COGS 260: Assignment 3

Saurabh Gupta
A53091070
sag043@ucsd.edu

Abstract

In this assignment, we work on three generations of Neural Networks: Perceptron learning, Feed-forward networks and Convolutional neural networks.

1 Perceptron Learning

1.1 Linear Separability of Iris dataset

As visible from the image below, the dataset is clearly linearly separable since a straight line can separate the two classes in each of the 2D feature-pair space.

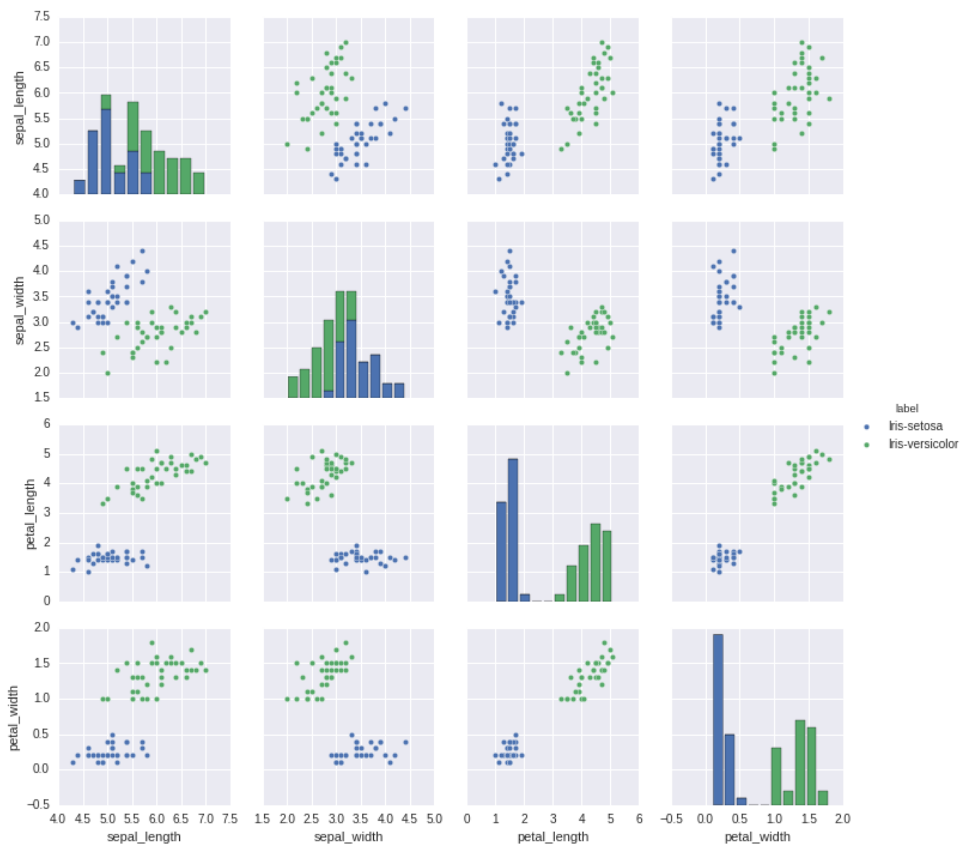

Figure 1: Scatter plot of Iris dataset features

1.2 Perceptron training

The perceptron was trained with learning rate = 1. It converged in 3 epochs i.e. gave training and test accuracy of 100% since the data is completely linearly separable. An epoch refers to a full pass through all the training examples.

A bias term was added to the weights vector which was then initialized by generating a random number array with each number between [-1, 1].

1.3 Perceptron training after Z-scoring

Z-scoring the data speeds up the learning significantly. After Z-scoring, convergence was achieved in just 1 epoch.

2 Feed Forward neural network

2.1 Image Data Preprocessing and Weights Initialization

All the images were Z-scored by subtracting the mean from every feature and then dividing by the standard deviation in the respective dimension.

