

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 $x_1 \dots x_n$, 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".

逻辑函数用专业术语来说又叫激励函数

参数theta用专业术语来说就是权重

Visually, a simplistic representation looks like:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow [] \rightarrow h_\theta(x)$$

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 $a_0^2 \dots a_n^2$ and call them "activation units."

$a_i^{(j)}$ = "activation" of unit i in layer j

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

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

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} \rightarrow h_\theta(x)$$

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

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

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

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

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

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

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:

$$\begin{aligned} a_1^{(2)} &= g(z_1^{(2)}) \\ a_2^{(2)} &= g(z_2^{(2)}) \\ a_3^{(2)} &= g(z_3^{(2)}) \end{aligned}$$

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

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

The vector representation of x and z^j is:

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

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 $s_j \times (n + 1)$ (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:

$$a^{(j)} = g(z^{(j)})$$

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:

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

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.

总结：神经网络的意义在于通过重复前向传递，在传递过程中自我生成新的训练集，能比一开始输入的训练集更精确更优质，因此能得出更好的假设（hypothesis）

Example 1

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:

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

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

Let's set our first theta matrix as:

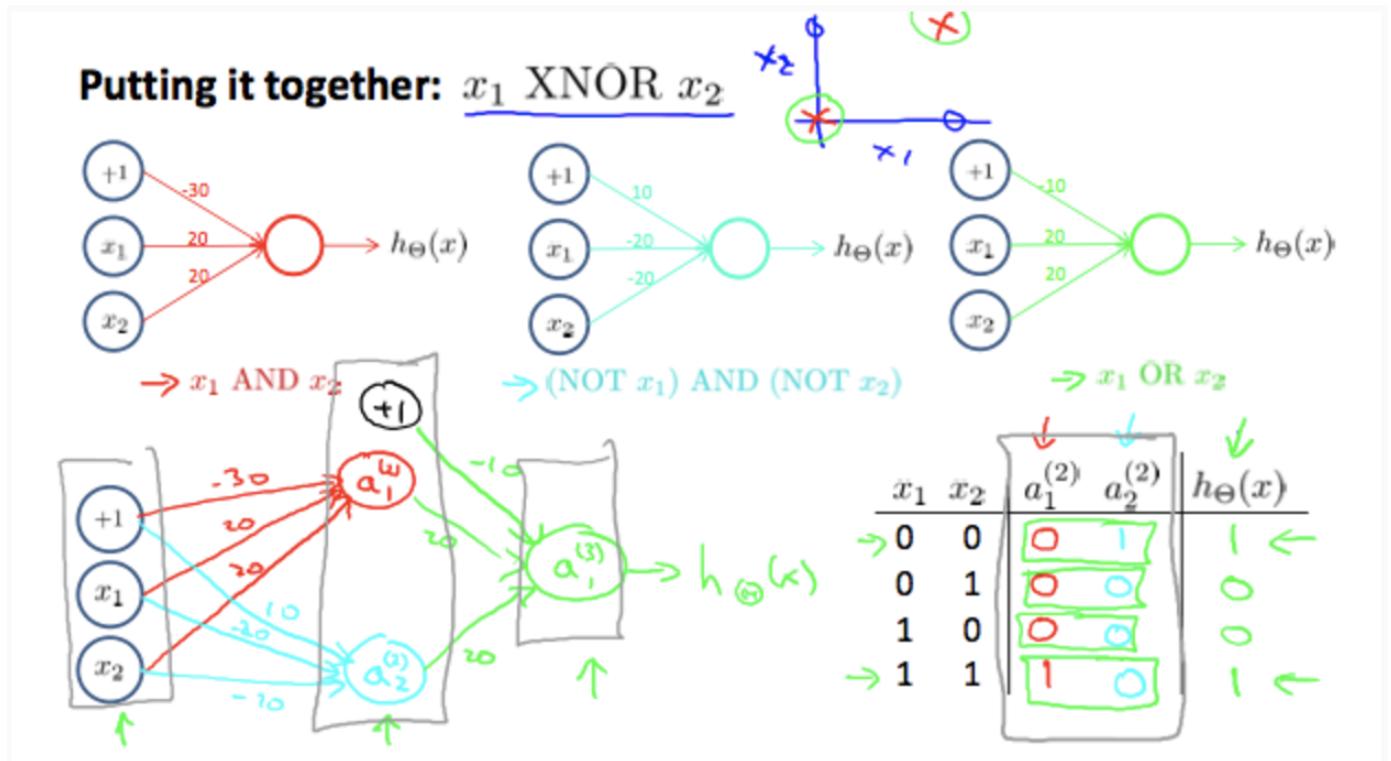
$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \end{bmatrix}$$

