Hand Drawn Digit Classification

Neural Net
(Bayesian Back-Propagation)

Thomas Carney

Overview

The goal of this project was to implement a neural net with backpropagation that is able to recognise hand draw digits. The neural net uses MNIST training data to adjust all weights and biases in the system. Once the neural net has been trained, one forward pass with MNIST test data results in quick classification of hand drawn digits.

The MNIST data is a 28x28 array of values between 0-1 representing pixel intensity. The neural net takes this data in with an input layer of 784 nodes (one for each pixel). The structure of the neural net can be edited by the “sizes” variable. Each value in the “sizes array” represents a layer in the neural net. For this comparison below sizes = [784, 30, 10], meaning there were 784 nodes on the input layer, 1 hidden layer with 30 nodes, and an output layer of 10 nodes each representing a digit 0-9. More hidden layers can be added by extending the sizes array.

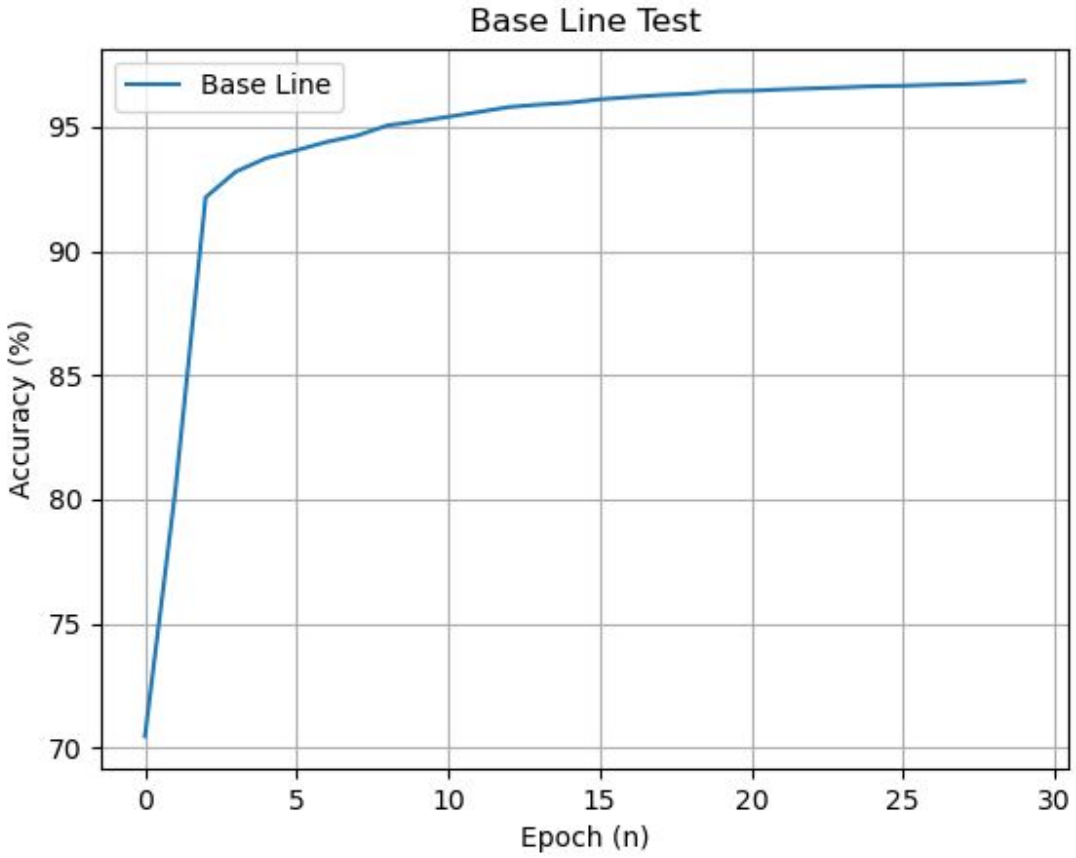
TrainDigit##.csv.gz is the MNIST training data. TestDigit##.csv.gz are the MNIST testing data.

Experiment

Once the Neural net had been implemented in python, the hyper parameters were experimented with to maximise the accuracy of the system over a set number of epochs.

