EE6132 Assignment No. 1: Report

-Akash Reddy A, EE17B001

September 5, 2019

Overview

The aim of this assignment has been to implement a Multilayer Feedforward Neural Network (MLP) with Backpropagation (BP) learning. Additionally, the effects of different activations, noise, regularisation, and comparison with other Machine Learning algorithms have been experimented with.

1 MLP Architecture and Training (baseline model)

Note: Average Error Rate has been computed as the mean of the error rates for each digit, where the error rate for each digit i has been evaluated as:

$$error_i = 1 - \frac{TP_i + TN_i}{TP_i + FP_i + FN_i + TN_i}$$

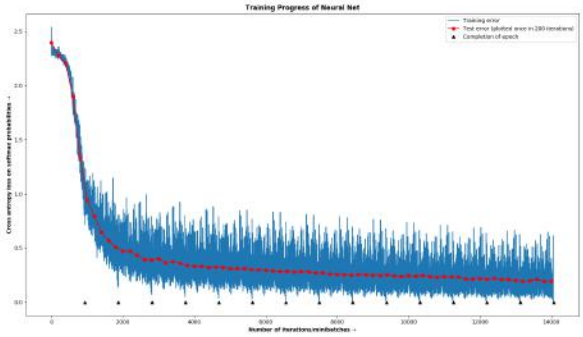
where for example for the digit 1, TP_1 = True Positives (all 1s predicted as 1), FP_1 = False Positives (all non-1s predicted as 1), FN_1 = False Negatives (all 1s predicted as non-1), TN_1 = True Negatives (all non-1s predicted as non-1).

Standard Deviation of the Error Rate is the standard deviation of the 10 error rates evaluated for each digit.

Training Loss, Test Loss Plots and Metrics for Sigmoidal Activation:

In each of the plots, each iteration is considered to be one run of gradient descent over a single minibatch of 64 samples. 15 epochs of a 60000 strong set would therefore lead to roughly 14000 iterations.

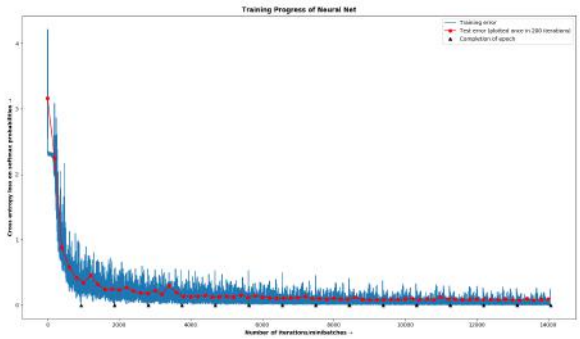
First, we explore different learning rates for the sigmoidal activated baseline MLP. The metrics for each algorithm are reported in the cropped terminal window screenshot to the right of the plot.




```
Total # of epochs: 15
Epoch #1 done...
Accuracy on test set after epoch #1 = 66.41%
Epoch #2 done...
Accuracy on test set after epoch #2 = 82.48%
Epoch #3 done...
Accuracy on test set after epoch #3 = 86.49%
Epoch #4 done...
Accuracy on test set after epoch #4 = 88.270000000000001%
Epoch #5 done...
Accuracy on test set after epoch #5 = 89.2%
Epoch #6 done...
Accuracy on test set after epoch #6 = 90.07%
Epoch #7 done...
Accuracy on test set after epoch #7 = 90.83%
Epoch #8 done...
Accuracy on test set after epoch #8 = 91.45%
Epoch #9 done...
Accuracy on test set after epoch #9 = 91.86%
Epoch #10 done...
Accuracy on test set after epoch #10 = 92.42%
Epoch #11 done...
Accuracy on test set after epoch #11 = 92.820000000000001%
Epoch #12 done...
Accuracy on test set after epoch #12 = 93.11%
Epoch #13 done...
Accuracy on test set after epoch #13 = 93.45%
Epoch #14 done...
Accuracy on test set after epoch #14 = 93.76%
Epoch #15 done...
Accuracy on test set after epoch #15 = 94.01%
Confusion Matrix:
[[ 960.    0.    1.    1.    0.    8.    6.    1.    2.    1.]
 [   0. 1111.    2.    2.    1.    1.    3.    1.   13.    1.]
 [  14.    8.  943.    5.    9.    1.   14.    8.   29.    1.]
 [   2.    1.   13.  929.    0.   26.    1.    9.   21.    8.]
 [   1.    2.    4.    0.  925.    0.    8.   10.    1.    4.   35.]
 [   7.    2.    0.   13.    3.  824.   13.    8.   22.    8.]
 [   9.    3.    1.    0.    8.   16.  915.    0.    6.    0.]
 [   2.   15.   15.    4.    5.    1.    0.  933.    1.   52.]
 [   5.    6.    1.   12.    9.   16.    5.    3.  912.    5.]
 [  10.    6.    1.    5.   23.    5.    0.    4.    6.  949.]]
...
Average error rate = 1.1979999999999%
Standard deviation of error rate = 0.33498656689485234%


Final accuracy on test set = 94.01%
Final precision on test set = 94.08855846997193%
Final recall on test set = 93.97312665766175%
Final F1-score on test set = 0.9395735579156144
```

