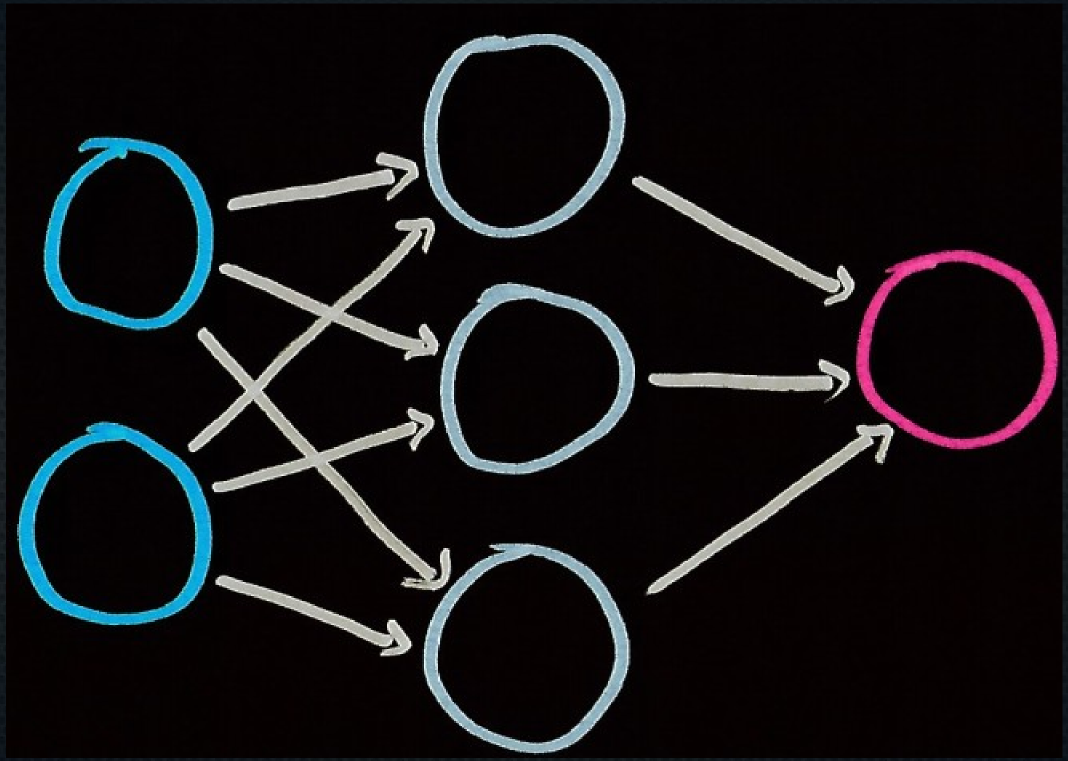
Switching gears: neural networks

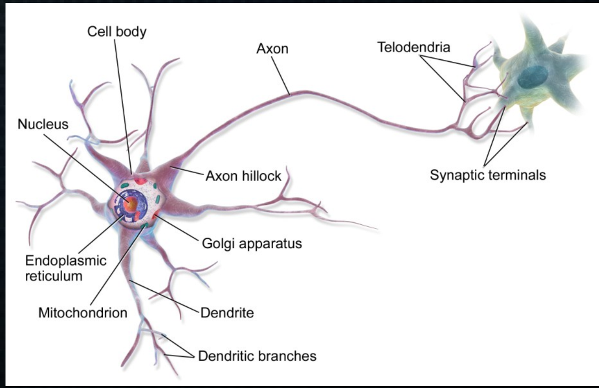


Source: https://thesharperdev.com/build-your-first-neural-network-part-2/

Why neural networks?

Why neural networks?

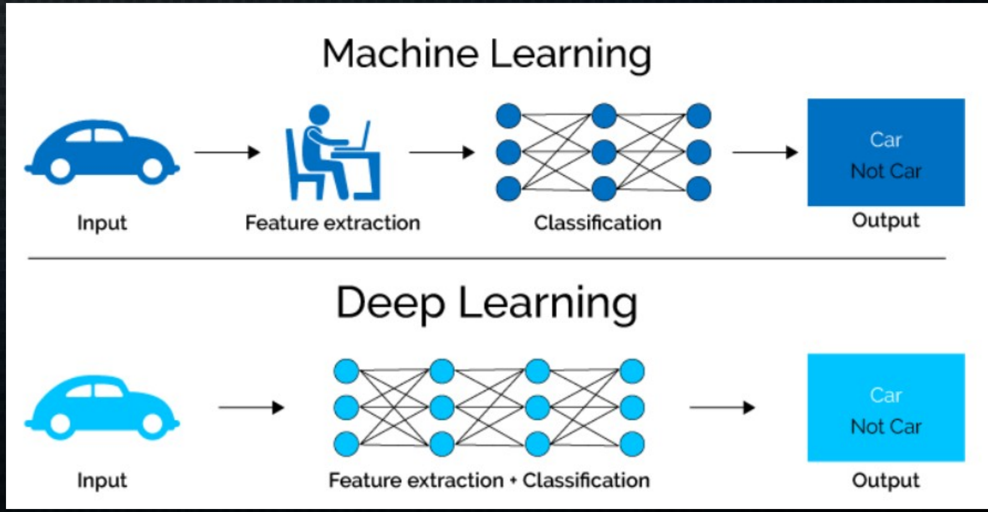
- Best performing algorithms for complex tasks, bar none.
- Known potential of hierarchical organisation of simple units because of biological examples (though neural networks are not good models of actual neurons)
- Observation in frogs and cat visual cortex: there are specific layers of neurons, where earlier layers detect basic shapes (lines, edges) with later layers incorporating this information into more complex features about what is seen.



Source: https://en.wikipedia.org/wiki/Axon#/media/File:Blausen_0657_MultipolarNeuron.png

Why neural networks?

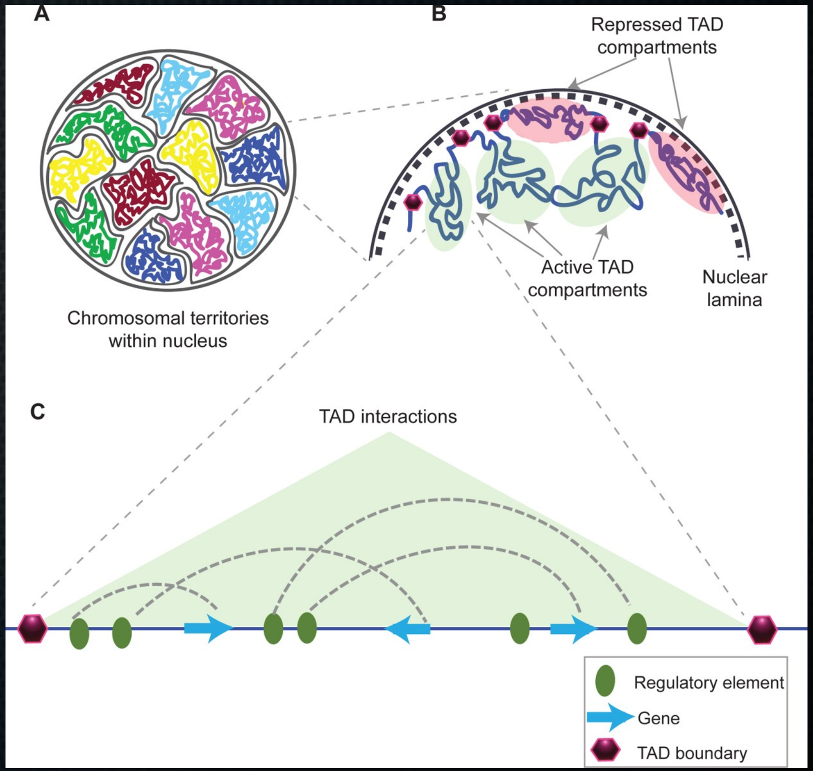
- Until now, we decided on the features to give to our algorithms: think tumour size, biopsy test scores, etc.
- With neural networks and images, the situation changes: we don't arduously describe what is in each image, but rather let the network learn to extract and combine features *so that* it can classify training examples correctly.



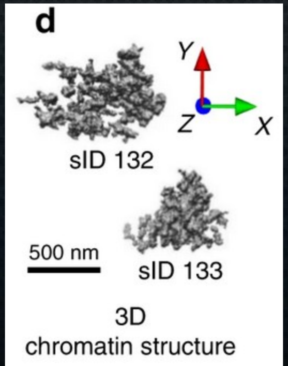
Source: https://towardsdatascience.com/cnn-application-on-structured-data-automated-feature-extraction-8f2cd28d9a7e

Why neural networks?

- Caveat for biology: it can be quite difficult to translate biological problems into a framework fit for deep learning, for reasons that will become clear later.
- Example: in images, nearby pixels probably hold similar information, i.e. are involved in the same thing. Due to (long-range) 3D-folding of DNA, linearly far DNA can be close together functionally. You need to encode network or input to accomodate this!



Source: https://en.wikipedia.org/wiki/Topologically_associating_domain#/media/File:Structural_organization_of_chromatin.png



Source: Trzaskoma, P., Ruszczycki, B., Lee, B., Pels, K. K., Krawczyk, K., Bokota, G., ... & Wilczyński, G. M. (2020). Ultrastructural visualization of 3D chromatin folding using volume electron microscopy and DNA in situ hybridization. Nature communications, 11(1), 1-9.

Why neural networks?

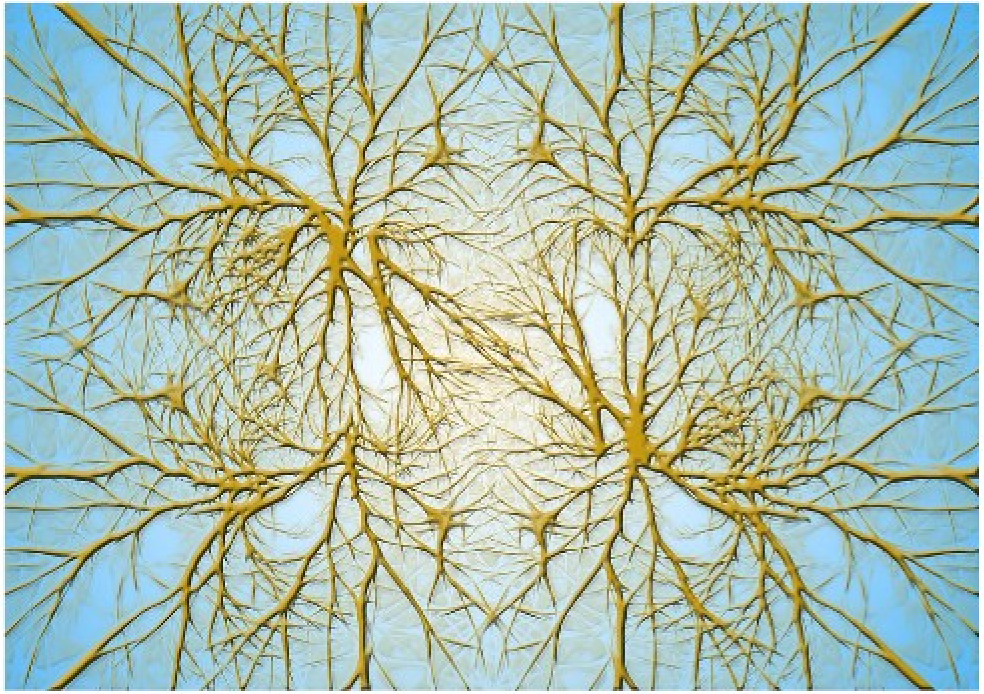
- The mythical property of *universal approximation*. This says that neural networks can approximate any function with arbitrary accuracy, even with only **1** hidden layer (given enough neurons in it).

Why neural networks?

- The mythical property of *universal approximation*. This says that neural networks can approximate any function with arbitrary accuracy, even with only **1** hidden layer (given enough neurons in it).
- Of course, that doesn't necessarily mean we would have the data to *train* such a neural network efficiently. Just that it is provable that for *any continuous function* a neural network can exist that approximates it as well as you like.

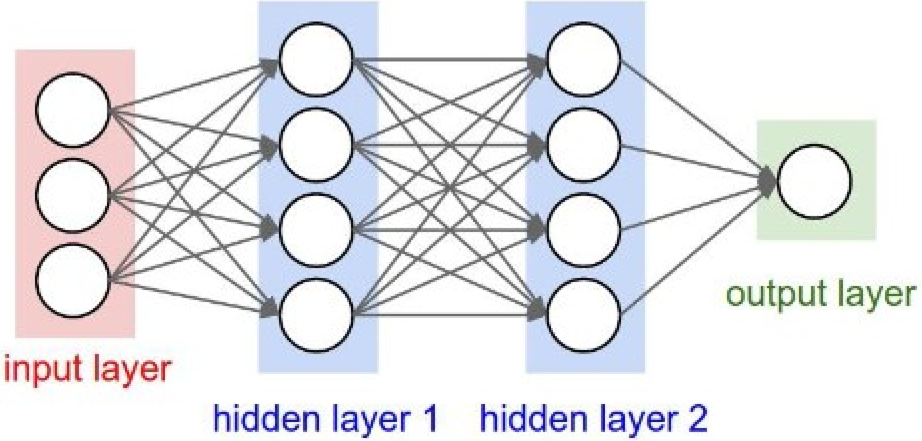
Like biology? **No**

Biological Neurons:
Complex connectivity patterns



This image is CC0 Public Domain

Neurons in a neural network:
Organized into regular layers for computational efficiency



Source : http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

Like biology? **No**

Biological Neurons:

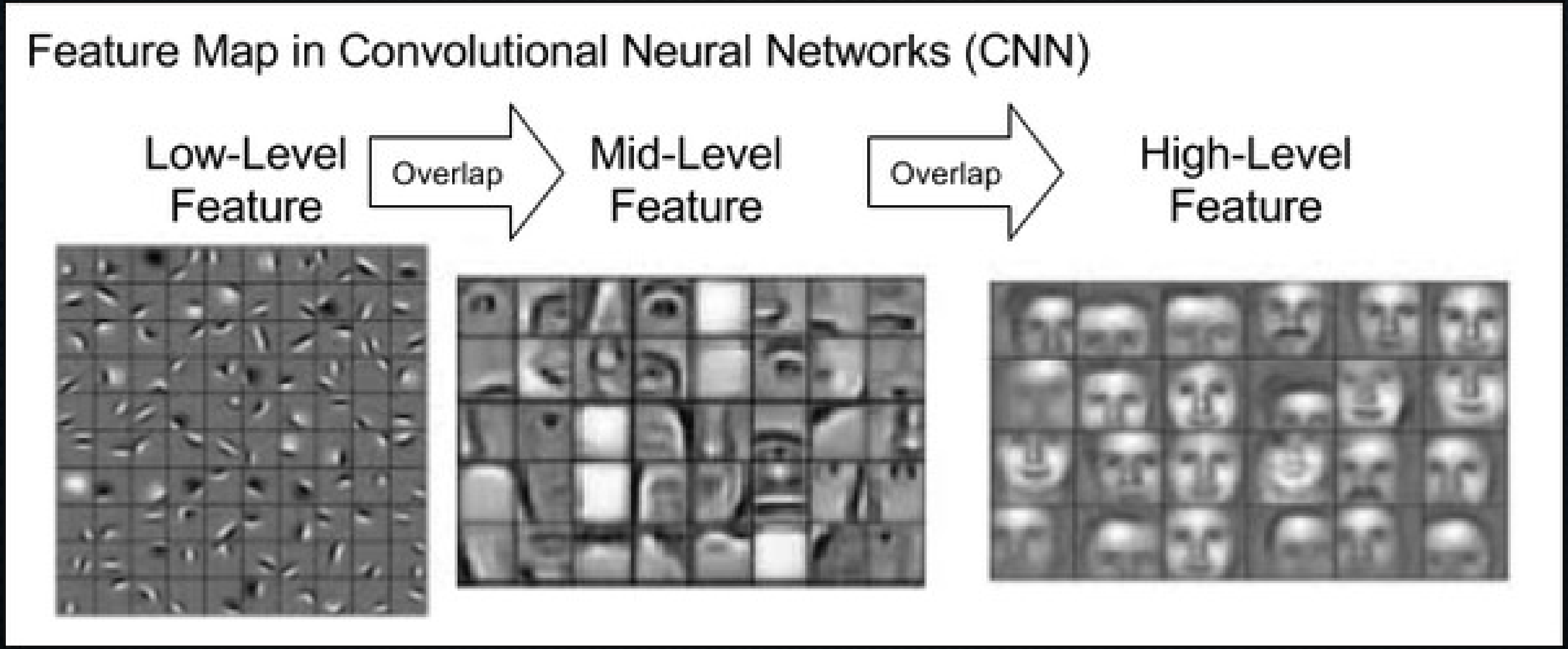
- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system

Source : http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

- Human brains ~a cool 86 billion neurons
- A neuron can have 400.000 dendrites
- Real brains *vastly outclass* their computational analogues

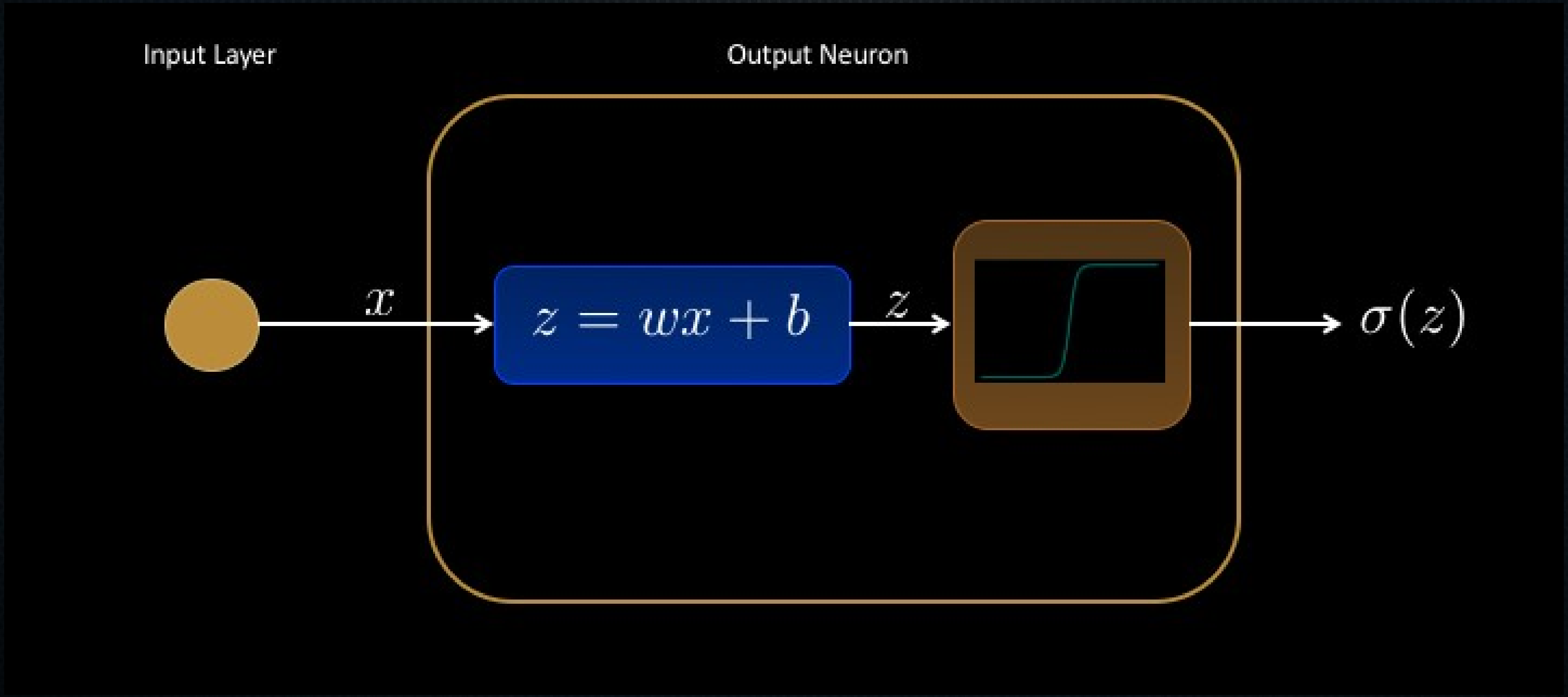
Like biology? **No**

- Still extremely useful
- Parts of how they learn *superficially resemble* how we learn



What do neural nets have to do with logistic regression?