The neural nets tested all had the same shape of 784 nodes in the input layer (28, a single hidden layer with 30 nodes and an output layer of 10 nodes (one for each number).

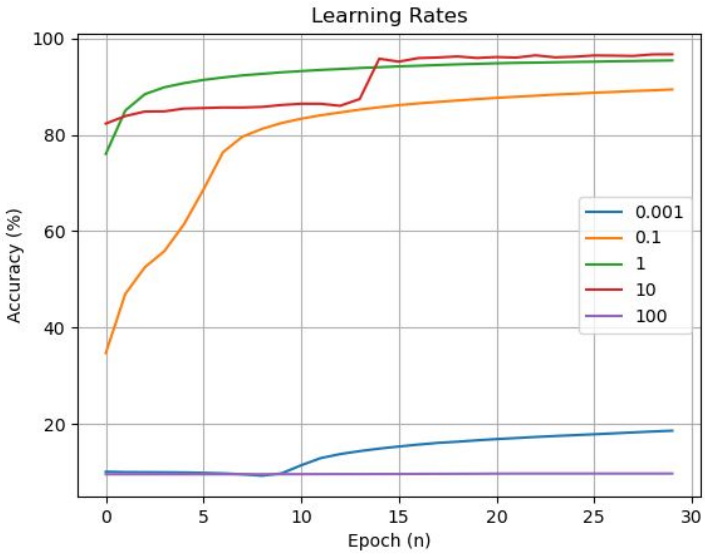
A baseline was set with the hyper parameters: Epochs = 30, minibatch = 20, LR = 3. The accuracy over epochs can be seen below.



The baseline had a maximum accuracy of 96.8%.

Learning Rates

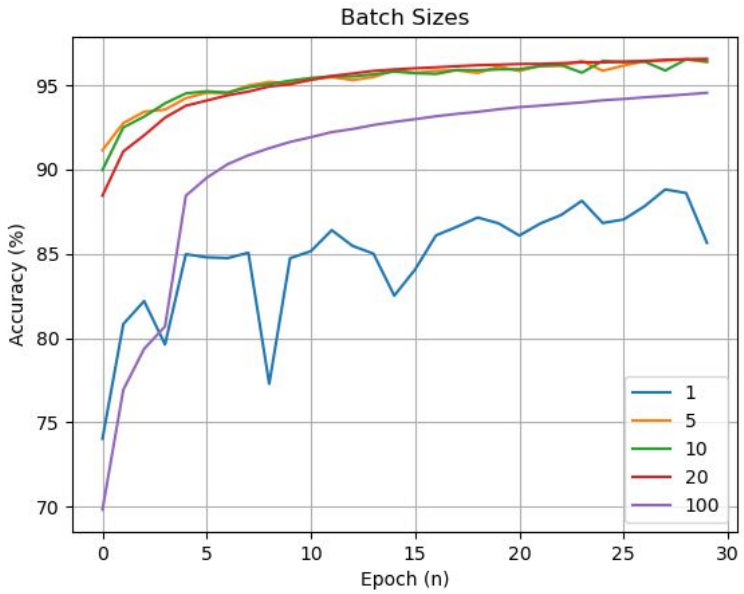
Then the learning rates were tested to see the impact across epochs.



As can be seen in the graph, the smaller learning rates needed a larger number of epochs to increase in accuracy when compared to the larger learning rates. However, the larger learning rates resulted in larger shifts in accuracy. The extremes of both the smallest and largest learning rates were very unsuccessful.

Batch Size

Next the batch size was tested to find the impact on accuracy across epochs.

