|  |
| --- |
| **COGS 260: Assignment 3** |

**Apoorve Dave**

**A53103070**

**a1dave@ucsd.edu**

**Abstract**

A report on Homework 3 of COGS Image Recognition Assignment.

**1 Perceptron Learning**

* 1. **Linear Separability**

As seen from below plots, across all pairs of dimensions, the two classes are linearly separable.

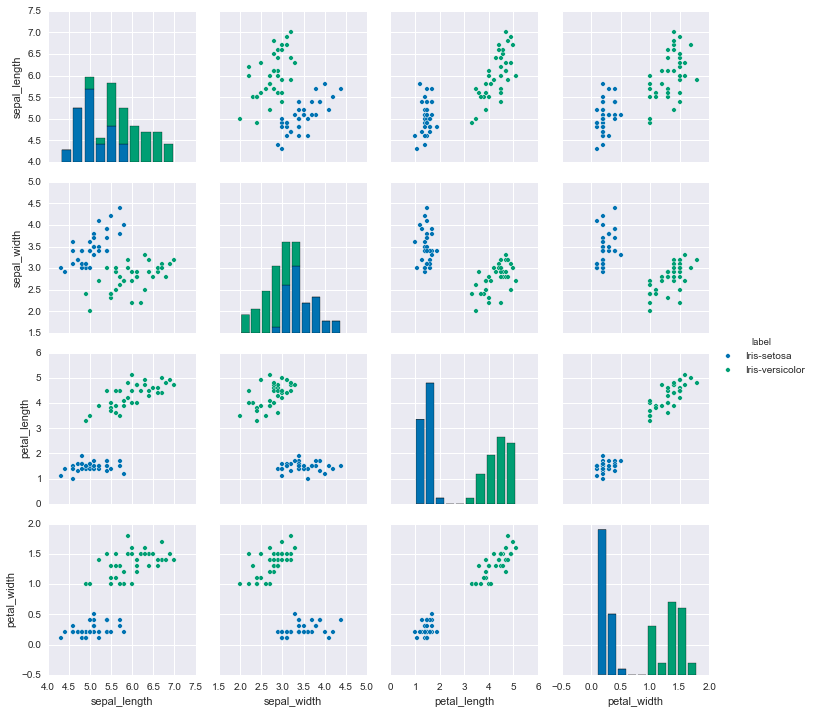
****

Figure 1: Linear separability

* 1. **Perceptron Learning without Z-scoring**

Without Z-scoring the data, it takes slightly more time(3 iterations over data set) for convergence.

Learning rate

Number of iterations = 3

Accuracy on test set = 100%

* 1. **Perceptron Learning with Z-scoring**

Observation: With Z-scoring, it takes just one iteration over data set to reach the 100% accuracy on test set.

Learning rate

Number of iterations = 1

Accuracy on test set = 100%

1. **Feed Forward Neural Network**
   1. **NN with 1 hidden layer**
      1. **Hyperparameters:**

**Architecture:** 784 -> 30 -> 10

**Learning Rate:** 3/#samples

**#Epochs:** 30

**Plots:**



* 1. **NN with 2 hidden layers**

Performance was slow as the network complexity increased. Higher accuracy on training data as well as test data was observed because of network learning more complex features.

**Architecture:** 784 -> 30 -> 30 -> 10

**Learning Rate:** 3/#samples

**#Epochs:** 30

**Plots:**



* 1. **Regularization and Momentum**
     1. **Regularization Only:**

As expected, the model performed better after regularization as it was able to generalize better.

**Architecture:** 784 -> 30 -> 10

**Learning Rate:** 0.1/#samples

**Regularization(lambda)**: 5/#samples

**#Epochs:** 30

**Plots:**



* + 1. **Momentum Only:**

Achieved faster convergence with momentum value of 0.9. Below are the figures for the same.

