Draft for Machine learning assignment coms 3007

# Purpose

The purpose of my assignment is to run several machine learning algorithms in order to correctly classify hand written digits as their corresponding ASCII equivalents, or at least predict within an acceptable order of magnitude. These sets of experiments will hopefully show that machine learning can be applied to the field of optical character recognition.

# Tools Used

I made used of the nod**e.**js framework for my coding and scripting of the various experiments. I used the Synaptic neural network library for node.js. This library was capable of handling constructing the given dimensions of a network, changing relevant values, randomising the data and calculating the error during training.

# Dataset Used

I made use of the MNIST database. This is a large database of hand written digits which has been compiled by taking a combination of two databases of hand written digits compiled in the United States. The website for this database can be found here: <http://yann.lecun.com/exdb/mnist/>

This database has a very large training set of 60 000 examples and a set of 10 000 test examples. All the images are normalised, and centred in the same fixed size of 28x28 pixels.

I used the mnist library for node.js to help me load the database into Synaptic. It can be found here: <https://github.com/cazala/mnist>

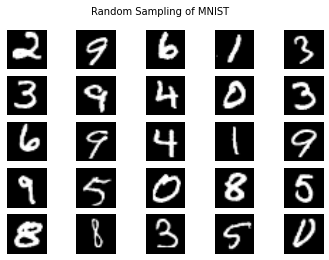
An example of the data used (several sets of samples compiled together) 

<https://camo.githubusercontent.com/d440ac2eee1cb3ea33340a2c5f6f15a0878e9275/687474703a2f2f692e7974696d672e636f6d2f76692f3051493378675875422d512f687164656661756c742e6a7067>

Using this library I was able to specify the number of training samples and testing samples I wanted. No two sets are the same using the functionality of this library. I specified 700 training examples and 20 testing examples.

My neural networks all had 784 input nodes (which correspond per pixel to the input image samples) and 10 output nodes (which correspond to the digits 0-9) as my attributes of classification. My networks setup using Synaptic JS accept a 784 length array of values (floats) and outputs a 10 length array of values (floats). I run the test data on the network after training and round the output values to the nearest integer after the network.

Here is an example of a random sampling of data:



<http://ampcamp.berkeley.edu/6/exercises/img/mnist-example-ipy.png>

# General Restrictions and Normalisation Applied

# Algorithms Used:

Note: See results for the error returned on the test set.

Synaptic runs multiple iterations of each network when training. Each iteration randmosies the data so the error has variation, each iteration. On all experiments I ran 10 iterations (detailed in Results).

Experiment 1: Neural network with one hidden layer with 100 nodes and learning rate of 0.3

Experiment 2: Neural network with two hidden layers with 50 nodes on each layer and a learning rate of 0.3

Experiment 3: Neural network with three hidden layers with 25 nodes on each layer and a learning rate of 0.3

Experiment 4: Neural network with three hidden layers with 50 nodes on each layer and a learning rate of 0.3

Experiment 5: Neural network with two hidden layers with 50 nodes on each layer and a learning rate of 0.2

Experiment 6: Neural network with three hidden layers with 20 nodes on each layer and a learning rate of 0.2

Experiment 7: Neural network with three hidden layers with 25 nodes on layer 1, 50 modes on layer, 25 nodes on layer 3 and a learning rate of 0.3

Experiment 8: RBF network with 25 RBF nodes and random centres and a learning rate of 0.3

# Results

The results here are pulled from the raw output that Synaptic returned when it was run at each configuration. One test set from each experiment is also shown. Both the expected result and actual values before being normalised are shown.

# Conclusion

The worst result of any experiment was experiment 1. It was also by far the slowest running of any of the other experiments. The errors produced per iteration were also largely inconsistent of other experiments. I ran all the experiments several times to make sure that the results remained consistent within each experiment. I think this network performed the worst and was the most inconsistent was because it only utilised one hidden layer of nodes. This meant that all the information about the features the network had to learn had to be stored in this network, whilst no reductions on features could be achieved and then processed by deeper layers of the network. It also was the network with the most nodes within a hidden layer. (Other networks had fewer nodes per internal layers) This is what made it the slowest as there were many more connections between the input and output layers and the hidden layer than in the other networks. This would have made this network vastly more inefficient.

Experiment 6 had the best results of any of the experiments; it was also the fastest experiment run. The errors produced per iteration were also very similar between all 10 iterations. It had a slightly lower learning rate of 0.2 and less nodes than experiment 3 but it performed much better than experiment 3. The error was lower and it was about doubly as fast. The speed can be explained the same as experiment 1 being very slow. As there were less nodes in experiment 6’s network than experiment 3’s, this means that there are fewer connections and therefore fewer calculations when computing the network. Experiment 6 being more accurate than experiment 3 can be explained due to the decrease in the learning rate thereby making experiment 6 slightly more accurate as the weights between the nodes are updated. The randomness of the inputs can also account for some variation between different runs.

# Discussion

Whilst the results of the different experiments were interesting several things could have been changed to help produce cleaner results. The input images could have been cropped smaller since the digits are surrounded by a black border – this is a process called data augmentation. This would produce less input nodes to deal with, making all the different network configurations faster and more accurate. Generalisation would also be improved using this technique.

RBF networks are hard to apply to images. RBF and K-Means both rely on Euclidean distance which in images is a very inaccurate measure. The luminosity between pixels in images, especially images scanned from reality tend to have massive variations. This causes massively different Euclidean distances to be produced between the data points. Also having radial basis functions trying to classify image data will cause other problems such as the curse of dimensionality. Due to the loss of data there would be a higher error rate in trying to learn hand written digits as every light coloured pixel counts in making a good prediction. The other issue here is that many clusters to make a decent classification but pixel location is lost when it is converted into a one dimensional array, and pixel colour is not a very good classification for solving this problem, as the pixels will either be black or white and two clusters is far too few for this kind of prediction. Randomises

Node.JS is not the best language and framework to use with machine learning. I chose it because it handles data and primitives very easily. However it is not built for massive calculations, and intense system usage. It is slower than other languages which do not utilise a virtual machine.