Neural Network

Fangfang Song

June 2018

1 Neural Network

1. Pick a network architecture. Usually there is 1 hidden layer or if > 1 hidden layer, have same number of hidden units in each layer by default.(the more hidden units the better)
2. Training a neural network
   (a) Randomly initialize weights
   (b) Implement forward propagation to get $h_\theta(x^{(i)})$ from $x^{(i)}$
   (c) Implement code to compute cost function$J(\Theta)$
   (d) Implement backward propagation to compute partial deriatives $\frac{1}{\partial\Theta^{(l)}{}_{ij}}\partial J(\Theta)$
   (e) use gradient checking to compare $\frac{1}{\partial\Theta^{(l)}{}_{ij}}\partial J(\Theta)$ computed using back-propogation vs. using numerical estimate of gradient of $J(\Theta)$; Then disable gradient checking code
   (f) Use gradient descent or advanced optimization method with back-propagation to try to minimize $J(\Theta)$ as a parameters Θ

2 How the network makes prediction

The network makes prediction using forward propagation, which is just a bunch of matrix multiplications and application of the activation functions. If the x is a 2 dimensional input to our network then we calculate our predictions $\vec{y}$ (also two dimensional) as follows:

$$\begin{aligned} z^1 &= {\Theta^1}^T X \\ a^1 &= sigmoid(z^1) \\ z^2 &= {\Theta^{(2)}}^T a^1 \\ a^2 &= \vec{y} = softmax(z^2) \end{aligned} \tag{1}$$

z^i is the input of layer i and a^i is the output of layer i, Θ^i is the matrix of weights controlling function mapping from layer j to layer$j+1$ is the matrix after applying the activation function. If network has s_j units in layer j, s_{j+1} units in layer j+1, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

3 Learning the parameters

Learning the parameters for our network means finding parameters (Θ_1, Θ_2) that minimize the error on our training data. We use negative log likelihood(also called cross entropy loss) to define the error. If we have m training examples and C classes then the loss for our prediction $\vec{y}$ with respect to the true label y is given by:

$$l(y, \vec{y}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{C} y_{i,j} log \vec{y}_{i,j}) \tag{2}$$

The formula looks complicated, but all it really does is sum over our training examples and add to the loss if we predicted the incorrect class. Or, if you use the column vector to represent the classification, the cost function can be written as:

$$h_\Theta(x) \in R^K \qquad (h_\Theta(x))_i = i^{th} output$$

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}(y_k^{(i)} log(h_\theta(x^{(i)})_k + (1-y_k^{(i)}) log(1-(h_\theta(x^{(i)}))_k + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ij}^{(l)})^2$$

where the last term is the regulation term.

We can use gradient descent to find the minimum, gradient descent needs the gradient of the loss function with respect to our parameters: $\frac{\partial L}{\partial \Theta_1}, \frac{\partial L}{\partial \Theta_2}$ To calculate these gradient we use the backpropagation algorithm.

4 backpropagation

$$\delta_i^l = \text{"error" of node i in layer l} \tag{3}$$

For example, we have a network have 4layers, (layer 1 is the input layer, layer 4 is the output layer), assume layer 4 have 4 output unit(multiclass classification problem).

$$\begin{aligned}
&\delta^4 = a^4 - y \\
&\\
&\delta^3 = (\Theta^3)^T \delta^4 * g\prime(z^3); \quad g\prime(z^3) = a^3 * (1 - a^3) \\
&\delta^2 = (\Theta^2)^T \delta^3 * g\prime(z^2); \quad g\prime(z^2) = a^2 * (1 - a^2) \\
&\text{No} \quad \delta^1 \\
&\text{we can prove that, if ignoring the regulation term} \\
&\frac{\partial}{\partial \Theta_{ij}^l} J(\Theta) = a_j^l \delta_i^{l+1}
\end{aligned} \tag{4}$$

where $*$ is the **element wise product** as in numpy.
Suppose we have a training set$(x^1, y^1), (x^2, y^2), ...(x^m, y^m)$
set $\Delta_{ij}^l = 0 for all l, i, j$, we will use Δ as accumulators to computer $\frac{\partial}{\partial \Theta_{ij}^l} J(\Theta)$
For i=1 to m, do the followings
Set $a^1 = X$

Perform forward propagation to computer a^l for $l = 1, 2, 3, ...l$
Using y, computer $\delta L = a^L - y$
Compute $\delta^{L-1}, \delta^{L-2}, ...\delta^2, \delta^1$
$\Delta^l_{ij} := \Delta^l_{ij} + a^l_j \delta^{l+1}_i$, or, in vector form $\Delta^l := \Delta + \delta^{j+1}(a^(l))^T$
Then:
$D^l_{ij} := \frac{1}{m}\Delta^l_{ij} + \Theta^l_{ij}$ if $j \neq 0$
$D^l_{ij} := \frac{1}{m}\Delta^l_{ij} +$ if $j = 0$
$\frac{\partial}{\partial \Theta^l_{ij}} J(\Theta) = D^l_{ij}$