Figure 1: Learning Rate α=0.1

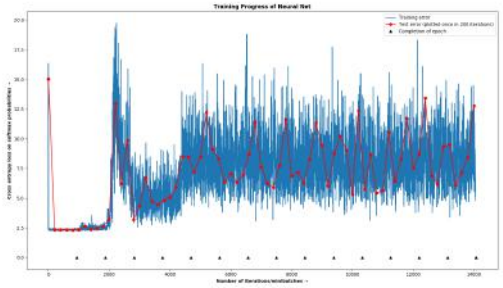
For $\alpha = 0.1$, we have a decent convergence of the losses, leading to a final test accuracy of $\approx 94\%$.




```
Total # of epochs: 15
Epoch #1 done...
Accuracy on test set after epoch #1 = 84.25%
Epoch #2 done...
Accuracy on test set after epoch #2 = 89.75%
Epoch #3 done...
Accuracy on test set after epoch #3 = 92.92%
Epoch #4 done...
Accuracy on test set after epoch #4 = 94.37%
Epoch #5 done...
Accuracy on test set after epoch #5 = 95.37%
Epoch #6 done...
Accuracy on test set after epoch #6 = 96.08%
Epoch #7 done...
Accuracy on test set after epoch #7 = 96.73%
Epoch #8 done...
Accuracy on test set after epoch #8 = 97.1%
Epoch #9 done...
Accuracy on test set after epoch #9 = 97.22%
Epoch #10 done...
Accuracy on test set after epoch #10 = 97.31%
Epoch #11 done...
Accuracy on test set after epoch #11 = 97.38%
Epoch #12 done...
Accuracy on test set after epoch #12 = 97.399999999999999%
Epoch #13 done...
Accuracy on test set after epoch #13 = 97.54%
Epoch #14 done...
Accuracy on test set after epoch #14 = 97.53%
Epoch #15 done...
Accuracy on test set after epoch #15 = 97.58%
Confusion Matrix:
[[ 973.    0.    1.    0.    1.    1.    0.    1.    3.    0.]
 [   0. 1130.    1.    1.    0.    1.    0.    0.    1.    1.]
 [   7.    3. 1000.    3.    4.    0.    3.    4.    2.    0.]
 [   0.    0.    6.  989.    0.    2.    0.    5.    4.    4.]
 [   1.    0.    1.    0.  963.    0.    4.    2.    0.   11.]
 [   4.    1.    0.   10.    2.  871.    1.    0.    0.    3.]
 [   5.    3.    2.    0.   14.   20.  911.    0.    3.    0.]
 [   1.    7.    8.    1.    2.    0.    0.  997.    1.   11.]
 [   5.    1.    3.    7.    6.    5.    2.    3.  938.    4.]
 [   4.    2.    0.    5.   10.    1.    0.    1.    0.  980.]]
...
Average error rate = 0.484%
Standard deviation of error rate = 0.12026637102698357%


Final accuracy on test set = 97.58%
Final precision on test set = 97.573750642638235%
Final recall on test set = 97.546566106253386%
Final F1-score on test set = 0.9755124260851021
```

