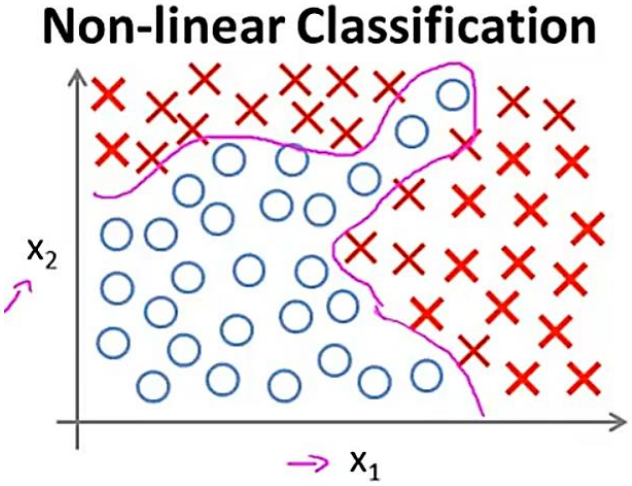
8. Neural Network Representation:

Neural networks == algorithm

WHAT IS THE PROBLEM WITH POLYNOMIAL HYPOTHESIS:



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

If there are more features:

$x_1 = \text{size}$
$x_2 = \#\text{ bedrooms}$
$x_3 = \#\text{ floors}$
$x_4 = \text{age}$
$\dots$
x_{100}

The no of quadratic features grows as **$O(n^2)$:**

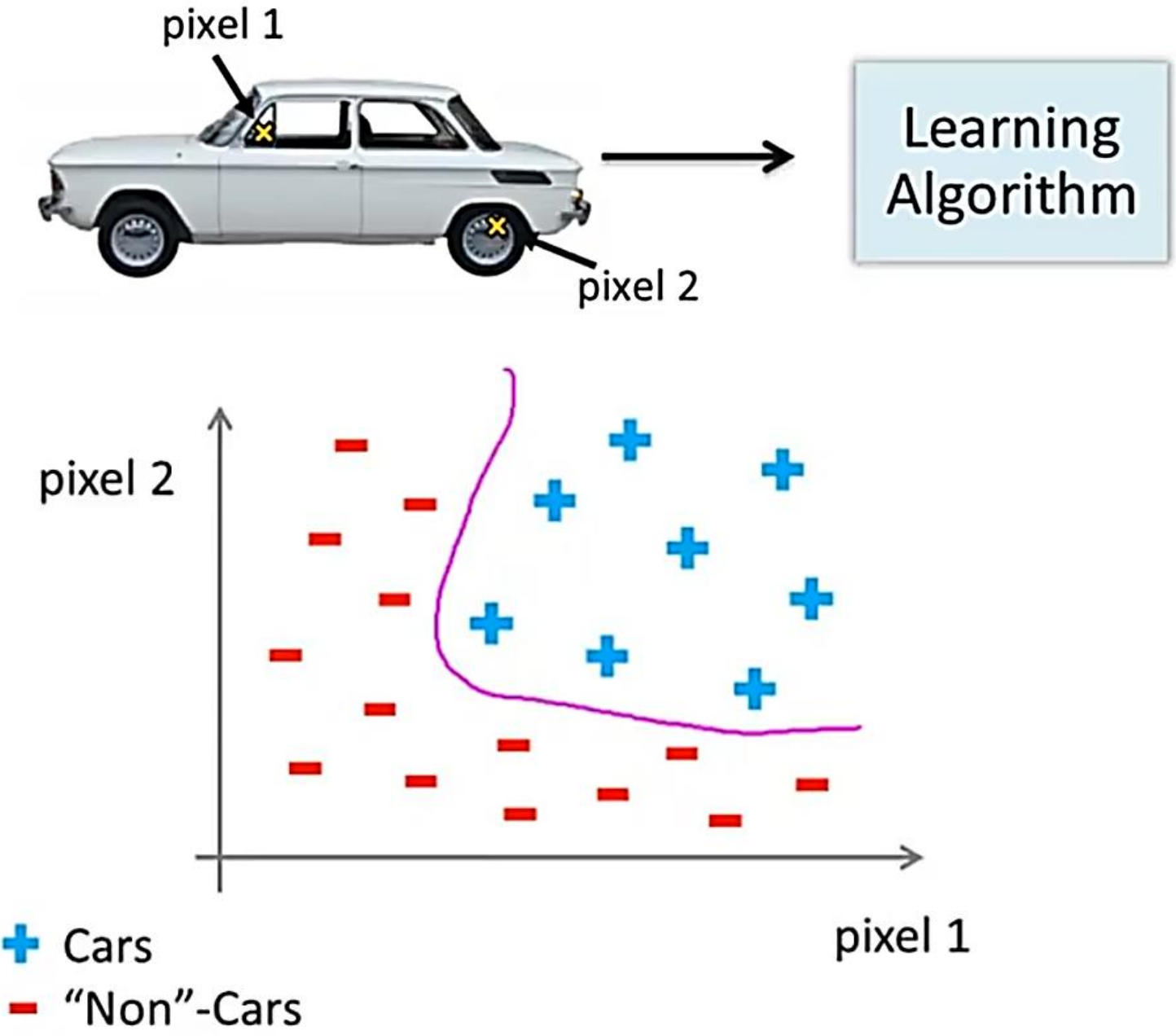
$x_1^2, x_1x_2, x_1x_3, x_1x_4 \dots x_1x_{100}$
$x_2^2, x_2x_3 \dots$
≈ 5000 feature $O(n^2)$

The no of cubic features grow: **O(n^3):**

$x_1 x_2 x_3$, $x_1^2 x_2$, $x_{10} x_{11} x_{17}$, ...

As the features grow: the problem of overfitting may also occur.

Example:



With linear or non-linear regression, the values required to predict if it's a car will be very high, it will require intensity information of all individual pixels of the image.

This will be a very tedious to check with hypothesis fxn.. (all pixel values being a feature).

50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

This is just for individual features.. quadratic features and cubic and higher order features may also be there.

