**Predicting handwritten digits using Neural Networks**

Ishank Pahwa

Implementing a basic (fully connected) neural network for multiclass classification of the provided images of handwritten digits.

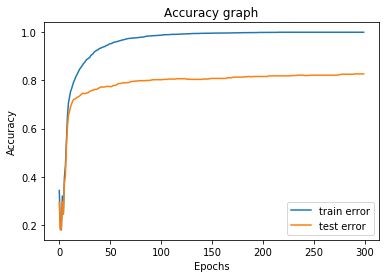
# Varying hyperparameters and modelling choices

1. Activation Function
2. Number of hidden layers
3. Number of units in hidden layer

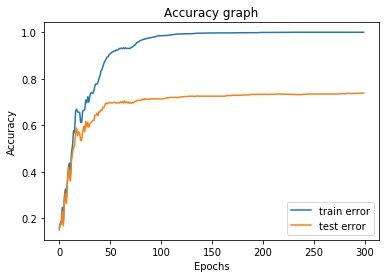
A. Sigmoid Activation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Activation Function Used | Number of  Neurons in Layer 1 | Number of  Neurons in Layer 2 | Training  Accuracy | Testing  Accuracy |
| Sigmoid  (1 Layer) | 300 | None | 99.95 | 82.6 |
| 400 | None | 99.91 | 83.2 |
| 500 | None | 99.91 | 82.66 |
| Sigmoid  (2 Layers) | 300 | 200 | 99.95 | 72.5 |
| 400 | 200 | 100 | 73.9 |
| 500 | 200 | 99.95 | 73.8 |

Plots for 1 layer network (400 neurons)



Plots for 2 layer network (400 neurons)



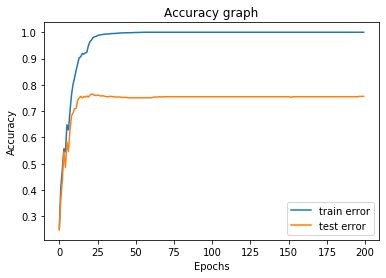
Inference

1. In this we have not used regularization and hence have overfitted the data
2. Time required for 2 layer network is greater than 1 layer network
3. As we increase the number of neurons, time increases as the number of weights increases and it takes more time to update them
4. It can be seen that if we choose number of neurons as 400 we get minimum testing error.

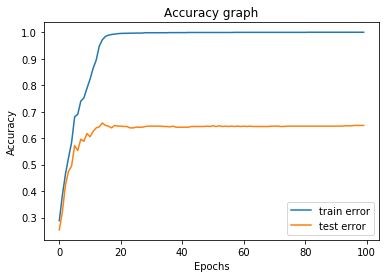
B. Tanh Activation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Activation Function Used | Number of  Neurons in Layer 1 | Number of  Neurons in Layer 2 | Training  Accuracy | Testing  Accuracy |
| Sigmoid  (1 Layer) | 300 | None | 100 | 74 |
| 400 | None | 100 | 74.2 |
| 500 | None | 100 | 75.6 |
| Sigmoid  (2 Layers) | 300 | 200 | 100 | 64.8 |
| 400 | 200 | 100 | 64.5 |
| 500 | 200 | 100 | 62 |

Plots for 1 layer network (400 neurons)



Plots for 2 layer network (400 neurons)



Inference

1. tanh overfits the data as the training accuracy goes to 100% but the testing accuracy remains less than what was obtained using Sigmoid activation function

