Artificial Neural Networks

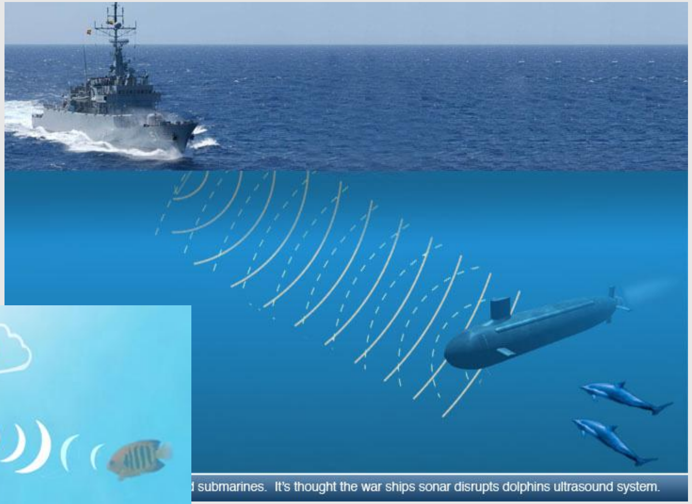
Machine Learning and Pattern Recognition

Prof. Sandra Avila

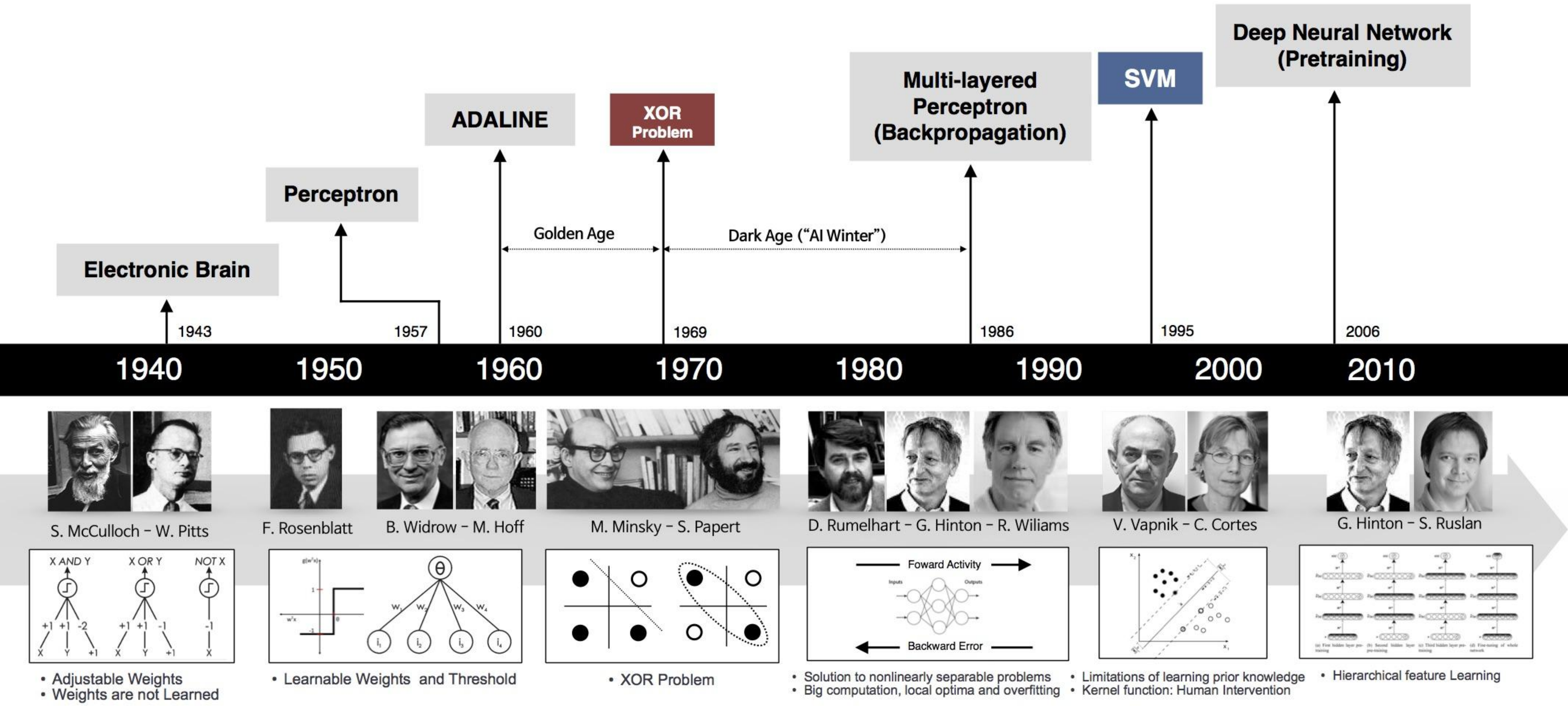
Institute of Computing (IC/Unicamp)

MC886/MO444, September 11, 2018

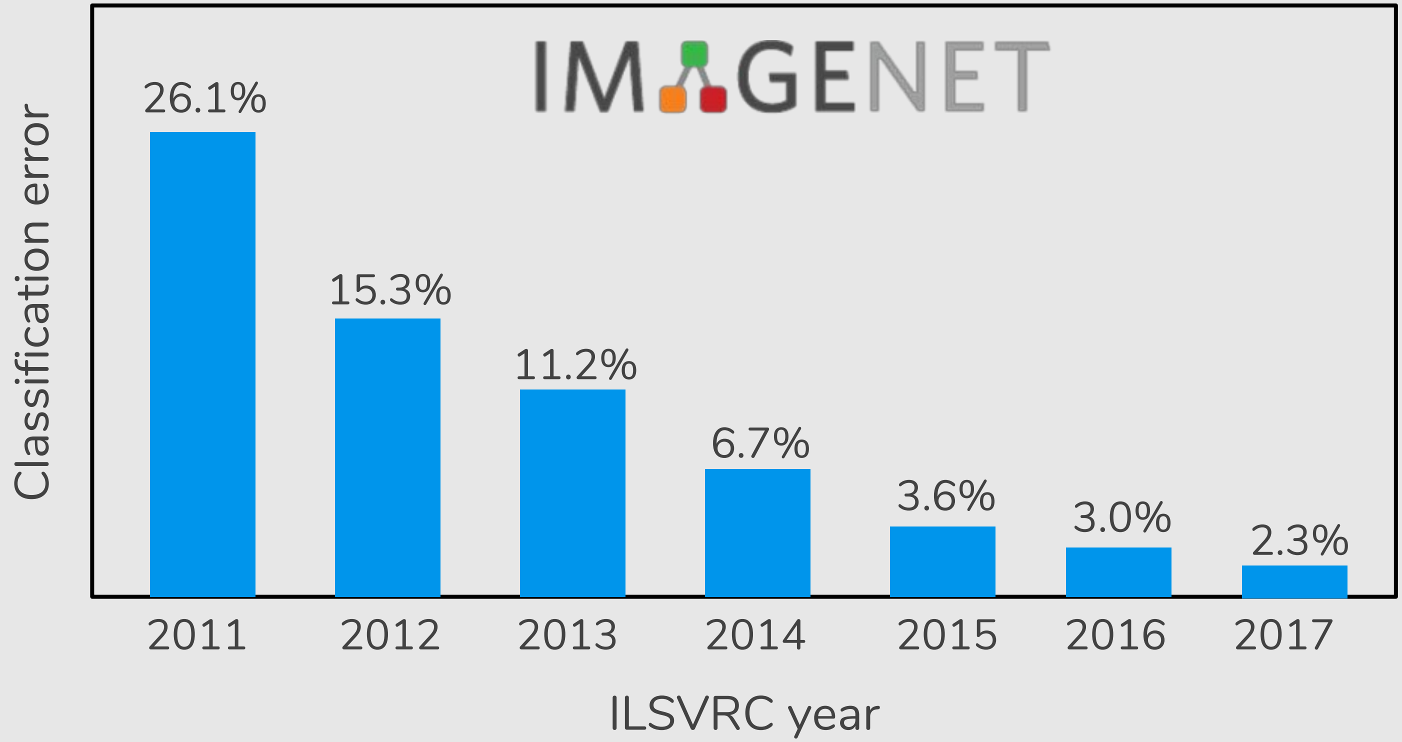
Many inventions were inspired by Nature ...



It seems logical to look at the brain's architecture for inspiration on how to build an intelligent machine.



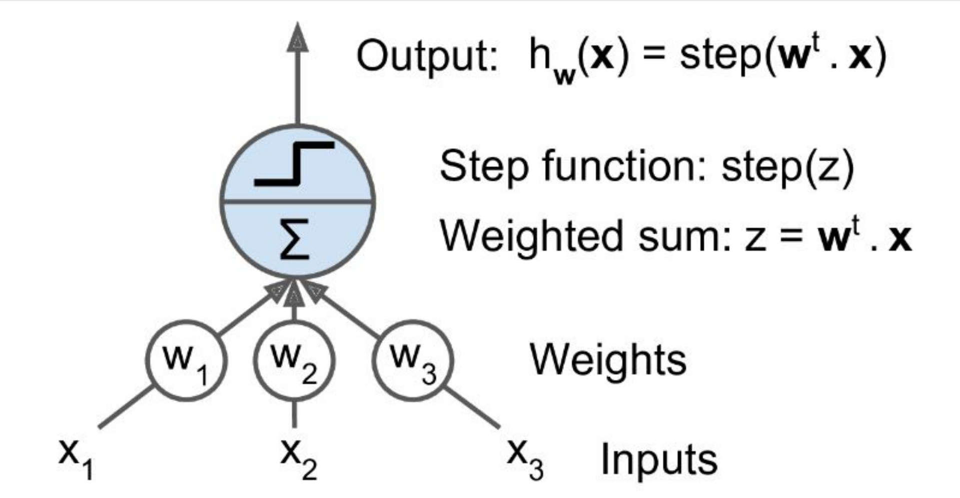
Electronic Brain
Perceptron
ADALINE
XOR Problem
Multi-layered Perceptron (Backpropagation)
SVM
Deep Neural Network (Pretraining)
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

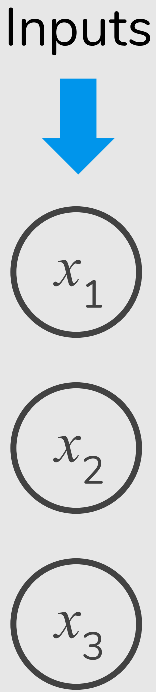


“ImageNet classification with deep convolutional neural networks”. Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton. In: NIPS, 2012.

