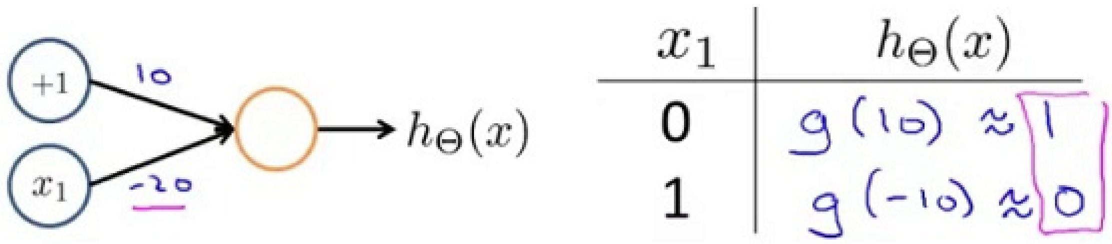
Examples and Intuitions II

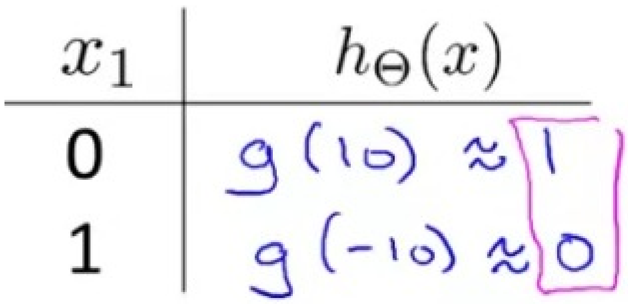
Previously we saw how a Neural Network can be used to compute the functions x_1 AND x_2, and the function x_1 OR x_2 when x_1 and x_2 are binary, that is when they take on values $0, 1$.

Negation

We can also have a network to compute negation, that is to compute the function not x_1.

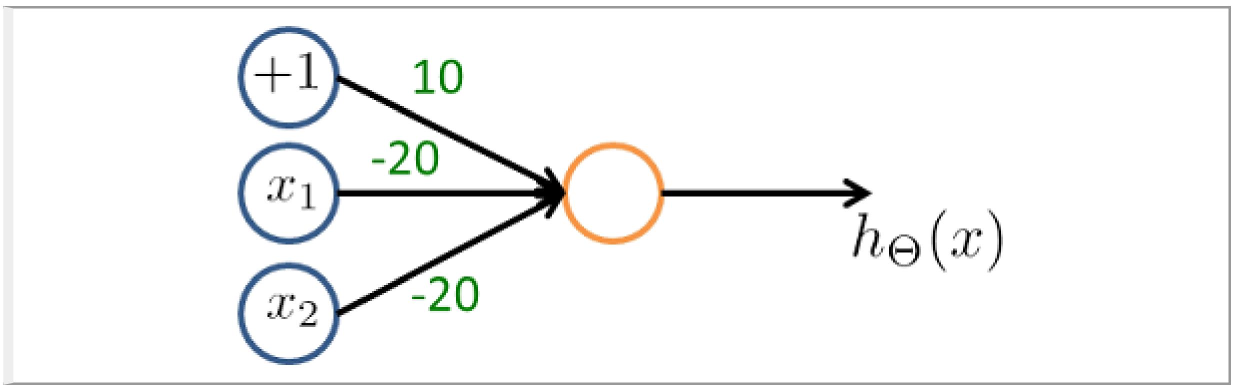


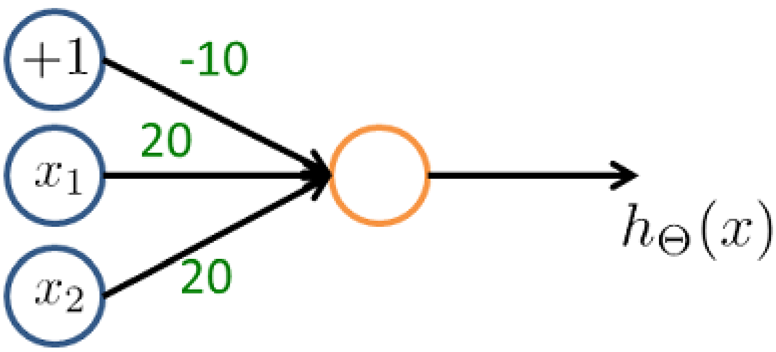
And if we associate the variables with the weights $+10$ and -20, then our hypothesis is computing: $h_\Theta(x) = g(10 - 20x_1)$. So when $x_1 = 0$, our hypothesis would be computing $g(10 - 20(0))$ that is just 10. And so that's approximately 1, and when $x_1 = 1$, this will be $g(-10)$ which is approximately equal to 0
.