Figure 2: Learning Rate α=1

For $\alpha = 1$, we have almost the best and fastest convergence of the losses, leading to a high final test accuracy of $\approx 97\%$.




```
Total # of epochs: 15
Epoch #1 done...
Accuracy on test set after epoch #1 = 11.49%
Epoch #2 done...
Accuracy on test set after epoch #2 = 9.74%
Epoch #3 done...
Accuracy on test set after epoch #3 = 9.74%
Epoch #4 done...
Accuracy on test set after epoch #4 = 10.280000000000001%
Epoch #5 done...
Accuracy on test set after epoch #5 = 10.100000000000001%
Epoch #6 done...
Accuracy on test set after epoch #6 = 10.280000000000001%
Epoch #7 done...
Accuracy on test set after epoch #7 = 10.09%
Epoch #8 done...
Accuracy on test set after epoch #8 = 9.74%
Epoch #9 done...
Accuracy on test set after epoch #9 = 10.100000000000001%
Epoch #10 done...
Accuracy on test set after epoch #10 = 9.74%
Epoch #11 done...
Accuracy on test set after epoch #11 = 9.74%
Epoch #12 done...
Accuracy on test set after epoch #12 = 8.92%
Epoch #13 done...
Accuracy on test set after epoch #13 = 9.74%
Epoch #14 done...
Accuracy on test set after epoch #14 = 9.74%
Epoch #15 done...
Accuracy on test set after epoch #15 = 9.74%
PA1_EC17B001.py:215: RuntimeWarning: invalid value encountered in true_divide
  pr=diag(conf)/sum(conf,axis=0)
Confusion Matrix:
[[  0.    0.    0.    0.    0.    0.    0.  980.    0.]
 [  0.    0.    0.    0.    0.    0.    0. 1135.    0.]
 [  0.    0.    0.    0.    0.    0.    0. 1032.    0.]
 [  0.    0.    0.    0.    0.    0.    0. 1010.    0.]
 [  0.    0.    0.    0.    0.    0.    0.  982.    0.]
 [  0.    0.    0.    0.    0.    0.    0.  892.    0.]
 [  0.    0.    0.    0.    0.    0.    0.  958.    0.]
 [  0.    0.    0.    0.    0.    0.    0. 1028.    0.]
 [  0.    0.    0.    0.    0.    0.    0.  974.    0.]
 [  0.    0.    0.    0.    0.    0.    0. 1009.    0.]]
...
Average error rate = 18.852%
Standard deviation of error rate = 24.07645563616039?%


Final accuracy on test set = 9.74%
Final precision on test set = nan%
Final recall on test set = 10.0%
Final F1-score on test set = nan
```

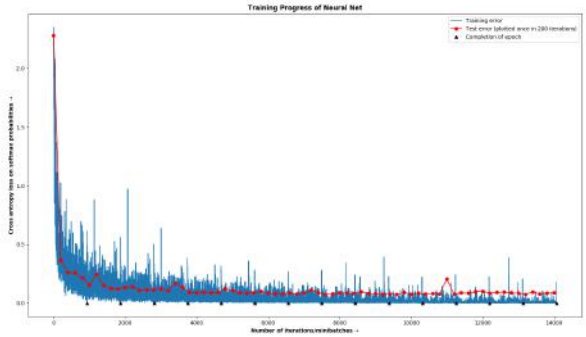
Figure 3: Learning Rate α=7

For the high $\alpha = 7$, the convergence of the losses is much worse. This is because the learning rate is too large and it cannot take small enough jumps to reach the minimum during gradient descent. Hence, it tends to overshoot to remain in the higher values of the cost function.

2 Activation Function

ReLU

When it comes to the ReLU, we find that the previous near-optimal learning rate $\alpha = 1$ is too large. Upon playing with the learning rate, we get a good convergence for $\alpha = 0.1$ with a final test accuracy of $\approx 98\%$.



