Contents

Neural Networks

June 2, 2016

Part I

Week 5

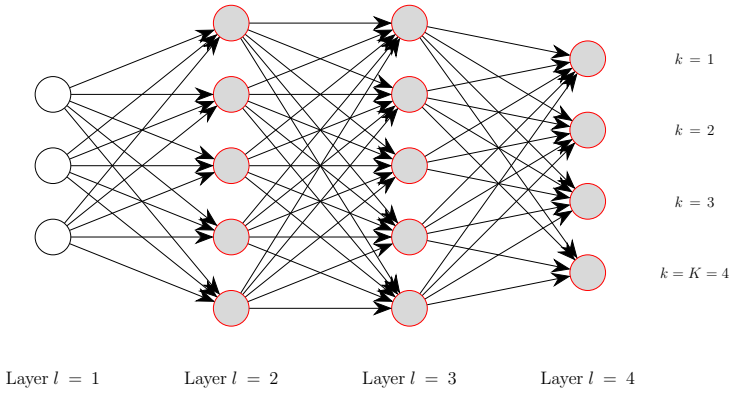
0.1 Pro and Cons

Advantages of NN	Disadvantages
+ Can represent non-linear boundary	+ Gradient descent is not guaranteed to reach a global optimum
+ Fast feed forward architecture	+ We do not know the optimal architecture (Nbr of inputs/hidden nodes/layers/output nodes)
	+ setting the adjustable parameter learning rate

Note that it is sometimes preferable to stop training early for better generalization performance. Generalization means performance on unseen input data - if you train too long, you can often get over-fitting. By stopping the training earlier, one hopes that the network will have learned the broad rules of the problem.

0.2 Cost function

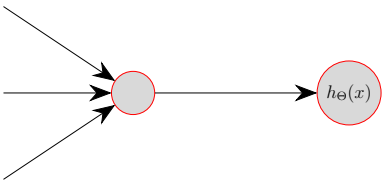
Let's consider a Neural Network with 4 layers. In this example with 2 features, one could use logistic regression with polynomial terms:



- Given a set of training examples are $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$ (i=i$^{\text{th}}$ training example).
  - L : number of layers in network (here $L = 4$)
  - s_l : total number of units (not including the bias unit) in layer l ($s_1 = 3, ..., s_4 = s_L = 4$)
  - K : number of output units (classes)

0.2.1 Binary vs. Multiclass

- For Binary Classification $y \in \{0, 1\}$
  - $s_L = 1$
  - $K = 1$ (1 ouput unit)
  - $h_\Theta(x) \in \mathbb{R}$



- For Multiclass classification
  - K output units ($y \in \mathbb{R}^K$)
    for example 4 classes: $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$
  - $h_\Theta(x)_k$ is the hypothesis that results in the k^{th} output
  - $s_L = K$ where $K \geq 3$

0.2.2 Calculate Cost function

- **Logistic regression**

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

- **Neural Network**

$h_\Theta(x) \in \mathbb{R}^K$, where $(h_\Theta(x))_k$ is the k^{th} output.

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

- 1st term: sum of the logistic regression costs calculated for each unit in the output layer, for each training example
- 2nd term is called the weight decay term[1]: sum of all the individual Θ's in the entire network.
  Note that the i in $\sum_{i=1}^{s_l}$ does not refer to the training example

0.3 Minimization of the Cost Function

We need to compute the cost function, and minimize it: $\min_\Theta J(\Theta)$:

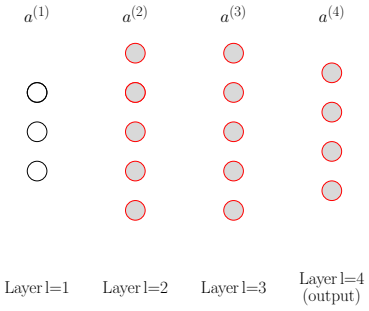
$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) \text{ with } \Theta_{ij}^{(l)} \in \mathbb{R}$$

[1] A most popular regularization method is known as weight decay: $1/2\sum_k w_k^2$ (sum of all squared weights). One weakness of the weight decay method is that it is not able to drive small iirelevant weights to zero, which may results in many small weights

