COGS 260: Assignment 3

Aditya Verma
A53219148
Department of Electrical and Computer Engineering
University of California, San Diego
arv018@ucsd.edu

Abstract

In this assignment, we start with the basics of neural network, ie perceptron learning, and go to the advanced techniques in CNN based image classification, in which we experiment with different optimizer, normalization techniques, etc.

1 Perceptron Learning

1.1 Linear seperability of Iris dataset

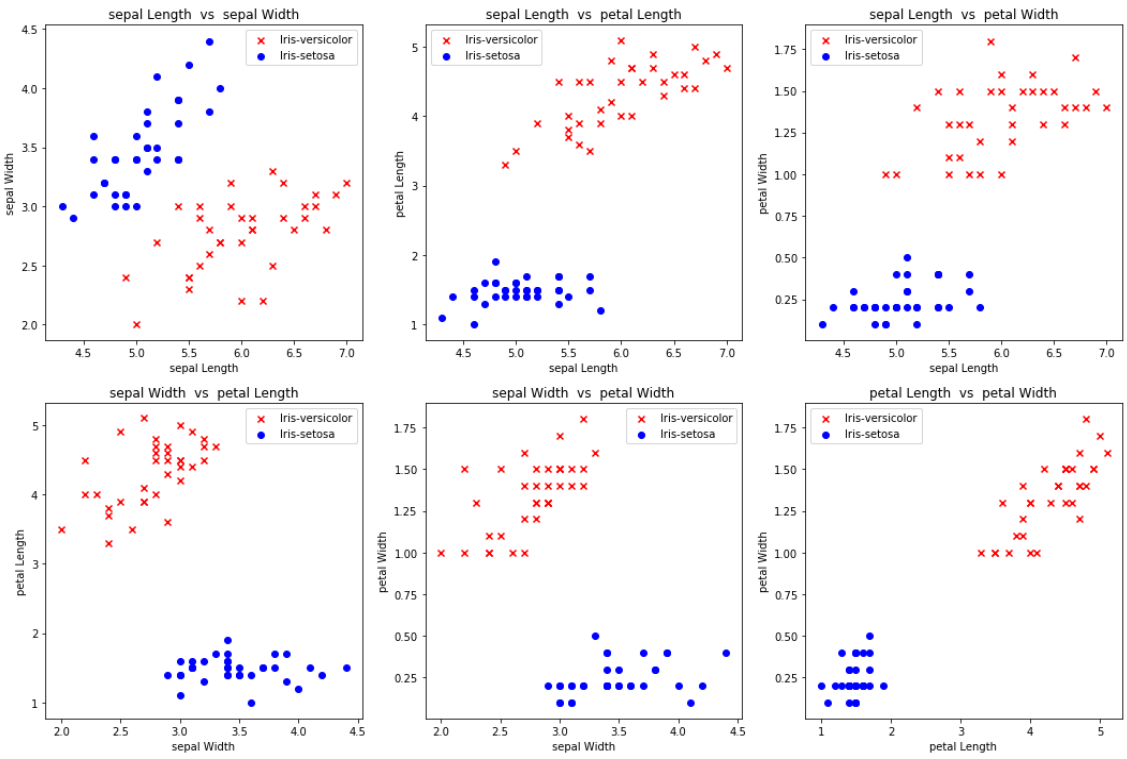

Figure 1: Feature pairwise scatter plot of Iris dataset

From all the plots above, we can see that the two classes are linearly seperable as we can easily draw a line seperating the two classes.

1.2 Perceptron Learning

The above network was implemented in tensorflow like a neural network architecture. Adam optimizer was used to train the network. The network converged to 100% in 2 epocs. This is due to the fact that the two classes are linearly seperable as seen in the above plots. With the above mentioned setting we achieve the below mentioned results:

- Train accuracy: 100%
- Number of Epocs: 2

1.3 Perceptron learning after z-scoring

The data was z-scored and the network was re-trained. Now the network converged pretty quickly in 1 epoc only

2 Feedforward Neural Network

2.1 Network architecture and hyperparameters

For the MNIST classsification with feedforward neural network we split the dataset as follows:

- Train size: 50000
- Test size: 10000

2.2 Experiment & Results

2.2.1 Using 1 hidden layer

- Hidden layers = 1
- Input units = 784
- Hidden units = 2
- Output units = 10
- Activation Func. = relu
- Learning rate = 0.01
- Batch size = 128