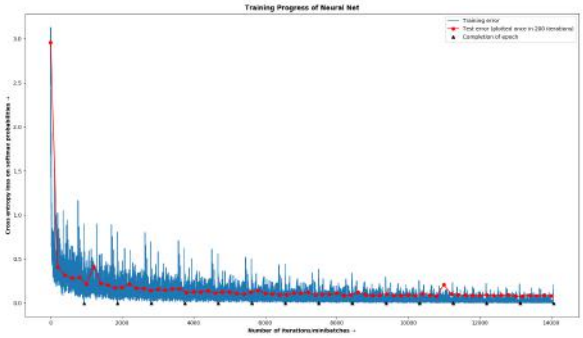

```
Total # of epochs: 15
Epoch #1 done...
Accuracy on test set after epoch #1 = 90.63%
Epoch #2 done...
Accuracy on test set after epoch #2 = 94.39999999999999%
Epoch #3 done...
Accuracy on test set after epoch #3 = 96.13000000000001%
Epoch #4 done...
Accuracy on test set after epoch #4 = 96.72%
Epoch #5 done...
Accuracy on test set after epoch #5 = 97.26%
Epoch #6 done...
Accuracy on test set after epoch #6 = 97.34%
Epoch #7 done...
Accuracy on test set after epoch #7 = 97.37%
Epoch #8 done...
Accuracy on test set after epoch #8 = 97.27%
Epoch #9 done...
Accuracy on test set after epoch #9 = 97.52%
Epoch #10 done...
Accuracy on test set after epoch #10 = 97.28%
Epoch #11 done...
Accuracy on test set after epoch #11 = 97.48%
Epoch #12 done...
Accuracy on test set after epoch #12 = 97.8%
Epoch #13 done...
Accuracy on test set after epoch #13 = 97.47%
Epoch #14 done...
Accuracy on test set after epoch #14 = 97.68%
Epoch #15 done...
Accuracy on test set after epoch #15 = 97.78%
Confusion Matrix:
[[ 974.    0.    1.    0.    0.    1.    1.    0.    2.    1.]
 [   0. 1120.    3.    2.    0.    0.    2.    0.    2.    0.]
 [   6.    1. 1015.    1.    1.    0.    1.    3.    4.    0.]
 [   0.    0.    7.  988.    0.    3.    0.    3.    3.    6.]
 [   0.    0.    5.    0.  964.    0.    3.    1.    0.    9.]
 [   2.    0.    0.    6.    1.  875.    2.    0.    4.    2.]
 [   6.    3.    0.    1.    6.   12.  930.    0.    0.    0.]
 [   3.    4.   13.    1.    6.    0.    0.  985.    5.   11.]
 [   3.    0.    5.    5.    2.    6.    1.    2.  944.    6.]
 [   2.    2.    0.    3.   14.    5.    1.    3.    2.  977.]]
...
Average error rate = 0.04399999999999995%
Standard deviation of error rate = 0.13809227494359493%


Final accuracy on test set = 97.78%
Final precision on test set = 97.76629489281331%
Final recall on test set = 97.76733012881792%
Final F1-score on test set = 0.9776157590344565
```

Figure 4: Learning Rate α=0.1

tanh

When it comes to tanh, we find that the previous near-optimal learning rate $\alpha = 1$ is slightly large, as the final test accuracy gets stuck at roughly 89% when $\alpha = 1$ is used. Upon playing with the learning rate, we get a good convergence for $\alpha = 0.1$ with a final test accuracy of $\approx 97\%$.

We also observe a slower convergence in tanh than in ReLU. This can be explained by the fact that the ReLU neurons which enter into the negative values for z are made equal to the fixed 0. They change no more after this. Therefore, the neurons in the ReLU algorithm become inactive quicker, leading to faster convergence.




```
Total # of epochs: 15
Epoch #1 done...
Accuracy on test set after epoch #1 = 90.78%
Epoch #2 done...
Accuracy on test set after epoch #2 = 91.979999999999%
Epoch #3 done...
Accuracy on test set after epoch #3 = 92.78%
Epoch #4 done...
Accuracy on test set after epoch #4 = 94.05%
Epoch #5 done...
Accuracy on test set after epoch #5 = 94.76%
Epoch #6 done...
Accuracy on test set after epoch #6 = 95.08%
Epoch #7 done...
Accuracy on test set after epoch #7 = 95.25%
Epoch #8 done...
Accuracy on test set after epoch #8 = 95.45%
Epoch #9 done...
Accuracy on test set after epoch #9 = 95.65%
Epoch #10 done...
Accuracy on test set after epoch #10 = 95.91%
Epoch #11 done...
Accuracy on test set after epoch #11 = 96.25%
Epoch #12 done...
Accuracy on test set after epoch #12 = 96.64%
Epoch #13 done...
Accuracy on test set after epoch #13 = 96.89%
Epoch #14 done...
Accuracy on test set after epoch #14 = 96.960000000000001%
Epoch #15 done...
Accuracy on test set after epoch #15 = 97.03%
Confusion Matrix:
[[ 970.    0.    1.    1.    1.    4.    8.    8.    1.    2.]
 [   0. 1129.    2.    1.    0.    0.    0.    0.    2.    1.]
 [   5.    2. 1006.    3.    1.    1.    1.    2.   11.    0.]
 [   0.    2.    9.  975.    1.    9.    0.    3.    3.    8.]
 [   1.    0.    3.    1.  963.    2.    4.    0.    0.    8.]
 [   3.    0.    0.    4.    1.  877.    2.    1.    2.    2.]
 [   7.    4.    1.    0.    8.   32.  904.    0.    2.    0.]
 [   1.   14.   15.    2.   11.    0.    0.  950.    4.   23.]
 [   4.    2.    2.    4.    2.    7.    1.    2.  947.    3.]
 [   1.    2.    2.    3.   18.    4.    1.    1.    3.  974.]]
...
Average error rate = 0.59399999999979%
Standard deviation of error rate = 0.16371927192606303%


Final accuracy on test set = 97.03%
Final precision on test set = 97.02881020291281%
Final recall on test set = 97.01628532325384%
Final F1-score on test set = 0.9699873917459267
```

Figure 5: Learning Rate α=0.1

Inactive Neurons

When the gradients of neurons don't change much with iterations, this means that they are influencing the change in the neural network parameters lesser and lesser. Therefore, the percentage of inactive neurons is an indicator of convergence of the algorithm to a cost minimum.

