Artificial Neural Network backpropagation intuition

Artificial Neural networks (ANN) have emerged in the past few years as an area of unusual opportunity for research, development and application to a variety of real world problems. Examples include handwritten digit recognition, language translation, scene understanding to list a few which are complex systems to program and model mathematically. This blog briefly talks about ANN inner mechanics of forward propagation, backward propagation and mathematical intuition behind it.

Before we delve more into internal mechanism of ANN; we need to first understand certain things. Consider the example B of nonlinear classification; this example require polynomial features like x_1^2, x_2^2 to separate the classes. The classes are not separable by a line as in example A, however, if we project the data points into higher dimensions i.e. instead of using features x_1 and x_2, if we add polynomial features for degree 2 then we are able to find a linear hyper plane $\Theta^T X = 0$ in higher dimensional space to separate the data into two classes.

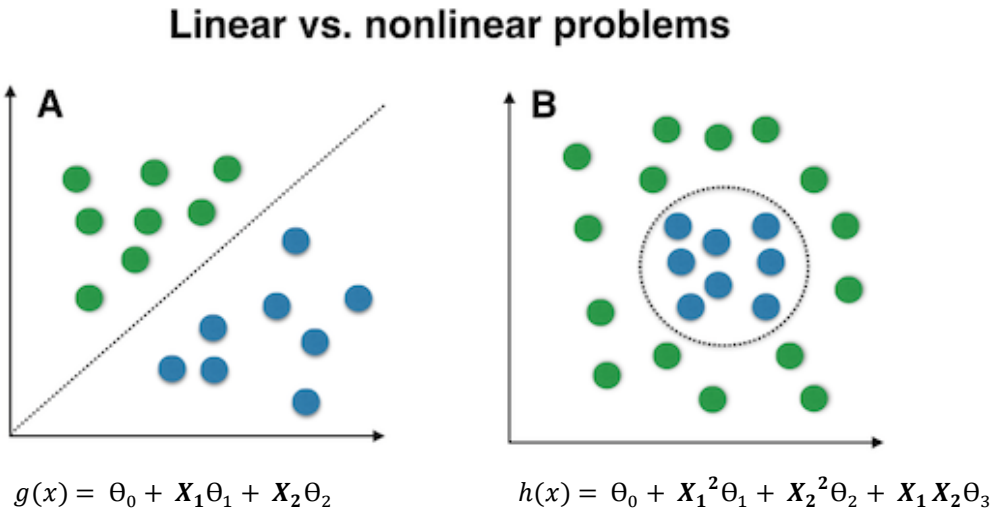

Figure 1: Linear Vs Non-Linear Classification [1]

The algorithm will find the right values of $\Theta's$ for B. In two dimension the projection will look like a circle as shown in example B.

A hyperplane with many polynomial features is not an easy way to learn complex non-linear hypothesis, whereas, ANN fine tunes and self learns parameters by itself which acts as a foundation for robust non-linear function approximation tool. In today's world, ANN is widely used in many practical applications including regression, classification, unsupervised and reinforcement learning problems.

Consider the following function $f(Z) = \frac{1}{1+e^{-Z}}$ this function takes an input z and squashes it to give an o/p like 'S' curve.

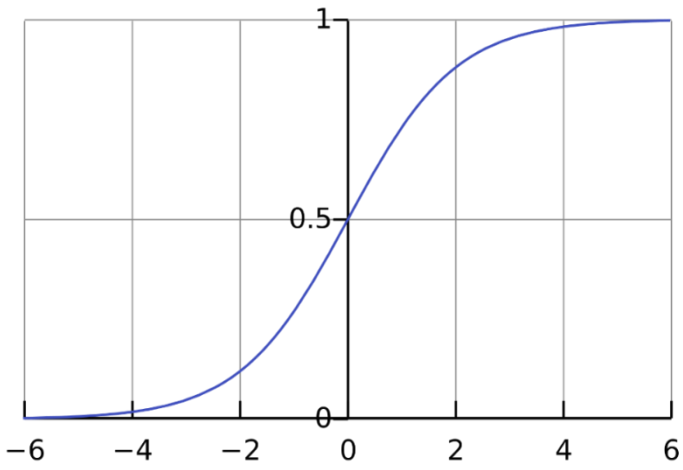

Figure 2: Logistic curve

This function varies between 0 and 1 which is useful for representing probability ($0 \leq f(Z) \leq 1$). It is differentiable and the differentiation of this function can be expressed in terms of the function itself. This is handy for computation, as once you know the function you know its differentiation and you do not compute anything extra. Moreover you can combine many such 'S' curves linearly to approximate a non-linear functions.