Quadratic features ($x_i \times x_j$): ≈3 million features

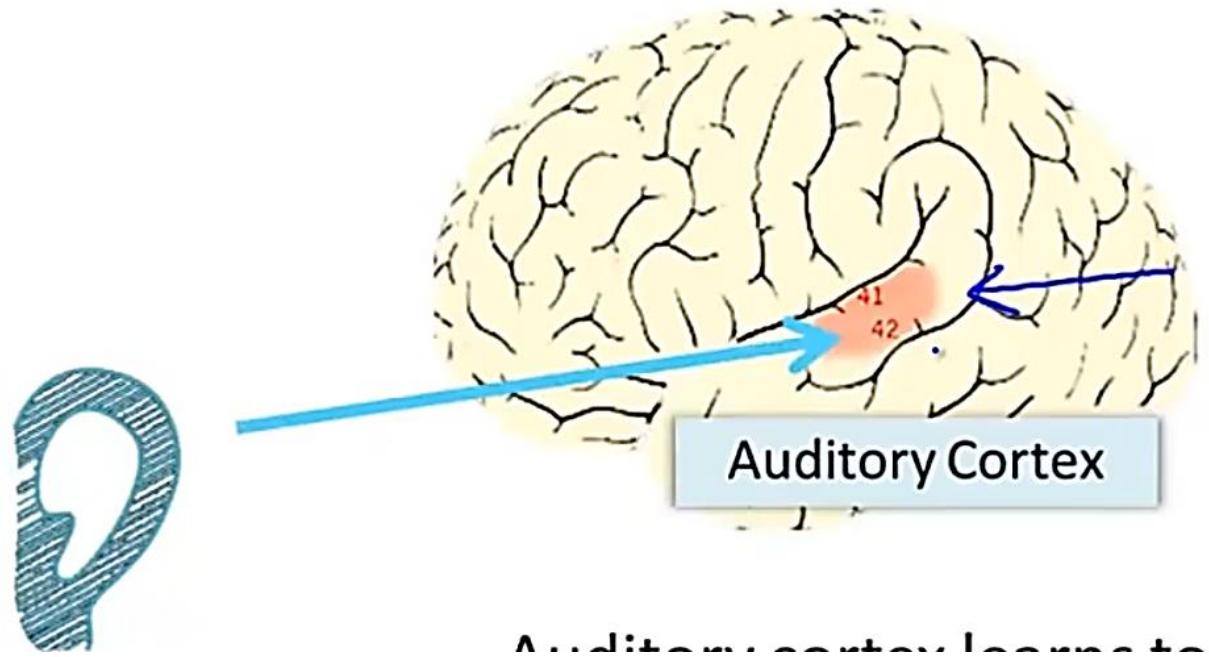
These algorithms take too many features and for simple tasks like computer vision. So, they are inefficient.

Therefore, we use neural networks algorithm

⇨ **Neurons and brain:**

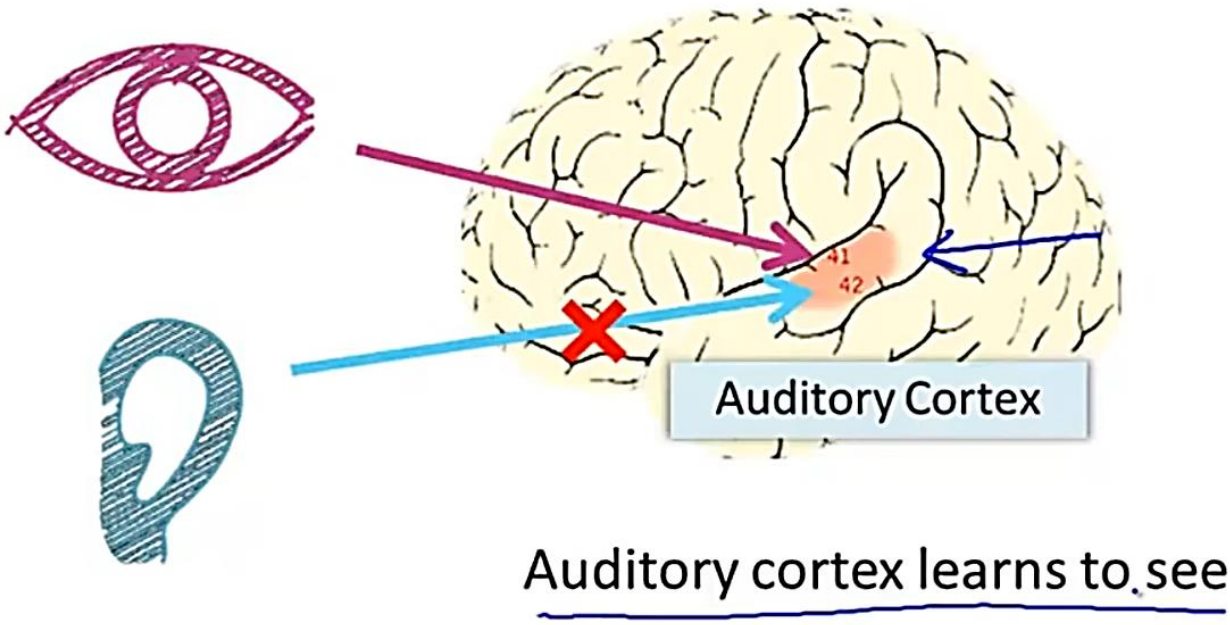
Algorithms that try to mimic the brain.

