Back-Propogation and Softmax

cxb980 pab734 gxk636 wxw870 txd844 zxw780

1 Introduction

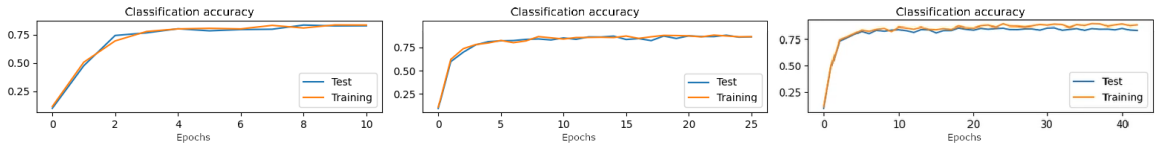
The input data was first normalised between the range [-0.5, 0.5]. Then, in turn, each hyper-parameter was explored individually with all others set to a default: batch size=50; epsilon=0.01; epochs=50; network shape=[748,20,20,20,10]; activation function=ReLU.

In each section, the effects of the parameter on the classification accuracy were explored and potential optimal values were established. Following this, the network was trained and tested with a full set of optimal hyper-parameters to achieve the best possible classification accuracy using the test set.

2 Methodology

2.1 Training Time

To measure the effect training time has on the neural network we varied the amount of epochs the network ran for making sure all other parameters are the default neural network. The default parameters are batch size = 50, epsilon = 0.01, and varied the number of epochs from 10 to 50.



The results show that the elbow point is very early on at 2 epochs and continues to rise until the 10th epoch. After that there are only slight differences seen to the accuracy, mostly fluctuating between 75 and 80 percent for the rest of the epochs. Furthermore, each epoch takes approximately the same amount of time to run so the amount of epochs and running time for the network are linearly related.

2.2 Minibatch Size

After some initial research, it was found that commonly chosen batch sizes range from 32 - 512 data points (`https://arxiv.org/abs/1609.04836v2`). Therefore, initial exploration-focused around the 1 - 100 range of batch sizes, incrementing in steps of 10. After analysing the resulting accuracies from this stage, it became clear that larger batch sizes would not improve the classification accuracy, so batch size steps were henceforth increased to 200, 500, and 1000.

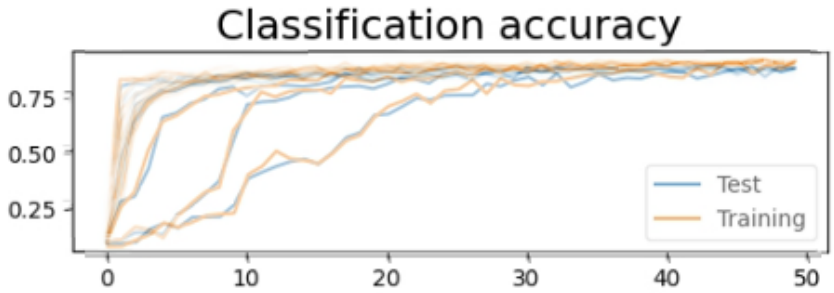


Figure 2: Graph showing convergence rates at each batch size overlaid. The sharpest convergence was achieved with stochastic GD (batch size = 1), and took longer with each increase in batch size in the range [10,20,30,40,50,60,70,80,90,100,200,500,1000].

From Figure 2 it is clear to see that as batch size increases, the algorithm takes longer to converge. Unfortunately, only the shape of each convergence is shown here, as the accuracy scale was not consistent throughout testing. Final classification accuracy is instead shown, along with training time for 50 epochs, in Figure 3.