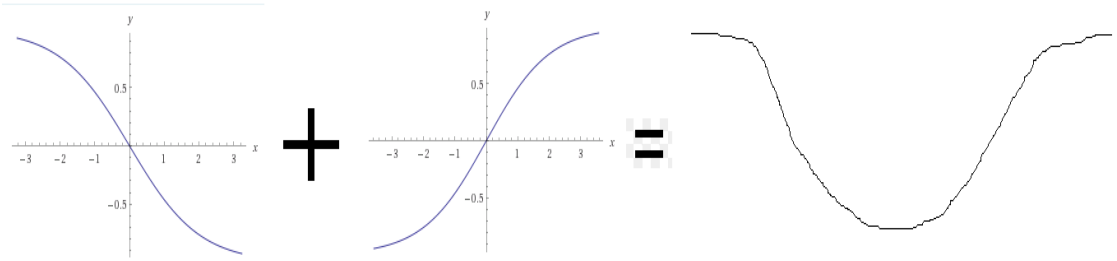

Figure 3: Combine logistic functions to approximate non-linear function

For example, the below diagram depicts the approximation of non-linear functions using linear combinations of logistics functions.

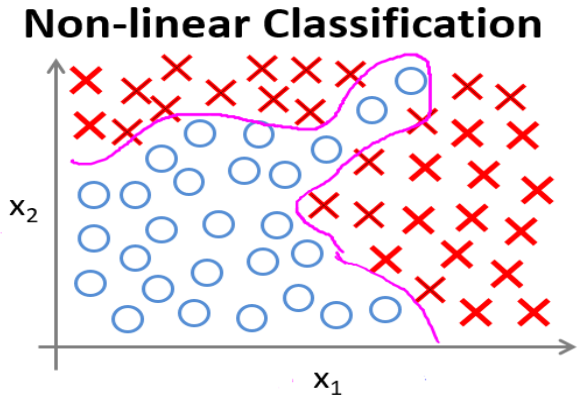

Figure 4[2]: Non Linear classification

This is perfect for computation as we can exploit the linear algebra libraries to linearly combine these sigmoid functions to approximate nonlinear hypothesis.

Now let us understand the mathematical intuition behind ANN forward propagation and back propagation mechanics based on below sample ANN.