Performance

- Training Accuracy : 91.49%
- Test Accuracy : 91.79%

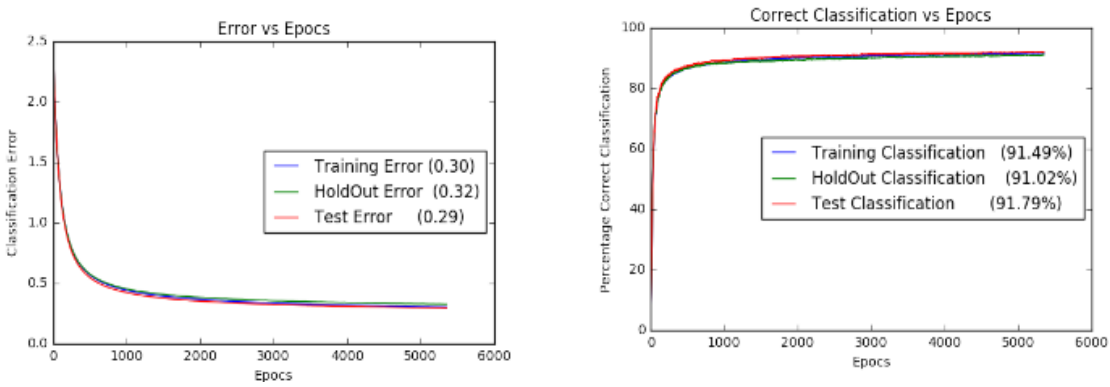

Figure 2: Feedforward neural network with 1 hidden layer

2.2.2 Using 2 hidden layer

- Hidden layers = 2
- Input units = 784
- Hidden layer1 units = 256
- Hidden layer2 units = 128
- Output units = 10
- Activation Func. = relu
- Learning rate = 0.01
- Batch size = 128

Performance

- Training Accuracy : 97.54%
- Test Accuracy : 96.50%

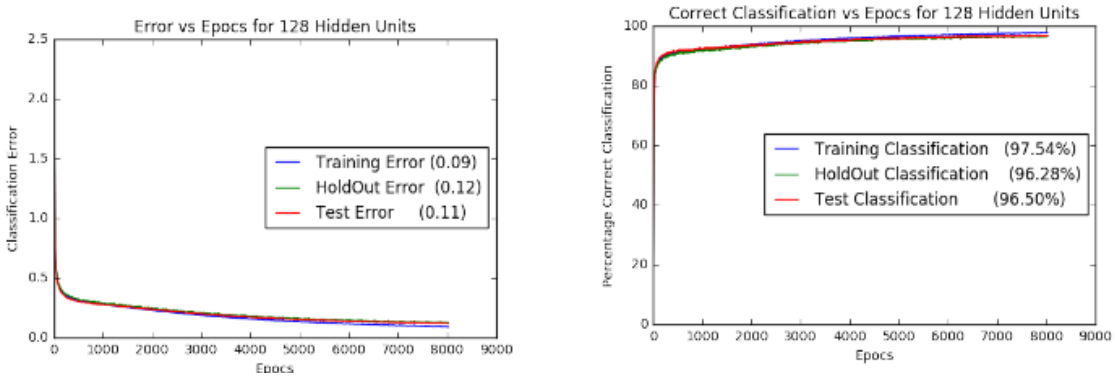

Figure 3: Feedforward neural network with 2 hidden layers

2.2.3 Using 1 hidden layer with regularization and momentum

- Momentum = 0.9
- Regularization = L2 with $\lambda = 0.001$

As we can see the plot, training the 1-hidden layer network with regularizatioin and momentum converges the network faster than in the first plot and gives better accuracy.

Performance

- Training Accuracy : 95.78%
- Test Accuracy : 95.41%

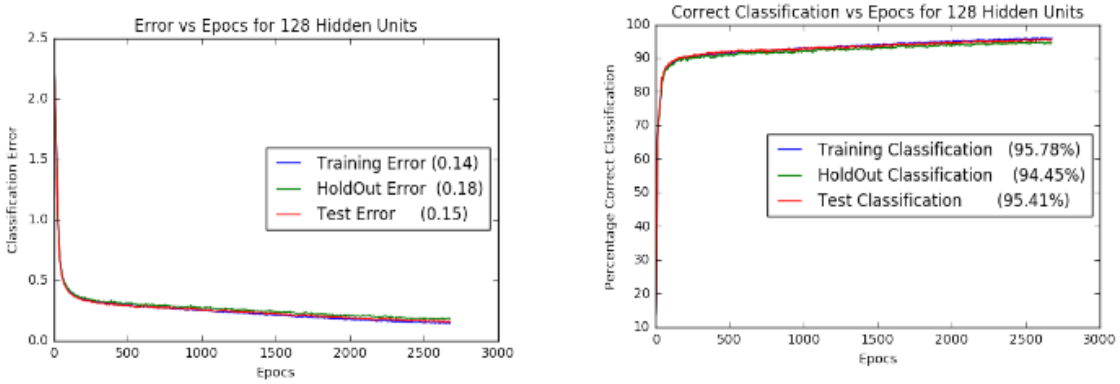

Figure 4: Accuracy and loss for training feedforward neural network with regularization and momentum

