2021/11/29

Train a Neural Network

1. Pick a Network Architecture 1
2. Randomly Initialize Weights 2
3. Implement Forward Propagation 2
4. Compute Cost Function 2
5. Implement BP to Compute Partial Derivatives 3
6. Use Gradient Checking to Confirm BP Works. (Then disable it) 3
7. Use Gradient Descent or a Built-in Optimization Function to Minimize the Cost Function with the Weights in Theta 3

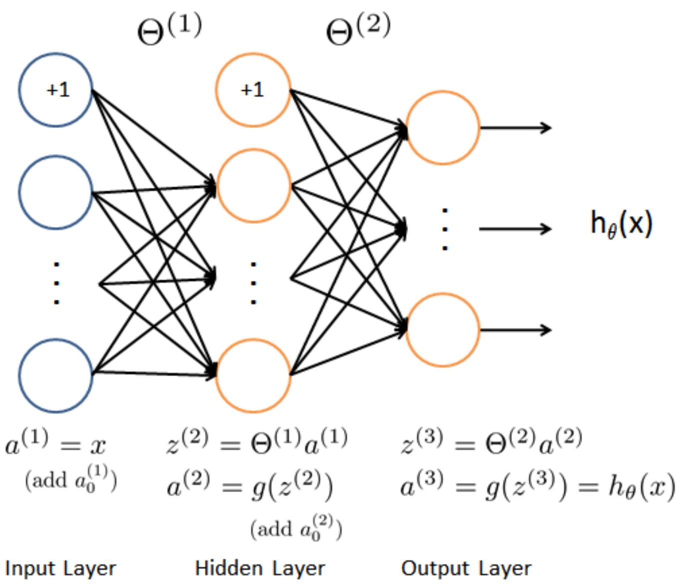
1. Pick a Network Architecture

The input X is a matrix consists of vectors $X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$, with m set of inputs.

Consider the network has L layers with each layer of units s_l.

Thus $z^{(l)}$ for every layer is $z^{(l)} = a^{(l-1)} \cdot (\Theta^{(l)})^T$.

$\Theta^{(l)}$ is $[\Theta_{ij}]_{s_{l+1} \times (s_l+1)}$, considering bias terms, and the column indices starts at zero.



2. Randomly Initialize Weights

Randomly initialize the parameters for symmetry breaking.

For each layer, initialize $\Theta^{(l)}$ uniformly in the range $[-\varepsilon_{init}, \varepsilon_{init}]$.

3. Implement Forward Propagation

Use FP to compute every $z^{(l)}$, $a^{(l)}$, including the output layer result $h_\Theta^{(i)}$.

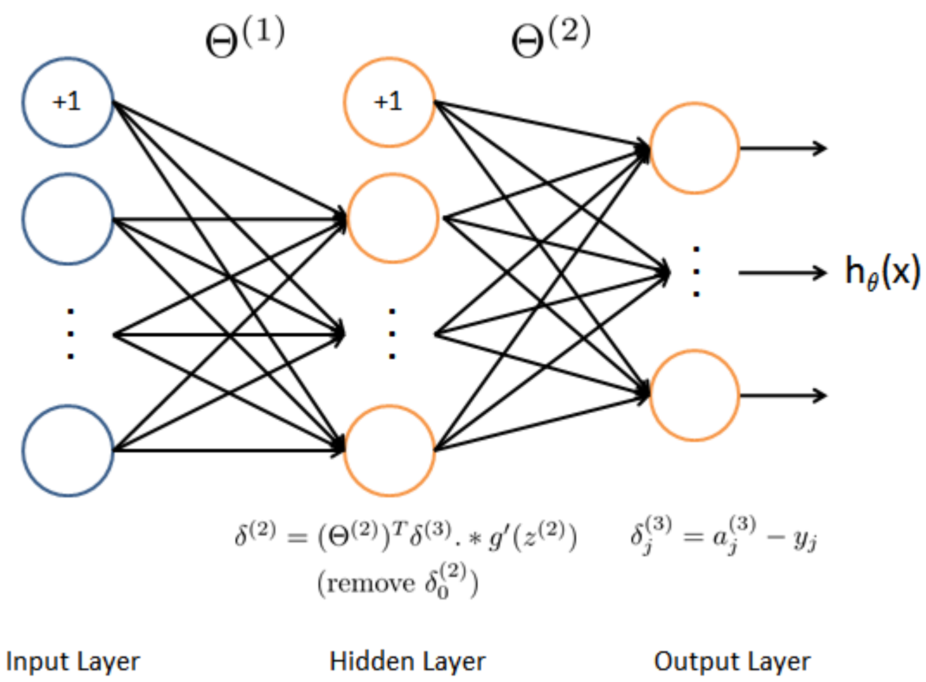
Remember to add x_0 and $z_0^{(l)}$ as bias terms for $l \in [1, L-1]$.

4. Compute Cost Function $J(\Theta)$

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

To compute $J(\Theta)$, y must be formatted, say, extend y to the same dimension of h_Θ.

i is ranged in $[0, s_{l+1}]$ and j is ranged in $[1, s_l]$, so as to exclude the bias terms.



5. **Implement BP to Compute Partial Derivatives**

The gradient we need is $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta) = D_{jk}^{(l)} = \frac{1}{m} \Delta_{jk}^{(l)} (+ \frac{\lambda}{m} \Theta_{jk}^{(l)} \ for \ k \neq 0)$.

For $i = 1 : m$

Perform FP and BP using example $(x^{(i)}, y^{(i)})$.

Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2,\ldots,L$.

Compute $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} \cdot (a^{(l)})^T$.

Compute the gradient for the neural network as follows

Compute the sigmoid gradient for each layer $\frac{d}{dz} g(z^{(l)}) = g(z^{(l)})(1 - g(z^{(l)}))$.

In output layer, define $\delta^{(L)} = a^{(L)} - y$, where $y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}_{m \times output\ units}$.

δ reflects error of inputs in each layer, thus $\delta^{(l)} = [\delta_{ij}^{(l)}]_{m \times s_l}$, same size as $z^{(l)}$.

For hidden layers l, $\delta^{(l)} = \delta^{(l+1)} \cdot \Theta^{(l)}(: ,2 : end) .* g'(z^{(l)})$.

PAY ATTENTION TO BIAS TERMS

6. **Use Gradient Checking to Confirm BP Works. (Then disable it)**
7. **Use Gradient Descent or A Built-in Optimization Function to Minimize the Cost Function with the Weights in Theta**