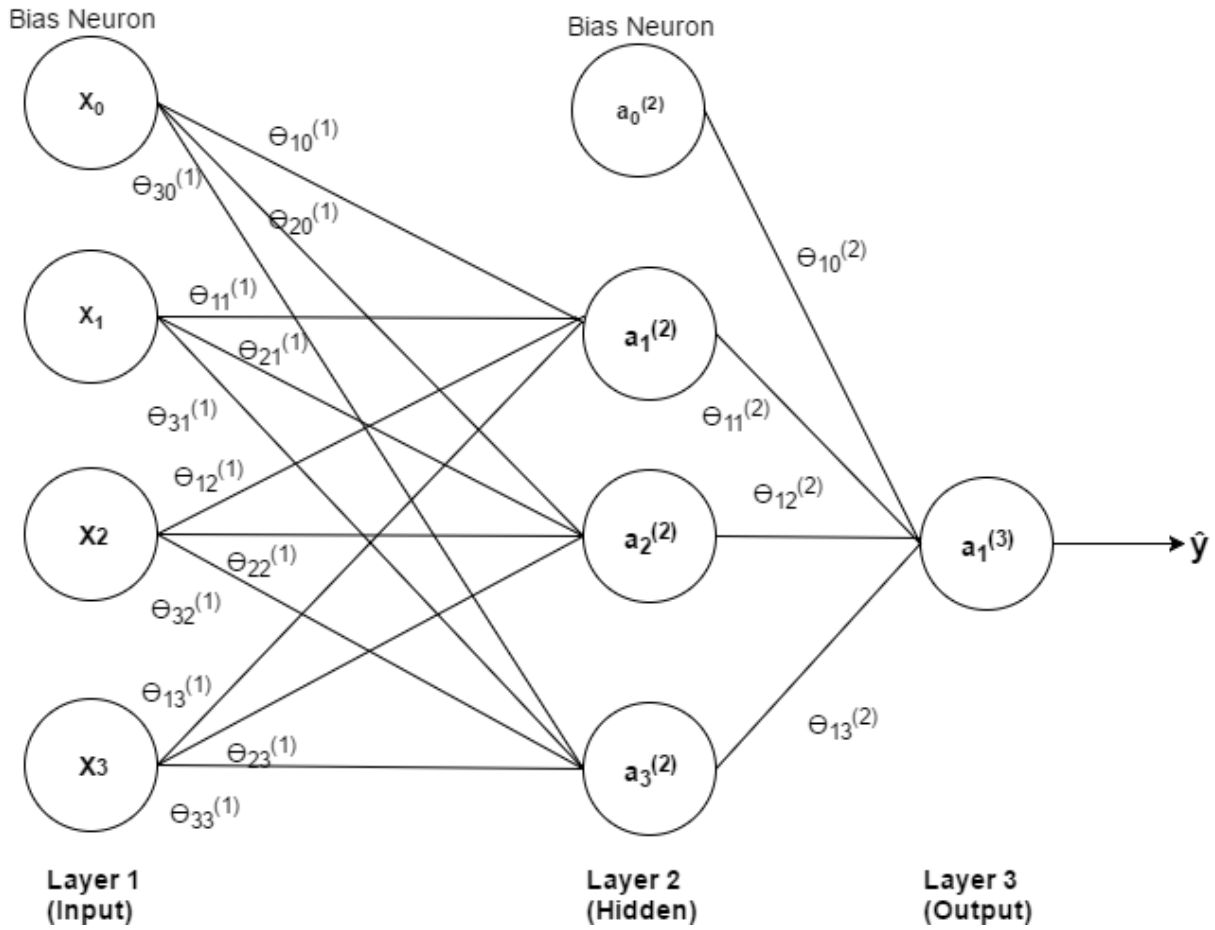

Figure 5: A basic artificial neural network

Notations Used [2]:

L = # of layers in network

M = # of training examples

N = # of features

X = Input variables/Features

Y = Output/Target variables

(x,y) = One training example

$(x^{(i)},y^{(i)})$ = i^{th} training example where $1 \leq i \leq M$

K = # of nodes (neurons) in layer 'l'

Θ = Parameters or weights

S_l = # of neurons/units (not counting bias neuron) in layer 'l'

X_j = j^{th} input feature where $1 \leq j \leq N$

$\Theta^{(j)}$ = Matrix of weights controlling function mapping from layer 'j' to layer 'j+1'

$a_i^{(j)}$ = Activation of unit 'i' in layer 'j'.

δ_j^l = Error of node 'j' in layer 'l'

In the above ANN diagram we have L=3 layers, N=3 features, one bias neuron each at layer 1 (x_0) and layer 2 ($a_0^{(2)}$) respectively, (4 X 1) input matrix 'X', (3 X 4) parameter matrix 'Θ' in layer 1 and (1 X 4) parameter matrix 'Θ' in layer2 (including bias neuron).

The below representation in the matrix form

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \Theta^{(1)} = \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix}, \Theta^{(2)} = \begin{bmatrix} \Theta_{10}^{(2)} & \Theta_{11}^{(2)} & \Theta_{12}^{(2)} & \Theta_{13}^{(2)} \end{bmatrix}$$

1. **Forward Propagation [3]:** The following steps depicts the computation of forward propagation for the neural network.

At layer 1: $Z^{(2)} = \Theta^{(1)}.\ X$ **Equation (1)**

The matrix multiplication of the same is shown below

$$Z^{(2)} = \Theta^{(1)}.\boldsymbol{X} = \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix} . \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$Z^{(2)} = \Theta^{(1)}.\boldsymbol{X}$$

$$= \begin{bmatrix} (\Theta_{\mathbf{10}}^{(1)} x_0 + \Theta_{\mathbf{11}}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\ (\Theta_{\mathbf{20}}^{(1)} x_0 + \Theta_{\mathbf{21}}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\ (\Theta_{\mathbf{30}}^{(1)} x_0 + \Theta_{\mathbf{31}}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \end{bmatrix}$$

Applying activation function (Ex: Sigmoid function $g(Z) = \frac{1}{1+e^{-Z}}$) on the above resultant matrix we have

At layer 2: $a^{(2)} = g(Z^{(2)})$ **Equation (2)**

$$a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

Where $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$ are

$$a_1^{(2)} = g(\Theta_{\mathbf{10}}^{(1)} x_0 + \Theta_{\mathbf{11}}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{\mathbf{20}}^{(\mathbf{1})}x_0 + \Theta_{\mathbf{21}}^{(\mathbf{1})}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{\mathbf{30}}^{(\mathbf{1})}x_0 + \Theta_{\mathbf{31}}^{(\mathbf{1})}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