The Perceptron

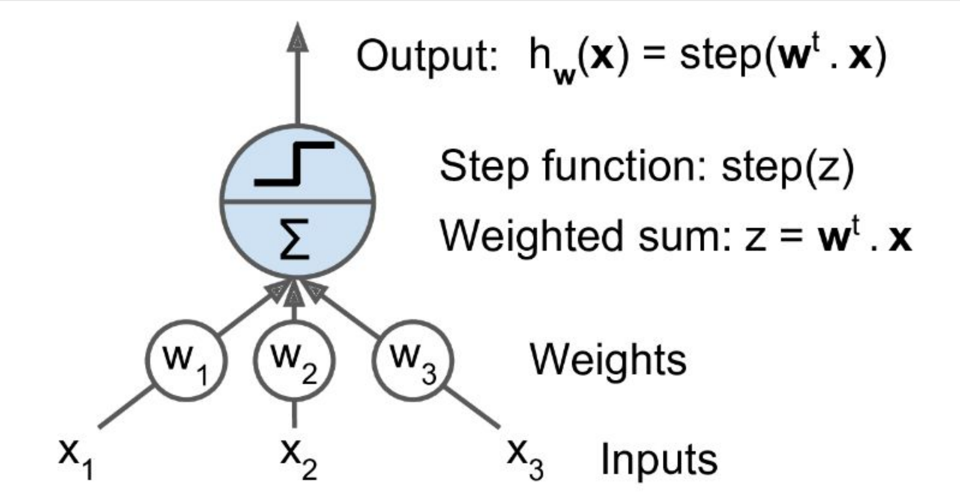
Neuron Model: Logistic Unit

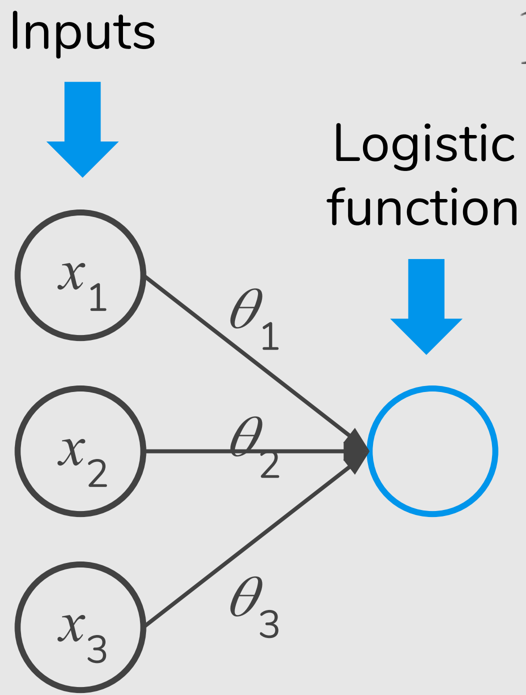




Neuron Model: Logistic Unit

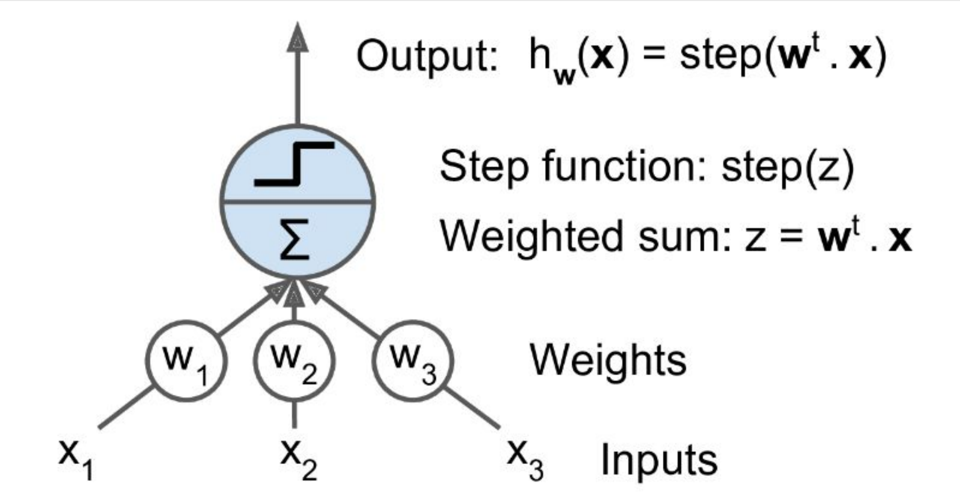
$$\frac{1}{1+e^{-\theta^{T}x}}$$

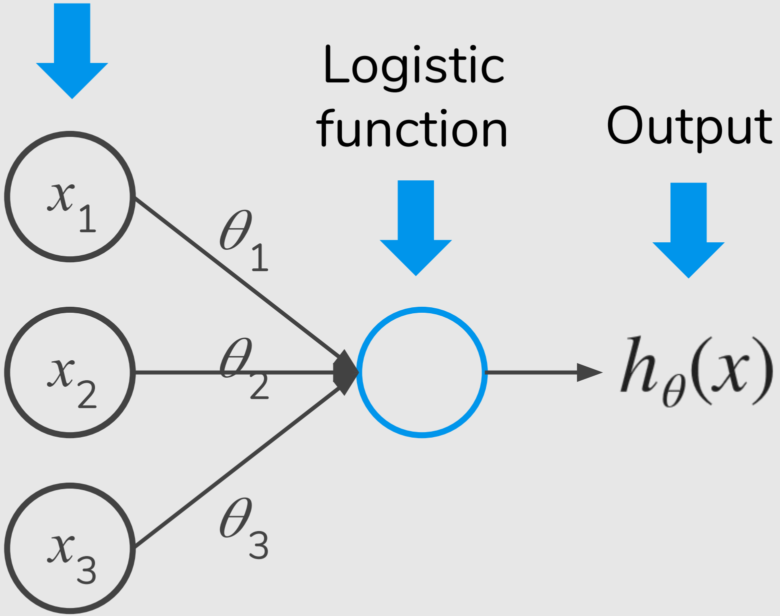




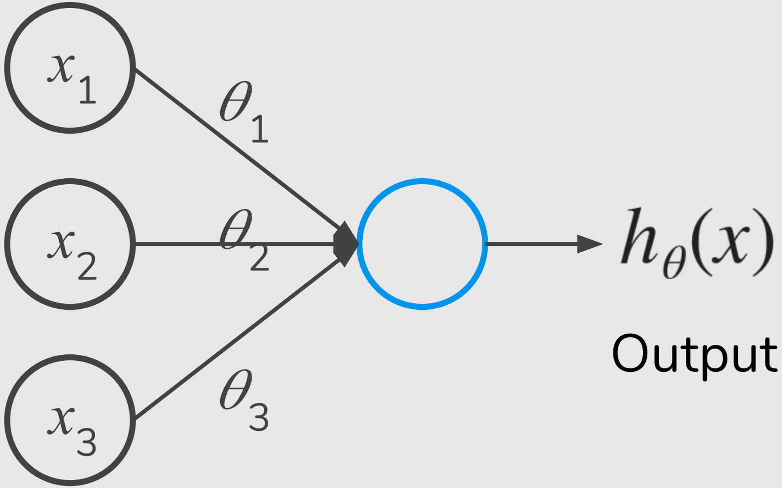
Neuron Model: Logistic Unit

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





Neuron Model: Logistic Unit



Inputs

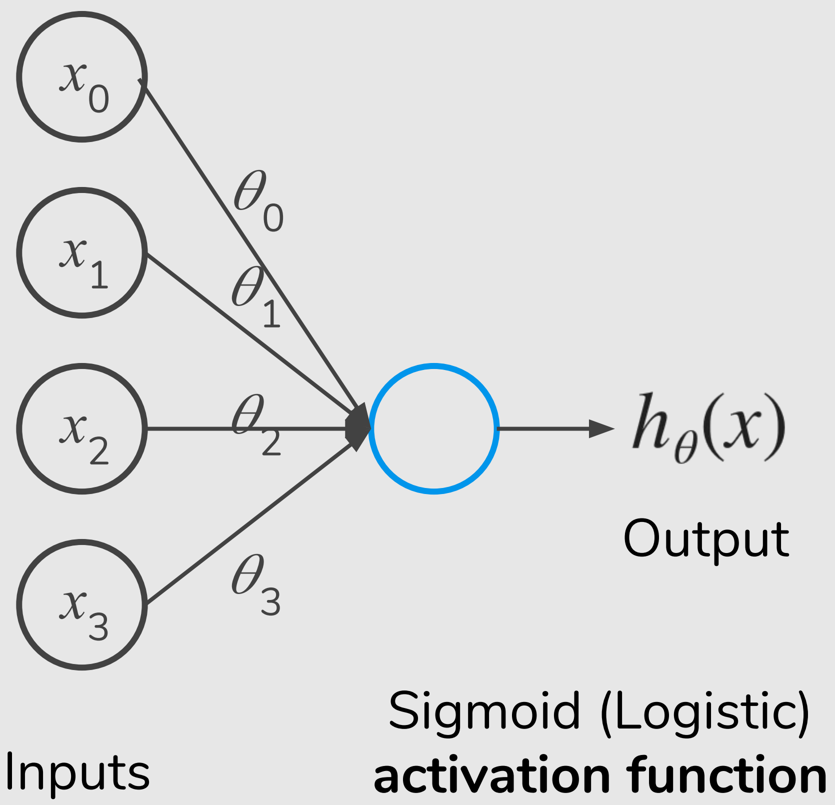
Sigmoid (Logistic)
activation function

weights

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

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

Neuron Model: Logistic Unit



weights

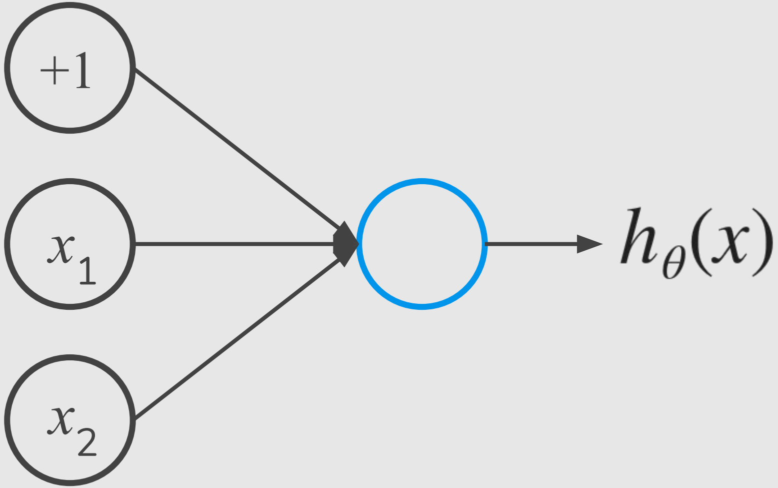
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

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

Examples

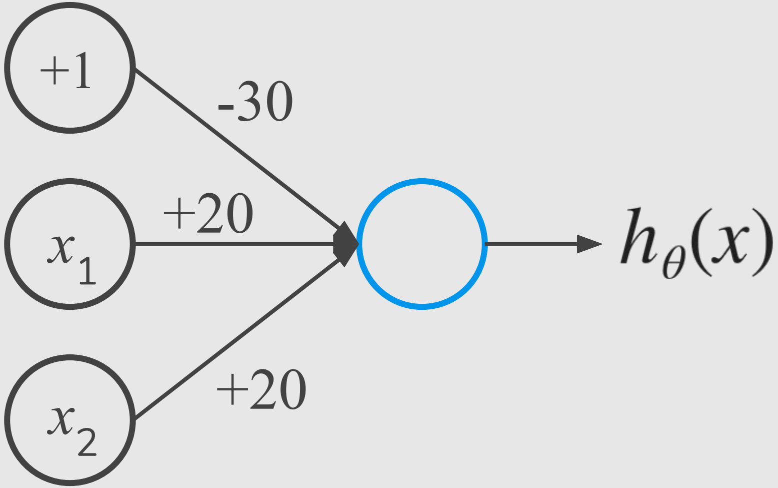
Simple Example: AND

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



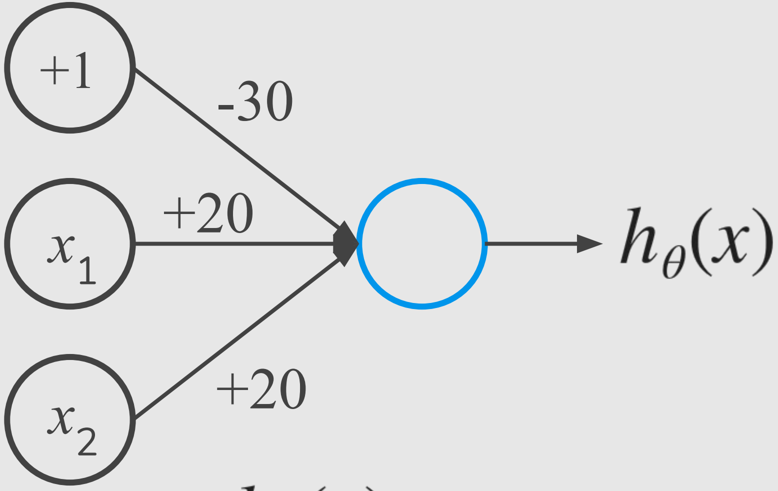
Simple Example: AND

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



Simple Example: AND

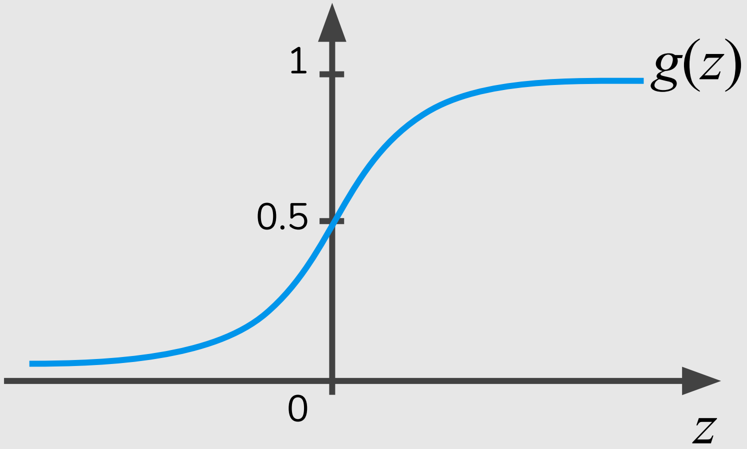
$x_1, x_2 \in \{0,1\}$ $y = x_1 \text{ AND } x_2$

