CS89 Milestone Report

Michael Downs

May 15, 2015

1 Proposed Goals

In my initial project proposal I stated the following as my milestone goals:

- Re-acquaint myself with Python's syntax and familiarize myself with the Theano framework
- Write code to handle the MNIST data, taking care of obtaining the data and performing any preprocessing
- Write code for a full neural network
- Write code for one to two of the autoencoder variants

2 Accomplishments

What I actually accomplished:

- Obtained a functioning understanding of Theano
- Wrote code to download the MNIST data set and store its parameters into Theano shared variables
- Wrote code for a fully connected neural network using Theano. My code has the following features:
    1. Adjustable network architecture
    2. Adjustable activation can be one of sigmoid, tanh, softplus, or rectified linear
    3. Adjustable cost function can be one of likelihood or entropy
    4. Adjustable batch size, learn rate, number of epochs, weight decay parameter, and momentum parameter
    5. Is able to tile weights in first layer and display them as an image or animation

  Initial experimentation with shallow networks suggests that a cross entropy learning objective with ReLu activations, batch sizes of 20, learning rate of 0.01, weight decay parameter of 0.0001, and momentum parameter of 0.2 work well. I was able to achieve around 1.6% accuracy on the test set which at the very least suggests that my script works.

3 Future Work

After considering feedback on the project proposal and milestone presentation I have decided to push experimentation on sparse autoencoders to the end of my priority list since they have been shown to yield superior results when the hidden layer in the autoencoder contains more nodes than the input layer. The premise for my experiments was to pre-train the networks using different autoencoders while keeping everything else constant. The aforementioned observation suggests that sparse autoencoders would yield inferior performance. I will also attempt to optimize the final classification error for different autoencoders using different network parameters after comparing their effect on final performance over a single network architecture.

The following still has to be done:

- Currently the NeuralNetwork class only allows for random initialization of network parameters. I have to allow for passing in weights as instantiation parameters for the later network stacking
- Code up the autoencoder variants using the NeuralNetwork class as a base
- Run experiments

4 Code

I used the examples at http://deeplearning.net/tutorial/ as a guide for writing my code. The code I have written thus far is available at https://github.com/michaeldowns/COSC89