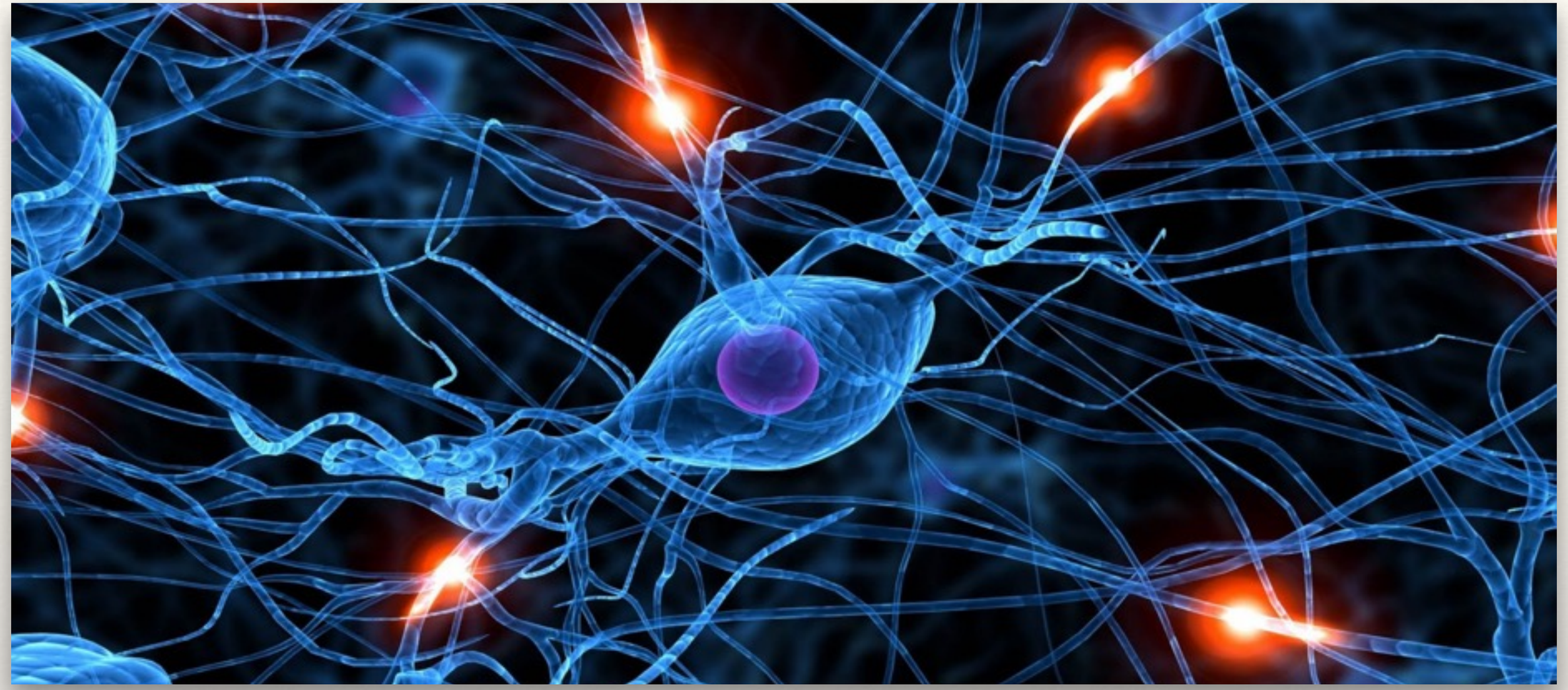


Learning from Data — 30/11/2015 — Johannes Bjerva | j.bjerva@rug.nl

Neural Networks and Deep Learning

Lecture overview

- Neural Networks
  - Background
  - Intuitions
  - Technicalities
- Deep Learning
  - Overview and examples