- A single neuron is a logistic regressor!*

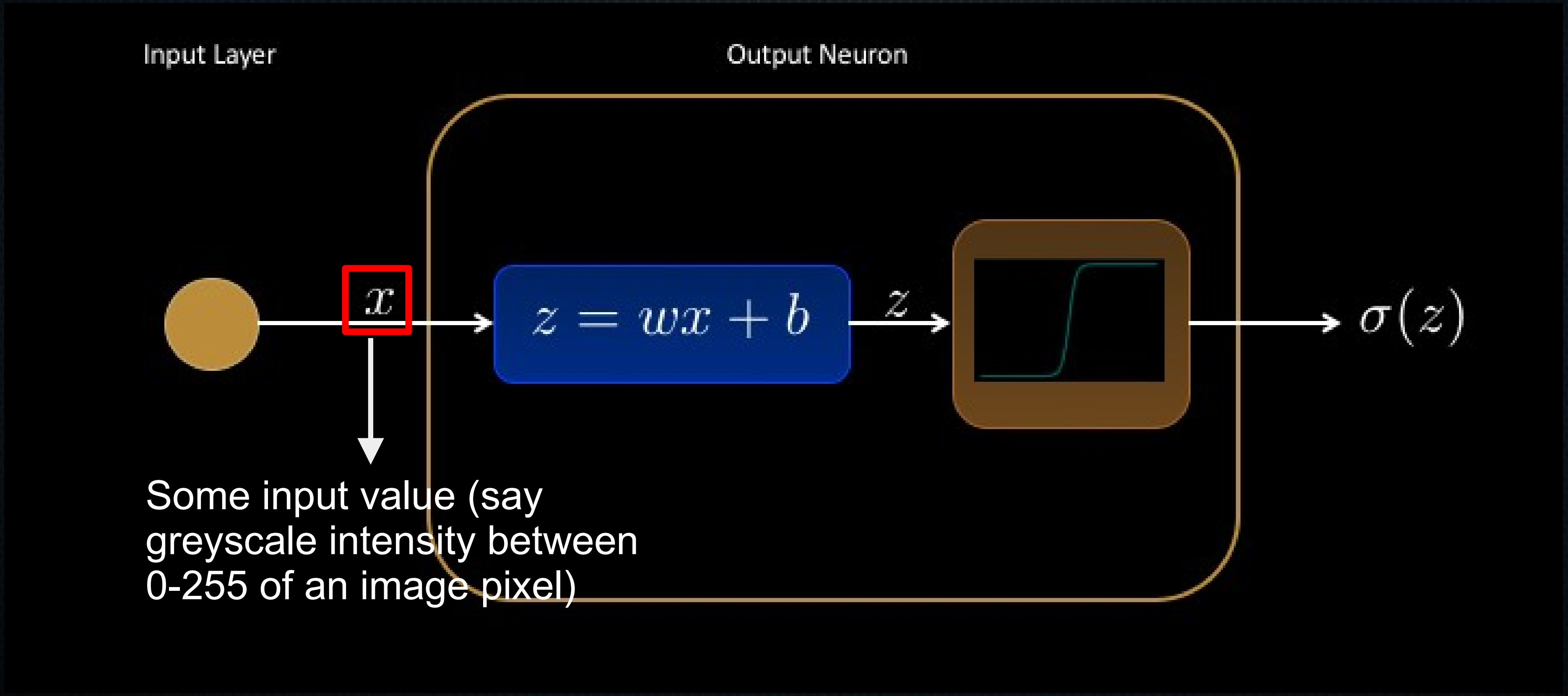


Source: https://thedatafrog.com/en/articles/logistic-regression/

*Given sigmoid activation function (which is not current standard practice anymore, but was at first)

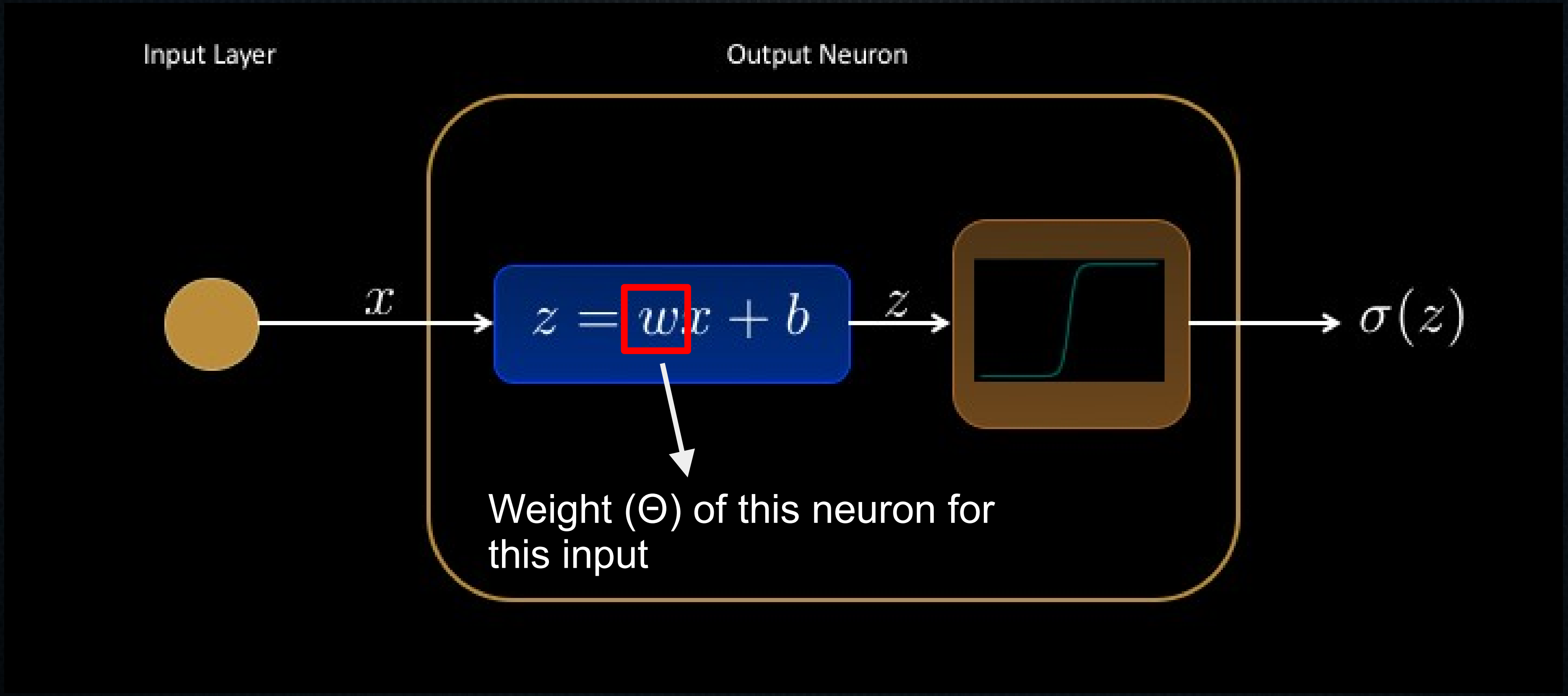
What do neural nets have to do with logistic regression?

- A single neuron is a logistic regressor!



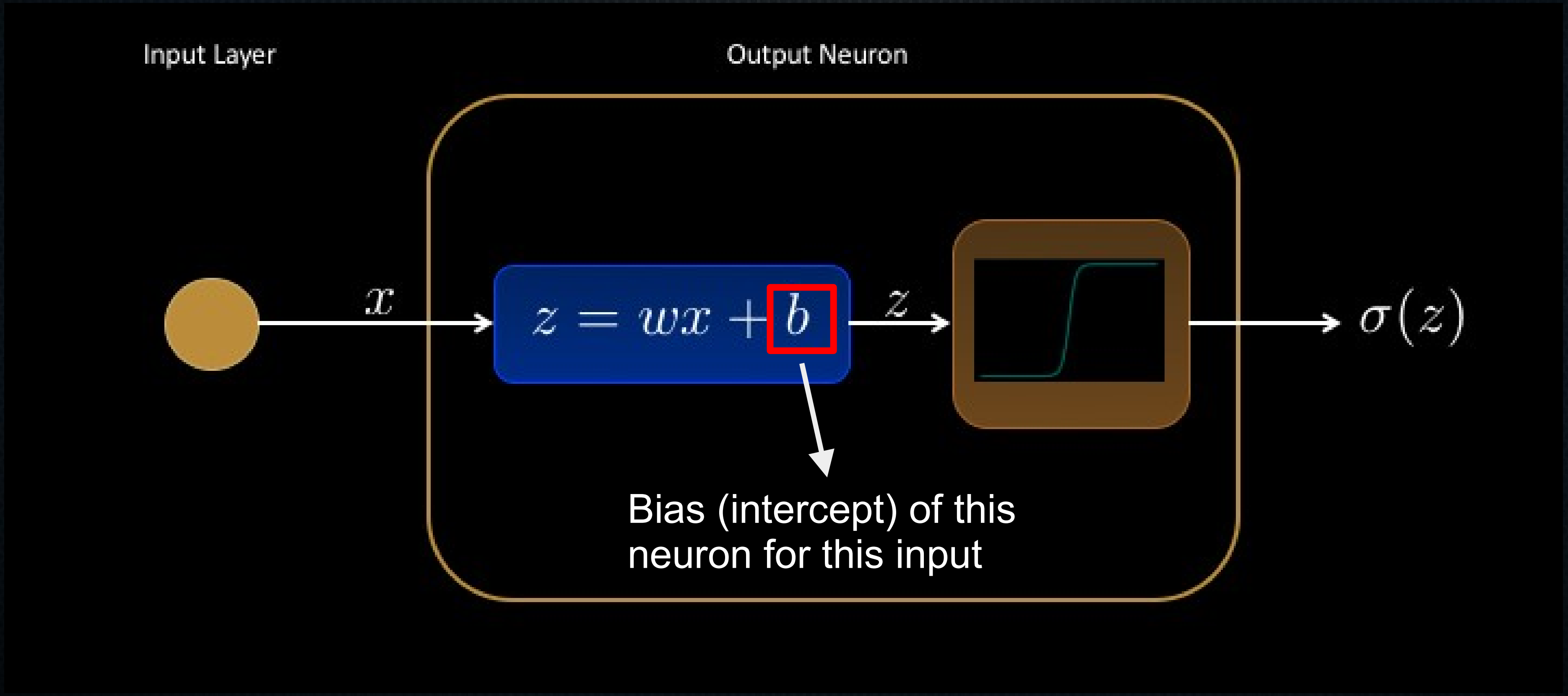
What do neural nets have to do with logistic regression?

- A single neuron is a logistic regressor!



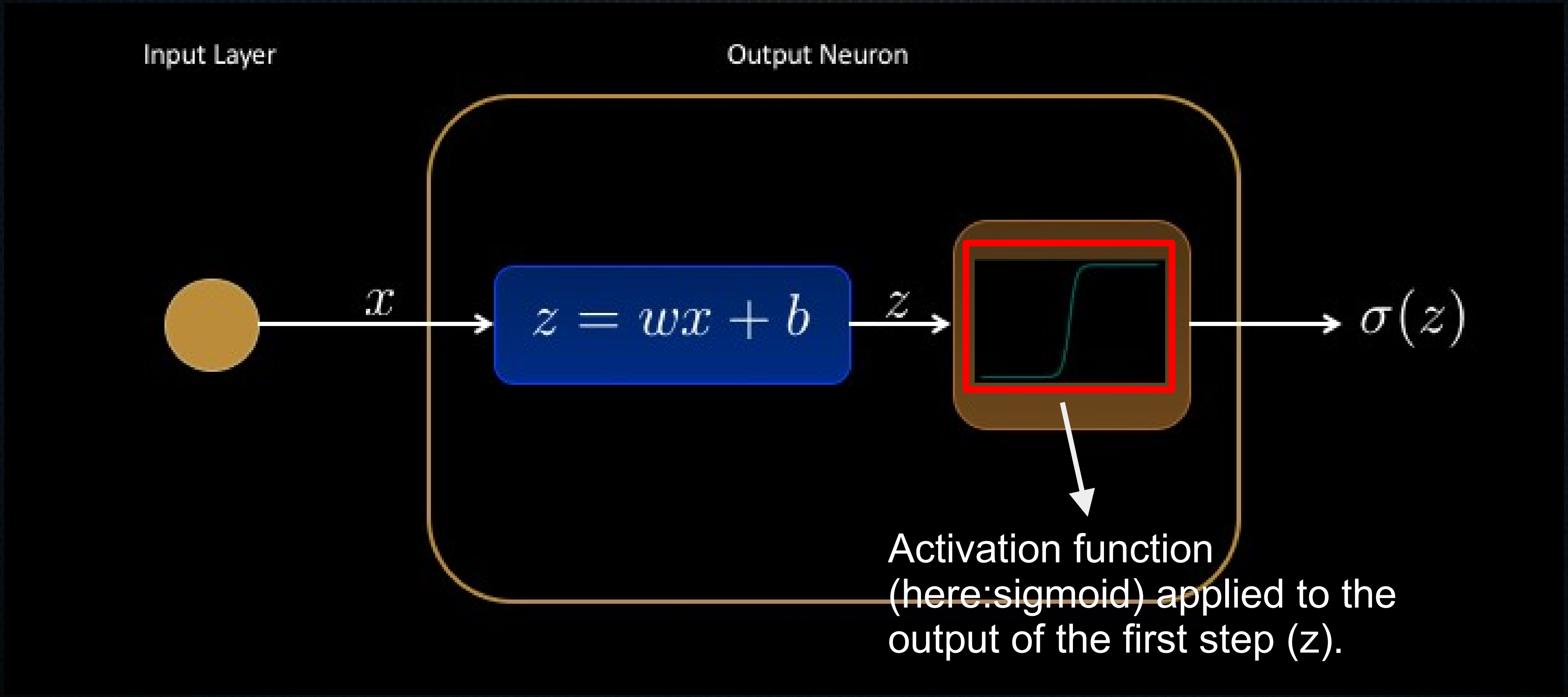
What do neural nets have to do with logistic regression?

- A single neuron is a logistic regressor!



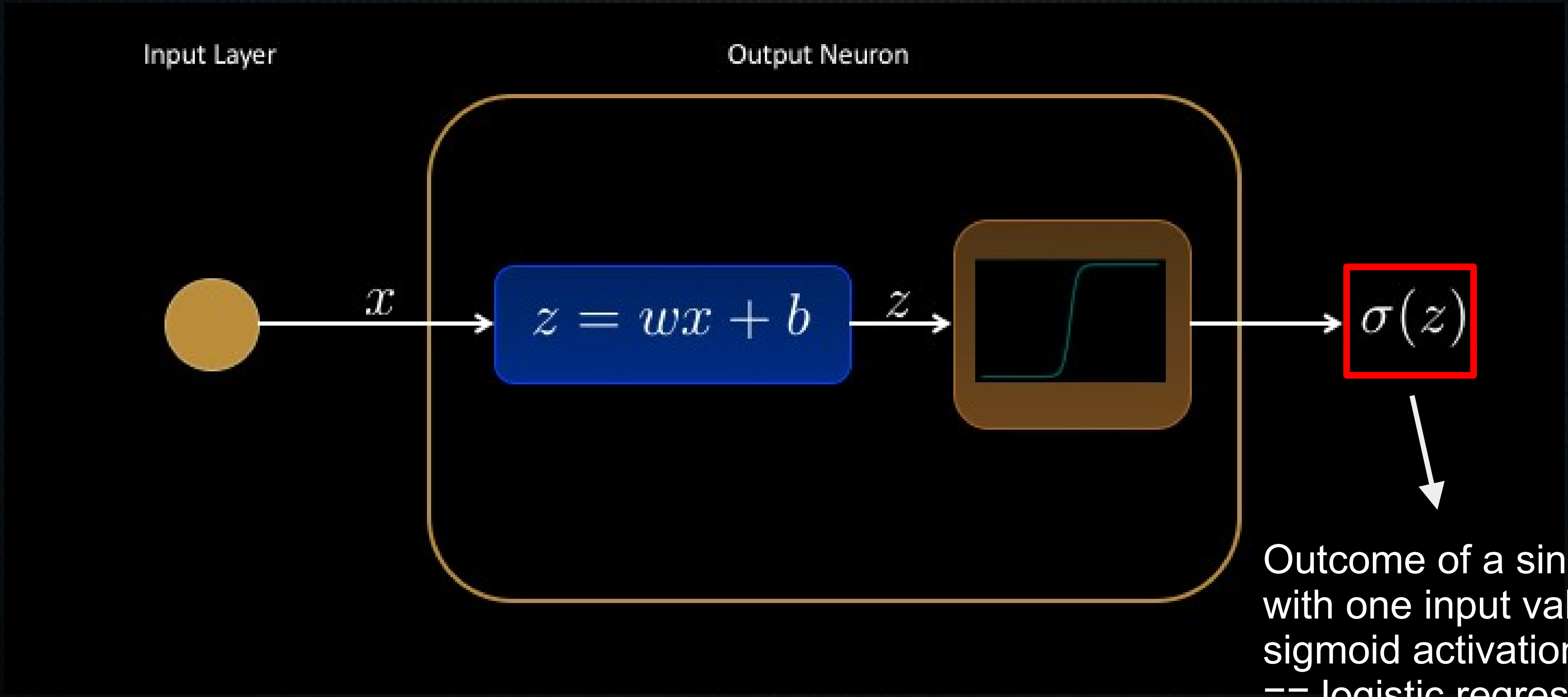
What do neural nets have to do with logistic regression?

- A single neuron is a logistic regressor!



What do neural nets have to do with logistic regression?

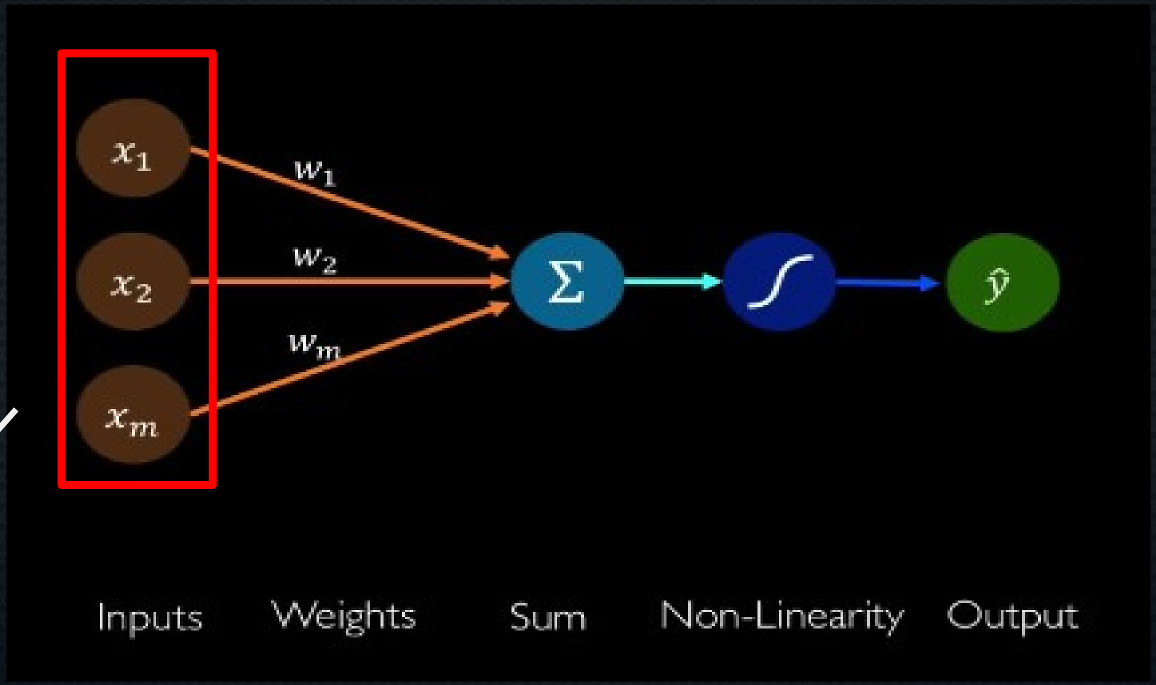
- A single neuron is a logistic regressor!



Outcome of a single neuron with one input value and sigmoid activation function == logistic regressor with 1 parameter (and bias)

What do neural nets have to do with logistic regression?

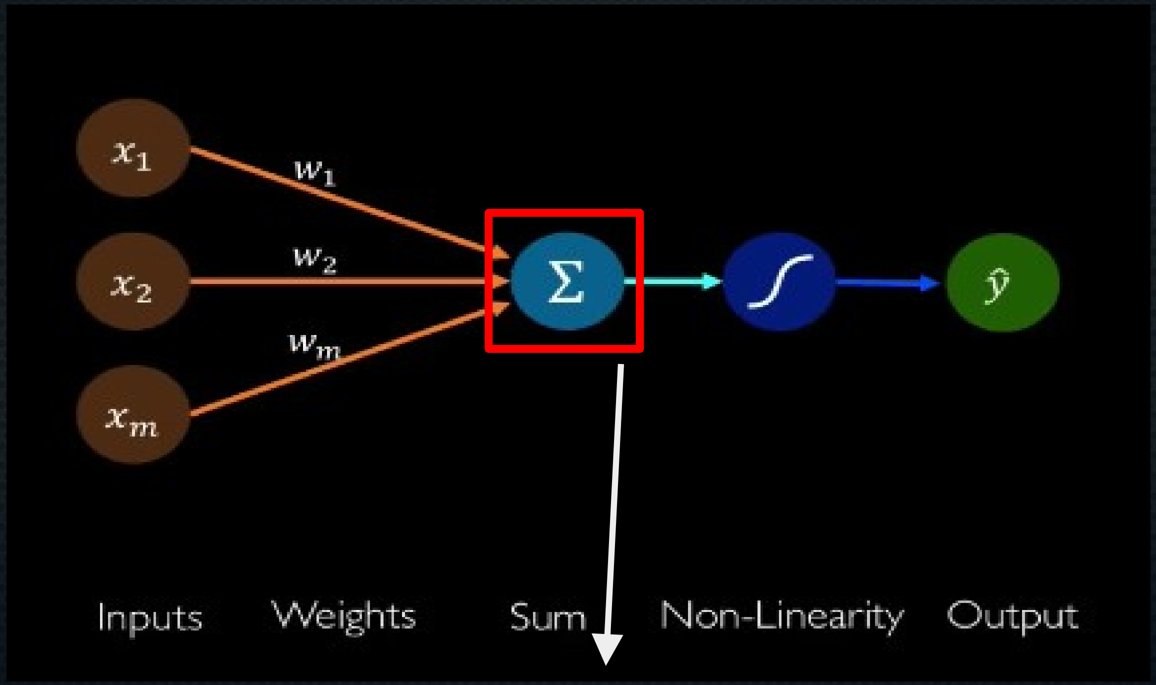
- A single neuron is a logistic regressor!