3 Convolutional Neural Network

3.1 Network Architecture

The network has been implemented in TensorFlow with the architecture shown in the table below. We experiment with different architecture/optimization techniques and compare the performance across all those techniques.

Table 1: Convolutional Neural Network Architecture

Layer Name	Layer Type	Activation function	Filter/ units	Filter size, stride and padding
Input	Input Layer			
Conv1	Convolution	Relu	32	3x3,1,SAME
Pool1	Max pooling			2x2,2,0
Conv2	Convolution	Relu	64	3x3,1,SAME
Pool2	Max pooling			2x2,2,0
Conv3	Convolution	Relu	128	3x3,1,SAME
Pool3	Max pooling			2x2,2,0
FC1	Fully Connected	Relu	256	
FC2	Fully Connected	Relu	128	
FC3	Fully Connected	Softmax	10	

The above network was implemented in Tensorflow, using some of the in-built libraries and default configuration.

3.2 Stochastic Gradient Descent

- Train Accuracy: 10%

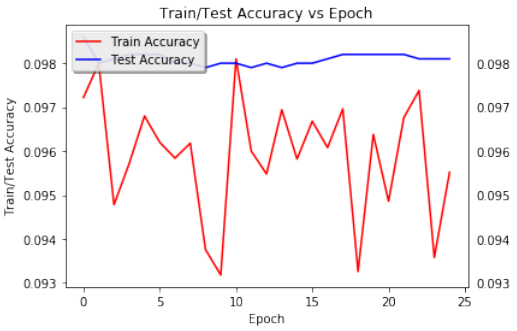

Figure 5: Accuracy plot for SGD learning

Figure 6: Loss plot for SGD learning

From the plot and values mentioned above, it can be seen that the learning of network is very slow. I had experimented with different combinations of learning rate, decay and annealing. Training the network to achieve similar accuracy as other optimizer, would require a lot of training time. However the accuracy of network can be measured by other optimization techniques employed below.

3.3 RMSprop

- Train Accuracy: 70%
- Test Accuracy: 68%

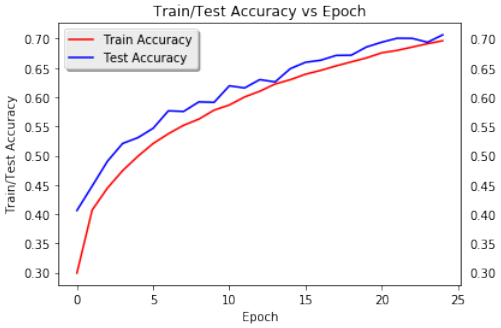

Figure 7: Accuracy plot for RMSprop learning

Figure 8: Loss plot for RMSprop learning

3.4 Batch normalization with RMSprop optimizer

- Train Accuracy: 60%
- Test Accuracy: 59%

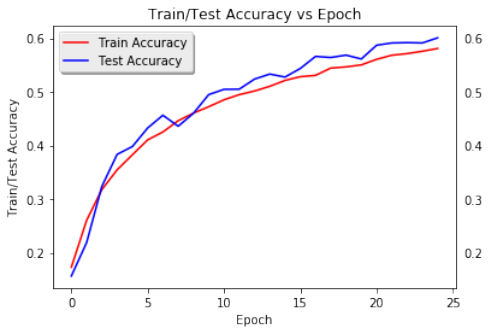

Figure 9: Accuracy plot for Batch normalized RMSprop optimizer

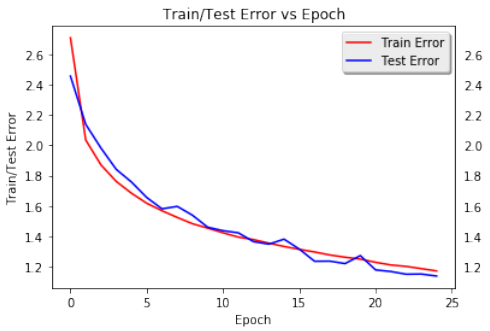

Figure 10: Loss plot for Batch normalized RMSprop optimized network

Ideally we expect that with batch normalization, we should achieve better accuracy and faster convergence. However for this particular dataset, we observe that we are not able to achieve the best possible results with Batch normalization.

