DEEP LEARNING LAB COURSE

ASSIGNMENT 1 – IMPLEMENTATION OF A FEED FORWARD NEURAL NETWORK

SUBMITTED BY: SANIEA AKHTAR

MATRICULATION NUMBER: 4156541

TASK

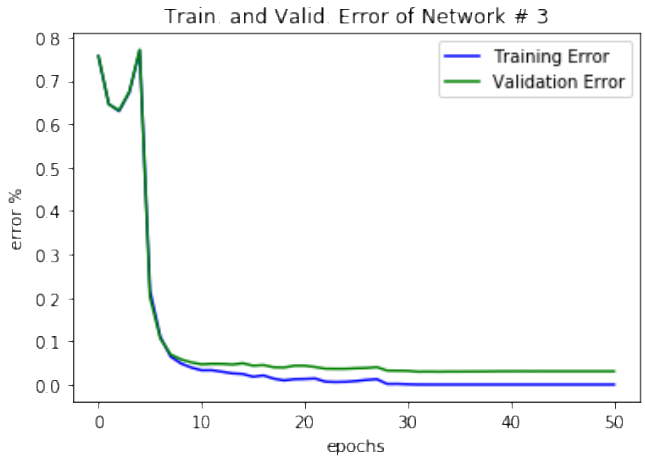
The aim of this task was to implement a feed forward neural network in Python. A skeleton of the code was provided to assist with the implementation. The neural network was to be trained on selected data using the MNIST handwriting database. The database contains a total of 60k training images and 10k testing images. At the end, the neural network should be able to classify images correctly to an accuracy of 1 – 3% on the validation set.

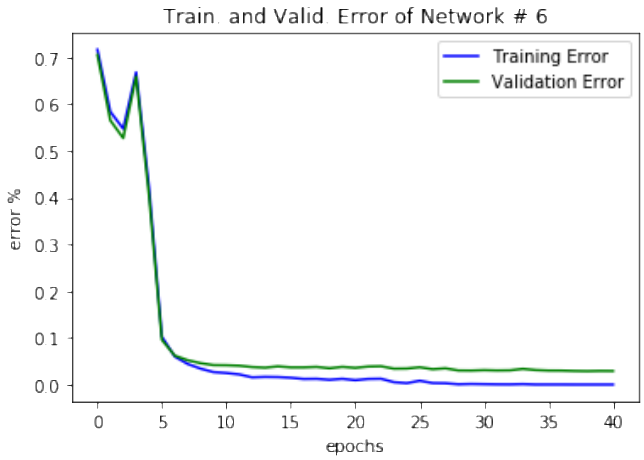
RESULTS

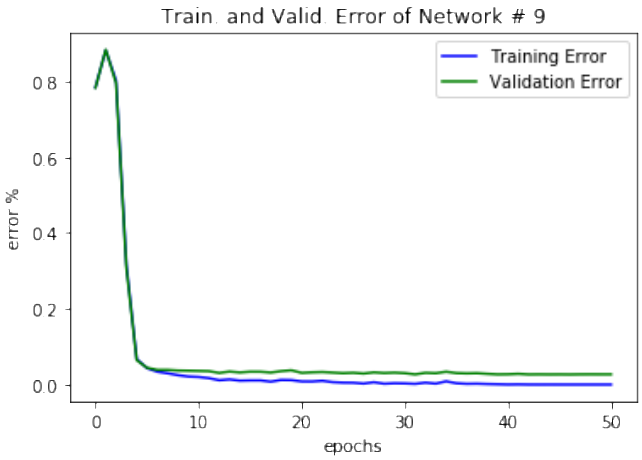
After tweaking the training parameters, I was able to achieve a maximum accuracy of 2.78% in the test set. I tried out the following set of tests and their outcomes are also listed. It was interesting to note that at some point, increasing the number of epochs was not helpful in reducing the validation error, rather made it worse. This proved that just running the network for a longer duration with a larger batch size, does not necessarily promise a better outcome.