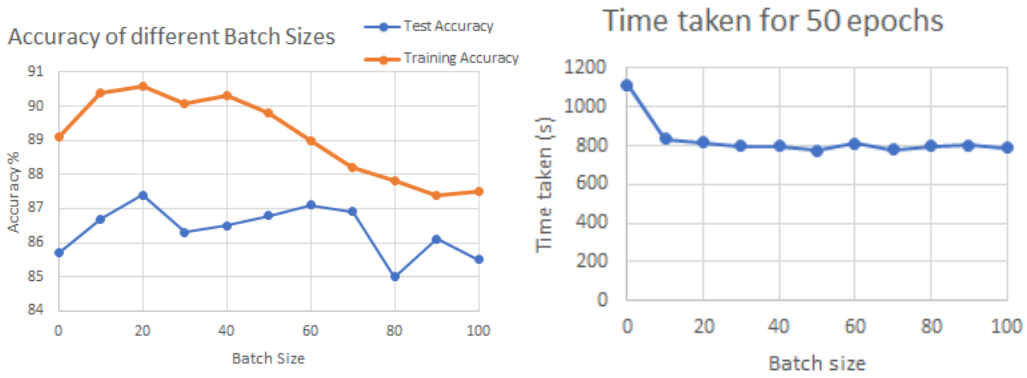


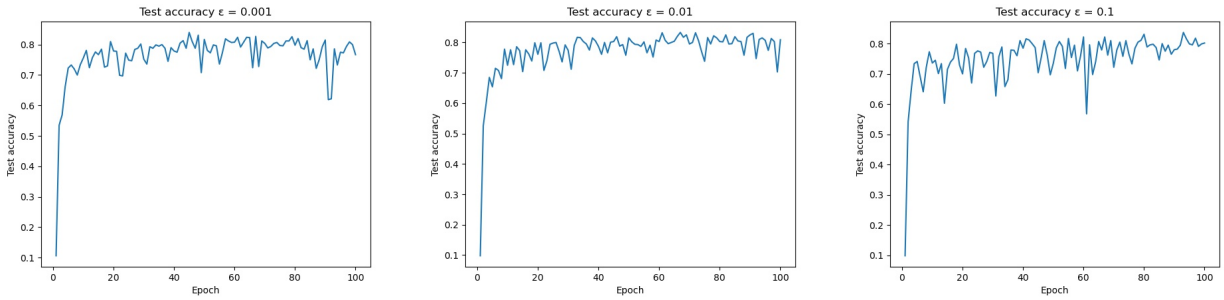
Figure 3: Graph (left) showing the average of the final 3 epochs' test and training accuracies for each batch size. Note that batch sizes of 200, 500, and 1000 gave [test, training] accuracies of [83.9%, 86.4%], [79.5%, 83.1%], and [77.3%, 78.7%] respectively. Graph (right) shows that using batch sizes over stochastic GD reduces training time by 282 seconds, while consecutive batch size increases have no effect.

This figure shows that the optimal batch size, with other parameters default, lies between 10 - 30. The maximum test accuracy reached was 87.4% with a batch size of 20. Interestingly, accuracy is reduced significantly at a batch size of 30, before climbing to reach almost optimal accuracy at 60. As batch size was increased from 200 to 1000, accuracy further declined.

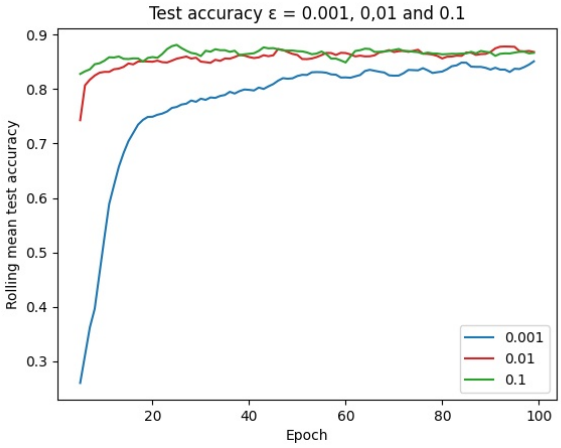
While time taken for 50 epochs remained consistent with each batch size including those up to 1000, it is worth noting that convergence was significantly faster at smaller batch sizes, and thus the network would not necessitate as many epochs to reach desirable results as larger batch sizes.