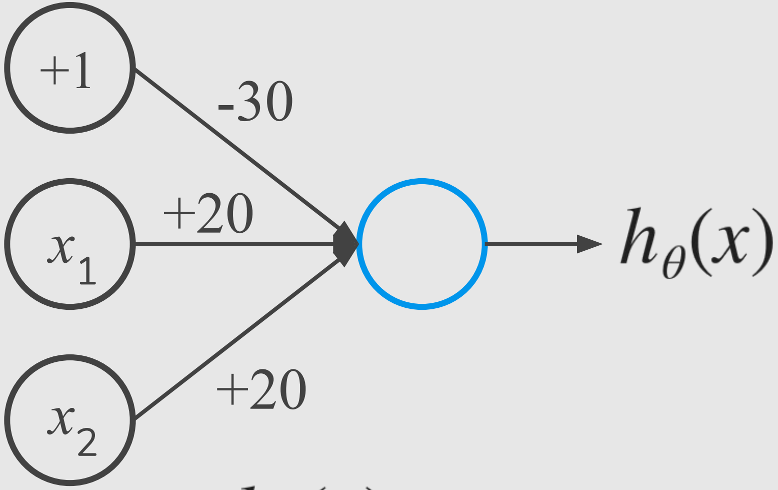


$$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$$

Simple Example: AND

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

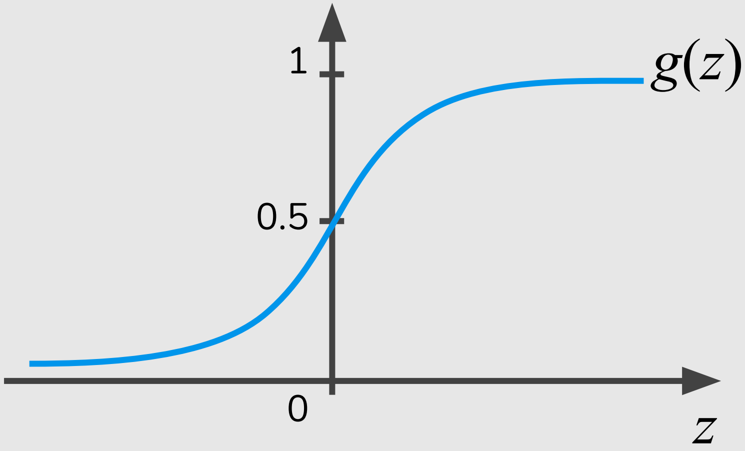




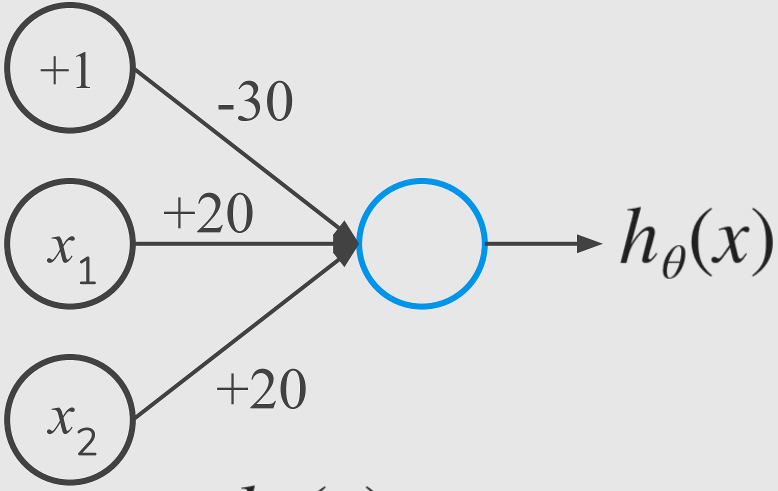
x_1	x_2	$h_\theta(x)$
0	0	
0	1	
1	0	
1	1	

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

Simple Example: AND



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

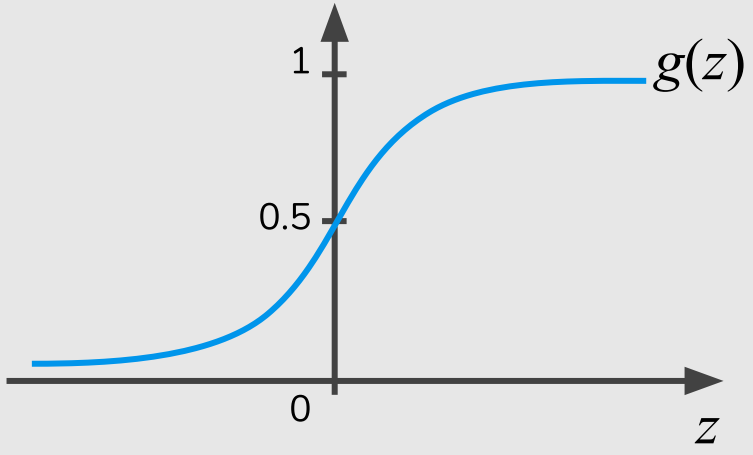


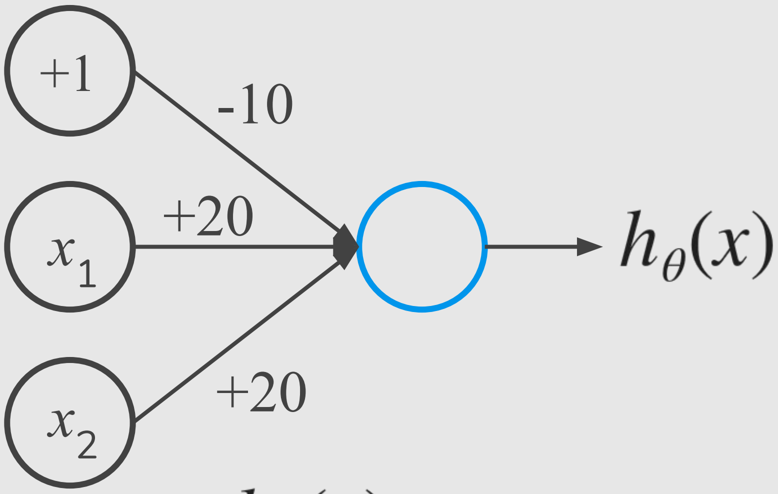
$$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$$

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	$g(10) \approx 1$

Simple Example: OR

$x_1, x_2 \in \{0,1\}$ $\quad y = x_1 \text{ OR } x_2$





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

$$h_\theta(x) = g(-10 + 20x_1 + 20x_2)$$

What does an artificial neuron do?

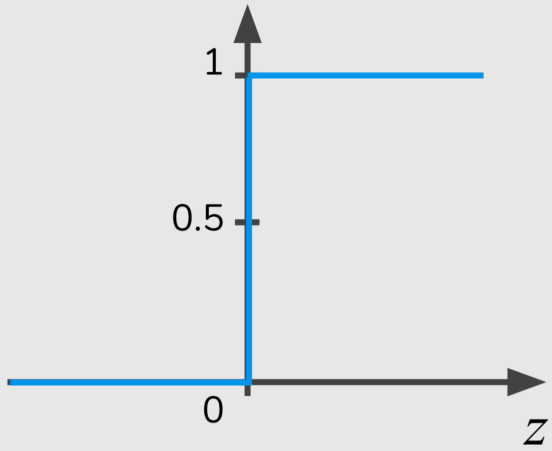
It calculates a “weighted sum” of its input, adds a bias and then decides whether it should be “fired” or not.

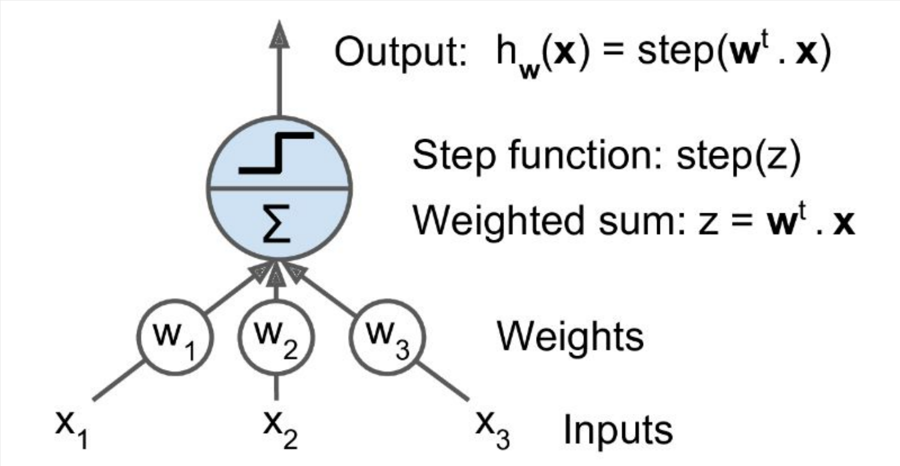
How do we decide whether the neuron should fire or not?

We decided to add “activation functions” for this purpose.

Step Function

Its output is **1 (activated)** when value > 0 (threshold) and outputs a **0 (not activated)** otherwise.





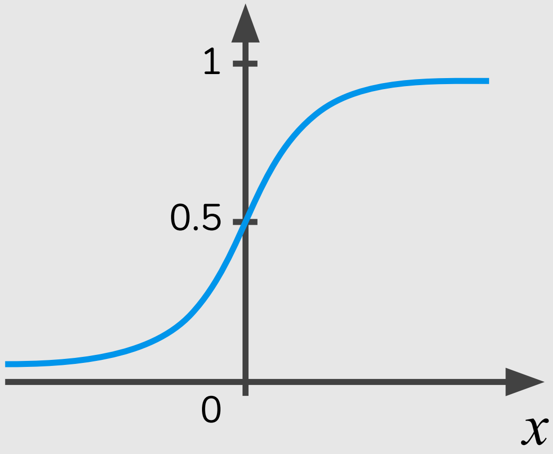
Step Function: **Problem?**

- Binary classifier ("yes" or "no", activate or not activate). A Step function could do that for you!

- Multi classifier (class1, class2, class3, etc). What will happen if more than 1 neuron is "activated"?

Sigmoid Function

- The output of the activation function is always going to be in range **(0,1)**.
- It is nonlinear in nature.
- Combinations of this function are also nonlinear! Great!!



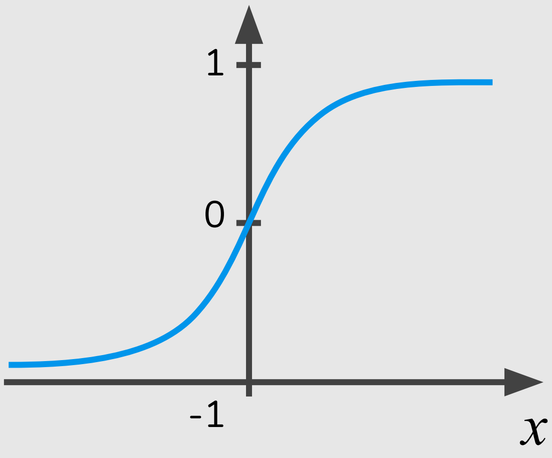
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Sigmoid Function: **Problem?**

- Towards either end of the sigmoid function, the $\sigma(x)$ values tend to respond very less to changes in x.
- The problem of "**vanishing gradients**".
    - Cannot make significant change because of the extremely small value.

Tanh Function

- The output of the activation function is always going to be in range **(-1,1)**.
- It is nonlinear in nature.
- Combinations of this function are also nonlinear! Great!!



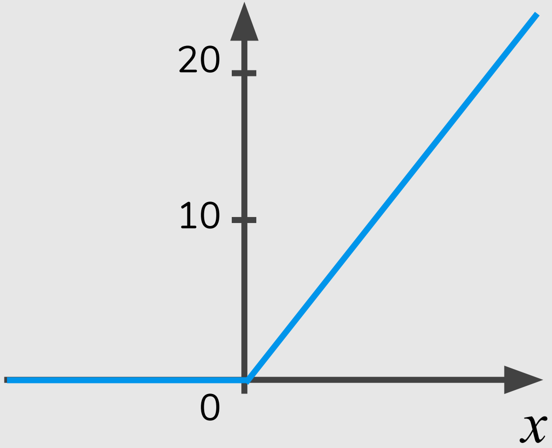
$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Tanh Function: **Problem?**

- Like sigmoid, tanh also has the vanishing gradient problem.

