Machine Learning 6.867 - Pset 3

November 7, 2015

1 Multi-Class SVM

2 Neural Networks

Neural networks are used in machine learning to make predictions, similar to logistic regression, SVM, or regression. We can represent neural networks using a graph with nodes and edges (see Bishop figure 5.1). Assume that we observe data $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}), n = 1, \ldots, N$, where $\mathbf{x}^{(n)} \in \mathbb{R}^{D+1}$ and $\mathbf{y}^{(n)} \in \{0,1\}^K$. Let $(\mathbf{x}, \mathbf{y}) = ([x_0, x_1, \ldots, x_D], [y_1, \ldots, y_K])$ be a general observation, where $x_0 = 1$ is a constant term for the bias. We create nodes for each of the features x_i, referred to as *inputs*, and nodes for each of the class labels y_i, referred to as *outputs*. Next, we introduce a series of nodes in the middle of the graph, called *hidden units*, and we draw edges connecting the **inputs** $\rightarrow$**hidden units**$\rightarrow$ **outputs**. The key idea in neural networks is that we can model the hidden units as functions of the inputs, and model the outputs as functions of the hidden units.

For 2-layer neural networks, we have one layer of hidden units denoted by $[z_0, z_1, \ldots, z_M]$. There are weights $\mathbf{w}_{ji}^{(1)}$ for each edge $x_i \rightarrow z_j$ and $\mathbf{w}_{kj}^{(2)}$ for each edge $z_j \rightarrow y_k$, which are unknown and will be learned through training the model. However, there are no edges from the inputs to z_0, because we assume $z_0 = 1$ is a constant term for the bias. Let $\sigma(\cdot)$ denote the logistic function, which we will use as the *activation function* for both the hidden and output layers of our neural network. The predicted value for output k given parameters $\mathbf{w} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)})$ and input $\mathbf{x}$ will be:

$$h_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} \sigma\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right) + w_{k0}^{(2)}\right). \tag{1}$$

2.1 Implementation

2.1.1 Gradient Descent

We implemented a 2-layer regularized neural network using gradient descent. The loss function, which is the negative log-likelihood, is equal to:

$$\ell(\mathbf{w}) \equiv \sum_{n=1}^{N} \sum_{k=1}^{K} \left[-y_k^{(n)} \log(h_k(\mathbf{x}^{(n)}, \mathbf{w})) - (1 - y_k^{(n)}) \log(1 - h_k(\mathbf{x}^{(n)}, \mathbf{w}))\right] \tag{2}$$

Let $\tilde{\mathbf{w}}^{(1)} = (\mathbf{w}_{ji}^{(1)})_{i \neq 0}$ and $\tilde{\mathbf{w}}^{(2)} = (\mathbf{w}_{kj}^{(2)})_{j \neq 0}$. We add a regularizer term, and the final cost function becomes:

$$J(\mathbf{w}) \equiv \ell(\mathbf{w}) + \lambda(\|\tilde{\mathbf{w}}^{(1)}\|_F^2 + \|\tilde{\mathbf{w}}^{(2)}\|_F^2), \tag{3}$$

where $\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$ is the matrix Frobenius norm, and λ is a fixed parameter chosen via cross-validation. Note that we exclude the bias terms from our regularizer in order to avoid penalty if the data is shifted. To take the derivative of this function, we introduce some more notation:

$$a_j^{(1)}(\mathbf{x}) \equiv \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{4}$$

$$z_j(\mathbf{x}) \equiv \begin{cases} \sigma(a_j^{(1)}(\mathbf{x})), & j = 1, \ldots, M \\ 1, & j = 0 \end{cases} \tag{5}$$

$$a_k^{(2)}(\mathbf{x}) \equiv \sum_{j=0}^{M} w_{kj}^{(2)} z_j(\mathbf{x}) \tag{6}$$