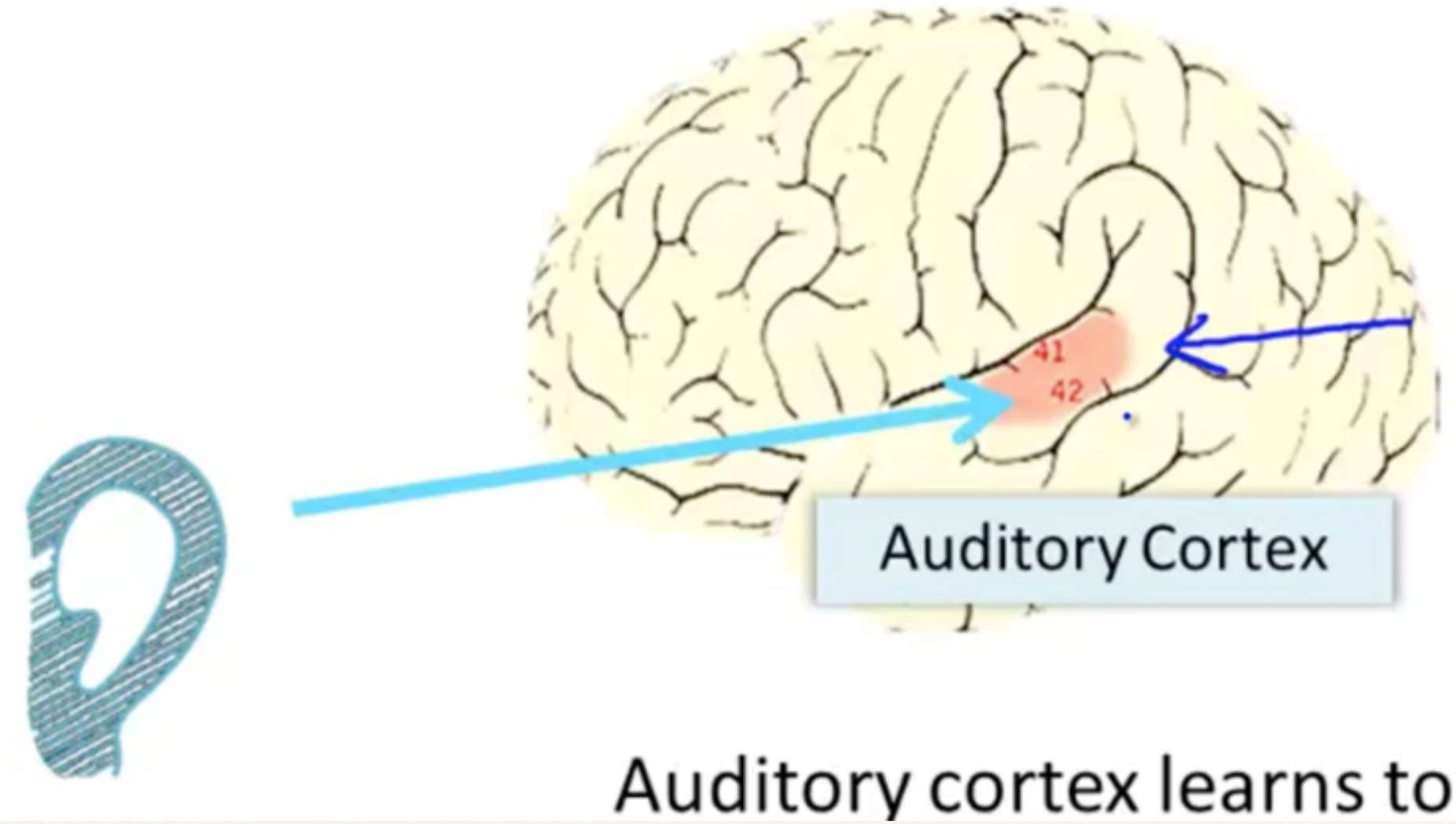
Many slides based on the Coursera ML slides by Andrew Ng

Background

- Attempts to mimic the brain
- Widely used in the ‘80s and ‘90s
- Recent resurgence in popularity

