Neural Networks

We implemented a simple 2-layer regularized neural network, trained it using gradient descent, and used it on the provided toy datasets and MNIST handwritten digits datasets.

We choose to use the softmax formulation as described by Bishop for our loss function. Please note that although this is a different loss function than we were asked to use in our assignment description, there was a follow-up discussion on Piazza (see note @504 if not familiar) which clarified that softmax is actually the correct formulation for 1-of-K classification.

The likelihood according to softmax is:

$$p(\mathbf{t} \mid \mathbf{X}, \mathbf{w}) = \prod_{k=1}^{K} \prod_{n=1}^{N} y_k(\mathbf{x}_n, \mathbf{w})^{t_{nk}} \tag{1}$$

And taking the negative log likelihood we have our unregularized loss function:

$$l(\mathbf{w}) = -\sum_{k=1}^{K} \sum_{n=1}^{N} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{w}) \tag{2}$$

We note that $\mathbf{w}$ can be considered a vector that represents all of the weights of the neural network, but it is preferable to think of the weights as being organized into two matrices, which we denote $W^{(1)}$ and $W^{(2)}$ and explain later in the context of forward propagation. The matrix representation, however, is useful for including regularization in our final cost function, which is formulated via the Frobenius norm:

$$J(w) = l(w) + \lambda(||W^{(1)}||_F^2 + ||W^{(2)}||_F^2) \tag{3}$$

Gradient Calculation

Having defined our cost function, we are now able to describe how the gradients $\nabla_{W^{(1)}} J(\mathbf{w})$ and $\nabla_{W^{(1)}} J(\mathbf{w})$ may be calculated analytically. Before this derivation, however, we note for clarity that each of these terms are matrices whose elements are simply the partial derivative of the cost function with respect to that element itself (a scalar).

In order to calculate the gradients, we will use error backpropagation, for which we will follow Bishop's nice explanation and follow these sequence of steps:

1. Forward propagation
2. Evaluate δ for the output units
3. Backpropagation of the δ's for each hidden unit
4. Evaluate derivatives with $\frac{\partial E_n}{\partial w_{ji}}$

1. Forward propagation

With the weights represented as matrices, we can vectorize the computation of the unit activations, for example for the first layer:

$$A^{(1)} = W^{(1)} X_{aug} \tag{4}$$

Where X_{aug} is a $(D+1) \times N$ augmented matrix for the input data, for the purpose of including the bias input unit in the vectorized computation. If we consider the original input data to be X, of dimension $D \times N$, where D is the dimensionality of each sample input and N is the number of sample inputs, then we form X_{aug} by augmenting X with a $1 \times N$ vector of 1s ($X_{aug} = \begin{bmatrix} 1_{N\times 1} & | & X^T \end{bmatrix}^T$). $W^{(1)}$ is then a $M \times (D+1)$ matrix that contains all of the weights from every input to every hidden unit, except the hidden bias unit, and $A^{(1)}$ is a $M \times N$ matrix where each column vector is individually the weights for all of the unit activations, given one sample input. The "output" of each unit is computed simply by applying the sigmoid function $g()$ element-wise to the matrix $A^{(1)}$:

$$Z_{ij}^{(1)} = g(A_{ij}^{(1)}) \tag{5}$$

Where as requested in the assignment, we use the logistic sigmoid function as our sigmoid function:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

To forward propagate through the second layer to the output units, we similarly use $A^{(2)} = W^{(2)} Z_{aug}^{(1)}$, where $Z^{(1)}$ has been augmented with a vector of 1s to include the bias unit, and so we have $W^{(2)}$ of dimension $K \times (M+1)$. At this point we also note that in order to do our 1-of-K classification, we have to transform the output from the form they were given in the assigned datasets, from a scalar k to $\mathbf{e}_k$ where $\mathbf{e}_k$ is a vector of length $K = k_{max}$ and contains a 0 for every element except k, for example if $k = 3$ then we form $e_3 = [0\ 0\ 1]$. Thus we consider $\mathbf{t}$ to be an $K \times N$ matrix. Applying $y_{ij} = g(A_{ij}^{(2)})$, we have our outputs and have completed forward propagation.

2. Evaluate δ_k for the output units

With our softmax formulation....

... we end up with:

$$\delta_{nk} = y_{nk} - t_{nk} \tag{7}$$

Which is simply vectorized by element-wise subtraction of $K \times N$ matrices: $\delta = \mathbf{y} - \mathbf{t}$.

3. Backpropagation of the δ's for each hidden unit

Having calculated the δ's for the output unit, we have only one step of backpropagation to perform, since we have only a 2-layer neural network. We first provide the backpropagation formula, unvectorized:

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k \tag{8}$$

where $g'(z) = g(z)(1 - g(z))$ signifies the gradient of our sigmoid function. We may start vectorizing this computation so that δ is an $M \times 1$ vector and we have:

$$\delta_{hidden} = g'(A^{(1)}) * W^{(2)^T}_{\text{no bias}} \delta_{outputs} \tag{9}$$

In the above, $*$ is used to signify element-wise multiplication, and $W^{(2)^T}_{\text{no bias}}$ is a transposed matrix of $W^{(2)}$ except without the vector that corresponds to the bias units. The above equation holds for calculating δ_{hidden} for one sample at a time, but also equivalently holds for batch computation N at a time, just by using $\delta_{outputs}$ as a $K \times N$ matrix from step 2, with $W^{(2)^T}_{\text{no bias}}$ of dimension $M \times K$, and $A^{(1)}$ of dimension $M \times N$.

4. Evaluate derivatives with $\frac{\partial E_n}{\partial w_{ji}}$

Finally, we use both δ's together with our inputs, $\mathbf{x}$, and our hidden unit outputs, $Z^{(1)}$, to calculate the gradients $\nabla_{W^{(1)}} J(\mathbf{w})$ and $\nabla_{W^{(1)}} J(\mathbf{w})$.

Following Bishop, we know that we can compute the derivative with respect to any element via:

$$\frac{\partial J_n}{\partial w_{ji}} = \delta_j z_i \tag{10}$$

In the above, δ represents the δ at the output of the weight, and z represents the activation at the input of the weight. The above only calculates the gradient for one sample, and for one element of the matrix, but to vectorize, we this can be vectorized as:

$$\nabla_{W^{(1)}} J(\mathbf{w}) = \delta_{outputs} Z^{(1)^T} \tag{11}$$

$$\nabla_{W^{(2)}} J(\mathbf{w}) = \delta_{hidden} X^T \tag{12}$$

where we have explicitly calculated for the gradients with respect to each of the matrices.

Implementing 2-Layer Neural Network

Stochastic Gradient Descent

Testing the Neural Network Code

Testing the Neural Network Code

MNIST Data (Parts 5 and 6)