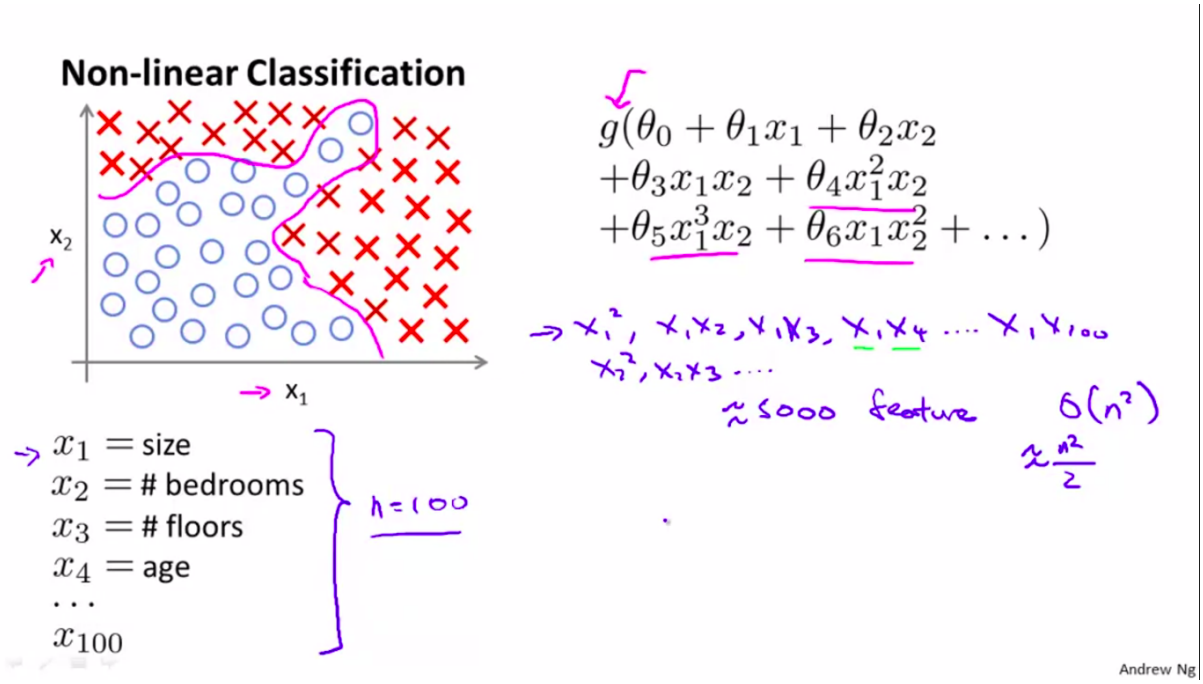
Week 4 - Lecture 1

Non-linear Hypotheses

Suppose, for instance, that we're interested in solving a classification problem similar to the one plotted below.



This problem is, essentially, not linearly separable (we cannot make a good fit just by separating points with a straight line). In cases like this, it may be more appropriate to suppose non-linear hypotheses. One could try to employ a **Logistic Regression** model to a problem like this one. This may be a good alternative in cases which we have a small number of features (say x_1, x_2 for example). However, there are many problems out there in which we have a big set of features (say x_1, x_2, $\dots$, x_{100}).

In these cases, including all cross-relation terms would increase the computational time significantly, by the order of $O(n^2)$. Additionally, you might end up overfitting the training set as you're incluting too many features to your hypothesis (in the case of 100 different features, you might end up with 5000 terms in your linear regression equation).