We refer to $a_j^{(1)}(\mathbf{x})$ and $a_k^{(2)}(\mathbf{x})$ as the *activations* for a fixed value of $\mathbf{x}$. It follows that $h_k(\mathbf{x}, \mathbf{w}) = \sigma(a_k^{(2)}(\mathbf{x}))$. Taking partial derivatives of J with respect to $\mathbf{w}_{kj}^{(2)}$ and $\mathbf{w}_{ji}^{(1)}$ for $i \neq 0$ and $j \neq 0$, we find:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_{kj}^{(2)}} = 2\lambda \mathbf{w}_{kj}^{(2)} + \sum_{n=1}^{N} \left(\frac{-y_k^{(n)}}{h_k(\mathbf{x}^{(n)}, \mathbf{w})} + \frac{1 - y_k^{(n)}}{1 - h_k(\mathbf{x}^{(n)}, \mathbf{w})} \right) h_k(\mathbf{x}^{(n)}, \mathbf{w})(1 - h_k(\mathbf{x}^{(n)}, \mathbf{w})) z_j(\mathbf{x}^{(n)}) \tag{7}$$

$$= 2\lambda \mathbf{w}_{kj}^{(2)} + \sum_{n=1}^{N} \left(-y_k^{(n)}(1 - h_k(\mathbf{x}^{(n)}, \mathbf{w})) + (1 - y_k^{(n)}) h_k(\mathbf{x}^{(n)}, \mathbf{w}) \right) z_j(\mathbf{x}^{(n)}) \tag{8}$$

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_{ji}^{(1)}} = 2\lambda \mathbf{w}_{ji}^{(1)} + \sum_{n=1}^{N} \sum_{k=1}^{K} \left(-y_k^{(n)}(1 - h_k(\mathbf{x}^{(n)}, \mathbf{w})) + (1 - y_k^{(n)}) h_k(\mathbf{x}^{(n)}, \mathbf{w}) \right) w_{kj}^{(2)} z_j(\mathbf{x}^{(n)})(1 - z_j(\mathbf{x}^{(n)})) x_i \tag{9}$$

For the bias terms, there is no regularizer penalty, so we obtain:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_{k0}^{(2)}} = \sum_{n=1}^{N} \left(-y_k^{(n)}(1 - h_k(\mathbf{x}^{(n)}, \mathbf{w})) + (1 - y_k^{(n)}) h_k(\mathbf{x}^{(n)}, \mathbf{w}) \right) \tag{10}$$

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_{j0}^{(1)}} = \sum_{n=1}^{N} \sum_{k=1}^{K} \left(-y_k^{(n)}(1 - h_k(\mathbf{x}^{(n)}, \mathbf{w})) + (1 - y_k^{(n)}) h_k(\mathbf{x}^{(n)}, \mathbf{w}) \right) w_{kj}^{(2)} z_j(\mathbf{x}^{(n)})(1 - z_j(\mathbf{x}^{(n)})) \tag{11}$$

Therefore, we have calculated analytic expressions for the gradients of J with respect to the different groups of parameters $\mathbf{w}^{(2)}$ and $\mathbf{w}^{(1)}$. From these expressions, we coded a function in MATLAB to train a our neural network using gradient descent. The function inputs are the data $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}), n = 1, \ldots, N$, regularization parameter λ, number of nodes in the hidden layer M, and initial solution $(\mathbf{w}_{init}^{(1)}, \mathbf{w}_{init}^{(2)})$. The function outputs the optimal solution $\mathbf{w} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)})$ that minimizes the final cost function $J(\mathbf{w})$.

Because the objective function is non-convex, we run our training function with multiple initial solutions to avoid reporting a single local optimum. After running multiple times, we report the optimal solution $\mathbf{w} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)})$ that minimizes $J(\mathbf{w})$ over all trials. The gradient descent function used

2.1.2 Stochastic Gradient Descent

We implemented an alternate function in MATLAB to train our neural network using stochastic gradient descent.

2.2 Computational Results

2.2.1 Toy Problem

2.2.2 MNIST Data