2.3 Learning Rate

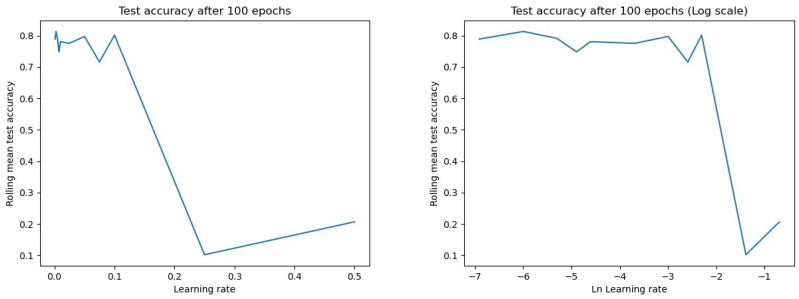
To measure the effect of varying the learning rate (epsilon) on the performance of the model, we selected 11 different epsilon values (0.001, 0.0025, 0.005, 0.0075, 0.01, 0.025, 0.05, 0.075, 0.1, 0.25 and 0.5) and tested these over 100 epochs. The test accuracy of 0.001, 0.01 and 0.1 can be seen in the figures below.



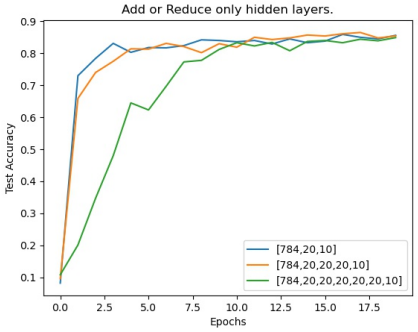
As expected, greater learning rates take longer to bear fruit but ultimately yield a greater accuracy. This can be seen in the graph below, wherein blue represents 0.001, red shows 0.01 and green 0.1. In this graph, a rolling mean of the last 5 accuracy values is taken to smooth the graph and improve interpretability.



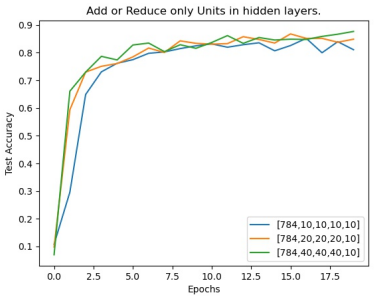
The final test accuracy after 100 epochs vs epsilon value can be seen below, along with the same graph with a logarithmic scale for readability. As can be seen, high learning rates yield a very inaccurate model, while the peak values are relatively low (0.0025 and 0.025). From this we can propose that the optimal learning rate for this system is 0.0025 given batch size =50, epochs = 100 and a network shape of [784, 20, 20, 20, 10]



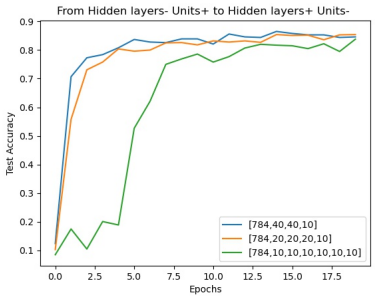
2.4 Network Topology



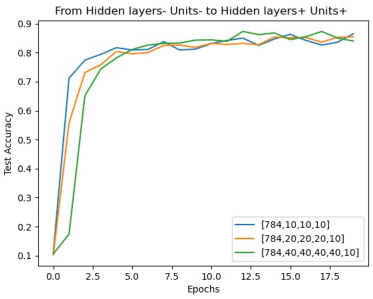
Test 1



Test 2



Test 3



Test 4

We can change either the number of hidden layers or the number of units inside of each layer to measure how it

might be affecting our neural network performance. So I did these tests by choosing the default network shape as the baseline:

From the results of Test 1, we can see, simply increasing or decreasing the number of hidden layers will only speed convergence, but will not improve accuracy.

From the results of Test 2, we change only the number of units inside of the hidden layer. With more units per layer, the neural network converges faster at the beginning of the process. After 15 epochs, we can see the network shape with more units per layer can reach higher accuracy and the highest accuracy recorded is 87.7%.

To see the impact of this combination on neural networks, I further do Test 3 and 4:

For Test 3, the green line shows the network shape with more hidden layers but fewer units have relatively poorer results. It has high convergence time and is less accurate than baseline. In contrast to the green line, the blue line which represents the network shape with less hidden layer but more units inside converges faster and has the same accuracy as the baseline.