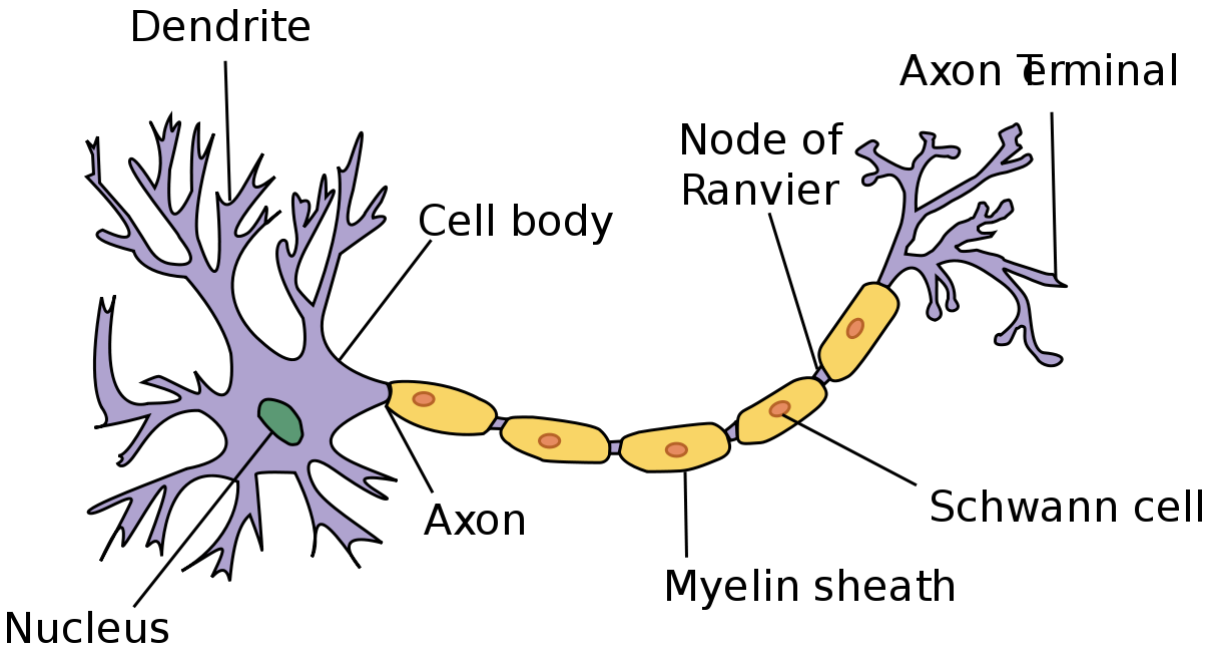
One alternative is to include only a subset of these 5000 terms, but other problems arise with that idea. First, it is difficult to say *a priori* which terms should be left aside. Additioinally, fitting the model with different subsets might also be computationally expensive.

> **Note:** To address this kind of problems (which induces "complicated" non-linear hypothesis), it is recommended to use a class of models know as **Neural Networks**.

Neural Networks

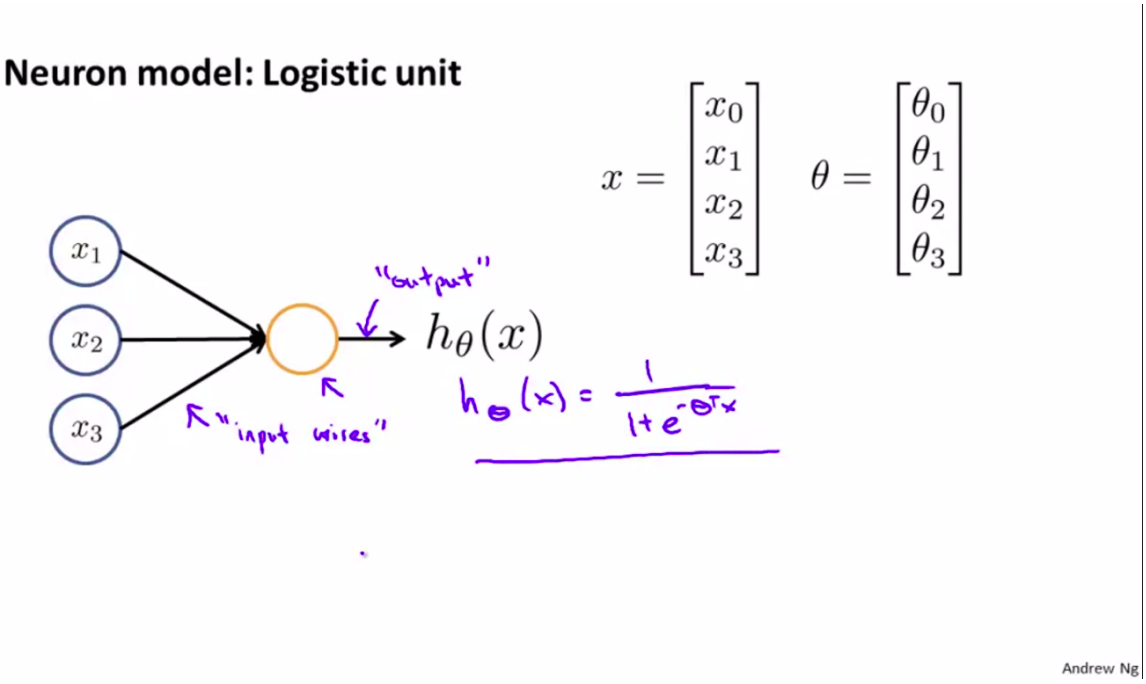
Model Representation I

Note: The following image was included just for analogy purposes.