Weights were drawn from a Gaussian distribution with standard deviation of $\sqrt{2/n}$, where n is the number of inputs to the neuron. Eg. w = np.random.randn(n) * sqrt(2.0/n)

2.2 Annealing the learning rate

The learning rate was reduced by a factor of .95 every epoch after 10 epochs. Before 10 epochs, it was kept constant.

2.3 Experiments and results

2.3.1 Using 1 hidden layer

Parameters and training details:

Hidden layers = 1
Hidden units = 100
Neurons in input, output layers = 784, 10
Activation function = Sigmoid
Learning rate = 0.1
Max Epochs = 30
Batch size = 50

Performance

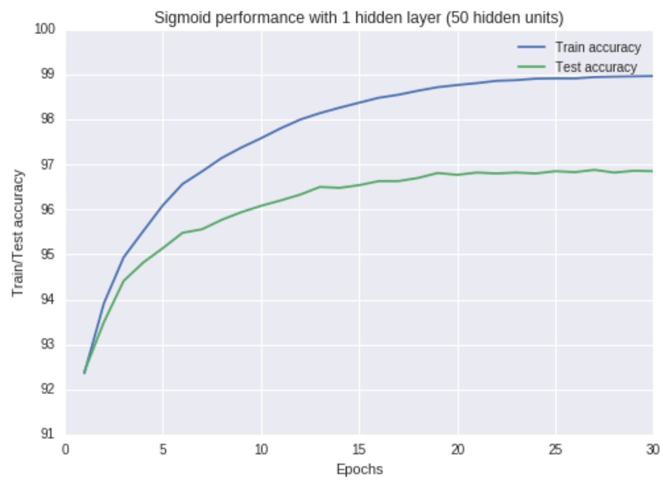

Figure 2: Train/Test accuracy

Running time: 310 seconds
Train accuracy: 98.96%
Test accuracy: 96.85%

2.3.2 Using 2 hidden layers

Parameters and training details:

Hidden layers = 2
Hidden units in 1st hidden layer = 50
Hidden units in 2nd hidden layer = 50
Neurons in input, output layers = 784, 10
Activation function = Sigmoid
Learning rate = 0.1
Max Epochs = 30
Batch size = 50

Performance

Figure 3: Train/Test accuracy

Running time: 368 seconds
Train accuracy: 99.04 %
Test accuracy: 96.24%

The network with 2 hidden layers has less accuracy than the network with 1 hidden layer (96.85%). This might be because the network is over-fitting (due to increased complexity). A lower learning rate might give better results.

2.3.3 Using 1 hidden layer with momentum

Parameters and training details:

Hidden layers = 1
Hidden units in 1st hidden layer = 50
Neurons in input, output layers = 784, 10
Activation function = Sigmoid
Learning rate = 0.1
Max Epochs = 30
Batch size = 50
Momentum = 0.9
Regularization = None

Performance

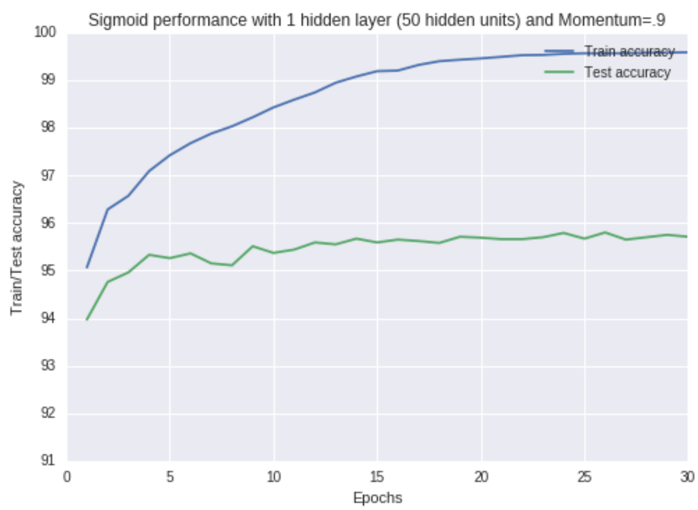

Figure 4: Train/Test accuracy

Running time: 290 seconds
Train accuracy: 99.58 %
Test accuracy: 95.7 %

Using momentum did speed up the learning as the training accuracy reached 99.6% in just 290 seconds. However, from the above plot, it is safe to conclude that the network is overfitting. So, the next step is to try using regularization.

