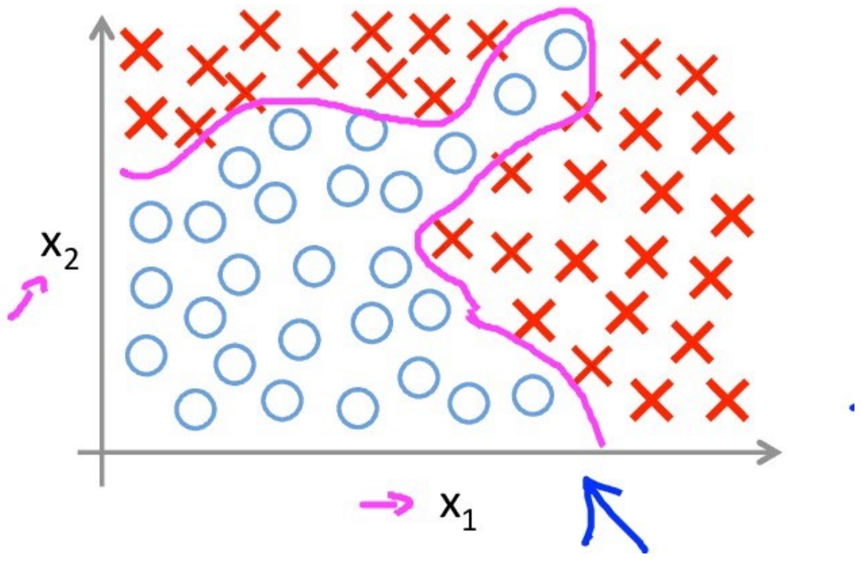
Neural Network - Representation

1. Non-linear Hypotheses

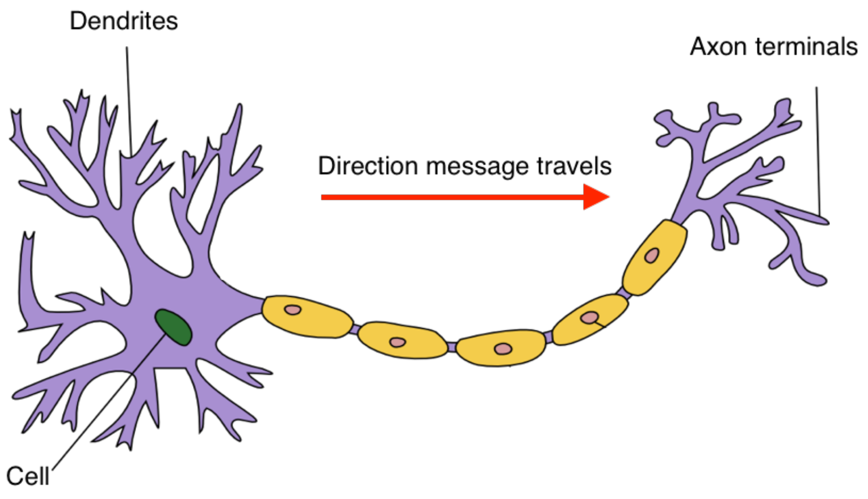
Consider a highly non-linear classification task, say something similar to the one shown in the plot below. In order to achieve a decision boundary like the one plotted, one needs to introduce non-linear features in the form of quadratic and other higher order terms.



As the number of features increase then number of terms in the hypotheses would also increase exponentially to get a good fit which would have high probability of overfitting the data. Hence, when the number of features is really high and the decision boundary is complex, logistic regression would not generalize the solution very well by leveraging the power of polynomial terms. So for highly complex tasks like the ones where one needs to classify objects from images, logistic regression would not perform well.

2. Neurons and the brain

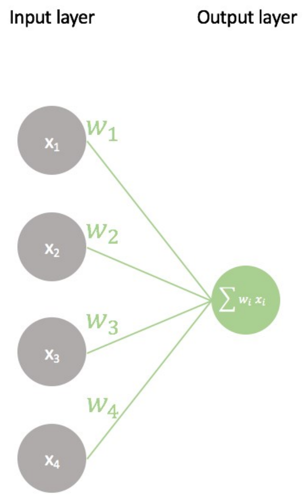
Neural Network,as the name suggests it draws inspiration from neurons in our brain and the way they are connected. Each neuron acts as a computational unit, accepting input from the dendrites and outputting signal through the axon terminals. Actions are triggered when a specific combination of neurons are activated. In essence, the cell acts a function in which we provide input (via the dendrites) and the cell churns out an output (via the axon terminals). The whole idea behind neural networks is finding a way to represent this function and connect neurons together in a useful way.