Adding bias term $a_0^{(2)} = 1$: $a^{(2)} = \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$

At layer 3: $Z^{(3)} = \Theta^{(2)}.\, a^{(2)}$ **Equation (3)**

$$Z^{(3)} = \Theta^{(2)}.a^{(2)} = \begin{bmatrix} \Theta_{10}^{(2)} & \Theta_{11}^{(2)} & \Theta_{12}^{(2)} & \Theta_{13}^{(2)} \end{bmatrix}. \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} = \left[\Theta_{\mathbf{10}}^{(\mathbf{2})}a_0^{(2)} + \Theta_{\mathbf{11}}^{(\mathbf{2})}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right]$$

$$\hat{y} = a^{(3)} = g(Z^{(3)}) \quad \textbf{Equation (4)}$$

$$\hat{y} = a^{(3)} = \left[a_1^{(3)}\right]$$

Where $a_1^{(3)}$ is

$$a_1^{(3)} = g(\Theta_{\mathbf{10}}^{(\mathbf{2})}a_0^{(2)} + \Theta_{\mathbf{11}}^{(\mathbf{2})}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

The output $\hat{y}$ depends on parameter Θ, we need to set the right values for parameter Θ. we will use chain rule and gradient descent to tune in the Θ parameters.

Though cost function for classification is defined in terms of cross entropy, you may for now imagine cost $J = \frac{1}{2}(\hat{y} - y)^2$

Rate of change of 'J' w.r.t 'ŷ' is $E = (\hat{y} - y) = ((f(f(X.\Theta^{(1)}).\Theta^{(2)})) - y)^2$ we will use chain rule to back propagate this error to tune in the Θ parameters.

Before understanding back propagation details we will first understand chain rule, matrix differentiation and sigmoid function differentiation.

2. **Chain Rule:** Let *y = f(g(x))* be any function; we would like to know how 'y' varies with 'x' then the chain rule differentiation can be represented as follows

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} . \frac{\partial u}{\partial x}$$

Let's consider an easy example to drive the point ***f(a,b,c) = (a + b) * c,*** this can be illustrated in following circuit diagram.

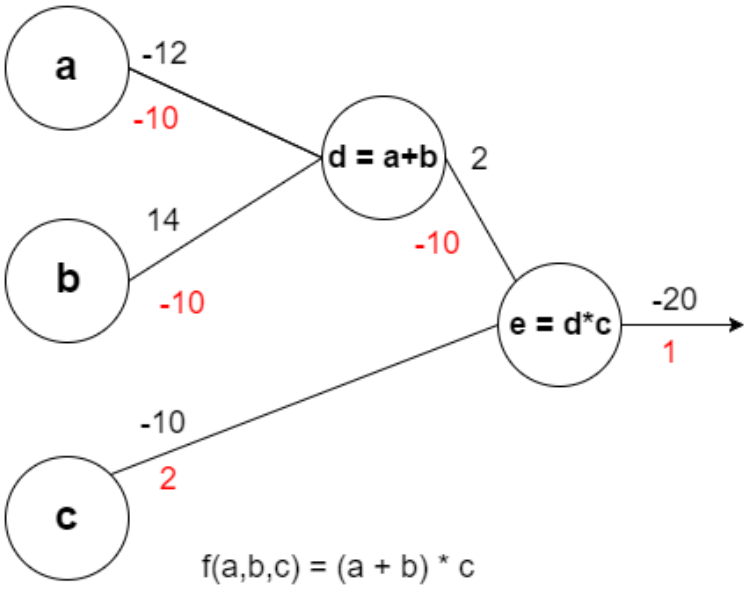

Figure 6: Network for function ***f(a,b,c) = (a + b) * c***

Forward Propagation: d = a + b = -12+14 = 2 and e = d * c = 2 * (-10) = -20

Backward Propagation: Computing backward error propagation by using chain rule

$$\frac{\partial e}{\partial c} = \frac{\partial (d * c)}{\partial c} = d = 2$$

$$\frac{\partial e}{\partial d} = \frac{\partial (d * c)}{\partial d} = c = -10$$

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial d} * \frac{\partial d}{\partial a} = -10 * \frac{\partial (a+b)}{\partial a} = -10 * 1 = -10$$

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial d} * \frac{\partial d}{\partial b} = -10 * \frac{\partial (a+b)}{\partial b} = -10 * 1 = -10$$

The significance of chain rule is, in order to drive 'e' upwards then we need to drive 'c' up in local vicinity and don't touch a or b.