The Power of the Brain

The "one learning algorithm" hypothesis

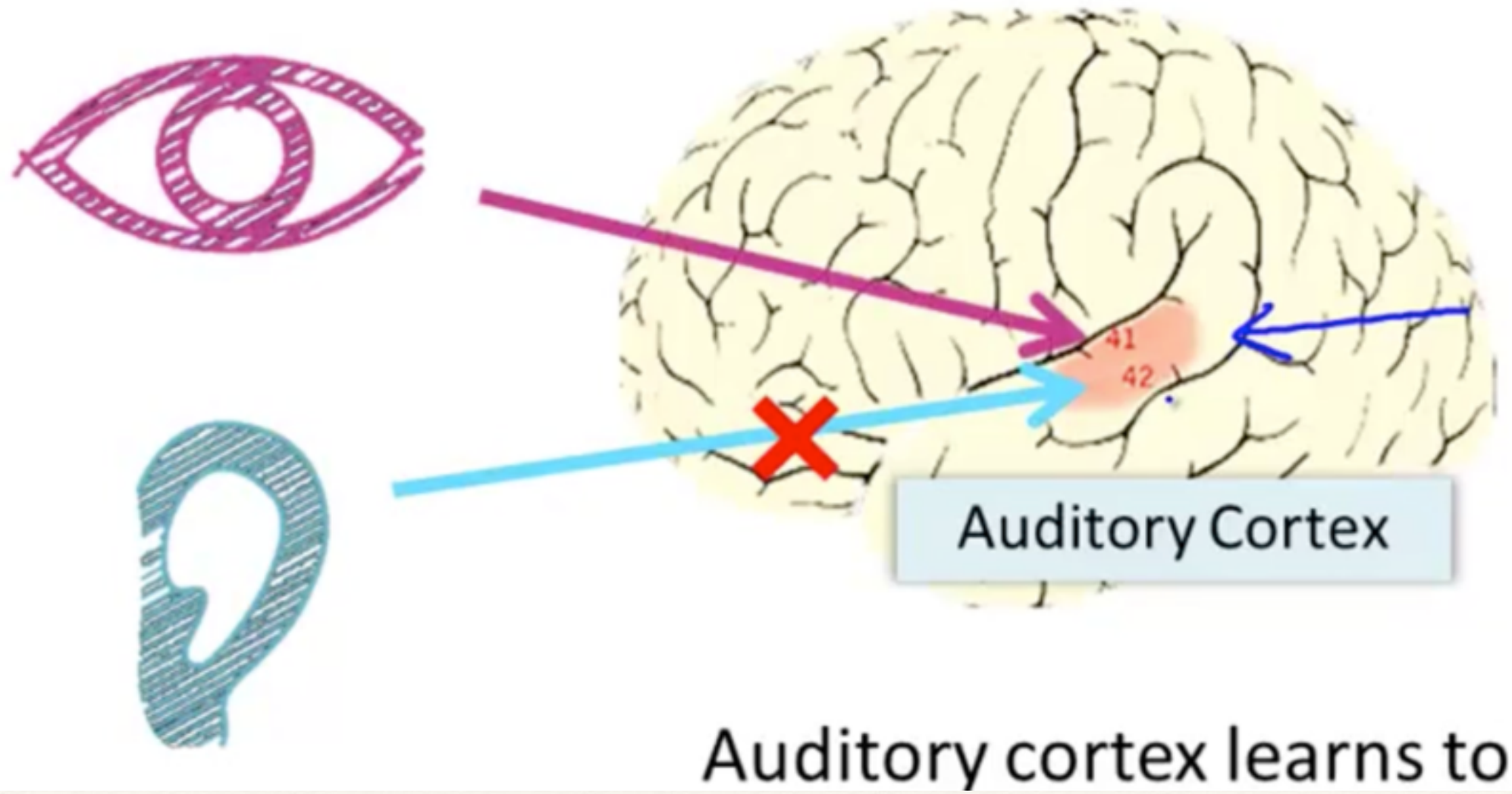


Auditory cortex learns to see

[Roe et al., 1992]

