Neural Networks:

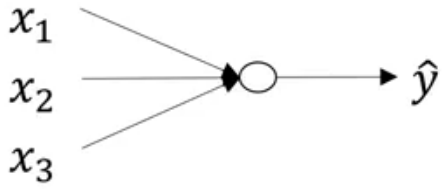
Why Neural Networks?
Neural Networks, or the reason for it's conception was borne out of the observation that our brains are essentially biological supercomputers that are able to learn and perform increasingly complex tasks.

This was observed through numerous experiments and it was found that any neuron in our brain has the same capacity to learn and understand how to perform tasks.

Hence it was hypothesized that there exists within our genome, a learning algorithm universal to all neurons. Hence, neural networks are an attempt to model this learning algorithm, or approximate it.

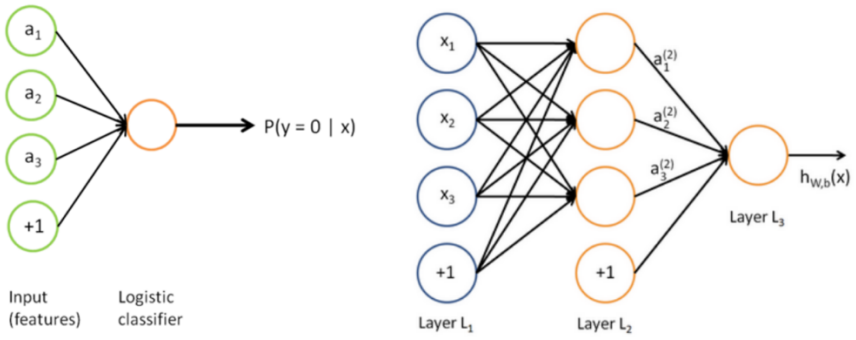
Furthermore, Logistic Regression and Polynomial Regression is not scalable when there are a large number of features, as the polynomial involved in the decision boundary become increasingly complex and computationally expensive to compute.

Model representation:
Logistic Regression as a single neuron:
A neuron can be said to take in input from the sensory input layer and apply a set of parameters ie "weights" to the inputs to evaluate the output.
Hence the sigmoid function could be said to act as an activation function, that evaluates the features of the input and gives an output.



logistic regression

Hence, when stuck together with other such neurons it is termed as a neural network.

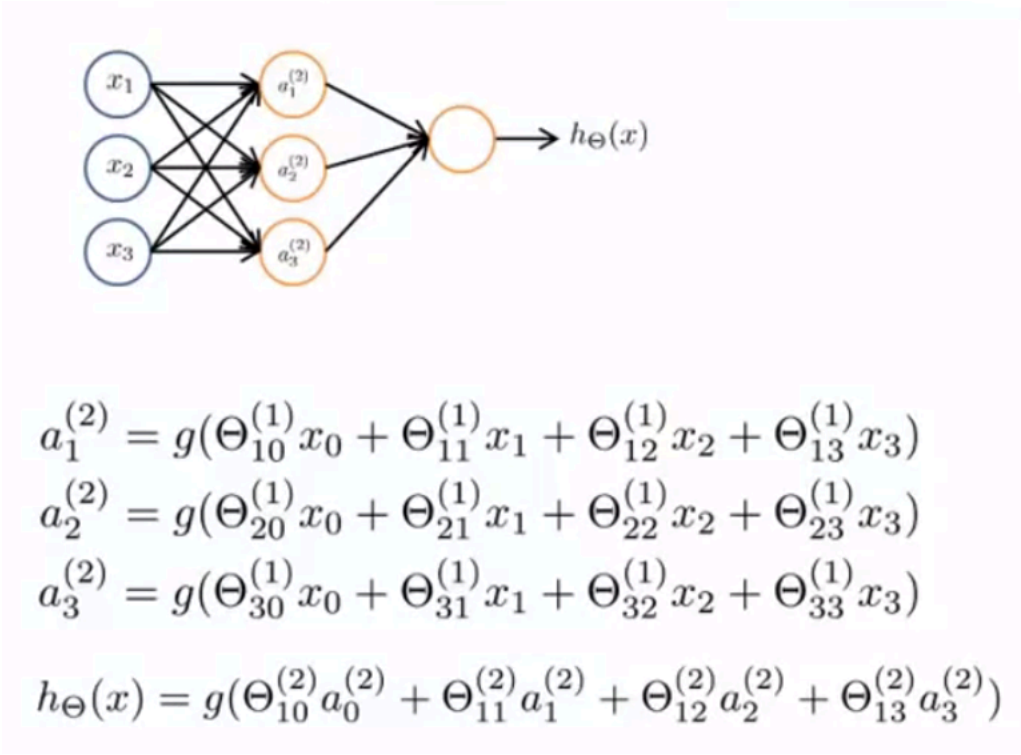


An neural network used for classification has a few defining features.