3. **Matrix/Vector differentiation**[4]: The following derivation illustrates how partial derivatives are applied with respect to vector for the equation $\boldsymbol{y = X^T X}$

Let $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, X^T = [x_1 \quad x_2]$

$$\frac{\partial y}{\partial x} = \frac{\partial [x_1 \quad x_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}{\partial x}$$

$$\frac{\partial y}{\partial x} = \frac{\partial [{x_1}^2 + {x_2}^2]}{\partial x}$$

Differentiate w.r.t $x_1 and\ x_2$ $\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \end{bmatrix}$

$$\frac{\partial y}{\partial x} = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} = 2 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 2X$$

4. **Sigmoid function differentiation**: This function $g(Z) = \frac{1}{1+e^{-Z}}$ differentiation are expressed in terms of function itself.

$$\frac{\partial g}{\partial z} = g'(Z) = \frac{e^{-Z} * \frac{\partial(1)}{\partial z} - 1 * \frac{\partial(e^{-Z})}{\partial z}}{(1+e^{-Z})^2}$$

$$\frac{\partial g}{\partial z} = g'(Z) = \frac{e^{-Z}}{(1+e^{-Z})^2}$$

$$\frac{\partial g}{\partial z} = g'(Z) = \frac{e^{-Z}}{(1+e^{-Z})} * \frac{1}{(1+e^{-Z})}$$

$$\frac{\partial g}{\partial z} = g'(Z) = \frac{1+e^{-Z}-1}{(1+e^{-Z})} * \frac{1}{(1+e^{-Z})}$$

$$\frac{\partial g}{\partial z} = g'(Z) = \left(1 - \frac{1}{(1+e^{-Z})}\right) * \frac{1}{(1+e^{-Z})}$$

$$\frac{\partial g}{\partial z} = g'(Z) = \big(1 - g(Z)\big) * g(Z)$$

5. **Backward Propagation:** We need to compute rate of change of 'J' with respect to 'Θ' parameters ($\Theta^{(1)}$and $\Theta^{(2)}$ each of which is a matrix indicating each weights connected nodes in two different layers).The function $(\widehat{y} - y)$ is a composite function that depends on Θ we need to compute partial derivatives.
Here we are considering all the training examples for each iteration, in deep learning class we will study in depth how to make the optimization algorithm better. We need to compute partial derivatives to find how $\hat{y}$ changes with respect to Θ.

Given classification value 'y' and output $\hat{y}$ let's compute error terms for each layer in vectorized form.