The "one learning algorithm" hypothesis

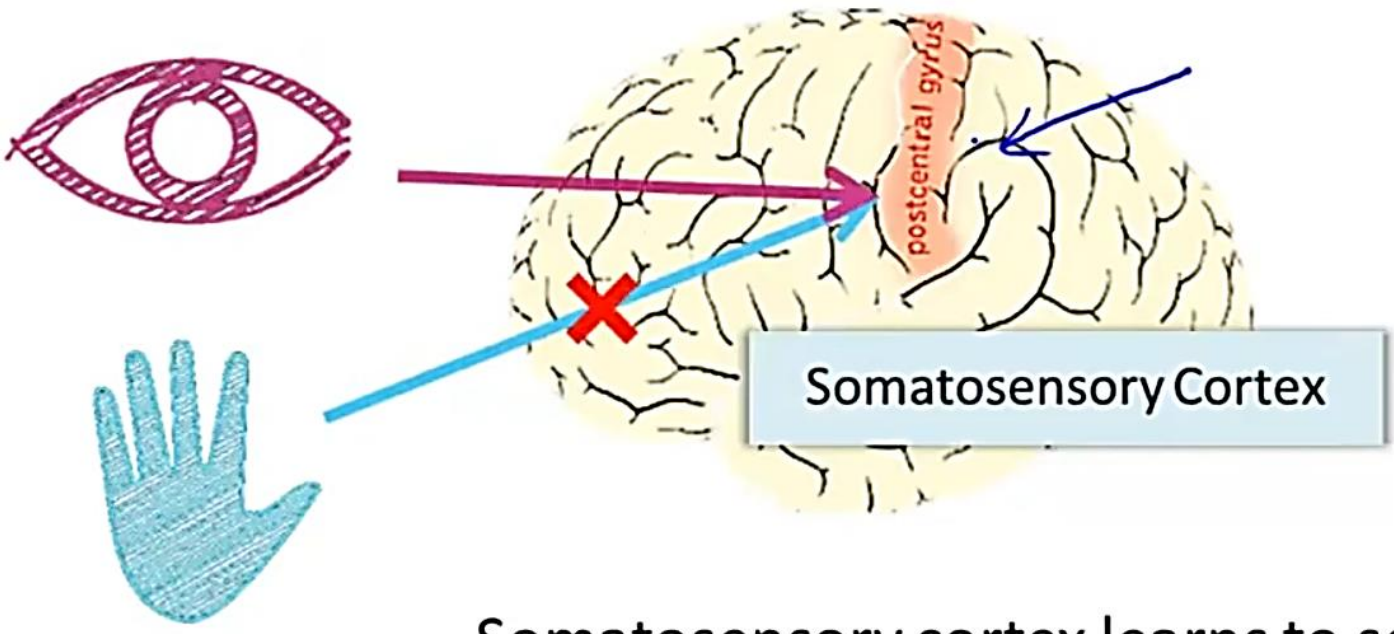


Auditory cortex of the brain is the part which is responsible for interpretation of what we hear.

If we cut the link bw ear and the auditory cortex.. and rewire it to the eye: It will learn to see.



Similarly: somatosensory cortex is responsible for the sense of touch.. but if we rewire it to the eye.. it will learn to see.



Somatosensory cortex learns to see

This shows that same type of brain tissues can learn all types of things.

SENSOR REPRESENTATION IN BRAIN:

Applications:

Seeing with your tongue

Here, the camera sees a low resolution b/w image of the object and this camera is wired to a device mounted on tongue ..

Now, for each pixel on the image. there is a small dot on this device on tongue. which applies a high voltage on pixels with high brightness and low voltage on pixels with low brightness.

The tongue senses this voltage and with time the brain may learn to see with tongue

Human echolocation (sonar)

Human brain can learn to sense distance from the echo of sound they produce

Haptic belt: Direction sense

In this belt, the speaker on the north direction always beeps

Implanting a 3rd eye

If we implant a 3rd eye on a frog it will learn to use it.

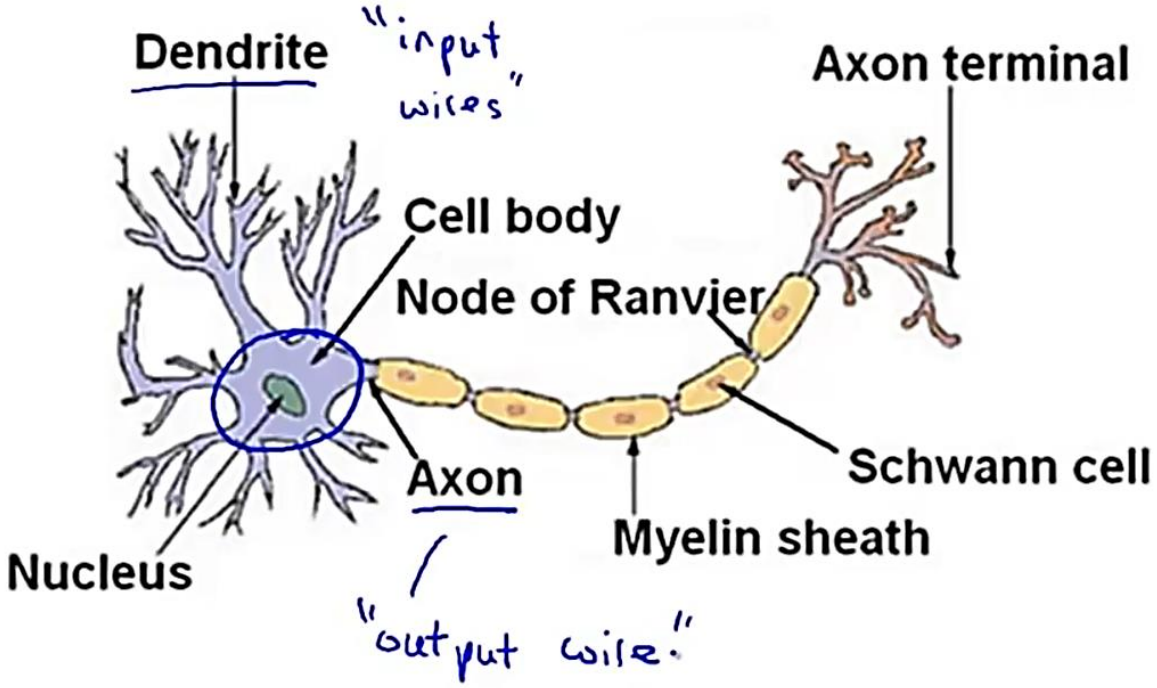
MODEL REPRESENTATION:

Dendrites = input

Cell body = computational unit

Axon = output carrier from one neuron to another

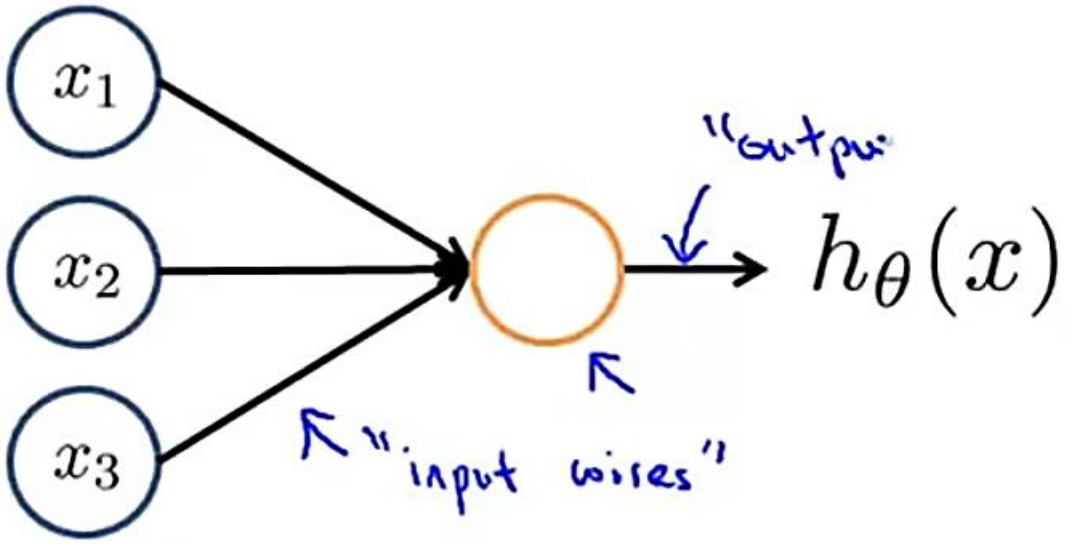
Neuron in the brain



Dendrites are like the input features

NEURON model: logistic unit:

