Exercise: Supervised Neural Networks

In this exercise, you will train a neural network classifier to classify the 10 digits in the MNIST dataset. The output unit of your neural network is identical to the softmax regression function you created in the Softmax Regression (http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression) exercise. The softmax regression function alone did not fit the training set well, an example of **underfitting**. In comparison, a neural network has lower bias and should better fit the training set. In the section on Multi-Layer Neural Networks (http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks) we covered the backpropagation algorithm to compute gradients for all parameters in the network using the squared error loss function. For this exercise, we need the same cost function as used for softmax regression (cross entropy), instead of the squared error function.

The cost function is nearly identical to the softmax regression cost function. Note that instead of making predictions from the input data x the softmax function takes as input the final hidden layer of the network $h_{W,b}(x)$. The loss function is thus,

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp(\theta^{(k)\top} h_{W,b}(x^{(i)}))}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} h_{W,b}(x)^{(i)}))}\right].$$

The difference in cost function results in a different value for the error term at the output layer ($\delta^{(n_l)}$). For the cross entropy cost we have,

$$\delta^{(n_l)} = -\sum_{i=1}^{m}\left[\left(1\{y^{(i)} = k\} - P(y^{(i)} = k|x^{(i)};\theta)\right)\right]$$

Using this term, you should be able to derive the full backpropagation algorithm to compute gradients for all network parameters.

Using the starter code given, create a cost function which does forward propagation of your neural network, and computes gradients. As before, we will use the minFunc optimization package to do gradient-based optimization. Remember to numerically check your gradient computations! Your implementation should support training neural networks with multiple hidden layers. As you develop your code, follow this path of milestones:

- Implement and gradient check a single hidden layer network. When performing the gradient check, you may want to reduce the input dimensionality and number of examples by cropping the training data matrix. Similarly, when gradient checking you should use a small number of hidden units to reduce computation time.
- Gradient check your implementation with a two hidden layer network.
- Train and test various network architectures. You should be able to achieve 100% training set accuracy with a single hidden layer of 256 hidden units. Because the network has many parameters, there is a danger of overfitting. Experiment with layer size, number of hidden layers, and weight decay penalty to understand what types of architectures perform best. Can you find a network with multiple hidden layers which outperforms your best single hidden layer architecture?
- (Optional) Extend your code to support multiple choices for hidden unit nonlinearity (sigmoid, tanh, and rectified linear).

Supervised Learning and Optimization

Linear Regression (http://ufldl.stanford.edu/tutor

Logistic Regression (http://ufldl.stanford.edu/tutor

Vectorization (http://ufldl.stanford.edu/tutor

Debugging: Gradient Checking (http://ufldl.stanford.edu/tutor

Softmax Regression (http://ufldl.stanford.edu/tutor

Debugging: Bias and Variance (http://ufldl.stanford.edu/tutor

Debugging: Optimizers and Objectives (http://ufldl.stanford.edu/tutor

Supervised Neural Networks

Multi-Layer Neural Networks (http://ufldl.stanford.edu/tutor

Exercise: Supervised Neural Network (http://ufldl.stanford.edu/tutor

Supervised Convolutional Neural Network

Feature Extraction Using Convolution (http://ufldl.stanford.edu/tutor

Pooling (http://ufldl.stanford.edu/tutor

Exercise: Convolution and Pooling (http://ufldl.stanford.edu/tutor

Optimization: Stochastic Gradient Descent (http://ufldl.stanford.edu/tutor

Convolutional Neural Network (http://ufldl.stanford.edu/tutor

Excercise: Convolutional Neural Network

(http://ufldl.stanford.edu/tutor

Unsupervised Learning

Autoencoders (http://ufldl.stanford.edu/tutor

PCA Whitening (http://ufldl.stanford.edu/tutor

Exercise: PCA Whitening (http://ufldl.stanford.edu/tutor

Sparse Coding (http://ufldl.stanford.edu/tutor

ICA (http://ufldl.stanford.edu/tutor

RICA (http://ufldl.stanford.edu/tutor

Exercise: RICA (http://ufldl.stanford.edu/tutor

Self-Taught Learning

Self-Taught Learning (http://ufldl.stanford.edu/tutor

Exercise: Self-Taught Learning (http://ufldl.stanford.edu/tutor