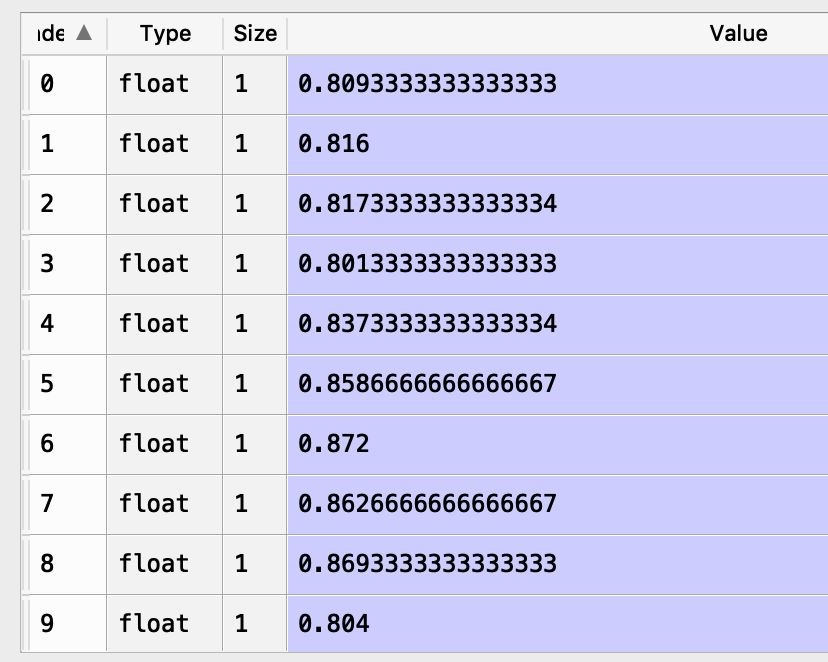
# Regularization

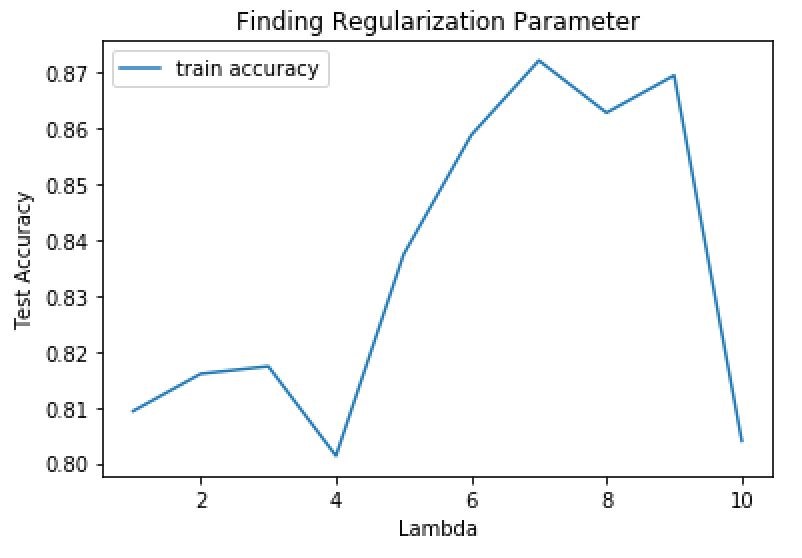
Finding optimal value of lambda to maximise testing accuracy using k fold cross validation with

1. Alpha as 0.0007
2. Hidden layer=1
3. Number of neurons=400
4. Max epochs - 200
5. k=4
6. Range of lambda - (1-10)

Optimal value comes as 7

Training accuracy at optimal value 87.2





Best Training accuracy came out to be 87.2% for lambda value 7.

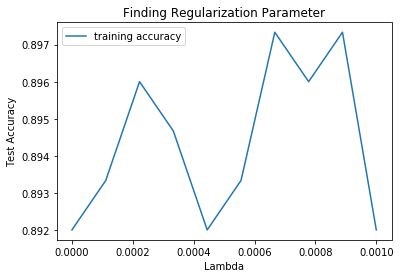
# Using Standard Library - Keras

1. Cross entropy error
2. L2 regularization

Model does not perform well using the parameter calculated earlier. Its accuracy comes out as 16%

It may be because of different equation used in the weights updation step in our model However when we find the hyper-parameter using keras we get the value of lambda as 0.00067