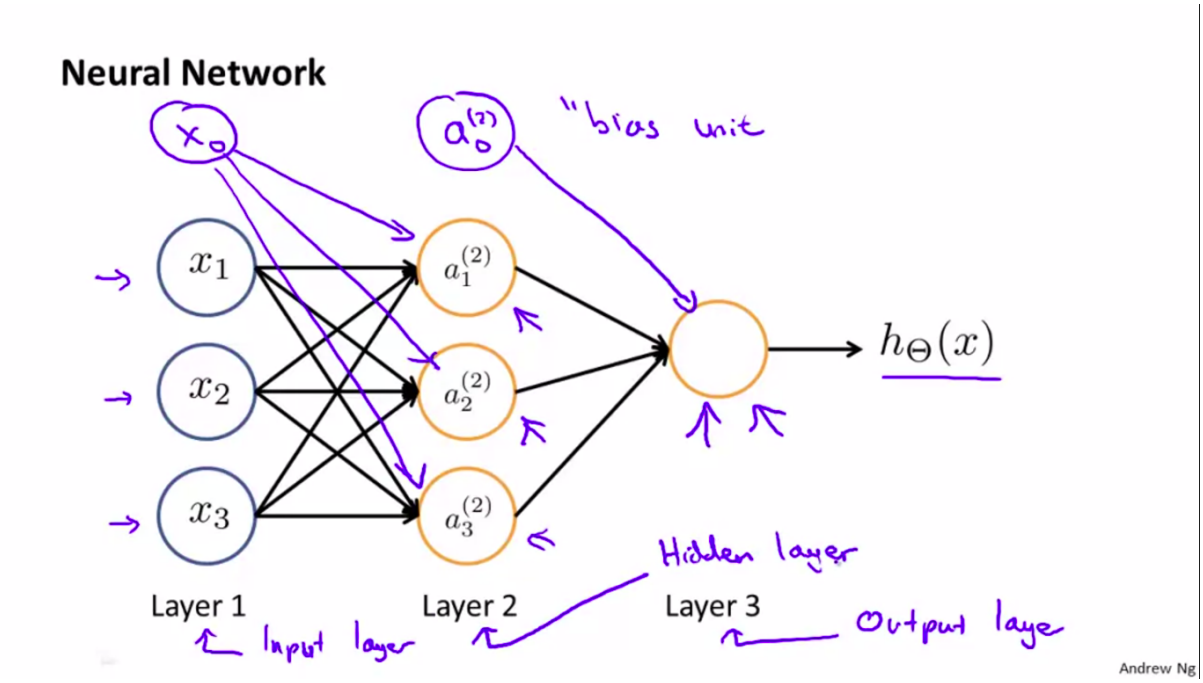
Let's examine how we will represent a hypothesis function using neural networks. At a very simple level, neurons are basically computational units that take inputs (**dendrites**) as electrical inputs (called "spikes") that are channeled to outputs (**axons**).

In our model, our dendrites are like the input features $x_1 \cdots x_n$, and the output is the result of our hypothesis function.



In this model our x_0 input node is sometimes called the "bias unit." It is always equal to 1. In neural networks, we use the same logistic function as in classification, $\frac{1}{1+e^{-\theta^T x}}$, yet we sometimes call it a **sigmoid** (logistic) **activation function**. In this situation, our "theta" parameters are sometimes called "weights". Visually, a simplistic representation looks like:

$[x_0 x_1 x_2] \rightarrow [\ ] \rightarrow h_\theta(x)$



Our input nodes (layer 1), also known as the "input layer", go into another node (layer 2), which finally outputs the hypothesis function, known as the "output layer".

We can have intermediate layers of nodes between the input and output layers called the "hidden layers."