0.3.1 Compute Gradient

Let's start with a system with one training example (x, y):

- **Forward Propagation**



$$
\begin{aligned}
a^{(1)} &= x \\
a^{(2)} &= g(z^{(2)}) \rightarrow z^{(2)} = \Theta^{(1)} a^{(1)} (\text{ add } a_0^{(2)}) \\
a^{(3)} &= g(z^{(3)}) \rightarrow z^{(3)} = \Theta^{(2)} a^{(2)} (\text{ add } a_0^{(3)}) \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)}) \rightarrow z^{(4)} = \Theta^{(3)} a^{(3)} \\
& \quad a^{(4)} \text{ is the output hypothesis (vector or Real)}
\end{aligned}
$$

- **Back Propagation**

Back propagation takes the output $h_\Theta(x)$, compares it to y and calculates how wrong the network was, i.e how wrong the Θ_{ij}^l were. Using the error calculated on the output layer, we then back-calculate the error associated with each unit from the preceding layer (i.e layer $(L-1)$, ... down to the input layer (where there is obviously no error). These error measurements are then used to calculate the partial derivatives, which gradient descent needs to minimize the cost function. This is repeated until convergence.

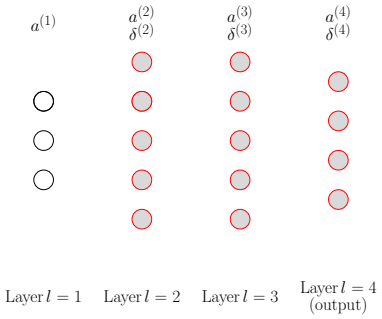


Figure 1: Example of a network [3-5-5-4]. l : layer , s: node in that layer

We define $\delta_j^{(l)}$ as the "error" of node j in layer l, i.e error of $a_j^{(l)}$ activation value. It can be shown that (see appendix):

- $\delta_j^{(4)} = a_j^{(4)} - y_j = [(h_\Theta(x))_j - y] \rightarrow$ Vectorized form: $\delta^4 = a^4 - y$
  (δ^4, a^4 and y are all vectors with dimension = Nbr of output unit in layer 4 (K)

- $\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* g'\left(z^{(3)}\right) = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* \left(a^{(3)} .* (1 - a^{(3)})\right)$

- $\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'\left(z^{(2)}\right) = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* a^{(2)} .* (1 - a^{(2)})$

There is no $\delta^{(1)}$ term for the first layer (input layer).

We can then calculate the derivative of the cost function (for $\lambda = 0$):

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \tag{1}$$

0.4 Implement Back Propagation Algorithm

1. Given a training set of m examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
2. Set: $\Delta_{ij}^{(l)} = 0$ for all l(layer), i(?????), j(node)
3. For training example $t = 1$ thru m:
   - set $a^{(1)} = x^{(t)}$
   - Perform Forward propagation to compute $a^{(l)}$ for $l = 2, 3, ..., L$
   - Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
   - Compute the error node $\delta^{(L-1)}, \delta^{(L-2)}, ..., \delta^{(2)}$ (no $\delta^{(1)}$) using:

$$\delta^{(l)} = \left((\Theta^{(l)})^T \delta^{(l+1)}\right) .* a^{(l)} .* (1 - a^{(l)}) \tag{2}$$

   - Compute accumulators Δ:

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \rightarrow \text{vectorized form} : \Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} \left(a^{(l)}\right)^T$$