Multiple inputs: summation of those inputs' weights and the neuron's bias

What do neural nets have to do with logistic regression?

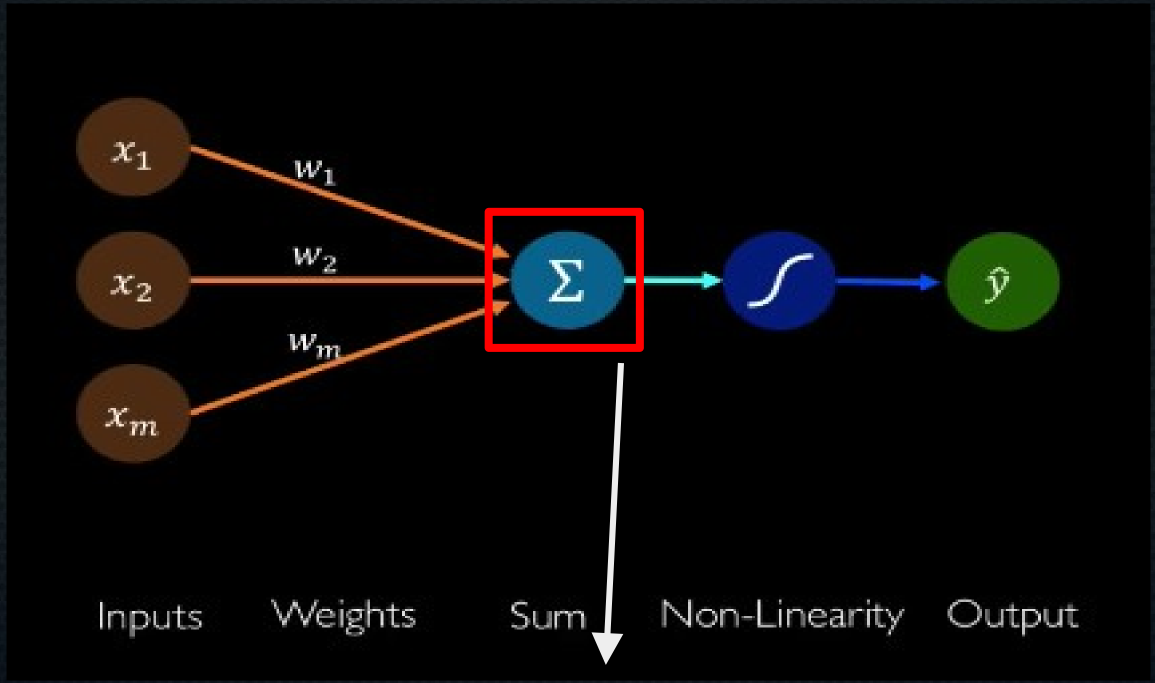
- A single neuron is a logistic regressor!



$$b+\sum_{i=1}^{m} w_i \cdot x_i$$

What do neural nets have to do with logistic regression?

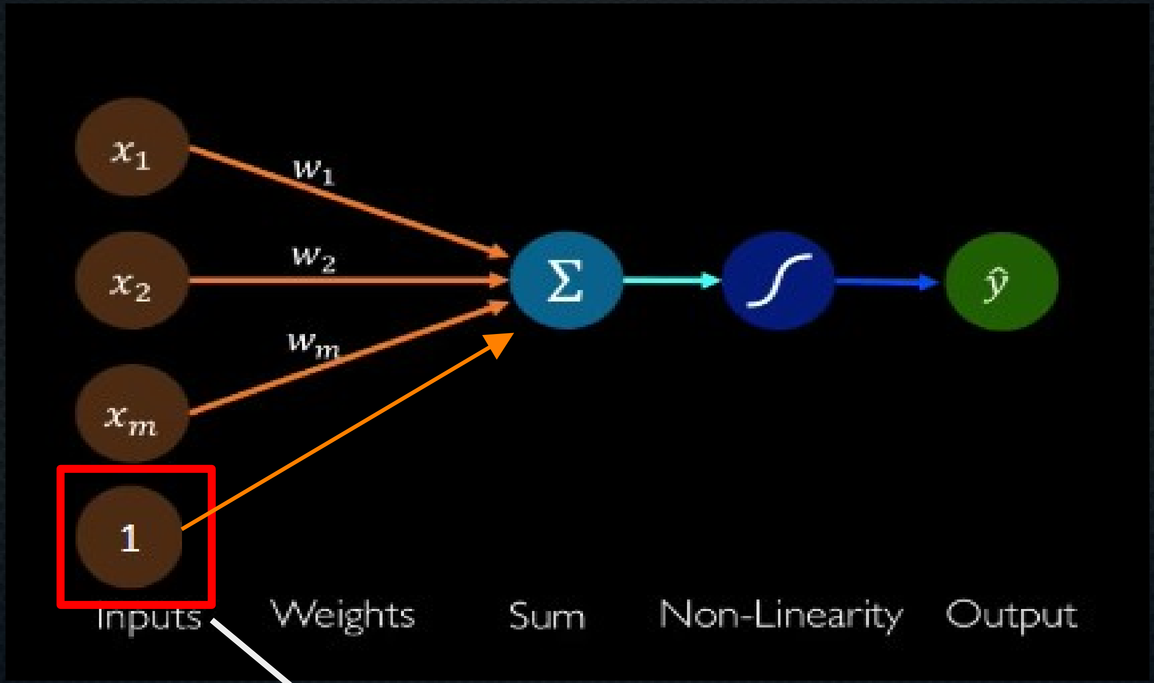
- A single neuron is a logistic regressor!



Or, in linear algebra notation: $\begin{bmatrix} 1 & x_1 & x_2 & x_m \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_m \end{bmatrix}$

What do neural nets have to do with logistic regression?

- A single neuron is a logistic regressor!

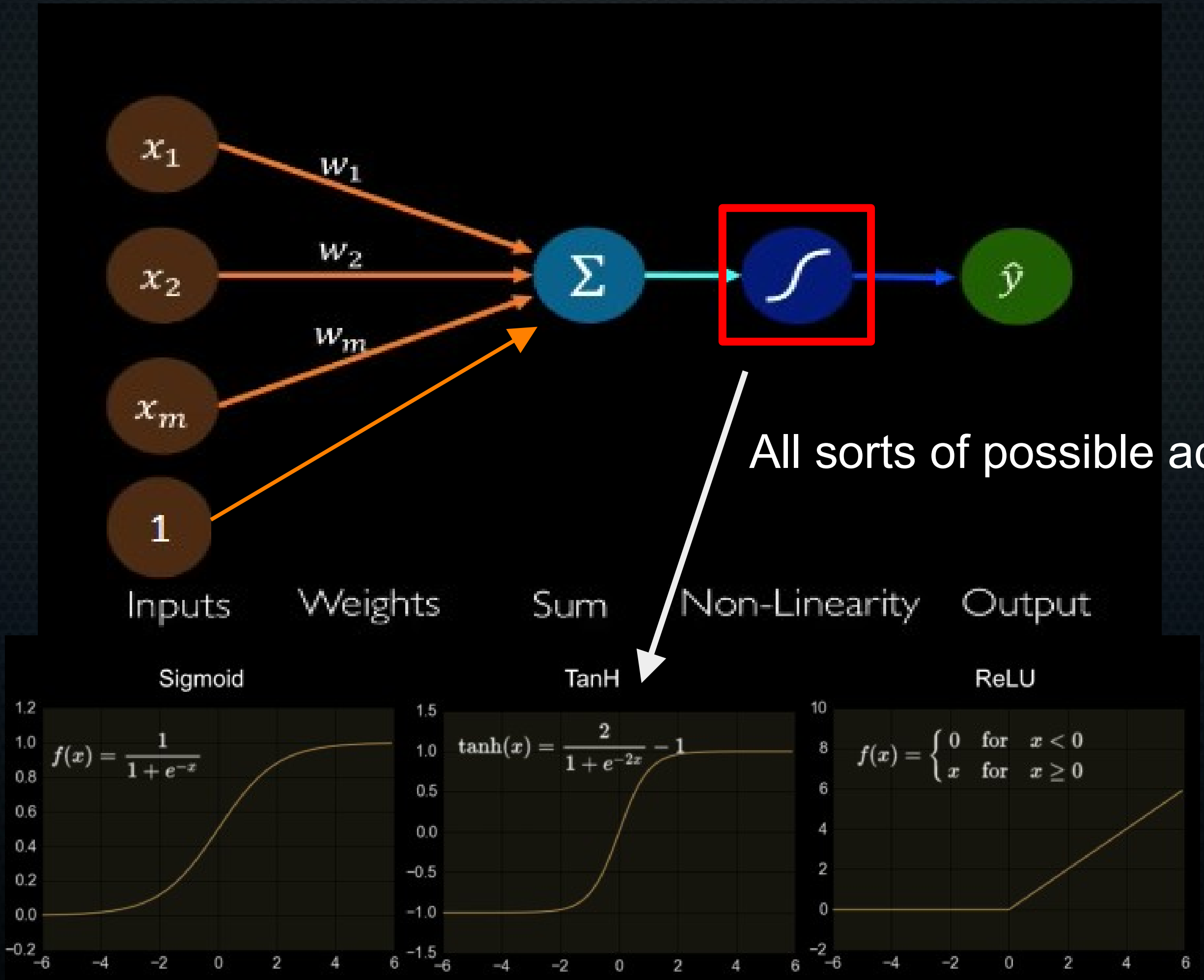


Bias unit (always 1, same for every neuron, most often not shown)

$$\begin{bmatrix} \boxed{1} & x_1 & x_2 & x_m \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_m \end{bmatrix}$$

What do neural nets have to do with logistic regression?

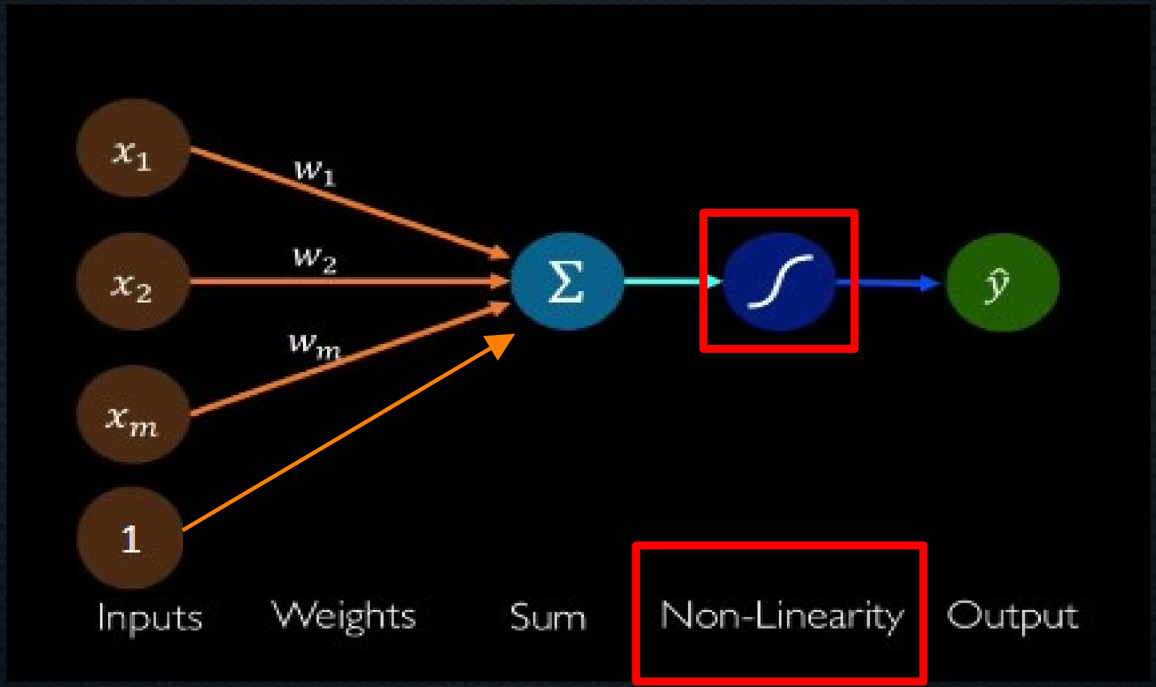
- A single neuron is a logistic regressor!



All sorts of possible activation functions

What do neural nets have to do with logistic regression?

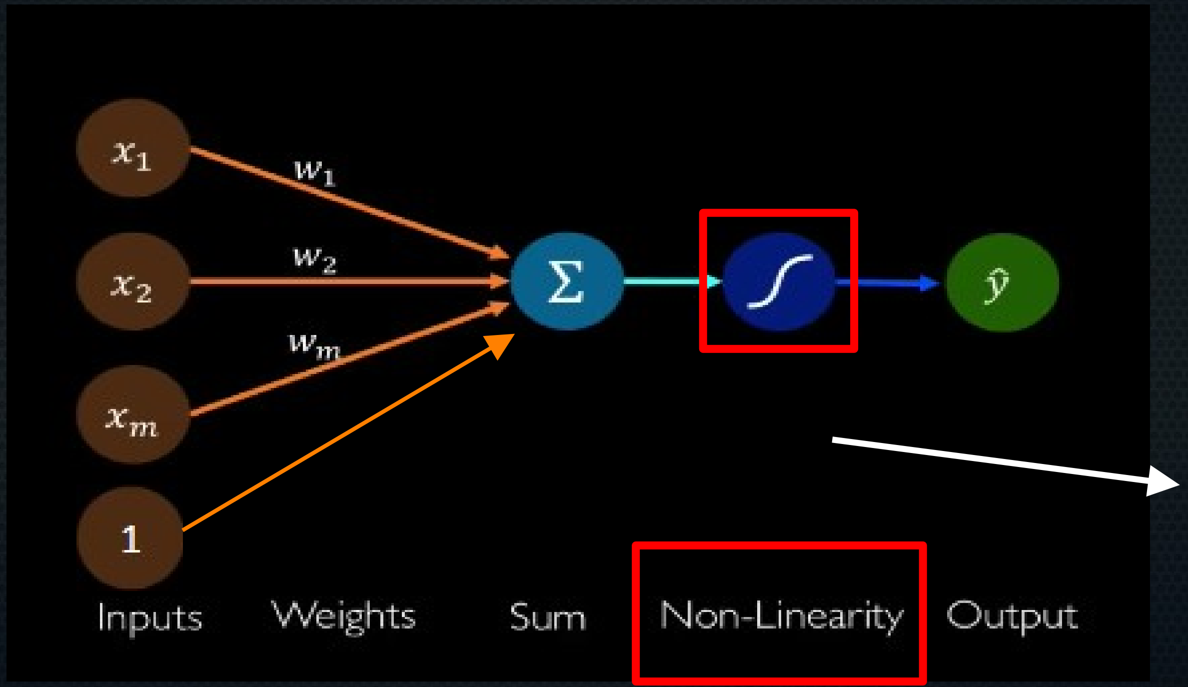
- A single neuron is a logistic regressor!

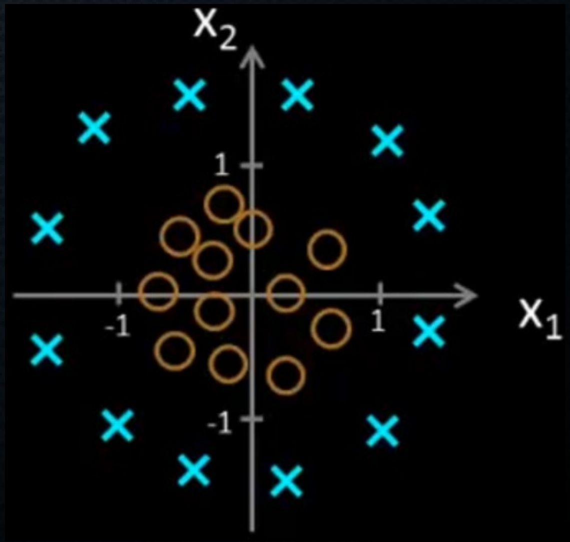


- Why non-linearity?

What do neural nets have to do with logistic regression?

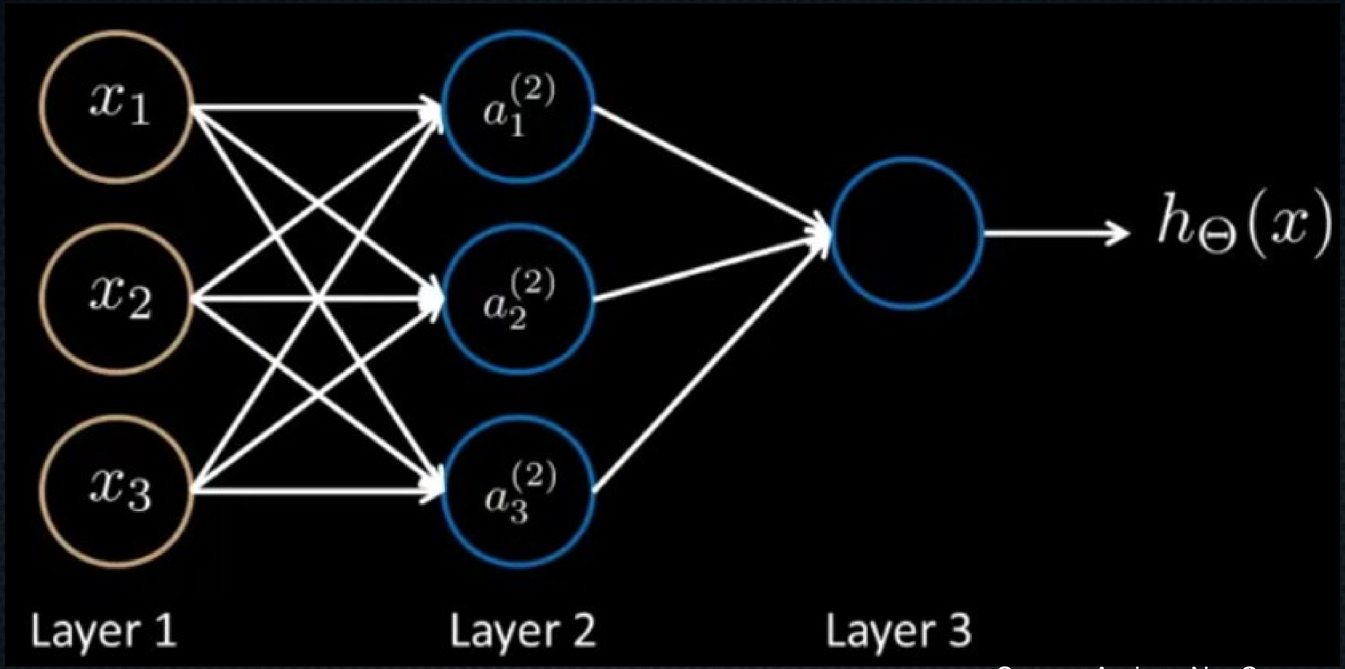
- A single neuron is a logistic regressor!





