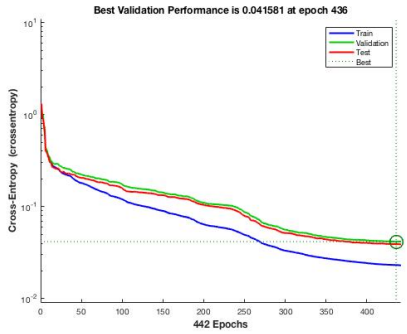

Figure 9: Cross entropy graph for positive linear transfer function.

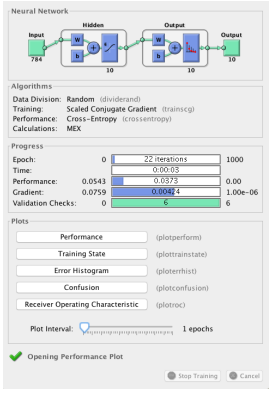

Figure 1: Summary of performance diagnostics for Elliot sigmoid transfer function.

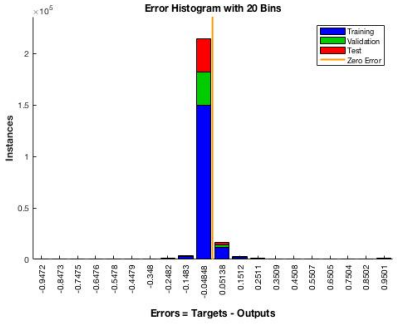

Figure 2: Error histogram for Elliot sigmoid transfer function.

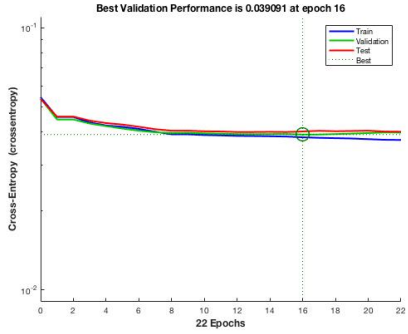

Figure 3: Cross entropy graph for Elliot sigmoid transfer function.

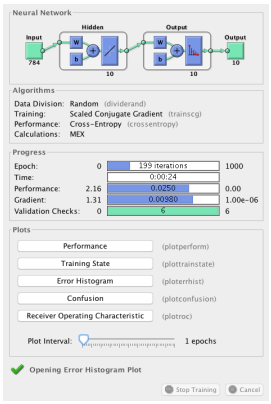

Figure 10: Summary of performance diagnostics for linear transfer function.

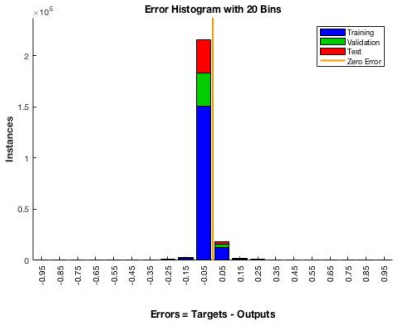

Figure 11: Error histogram for linear transfer function.

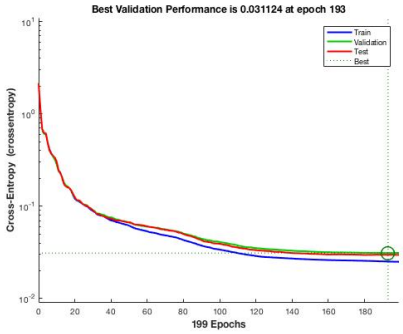

Figure 12: Cross entropy graph for linear transfer function.

Finally if we compare the time taken to compute the network for each of the transfer functions we find that the functions' execution time ranges from 3-55 seconds. The Elliot sigmoid function is the fastest and the positive linear function is the slowest, this correlates to what we found about the Epochs. Overall we can conclude that the transfer function that gives us the best overall performance including accuracy, cross entropy and time is the Elliot sigmoid function.

5 Summary and Conclusions

The project has shown that when classifying data an improved accuracy can be achieved by upgrading a single layer simple perceptron algorithm to a multi-layer neural net. The comparison between the use of different transfer functions concluded that the Elliot sigmoid function was the most efficient and accurate.