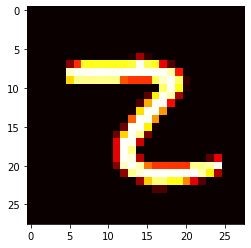
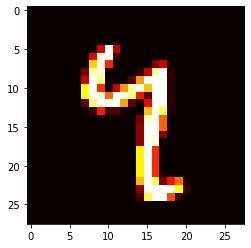
Training accuracy comes out as 89.7%



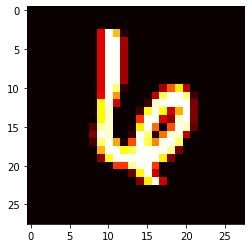
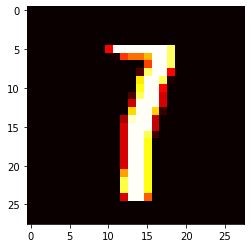
# Misclassified Images

After training the model, got test accuracy as 86.5

Visualizing some of the mis-classified points



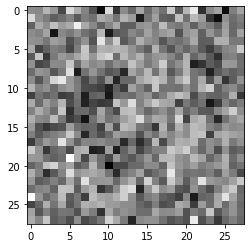
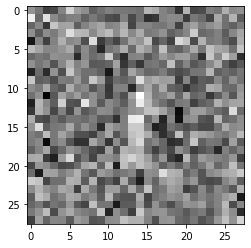
Value=4 Predicted=7 Value=2 Predicted=7



Value=7 Predicted=1 Value=6 Predicted =4

# Hidden Layer Representation

Representations of weights learned



Target=1 Target=3

# Comparison with PCA features-

Using no hidden layer with the PCA features we get

1. Training accuracy of 89.5%
2. Testing accuracy of 84.6%

If we use neural network with the following architecture 784->25->10 we get

1. Training accuracy of 93.6%
2. Testing accuracy of 86.4%

Inference-

Both of these are almost same and comparable. Since the variation captured is less than the original, hence the accuracy with PCA features is slightly less.

Now we add a hidden layer to the PCA features

Now the architecture becomes 25->20->10 and we get

1. Training accuracy of 90.4%
2. Testing accuracy of 86.1%

Inference-

Adding a hidden layer increases both the training as well as testing accuracy

# Advanced neural networks

Used Keras library to implement Convolutional neural network

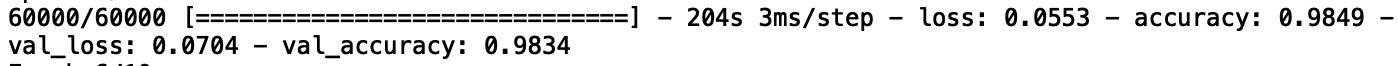
Used the dataset from mnist

Varied the following parameters - (Dropout parameter used=0.2)

1. Number of neurons
2. Kernel size
3. Strides
4. Pool size
5. Dropout Parameter

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of neurons | Kernel Size | Strides | Pool Size | Test Accuracy |
| 400 | 5 | 1 | 2 | 98.34% |
| 400 | 5 | 2 | 2 | 98.17% |
| 400 | 3 | 2 | 3 | 96.7% |
| 400 | 3 | 1 | 2 | 98.32% |

For kernel size=5 strides=1 pool size=2



Varying the dropout parameter in range 0.1-0.5 kernel\_size=5 Strides=2 pool\_size=2

|  |  |  |
| --- | --- | --- |
| Dropout parameter | Training Accuracy | Test Accuracy |
| 0.1 | 98.8 | 98.1 |
| 0.2 | 98.48 | 97.96 |
| 0.3 | 98.28 | 98.26 |
| 0.4 | 98.26 | 98.09 |
| 0.5 | 98.21 | 97.7 |

Hence the optimal value of the hyper-parameter is 0.3

## Comparing CNN with the neural network implemented earlier

Using the neural network from keras on this dataset we get

1. Training accuracy of 98.7
2. Testing accuracy of 97.3

Inference-

CNN gives better accuracy than ANN as the convolutional layers help in better learning of the image