3.5 AdaGrad

- Train Accuracy: 69.2%
- Test Accuracy: 68.3%

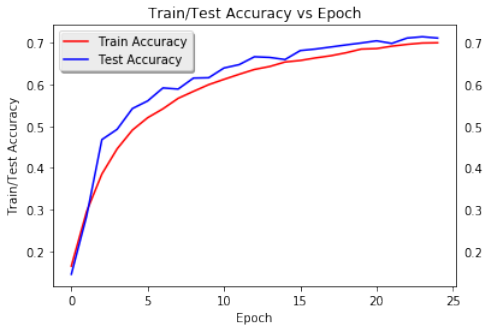

Figure 11: Accuracy plot for AdaGrad optimizer

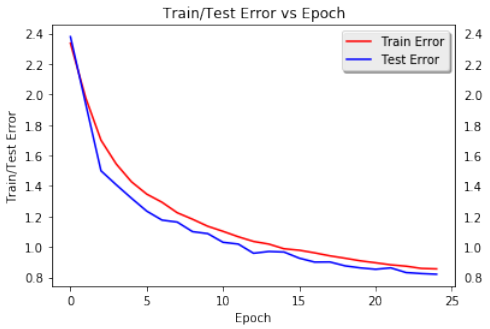

Figure 12: Loss plot for AdaGrad optimizer

3.6 Nesterov Acclerated gradient

- Train Accuracy: 80%
- Test Accuracy: 76%

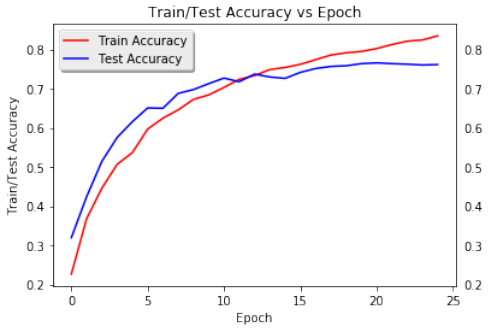

Figure 13: Accuracy plot for Nesterov Acclerated gradient learning

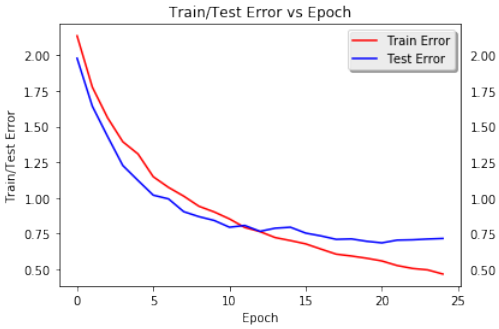

Figure 14: Loss plot for Nesterov Acclerated gradient learning

3.7 Global Average pooling

- Train Accuracy: 54%
- Test Accuracy: 52%

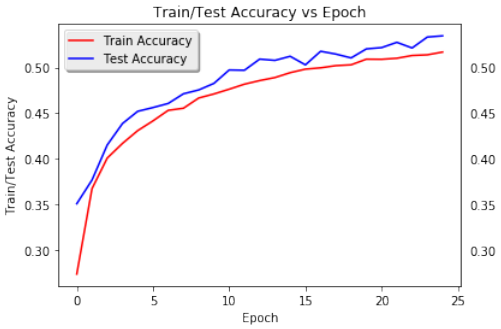

Figure 15: Accuracy plot for Globalaverage pooling network

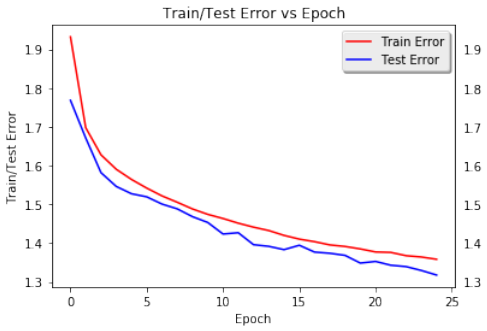

Figure 16: Loss plot for global average pooling network

As we can see, for network with global average pooling the accuracy decreases compared to the network with Fully-connected layers.

4 Source code

All the codes for this assignment are avaialbe at HW3 Solution

To accomplish this assignment, I have used several Scikit in-built functionalities, Tensorflow libraries, and open source repositories.

The reference to every open source library is given in the github repository at the link given above