Further development of this project could included adding additional layers to the system and possibly exploring the implementation of deep convolution neural nets which could improve the accuracy of the classification system.

References

[1] *Neural Networks and Deep Learning*, J. Nathan Kutz

[2] *https://www.mathworks.com/*

6 Appendix

A

- M = csvread(filename,R1,C1) reads data from the file starting at row offset R1 and column offset C1. For example, the offsets R1=0, C1=0 specify the first value in the file.[2]
- p = randperm(n) returns a row vector containing a random permutation of the integers from 1 to n inclusive.[2]
- B = pinv(A) returns the Moore-Penrose Pseudoinverse of matrix A. [2]
- patternnet(hiddenSizes,trainFcn,performFcn). Pattern recognition networks are feedforward networks that can be trained to classify inputs according to target classes. The target data for pattern recognition networks should consist of vectors of all zero values except for a 1 in element i, where i is the class they are to represent.[2]
- net.layersi.transferFcn This function defines which of the transfer functions is used to calculate the ith layer's output, given the layer's net input, during simulation and training.[2]
- net = train(net,X,T), output net = newly trained network, input net = network, X = network inputs, T = network targets. train trains a network net according to net.trainFcn and net.trainParam.[2]
- perform(net,t,y), net = neural network, t = target data, y = output data.
- vec2ind(vec) allow indices to be represented either by themselves or as vectors containing a 1 in the row of the index they represent.[2]

B

Appendix B MATLAB codes.
Single layer perceptron neural network code.

```
trainlabel = csvread('train.csv',1,0);
test = csvread('test.csv',1,0);

%%

[m, n] = size(trainlabel);

q = randperm(m);

labels = trainlabel(:,1);

matrix_labels = zeros(10,m);

for i = 1:m
    k = labels(i);
    col = zeros(10,1);
    col(k+1) = 1;
    matrix_labels(:,i)=col;
end

train_labels = matrix_labels(1:10,q(1:24723));
test_labels = matrix_labels(1:10,q(24724:end));

%%

train = trainlabel(q(1:24723),2:end);
test = trainlabel(q(24724:end),2:end);
```

```
A=train_labels*pinv(train)';
prediction=(A*test');

%%

correct = 0;

for i = 1:length(test_labels)
    if find(max(prediction(:,i))==prediction(:,i))==(find(test_labels(:,i)==1))
            correct = correct + 1;
    end
end

correct/1000
```

Multi-layer neural network code.

```
trainlabel = csvread('train.csv',1,0);
test = csvread('test.csv',1,0);

%%

[m, n] = size(trainlabel);

q = randperm(m);

labels = trainlabel(:,1);

matrix_labels = zeros(10,m);

for i = 1:m
    k = labels(i);
    col = zeros(10,1);
    col(k+1) = 1;
    matrix_labels(:,i)=col;
end

train_labels = matrix_labels(1:10,q(1:24723));
test_labels = matrix_labels(1:10,q(24724:end));

x = trainlabel(q(1:24723),2:end)';
x2 = trainlabel(q(24724:end),2:end)';
%%
net = patternnet(10, 'trainscg');

%%
% net.layers{1}.transferFcn = 'compet';
% net.layers{1}.transferFcn = 'elliotsig';
% net.layers{1}.transferFcn = 'hardlim';
% net.layers{1}.transferFcn = 'logsig';
% net.layers{1}.transferFcn = 'hardlims';
% net.layers{1}.transferFcn = 'netinv';
% net.layers{1}.transferFcn = 'poslin';
net.layers{1}.transferFcn = 'purelin';
% net.layers{1}.transferFcn = 'radbas';
% net.layers{1}.transferFcn = 'radbasn';
% net.layers{1}.transferFcn = 'satlin';
% net.layers{1}.transferFcn = 'satlins';
```

```
% net.layers{1}.transferFcn = 'softmax';
% net.layers{1}.transferFcn = 'tansig';
% net.layers{1}.transferFcn = 'tribas';

net = train(net,x,train_labels);

view(net)

y = net(x2);

perf = perform(net,train_labels,net(x));

classes = vec2ind(y);

%%
correct = 0;

for i = 1:length(test_labels)
    if (classes(i))==(find(test_labels(:,i)==1))
            correct = correct + 1;
    end
end

correct/1000
```