Neural networks simulate neurons



Here h(x) is the output

We can also include an x_0 == bias unit which is always ==1

For logistic unit:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

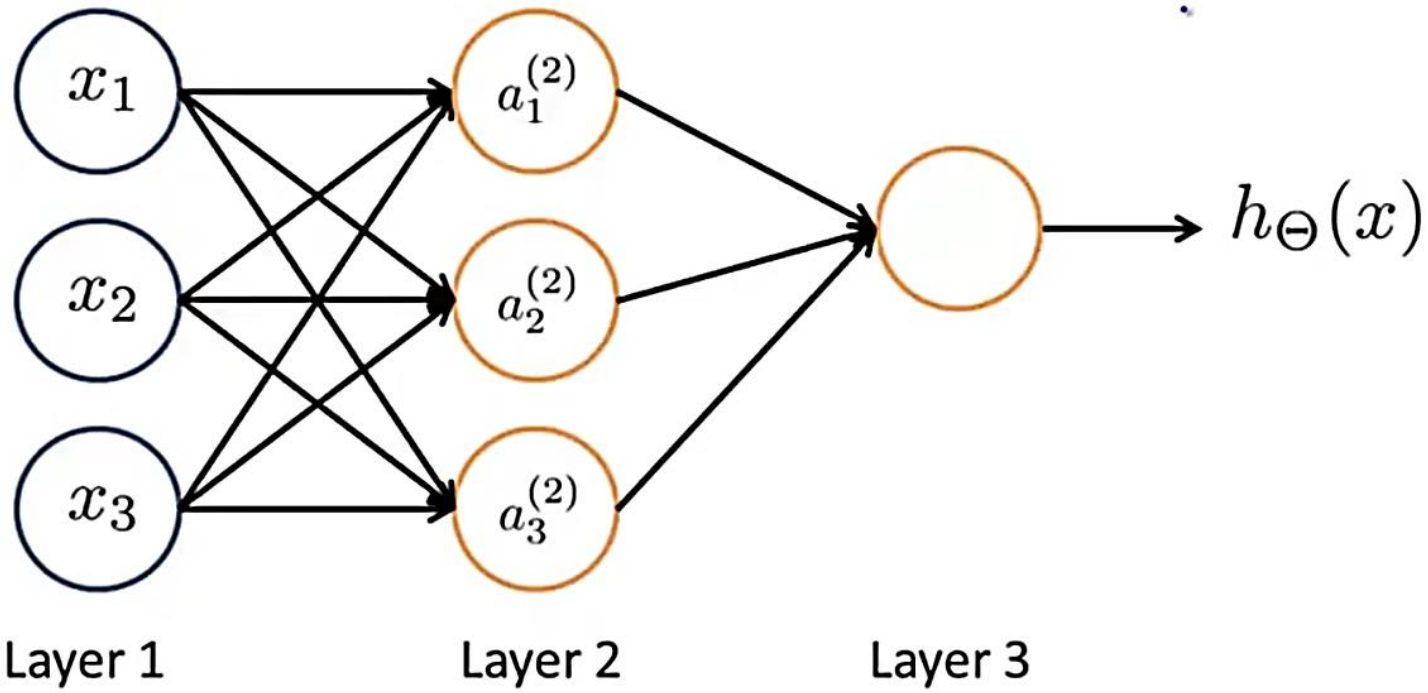
Sigmoid (logistic) activation function.

$$g(z) = \frac{1}{1+e^{-z}}$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Here **Θ** is called == **weights/parameters**

NEURAL NETWORK:



Layer 1 = input layer

Layer 2 == hidden layer—contains activation units

Layer 3 = output layer

- We can also have x_0 and a_0 – **bias unit**

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

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

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

- Here, all the **superscripts** denote the **no of layer**
- **Subscripts** of a denote different types of algos used in them during computation
- **Subscripts of x** denote – different **input features**
- **Subscripts of Θ** are combination of subscripts of a and x on which they are applied

⇨In a_1, a2, a3 – all Θ have level=1 -- it controls mapping from layer 1 to layer 2

All a have level =2

⇨In h(x) – all Θ have level=2 -- it controls mapping from layer 2 to layer 3

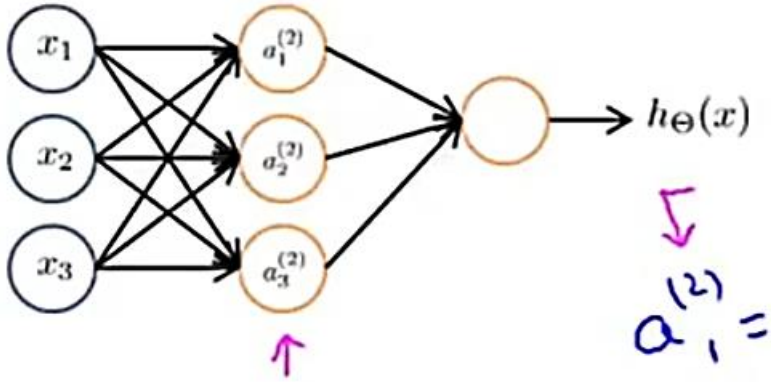
All a have level=2

If network has s_j units in layer j, 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.

Vectorized implementation:

Each a can be written as g(z)



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

The insides of brackets are denoted by z:

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

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

Now a vector of all 3 a's can be written as Θ.X:

Θ = vector of all values of Θ

X = vector of values of all features

Therefore,:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

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

We can also call all the features in the input layer as activations of level 1

$a^{(1)} = x$

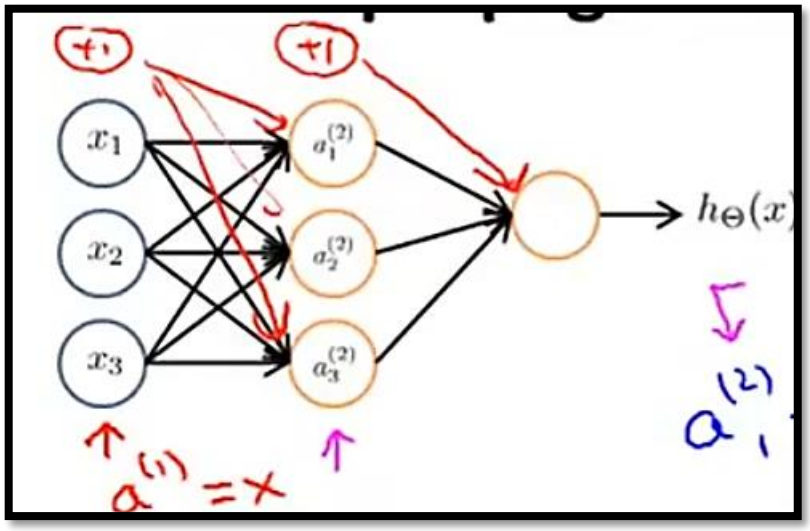
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

In general:

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

Also, we need an extra bias activation: always =1

We can then add a bias unit (equal to 1) to layer j after we have computed all $a^{(j)}$. This will be element and will be equal to 1.