For Test 4, the network shape with more hidden layers and units inside has relatively better results. It can reach higher accuracy and the convergence time has only increased a little. Whereas, the network shape with less hidden layers and units inside can only converge faster.

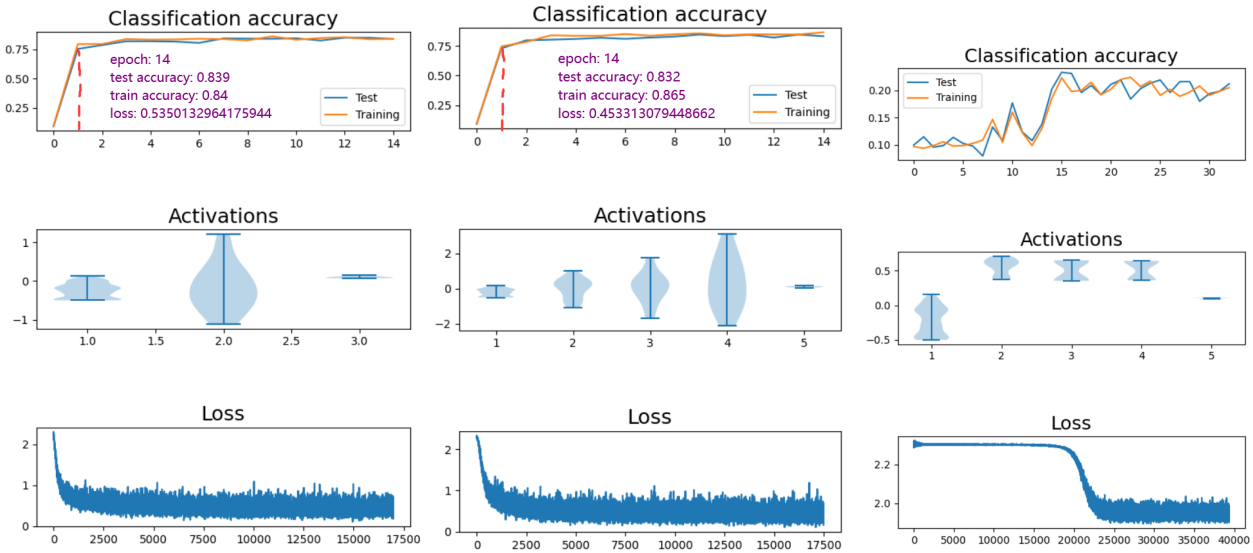
In addition to these simple tests, I did further research by reading the Deep Learning textbook. Here is the summary:

For generally simple data sets, two or three hidden layers are usually sufficient. Extra hidden layers can be used to fit nonlinear functions or to learn complex descriptions. Too few units in hidden layers will lead to under-fitting, too many may lead to over-fitting, using the same number of units for all hidden layers is sufficient.

It is important to note that adding layers will result in greater performance gains than adding more units per layer. It is therefore important not to add too many units to a hidden layer.

2.5 Activation Function

1. If we use the Identity activation function (i.e. a[i] = z[i]) in the hidden layer and SoftMax function as output layer, the performance (Image 1.1) is good. It indicates that this might be a potentially linear problem. To verify, we use one hidden layer instead of three, because the last layer is always the linear function of the first layer, no matter how many hidden layers there are in the neural network. As expected, the performance (Image 1.2) is almost the same as 3 hidden layers.