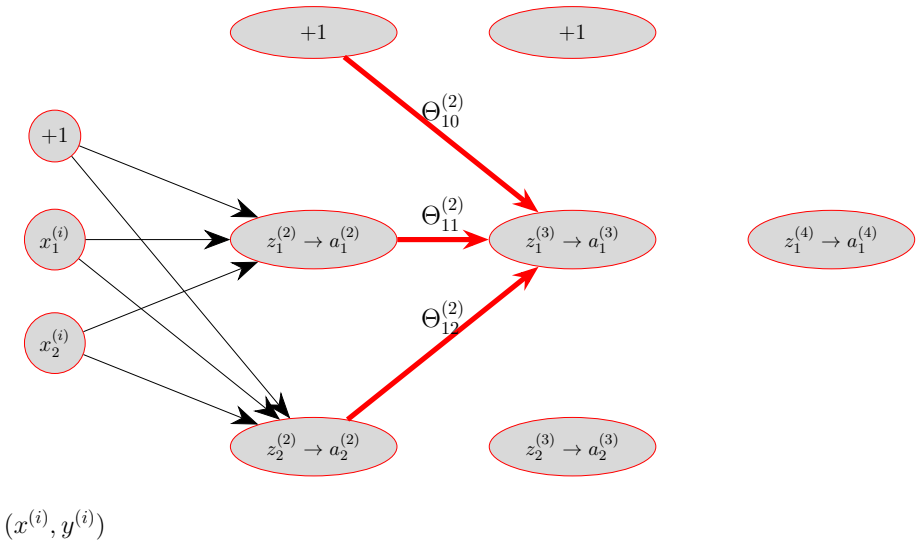
   - Exit Loop
4. Calculate the partial derivatives $D_{ij}^{(l)} = \frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$

$$D_{ij}^{(l)} := \begin{aligned} &\frac{1}{m}\left(\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}\right) && \text{if } j \neq 0 \\ &\frac{1}{m}\Delta_{ij}^{(l)} && \text{if } j = 0 \end{aligned}$$

0.5 Back propagation Intuition

0.5.1 Forward Propagation

Let's consider the case of a single training example. The count of the units does not include the bias unit.



If we consider node 2 in layer 3:

$$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} \times a_1^{(2)} + \Theta_{12}^{(2)} \times a_1^{(2)} \tag{3}$$

We can then apply the activation function to $z_1^{(3)}$ to get the activation value $a_1^{(3)}$.

0.5.2 Back Propagation

If we assume $\lambda = 0$, the cost function for a system with 1 output is defined by:

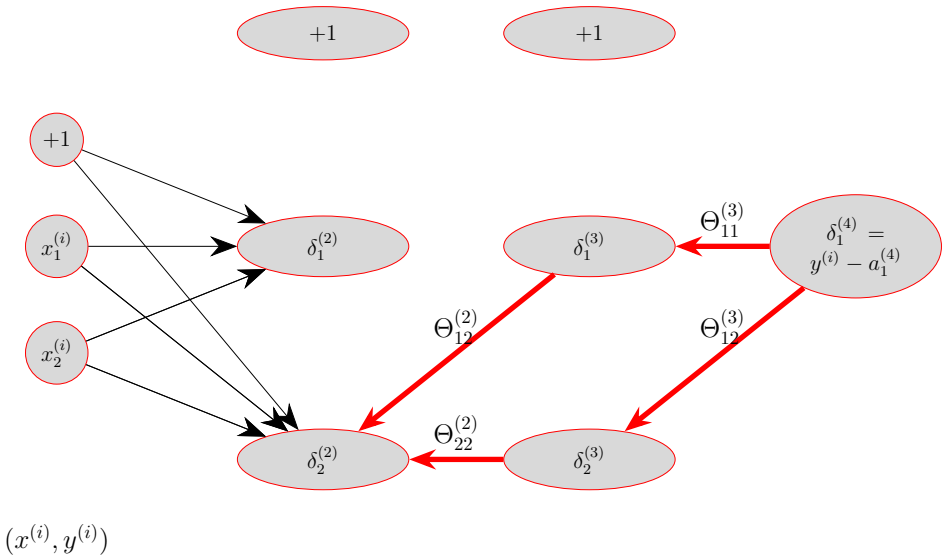
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right]$$

For a single training example (x^i, y^i), the cost is:

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

We can think of cost(i) as being approximately the square error: cost$(i) \approx (h_\Theta(x^{(i)} - y^{(i)})^2$. The cost is a measure of how well is the network doing on example (i).
Back propagation is computing $\delta_j^{(l)}$, the error of activation value calculated for unit j in l^{th} layer. Formally (for $j \geq 0$):

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}}\text{cost}(i)$$



The error terms can be written as :

$$\delta_2^{(2)} = \Theta_{12}^{(2)}\delta_1^{(3)} + \Theta_{22}^{(2)}\delta_2^{(3)} * g'(z_{(2)})$$
$$\delta_2^{(3)} = \Theta_{12}^{(3)}\delta_1^{(4)} * g'(z_2^{(3)}) \tag{4}$$

0.6 Implementation note

In previous assignments, we used the following lines to get the optimum **theta**, where **fminunc()** was the optimization algorithm. This routine assumes **initialTheta**, **theta** and **gradient** are all vectors.

```
function [jVal, gradient] = costFunction(theta);
optTheta = fminunc(@costFunction, initialTheta, options)
```

With Neural Network, gradient **D**, **theta** are not vectors but matrices, so we need to unroll those matrices into vectors.