ReLU (Rectified Linear Unit) Function

- It gives an output x if x is positive and 0 otherwise. The range is **[0, inf)**.
- It is nonlinear in nature. Combinations of this function are also nonlinear!
- Sparsity of the activation!



$$\mathrm{ReLU}(x) = max(0,x)$$

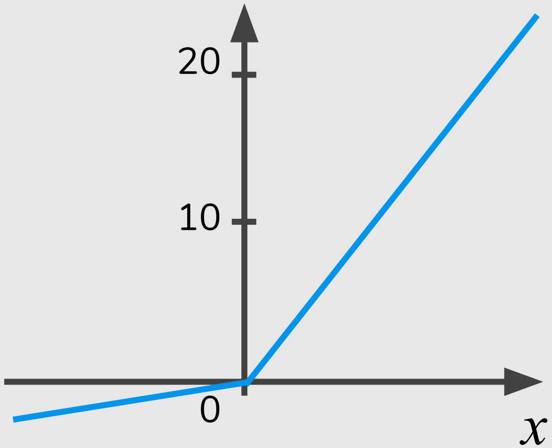
ReLU Function: **Problem?**

- Because of the horizontal line in ReLU(for negative x), the gradient can go towards 0.

- "Dying ReLU problem": several neurons can just die and not respond making a substantial part of the network passive.

Leaky ReLU Function

- It gives an output x if x is positive and 0 otherwise. The range is **[0, inf)**.
- (Leaky) ReLU is less computationally expensive than *tanh* and *sigmoid* because it involves simpler mathematical operations.



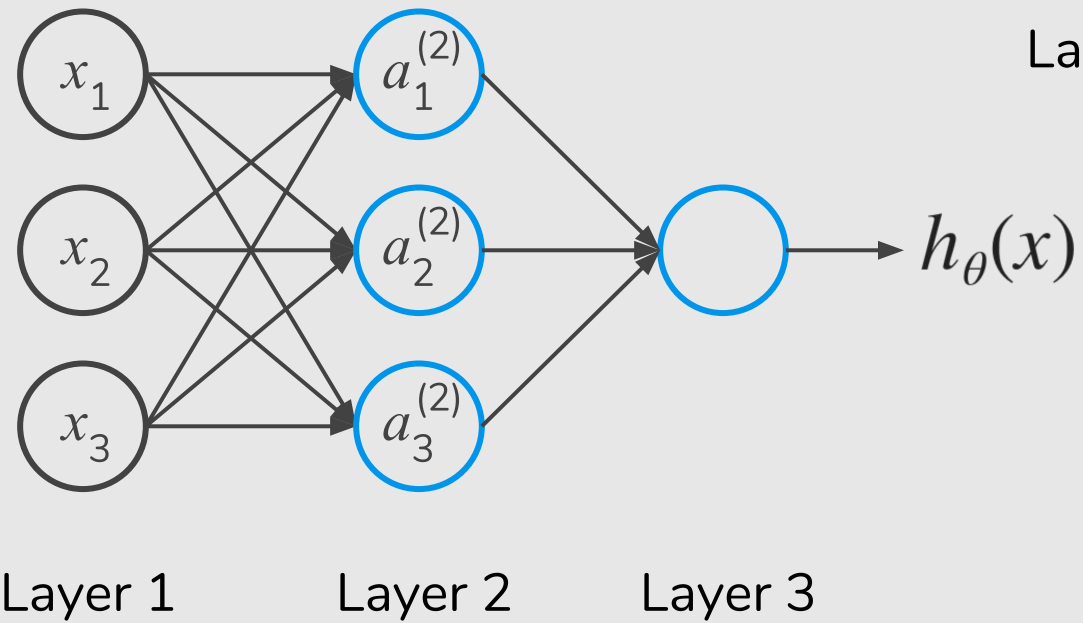
$$\text{Leaky ReLU}(x) = \begin{cases} x \text{ if } x > 0 \\ 0.01x \text{ otherwise} \end{cases}$$

Ok! Which One Do We Use?

- If you don’t know the nature of the function you are trying to learn, start with ReLU.
- You can use your own custom functions too!

Neural Network

Neural Network



Layer 1 = Input layer

Layer 2 = Hidden layer

Layer 3 = Output layer

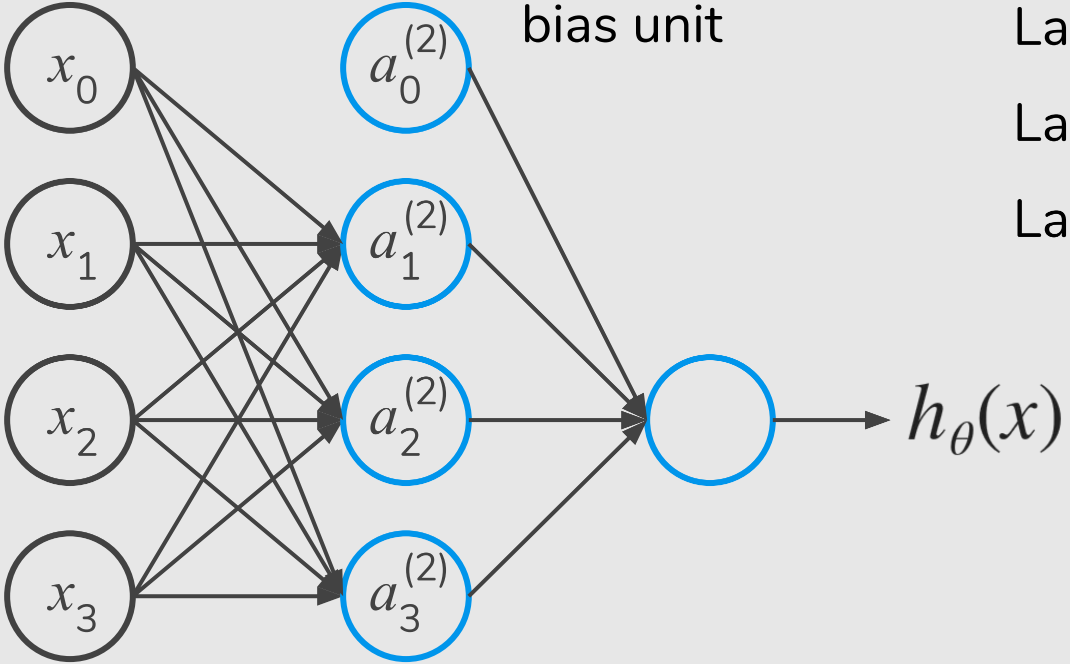
Neural Network

bias unit

Layer 1 = Input layer

Layer 2 = Hidden layer

Layer 3 = Output layer

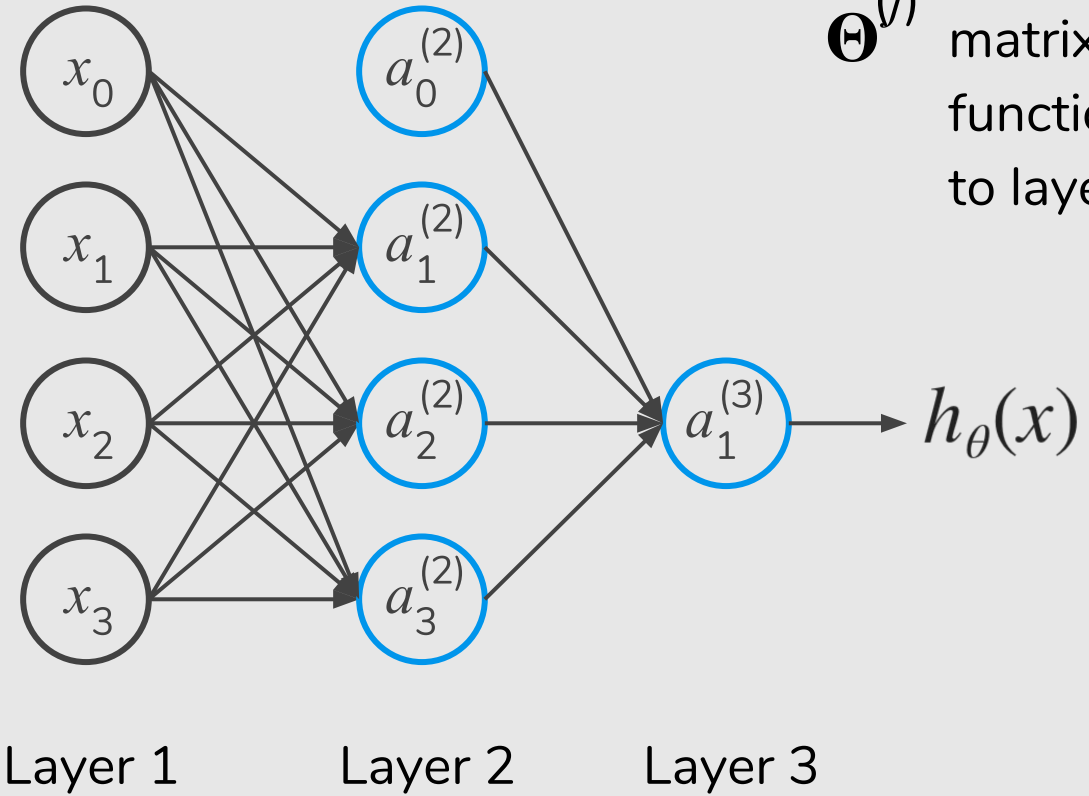


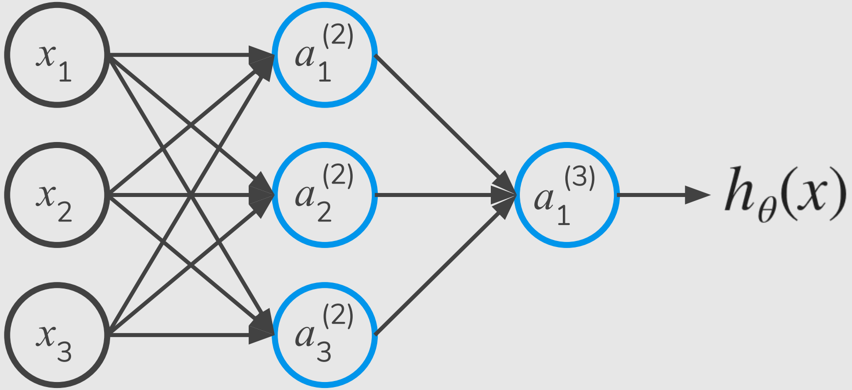
Layer 1 Layer 2 Layer 3

Neural Network

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

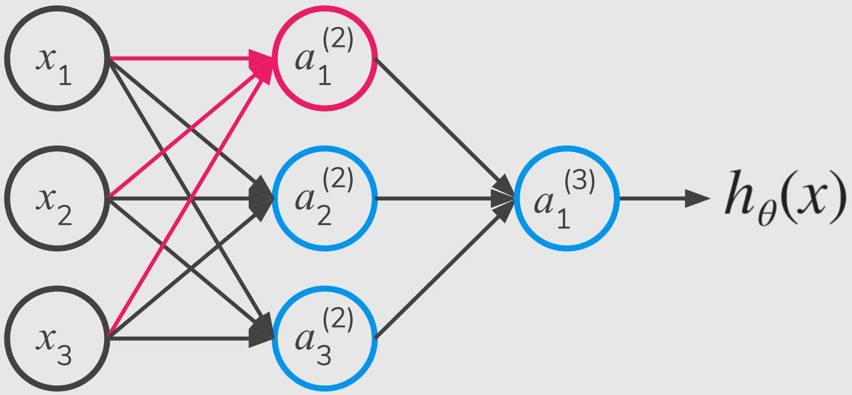
$\Theta^{(j)}$ matrix of weights controlling function mapping from layer j to layer $j+1$





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

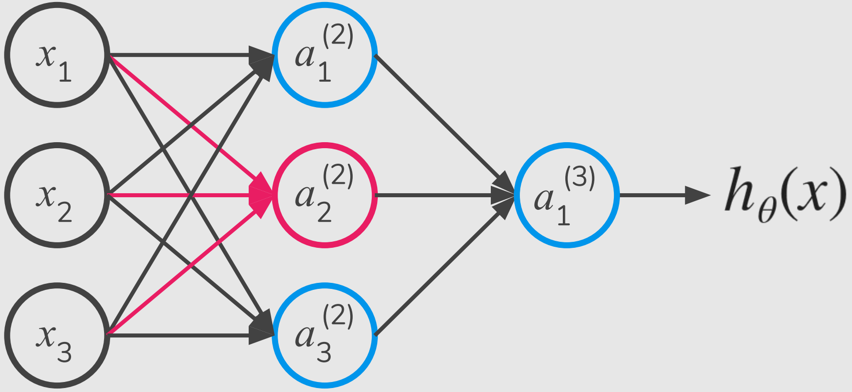
$\mathbf{\Theta}^{(j)}$ matrix of weights controlling function mapping from layer j to layer $j+1$