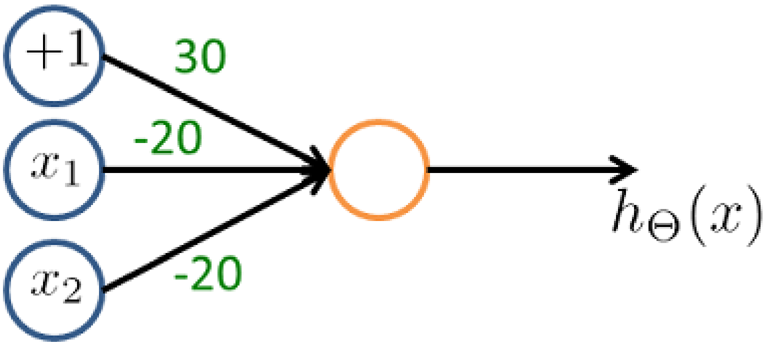


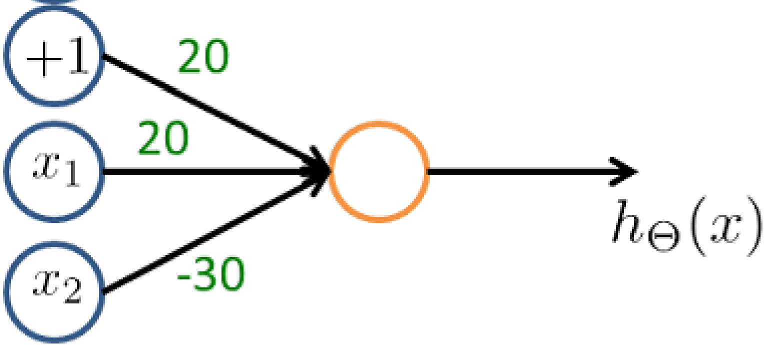
The general idea is to put that large negative weight in front of the variable we want to negate.

Video Question: Suppose that x_1 and x_2 are binary valued (0 or 1). Which of the following networks (approximately) computes the boolean function (NOT x_1) AND (NOT x_2)?



- 

- 

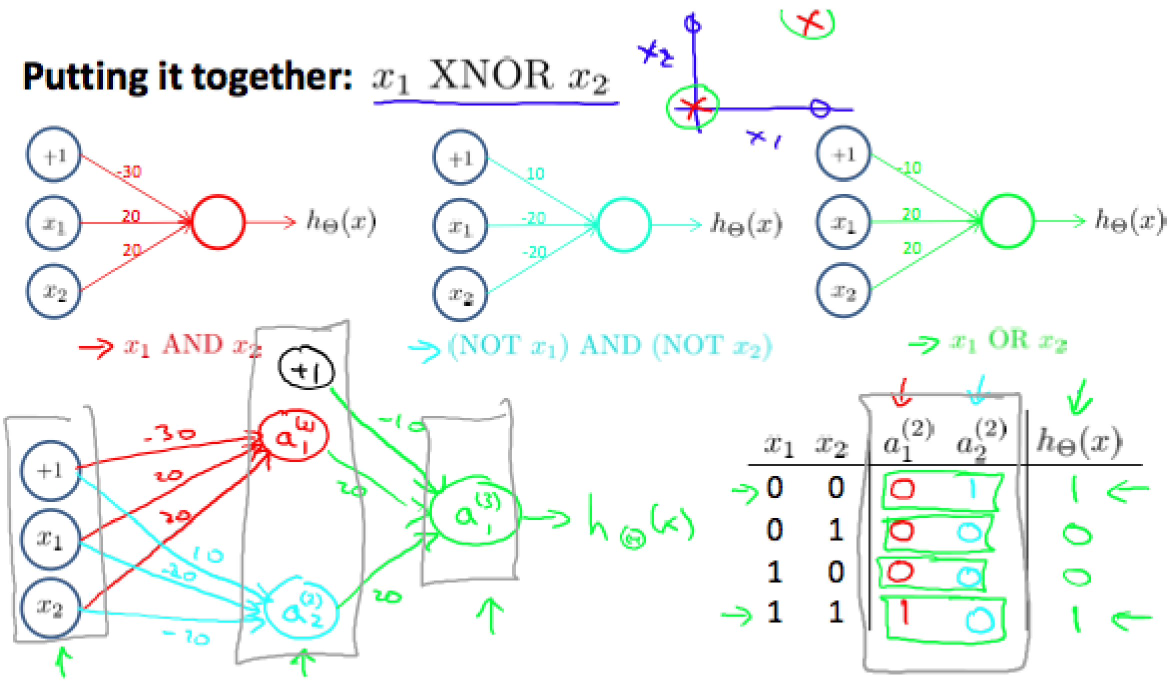
- 

Putting it all together: x_1 XNOR x_2

So how do we make the XNOR function work?

- XNOR is short for NOT XOR
- i.e. NOT an exclusive or, so either go big (1,1) or go home (0,0)

So we want to structure this so the input which produce a positive output are: AND (i.e. both true) **OR** Neither (which we can shortcut by saying not only one being true). So we combine these into a neural network as shown below;



And thus will this neural network, which has a input layer, one hidden layer, and one output layer, we end up with a nonlinear decision boundary that computes the XOR function.

Summary

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

$$AND:$$
$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \end{bmatrix}$$
$$NOR:$$
$$\Theta^{(1)} = \begin{bmatrix} 10 & -20 & -20 \end{bmatrix}$$
$$OR:$$
$$\Theta^{(1)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix}$$

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).

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

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

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

For the 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:

$$a^{(2)} = g(\Theta^{(1)} \cdot x)$$
$$a^{(3)} = g(\Theta^{(2)} \cdot a^{(2)})$$
$$h_\Theta(x) = a^{(3)}$$

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