- Why non-linearity? → without them, a NN (no matter how deep) could only approximate linear functions

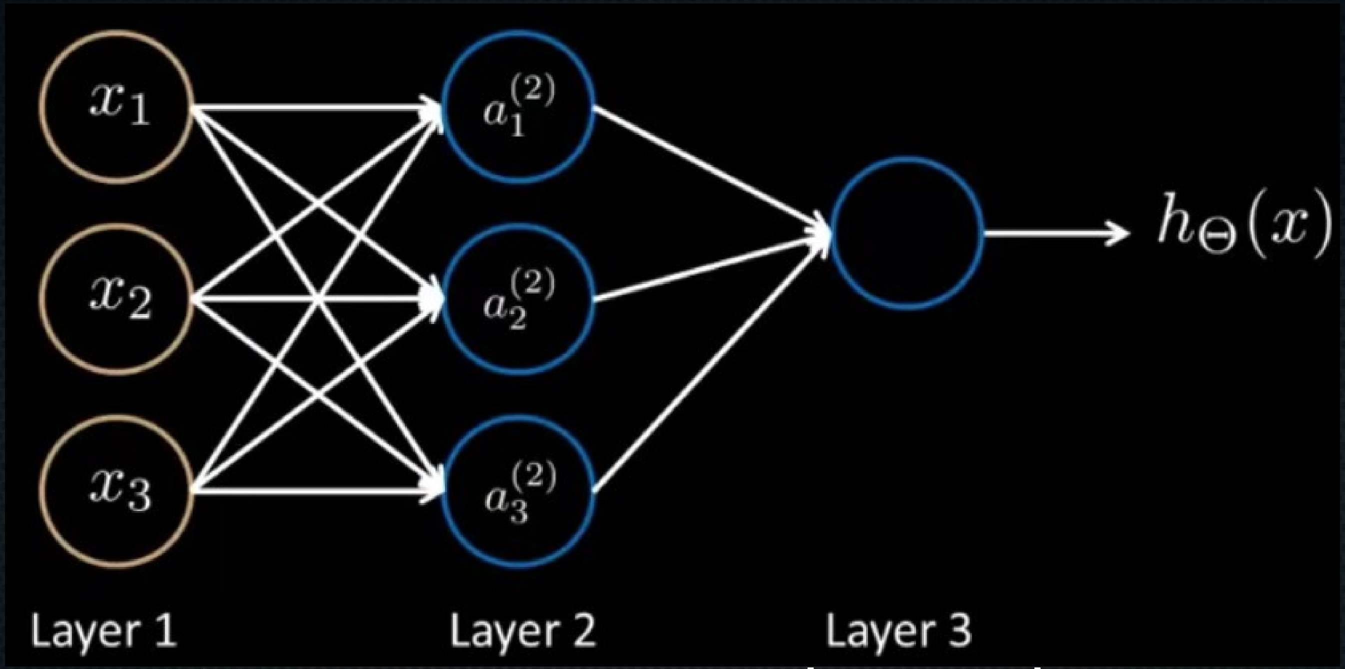
Workings of a simple neural network



Source: Andrew Ng, Coursera

Input layer: shown as neurons, are just features

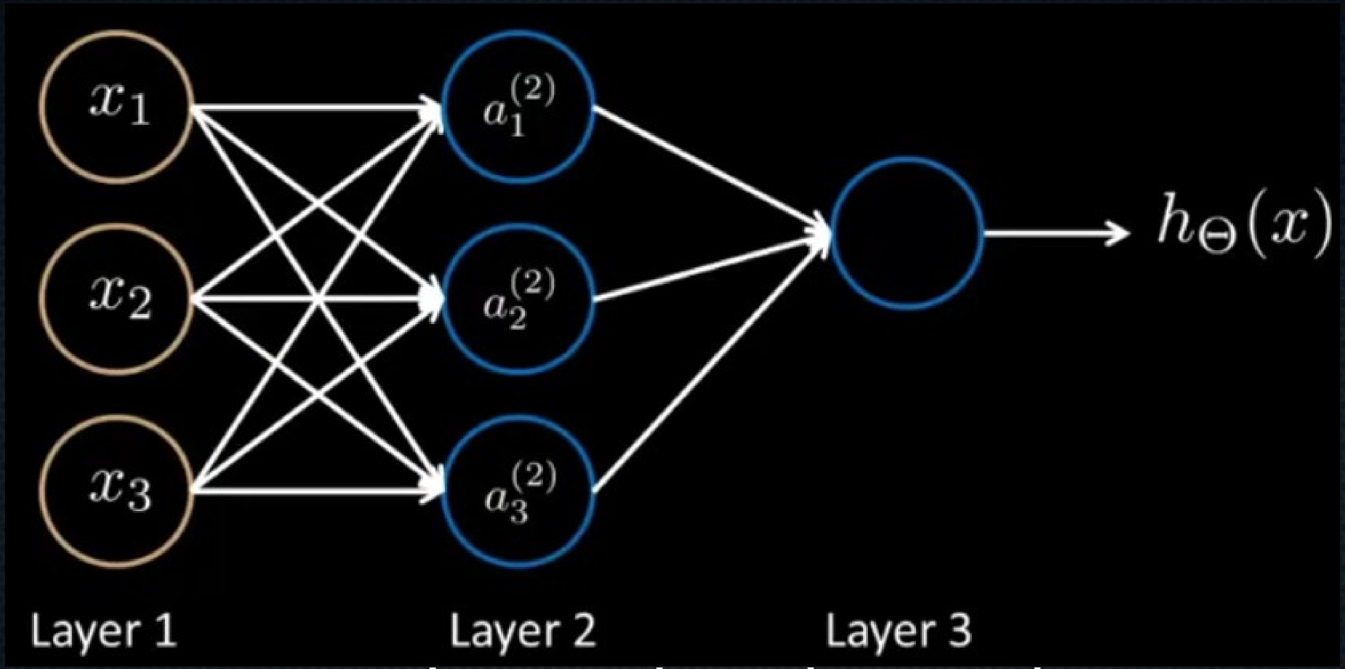
Workings of a simple neural network



Input layer: shown as neurons, are just features

Output layer: final result of the neural net

Workings of a simple neural network

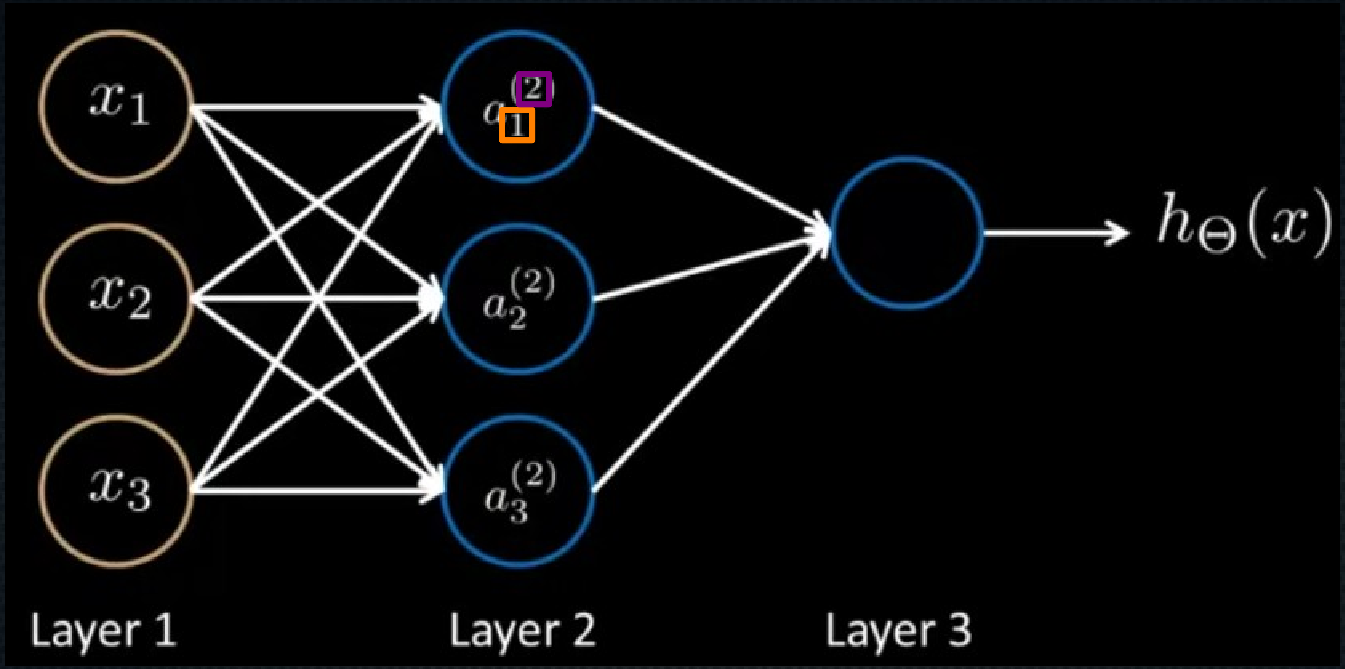


Input layer: shown as neurons, are just features

Output layer: final result of the neural net

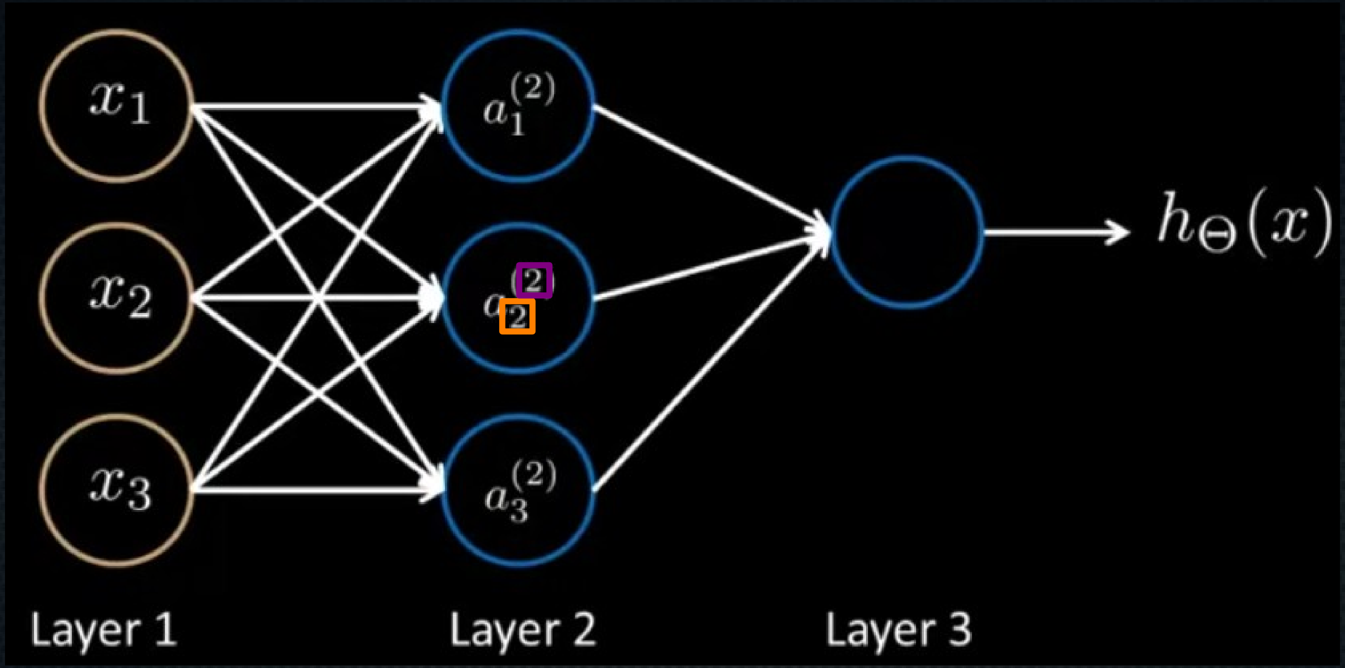
Hidden layer(s): intermediate layers whose outputs are not directly observed (hence hidden). Here: 1 HL. Facebook's DenseNet family of NNs had 121-264 HLs in 2016 (0.8-15.3 million parameters).

Workings of a simple neural network



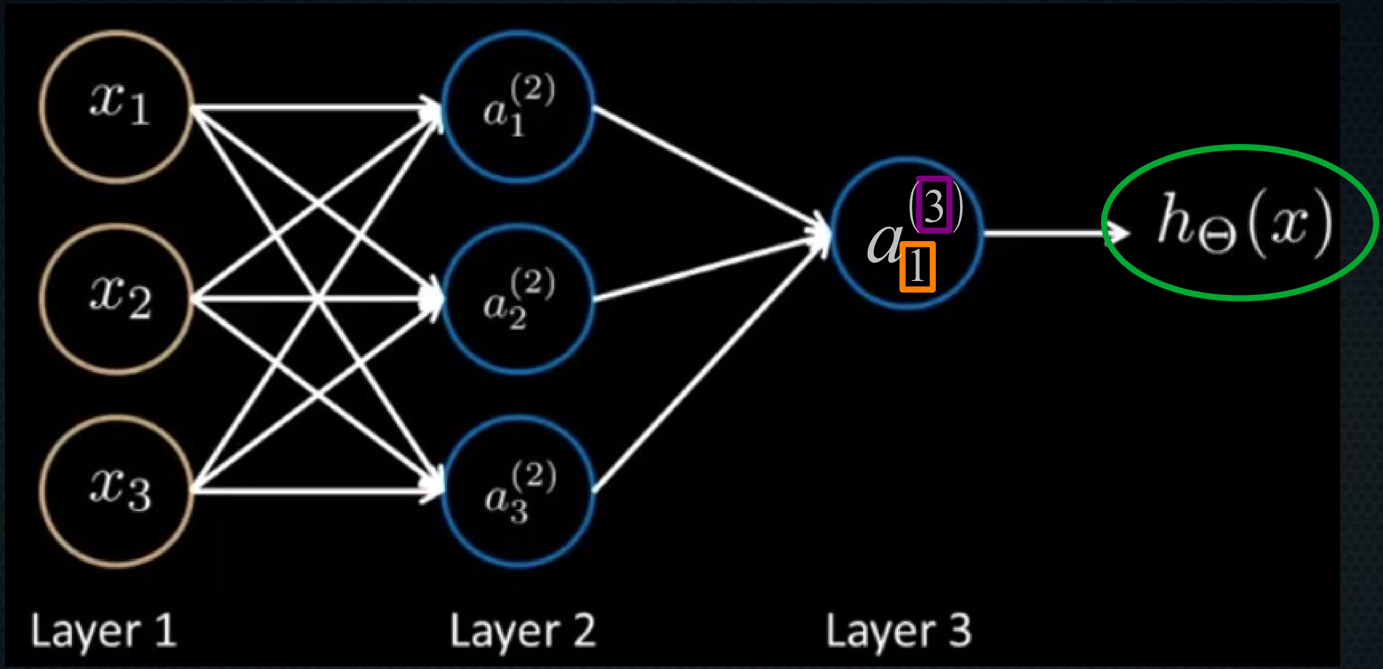
Activation of neuron 1 in the 2nd layer of the network.

Workings of a simple neural network



Activation of neuron **2** in the **2**nd layer of the network.

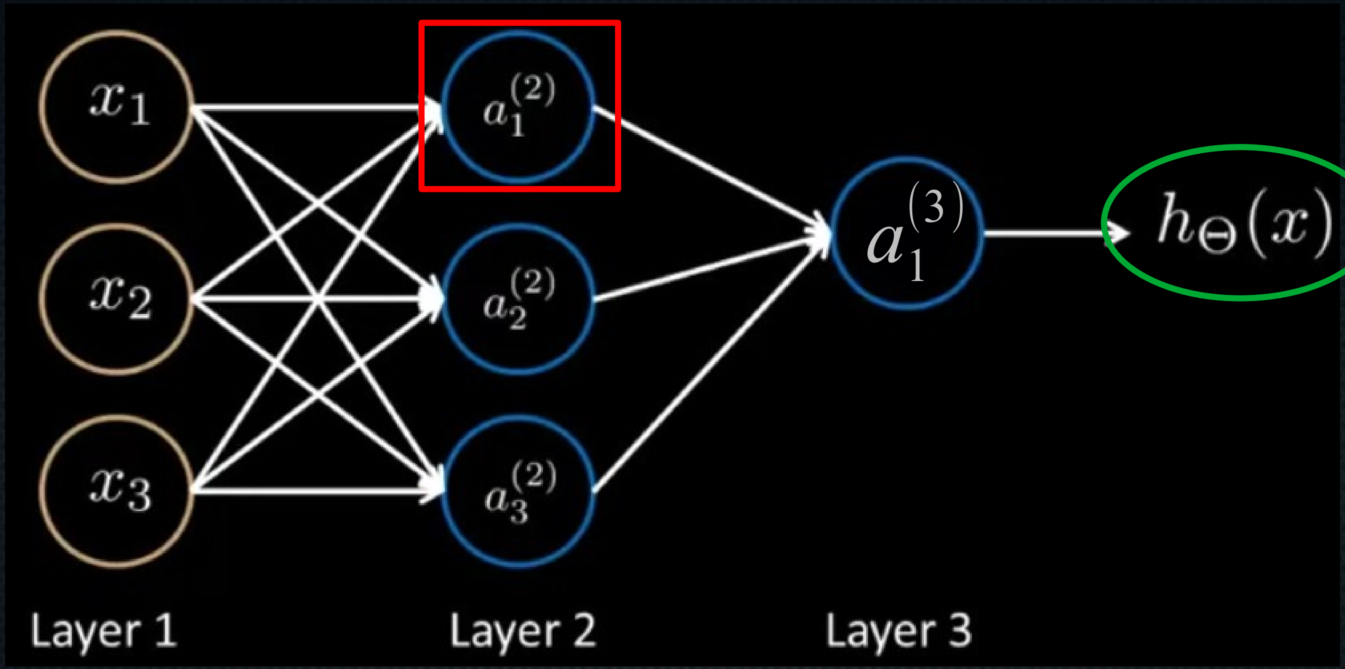
Workings of a simple neural network



https://www.louisbouchard.ai/densenet-explained/

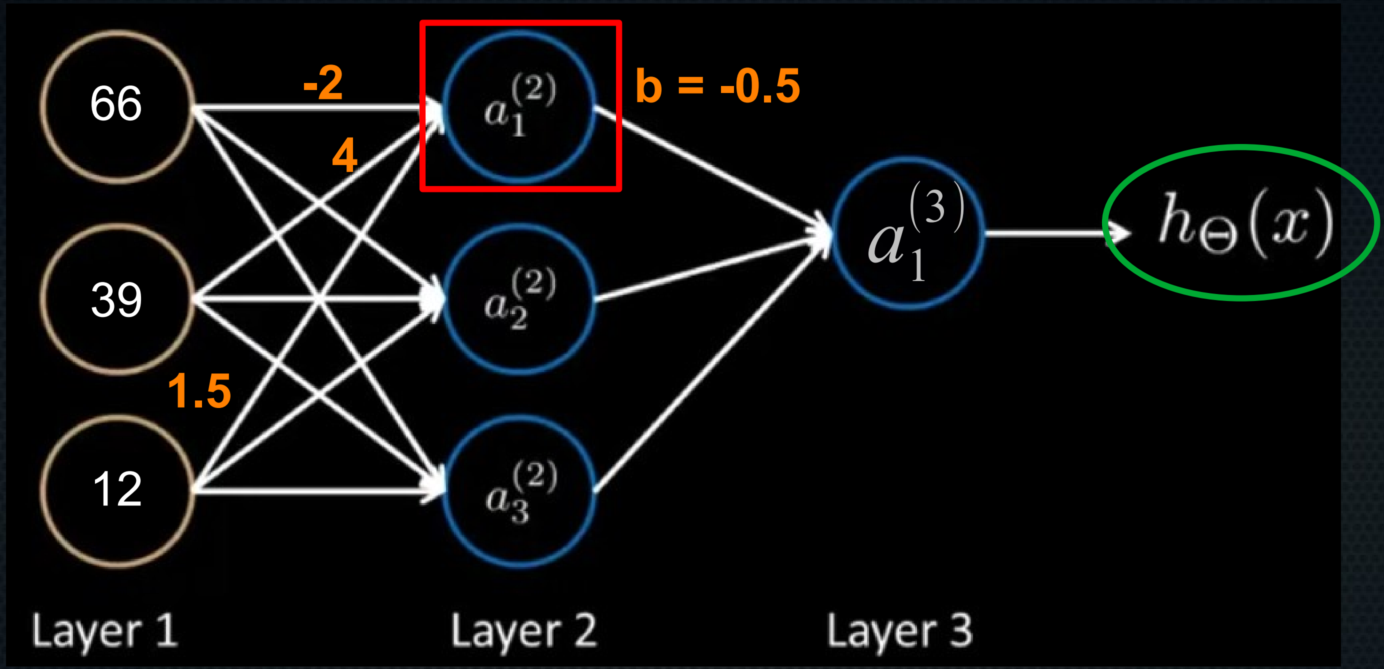
Activation of neuron **1** in the **3**rd layer of the network.

What is this network calculating?



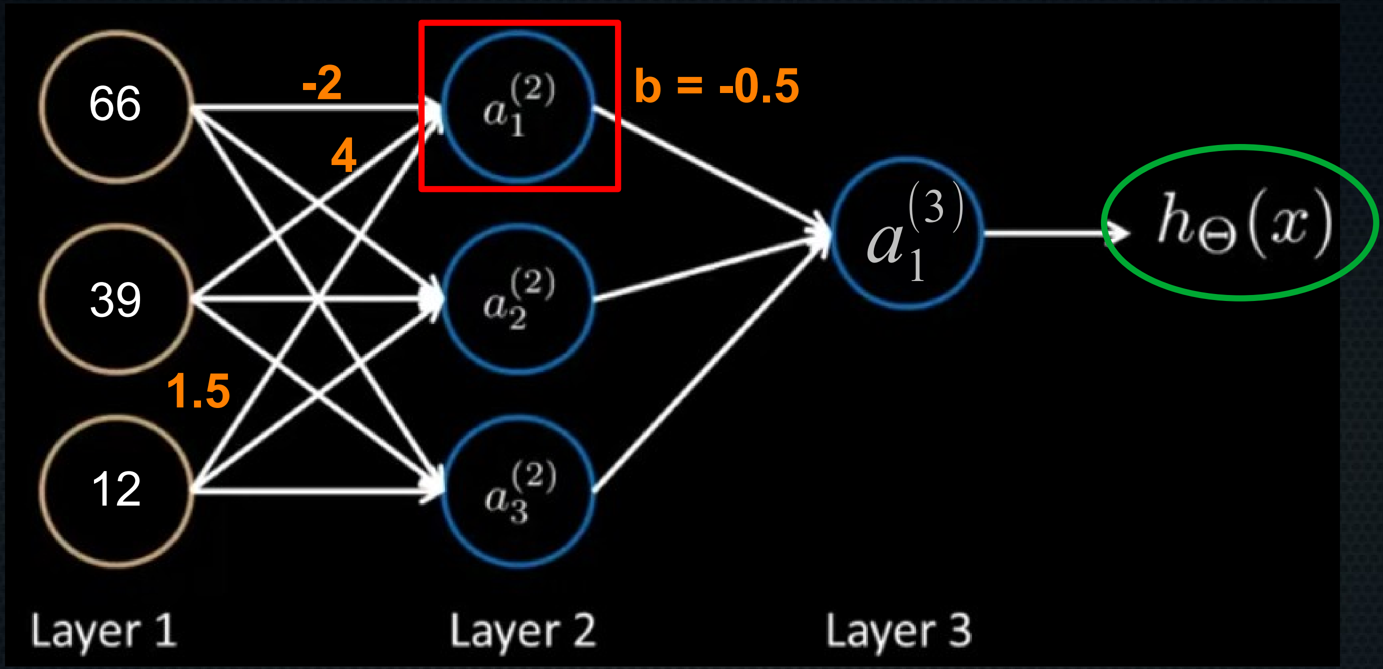
$$\sigma\left(\begin{bmatrix} 1 & x_1 & x_2 & x_3 \end{bmatrix}\begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}\right)$$

What is this network calculating?