3. Perceptron

Perceptron is a single layer neural network and a multi-layer perceptron is called Neural Networks.Perceptron is a binary classifier. Also, it is used in supervised learning. It helps to classify the given input data.

The perceptron is the simplest neural unit that we can build. It takes a series of inputs, x_i , combined with a series of weights, w_i , which are compared against a threshold value, θ . If the linear combination of inputs and weights is higher than the threshold, the neuron fires, and if the combination is less than the threshold it doesn't fire.



Perceptron Unit

$$f_w(x) = \begin{cases} \sum w_i x_i \geq \theta \rightarrow \text{neuron fires} \\ \sum w_i x_i < \theta \rightarrow \text{neuron doesn't fire} \end{cases}$$

At a high level, Logistic Regression and Neural Networks are practically identical - the main difference being the activation function, $g(z)$, used to control neuron firing. The perceptron activation is a step-function from 0 (when the neuron doesn't fire) to 1 (when the neuron fires) while the logistic regression model has a smoother activation function with values ranging from 0 to 1."

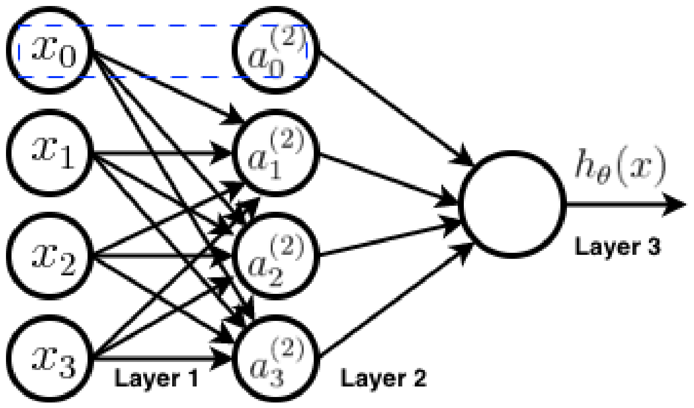
4. Model Representation

A neural network is a simplified model of the way the human brain processes information. It works by simulating a large number of interconnected processing units that resemble abstract versions of neurons. The processing units are arranged in layers. There are typically three parts in a neural network: an input layer, with units representing the input fields; one or more hidden layers; and an output layer, with a unit or units representing the target field(s). The units are connected with varying connection strengths (or weights). Input data are presented to the first layer, and values are propagated from each neuron to every neuron in the next layer. Eventually, a result is delivered from the output layer.

The network learns by examining individual records, generating a prediction for each record, and making adjustments to the weights whenever it makes an incorrect prediction. This process is repeated many times, and the network continues to improve its predictions until one or more of the stopping criteria have been met.

Initially, all weights are random, and the answers that come out of the net are probably nonsensical. The network learns through training. Examples for which the output is known are repeatedly presented to the network, and the answers it gives

Below is a representation of neural network,



where network has 3 layers. layer 1 is called input layer, layer 3 is called output layer and the remaining layers are called hidden layers. In this case there is only one hidden layer (layer 2).

- x_0 and $a_0^{(2)}$ are the bias terms and equal 1 always. They are generally not counted when the number of units in a layer are calculated.
- Thus, layer 1 and layer 2 have 3 units each.
- $a_i^{(j)}$ is the activation of unit i in layer j
- $\theta^{(j)}$ is the matrix of weights controlling function mapping from layer j to layer j+1.
- From the equation above one can generalize, if a network has s_j units in layer j and s_{j+1} units in layer j+1, then $\theta^{(j)}$ is a matrix of dimension $(s_{j+1} * (s_j + 1))$.

$$\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})
\end{aligned}$$

$\Theta_{13}^{(k)}$ means:

- 1 - we're mapping to node 1 in layer k+1

Vectorized Implementation

In this section we'll do a vectorized implementation of the above functions. We're going to define a new variable $z_k^{(j)}$ that encompasses the parameters inside our g function. In our previous example, if we replaced by the variable z for all the parameters we would get:

$a_1^{(2)}$ = $g(z_1^{(2)})$

$a_2^{(2)}$ = $g(z_2^{(2)})$

$a_3^{(2)}$ = $g(z_3^{(2)})$

In other words, for layer j=2 and node k, the variable z will be:

$z_k^{(2)}$ = $\Theta_{k,0}^{(1)}$ x_0 + $\Theta_{k,1}^{(1)}$ x_1 + ... + $\Theta_{k,n}^{(1)}$ x_n

The vector representation of x and z_j is :

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix}, z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \cdots \\ z_n^{(j)} \end{bmatrix}$$