2.3.4 Using 1 hidden layer with momentum and regularization

Parameters and training details:

Hidden layers = 1
Hidden units in 1st hidden layer = 50
Neurons in input, output layers = 784, 10
Activation function = Sigmoid
Learning rate = 0.1
Max Epochs = 30
Batch size = 50
Momentum = 0.9
Regularization = L2 with lambda = .0001

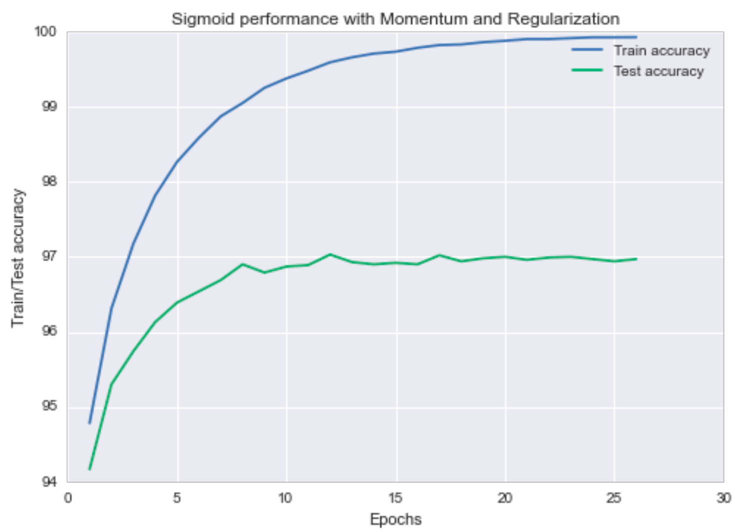

Figure 5: Train/Test accuracy

Running time: 299 seconds
Train accuracy: 99.92 %
Test accuracy: 96.98 %

As expected, momentum with regularization gives the highest test accuracy of 97%. Also, as clear from the graph, convergence was achieved pretty quickly – in just 8 epochs.

3 Convolutional Neural Network

3.1 Network architecture

The network architecture chosen for this problem can be summarized as:

[conv-relu-conv-relu-pool] x 2 → [FC] x 2 → [Softmax output (10 classes)]

Following table describes the network architecture in detail:

Table 1: Network architecture

Layer Name	Layer Type	Activation function	Filters/Units	Filter size, stride and padding	Output Size
Input	Input layer				3x32x32
Conv1	Covolution	Relu	64	3x3, 1, 1	64x32x32
Conv2	Covolution	Relu	64	3x3, 1, 1	64x32x32
Pool1	Max Pooling			2x2, 2, 0	64x16x16
Conv3	Covolution	Relu	128	3x3, 1, 1	128x16x16
Conv4	Covolution	Relu	128	3x3, 1, 1	128x16x16
Pool2	Max Pooling			2x2, 2, 0	128x8x8
FC1	Fully Connected	Relu	256		256
FC2	Fully Connected	Relu	256		256
FC3	Fully Connected	Softmax	10		10

The above network was modeled and trained using Lasagne (Theano) using different optimization techniques that are discussed in the following sections.

Lasagne defaults were used wherever the parameters pertaining to a specific optimization algorithm are not mentioned.

3.2 Stochastic Gradient Descent

Parameters and training details:

learning_rate = .01
learning_rate_decay=0.5
decay_after_epochs=10
batch_size=128
num_epochs=25

Regularization: Dropout (p = 0.5) layers were used before feeding inputs to all the fully connected layers (FC1, FC2 and FC3).