In this example, we label these intermediate or "hidden" layer nodes $a_0^2 \cdots a_n^2$ and call them "activation units."

$$a_i^{(j)} = \text{"activation" of unit } i \text{ in layer } j.$$
$$\Theta^{(j)} = \text{matrix of weights controlling function mapping from layer } j \text{ to } j+1.$$

If we had one hidden layer, it would look like:

$$[x_0 x_1 x_2 x_3] \rightarrow [a_1^{(2)} a_2^{(2)} a_3^{(2)}] \rightarrow h_\theta(x)$$

The values for each of the "activation" nodes is obtained as follows:

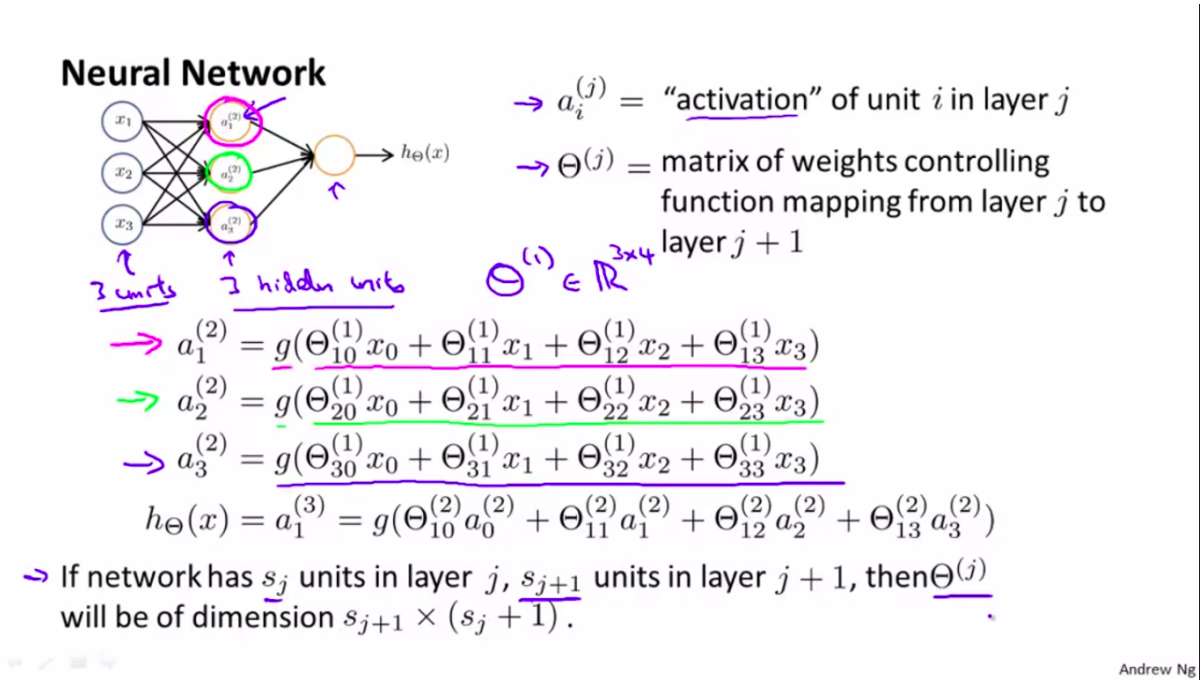
$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)}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

This is saying that we compute our activation nodes by using a 3×4 matrix of parameters. We apply each row of the parameters to our inputs to obtain the value for one activation node. Our hypothesis output is the logistic function applied to the sum of the values of our activation nodes, which have been multiplied by yet another parameter matrix $\Theta^{(2)}$ containing the weights for our second layer of nodes.

Each layer gets its own matrix of weights, $\Theta^{(j)}$. The dimensions of these matrices of weights is determined as follows:

If a network has s_j units in layer j and $s_j + 1$ units in layer $j + 1$, then $\Theta^(j)$ will be of dimension $s_{j+1} \times s_j + 1$.

The $+1$ comes from the addition in $\Theta^{(j)}$ of the "bias nodes," x_0 and $\Theta_0^{(j)}$. In other words the output nodes will not include the bias nodes while the inputs will. The following image summarizes our model representation:



Example: If layer 1 has 2 input nodes and layer 2 has 4 activation nodes. Dimension of $\Theta^{(1)}$ is going to be 4×3 where $s_j = 2$ and $s_{j+1} = 4$, so $s_{j+1} \times (s_j + 1) = 4 \times 3$.

Model Representation II

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

From the representation of a neural network stated in the previous section (repeated above), we can do a vectorized implementation of these 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:

$$a_1^{(2)} = g\left(z_1^{(2)}\right)$$
$$a_2^{(2)} = g\left(z_2^{(2)}\right)$$
$$a_3^{(2)} = g\left(z_3^{(2)}\right)$$

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

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

The vector representation of x and z^j is:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \vdots \\ z_n^{(j)} \end{bmatrix} \tag{2}$$

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

$$z^{(j)} = \Theta^{(j-1)} a^{(j-1)} \tag{3}$$

We are multiplying our matrix $\Theta^{(j-1)}$ with dimensions $s_j \times (n+1)$ (where s_j is the 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:

$$a^{(j)} = g\left(z^{(j)}\right)$$

where our function g can be applied element-wise to our vector $z^{(j)}$.