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

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

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

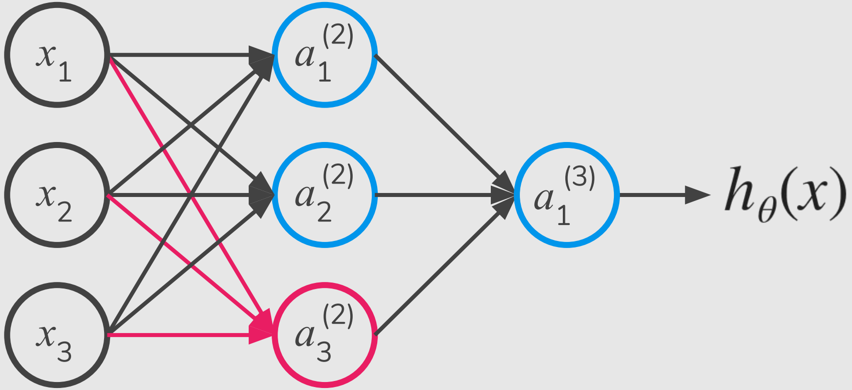


$a_i^{(j)}$ "activation" of unit i in layer j

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

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

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



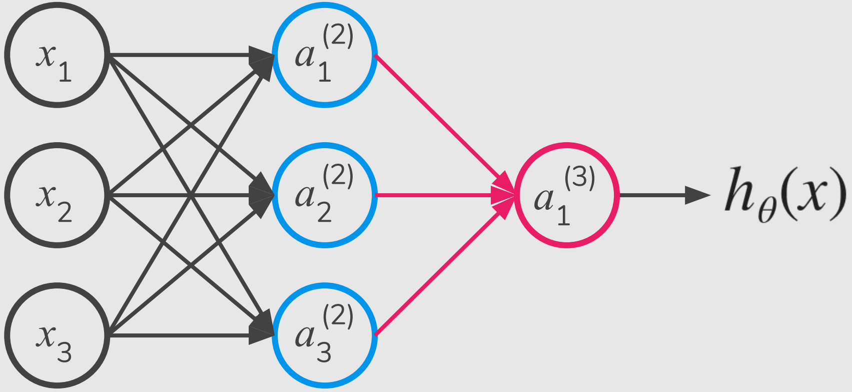
$a_i^{(j)}$ "activation" of unit i in layer j

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

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

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

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



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

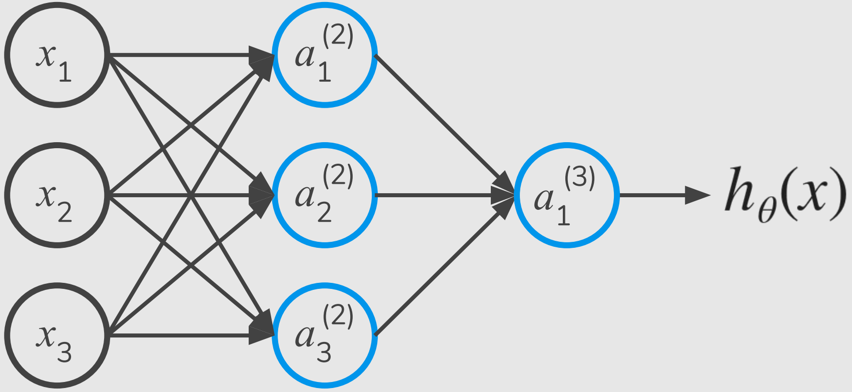
$\mathbf{\Theta}^{(j)}$ matrix of weights controlling function mapping from layer j to layer $j+1$

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

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

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

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



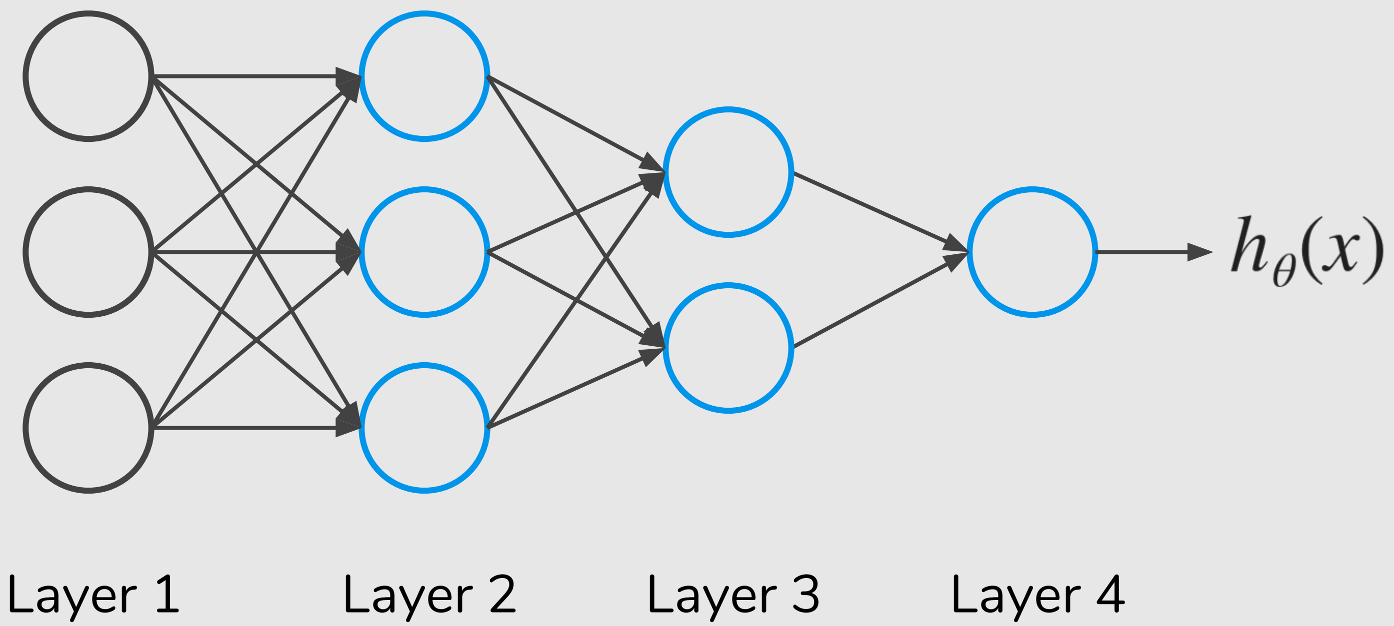
$a_i^{(j)}$ “activation” of unit i in layer j

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

Feedforward Neural Network (forward propagating)

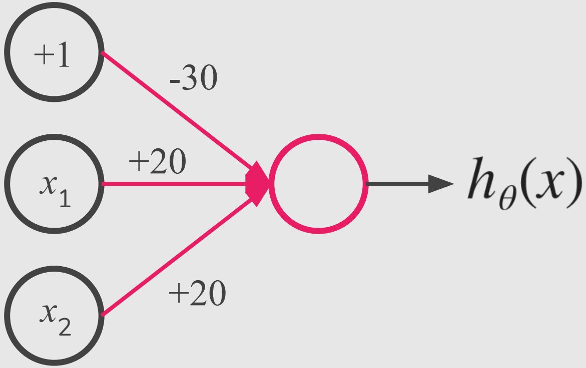
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

Other Network Architectures

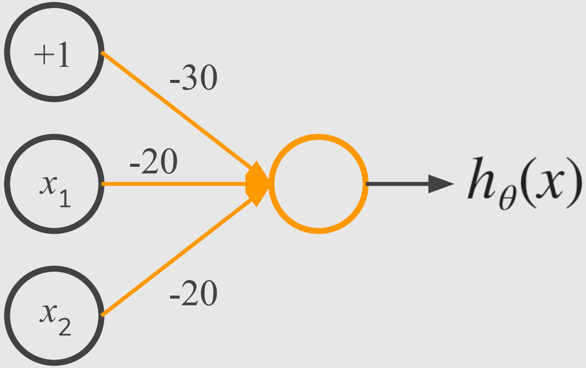


Example: XNOR $x_1, x_2 \in \{0,1\}$ $y = x_1 \text{ XNOR } x_2$

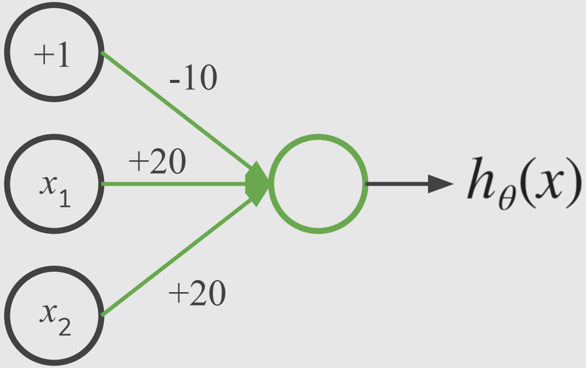
Example: XNOR $x_1, x_2 \in \{0,1\}$ $y = x_1$ XNOR x_2



x_1 AND x_2

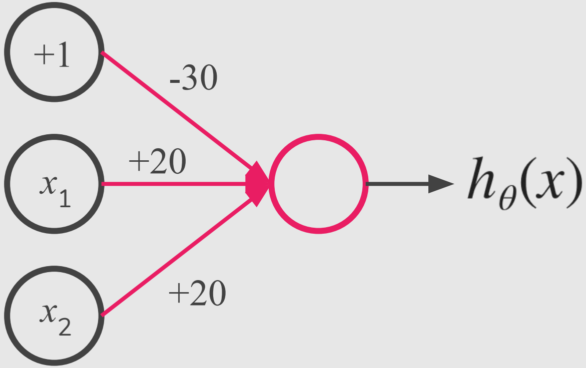


(NOT x_1) AND (NOT x_2)

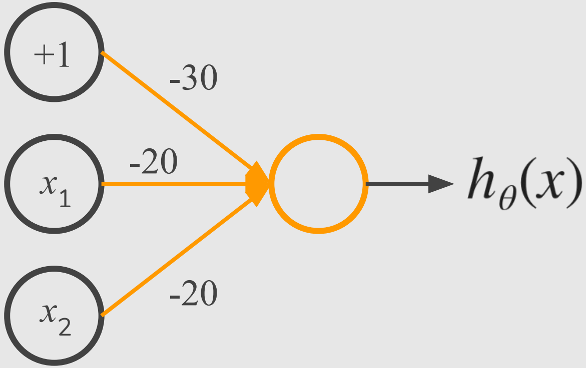


x_1 OR x_2

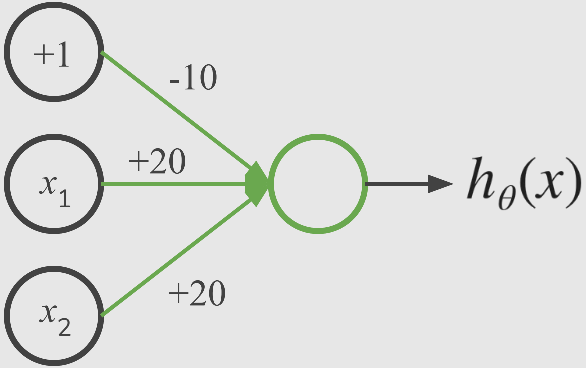
Example: XNOR $x_1, x_2 \in \{0,1\}$ $y = x_1$ XNOR x_2



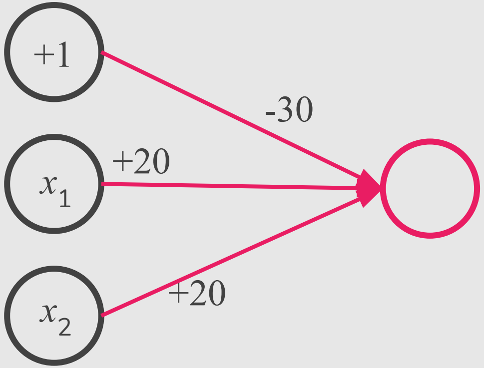
x_1 AND x_2



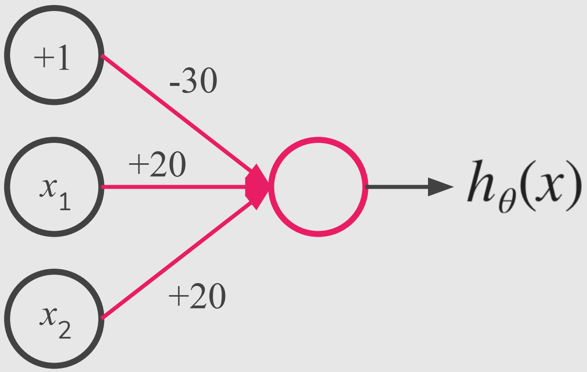
(NOT x_1) AND (NOT x_2)