Add $a_0^{(2)} = 1.$ $\longrightarrow$ $a^{(2)} \in \mathbb{R}^4$

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

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

Or in general:

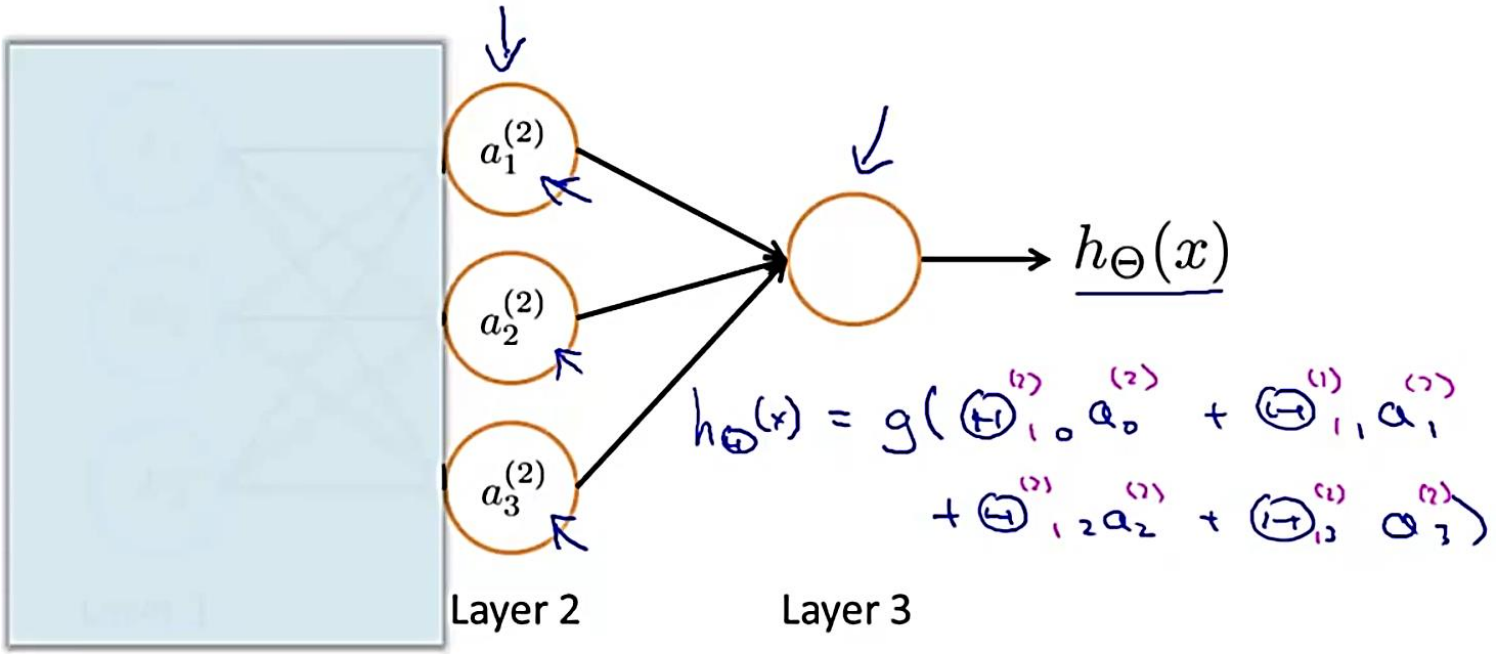
$$z^{(j+1)} = \Theta^{(j)} a^{(j)}$$

This last theta matrix will have only one row $\Theta^{(j)}$ which is multiplied by one column $a^{(j)}$ so that our result is a single number.

We then get our final result with:

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

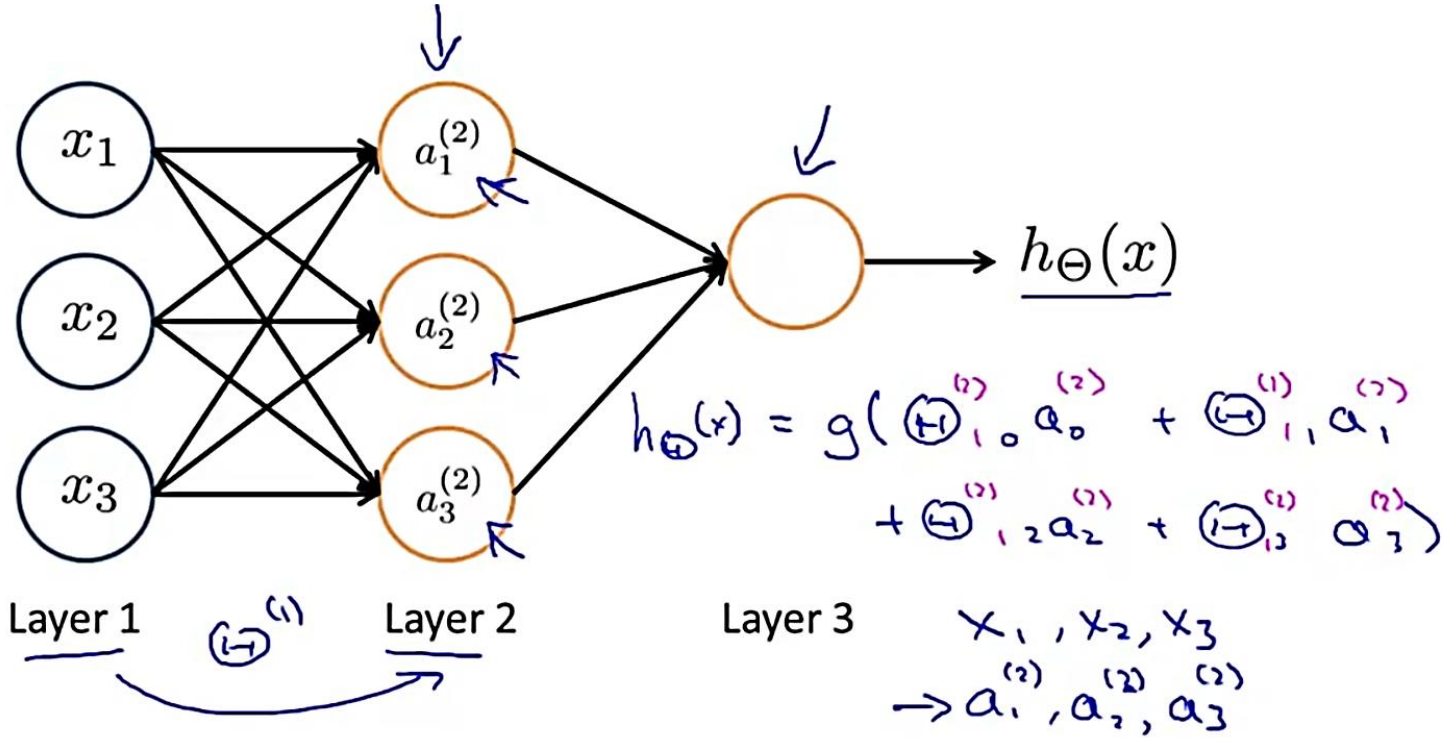
Neural Network learning its own features