- Let's take a NN with $s_1 = 10$ (units in layer1), $s_2 = 10$ (units in layer2), $s_3 = 10$ (units in layer3)
- $\Theta^{(1)} \in \mathbb{R}^{(10\times 11)}$ and $\Theta^{(2)} \in \mathbb{R}^{(1\times 11)}$
- $D^{(1)} \in \mathbb{R}^{(10\times 11)}$ and $D^{(2)} \in \mathbb{R}^{(1\times 11)}$
- before sending Θ and D to **fminunc**, we will unroll them:

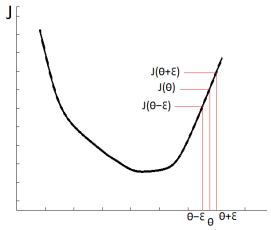
```
thetaVec = [Theta1(:), Theta2(:)];
DVec = [D1(:), D2(:)];
```

- when the result of **theta** is sent back from **fminunc**, the cost and gradient needs to be calculated but with **theta** in a matrix format:

```
Theta1 = reshape(thetaVec(1:110), 10,11));
Theta1 = reshape(thetaVec(111:121), 1,11));
```

0.7 Gradient checking

This section shows a method to calculate a numerical estimation of gradient descent.



$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\epsilon)+J(\theta-\epsilon)}{2\epsilon}$
typically use $\epsilon = 10^{-4}$
Note that $\theta \in \mathbb{R}$

- Octave implementation

For $\theta \in \mathbb{R}^n$ is a vector (unrolled version) of $\Theta^{(1)}$, $\Theta^{(2)}$..., i.e $\theta = [\theta_1, \theta_2, ...\theta_n$. The partial derivatives are:

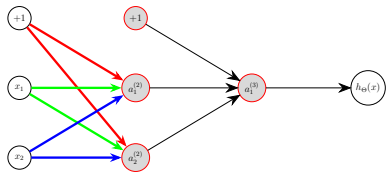
$$\begin{aligned}
\frac{\partial}{\partial\theta_1}J(\theta) &\approx \frac{J(\theta_1+\epsilon,\theta_2,...,\theta_n) - J(\theta_1-\epsilon,\theta_2,...,\theta_n)}{2\epsilon} \\
\frac{\partial}{\partial\theta_2}J(\theta) &\approx \frac{J(\theta_1,\theta_2+\epsilon,...,\theta_n) - J(\theta_1,\theta_2-\epsilon,...,\theta_n)}{2\epsilon} \\
&..... \\
&..... \\
\frac{\partial}{\partial\theta_n}J(\theta) &\approx \frac{J(\theta_1,\theta_2,...,\theta_n+\epsilon) - J(\theta_1,\theta_2,...,\theta_n-\epsilon)}{2\epsilon}
\end{aligned} \tag{5}$$

```
for i=1:n,
        thetaPlus = theta;
        thetaPlus(i) = thetaPlus(i) + EPSILON;
        thetaMinus = theta;
        thetaMinus(i) = thetaMinus(i) - EPSILON;
        gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*EPSILON);
end
```

- We then check that gradApprox $\approx$ Dvec (Dvec from backprop)

0.8 Random Initialization for NN

We have typically been using **initialTheta=zeros(1,n)**, but for NN this does not work.