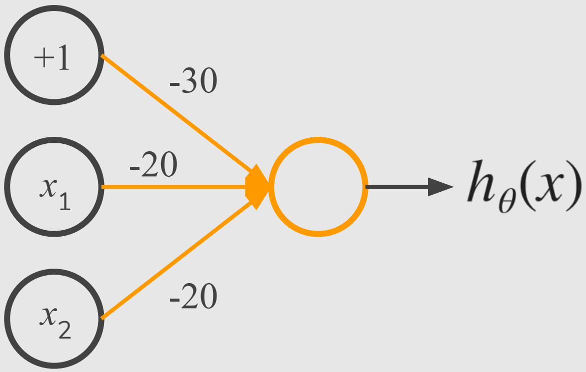
x_1 OR x_2



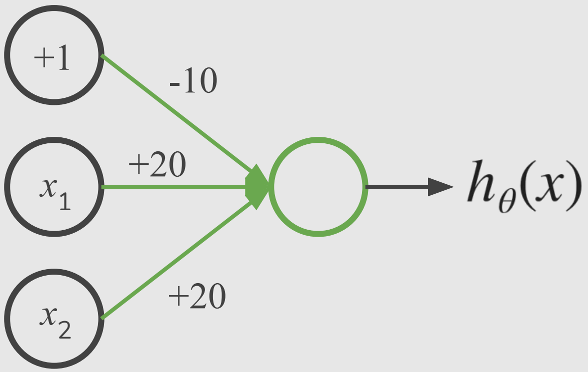
Example: XNOR $x_1, x_2 \in \{0,1\}$ $y = x_1 \text{ XNOR } x_2$



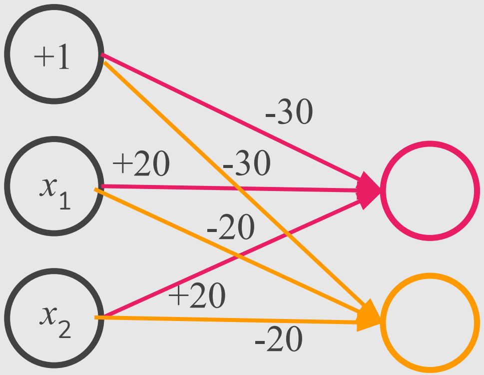
x_1 AND x_2



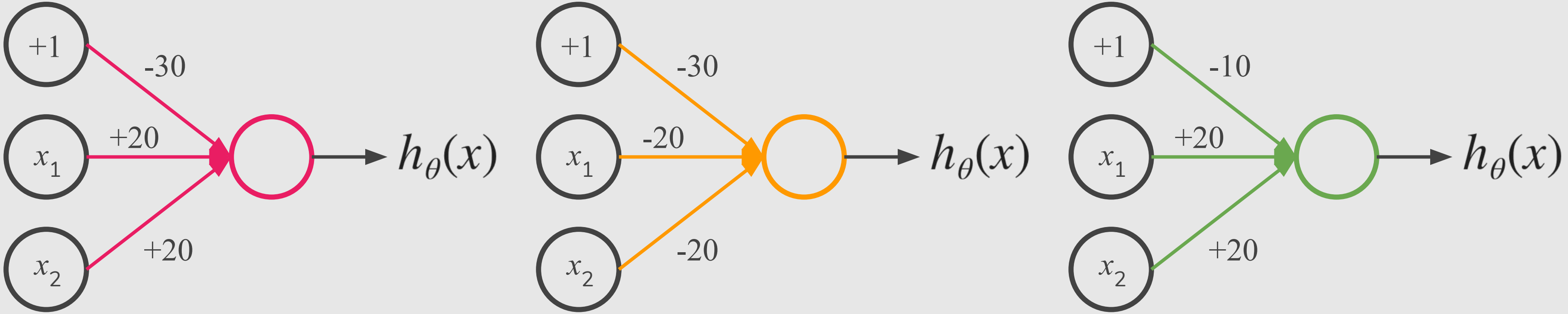
(NOT x_1) AND (NOT x_2)



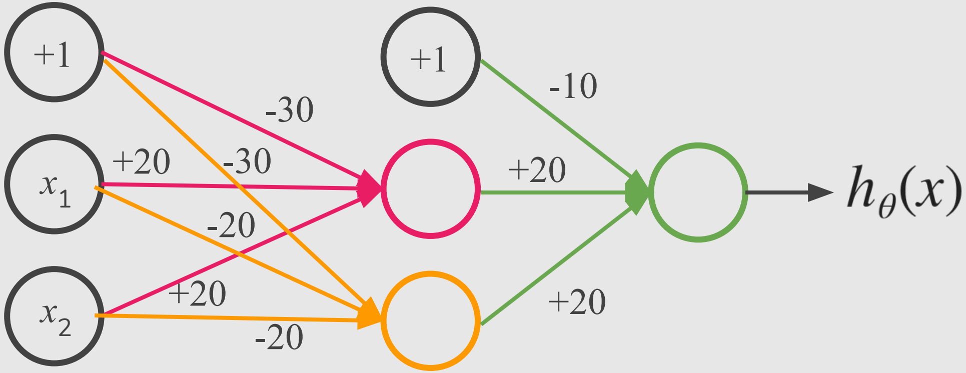
x_1 OR x_2



Example: XNOR $x_1, x_2 \in \{0,1\}$ $y = x_1 \text{ XNOR } x_2$

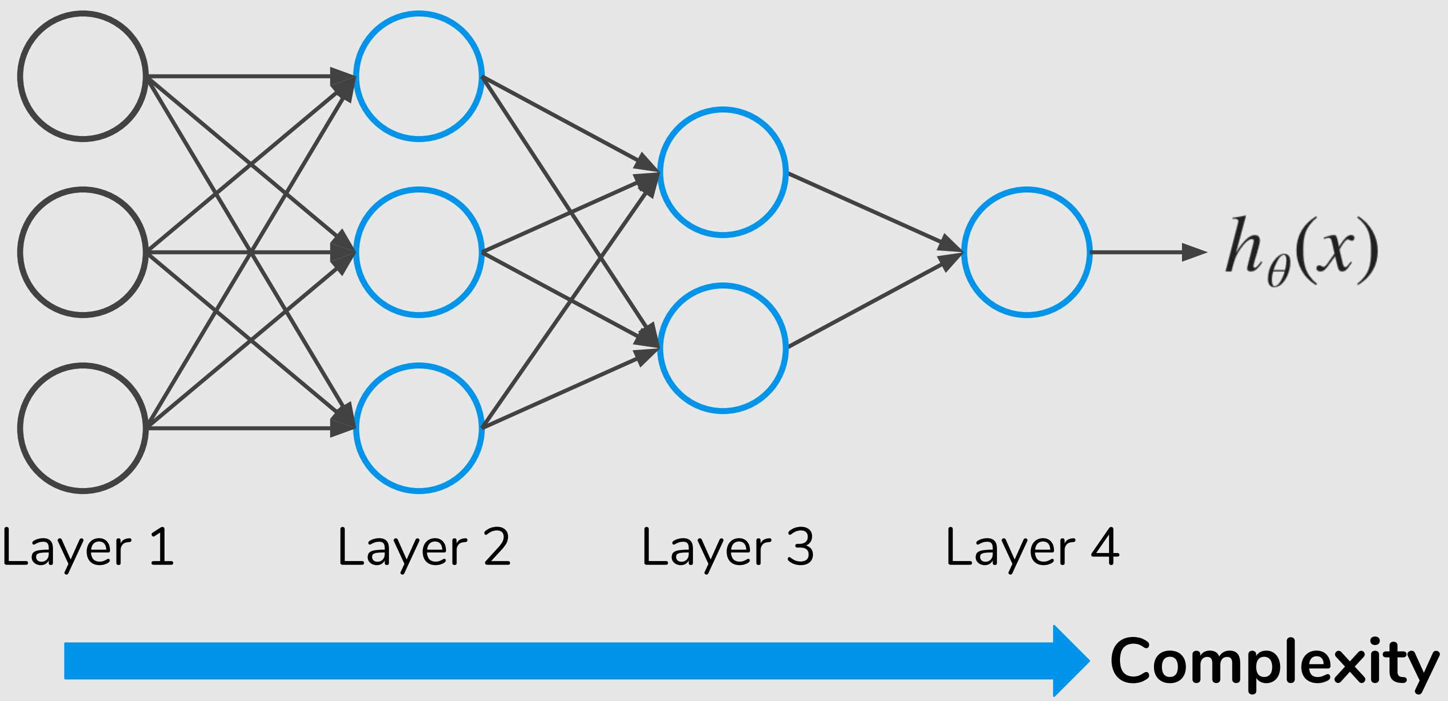


x_1 AND x_2 (NOT x_1) AND (NOT x_2) x_1 OR x_2



x_1	x_2	$h_\theta(x)$
0	0	1
0	1	0
1	0	0
1	1	1

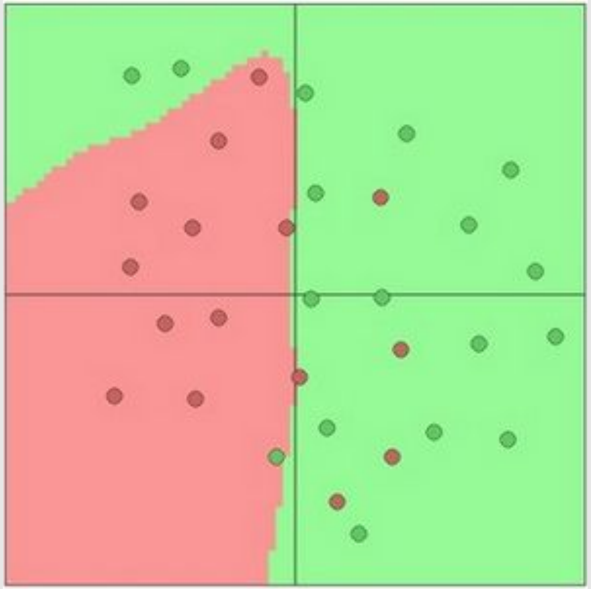
Neural Network Intuition



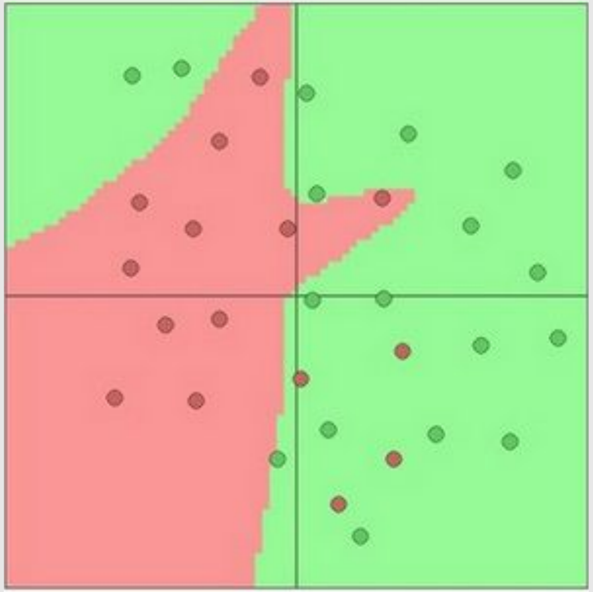
Neural Network Intuition

Toy 2d classification with 2-layer neural network

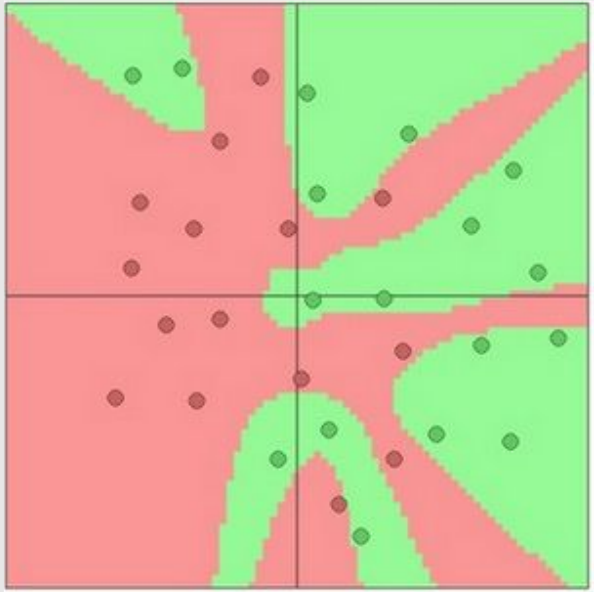
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html