Setting x = $a^{(1)}$, we can rewrite the equation as :

$z^{(j)} = \Theta^{(j-1)} a^{(j-1)}$

We are multiplying our matrix $\Theta^{(j-1)}$ with dimensions $s_j * (n+1)$ (where s_j is number of our activation nodes) by our vector $a^{(j-1)}$ with height (n+1). This gives us our vector $z(j)$ with height s_j . Now we can get a vector of our activation nodes for layer j as follows:

We can then add a bias unit (equal to 1) to layer j after we have computed $a^{(j)}$. This will be element $a_0^{(j)}$ and will be equal to 1.To compute our final hypothesis, let's first compute another z vector:

$z^{(j+1)}$ = $\Theta^{(j)}$ $a^{(j)}$

We get this final z vector by multiplying the next theta matrix after $\Theta^{(j-1)}$ with the values of all the activation nodes we just got. This last theta matrix $\Theta^{(j)}$ will have only one row which is multiplied by one column $a^{(j)}$ so that our result is a single number. We then get the final result with:

$h_\Theta(x)$ = $a^{(j+1)}$ = $g(z^{(j+1)})$

Notice that in this last step, between layer j and layer j+1, we are doing exactly the same thing as in logistic regression. Adding all these intermediate layers in neural networks allows us to more elegantly produce interesting and more complex non-linear hypotheses.

5. Intuition behind Neural Network

A simple example of applying neural networks is by predicting x 1 AND x 2, which is the logical 'AND' operator and is only true if both x 1 and x 2 are 1. The graph of our functions will look like:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \left[g(z^{(2)}) \right] \rightarrow h_\Theta(x)$$

Remember that x_0 is our bias variable and is always 1. Let's set our first theta matrix as:

$\Theta^{(1)}$ = $[$ -30 20 20 $]$

This will cause the output of our hypothesis to only be positive if both x1 and x2 are 1. In other words,

$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$

$$\begin{aligned}
&x_1 = 0 \;\; and \;\; x_2 = 0 \;\; then \;\; g(-30) \approx 0 \\
&x_1 = 0 \;\; and \;\; x_2 = 1 \;\; then \;\; g(-10) \approx 0 \\
&x_1 = 1 \;\; and \;\; x_2 = 0 \;\; then \;\; g(-10) \approx 0 \\
&x_1 = 1 \;\; and \;\; x_2 = 1 \;\; then \;\; g(10) \approx 1
\end{aligned}$$

So we have constructed one of the fundamental operations in computers by using a small neural network rather than using an actual AND gate. Neural networks can also be used to simulate all the other logical gates. The $\Theta^{(1)}$ matrices for AND, OR and NOR are:

AND : $\Theta^{(1)}$ = $[$ -30 20 20 $]$

OR : $\Theta^{(1)}$ = $[$ -10 20 20 $]$

NOR : $\Theta^{(1)}$ = $[$ 10 -20 -20 $]$

We can combine these to get the XNOR logical operator (which gives 1 if x 1 and x 2 are both 0 or both 1).

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix} \rightarrow \begin{bmatrix} a^{(3)} \end{bmatrix} \rightarrow h_\Theta(x)$$

For the transition between the first and second layer, we'll use a $\Theta^{(1)}$ matrix that

$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$$

For transition between the second and third layer, we'll use a $\Theta^{(2)}$ matrix that uses the value for OR:

$$\Theta^{(2)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix}$$

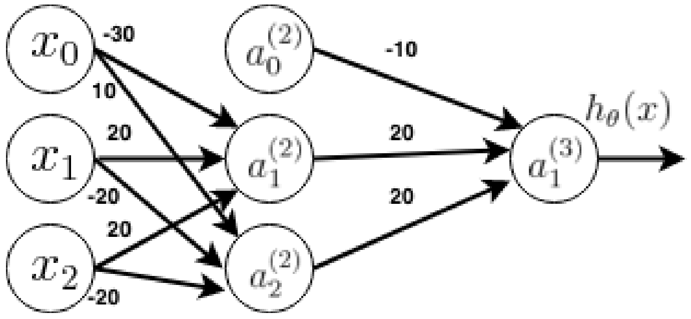
Let's write out the values for all our nodes:

$a^{(2)} = g(\Theta^{(1)}.x)$

$a^{(3)} = g(\Theta^{(2)}.a^{(2)})$

$h_\Theta(x) = a^{(3)}$

And here we have the XNOR operator using a hidden layer with two nodes! The following summarizes the above algorithm:



```
In [ ]:
```

```
In [ ]:
```