# B551 - Assignment 5

Authors: Anup Bharadwaj, Raghuveer Kanchibail, Supreeth Keragod

## Neural Network:

A brief description of how the program works, assumptions and formulation of the problem is provided as a comment on top of ‘nnet.py’ file.

We tried varying some of the parameters of the neural network to see if they improve in accuracies. Following are some of the results observed with varying parameters:

### Size of training set:

The accuracies increased greatly with increase in the size of the training set initially and slowly the difference became smaller. We couldn’t record results of all the tests we made, but below are some of the results observed:

|  |  |  |
| --- | --- | --- |
| No. of images in the training set | Accuracy (%) | Running time (hh:mm:ss) |
| 40000 | 64.36 | 00:23:40 |
| 12000 | 63.62 | 00:04:40 |
| 2000 | 55.99 | 00:00:47 |

### Learning rate:

Learning rate had the most impact in accuracies for training a neural network. The accuracies dropped if the learning rate was too high or too low. Since I was using a smooth approximation for rectifier functions, the values would overflow out of the supported range for most learning rates. Following are some of the results observed for varied learning rate that successfully trained the model:

|  |  |  |
| --- | --- | --- |
| Learning rate | Accuracy (%) | Running time (hh:mm:ss) |
| 1/length of training data | 55 – 65 (depends on training data) | 00:00:47 – 00:23:40 |
| 0.001 | 59.59 (12000 files) | 00:06:39 |
| 0.1 | 49.2 (12000 files) | 00:13:37 |
| 0.00001 | 62.77 (12000 files) | 00:05:21 |

### Activation Function:

I experimented on 4 Activation functions – ReLu, Leaky ReLu, smooth approximation of ReLu and sigmoid functions. The best performance was by leaky reLu and the smooth approximation of ReLu. ReLu performed worst as most outputs would come to zero and the network stopped converging. Below are the results when ran for the full training dataset:

|  |  |  |
| --- | --- | --- |
| Activation Function | Accuracy (%) | Running time (hh:mm:ss) |
| Leaky ReLu | 64.26 | 00:23:29 |
| Smooth approximation of ReLu | 64.36 | 00:23:40 |
| ReLu | ~25% (converged to a single orientation) | ~00:10:00 (Lost the output file) |
| Sigmoid | 61.86% | ~ 00:23:00 (Lost the output file) |

### Traditional vs stochastic gradient:

Both gave very similar results but traditional way took longer to train the model. Since the initial weights are random, there were slight differences in the accuracies observed. The only significant difference noticed was the time taken. Stochastic gradient was able to train 40000 images in under 30 minutes while traditional approach took over an hour.

### Initial weight distribution:

Since, we assign the initial weights randomly, we tried two ways of assigning initial weights – pseudorandom generator and Gaussian. The range did matter as higher ranges resulted in overflow errors and Guassian distribution of randomness yielded better accuracies.

|  |  |  |
| --- | --- | --- |
| Random generator | Accuracies | Time taken |
| Pseudo-random generator | ~53-61 | ~00:23:00 |
| Gaussian | ~62-64 | ~00:23:00 |

### Sample run

Some of the classifications of a sample run is as follows:

|  |  |  |
| --- | --- | --- |
| File name | True Orientation | Prediction |
| 13962200152 | 180 | 0 |
| 434740832 | 90 | 90 |
| 2304830497 | 180 | 180 |
| 5430983827 | 180 | 90 |
| 4413965131 | 0 | 180 |

In most cases, the model got confused with inverted pictures, the errors were mostly misclassification of 0 as 180 (vice versa) and 90 as 270 (and vice versa). Full results can be found at the file: sample\_test in github

### Recommendation:

We would recommend using a Neural net classifier with Leaky ReLu as activation function, 1/len(dataset) as the learning rate, 64 hidden nodes and a Gaussian random value generator to converge faster. Even though the smooth approximation of ReLu provides better accuracies, it is vulnerable to overflows unless handled correctly.