If $\Theta_{ij}^{(l)} = 0$ forall i, j, l
$\rightarrow a_1^{(2)} = a_2^{(2)}$
$\rightarrow \delta_1^{(2)} = \delta_2^{(2)}$
$\rightarrow \frac{\partial}{\partial\Theta_{01}^{(1)}}J(\Theta) = \frac{\partial}{\partial\Theta_{02}^{(1)}}J(\Theta)$
$\rightarrow$ so even after 1 gradient descent update$\text{new_}\Theta_{01}^{(1)} = \text{new_}\Theta_{02}^{(1)}$
$\rightarrow \text{new\_}a_1^{(2)} = \text{new\_}a_2^{(2)}$
This is the problem of weight symmetry

For NN, we use random **initialTheta** to break the symmetry: i.e initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$ (small value close to 0).

```
Theta1 = rand(10,11) \times (2 \times INIT_EPSILON) - INIT_EPSILON
```

This creates a (10,11) matrix with values between 0 and 1.

- Typically ϵ number is based on the different units in the network:

$$\epsilon = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}} \tag{6}$$

where $L_{in} = s_l$ and $L_{out} = s_{l+1}$, i.e the number of units adjacent to $\Theta^{(l)}$

0.9 Summary of NN implementation

1. **Pick a network architecture**
   - Nbr of input units (=Nbr of features) is set by the problem
   - Nbr of output units (=Nbr of classes) is set by the problem
   - number of hidden layer: a reasonable default is 1 hidden layer or if > 1, the hidden layers have same Nbr of units
2. **Training a Neural Network**
   - randomly initialize weights
   - implement forward propagation to get $h_\Theta(x^{(i)})$ for example $x^{(i)}$
   - implement code to compute $J(\Theta)$
   - implement backprop to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

   This is done with a for-loop:

```
for i = 1:m
        %Perform forward propagation and backprop on exampel i
        %get activations a(l) and delta(l)
        for l = 2:L
                Delta^(l) = Delta^(l) + delta^(l+1) Transpose[(a^(l))]

        end
end
...
%Compute partial derivative
```

3. **Use gradient checking**
   - compare numerical estimate of gradient of $J(\Theta)$ with $\frac{\partial}{\partial \Theta_{ij^l}} J(\Theta)$ (from backprop)

- Disable gradient checking

4. Use gradient descent or advanced optimization method (fminunc, etc..) with backprop to try to minimize $J(\Theta)$ as a function of Θ

   - compute $\frac{\partial}{\partial \Theta_i j^{(l)}} J(\Theta)$

   for NN, $J(\Theta)$ is not convex, and gradient descent or other minimization algorithm can in theory be stucked in local minimum.

Appendices

.1 Derivation of the node error term δ

- We know that for a logistic regression classifier, the cost function is defined as:

$$J(\Theta) = -y\log[h_\Theta)(x)] - (1-y)\log[1-h_\Theta(x)] \tag{7}$$

applied over the K output neurons, and for all m examples.

- The partial derivatives of $J(\Theta)$ over the Θ in the output layer $(\partial J(\Theta)/\partial\Theta)$ is:

$$\frac{\partial J(\Theta)}{\partial \Theta^{(L-1)}} = \frac{\partial J(\Theta)}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial \Theta^{(L-1)}} \tag{8}$$

- The partial derivatives of $J(\Theta)$ over the Θ in the last hidden layer $(\partial J(\Theta)/\partial\Theta)$ is:

$$\frac{\partial J(\Theta)}{\partial \Theta^{(L-2)}} = \frac{\partial J(\Theta)}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial \Theta^{(L-1)}}\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}\frac{\partial z^{(L-1)}}{\partial \Theta^{(L-2)}} \tag{9}$$

- Clearly the output layer and the hidden layer immediately before it share some pieces in common, which we denotes $\delta^{(L)}$, the error node

$$\delta^{(L)} = \frac{\partial J(\Theta)}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}} \tag{10}$$

- Similarly $\delta^{(L-1)}$ would be the pieces shared by the final hidden layer $(L-1)$ and a hidden layer before it $(L-2)$:

$$\begin{aligned}\delta^{(L-1)} &= \frac{\partial J(\Theta)}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \\ &= \delta^{(L)}\frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}\end{aligned} \tag{11}$$