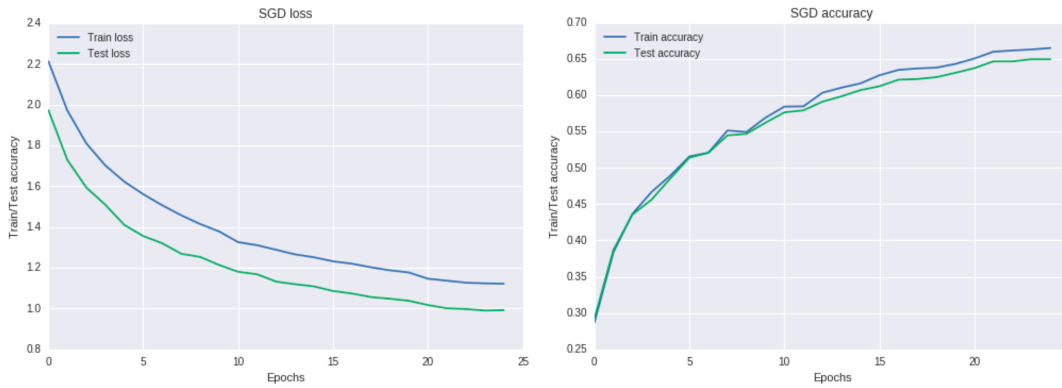

Figure 6: Performance curves for SGD

Running time: ~50 seconds per epoch
Train accuracy: 66.5 %
Test accuracy: 65.0 %

3.3 RMSprop

Parameters and training details:

Initial learning_rate = .01
batch_size=128
num_epochs=25

Regularization: Dropout (p = 0.5) layers were used before feeding inputs to all the fully connected layers (FC1, FC2 and FC3).

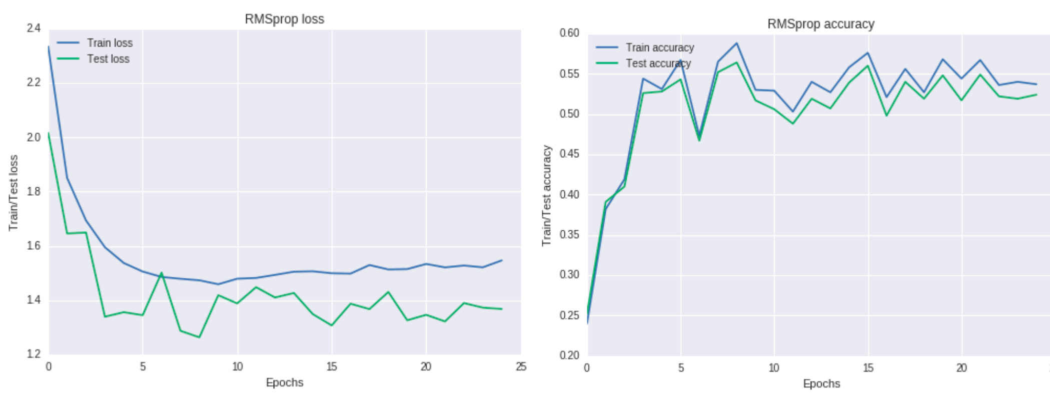

Figure 7: Performance curves for RMSprop

Running time: ~55 seconds per epoch
Train accuracy: 53.7 %
Test accuracy: 52.4 %

RMSprop doesn't perform as well as SGD. It reached test accuracy of just 53.7% and the learning appears to be extremely noisy (may be because of large changes in learning rate).

3.4 ADAgrad

Parameters and training details:

Initial learning_rate = .01
batch_size=128
num_epochs=25

Regularization: Dropout (p = 0.5) layers were used before feeding inputs to all the fully connected layers (FC1, FC2 and FC3).