It can be clearly seen that in all three cases, the overall percentage of inactive neurons increases as a general trend with iterations. This is because, as the algorithm converges, more and more neurons become inactive.

The percentage of inactive neurons for the sigmoidal activation for its best learning rate, and also the ReLU for its best learning rate are more or less equal. This is because the rate of convergence (if one observes the training/test loss plots) is similar for both these activations. On the other hand, while the percentage of inactive neurons is also increasing for tanh, it doesn't increase as much due to slower convergence.

The percentage of inactive neurons has been plotted once every 200 iterations for better visibility.

Sigmoidal Activation

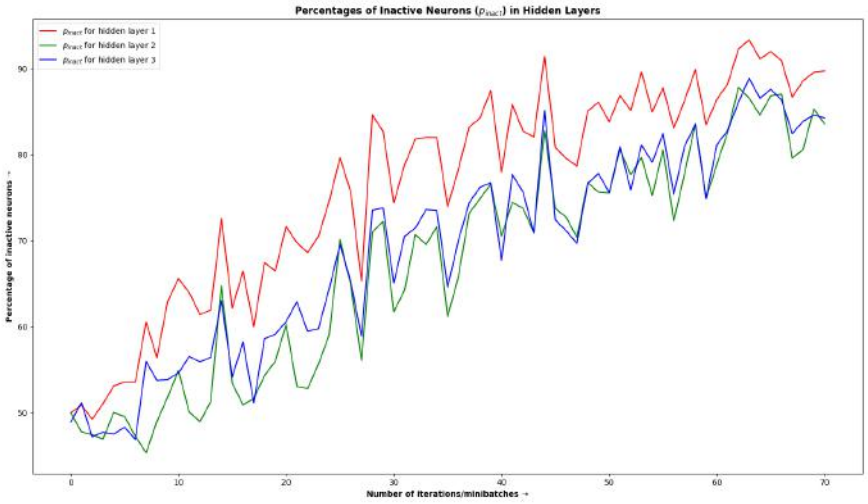


Figure 6: Learning Rate α=1

ReLU Activation

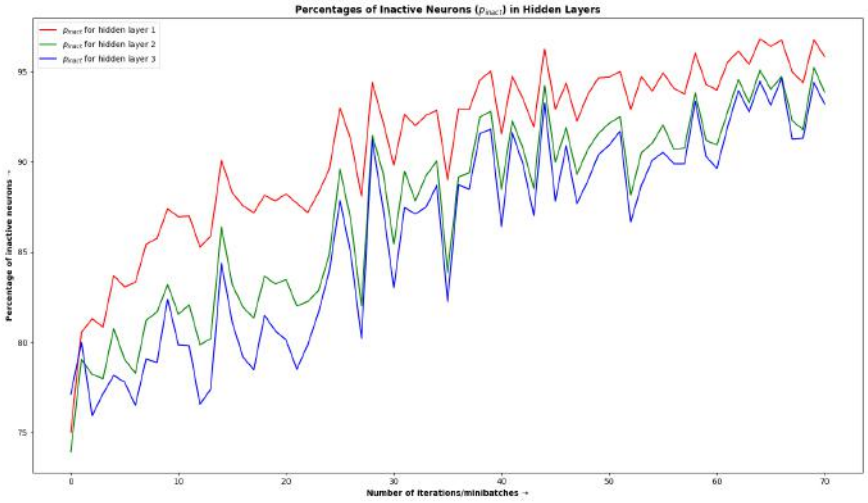


Figure 7: Learning Rate α=0.1

tanh Activation

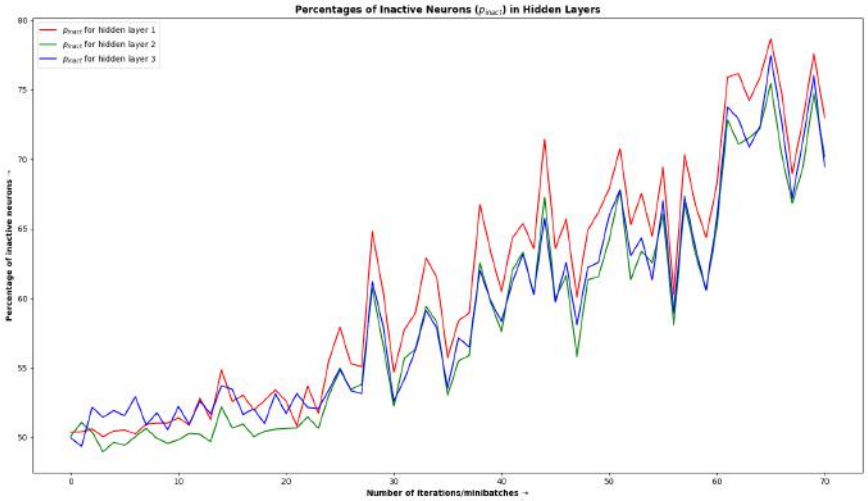


Figure 8: Learning Rate α=0.1

3 Regularisation

1

The addition of Gaussian noise to the hidden layers in the training of the neural network is a mild regularisation method. This is because, by adding noise, we make each of our hidden layer neuron values a stochastic model, with a probability of a Gaussian distribution centred around the original value. This makes the hidden layer values more "general", accounting for deviations from the training data. This is what creates the regularising effect.

