Neural Networks: Back-propagation Algorithm
AI4003-Applied Machine Learning

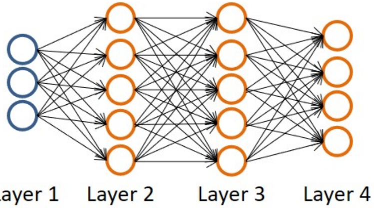
Dr. Mohsin Kamal

Department of Electrical Engineering
National University of Computer and Emerging Sciences, Lahore, Pakistan

1 Cost Function

2 Backpropagation algorithm

3 Backpropagation intuition



$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$
L = total no. of layers in network (e.g., L =4)
s_l = No. of units (not including bias unit) in layer l e.g., $s_1 = 3$, $s_2 = 5$, $s_3 = 5$ and $s_4 = s_L = 4$.

Binary classification	Multi-class classification (K classes)
$y = 0 \text{ or } 1$	$y \in \Re^K$ e.g., $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$
1 output unit	K output units

The cost function for neural network will be the generalization that we used for logistic regression.
In **logistic regression**, when considering regularization, we defined cost function as:

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

In neural networks, we might have K logistic regression output units. So the above equation becomes:

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

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

1 Cost Function

2 Backpropagation algorithm

3 Backpropagation intuition

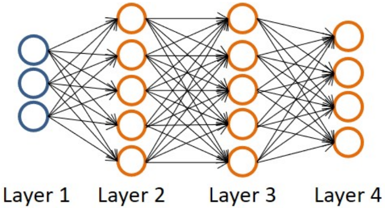
Gradient computation

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

Our goal is to find parameters Θ to minimize cost function $J(\theta)$
We need a code which will take inputs as parameters Θ and computes:

- $J(\theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta)$

Gradient computation

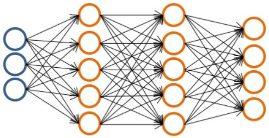


Given one training example (x, y):
Forward propagation:

$a^{(1)} = x$
$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$)
$z^{(4)} = \Theta^{(3)} a^{(3)}$
$a^{(4)} = h_\Theta(x) = g(z^{(4)})$

Gradient computation: Backpropagation algorithm

In order to compute derivatives, we will use an algorithm called "Backpropagation".
Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l. The will capture the error in activating a node $a_j^{(l)}$



- For each output unit (layer $L = 4$)
- $\delta_j^{(4)} = a_j^{(4)} - y_j$
- $a_j^{(4)}$ is the $(h_\Theta(x))_j$
- If we want to write it in vector form then $\delta^{(4)} = a^{(4)} - y$ where, each is vector whos dimension is equal to number of output units in our network.
- $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$
- $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$
- No $\delta^{(1)}$ because layer 1 is the input layer which has features with no errors associated with it.
- Where, $g'(z^{(i)}) = a^{(i)} .* (1 - a^{(i)})$.
- It is proved that after applying the above equations that: $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$ if λ is ignored.

Backpropagation algorithm

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

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

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

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

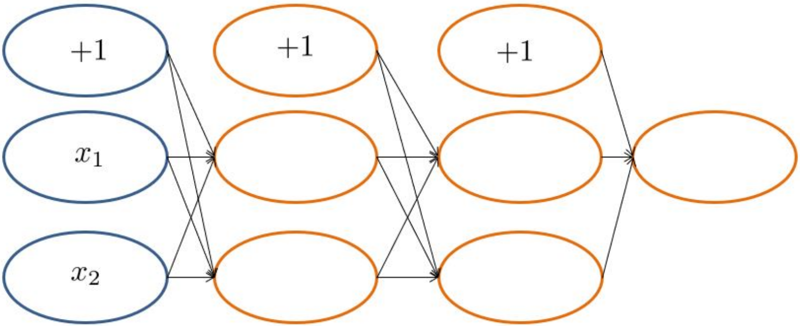
$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

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

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} \quad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Forward propagation



What is backpropagation doing?

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

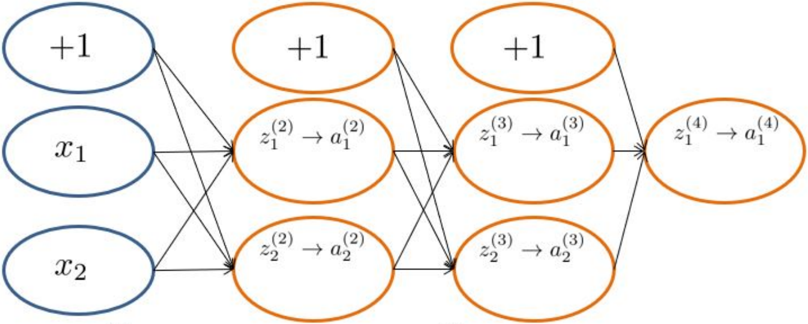
Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\mathrm{cost(i)} = y^{(i)}\log h_\Theta(x^{(i)}) + (1-y^{(i)})\log h_\Theta(x^{(i)})$$

(Think of $\mathrm{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

I.e. how well is the network doing on example i?

Forward Propagation



$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l).

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost(i)}$ (for $j \geq 0$), where

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$