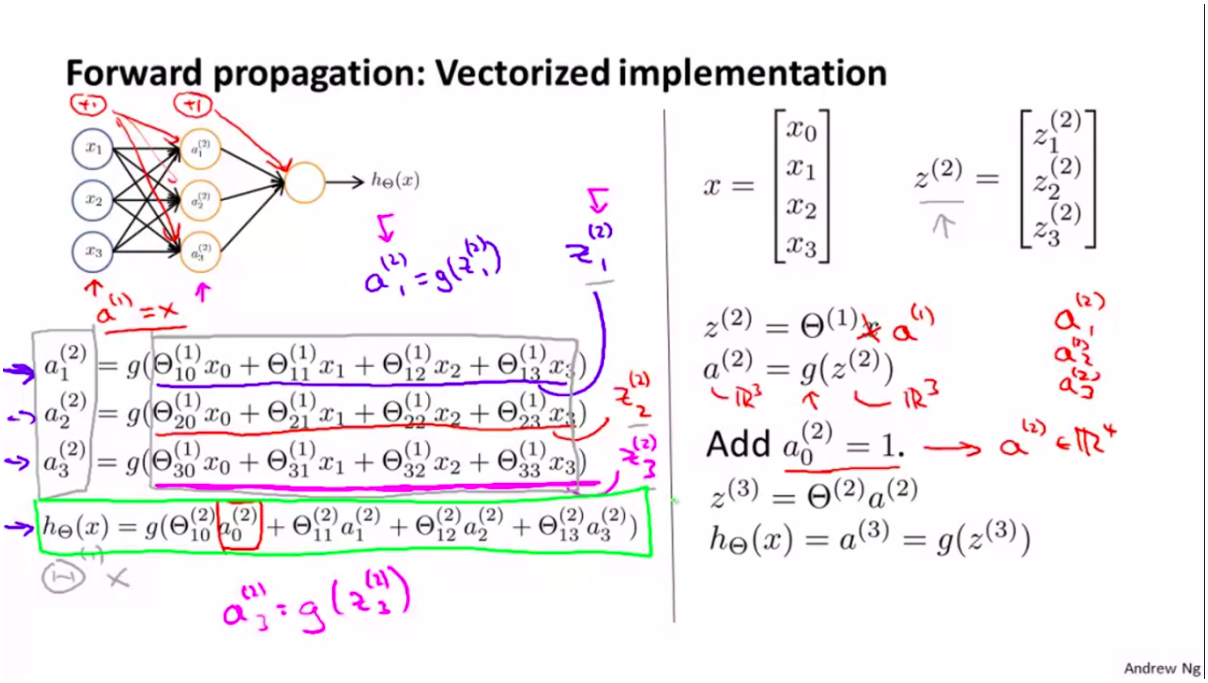
We can then add a bias unit (equal to 1) to layer j after we have computed $a^{(j)}$. This will be element $a_0^{(j)}$ and will be equal to 1. To compute our final hypothesis, let's first compute another z vector:

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

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 our final result with:

$$h_\Theta(x) = a^{(j+1)} = g\left(z^{(j+1)}\right) \tag{4}$$

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



Examples and Intuitions I

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:

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

Remember that x_0 is our bias variable and is always 1.