However, in our particular case, the regularisation isn't much visible. This is because the training set is already quite large and varied, and hence not much overfitting occurs during training.

It is seen that adding noise in the forward pass is better- in the sense that it converges faster and to a slightly better test set accuracy- than adding noise in the backward pass. This is probably because, adding noise in the forward pass is like adding noise to the input data itself, because the noise features in the calculation of the a_i's during forward pass.

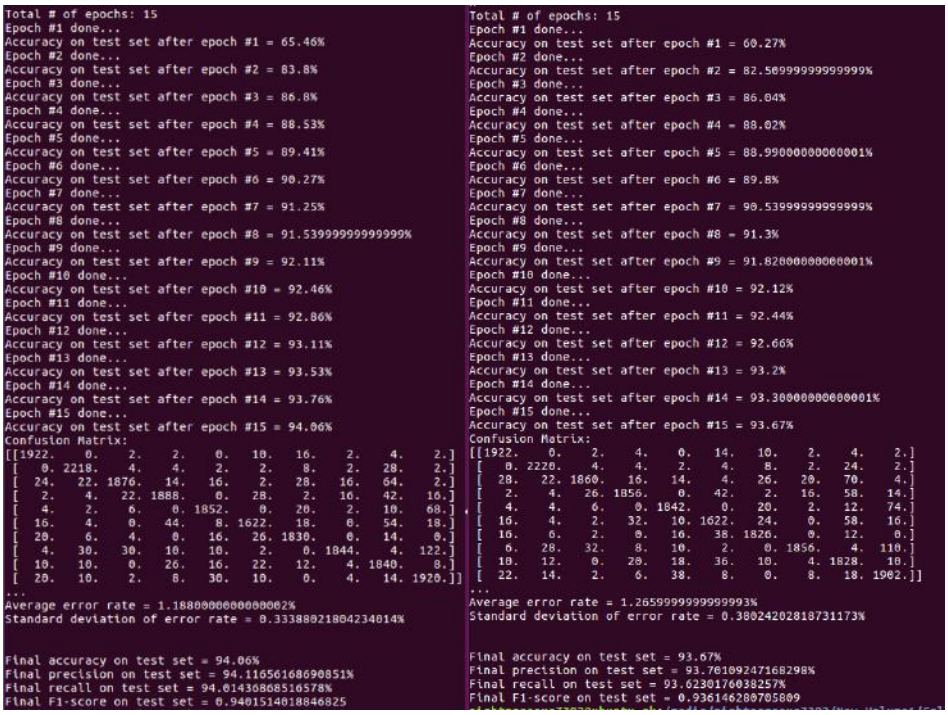


Figure 9: Metrics after adding Forward and Backward Noise respectively- α=0.1 and a Sigmoidal Activation (the confidence matrices have been doubled by mistake)

2

Data Augmentation: Data augmentation has been done by adding noise to the training set and augmenting the original training set with the noisy one. This increases 60000 training examples to 120000 of them, and hence takes nearly twice the time to learn for 15 epochs. This is one of the main disadvantages of data augmentation.

We use sigmoidal activation and learning rate $\alpha = 0.1$ to observe the effects of regularisation. Hence, the beauty of this can be appreciated by comparing with Figure 1- the metrics of the *unregularised*, baseline neural network with sigmoidal activation and learning rate $\alpha = 0.1$.

We can very clearly observe a jump in the accuracy during the first few epochs, as compared to the unregularised case. The final test accuracy is also significantly better. Therefore, data augmentation is very effective at regularising the neural network for better test set accuracy.