This looks like logistic regression in which input features are "a's"

But a's are themselves derived from fxns of x.

The neural networks learnt its own new features from the old features and used them into a new logistic regression while going from level 2 to level 3.

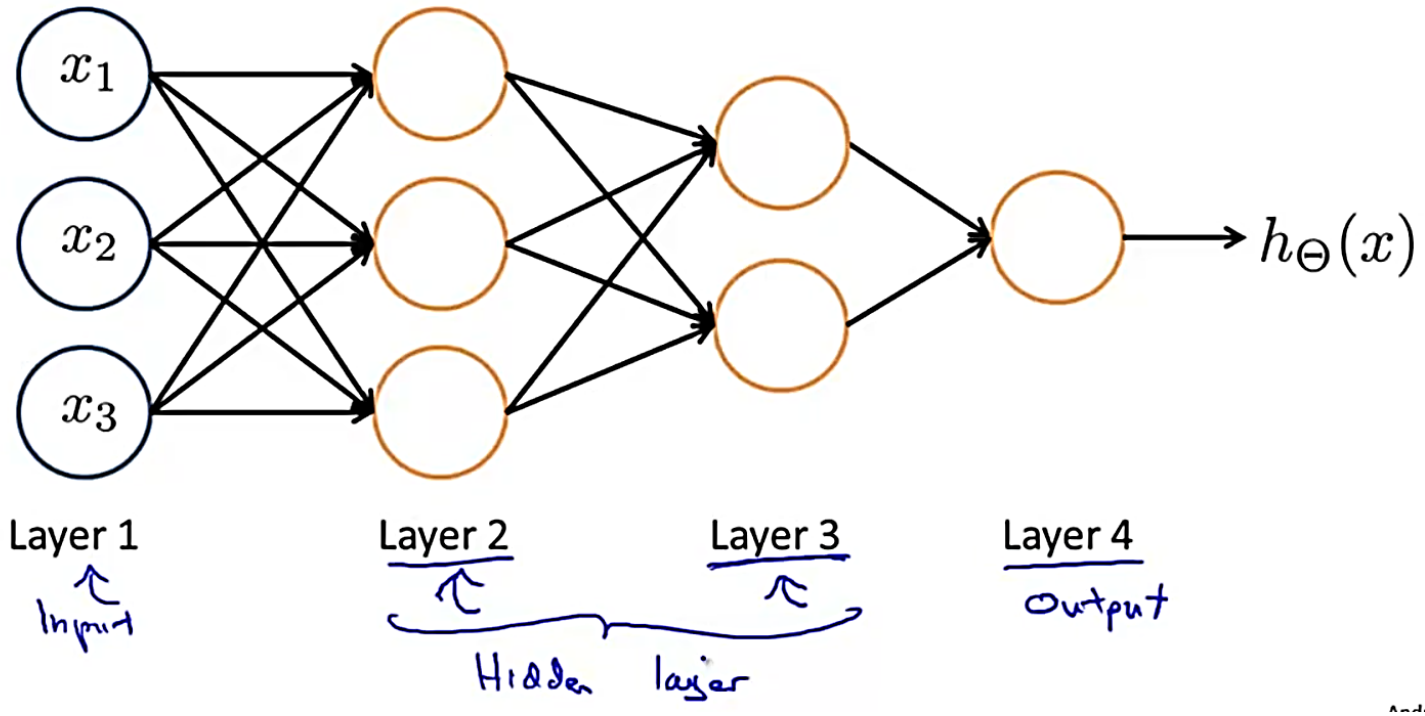


So, here we transformed our original input features(x's) into new features(a's), and they are used in logistic regression.

So, we are saved from using high order features like quadratics (x^2 or x_1x_2) or cubics (x^3 or $x_1x_2x_3$)

Network architectures may have more layers, o/p of each layer is input to next.

