Neural Networks -

Programming Club

IIT Kanpur

pclub.iitk@gmail.com

February 26, 2016
Why Neural Networks?

- Hard to incorporate higher order terms in our hypothesis in regressions
- Number of higher order terms increases rapidly with order
- Example -
  If number of features = 100
  Number of 2nd order features ≈ 5000
  Number of 3rd order features ≈ 1,70,000
- Hard to decide which of the higher order terms are relevant.
Won’t it be awesome if our learning algorithm could decide by itself which features to use in our hypothesis and which ones to neglect?

That’s exactly what Neural Networks do
• $x$ is the data vector that is fed as input to our neuron model
• $\theta$ is the parameter or weight vector that we need to learn for the model
• The yellow unit of neuron model computes the sigmoid function on the input, and gives the value of $h_\theta(x)$ as the output
• $h_\theta(x)$ is the sigmoid or logistic activation function

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$
Layer 1 is the input layer
Layer 3 is the output layer
Layer 2 is called the "Hidden Layer"
We can have more than one hidden layers in our neural network
We can also add a bias units to our input and hidden layers, which can act as a constant input for the next layer.
If network has $s_j$ units in layer $j$, $s_{j+1}$ units in layer $j + 1$, then $\theta^j$ will be $(s_{j+1}) \times (s_j + 1)$ dimensional matrix
Neural Network - Forward Propagation

- Neural Network can be viewed as nested Logistic Regression in each layer.
- In each layer, the previous layer acts as input, and current layer performs Logistic Regression to generate a new set of inputs for further layers.
An architecture with multiple hidden layers
Neural Network - Classification

- \( \{(X^{(1)}, y^{(1)}), \ldots, (X^{(m)}, y^{(m)})\} \) : Data Set
- \( L = \) Total no. of layers in the network
- \( s_l = \) no. of units (not counting bias unit) in layer \( l \)

This network can be used for multi-class classification, as it has 4 units in the last layer, representing the 4 classes that can be used for classification.
Neural Network - Cost Function

- Cost function for Logistic Regression -

\[
J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2
\]

- Cost function for Neural Network is a generalization of the above -

\[
h_\theta(x) \in \mathbb{R}^K \quad (h_\theta(x))_i = i^{th} \text{ output}
\]

\[
J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k) \right]
\]

\[
+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta^l_{ji})^2
\]

- The extra summation over \( k \) is the summation over the \( k \) output units
\[
J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k) \right] \\
+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{s_l} \sum_{s_{l+1}} (\Theta^{(l)}_{ij})^2
\]

\[
\min_{\Theta} J(\Theta)
\]

**Need code to compute:**

- \( J(\Theta) \)
- \( \frac{\partial}{\partial \Theta^{(i)}_{ij}} J(\Theta) \)
The Forward Propagation algorithm we saw previously -

\[
\begin{align*}
  a^{(1)} &= x \\
  z^{(2)} &= \Theta^{(1)} a^{(1)} \\
  a^{(2)} &= g(z^{(2)}) \quad \text{(add } a_0^{(2)}\text{)} \\
  z^{(3)} &= \Theta^{(2)} a^{(2)} \\
  a^{(3)} &= g(z^{(3)}) \quad \text{(add } a_0^{(3)}\text{)} \\
  z^{(4)} &= \Theta^{(3)} a^{(3)} \\
  a^{(4)} &= h_{\Theta}(x) = g(z^{(4)})
\end{align*}
\]
To compute the gradient, we use the backpropagation algorithm

- **Step 1** - Compute $\delta_j^{(l)} = "error"$ in node $j$ of layer $l$

- $\delta_j^{(l)}$ is computed in a fashion similar to Forward Propagation, but in the reverse direction
For each output unit (layer $L=4$)

$$\delta^{(4)}_j = a^{(4)}_j - y_j$$

For units in hidden layers

$$\delta^{(i)}_j = (\Theta^{(i)})^T \delta^{(i+1)} \cdot g'(z^{(i)}) \quad i = 2, 3$$

We don’t compute $\delta^{(1)}_j$ as it is the input, which won’t have any error.

Also, $g'(z^{(i)})$ is the derivative of the activation function $g$ evaluated at input values given by $z^{(i)}$, given by

$$g'(z^{(i)}) = a^{(i)} \cdot (1 - a^{(i)})$$
For each example in \( \{(X^{(1)}, y^{(1)}), \ldots, (X^{(m)}, y^{(m)})\} \) -

- **Step 1** - Compute \( \delta_{j}^{(l)} = "error" \) in node \( j \) of layer \( l \)

- **Step 2** - Compute \( \Delta_{ij}^{(l)} \)

\[
\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_{j}^{(l)} \delta_{i}^{(l+1)} \quad \forall \ l, i, j
\]

- **Step 3** - Compute \( D_{ij}^{(l)} \)

\[
D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} \quad \text{if } j \neq 0
\]

\[
D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0
\]

- And in the last step

\[
\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}
\]
Use some in built library functions to minimize the cost function.

Initialize the parameters $\Theta$ (in range $[-\epsilon, \epsilon]$ for some value of $\epsilon$) randomly, else all hidden layers will have same values after each iteration.

Implement some numerical gradient checking initially, to check if your gradient computation is correct.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{J(\Theta+\epsilon) - J(\Theta-\epsilon)}{2\epsilon}$$

The numerical gradient computation is slow, hence, we use it only as a check for our implementation.
Homework - Try to get the matrix based implementation of Forward Propagation and Backpropagation Algorithms