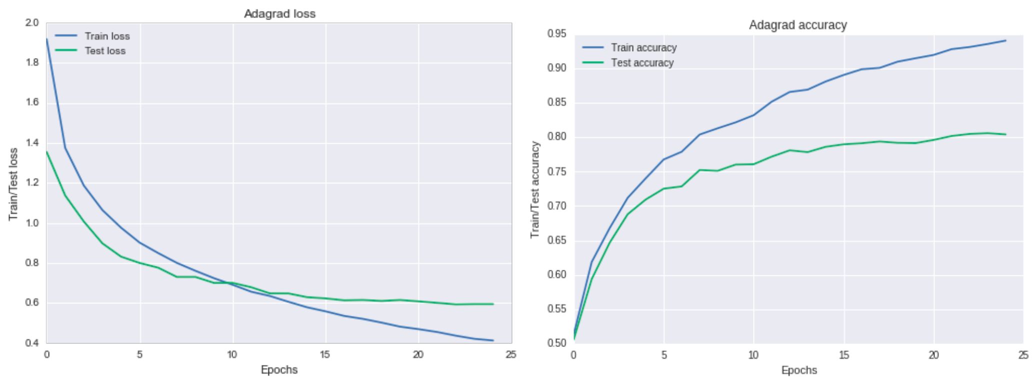

Figure 8: Performance curves for Adagrad

Running time: ~51 seconds per epoch
Train accuracy: 94.1 %
Test accuracy: 80.4 %

Adagrad gives really good performance as compared to SGD and RMRprop. From the loss curve, it is pretty clear that test loss is saturated and so the model has converged.

3.5 Nesterov Accelerated Gradient (NAG)

Parameters and training details:

learning_rate = .01
learning_rate_decay=0.5
decay_after_epochs=10
momentum=0.9
momentum_decay=0.5
batch_size=128
num_epochs=25

Regularization: Dropout (p = 0.5) layers were used before feeding inputs to all the fully connected layers (FC1, FC2 and FC3).

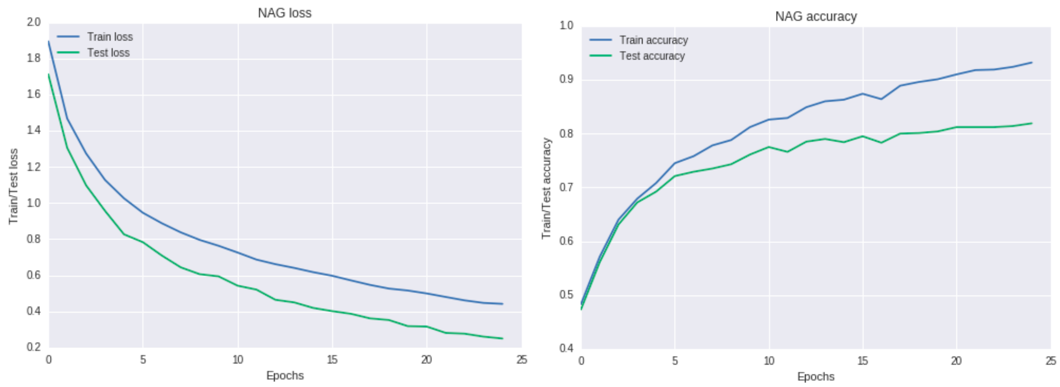

Figure 9: Performance curves for NAG

Running time: ~59 seconds per epoch
Train accuracy: 93.2 %
Test accuracy: 81.9 %

NAG gives the best performance among all the optimization algorithms. From the performance curves, it appears that training has still not converged and the algorithm could still do better if it is run for more iterations.

3.6 Nesterov Accelerated Gradient with Batch Normalization (NAG)

Batch normalization layers were inserted between every Convolutional layer and its non-linearity i.e. Relu layer. All the default parameters (provided by Lasagne) were used.

Parameters and training details:

learning_rate = .01
learning_rate_decay=0.5
decay_after_epochs=10
momentum=0.9
momentum_decay=0.5
batch_size=128
num_epochs=35

Regularization: Dropout (p = 0.5) layers were used before feeding inputs to all the fully connected layers (FC1, FC2 and FC3).