The Power of the Brain

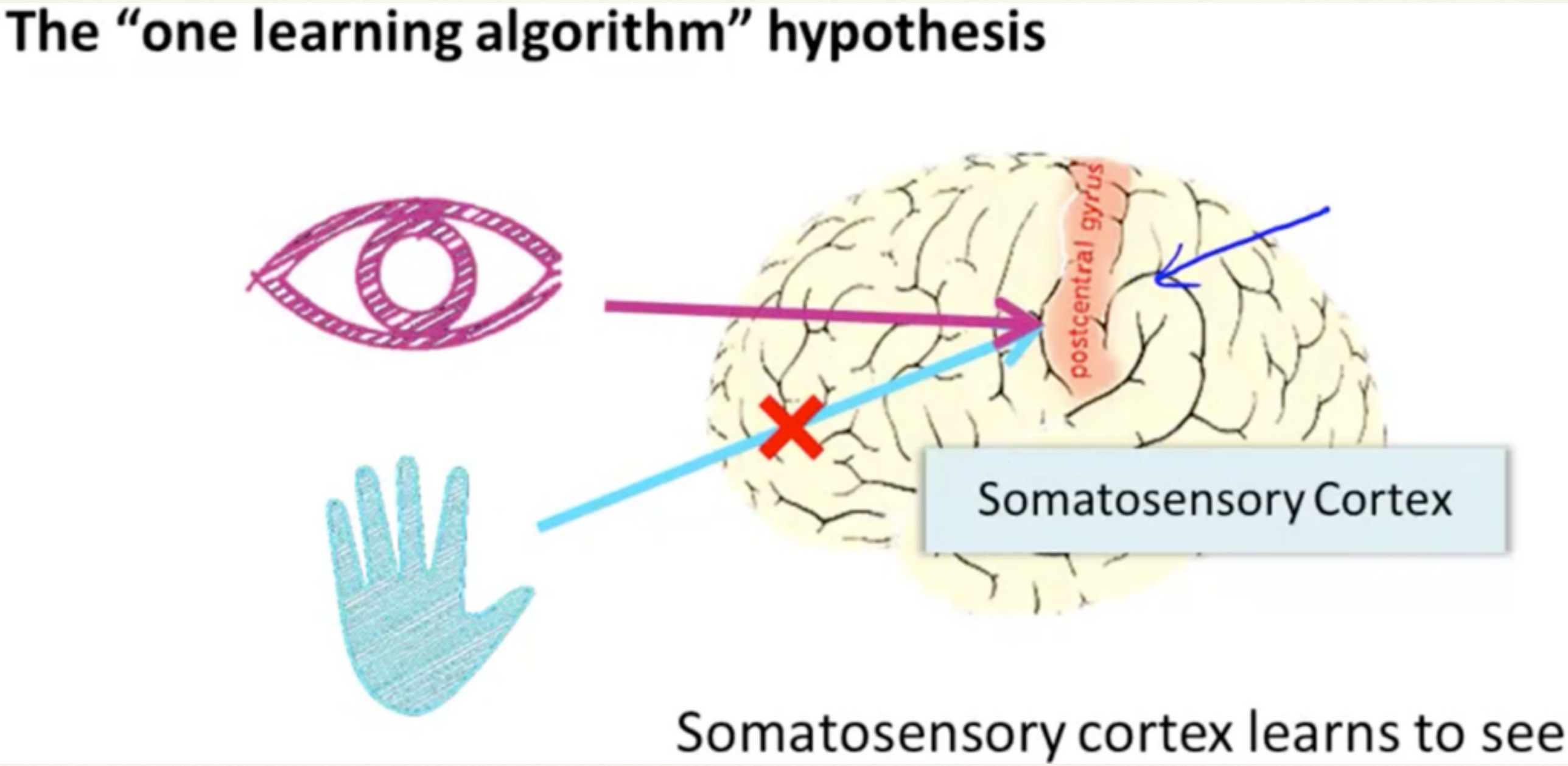
The "one learning algorithm" hypothesis