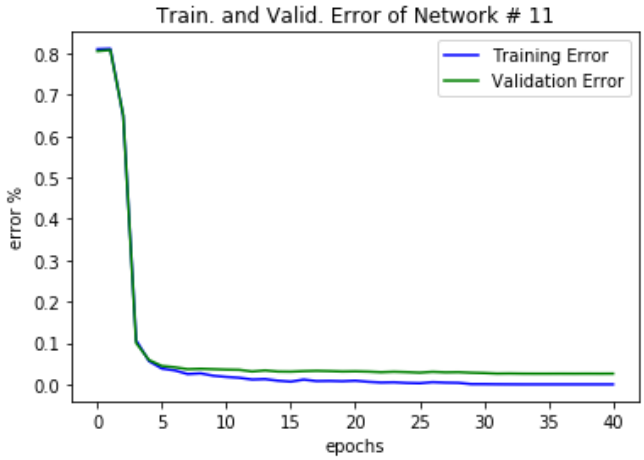
NETWORK #	TRAINING SET	EPOCHS	BATCH SIZE	LEARNING RATE	TRAINING ERROR %	VALIDATION ERROR %	TEST ERROR %
Standard	10,000	30	64	0.10	0.2800	4.60	4.73
1	30,000	50	84	0.15	0.0000	2.91	3.29
2	30,000	50	64	0.15	0.0067	2.63	2.94
3	30,000	50	64	0.13	0.0067	3.06	3.21
4	30,000	40	84	0.15	0.0033	3.22	3.22
5	30,000	40	64	0.15	0.0067	2.94	3.19
6	30,000	40	64	0.13	0.0100	2.90	3.01
7	40,000	50	84	0.15	0.0000	2.97	2.93
8	40,000	50	64	0.15	0.0000	2.75	2.68
9	40,000	50	64	0.13	0.0000	2.69	2.64
10	40,000	40	84	0.15	0.0025	2.74	2.76
11	40,000	40	64	0.15	0.0025	2.56	2.57
12	40,000	40	64	0.13	0.0000	2.53	2.74

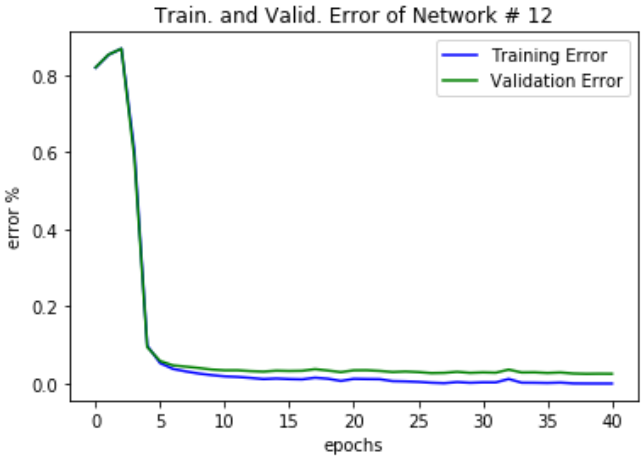
Even though almost all the networks with batch size of 40,000 have validation and test error of < 3%, I chose network 11 as my final network. This network results in the lowest test error at 2.57% and is less prone to overfitting as compared to e.g. network 12 which has a training error of zero and spikes in the validation error plot











Another observation I noted was that the output of the networks was not reproducible, the same network with the same input parameters gives a slightly different error percentage on every run. I think this is because the training set is randomly chosen every time the training is ran from the start, this causes minor differences in parameters learned which then translate into changes in error percentages.

Three correctly and incorrectly classified images have been reproduced and displayed in the jupyter notebook.

DISCUSSION

Since this was the first time implementing a neural network, the task was relatively complicated to understand at the start. While the code stub was provided to help get us started with the exercise, it would have been helpful if the code was commented to explain a bit more as to how the segments of code actually connect and work, in addition to containing #TODO points (which are very helpful).

While implementing the code, at least for me, it was difficult to figure out the matrix multiplications since it wasn't very intuitive to see the format in which the data was imported i.e. (number of samples) x (number of dimensions).

In the gradient descent functions, so far as I understood, gradient descent should continue until the gradient becomes less than an epsilon value. However, in the skeleton code provided, it was required to run the gradient descent over all the batches of data. I think it would be better for parameter optimization if the descent was stopped as soon as the gradient goes below epsilon since afterwards the gradient may start going up again.

Another issue I feel relevant is that it's impossible to troubleshoot the code in segments since the code only runs when it's entirely complete. At that point, troubleshooting becomes a tedious task.