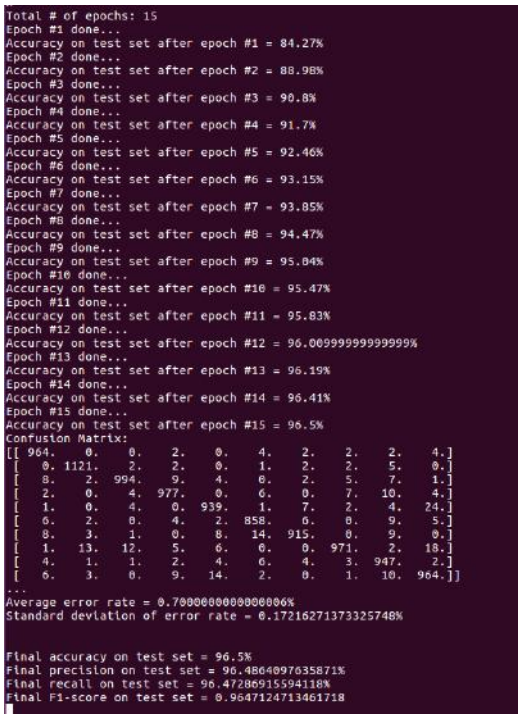


Figure 10: Learning Rate α=0.1 and a Sigmoidal Activation

L2 Regularisation: The implementation of L2 regularisation is done as follows:

- The baseline loss function is now modified to include an L2 weight term:

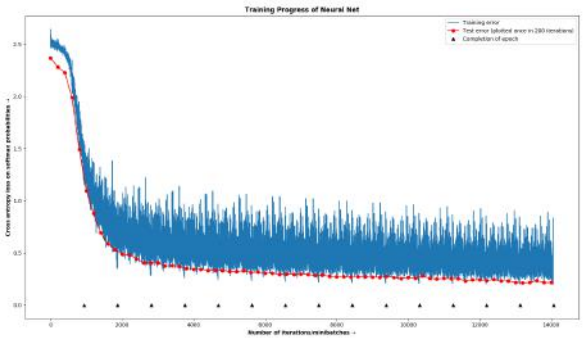
$$J'(W; x, y) = J(W; x, y) + \frac{\lambda}{2m} \sum ||W^2||$$

- The gradient now turns out to be, for each weight:

$$\frac{\partial J'(W; x, y)}{\partial W} = \frac{\partial J(W; x, y)}{\partial W} + \frac{\lambda}{m} W$$

The new values for loss and gradient step are implemented in the code, in order to implement L2 regularisation. The plot is shown below. We observe a faster reduction of test loss

during the initial iterations. Also, during the whole training process, the test loss is at the lower end of the training loss. This is due to good generalisation, as opposed to the plot for the baseline model.




```
Total # of epochs: 15
Epoch #1 done...
Accuracy on test set after epoch #1 = 63.759999999999998%
Epoch #2 done...
Accuracy on test set after epoch #2 = 83.31%
Epoch #3 done...
Accuracy on test set after epoch #3 = 86.78%
Epoch #4 done...
Accuracy on test set after epoch #4 = 88.3%
Epoch #5 done...
Accuracy on test set after epoch #5 = 89.259999999999998%
Epoch #6 done...
Accuracy on test set after epoch #6 = 89.98%
Epoch #7 done...
Accuracy on test set after epoch #7 = 90.59%
Epoch #8 done...
Accuracy on test set after epoch #8 = 91.01%
Epoch #9 done...
Accuracy on test set after epoch #9 = 91.43%
Epoch #10 done...
Accuracy on test set after epoch #10 = 91.8%
Epoch #11 done...
Accuracy on test set after epoch #11 = 92.190000000000001%
Epoch #12 done...
Accuracy on test set after epoch #12 = 92.369999999999999%
Epoch #13 done...
Accuracy on test set after epoch #13 = 92.800000000000001%
Epoch #14 done...
Accuracy on test set after epoch #14 = 93.08%
Epoch #15 done...
Accuracy on test set after epoch #15 = 93.36%
Confusion Matrix:
[[ 968.    0.    1.    1.    0.    7.    6.    1.    3.    1.]
 [   0. 1109.    1.    3.    1.    2.    4.    1.   13.    1.]
 [  15.   13.  918.   18.   10.    1.   15.   10.   37.    3.]
 [   1.    2.   12.  924.    0.   27.    1.    5.   25.   13.]
 [   2.    2.    3.    0.  925.    0.   10.    1.    4.   35.]
 [   9.    2.    1.   18.    7.  816.   10.    8.   20.    9.]
 [  10.    3.    2.    0.    9.   17.  912.    0.    5.    0.]
 [   2.   16.   10.    5.    5.    1.    0.  911.    2.   60.]
 [   5.    6.    0.   14.    9.   13.    6.    3.  907.   11.]
 [  10.    5.    0.    4.   20.    5.    0.    4.    7.  954.]]
...
Average error rate = 1.3279999999999992%
Standard deviation of error rate = 0.3940253798932456%


Final accuracy on test set = 93.36%
Final precision on test set = 93.417112183247514%
Final recall on test set = 93.329020998553778%
Final F1-score on test set = 0.9331128233337912
```