Auditory cortex learns to see

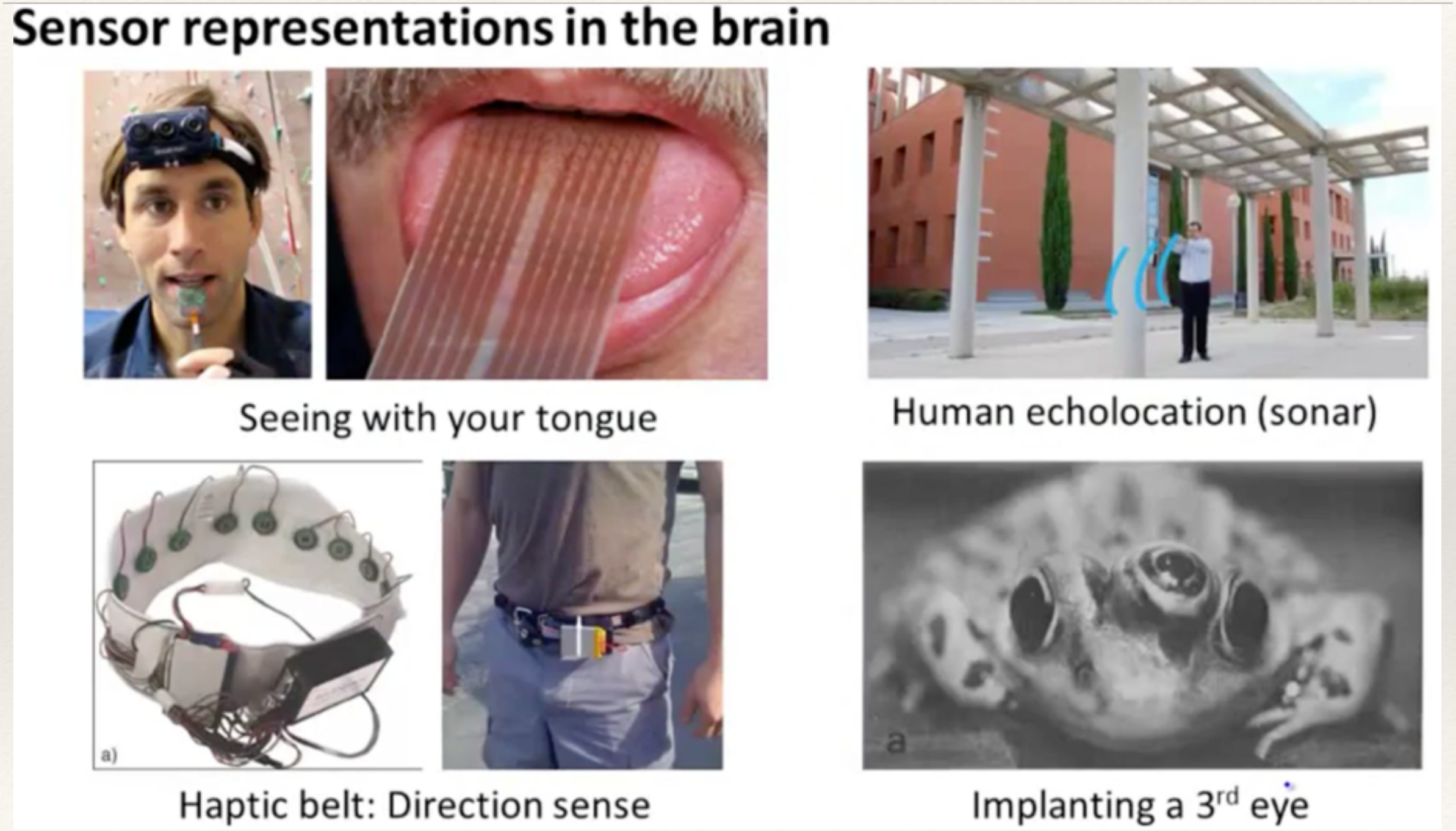
[Roe et al., 1992]