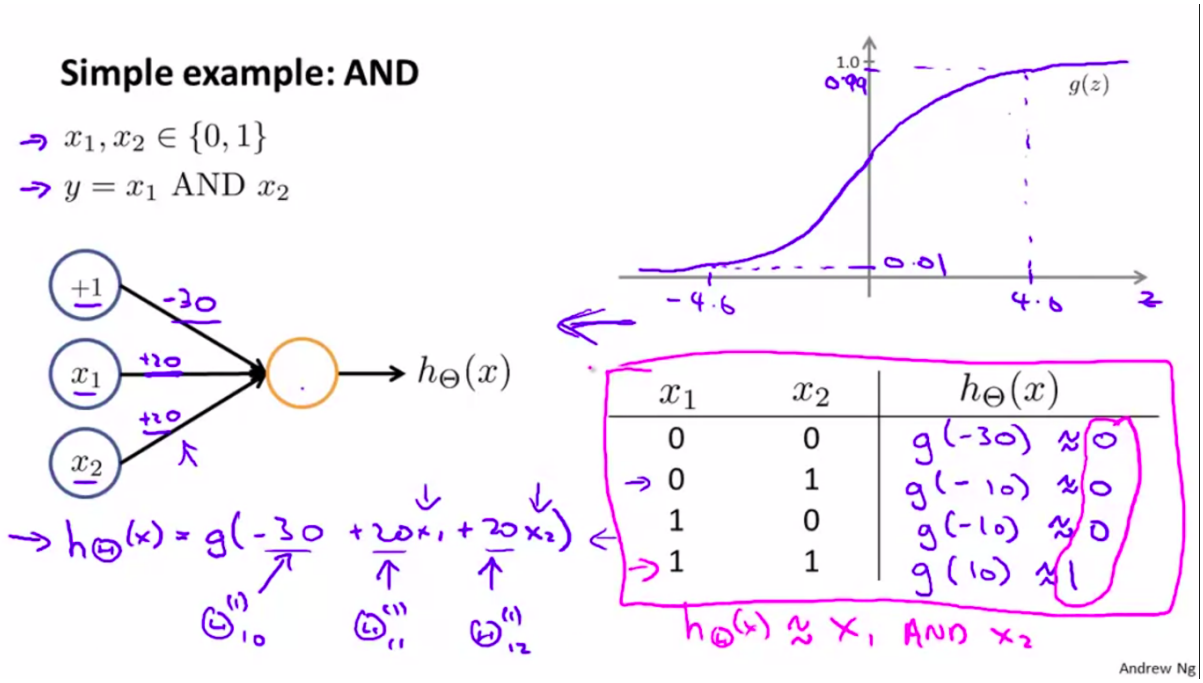
Let's set our first theta matrix as:

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

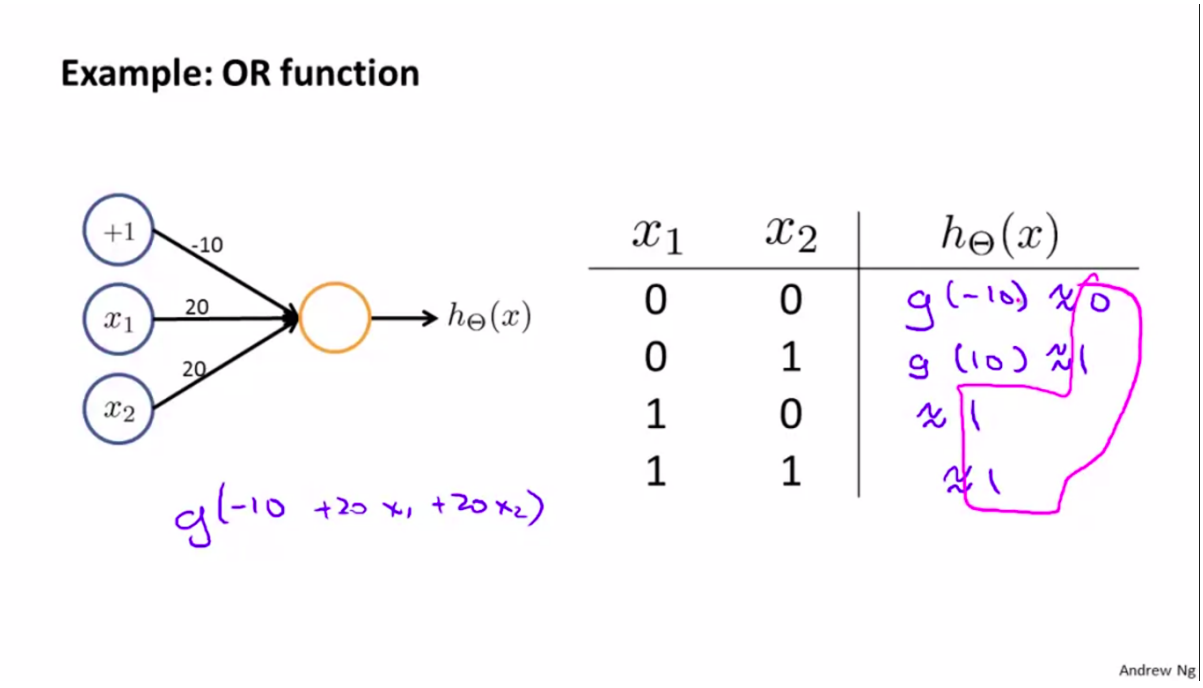
This will cause the output of our hypothesis to only be positive if both x_1 and x_2 are 1. In other words:

$$_\Theta(x) = g(-30 + 20x_1 + 20x_2)$$

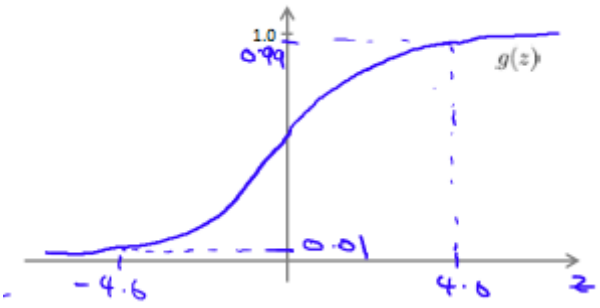
$$\begin{aligned} x_1 &= 0 \text{ and } x_2 = 0 \text{ then } g(-30) \approx 0 \\ x_1 &= 1 \text{ and } x_2 = 0 \text{ then } g(-10) \approx 0 \\ x_1 &= 0 \text{ and } x_2 = 1 \text{ then } g(-10) \approx 0 \\ x_1 &= 1 \text{ and } x_2 = 1 \text{ then } g(10) \approx 1 \end{aligned}$$



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 following is an example of the logical operator 'OR', meaning either x_1 is true or x_2 is true, or both:



Where $g(z)$ is the following:



Examples and Intuitions II

The $\Theta^{(1)}$ matrices for AND, NOR, and OR are:

$$
\begin{aligned}
AND: &\quad \Theta^{(1)} = [-30 \;\; 20 \;\; 20] \\
NOR: &\quad \Theta^{(1)} = [10 \;\; -20 \;\; -20] \\
OR: &\quad \Theta^{(1)} = [-10 \;\; 20 \;\; 20]
\end{aligned}
$$

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).

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

For the transition between the first and second layer, we'll use a $\Theta^{(1)}$ matrix that combines the values for AND and NOR:

$$
\Theta^{(1)} = [-30 \;\; 20 \;\; 2010 \;\; -20 \;\; -20]
$$

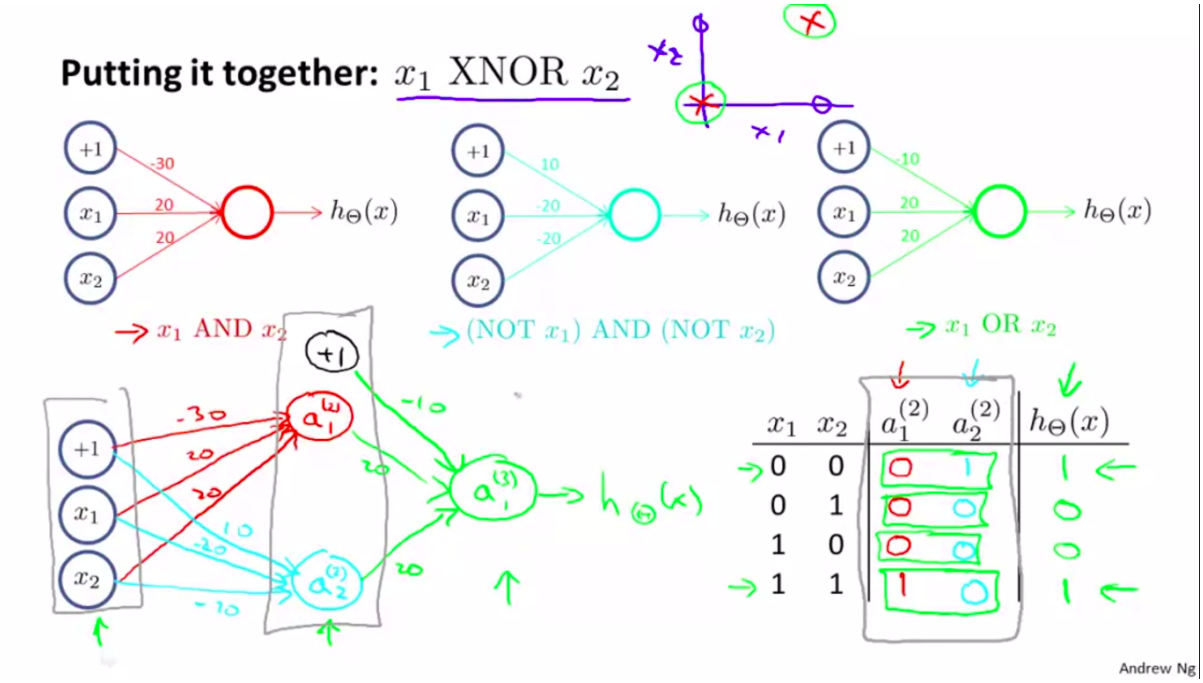
For the transition between the second and third layer, we'll use a $\Theta^{(2)}$ matrix that uses the value for OR:

$$
\Theta^{(2)} = [-10 \;\; 20 \;\; 20]
$$

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

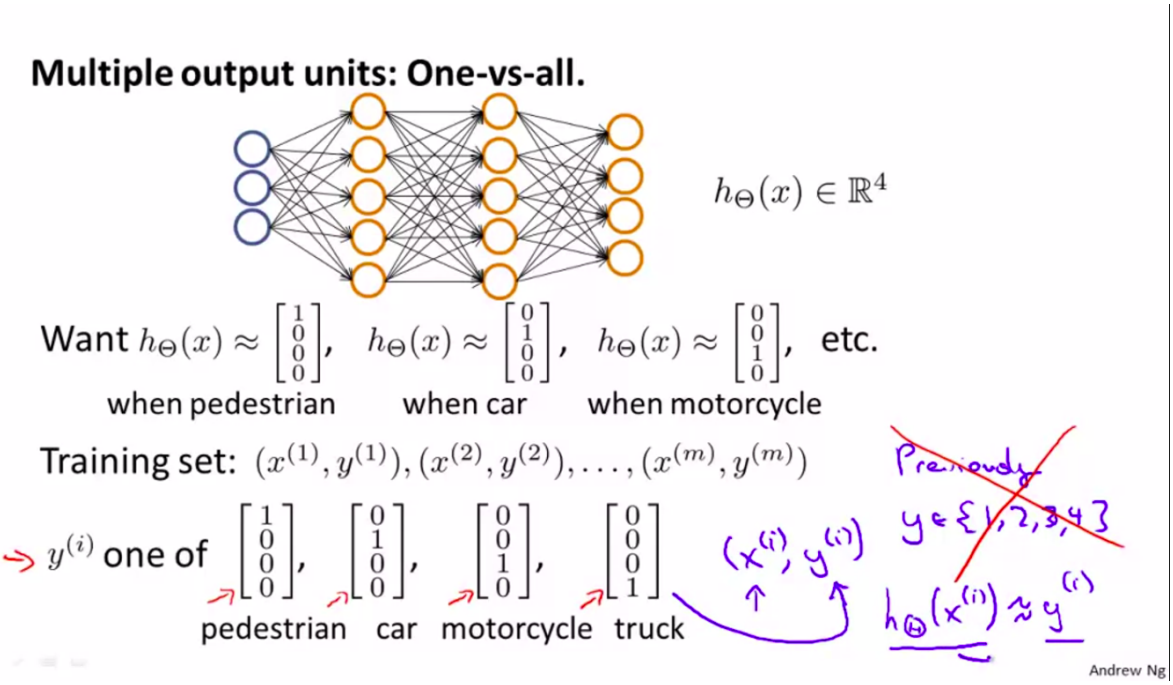
$$
\begin{aligned}
a^{(2)} &= g\left(\Theta^{(1)} \cdot x\right) \\
a^{(3)} &= g\left(\Theta^{(2)} \cdot a^{(2)}\right) \\
h_\Theta(x) &= a^{(3)}
\end{aligned}
$$

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



Multiclass Classification

To classify data into multiple classes, we let our hypothesis function return a vector of values. Say we wanted to classify our data into one of four categories. We will use the following example to see how this classification is done. This algorithm takes as input an image and classifies it accordingly:



We can define our set of resulting classes as y:

$$y^{(i)} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Each $y^{(i)}$ represents a different image corresponding to either a car, pedestrian, truck, or motorcycle. The inner layers, each provide us with some new information which leads to our final hypothesis function. The setup looks like:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ \vdots \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(3)} \\ a_1^{(3)} \\ a_2^{(3)} \\ \vdots \end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix} h_\Theta(x)_1 \\ h_\Theta(x)_2 \\ h_\Theta(x)_3 \\ h_\Theta(x)_4 \end{bmatrix}$$

Our resulting hypothesis for one set of inputs may look like:

$$h_\Theta(x) = [0 \;\; 0 \;\; 1 \;\; 0]$$

In which case our resulting class is the third one down, or $h_\Theta(x)_3$, which represents the motorcycle.

References

[1] Machine Learning - Stanford University (https://www.coursera.org/learn/machine-learning)