6.867 Homework 3: Neural Networks

November 9, 2015

1 Approach

So far we have only discussed machine learning problems in which we have specified the basis functions we intend to use. However, since the optimal basis functions could be complicated and non-intuitive, such as in the case of images, we may want an approach that allows us to learn the basis functions. Such an approach is Artificial Neural Networks(ANN).

The idea of a neural network replaces the known basis functions with features, z_j which are parametric functions of activations, a_j learned from the input data in the first layer of learning, and then utilize a second layer to learn the relationship of the output data from those selected features.

$$a_j^{(1)} = \sum_{i=1}^{N} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{1}$$

$$z_j = g(a_j) \tag{2}$$

Where $g(x)$ is a nonlinear map of input to $(0,1)$, generally tanh or sigmoid, and a_j is an activiation, which depends on the data, $x_i,\ i = \{1, ...N\}$ and learned weights $w_j i^{(1)}$ and bias $w_0^{(1)}$.

The second layer learns in a similar fashion and can be describe for K output classes and M features in the hidden layer is thus

$$a_k^{(2)} = \sum_{j=1}^{M} w_j k^{(2)} z_j + w_{k0}^{(2)} \tag{3}$$

$$f_k = \tilde{g}(a_k^{(2)}) \tag{4}$$

Where $\tilde{g}(x)$ is the activation function, another nonlinear map, not necessarily the same as $g(x)$ and f_k is really our prediction $h_k(x, w)$. However for this problem, we will consider them equal, both sigmoids,

$$\tilde{g}(x) = g(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

1.1 Gradient Calculation

First of all we we be utilize stochastic gradient descent to train our network by updating the weights with each incoming new data point, as opposed to a batch implementation. To do so, we must first find the gradient of our cost function, J which depends on the loss function $l(w)$ but also applies a regularization of both sets of weights, $w^{(1)}, w^{(2)}$

$$J = l(w) + \lambda(||w^{(1)}||_F^2 + ||w^{(2)}||_F^2) \tag{6}$$

Where the norms above are the matrix Frobenius norm and the loss function, $l(w)$ is the negative log-likelihood given by

$$l(w) = \sum_{i=1}^{N} \sum_{k=1}^{K} -y_k^{(i)} \log(h_k(x^{(i)}, w)) - (1 - y_k^{(i)}) \log(1 - (h_k(x^{(i)}, w)) \tag{7}$$

Taking the partial differential of the cost with respect to each variable, $w^{(1)}, w^{(2)}$ leads to the gradients. For the gradient with respect to the second layer weights

$$\nabla_{w_k^{(2)}} J(w) = \frac{\partial J}{\partial a_{nk}^{(2)}} (\nabla w_k^{(2)} a_{nk}^{(2)}) + 2\lambda w_k^{(2)} \tag{8}$$

$$= \frac{\partial J_n}{\partial h_{nk}} (\tilde{q}'(a_{nk}^{(2)})) z_n + \lambda w_k^{(2)} \tag{9}$$

algebraically we can solve for the partial of the cost with respect to each prediction where

$$\frac{\partial J_n}{\partial h_{nk}} = -\frac{y_k^{(n)}}{h_k(x^{(n)}, w)} + \frac{1 - y_k^{(n)}}{1 - h_k(x^{(n)}, w)} \tag{10}$$

Introducing a new variable, $\delta_{nk}^{(2)}$,

$$\delta_{nk}^{(2)} = \frac{\partial J}{\partial a_{nk}^{(2)}} \tag{11}$$

Looking at the partial with respect to the first layer

$$\nabla_{w_j^{(1)}} J(w) = \frac{\partial J}{\partial a_{nj}^{(1)}} (\nabla w_j^{(1)} a_{nj}^{(1)}) + 2\lambda w_j^{(1)} \tag{12}$$

$$= \delta_{nj}^{(1)} x^{(n)} \tag{13}$$

Introducing a new variable, $\delta_{nj}^{(1)}$,

$$\delta_{nj}^{(1)} = \sum_{k=1}^{K} \delta_{nk}^{(2)} w_{kj}^{(2)} g'(a_{nj}^{(1)}) \tag{14}$$

1.1.1 Stochastic Gradient Descent

To actually optimal weights we will utilized stochastic gradient descend which updates the weights with each new data point,

$$w^{(t+1)} = w^{(t)} + \eta_t \nabla_w J(w^{(t)}) \tag{15}$$

Where η is the learning rate

1.2 Implementation

We will use the back prop algorithm to implement our neural network

2 Results

2.1 Toy Data Set

2.2 MNIST