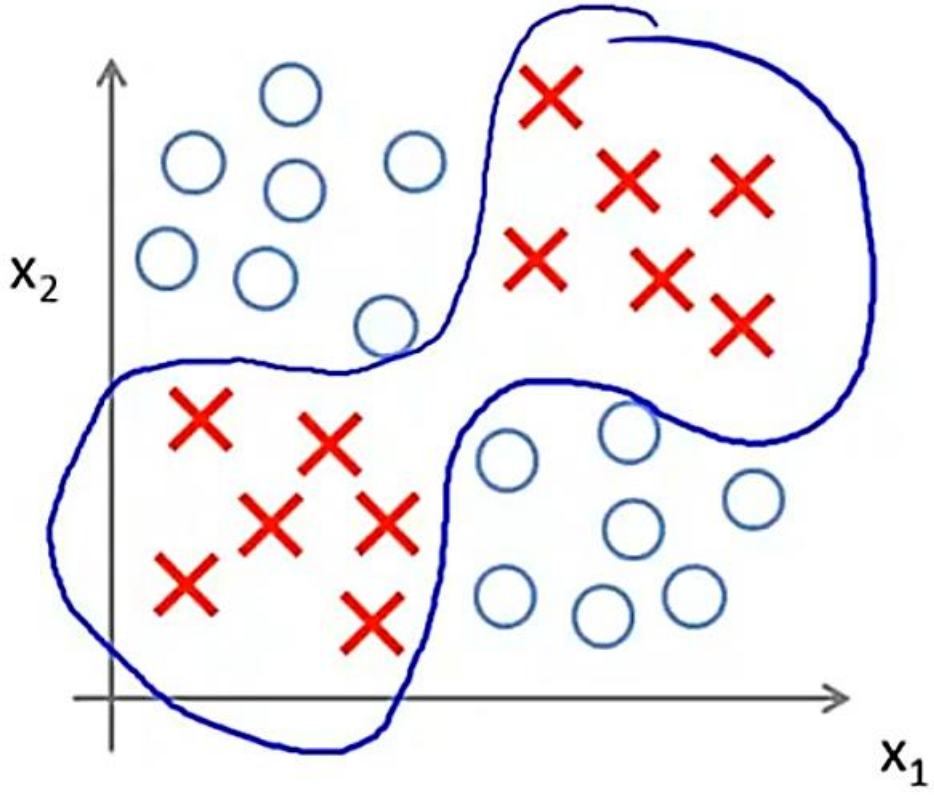
Other network architectures



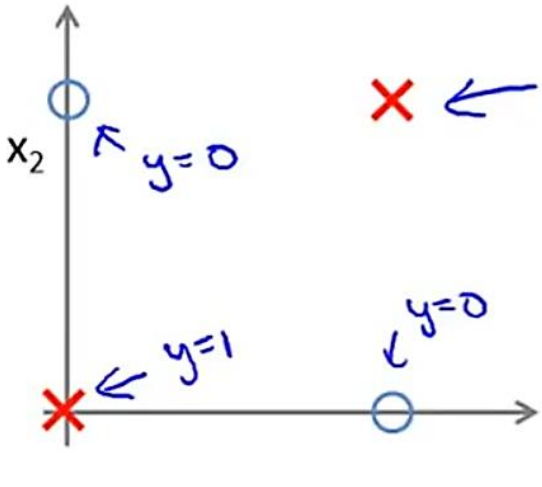
INTUITIONS OF NEURAL NETWORKS:

Non-linear classification example: XOR/XNOR

- x_1, x_2 are binary (0 or 1).



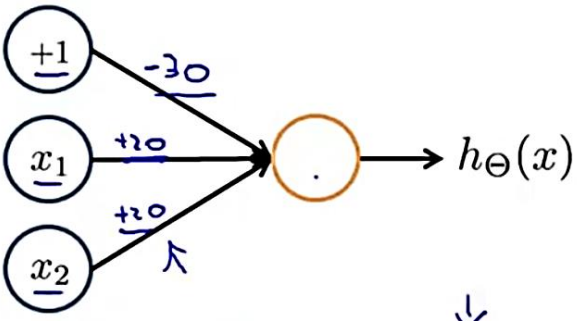
Simplified version:



$$y = x_1 \text{ XOR } x_2$$
$$x_1 \text{ XNOR } x_2$$
$$\text{NOT } (x_1 \text{ XOR } x_2)$$

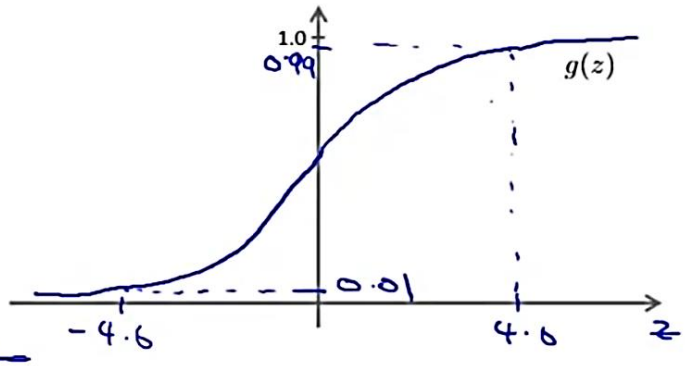
Simple example: AND

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$



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

$\Theta^{(1)}_{10}$ $\Theta^{(1)}_{11}$ $\Theta^{(1)}_{12}$



x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

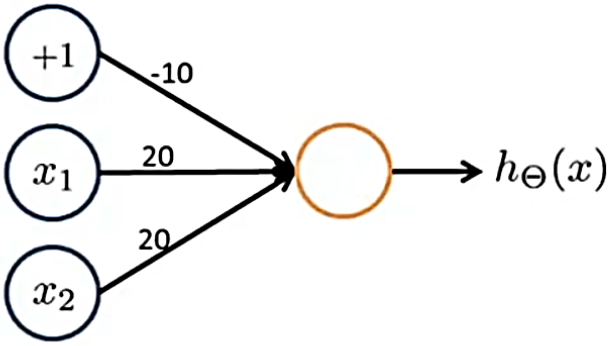
$h_\Theta(x) \approx x_1 \text{ AND } x_2$

The graph of our functions will look like:

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

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

Example: OR function

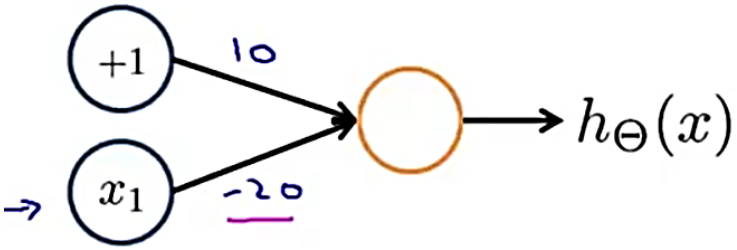


$g(-10 + 20x_1 + 20x_2)$

x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	≈ 1
1	1	≈ 1

Negation:

NOT x_1

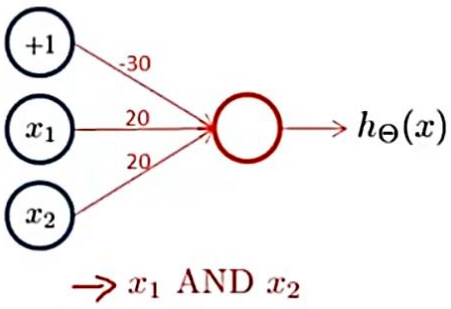


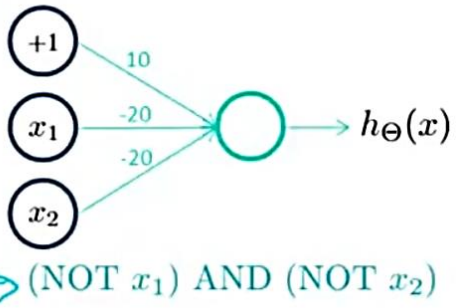
x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