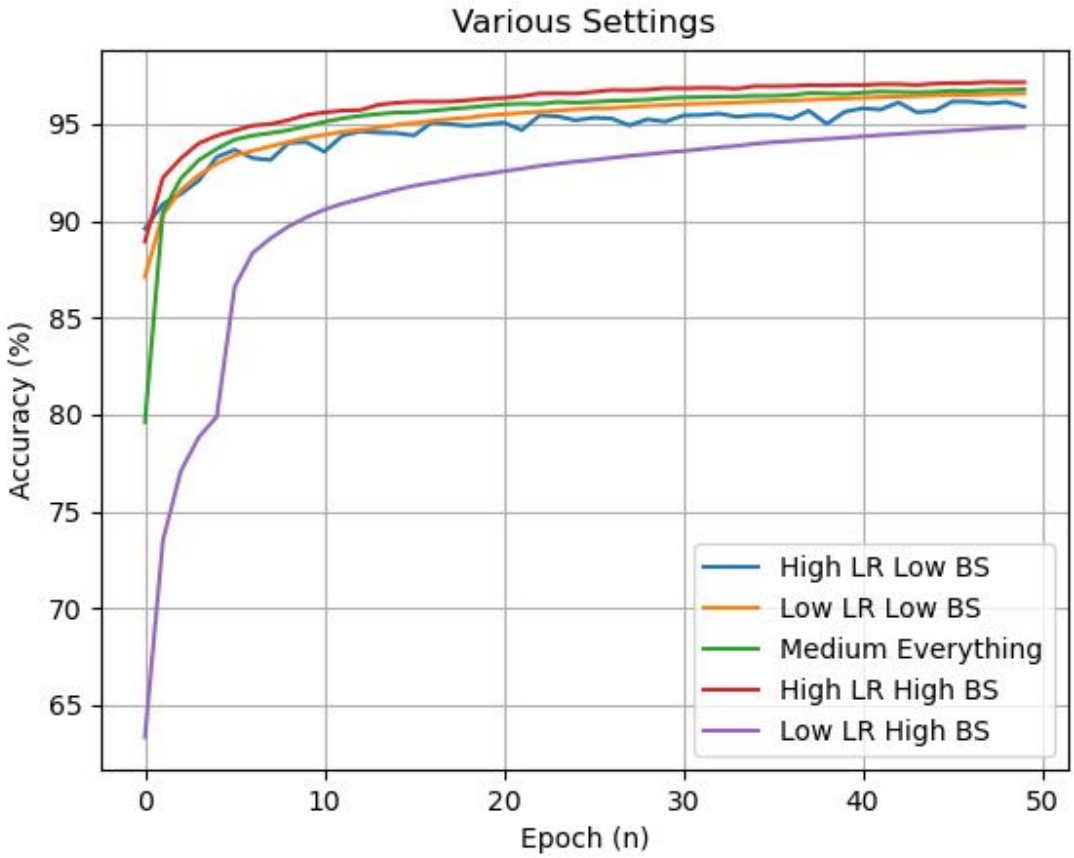


Here we can see that the smaller the batch size, the more the system starts to overfit each set of training data. With a batch size of 1, every forward and backwards pass through the system has a large effect on the weight, resulting in a large shift in accuracy. The larger batch sizes made slower adjustments to accuracy and were less volatile. The greatest accuracy was achieved with a batch size of 20.

Maximising Accuracy

Lastly a test was done with various ranges in batchsize and learning rate. The parameters can be seen below.

```
#  Args: sizes, learningRate, batchSize, epochs
NN1 = NeuralNet(sizes, 10, 10, 50)
NN2 = NeuralNet(sizes, 1, 10, 50)
NN3 = NeuralNet(sizes, 3, 20, 50)
NN4 = NeuralNet(sizes, 10, 50, 50)
NN5 = NeuralNet(sizes, 1, 50, 50)
```



Many of the results were expected. The high learning rate and low batch size was very volatile, as it over fitting each set of training data. The low learning rate, high batch size made a steady slow increase in accuracy over epochs. However, surprisingly the high learning rate and high batch size had the highest accuracy of all (96.9%) and held that for the majority of the epochs.

Conclusion

Changing the hyper parameters of the neural net greatly affects the accuracy of the system. The greatest accuracy was achieved with a learning rate of 10, a batch size of 50. This was quite surprising, especially for the learning rate as it is much higher than you would expect. However, these results could change as the underlying structure of the neural net changes. E.g. if you added another hidden layer different hyper parameters would most likely have different results.

Future Additions

- Storage for trained weights and biases so training is not needed before use
- Test data can be drawn by the user
- Comparisons for accuracy for different shaped neural nets