3 hidden neurons



6 hidden neurons



20 hidden neurons

http://cs231n.github.io/assets/nn1/layer_sizes.jpeg

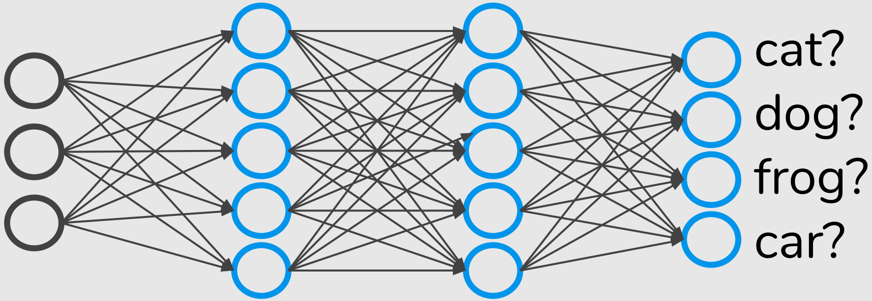
Multi-class Classification

Cat

Dog

Frog

Car

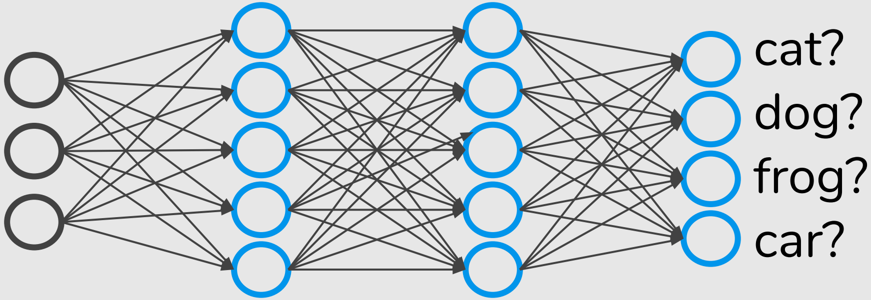



Cat


Dog


Frog


Car

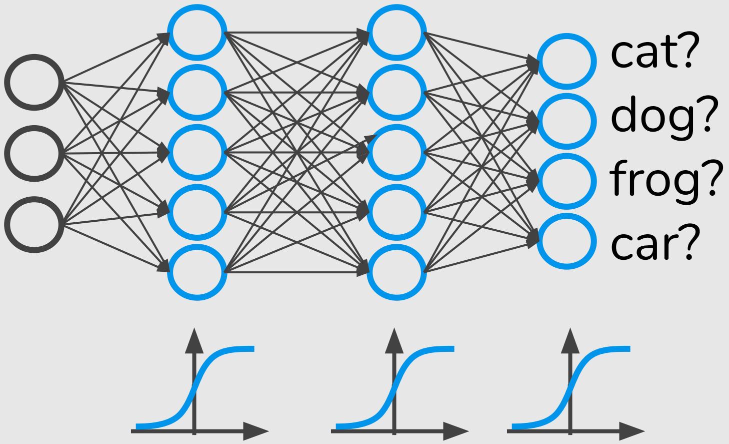


Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

when cat　　when dog　　when frog　　when car

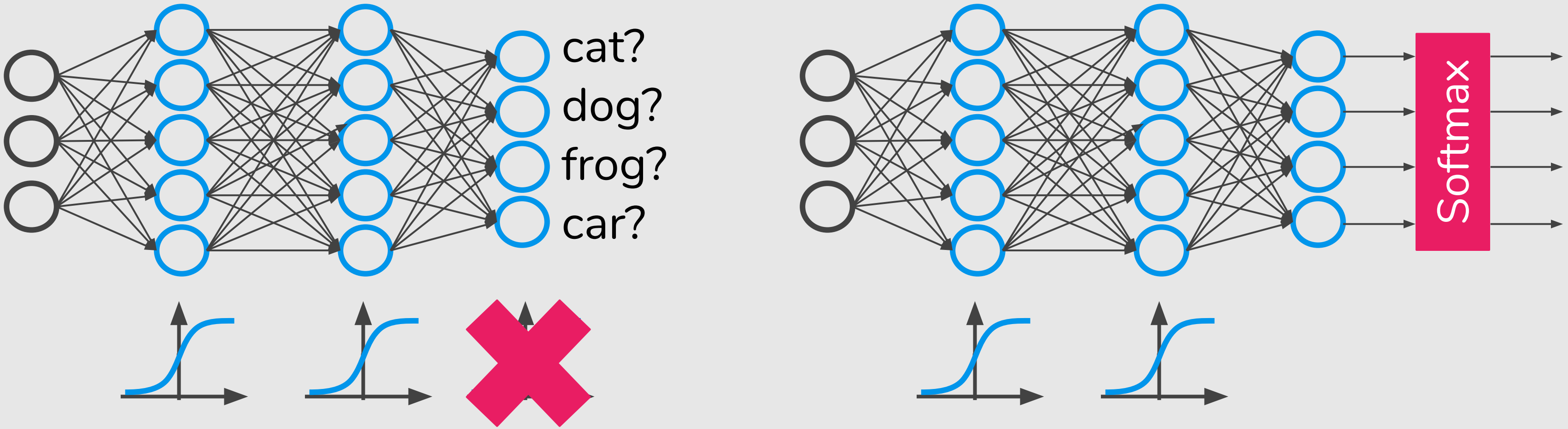
Softmax Classification

The **output layer** is typically modified **by replacing** the individual activation functions **by a shared softmax** function.



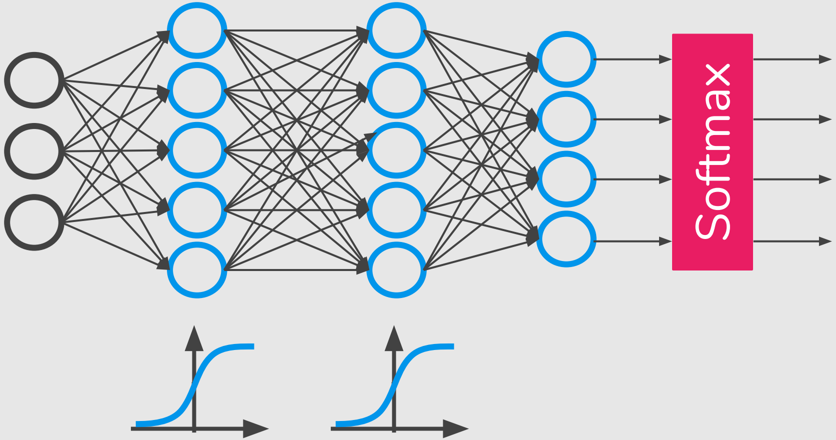
Softmax Classification

The **output layer** is typically modified **by replacing** the individual activation functions **by a shared softmax** function.



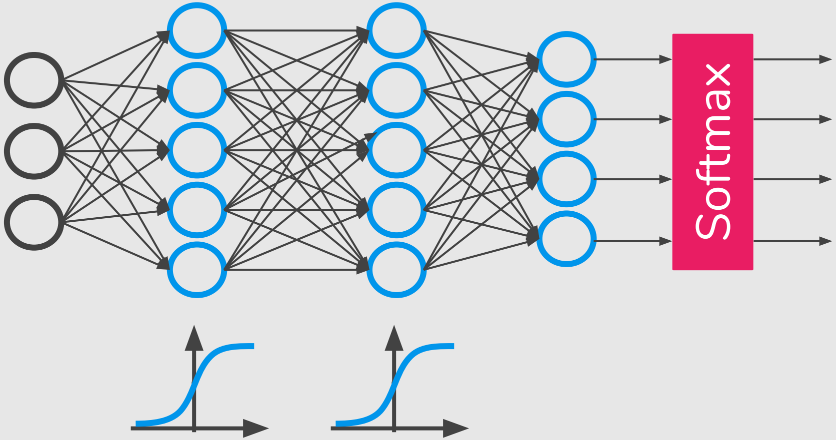
Softmax Classification

The **output layer** is typically modified **by replacing** the individual activation functions **by a shared softmax** function.



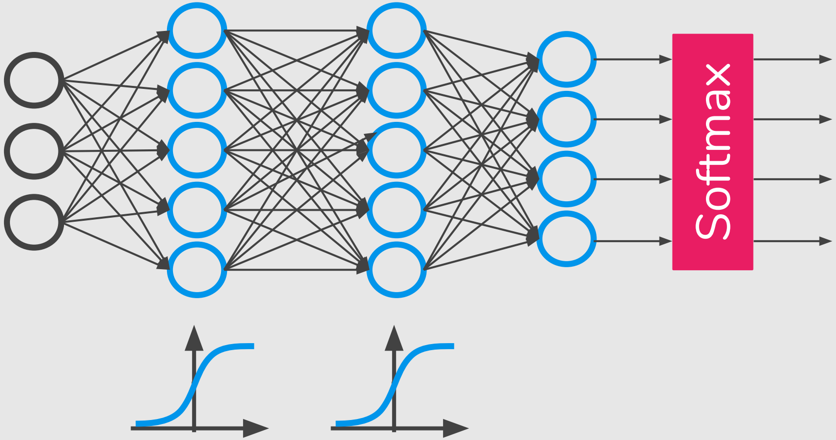
$$f(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax Classification



$$f(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

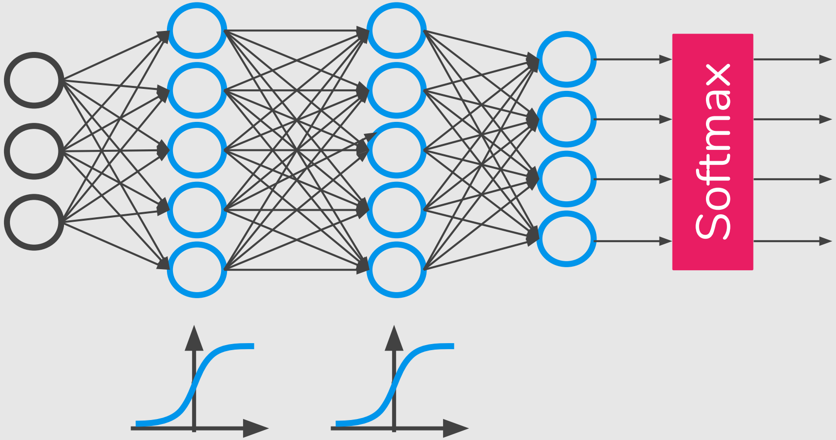
Softmax Classification



Cat	5.1
Dog	3.2
Frog	-1.7
Car	-2.0

$$f(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

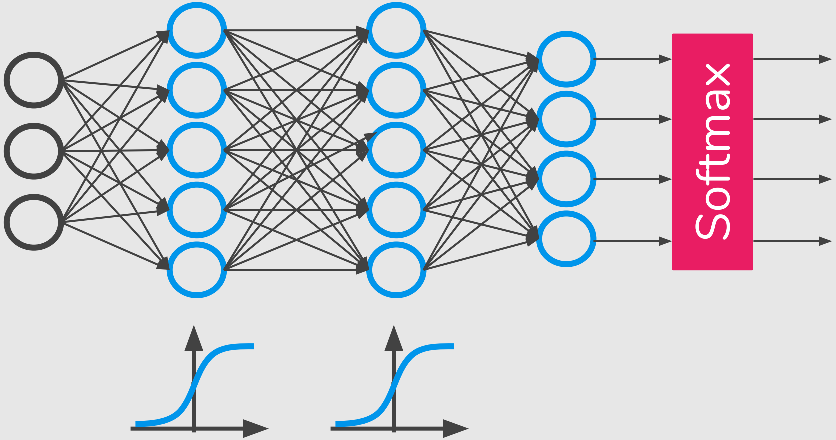
Softmax Classification



Cat	5.1	164.0
Dog	3.2	24.5
Frog	-1.7	0.18
Car	-2.0	0.13

$$f(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Softmax Classification



Cat	5.1	164.0	0.87
Dog	3.2	24.5	0.13
Frog	-1.7	0.18	0.00
Car	-2.0	0.13	0.00

$$f(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Cost Function

Cost Function

Let's first define a few variables that we will need to use:

- L = total number of **layers** in the network
- s_l = number of **units** (not counting bias unit) in layer l
- K = number of **output** units/classes

Cost Function

Let's first define a few variables that we will need to use:

- L = total number of **layers** in the network
- s_l = number of **units** (not counting bias unit) in layer l
- K = number of **output** units/classes

Our cost function for neural networks is going to be a generalization of the one we used for **logistic regression**.