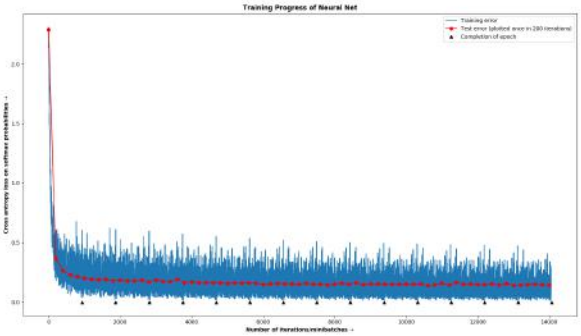
Figure 11: α=0.1 and a Sigmoidal Activation

4 Hand-Crafted Features

I have used `scikit-learn` and `scikit-image` to implement both, my feature extractor- the Histogram of Oriented Gradients (HOG) and the K Nearest Neighbors and SVM algorithms.

HOG works by replacing each cell (of a certain number of pixels) by a single vector representing the histogram of the total gradient magnitudes of all pixels in that cell. The bins of the histogram are the bins of angle that a gradient can point towards, obtained by the division of the total angle. This makes HOG successfully and efficiently contain any edge-related information while compressing the amount of information there is to hold.

In our case, for example, an HOG when implemented on each 28×28 image with 9 orientations and a 7×7 cell gives 16 pixels, with each pixel containing a histogram (or 9 vector values). Therefore, each input has been reduced from 784 intensity values to 144 histogram values.



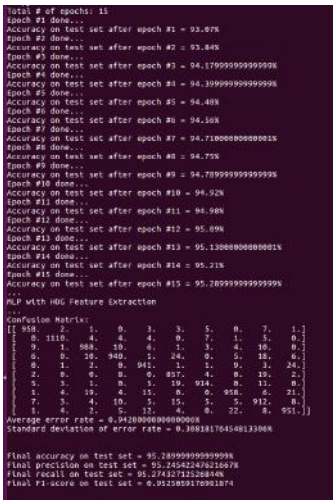


Figure 12: α=0.1 and a tanh Activation

We observe that the K Nearest Neighbors algorithm has less accuracy, and the SVM has more accuracy. This can be explained by considering how the KNN algorithm only considers Euclidean distance as a metric. This can lead to a noisy 8 and a clean 8 being far off from each other and not being clustered together, for instance. A noisy 8 could even end up being closer to a clean 3. Hence, this leads to misclassification, higher error rate and lower accuracy.

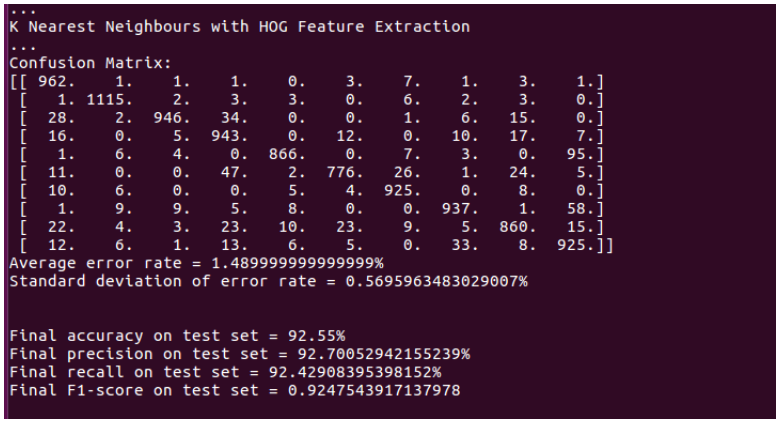


Figure 13: The Metrics of the KNN Algorithm

The SVM, however, does a better job as it finds the best margin hyperplane between supervised data sets with different labels. Besides, the linear SVM has less chance of overfitting, and hence does a better job of generalising to the test data.

Figure 14: The Metrics of the SVM Algorithm

****