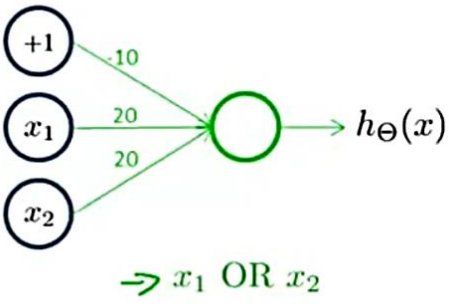
$$h_\Theta(x) = g(10 - 20x_1)$$

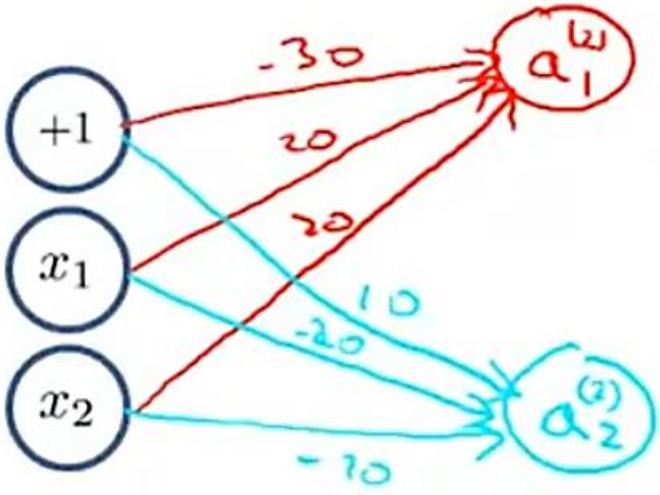
XNOR: (a AND b) OR (a' AND b')

Putting it together: x_1 XNOR x_2



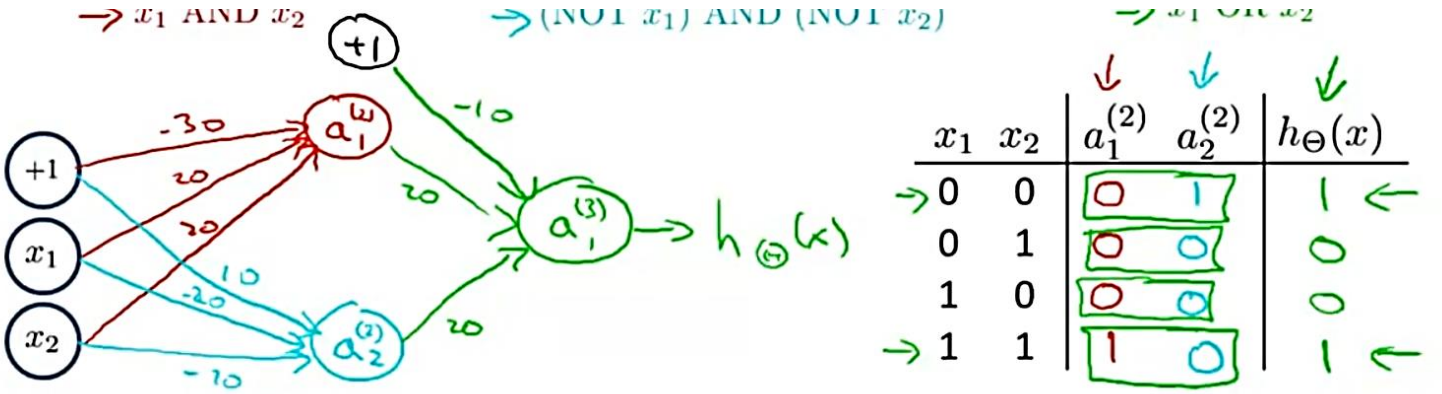






x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	0

Adding the next layer:



MULTICLASS CLASSIFICATION:

Its just an extension of **One vs all** technique:

Suppose we want to classify amongn 4 objets:

Multiple output units: One-vs-all.


Pedestrian


Car


Motorcycle


Truck

To classify data into multiple classes, we let our hypothesis function return a vector of values.

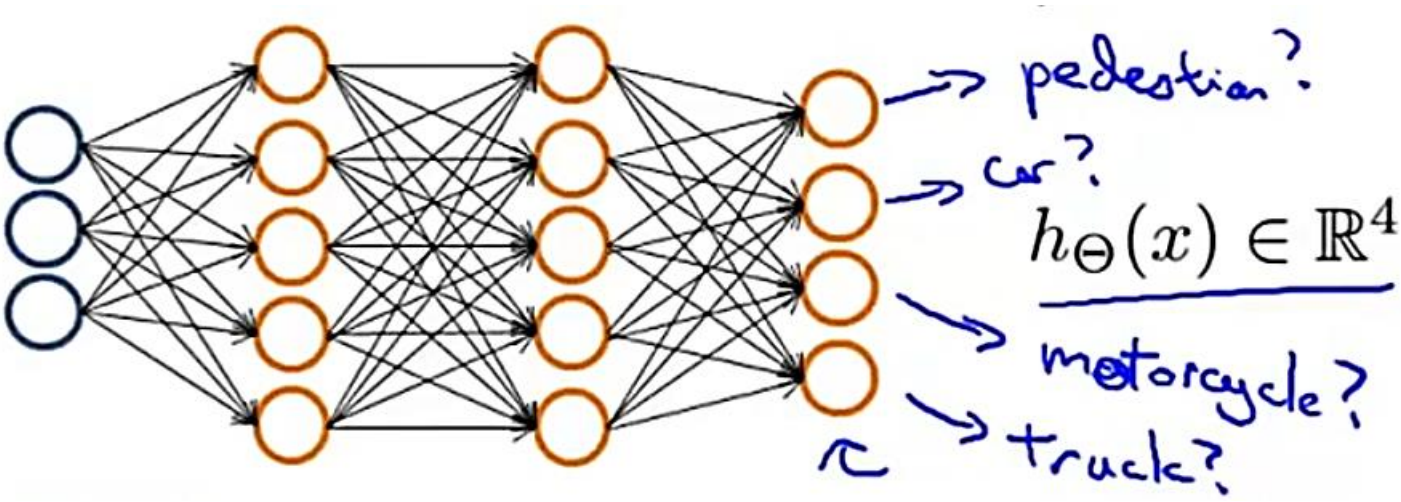
Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

This means we want 4 o/p all at one.. each o/p corresponds to one of our objects

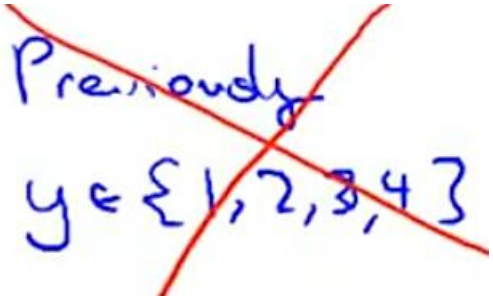
Therefore, our o/p is a 4 dimentional vector:

Each dimention can be 1 or 0 depending on which object is identified



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:

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



we used to represent y in this way, but now:

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

pedestrian car motorcycle truck