Logistic Regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$$

Logistic Regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$$

Neural Network:

$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th}$ output

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$

Logistic Regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta^2$$

Neural Network:

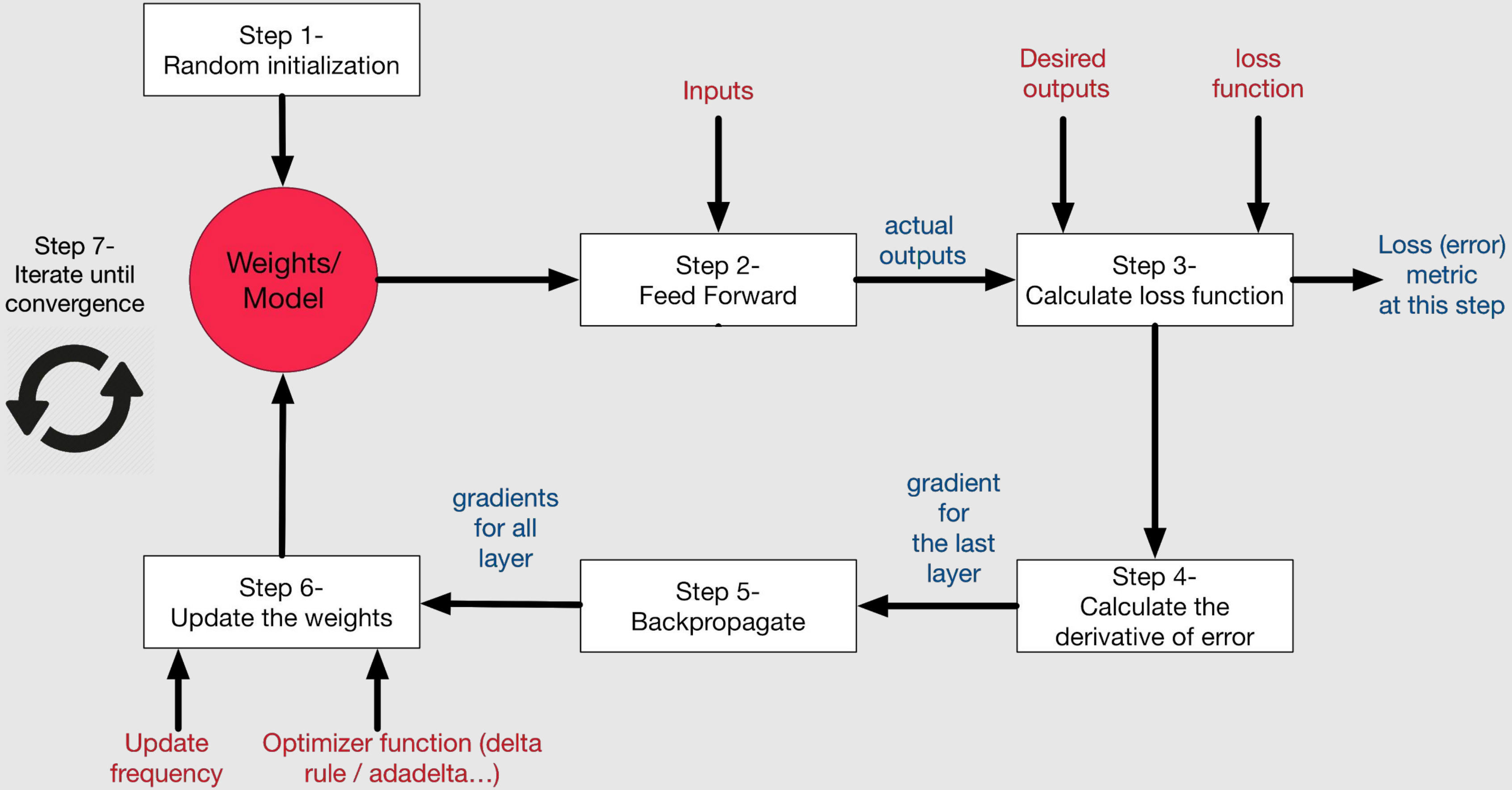
$h_\Theta(x) \in \mathbb{R}^K \qquad (h_\Theta(x))_i = i^{th}$ output

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$

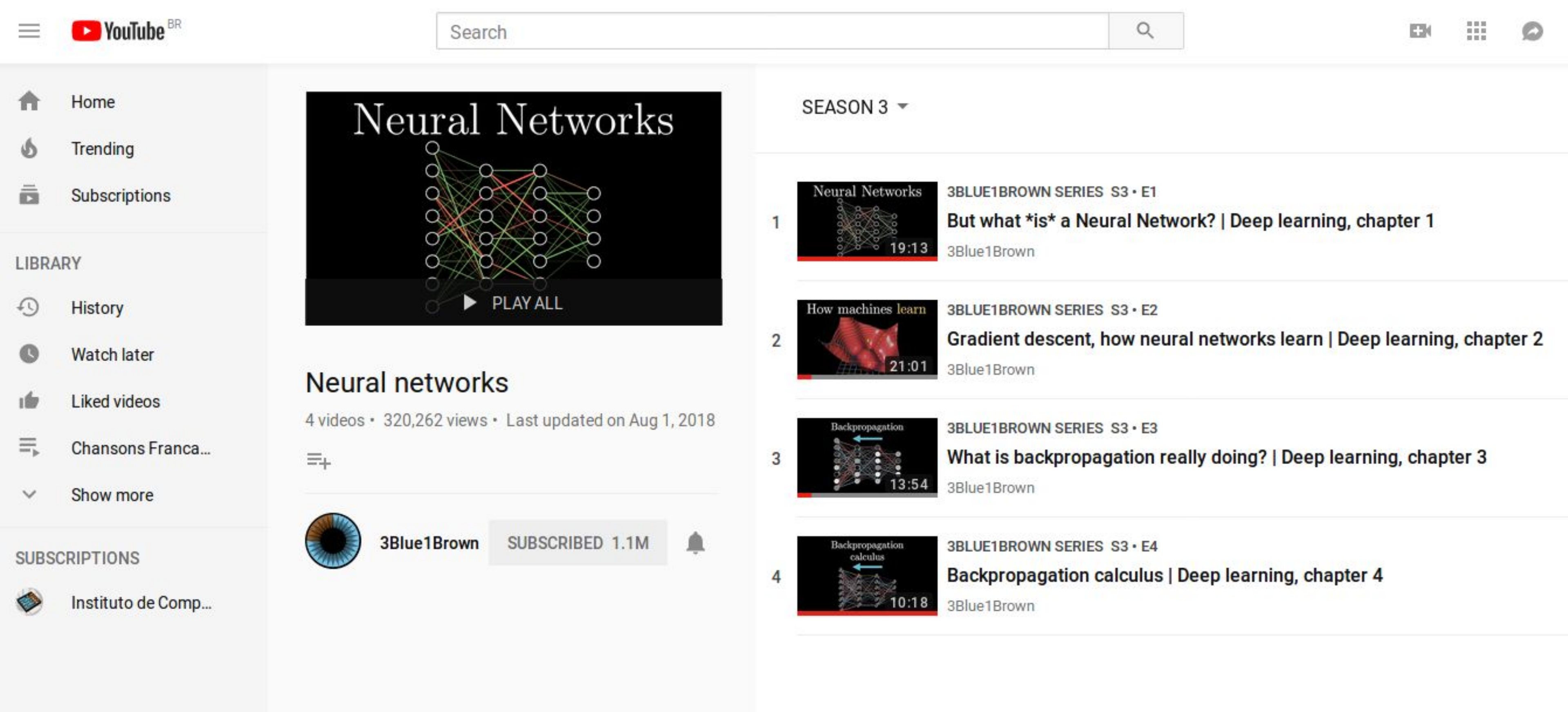
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Training a Neural Network

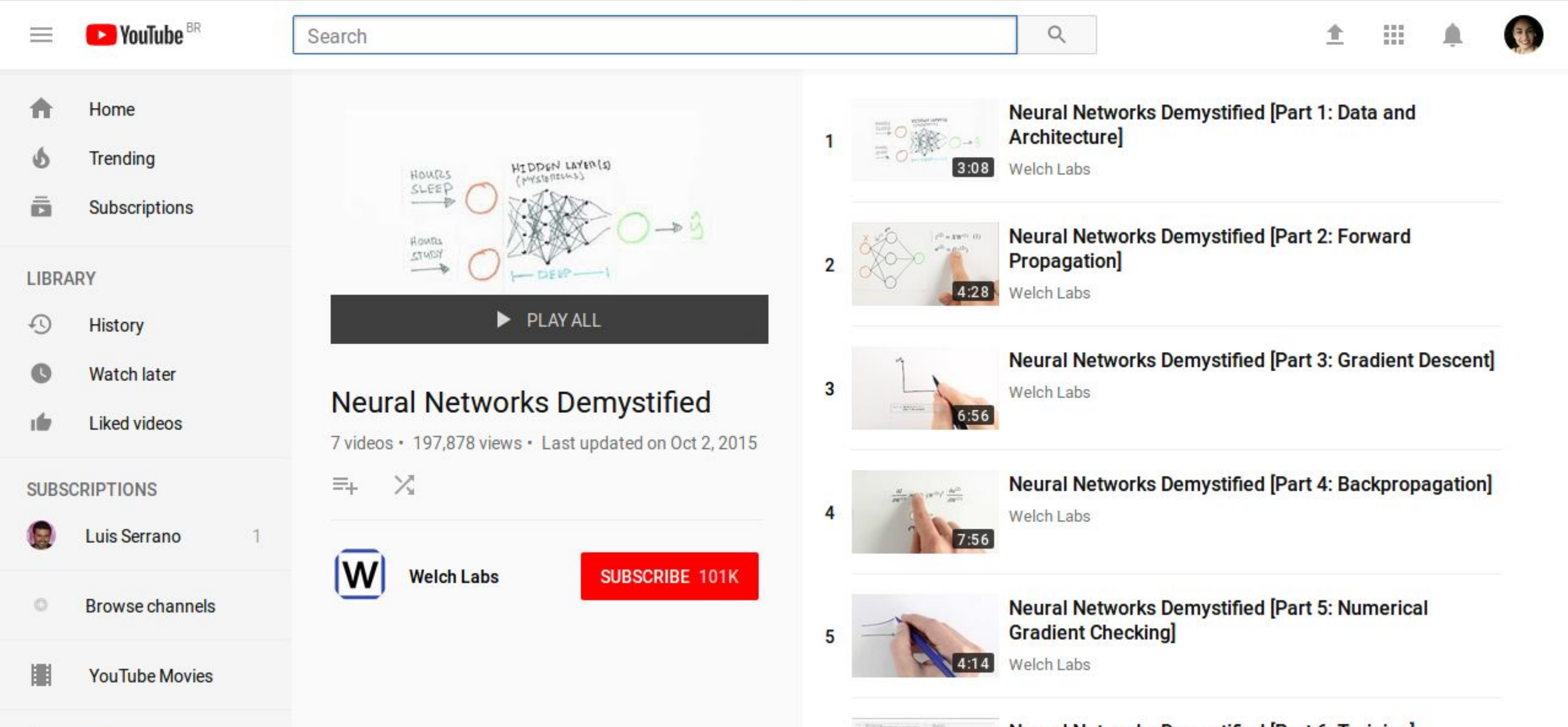
Training a Neural Network



Neural Networks (3Blue1Brown)



Neural Networks Demystified (in Python)



References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 10
- Pattern Recognition and Machine Learning, Chap. 5
- Pattern Classification, Chap. 6
- Free online book: **http://neuralnetworksanddeeplearning.com**

Machine Learning Courses

- https://www.coursera.org/learn/machine-learning, Week 4 & 5
- https://www.coursera.org/learn/neural-networks