**Architecture:** 784 -> 30 -> 10

**Learning Rate:** 0.1/#samples

**Momentum(mu)**: 0.9

**#Epochs:** 30

**Plots:**



* + 1. **Momentum And Regularization:**

**Architecture:** 784 -> 30 -> 10

**Learning Rate:** 0.1/#samples

**Momentum(mu)**: 0.9

**Regularization(lambda)**: 5/#samples

**#Epochs:** 30

**Plots:**

****

1. **CNN**

**Base Network Architecture:**

Model Layers in order:

Spatial**Conv**olution(Filter size 5x5)

Spatial**MaxPool**ing(Filter size 3x3)

**ReLU**

Spatial**Conv**olution(Filter size 5x5)

**ReLU**

Spatial**MaxPool**ing(Filter size 3x3)

Spatial**Conv**olution(Filter size 5x5)

**ReLU**

Spatial**MaxPool**ing(Filter size 3x3)

Linear(64\*3\*3 -> 64)) – **(Fully Connected)**

Linear(64 -> 10)) - **(Fully Connected)**

**LogSoftMax**

* 1. **SGD**

It takes more than 20 iterations when we can see the flattening of test loss with #iterations. The best accuracy achieved with this method was 72%.

**Iterations for convergence**: 22+

**Best Test Accuracy:** 72%

**Plots:**

****

* 1. **Batch Normalization**

**Observations:** After batch normalization, the convergence has sped up much faster compared to SGD with a little lower accuracy value.

**Network:**

Spatial**Conv**olution

**SpatialBatchNormalization**

**ReLU**

Spatial**MaxPool**ing

Spatial**Conv**olution

**SpatialBatchNormalization**

**ReLU**

Spatial**MaxPool**ing

Spatial**Conv**olution

**SpatialBatchNormalization**

**ReLU**

Spatial**MaxPool**ing

**Fully Connected**

**BatchNormalization**

**Fully Connected**

**LogSoftMax**

#**Epochs for best Test Accuracy**: 9

#**Epochs for least Test Loss**: 8

**Best Test Accuracy**: 67%

**Plots:**



* 1. **Replace the fully connected layer by average pooling layer**

**Network:**

SpatialConvolution

ReLU

SpatialMaxPooling

SpatialConvolution

ReLU

SpatialMaxPooling

SpatialConvolution

ReLU

SpatialMaxPooling

**SpatialAveragePooling**

Linear

LogSoftMax

**Observations:** This gives a faster convergence wrt Adagrad/SGD but is much faster as there are much less weights to learn for the system. Better results than Adagrad are obtained.

**Iterations for convergence:** 30

**Best Test Accuracy:** 72.5%

**Plots:**

****

* 1. **Adaptive Gradient**
     1. **Without Batch Normalization:**

For this case Adagrad was run with and without batch normalization. Without normalization, the test loss did not converge even after 50 iterations where I had to stop it because of time constraints.

**Iterations for convergence:** More than 50

**Test Accuracy in 50 iterations:** 63.6%

**Plots:**



* + 1. **With Batch Normalization**

Faster convergence achieved but with a little less accuracy as compared to adagrad without batch normalization.

#**Epochs for best Test Accuracy**: 32

#**Epochs for least Test Loss**: 34

**Best Test Accuracy**: 63.5%

**Plots**:



* 1. **Nesterovs Accelerated Gradient**

Note that this was done with batch normalization already in place.

From the plot below we can see that NAG converges much faster for test error. The best accuracy on test set (=68.2%) is achieved just after 3 Epochs

#**Epochs for best Test Accuracy**: 7

#**Epochs for least Test Loss**: 3

**Best Test Accuracy**: 68.2%

**Plots**:



* 1. **RMSprop**

Note that this was done with batch normalization already in place.

Results similar to NAG were achieved with even better #epochs for convergence (4 vs 7)

#**Epochs for best Test Accuracy**: 4

#**Epochs for least Test Loss**: 3

**Best Test Accuracy**: 68.6%

**Plots**:



**Reference:**

1. [**http://neuralnetworksanddeeplearning.com/**](http://neuralnetworksanddeeplearning.com/)
2. [**https://github.com/gcr/torch-residual-networks/blob/master/train-cifar.lua**](https://github.com/gcr/torch-residual-networks/blob/master/train-cifar.lua)

**Codes:**

1. **[Github Repository](https://github.com/apoorvedave/ImageClassification-Project3)**