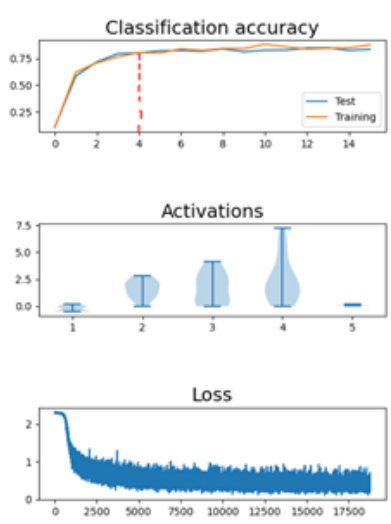
(a) Image 1.1

(b) Image 1.2

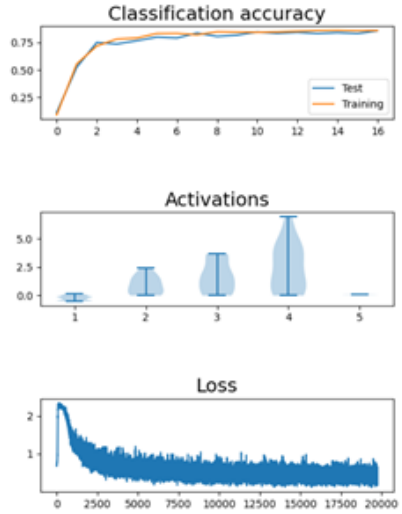
(c) Image 2

2. (Image 2) If we use sigmoid as the activation function in the hidden layer and SoftMax function for the output layer, the accuracy grows very slowly and the loss is still high. The image 2 is about 20 minutes runtime but only with less than 0.25 accuracy. The accuracy can keep growing but will cost a lot of time. This is because when backpropagation through the derivative of the sigmoid, the point which is far from the 0, the gradient becomes very small or even disappears. It is also called the vanishing gradient problem. This causes the weight to have low or almost no effect on the loss function. In addition, the sigmoid function is exponent arithmetic, which has a high computational cost.

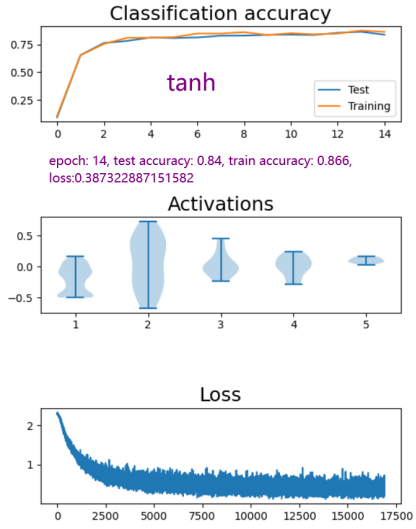
3. (Image 3) If we use ReLU as the activation function in the hidden layer, and SoftMax function for the output layer, we can obtain a high-performance model. It can make predictions accurately. ReLU function learning fast as the accuracy becomes ideal with x-axis only 4. Compared with the Identity function, ReLU only accepts numbers greater than 0. Thus, the hidden layer data distributed greater than 0 but with Identity function layer data can distribute less than 0. In summary, ReLU has nonlinear factors and can be applied to more models.



(a) Image 3



(b) Image 4

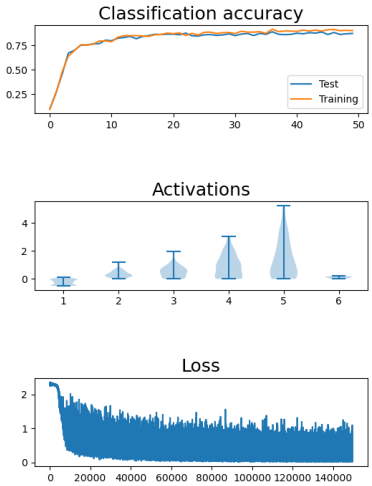


(c) Image 5

4. (Image 4) If we use ReLU as the activation function in the hidden layer, and Sigmoid function for the output layer, it has a good performance in this dataset. In this case, Sigmoid function has a similar action as SoftMax. They both convert a value to the range (0,1), the difference is all outputs of SoftMax are sum to one. Thus, SoftMax is often used to solve the multi categories problem and Sigmoid is often used to solve the problem with only two categories.

5. (Image 5) This is using tanh activation function in hidden layer and SoftMax as output layer.

3 Conclusion



In this report, the effects of each hyper-parameter of a mini-batch gradient descent algorithm were systematically explored in isolation from each other, and potential optimal values for this data set were found. Namely: *batch size = 20; epsilon = 0.0025; 50 epochs; network shape = [784, 60, 60, 60, 60, 10]; ReLU activation function for each hidden layer into a Softmax function in the output layer.* Using these hyper-parameter values, one further test was conducted to verify these results by reaching the highest test-set classification accuracy. Indeed. in this test, shown in the figure above, a maximum test accuracy of 89.3% was reached, verifying the findings in this report.