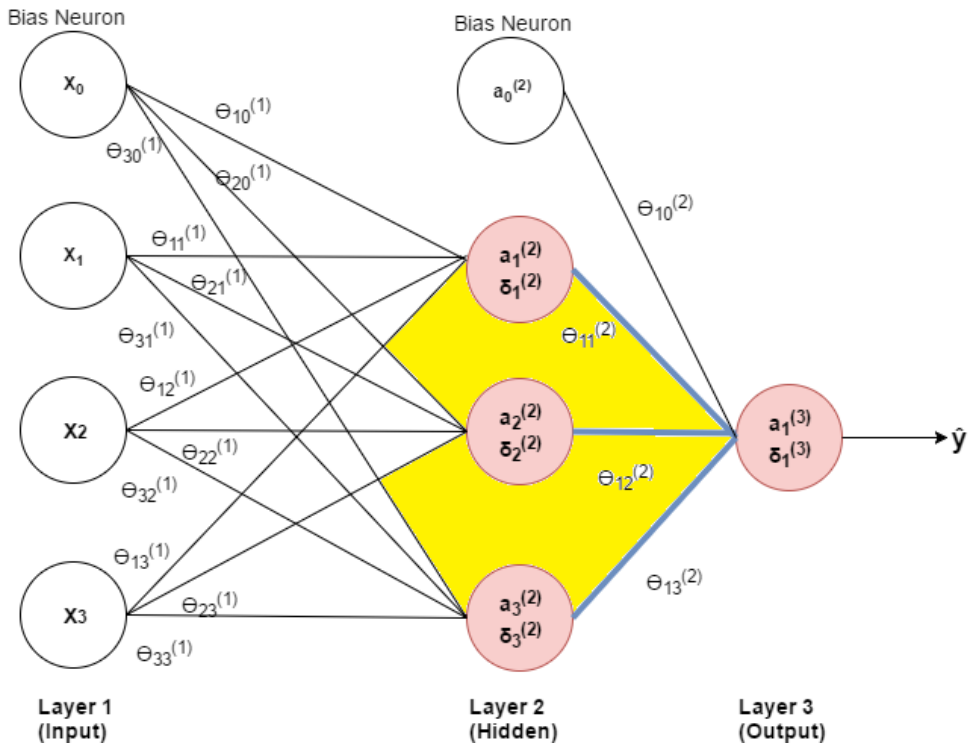

Figure 7: Computing backward propagation w.r.t $\Theta^{(2)}$

Error at Layer 3:

$$\delta_1^{(3)} = \left(a_1^{(3)} - \mathrm{y}^{(i)}\right)$$

Error at Layer 2:

$$\delta_1^{(2)} = \Theta_{11}^{(2)} . \delta_1^{(3)}$$
$$\delta_2^{(2)} = \Theta_{12}^{(2)} . \delta_1^{(3)}$$
$$\delta_3^{(2)} = \Theta_{13}^{(2)} . \delta_1^{(3)}$$

No error term computation for bias neuron i.e $\delta_0^{(2)}$
Generalized into vectorized form.

$$g'\left(Z^{(2)}\right) = a^{(2)} .* \left(1 - a^{(2)}\right)$$

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T . \delta^{(3)} .* g'\left(Z^{(2)}\right)$$

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T . \delta^{(3)} .* \left(a^{(2)} .* \left(1 - a^{(2)}\right)\right)$$

a) $\Theta^{(2)}$ is the vector of parameters for the 2->3 layer mapping
b) $\delta^{(3)}$ is the error value for Layer 3.
c) $g'\left(Z^{(2)}\right)$ is the first derivative of the activation function g evaluated by the input values given by $Z^{(2)}$
d) **. *** is the element wise multiplication between the two vectors.
e) Since the error in the output of one node is sucked in from the errors of the subsequent layer nodes weighted by theta edge we need to transpose $\Theta^{(2)}$ which is $\left(\Theta^{(2)}\right)^T$.

f) Let's consider the dimensionality of $\Theta^{(2)}$ is (1X3) without bias term [(1X4) with bias term], $\delta^{(3)}$is (1X1) and $(\Theta^{(2)})^T$ is (3X1), so when we multiply $(\Theta^{(2)})^T . \delta^{(3)}$ which are (3X1) with (1X1) will result in to (3X1) is the same dimensionality as $a^{(2)}$

No error at Layer 1 as this is the input layer.

We get all this δ terms, we want the δ terms to get the partial derivative of Θ with respect to individual parameters.

Generalizing above equation we have

$$\frac{\partial}{\partial \Theta^{(l)}} J(\Theta) = a^{(l)} . \delta^{(l+1)}$$

By doing back propagation and computing the δ terms you can then compute the **partial derivative terms.** We need the partial derivatives to minimize the cost function.

Backpropagation Algorithm:

Training set of m examples: {($x^{(1)}$, $y^{(1)}$), ($x^{(2)}$, $y^{(2)}$), ……., ($x^{(m)}$, $y^{(m)}$)}
Set $\Delta_{ij}^{l} = 0 \ (for\ all\ l, i, j)$
For i=1 to m
Set $a^{(1)} = x^{(i)}$
Perform forward propagation to compute $a^{(1)}$ for l=2,3,….,L
Using $y^{(i)}$ compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$
$\Delta_{ij}^{l} = \Delta_{ij}^{l} + a^{(l)} . \delta^{(l+1)}$

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

$$D_{ij}^{l} = \frac{1}{m}\Delta_{ij}^{l} \qquad if\ j = 0$$ (No regularization term)

We have calculated D terms above using Δ which is $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$ we can use this in gradient descent or any other advanced optimization techniques.

Conclusion: The above steps illustrates the mathematical intuition of forward and backward propagation of a given sample ANN diagram. We also got intuition behind how Θ parameters are learned by itself which in turn helps to approximate non-linear functions. The same concept can be extended to N hidden layers between input and output layer.

References:

[1]: http://sebastianraschka.com/Articles/2014_kernel_pca.html

[2]: http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning

[3]:http://www.welchlabs.com/blog

[4]:http://www.atmos.washington.edu/~dennis/MatrixCalculus.pdf