1) A input layer of neurons(representing the input data), starting from $x_1, \ldots, x_n$ along with a bias neuron that is always a value of 1.

2) A number of hidden layers of neurons, each overlaying over each other, hence allowing each subsequent layer to learn more complex features from the previous layer of neurons.
3) An Output Layer, a layer with m (number of classes) neurons which each act as a logistic regression prediction for that specific class in a 'ovr' mapping.

Forward Propagation:



$$z^{(2)} = X^T \Theta$$

$$a_j^{(i)} = g(X^T \Theta) \; where \; g \; is \; the \; activation \; function, in \; this \; case, the \; sigmoid \; fn$$

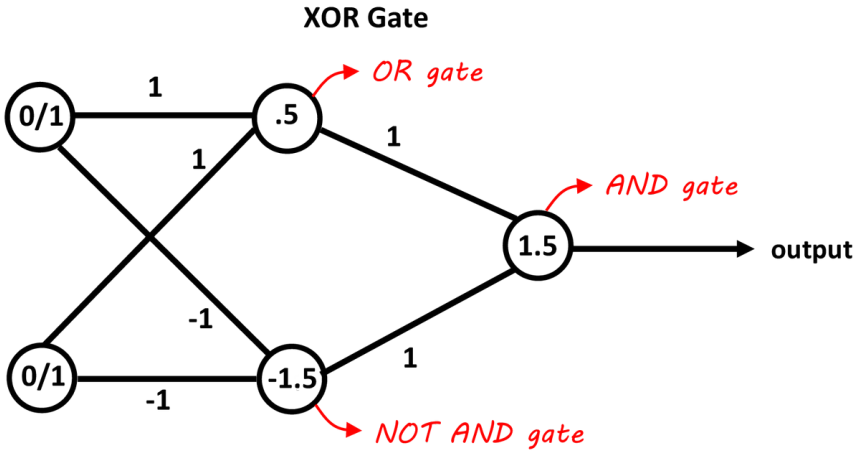
Where i = layer of the neuron, $\theta = the \; weights \; of \; that \; neuron$

$$h_\Theta(x) = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

$$where \; h_\Theta(x) \; represents \; the \; output \; of \; the \; model.$$

Note: The code from one layer to the next can be easily vectorized into matrix multiplication.

Some functions which can be modeled easily in a NN:



Cost Function:
The cost function for a NN is a generalization of the cost function used in logistic regression.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right)_k + (1-y)\log\left(1-(h_\Theta(x))_k\right.$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

$$where\ y_k\ is\ a\ one\ hot\ encoded\ vector,$$
$$m = num\ of\ training\ egs$$
$$K = number\ of\ classes$$
$$L = number\ of\ layers$$

How this function is minimized, is through backpropagation, the learning algorithm used by neural networks.

Backpropagation:
Backpropagation is quite a complex algorithm to define.
However the intuition is about the same, to calculate the "gradient" and minimize the cost function.

For t=1:m training eg

1. Set the input layer's values (a) to the t-th training example x .

Perform a feedforward pass (Figure 2), computing the activations ($z^{(2)}$, $a^{(2)}$, $z^{(3)}$, $a^{(3)}$) for layers 2 and 3. Note that you need to add a +1 term to ensure that the vectors of activations for layers $a^{(1)}$ and $a^{(2)}$ also include the bias unit.

2. For each output unit k in layer 3 (the output layer), set

$$\delta_k^{(3)} = (a_k^{(3)} - y_k),$$

where $y_k \in \{0,1\}$ indicates whether the current training example be- longs to class k ($y_k = 1$), or if it belongs to a different class ($y_k = 0$

3. Eg For the hidden layer l = 2, set

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

4. Accumulate the gradient from this example using the following formula. Note that you should skip or remove δ_0.

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

5. Obtain the (unregularized) gradient for the neural network cost function by dividing the accumulated gradients by $\frac{1}{m}$:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

Then to Regularise the output:
If j = 0, do not regularize

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} \; for \; j \geq 1$$

Gradient Checking:

Gradient Checking is a computationally expensive algorithm that computes the partial gradients of the cost functions, used to check if your implementation of backpropagation is correct.

$$f_i(\theta) \approx \frac{J(\theta^{i+}) - J(\theta^{i-})}{2\varepsilon}$$

Eg for a vector θ:

$$\theta^{i+} = \theta + \begin{bmatrix} 0 \\ \vdots \\ \varepsilon \\ \vdots \\ 0 \end{bmatrix} \; and \; \theta^{i-} = \theta - \begin{bmatrix} 0 \\ \vdots \\ \varepsilon \\ \vdots \\ 0 \end{bmatrix}$$