Math 345 - Notes
Neural Networks II
November 26 and 28, 2018

Neural Networks II

Parts to building a neural network for the data set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$ are as follows.

I. Decide on a network architecture

- The input layer is given, its size equals the number of features in the data set, the dimension of the input vector $\mathbf{x}$. Let it be denoted by d.
- The output layer is also given, it equals the number of labels or categories in the output. If the output is binary, then the output layer has size one.
- Choose the number of hidden layers (at least one). Denote the total number of layers by L, thus there are $L - 2$ hidden layers in the network.
- Choose the number of activation units in each layer, which is typically kept the same for all hidden layers. Let s_k denote the size of the layer k, for $1 \leq k \leq L$.

II. Train the network

- Randomly initialize weights in the network with small weights.
- Forward feed the input vector to compute activation units in the network
- Back-propagate computed errors to derive gradients (partial derivatives)
- (optional) Use gradient check to verify back-propagation was coded correctly
- Minimize cost function using Gradient Descent or some other optimization algorithm.
- Output a set of weights that approximate a local minimum for the cost function. The algorithm may not lead to an optimal set of weights!

III. Test the network

- Use a different testing data set $\mathcal{D}_2$
- Use the Feed Forward Algorithm to predict the output for data points in $\mathcal{D}_2$ by choosing the label with the highest activation value of at least 0.5.
- Count how many data points from $\mathcal{D}_2$ are accurately predicted and use it to compute accuracy.

Training a neural network

Setup and notation

L = number of layers
s_k = number of units in layer k, with $1 \leq k \leq L$.
$a_j^{(k)}$ = activation unit j in layer k, with $0 \leq j \leq s_k$. Here, $a_0^{(k)} = 1$, the bias term.
$\mathbf{a}^{(k)}$ is a column vector of dimension $s_k + 1$.
$W^{(k)}$ = matrix of weights controlling the map from layer k to layer $k+1$. It has dimensions s_{k+1} by $s_k + 1$, for $1 \leq k \leq L-1$.
$\mathbf{z}^{(k+1)} = W^{(k)}\mathbf{a}^{(k)}$, for $1 \leq k \leq L-1$, a column vector of dimension s_{k+1}.
To compute the value of each activation unit $a_j^{(k)}$, we use as activation function the sigmoid $\theta(x)$ discussed in class for logistic regression

$$\theta(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}.$$

For a vector $\mathbf{z} = [u, v]$, the sigmoid $\theta(\mathbf{z})$ is computed coordinate-wise: $\theta(\mathbf{z}) = \left[\frac{e^u}{1+e^u}, \frac{e^v}{1+e^v}\right]$.
For $2 \leq k \leq L$, let $a_0^{(k)} = 1$ and $a_j^{(k)} = \theta\left(z_j^{(k)}\right)$ for $1 \leq j \leq s_k$.

For $1 \leq k \leq L-1$, initialize the weights in the matrices $W^{(k)}$ with random values in $[-\epsilon, \epsilon]$, for some small ϵ. We are not initializing with zero entries in order to avoid symmetry. See class discussion for more detail. Once weights are given, based on each input, we compute activation units in the network, using the Feed Forward Algorithm.

Feed forward algorithm

For each data point $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ we populate the activation units (nodes) in the network as follows.

Let $\mathbf{a}^{(1)} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$, that is, add the bias term $a_0^{(1)} = 1$.

Let $\mathbf{z}^{(2)} = W^{(1)}\mathbf{a}^{(1)}$ (it has dimension $s_2 \times 1$)

Let $\mathbf{a}^{(2)} = \begin{bmatrix} 1 \\ \theta\left(\mathbf{z}^{(2)}\right) \end{bmatrix}$.

Let $\mathbf{z}^{(3)} = W^{(2)}\mathbf{a}^{(2)}$ (it has dimension $s_3 \times 1$)

Let $\mathbf{a}^{(3)} = \begin{bmatrix} 1 \\ \theta\left(\mathbf{z}^{(3)}\right) \end{bmatrix}$.

...

Let $\mathbf{z}^{(L)} = W^{(L-1)}\mathbf{a}^{(L-1)}$ (it has dimension $s_L \times 1$)

Let $\mathbf{a}^{(L)} = \theta\left(\mathbf{z}^{(L)}\right)$, with no bias term needed in the last layer!