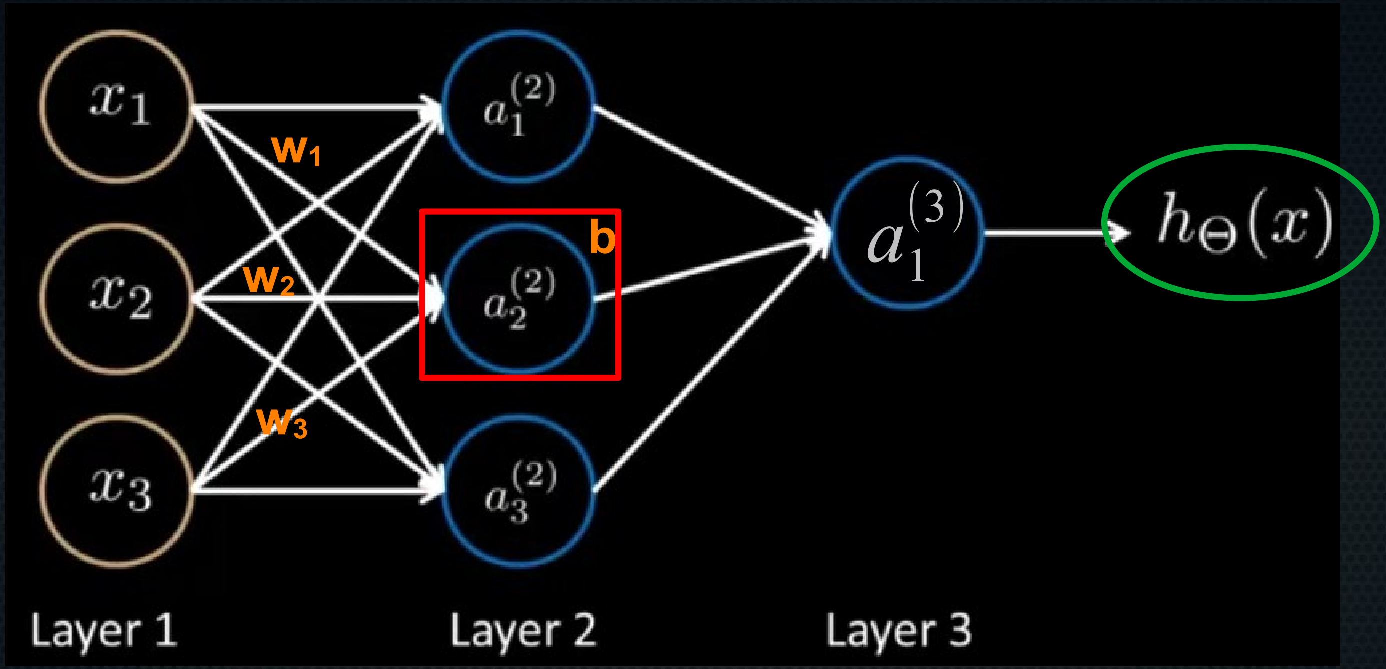
$$\sigma\left(\begin{bmatrix}1 & x_1 & x_2 & x_3\end{bmatrix}\begin{bmatrix}b \\ w_1 \\ w_2 \\ w_3\end{bmatrix}\right) \rightarrow \sigma\left(\begin{bmatrix}1 & 66 & 39 & 12\end{bmatrix}\begin{bmatrix}-0.5 \\ -2 \\ 4 \\ 1.5\end{bmatrix}\right)$$

What is this network calculating?



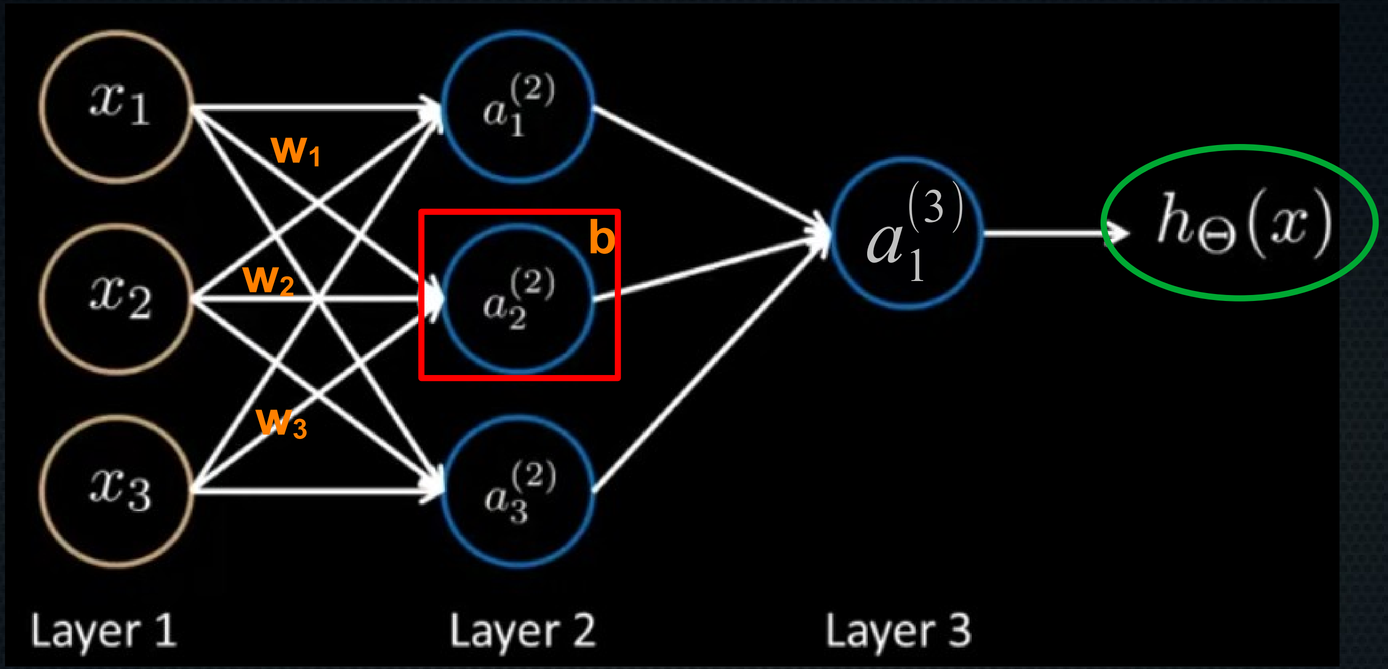
$$\sigma\left(\begin{bmatrix}1 & 66 & 39 & 12\end{bmatrix}\begin{bmatrix}-0.5\\-2\\4\\1.5\end{bmatrix}\right) \rightarrow \sigma(41.5) \rightarrow 0.999\ldots$$

What is this network calculating?



$$\sigma\left(\begin{bmatrix} 1 & x_1 & x_2 & x_3 \end{bmatrix}\begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}\right)$$

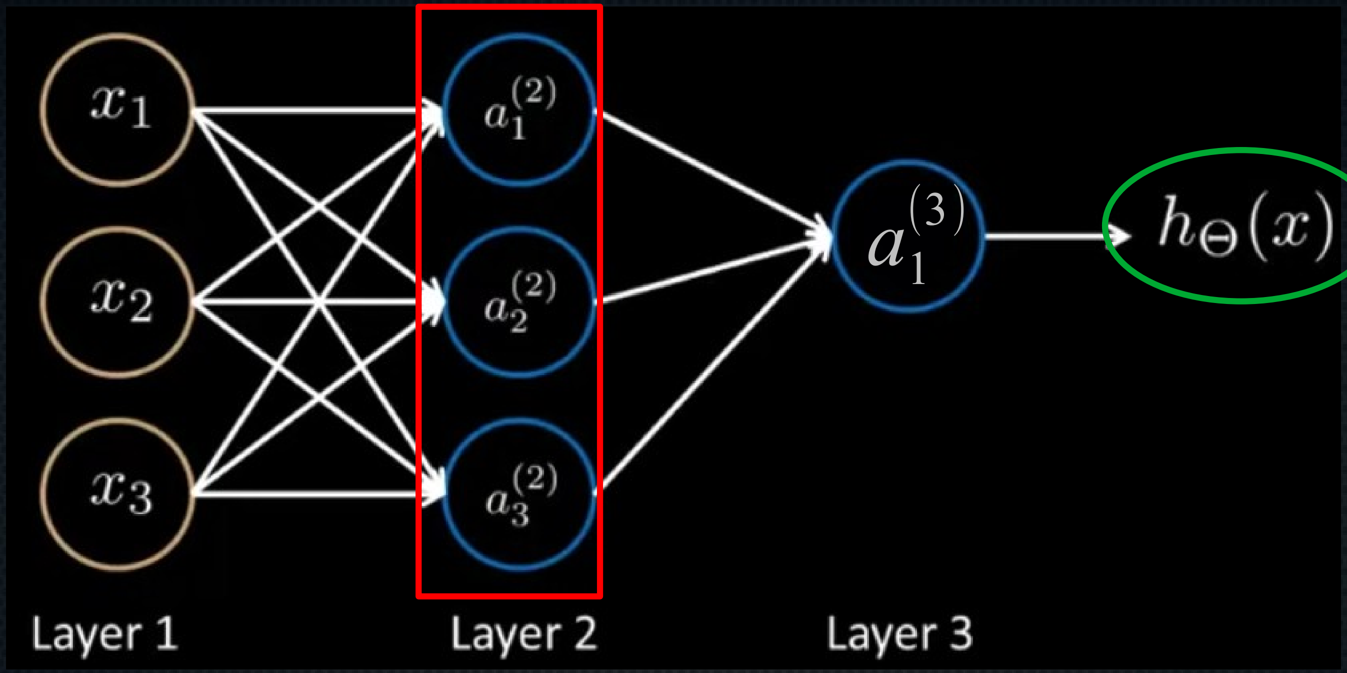
What is this network calculating?



$$\sigma\left(\begin{bmatrix} 1 & x_1 & x_2 & x_3 \end{bmatrix}\begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}\right) \longrightarrow$$

Calculate for all units at the same time with a theta matrix $\Theta^{(j)}$

What is this network calculating?

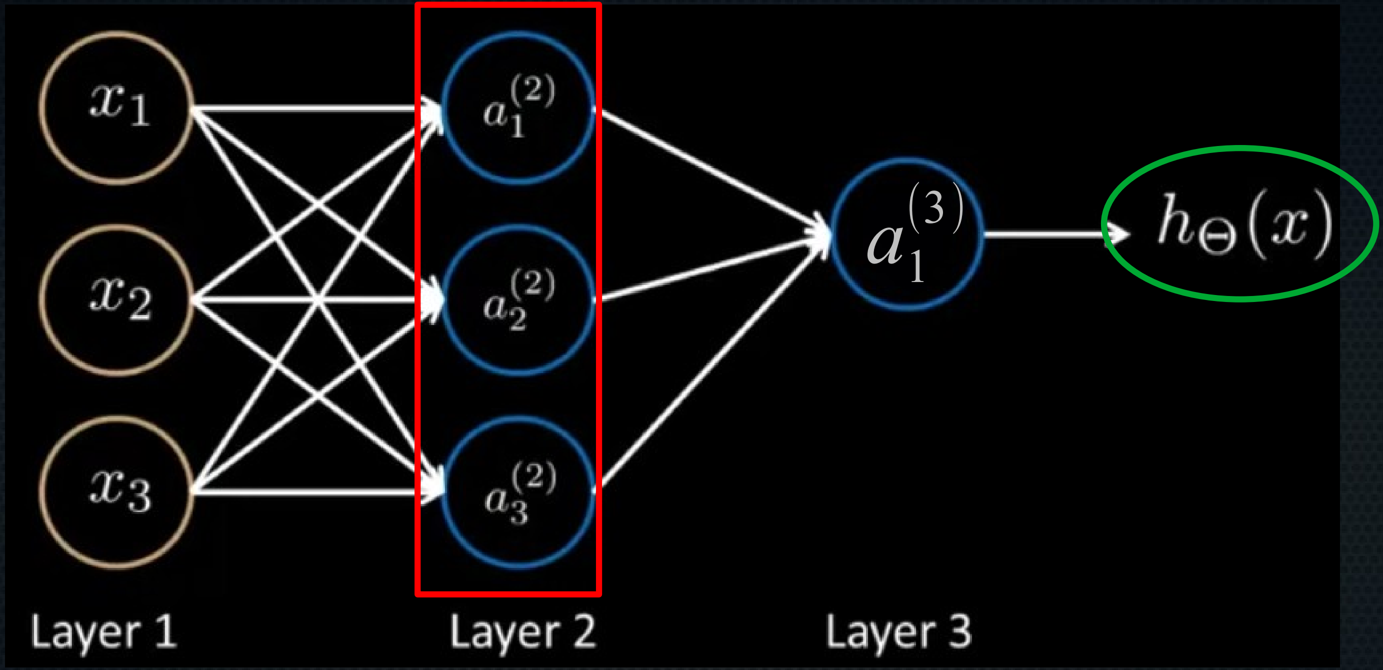


$$\sigma\left(\begin{bmatrix}1 & x_1 & x_2 & x_3\end{bmatrix}\begin{bmatrix} b_1 & b_2 & b_3 \\ w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33}\end{bmatrix}\right) \longrightarrow \sigma\left(\begin{bmatrix}1 & 2 & 3\end{bmatrix}\right)$$

$$\downarrow$$

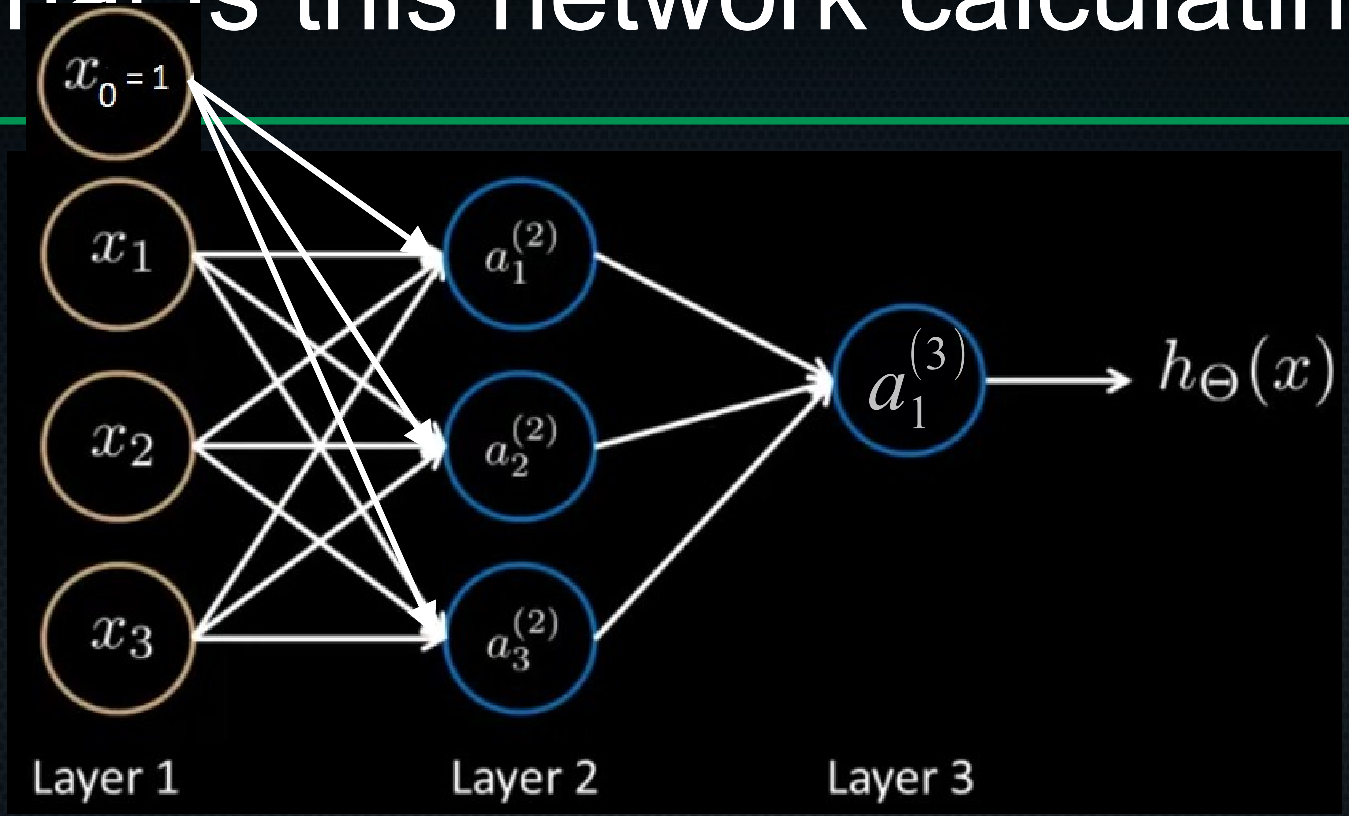
$$\boxed{\begin{bmatrix}\sigma(1) & \sigma(2) & \sigma(3)\end{bmatrix}}$$

What is this network calculating?



$$\sigma\left(\begin{bmatrix} b_1 & w_{11} & w_{12} & w_{13} \\ b_2 & w_{21} & w_{22} & w_{23} \\ b_3 & w_{31} & w_{32} & w_{33} \end{bmatrix}\begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) \longrightarrow \sigma\left(\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}\right) \longrightarrow \boxed{\begin{bmatrix} \sigma(1) \\ \sigma(2) \\ \sigma(3) \end{bmatrix}}$$

What is this network calculating?

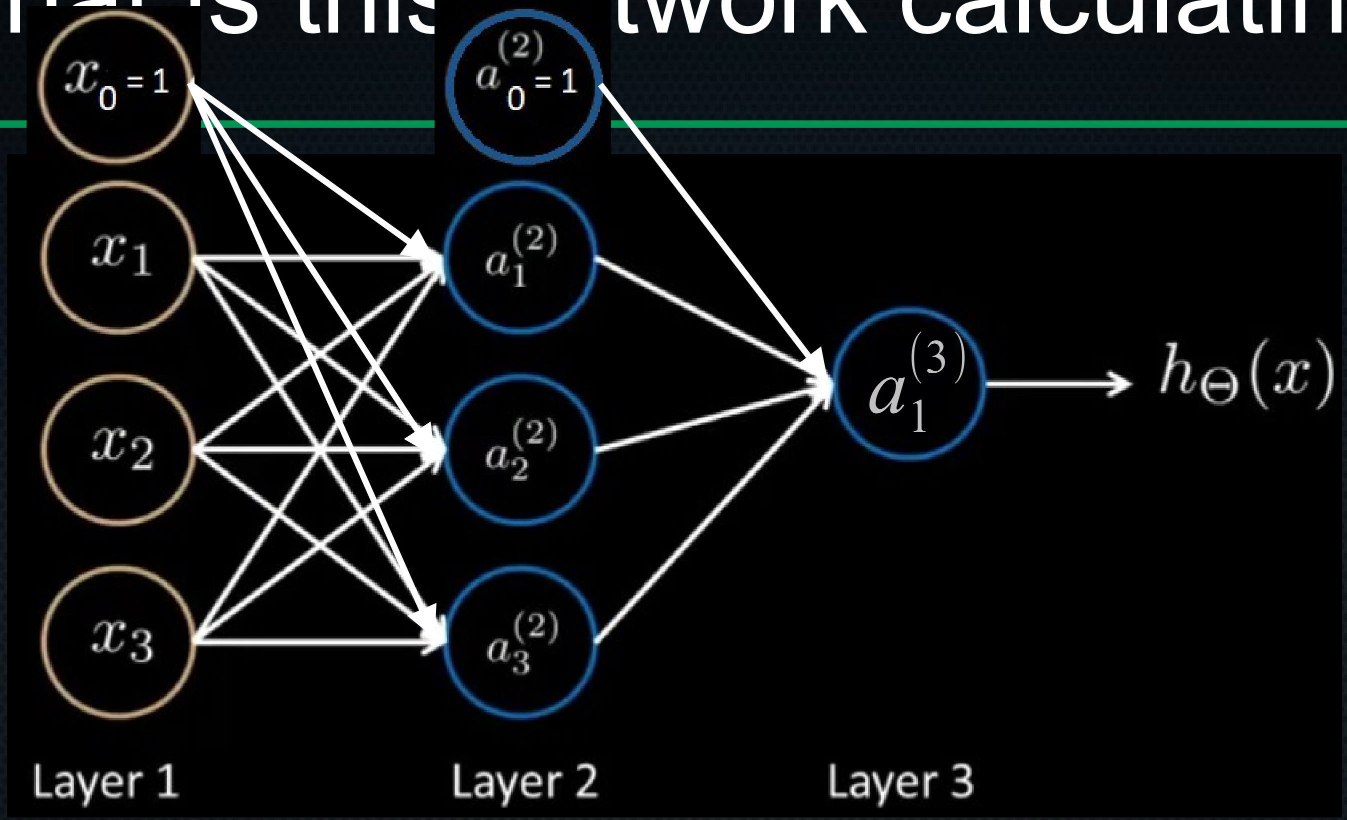


$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)}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

$\Theta^{(1)}$ (layer 1 to layer 2)

$$\begin{bmatrix} b_1 & w_{11} & w_{12} & w_{13} \\ b_2 & w_{21} & w_{22} & w_{23} \\ b_3 & w_{31} & w_{32} & w_{33} \end{bmatrix}$$

What is this network calculating?