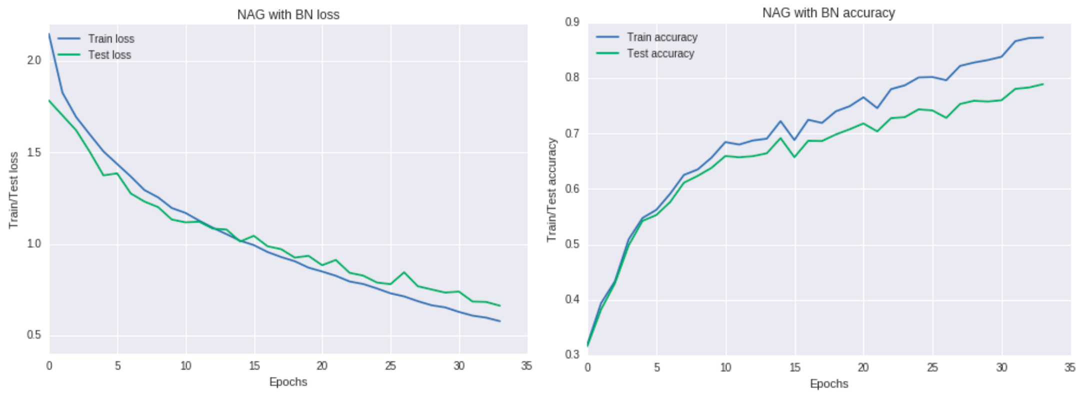

Figure 10: Performance curves for NAG with BN

Running time: ~65 seconds per epoch
Train accuracy: 87.3 %
Test accuracy: 78.9 %

Although it produces less test accuracy than NAG alone, the loss graph clearly indicates that the model could perform better if it is run for more epochs. Introduction of batch normalization layer requires more parameters to be trained and hence training requires more time but can give better performance.

3.8 Average Pooling Layer instead of fully connected (with NAG)

FC1 and FC2 layers of the original model were replaced by an average pooling layer of poolsize 8x8. So, the pooling layer takes an input volume of 128x8x8 and outputs a volume of 128x1x1. This output is then used by the fully connected softmax output layer.

Parameters and training details:

learning_rate = .01
learning_rate_decay=0.5
decay_after_epochs=10
momentum=0.9
momentum_decay=0.5
batch_size=128
num_epochs=25

Regularization: Dropout (p = 0.5) layer was used before feeding inputs to FC3.

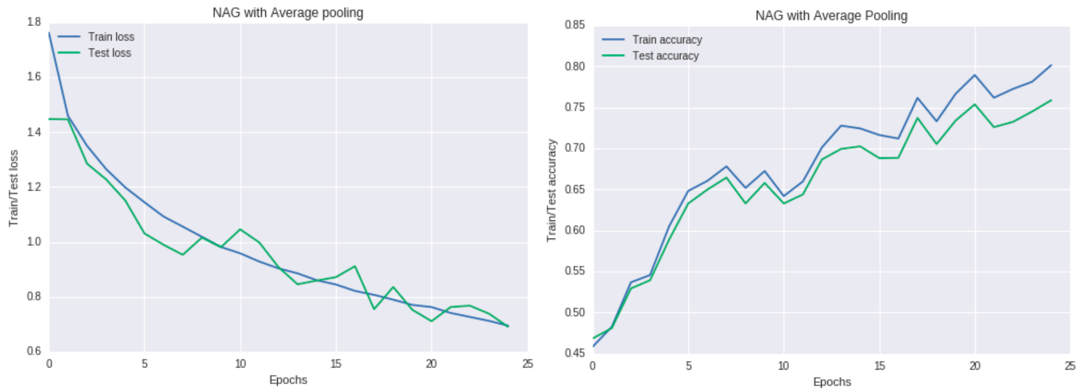

Figure 11: Performance curves for NAG with FC replaced by Average Pooling

Running time: ~63 seconds per epoch
Train accuracy: 80.1 %
Test accuracy: 75.9 %

Replacing the fully connected layers with average pooling simplifies the network while almost maintaining its representational power. As clear from the loss curve, the network can give higher accuracy if it is run for more epochs.

References

Lasagne Documentation (http://lasagne.readthedocs.io/)

NeuralNetworksAndDeepLearning.com (By Michael Nielsen)

Neural Networks (CS 231N), Andrej Karpathy

Appendix

All the codes used for this assignment are available at this GitHub repository – https://github.com/saurabh3949/UCSD-COGS-260