- With this δ-terms, our equations become:

$$\begin{aligned}\frac{\partial J(\Theta)}{\partial \Theta^{(L-1)}} &= \delta^{(L)}\frac{\partial z^{(L)}}{\partial a^{(L-1)}} \\ \frac{\partial J(\Theta)}{\partial \Theta^{(L-2)}} &= \delta^{(L-1)}\frac{\partial z^{(L-1)}}{\partial a^{(L-2)}}\end{aligned} \tag{12}$$

Now we evaluate those derivatives:

- Let's start with the output layer:

$$\frac{\partial J(\Theta)}{\partial \Theta^{(L-1)}} = \delta^{(L)}\frac{\partial z^{(L)}}{\partial a^{(L-1)}} \tag{13}$$

Using $\delta^{(L)} = \frac{\partial J(\Theta}{\partial a^{(L)}}\partial z^{(L)}$ and $J(\Theta) = -y\log[a^{(L)}] - (1-y)\log[1-a^{(L)}]$ where $a^{(L)} = h_\Theta(x)$, we get:

$$\frac{\partial J(\Theta)}{\partial a^{(L)}} = \frac{y-1}{a^{(L)}-1} - \frac{y}{a^{(L)}} \tag{14}$$

and given $a = g(z)$, where $g = \frac{1}{1+e^{-z}}$:

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = a^{(L)}(1-a^{(L)}) \tag{15}$$

- We now substitute in the definition of $\delta^{(L)}$:

$$\begin{aligned}\delta^{(L)} &= \frac{\partial J(\Theta)}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}} = \left(\frac{y-1}{a^{(L)}-1} - \frac{y}{a^{(L)}}\right)(a^{(L)}(1-a^{(L)})) \\ &= a^{(L)} - y\end{aligned} \tag{16}$$

- Given $z = \Theta * input$ where in layer L, the input is $a^{(L-1)}$:

$$\begin{aligned}\frac{\partial z^{(L)}}{\partial \Theta^{(L-1)}} &= \delta^{(L)}\frac{\partial z^{(L)}}{\partial \Theta^{(L-1)}} \\ &= (a^{(L)} - y)a^{(L-1)}\end{aligned} \tag{17}$$

- Let's run the calculation for the hidden layer $\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \delta^{(L-1)}\frac{\partial z^{(L-1)}}{\partial \Theta^{(L-2)}}$ (we assume we have 1 hidden layer):

Once again, given $z = \Theta * input$:

$$\begin{aligned}\frac{\partial z^{(L)}}{\partial a^{(L-1)}} &= \Theta^{(L-1)} \\ \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} &= a^{(L-1)}(1 - a^{(L-1)}\end{aligned} \tag{18}$$

- We then substitute in $\delta^{(L-1)}$

$$\begin{aligned}\delta^{(L-1)} &= \delta^{(L)}\frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \\ &= \delta^{(L)}\Theta^{(L-1)}(a^{(L-1)}(1-a^{(L-1)}))\end{aligned} \tag{19}$$

- So, for a 3 layer network:

$$\delta^{(2)} = \delta^{(2)}\Theta^{(2)}(a^{(2)}(1-a^{(2)})) \tag{20}$$

- The derivative of the cost function about the weights Θ in the last hidden layer:

$$\begin{aligned}\frac{\partial J(\Theta)}{\partial \Theta^{(L-2)}} &= \delta^{(L-1)}\frac{\partial z^{(L-1)}}{\partial \Theta^{(L-2)}} \\ &= (\delta^{(L)}\frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}})a^{(L-2)} \\ &= ((a^{(L)} - y)(\Theta^{(L-1)})(a^{(L-1)}(1-a^{(L-1)})))(a^{(L-2)})\end{aligned} \tag{21}$$

Derivative $g'(z)$:

$$\begin{aligned}g(z)' =& (\frac{1}{1+e^{-z}})' = -\frac{-e^{-z}}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})}g(z) = \frac{1-1+e^{-z}}{(1+e^{-z})}g(z) \\ =& (1 - \frac{1}{(1+e^{-z})})g(z) = (1-g(z))g(z)\end{aligned} \tag{22}$$

.2 links

.3 Good to know

- The Hadamard or Schur product denotes "element-wise" product of 2 matrices:

$$s \odot t \tag{23}$$

- An alternative to logistic function (sigmoid) is :

$$g(x) = \tanh(x)$$

- Cross entropy cost function ???