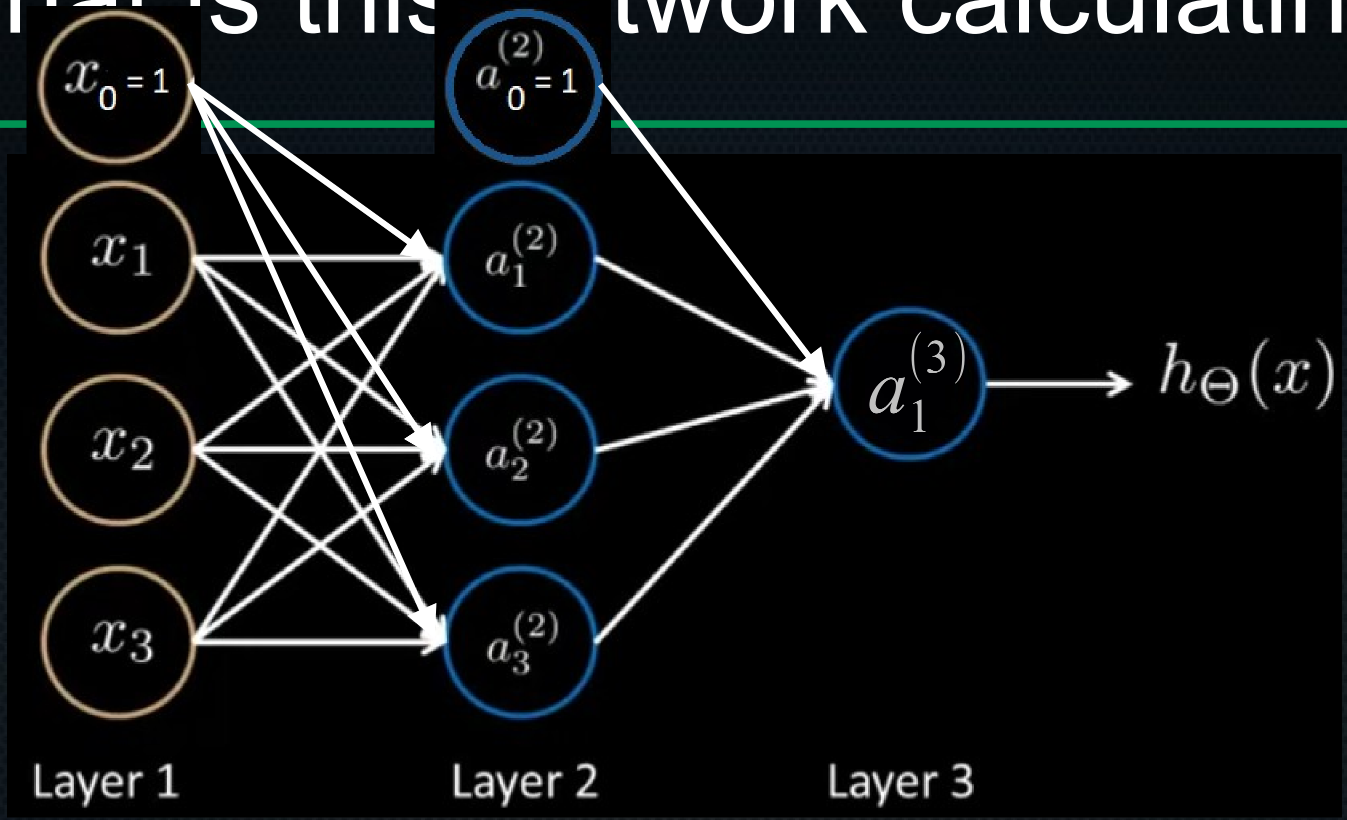


$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$\Theta^{(2)}$ (layer 2 to layer 3)

$$\begin{bmatrix} b_1 & w_{11} & w_{12} & w_{13} \end{bmatrix}$$

What is this network calculating?



- This calculation of the output of the network is called forward propagation

How do we perform?

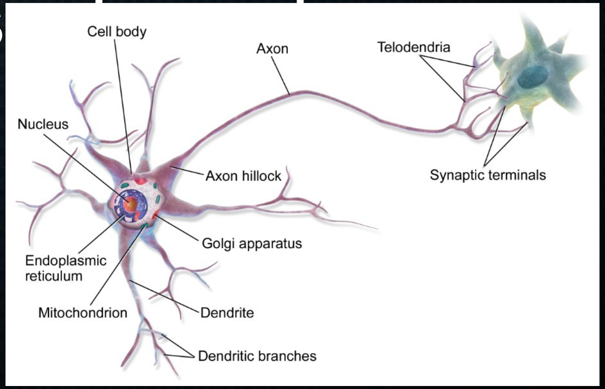
- Just like before, there is a cost function.
- But we will talk about that and its implementation tomorrow!

How do we get parameters?

- Just like before, there is a cost function and a way to minimise this. But it’s a bit more involved.
- To get parameters, we will use the principle of backpropagation. We’ll get to that tomorrow.
- First, we need to discuss *why* we want to use neural networks at all!

Why neural networks?

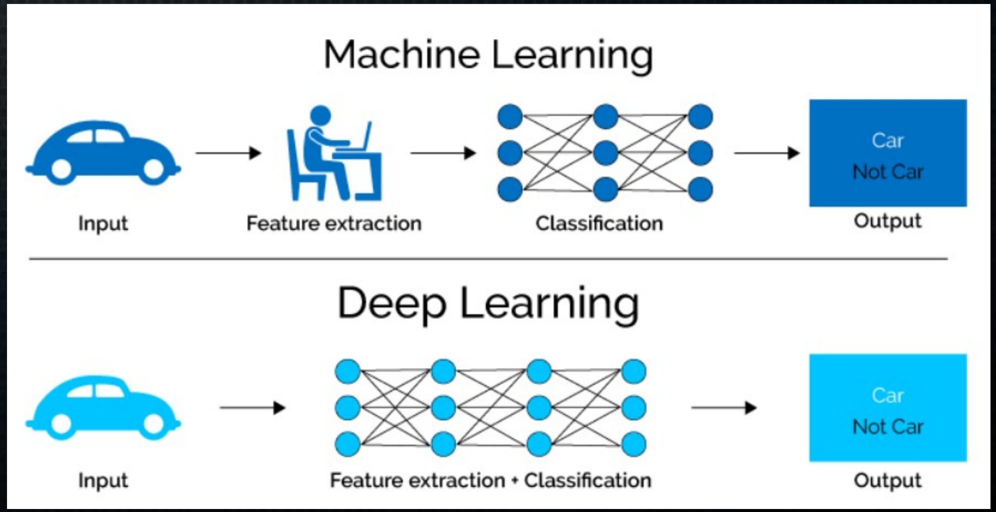
- Best performing algorithms for complex tasks, bar none.
- Known potential of hierarchical organisation of simple units because of biological examples (though neural networks are not good models of actual neurons)
- Observation in frogs and cat visual cortex: there are specific layers of neurons, where earlier layers detect basic shapes (lines, edges) with later layers incorporating this information into more complex features about what is seen.



Source: https://en.wikipedia.org/wiki/Axon#/media/File:Blausen_0657_MultipolarNeuron.png

Why neural networks?

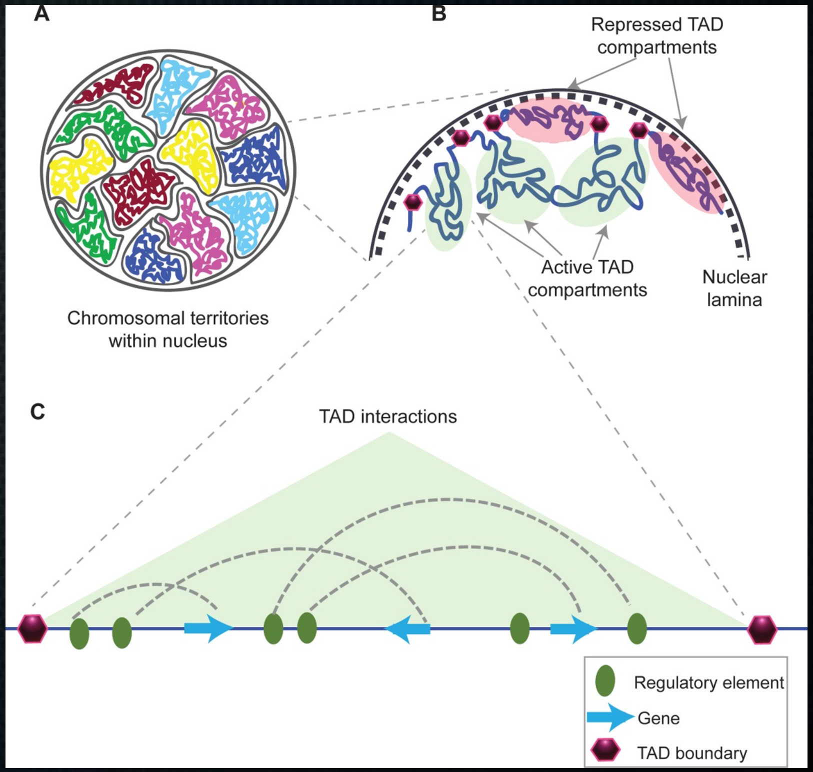
- Until now, we decided on the features to give to our algorithms: think tumour size, biopsy test scores, etc.
- With neural networks (and images), the situation changes: we don’t arduously describe what is in each image, but rather let the network learn to extract and combine features *so that* it can classify training examples correctly.



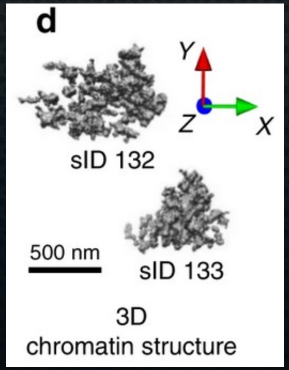
Source: https://towardsdatascience.com/cnn-application-on-structured-data-automated-feature-extraction-8f2cd28d9a7e

Why neural networks?

- Caveat for biology: it can be quite difficult to translate biological problems into a framework fit for deep learning, for reasons that will become clear later.
- Example: in images, nearby pixels probably hold similar information, i.e. are involved in the same thing. Due to (long-range) 3D-folding of DNA, linearly far DNA can be close together functionally. You need to encode network or input to accomodate this!



Source: https://en.wikipedia.org/wiki/Topologically_associating_domain#/media/File:Structural_organization_of_chromatin.png



Source: Trzaskoma, P., Ruszczycki, B., Lee, B., Pels, K. K., Krawczyk, K., Bokota, G., ... & Wilczyński, G. M. (2020). Ultrastructural visualization of 3D chromatin folding using volume electron microscopy and DNA in situ hybridization. Nature communications, 11(1), 1-9.

Why neural networks?

- The mythical property of *universal approximation*. This says that neural networks can approximate any function with arbitrary accuracy, even with only **1** hidden layer (given enough neurons in it).

Why neural networks?

- The mythical property of *universal approximation*. This says that neural networks can approximate any function with arbitrary accuracy, even with only **1** hidden layer (given enough neurons in it).
- Of course, that doesn't necessarily mean we would have the data to *train* such a neural network efficiently. Just that it is provable that for *any continuous function* a neural network can exist that approximates it as well as you like.

Recap so far

- Neurons in neural networks are not really like biological neurons, except superficially
- Neural networks can be thought of as hiërarchical sets of logistic regressors
- We essentially make earlier layers learn useful features for distinction on their own, and can use these best possible learned features for the classification by the final unit(s)
- Parsing an example through the network and getting the output is called forward propagation
- *Universal approximation* holds that, in principle, neural networks can learn any continuous function arbitrarily well

Making simple functions ourselves

- Networks with a single hidden layer can approximate any function. Let‘s get some more intuition by building our own logic circuit.

Making simple functions ourselves

- Networks with a single hidden layer can approximate any function. Let‘s get some more intuition by building our own logic circuit.
- Boolean operators:

NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

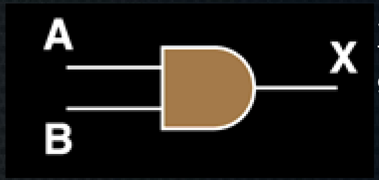
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Source: https://introcs.cs.princeton.edu/java/71boolean/

Making simple functions ourselves

- Networks with a single hidden layer can approximate any function. Let‘s get some more intuition by building our own logic circuit.
- Boolean operators:

Only if two incoming connections are on, the output is on



Source: https://www.electronics-tutorial.net/digital-logic-gates/and-gate/

NOT

x	x'
0	1
1	0

$\boxed{AND}$

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Source: https://introcs.cs.princeton.edu/java/71boolean/

Making simple functions ourselves

- Networks with a single hidden layer can approximate any function. Let‘s get some more intuition by building our own logic circuit.
- Boolean operators:

Only if either incoming connection is on, the output is on

Source: https://projectiot123.com/2019/05/26/introduction-to-xor-gate/

NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

$\boxed{\text{XOR}}$

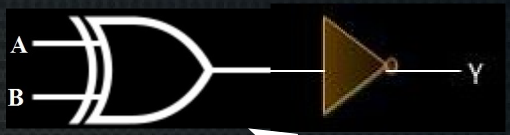
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Source: https://introcs.cs.princeton.edu/java/71boolean/

Making simple functions ourselves

- Networks with a single hidden layer can approximate any function. Let‘s get some more intuition by building our own logic circuit.
- Boolean operators:

Combine: XNOR



Source: https://www.tutorialspoint.com/computer_logical_organization/logic_gates.htm

NOT

x	x'
0	1
1	0

XNOR

Inputs		Output
A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

XOR

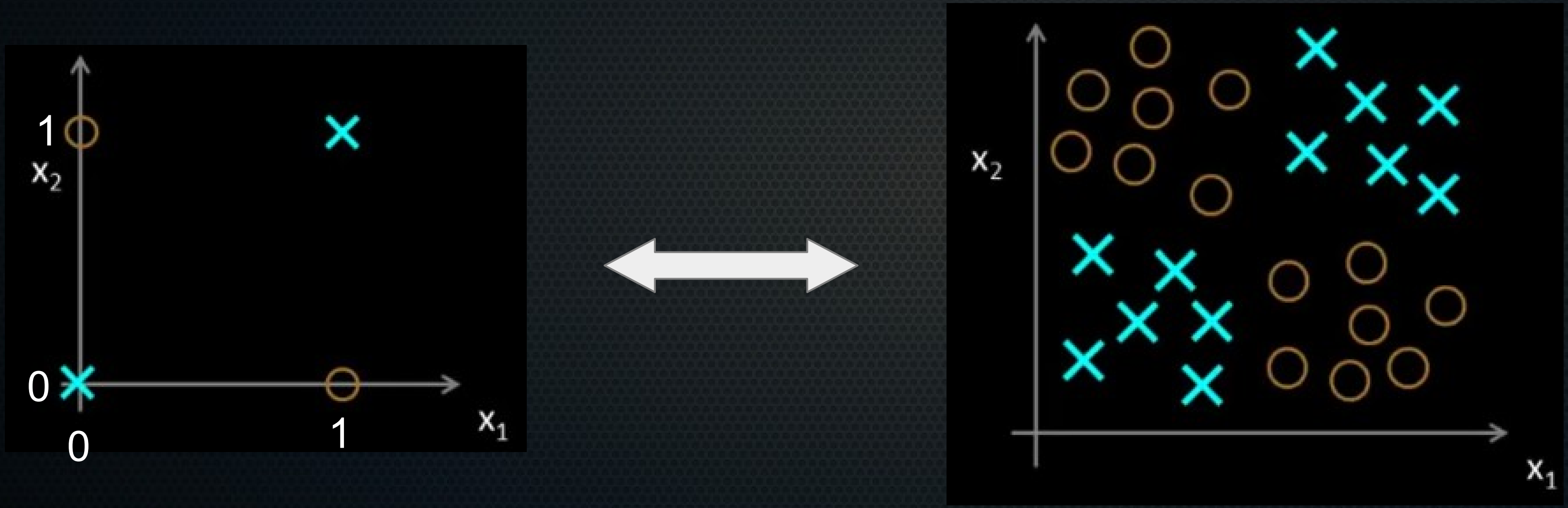
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Source: https://introcs.cs.princeton.edu/java/71boolean/

See for more info: https://www.tutorialspoint.com/computer_logical_organization/logic_gates.htm

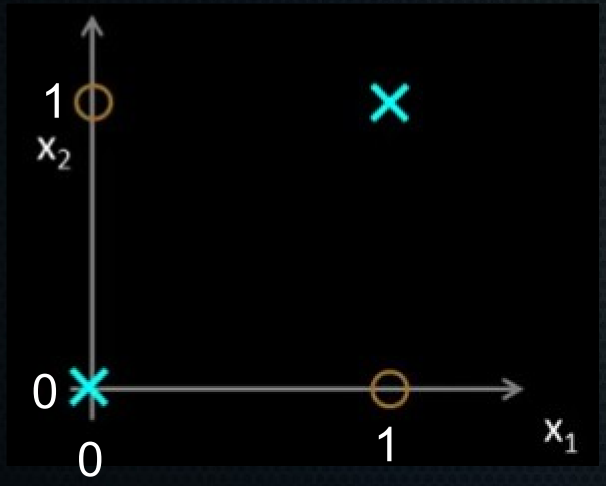
Making simple functions ourselves

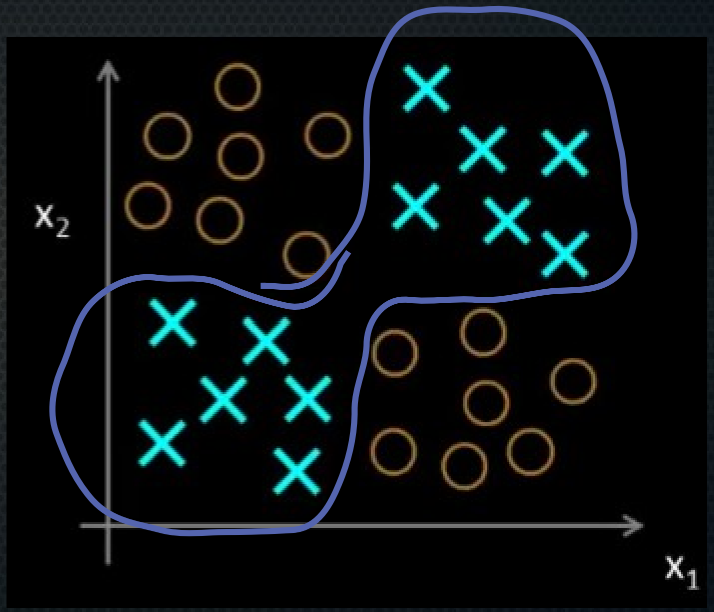
- Motivating example:



Making simple functions ourselves

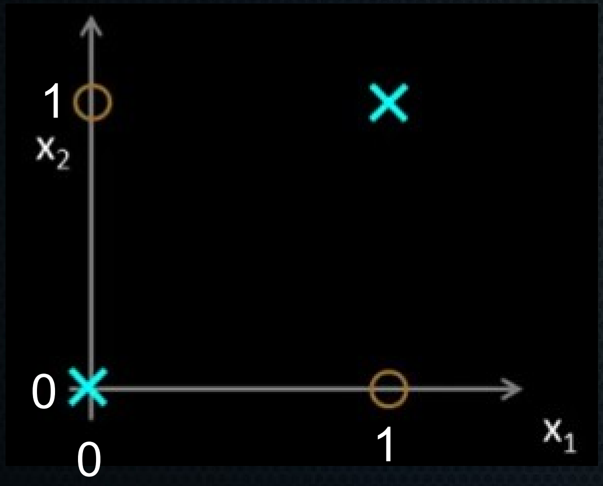
- Motivating example:





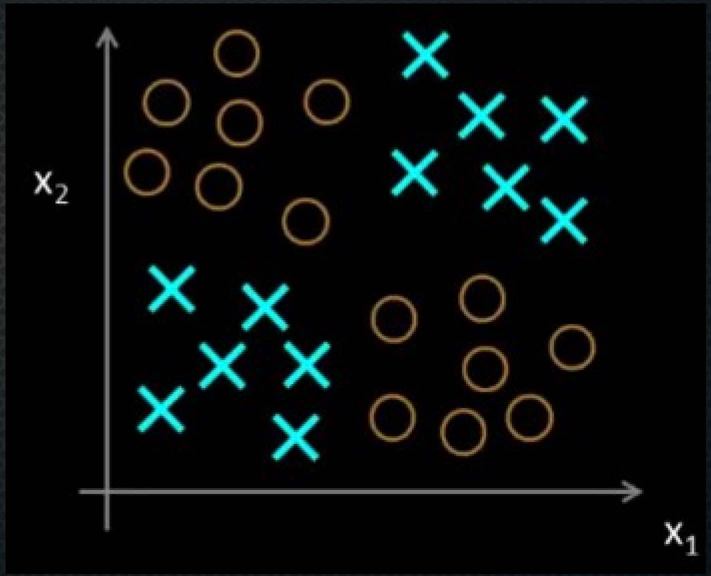
Making simple functions ourselves

- Motivating example:



 = 0

 = 1

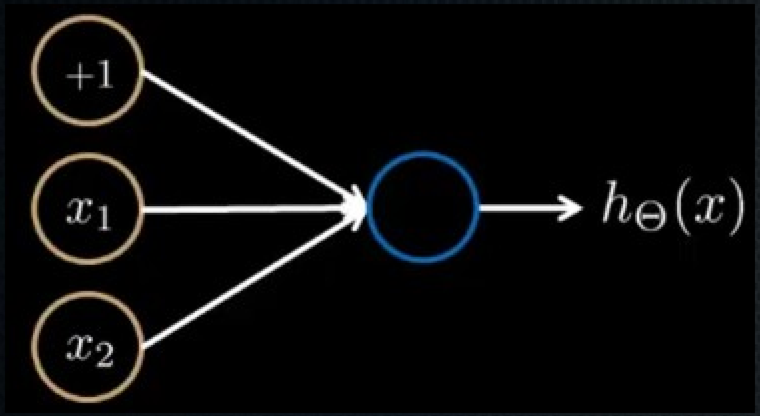


x_1 XNOR x_2

Inputs		Output
x_1	x_2	x_1 XNOR x_2
0	0	1
0	1	0
1	0	0
1	1	1

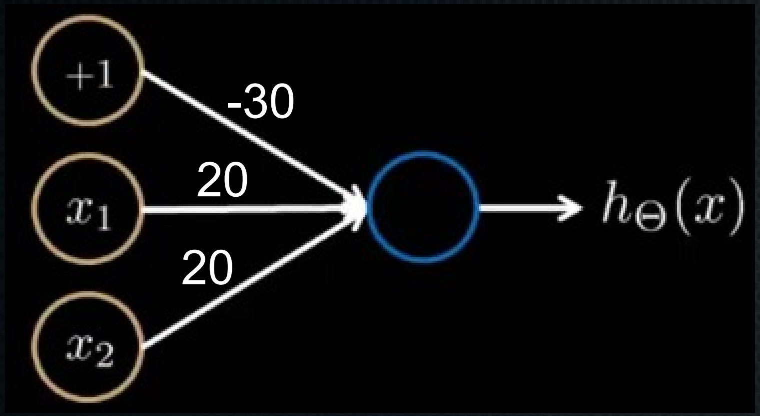
Making simple functions ourselves

- AND function → x_1 and x_2 can be either 0 or 1



Making simple functions ourselves

- AND function → x_1 and x_2 can be either 0 or 1

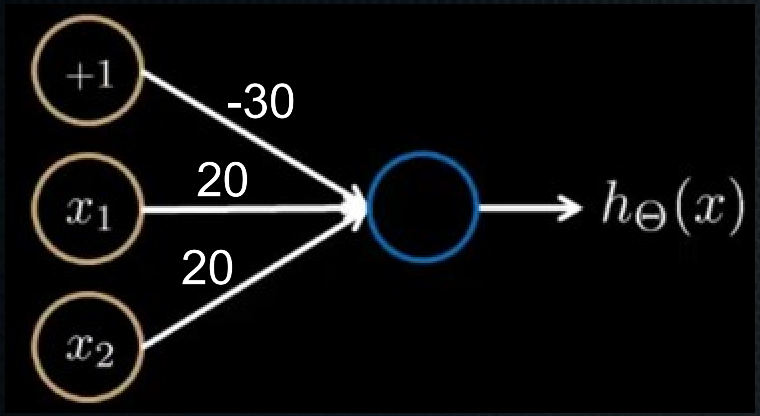


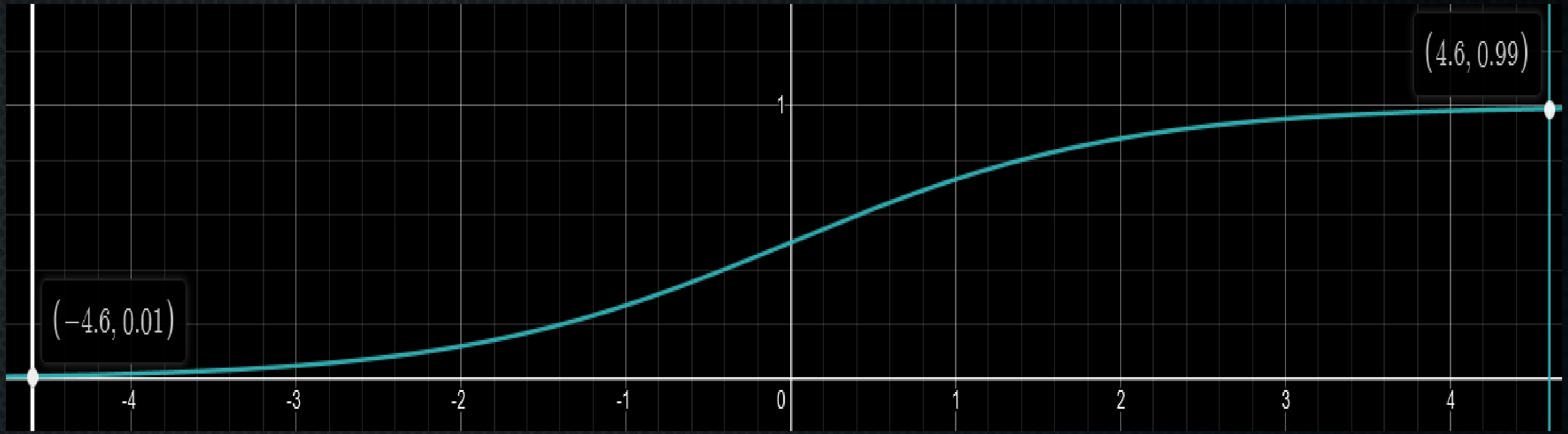
$$h_\theta(x) = sigmoid(-30 + 20 \cdot x_1 + 20 \cdot x_2)$$

$$h_\theta(x) = \sigma\left(\begin{bmatrix} -30 & 20 & 20 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}\right)$$

Making simple functions ourselves

- AND function → x_1 and x_2 can be either 0 or 1

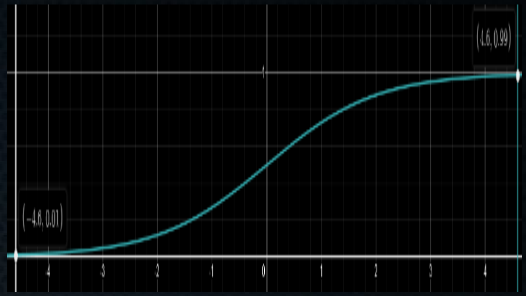


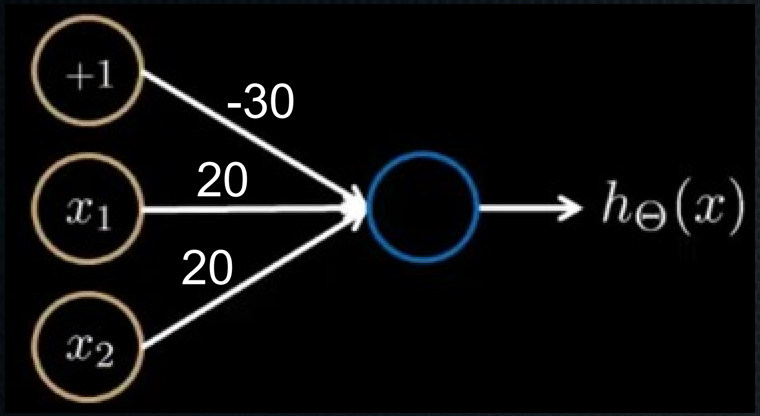


$$h_\theta(x) = sigmoid(-30 + 20 \cdot x_1 + 20 \cdot x_2)$$

Making simple functions ourselves: AND

- AND function → x_1 and x_2 can be either 0 or 1



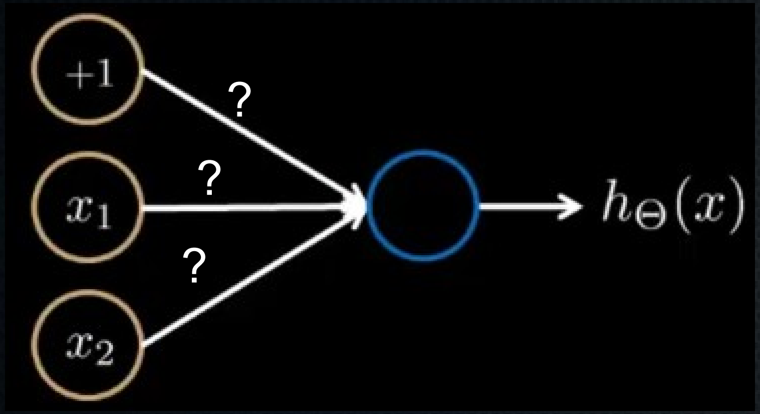


$$h_\theta(x) = sigmoid(-30 + 20 \cdot x_1 + 20 \cdot x_2)$$

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	$g(10) \approx 1$

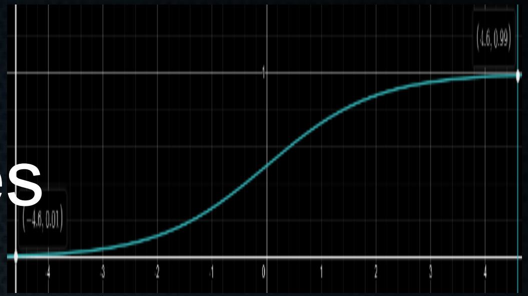
Making simple functions ourselves: OR

- Over to you: make an OR function → try it yourself and discuss with neighbours for 2 minutes



OR

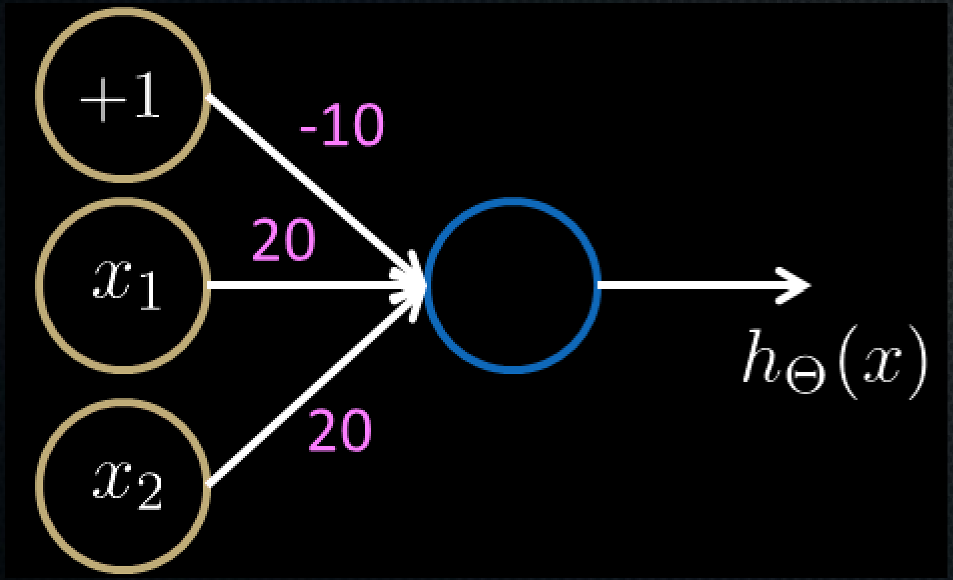
x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

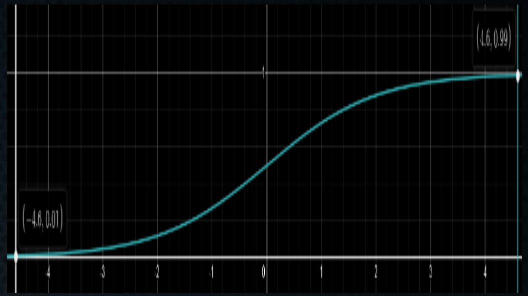


$$h_\theta(x) = sigmoid(? + ? \cdot x_1 + ? \cdot x_2)$$

Making simple functions ourselves: OR

- Answer

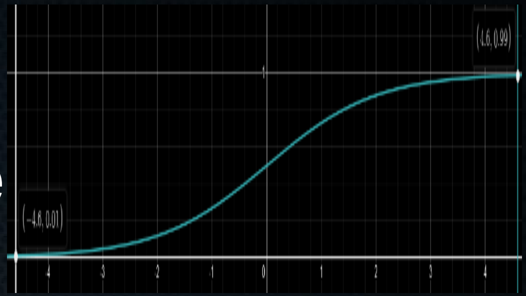


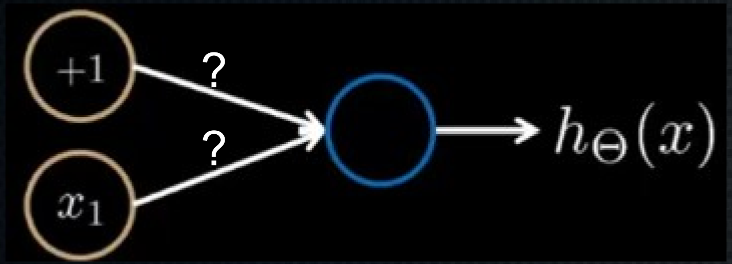


x_1	x_2	$h_\Theta(x)$
0	0	g(-10) ~= 0
0	1	g(-10+20) ~= 1
1	0	g(-10+20) ~= 1
1	1	g(-10 + 20 + 20) ~= 1

Making simple functions ourselves: NOT

- Over to you: make a NOT function → try it yourself and discuss with neighbours for 2 minute





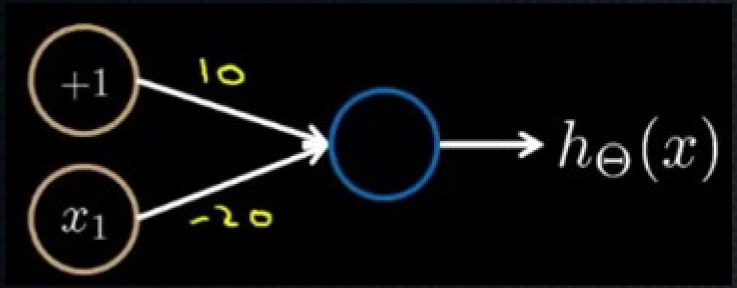
NOT

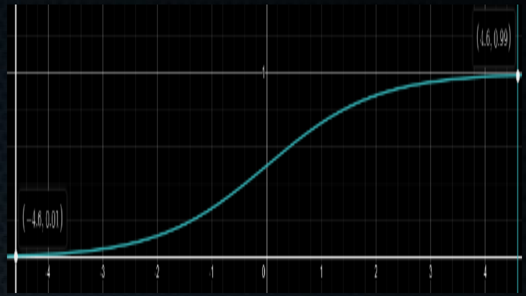
x	x'
0	1
1	0

$$h_\theta(x) = sigmoid(? + ? \cdot x_1)$$

Making simple functions ourselves: NOT

- Answer

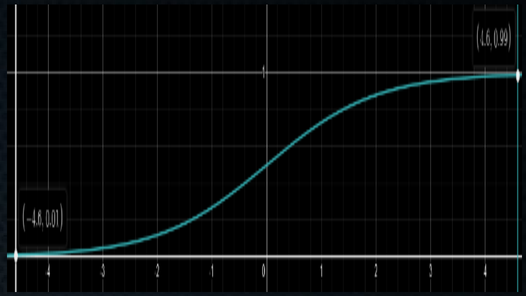


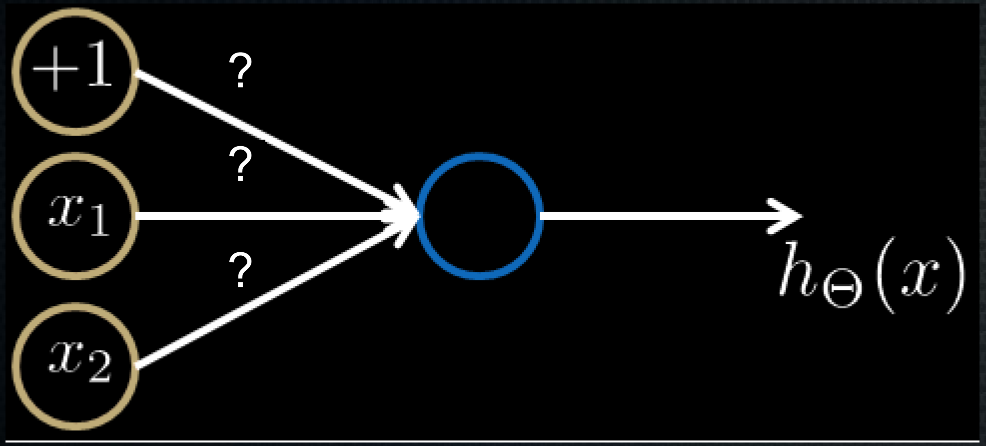


Just put a large negative weight in front of whatever you want to negate (enough to overcome the bias)

Making simple functions ourselves: (NOT x_1) AND (NOT x_2)

- Make NOT x_1 AND NOT x_2 → try it yourself and discuss with neighbours for 2 minutes

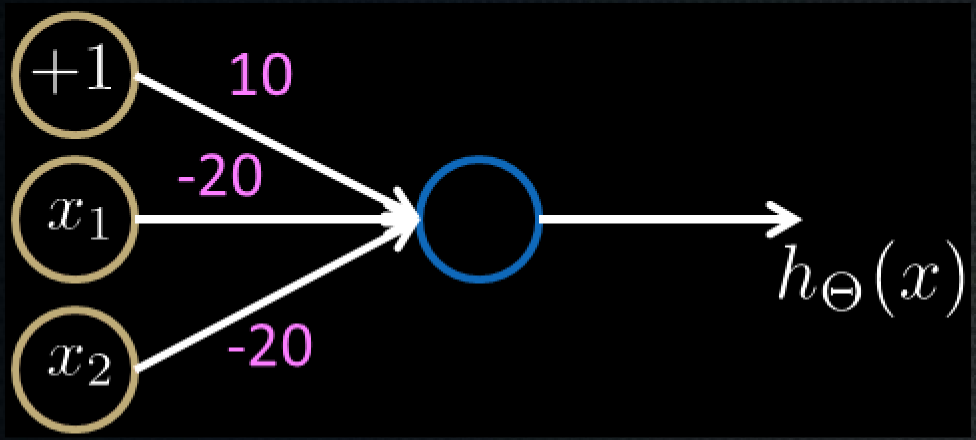


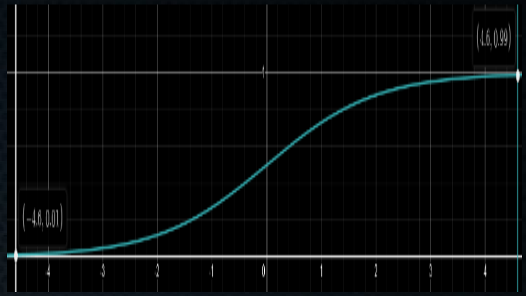


$$h_\theta(x) = sigmoid(? + ? \cdot x_1 + ? \cdot x_2)$$

Making simple functions ourselves: (NOT x_1) AND (NOT x_2)

- Answer

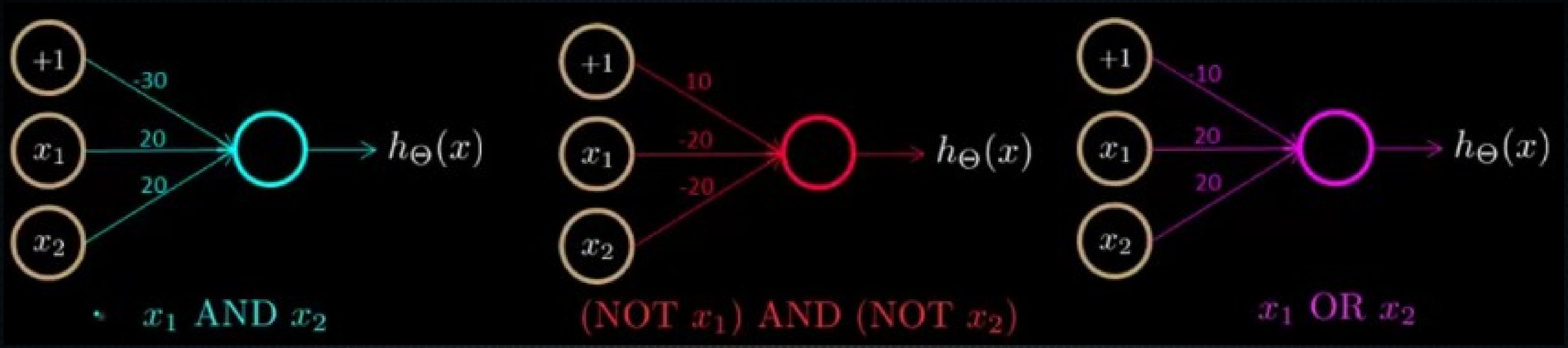


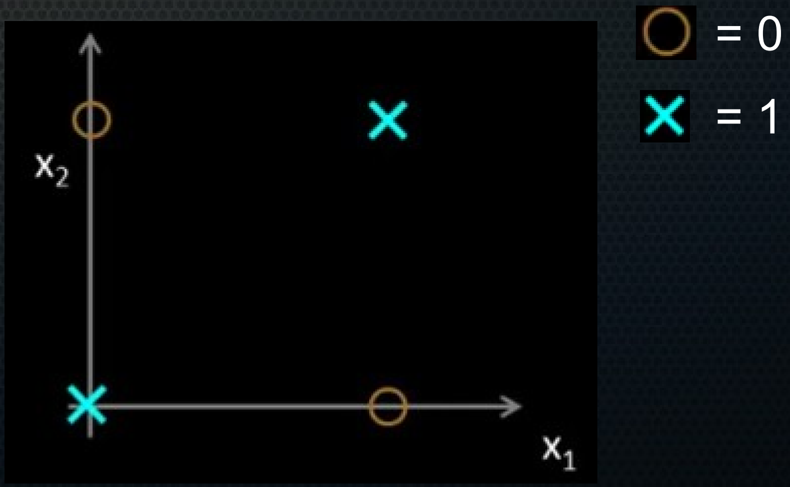


Just put a large negative weight in front of whatever you want to negate (enough to overcome the bias)

