Backpropagation Algorithm

Gordon Gao 2019.09.25

Neural networks are one of the most powerful learning algorithms that we have today. I will take about a learning algorithm, Backpropagation Algorithm, for fitting the parameters of a neural network given a training set.

For neural network, we also need the cost function to calculate the thetas for each layer. Its cost function is different from the Logistic Regression cost function.
Cost function of Logistic Regression with regularization:

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

Cost function of Neural Network with regularization:

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

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \cdot log\left(h_\Theta(x^i)\right)_k + \left(1-y_k^i\right)\cdot log\left(1-\left(h_\Theta(x^i)\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta_{ji}^l\right)^2$$

K is the number of output units.

Gradient Computation

We also need to minimize the error of each layer.
In order to use gradient descent or other advanced optimization algorithms, we need to compute the partial derivatives $\frac{\partial}{\partial\theta_{if}^l}J(\Theta)$.

The first thing we do is we apply forward propagation in order to compute whether a hypotheses actually outputs given the input x.

For example, here we have a neural network(layer L=4) with 1 input layer, 2 hidden layers, and 1 output layer.
Forward Propagation:

$$a^1 = x$$
$$z^2 = \Theta^1 a^1$$
$$a^2 = g(z^2)\ \left(add\ a_0^2\right)$$
$$z^3 = \Theta^2 a^2$$
$$a^3 = g(z^3)\ \left(add\ a_0^3\right)$$

$z^4 = \Theta^3 a^3$

$a^4 = h_\Theta(x) = g(z^4)$

Back propagation(Backpropagation Algorithm):

Intuition: $\delta_j^l = \text{"}error\text{" of node } j \text{ in layer } l$.

For each output unit:

$\delta_j^4 = a_j^4 - y_j$, $a_j^4 = h_\theta(x)_j^4$, Vectorize: $\delta^4 = a^4 - y$

$\delta^3 = (\Theta^3)^T \cdot \delta^4 .\times g'(z^3)$, $.\times$ is elements multiplication.

$\delta^2 = (\Theta^2)^T \cdot \delta^3 .\times g'(z^2)$

$g'(z^2)$ is the derivative of the activation function g evaluated at the input values given by z^2. $g'(z^2) = a^2 .\times (1 - a^2)$.

There is not δ^1 due to the first layer is the input layer. We do not want to change the input values.

Finally, by performing backpropagation algorithm and computing these delta terms, we can pretty quickly compute these partial derivative terms for all of our parameters:

$$\frac{\partial}{\partial \Theta_{ij}^l} J(\Theta) = a_j^l \cdot \delta_i^{l+1} \text{ } (ignoring \ \lambda \ or \ if \ \lambda = 0)$$

Backpropagation Algorithm

Training set $\{(x^1, y^1), \ldots, (x^m, y^m)\}$,

Set $\Delta_{ij}^l = 0 \ (for \ all \ l, i, j)$

For i = 1 to m:

Set $a^1 = x^i$, input layer.

Perform forward propagation to compute a^l for l = 2, 3, …, L.

Using y^i, compute $\delta^L = a^L - y^i$.

Compute $\delta^{L-1}, \delta^{L-2}, \ldots, \delta^2$.

Finally, we use these capital delta terms to accumulate these partial derivative terms: $\Delta_{ij}^{l} := \Delta_{ij}^{l} + a_j^l \cdot \delta^{l+1}$, Vectorize: $\Delta^{l} := \Delta^{l} + \delta^{l+1} \cdot (a^l)^T$.

Out side for loop:

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

Then, we can use $\frac{\partial}{\partial\Theta_{ij}^{l}} J(\Theta) = D_{ij}^{l}$ in either gradient descent or in one of the advanced optimization algorithms.