CS89 Project Proposal: An analysis of performance improvement in deep architectures via different pretraining schemes

Michael Downs

April 22, 2015

1 Motivation

Properly training deep neural networks is a difficult task. There is no closed form solution to fit a neural network so any learning objective must be optimized using an iterative approach such as gradient descent. In addition, due to the amount of data typically used to train a neural network as well as the large number of model parameters, performing the descent steps is computationally and time intensive, and not guaranteed to converge to a global extremum due to the highly non-convex nature of the learning objective. Finally, there are numerical concerns present in the backpropagation algorithm: the computed gradients can become too small and slow down the gradient descent steps.

There are several "learning aides" to improve training neural networks. Some of these might include using a different cost function such as the cross entropy cost function, using weight decay, or adding momentum to the gradient descent steps. These procedures modify the learning algorithm as it takes place – there are also ways to improve learning by better initializing the network parameters rather than drawing them from random distributions.

A popular way to "pre-train" a network is to initialize the network weights by sequentially training a series of autoencoders and then splicing them into a full network for further learning. For example, the first autoencoder is trained, then the data is encoded using this autoencoder, then another autoencoder is trained using the encoded data, and so on and so forth. This is called a stacked autoencoder. The effect of this seems to be an initialization of the weights in a region of the parameter space that yields a gradient descent trajectory with final learned parameters that better generalize to out of sample data.

Because there are a variety of different autoencoders types, there are several different ways to pretrain a network. A natural question based on this observation is, holding everything else constant, which autoencoders yield superior pretraining?

2 Methods

I intend to determine a suitable network architecture for each data set I run experiments on as well as an optimal training strategy using some combination

Figure 1: MNIST Data Set

of the aforementioned ”learning aides.” The weights for the initial model will be drawn from a random distribution. Then, keeping the architecture and learning procedure fixed throughout, I will initialize the weights using the different autoencoder types, fine-tune the model, and then record the classification error of the final result.

3 Data

I will be primarily running experiments using the MNIST data set. I have elected to use MNIST because it requires minimal preprocessing, has only one color channel (and therefore reasonable classification error can be achieved without convolutional neural networks), and is binary-valued. If time permits, I will conduct experiments on other data sets.

4 Tools

For this project I will use Theano. Theano offers several attractive features for working with neural networks. Variables in Theano are symbolic and any functions constructed from the symbolic variables are automatically optimized. Theano also allows for automatic differentiation meaning that, with relative ease, one can compute the numerical gradient of a function with respect to the input variables without having to derive an analytic expression, no matter how complicated the relationship between the function and the variables.

5 Technical Details

Autoencoders are three layer neural networks that learn lower dimensional representations of data by attempting to reconstruct the original input after compressing it down to lower dimensions, as seen in figure 2.

The autoencoders that I hope to analyze are:

- Autoencoder

  Regular autoencoder as seen in figure 2. Each input is given an associated label that is identical to the input.

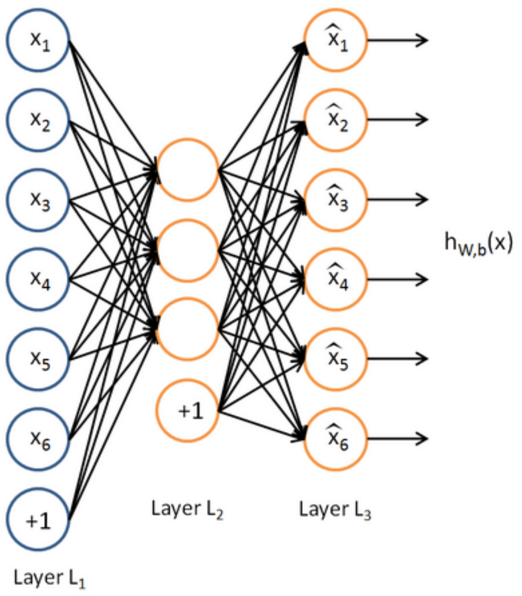

Figure 2: Autoencoder

- Sparse Autoencoder

  Adds a penalty term enforcing the activations in the hidden layer to be sparse. The new cost function is

  $$C_{Sparse}(W,b) = C(W,b) + \lambda \sum_{i=1}^{n_2} KL(\rho||\hat{\rho}_i) \tag{1}$$

  Where the additional penalty term is a sum over the KL divergence between the desired activation level and the mean activations in each hidden node.

- Denoising Autoencoder

  Denoising autoencoders have the typical autoencoder learning objective with the modification that the input data is stochastically corrupted during each gradient descent step. The goal is to train an autoencoder that is robust to small variations in the data.

- Contractive Autoencoder

  The contractive autoencoder aims to perform a similar task to the denoising autoencoder in that it trains weights that are robust to variations in the input by optimizing over the alternative learning objective:

  $$C_{Contractive}(W,b) = C(W,b) + \lambda||\nabla_{x^{(i)}} h(x^{(i)})||_F^2 \tag{2}$$

  This attempts to minimize the Jacobian of the hidden layer with respect to the input layer

- Restrictive Autoencoder

  This is a novel type of Autoencoder that I hope to explore during this project. The premise is taken from a paper co-authored by Nando de Freitas where the weight matrices are represented as products of two matrices that have an inner dimension much smaller than their respective outer dimensions. Holding one of these factors fixed and learning the other, we would be able to reconstruct the entire weight matrix after having learned only a few parameters.

6 Milestone Goals

By the milestone I hope to have working code to load and preprocess the data as well as working code for a full neural network and one to two of the autoencoders.

7 References

- http://www.cs.toronto.edu/ larocheh/publications/icml-2008-denoising-autoencoders.pdf
- http://www.icml-2011.org/papers/455_icmlpaper.pdf
- http://arxiv.org/pdf/1306.0543v2.pdf
- http://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf
- http://deeplearning.net/software/theano/tutorial/