Back-propagation Algorithm

Consider the error in output resulting from using the weights $W^{(k)}$ on the data point $(\mathbf{x}, \mathbf{y})$:

$$\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}.$$

How does this propagate to errors in each layer? In particular, how does changing the weights a little bit will change the output, and in particular the error? We will need to find partial derivatives for a cost (error) function and apply gradient descent to minimize it. The cost function we are using is

$$\text{Cost}(W) = -\frac{1}{N}\sum_{i=1}^{N}\left[\mathbf{y}_i \cdot \log(\theta(W\mathbf{x}_i)) + (\mathbf{1} - \mathbf{y}_i) \cdot \log(\mathbf{1} - \theta(W\mathbf{x}_i))\right],$$

where we are taking dot products and averaging over all data points in $\mathcal{D}$. We found that this cost function is very similar to what we used in logistic regression. In order to implement the Gradient Descent Algorithm, we need to find

$$gW^{(k)} = \frac{\partial}{\partial W^{(k)}}\text{Cost}(W).$$

Let $\delta_j^{(k)}$ denote how much node j in layer k was responsible for the error in output. That is,

$$\delta_j^{(k+1)} = \frac{\partial}{\partial z_j^{(k+1)}}\text{Cost}(W^{(k)}).$$

Since $z_j^{(k+1)} = \sum_i W_{ji}^{(k)} a_i^{(k)}$,

$$\frac{\partial}{\partial W_{ji}^{(k)}}\text{Cost}(W^{(k)}) = \frac{\partial}{\partial z_j^{(k)}}\text{Cost}(W^{(k)})\frac{\partial z_j^{(k+1)}}{\partial W_{ji}^{(k)}} = \delta_j^{(k+1)}\, a_i^{(k)}.$$

Therefore,

$$gW^{(k)} = \delta^{(k+1)}[\mathbf{a}^{(k)}]^T.$$

To compute $\delta^{(k)}$, let $\tilde{W}^{(k)}$ be the matrix $W^{(k)}$ with the column of ones removed. Let

$$\delta^{(k)} = [\tilde{W}^{(k)}]^T\delta^{(k+1)} \odot \theta'(\mathbf{z}^{(k)})$$

where $\odot$ stands for coordinate-wise multiplication, i.e, if $\mathbf{u} = [u_1, u_2, u_3]$ and $\mathbf{v} = [v_1, v_2, v_3]$, then

$$\mathbf{u} \odot \mathbf{v} = [u_1v_1, u_2v_2, u_3v_3]$$

Note that $\theta'(x) = \dfrac{(1+e^x)e^x - (e^x)e^x}{(1+e^x)^2} = \dfrac{e^x}{(1+e^x)^2} = \theta(x)\left[1 - \theta(x)\right]$, hence

$$\theta'(\mathbf{z}^{(k)}) = \theta(\mathbf{z}^{(k)}) \odot \left(\mathbf{1} - \theta(\mathbf{z}^{(k)})\right).$$

We will use the Back Propagation algorithm:

- Let $\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}$ $\quad \rightarrow$ size $s_L \times 1$
- For $1 \leq k \leq L-1$,

  - let $\tilde{W}^{(k)}$ be the matrix $W^{(k)}$ with the column of ones removed $\quad \rightarrow$ size $s_{k+1} \times s_k$

  - let $\delta^{(k)} = [\tilde{W}^{(k)}]^T \delta^{(k+1)} \odot \theta'(\mathbf{z}^{(k)})$ $\quad \rightarrow$ size $s_k \times 1$.

  - compute the set of partial derivatives with respect to the weights

$$gW^{(k)}(\mathbf{x}, \mathbf{y}) = \delta^{(k+1)}[\mathbf{a}^{(k)}]^T \quad \rightarrow \text{size } s_{k+1} \times s_k.$$

Minimize the cost function

Compute the gradients for the *training set* $\mathcal{D}$ by averaging over all data points: for $1 \leq k \leq L-1$

$$\text{grad}W^{(k)} = \frac{1}{N} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} gW^{(k)}(\mathbf{x}, \mathbf{y}).$$

Run the **Gradient Descent** algorithm to find the weights that minimize the cost function.

- Initialize weights with random values in $[-\epsilon, \epsilon]$ for small ϵ.
- Choose η.
- Update rule: for $1 \leq k \leq L-1$

$$W_{t+1}^{(k)} = W_t^{(k)} - \eta \cdot \text{grad}W_t^{(k)}$$

  Note that the matrices and gradients can be "unrolled" as discussed in class, so one can work with vectors instead of matrices, but that is not necessary.

- Output set of weights resulting from Gradient Descent.