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

$$h_{\Theta}(x) = g(-30 + 20x_1 + 20x_2)$$

- $x_1 = 0$ 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 $g(10) \approx 1$

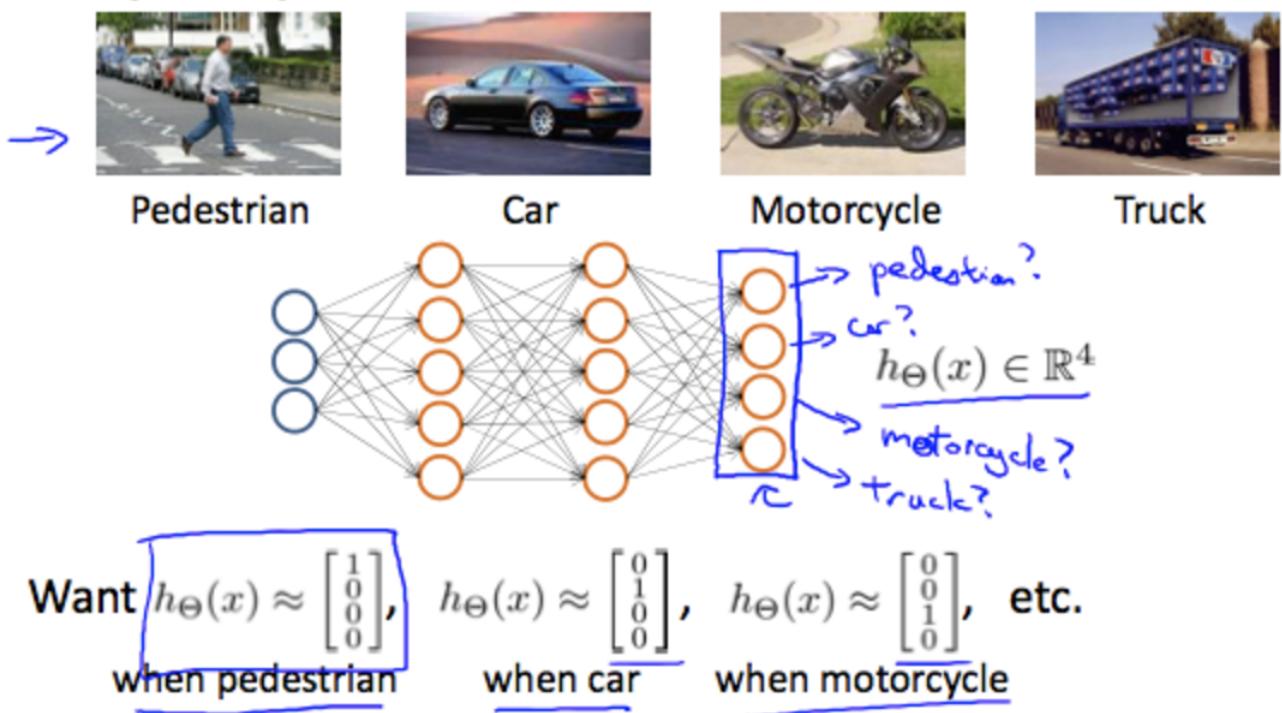
从第一层到第二层的过程中神经网络形成了较为复杂的逻辑，并最终在第三层得出了排他性或门



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:

Multiple output units: One-vs-all.



We can define our set of resulting classes as y :



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:



Octave/MATLAB Tip:

Logical arrays in Octave/MATLAB are arrays which contain binary (0 or 1) elements. In Octave/MATLAB, evaluating the expression $a == b$ for a vector a (of size $m \times 1$) and scalar b will return a vector of the same size as a with ones at positions where the elements of a are equal to b and zeroes where they are different. To see how this works for yourself, try the following code in Octave/MATLAB:

```
a = 1:10; % Create a and b  
b = 3;  
a == b % You should try different values of b here
```