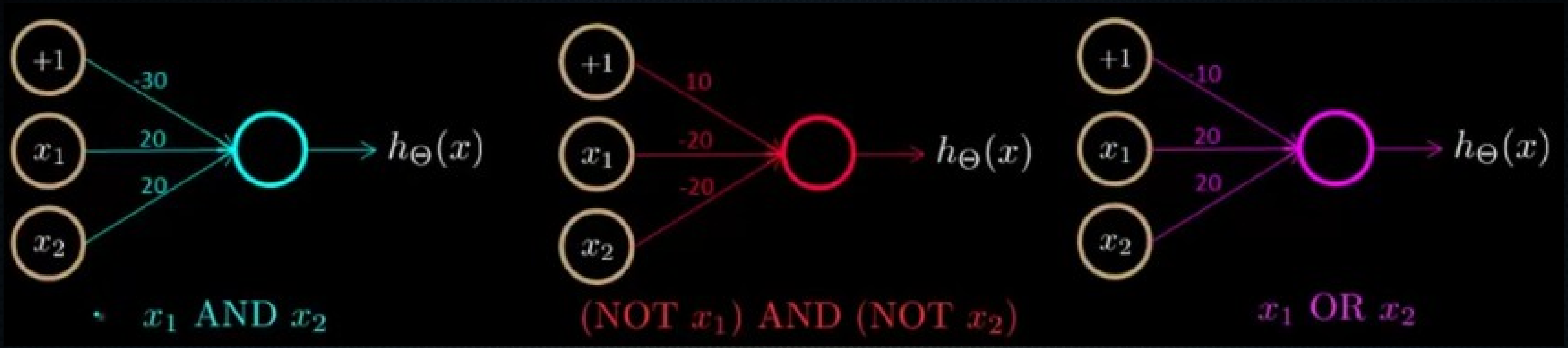
Only positive when $x_1 = x_2 = 0$

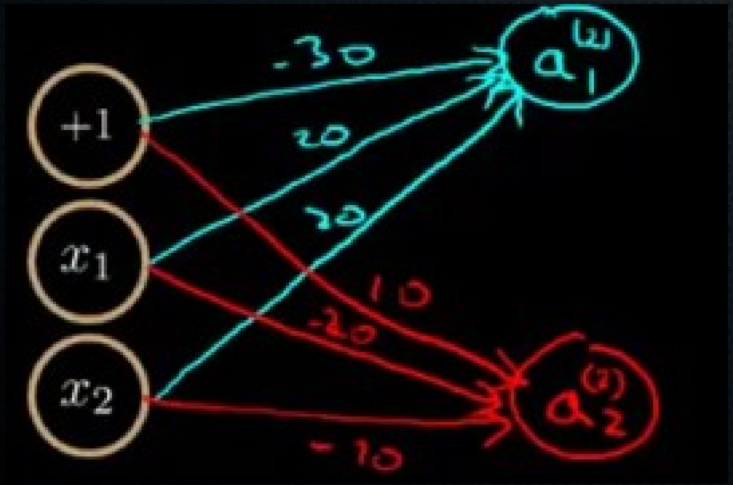
Finally: x_1 XNOR x_2



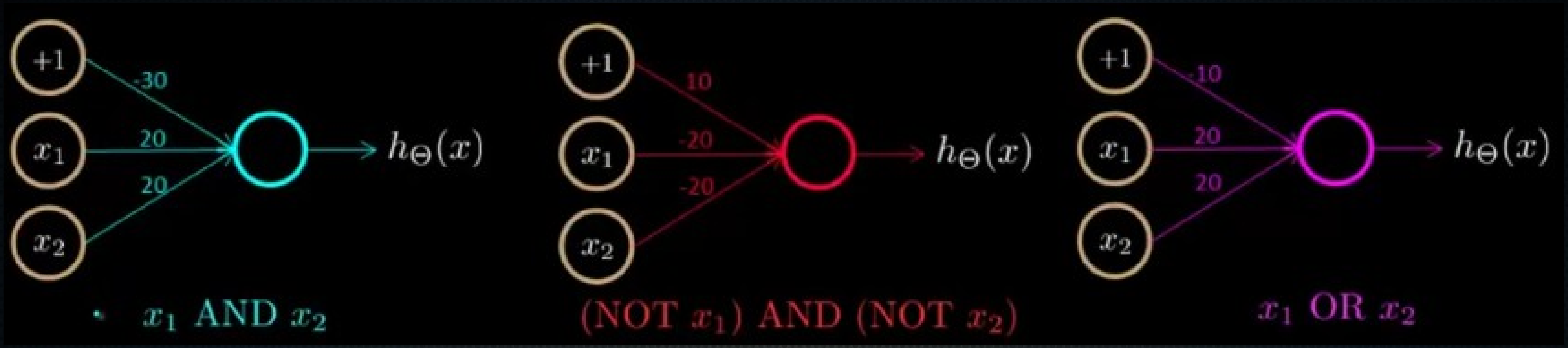


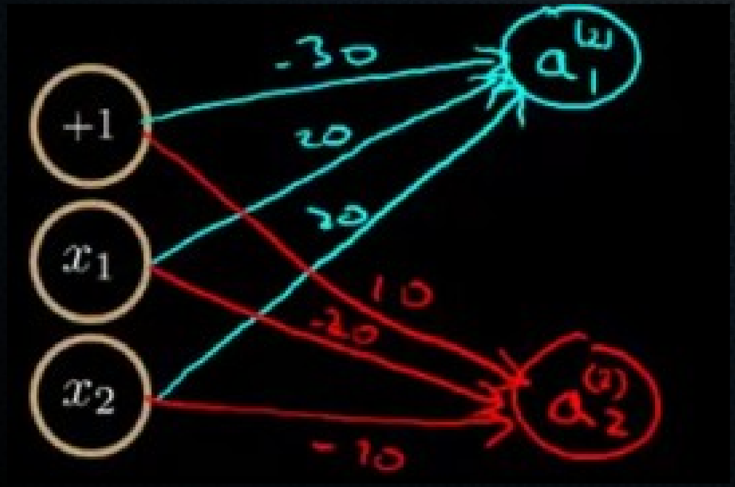
Finally: x_1 XNOR x_2





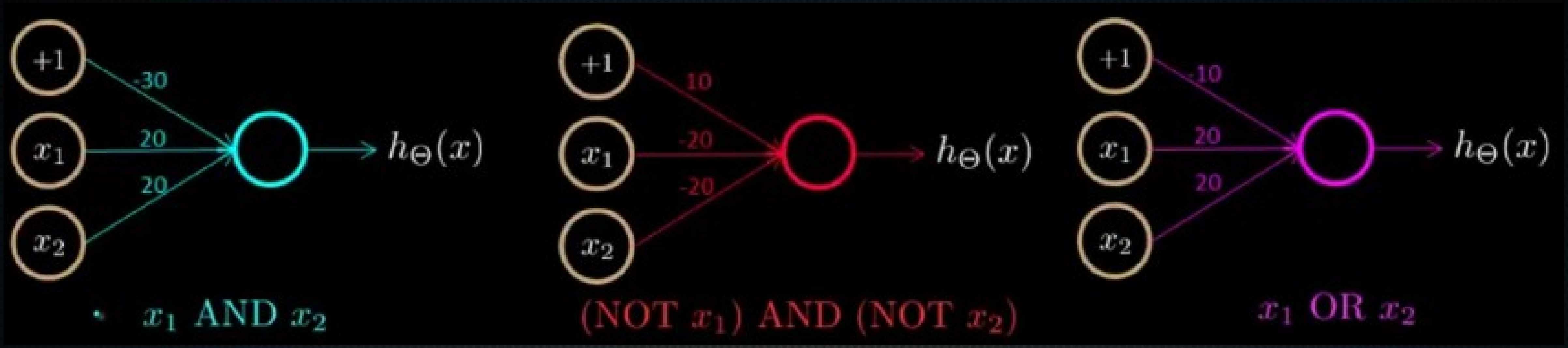
Finally: x_1 XNOR x_2





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

Finally: x_1 XNOR x_2

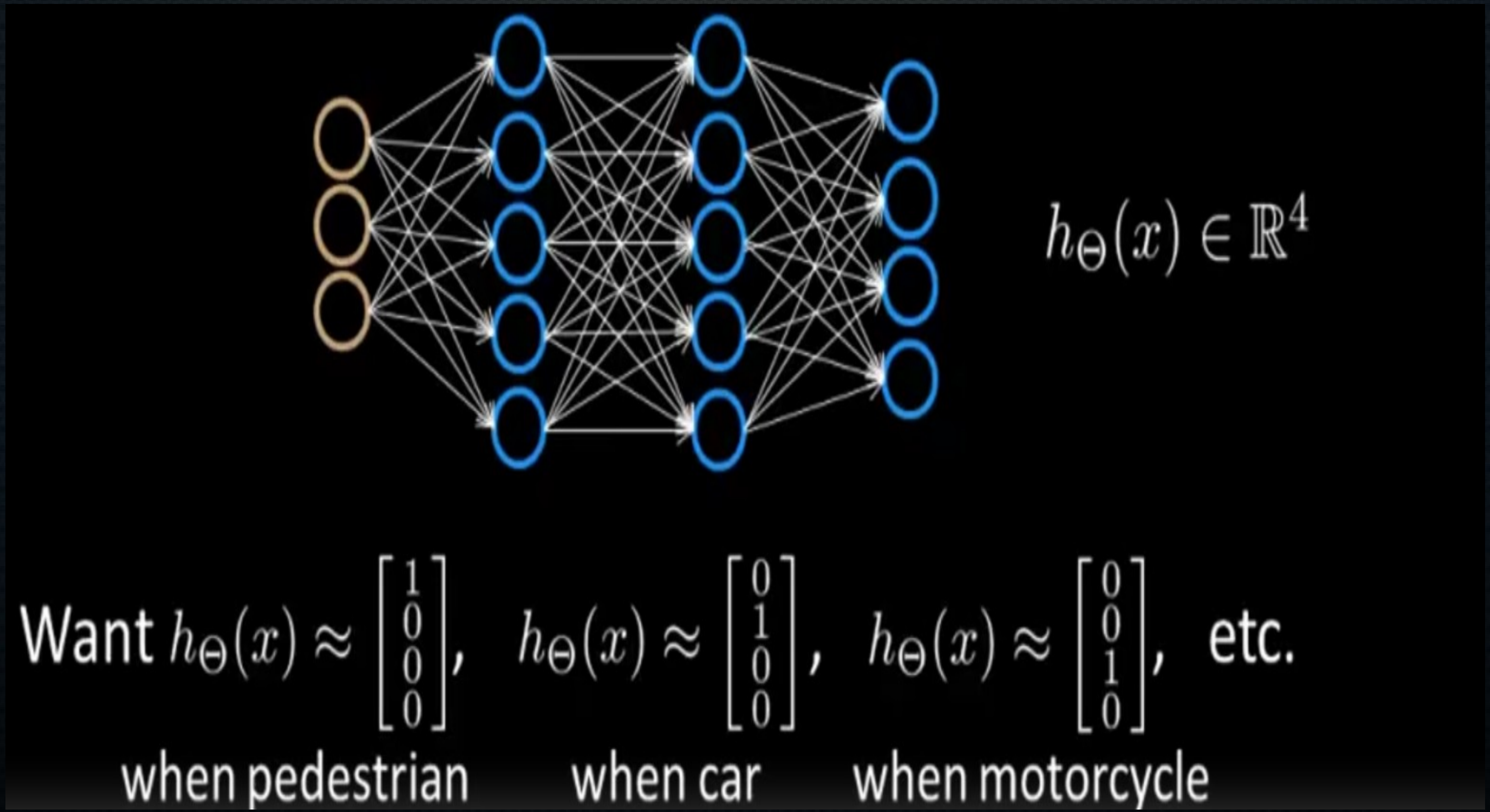


x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Computing complex functions

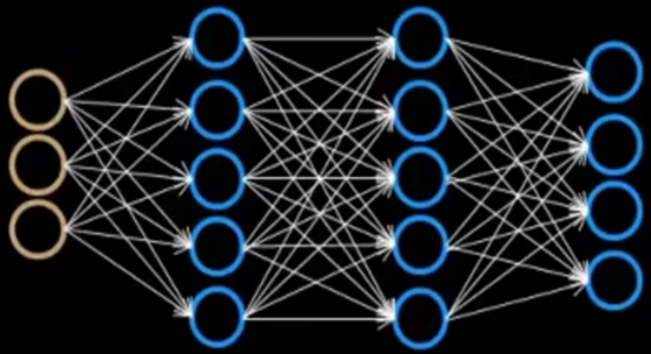
- This is an illustration of how neural networks work: earlier layers can compute simple functions like AND and OR. By combining those outputs, you can compute more complex functions.

Multiclass classification in neural nets



Multiclass classification in neural nets

Pedestrian Car Motorcycle Truck



$h_\Theta(x) \in \mathbb{R}^4$

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

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$y^{(i)}$ 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

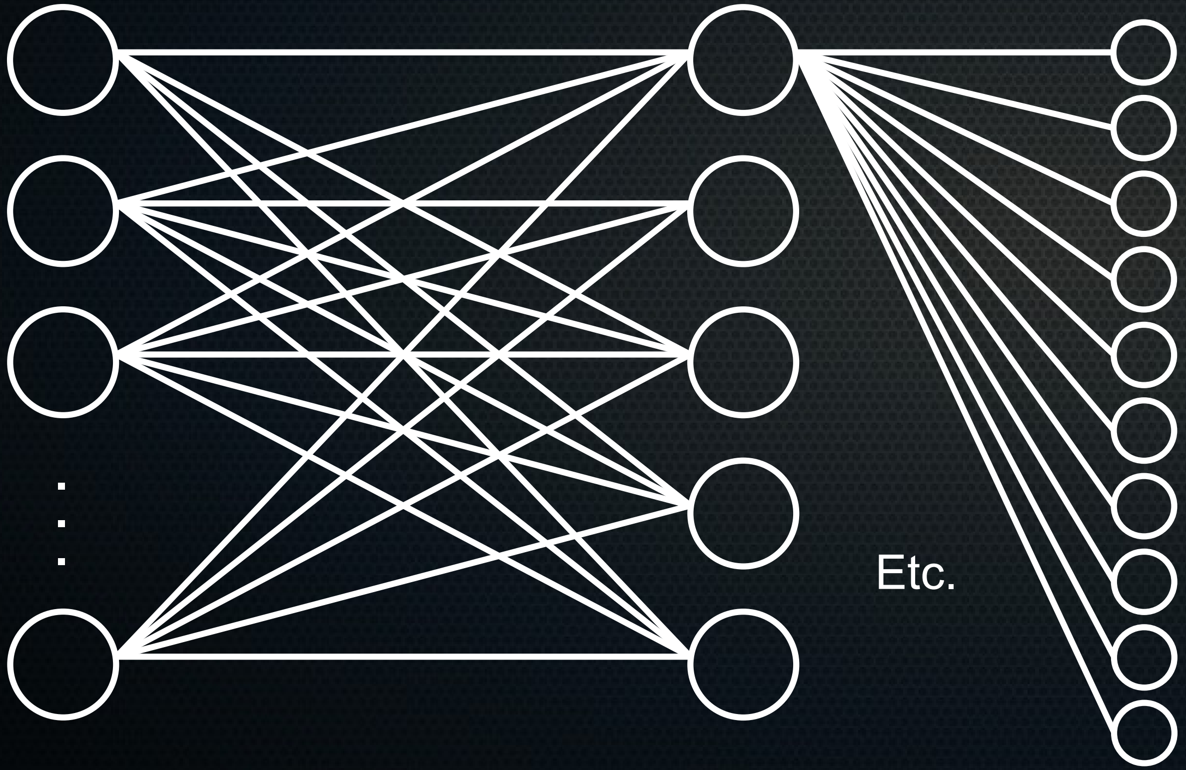
~~Previously~~

~~$y \in \{1,2,3,4\}$~~

Question to you

- Say we have 10 classes and the following network, how many parameters in $\Theta^{(2)}$?

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



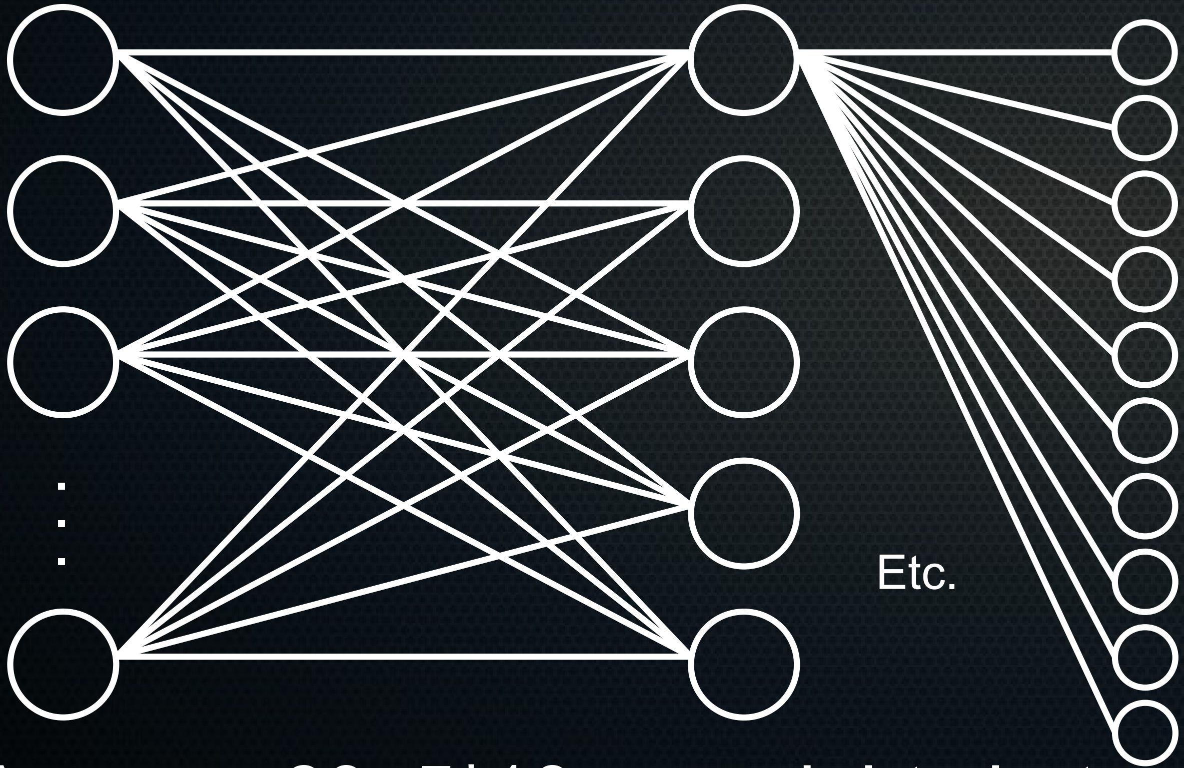
$$y^{(10)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Example:
10th training sample is class 3

Question to you

- Say we have 10 classes and the following network, how many parameters in $\Theta^{(2)}$?

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



$$y^{(10)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

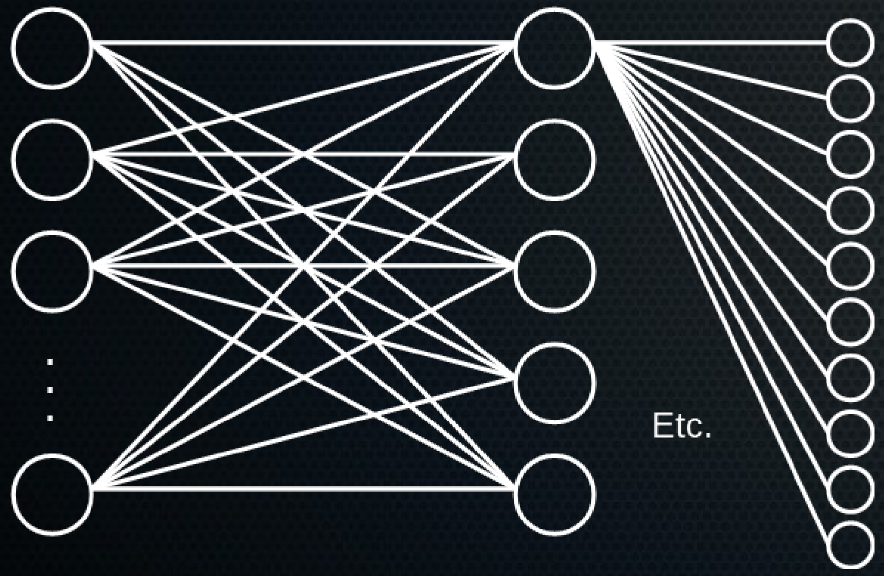
Example:
10th training sample is class 3

- Answer: 60. 5*10 → weights between units.
  + 10 → bias of each unit in output layer

Question to you

- Say we have 10 classes and the following network, how many parameters in $\Theta^{(2)}$?

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



$$\sigma\left(\overbrace{\begin{bmatrix} b_1 & w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ b_2 & w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ b_3 & w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ b_4 & w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \\ b_5 & w_{51} & w_{52} & w_{53} & w_{54} & w_{55} \end{bmatrix}}^{\Theta^{(2)}} \begin{bmatrix} 1 \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \\ a_5^{(2)} \end{bmatrix}\right)$$

- Answer: 60. 5*10 → weights between units.
  + 10 → bias of each unit in output layer

Summary

- We can use individual neurons to calculate simple logic functions
- We can combine the outputs of single neurons to calculate more complex (logic) functions
- For multiclass classification, we simply make class a vector, where we strive for the real class to be ~1, and all other classes 0.

Time for the afternoon practical!