The Power of the Brain



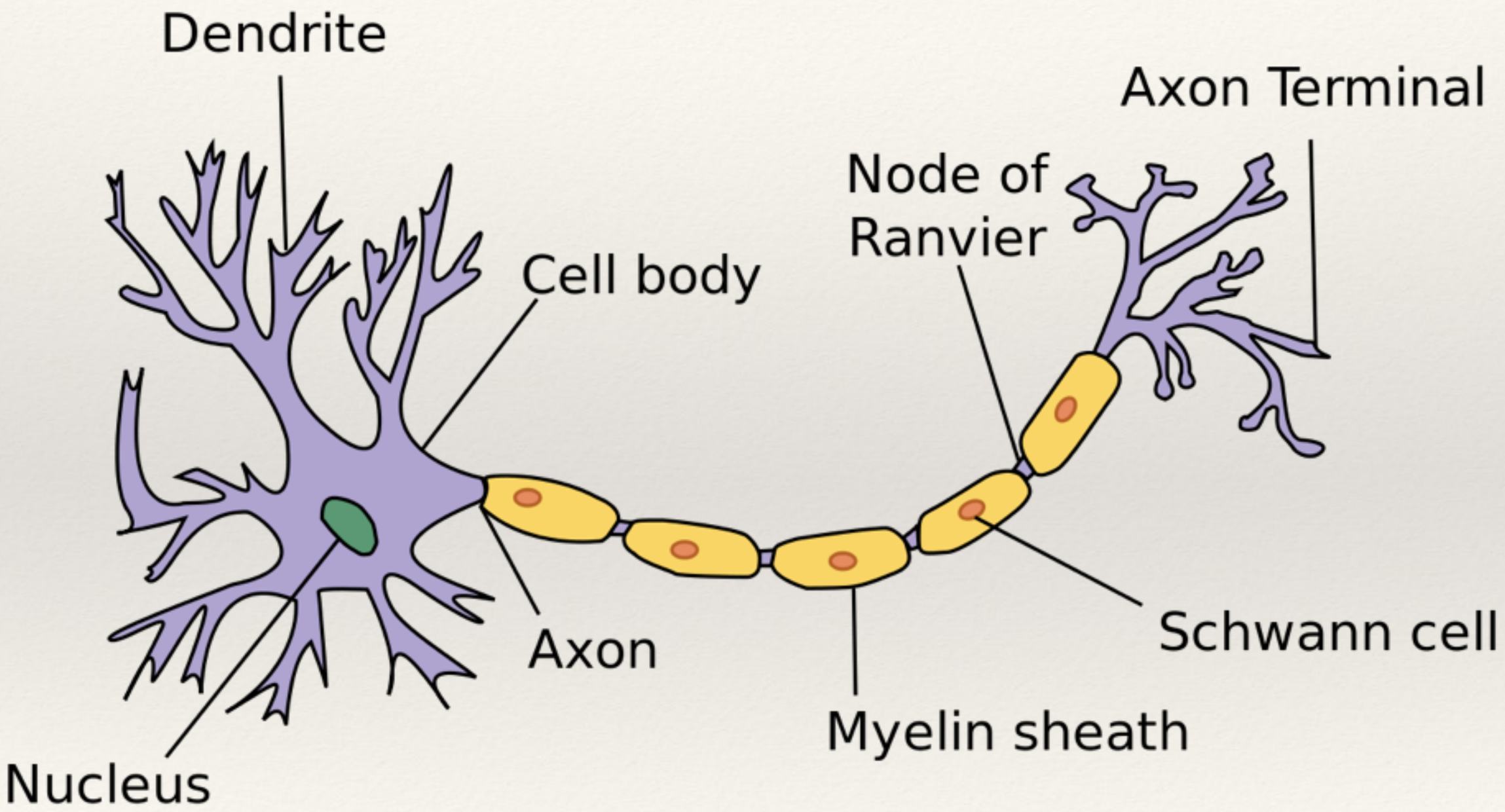
[Metin & Frost, 1989]

The Power of the Brain



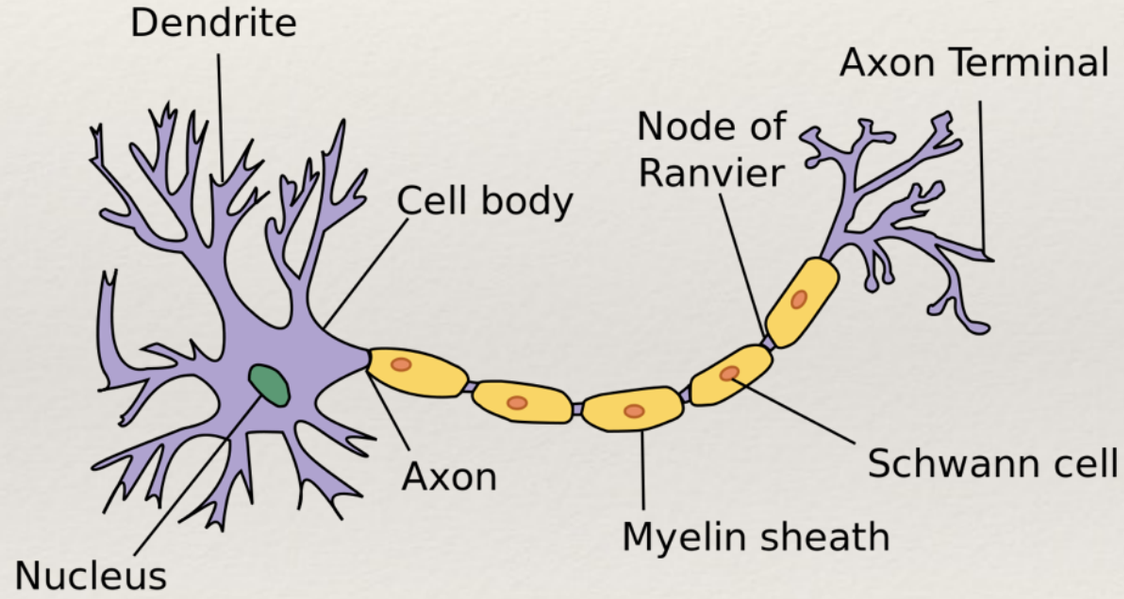
[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]

Biological basis



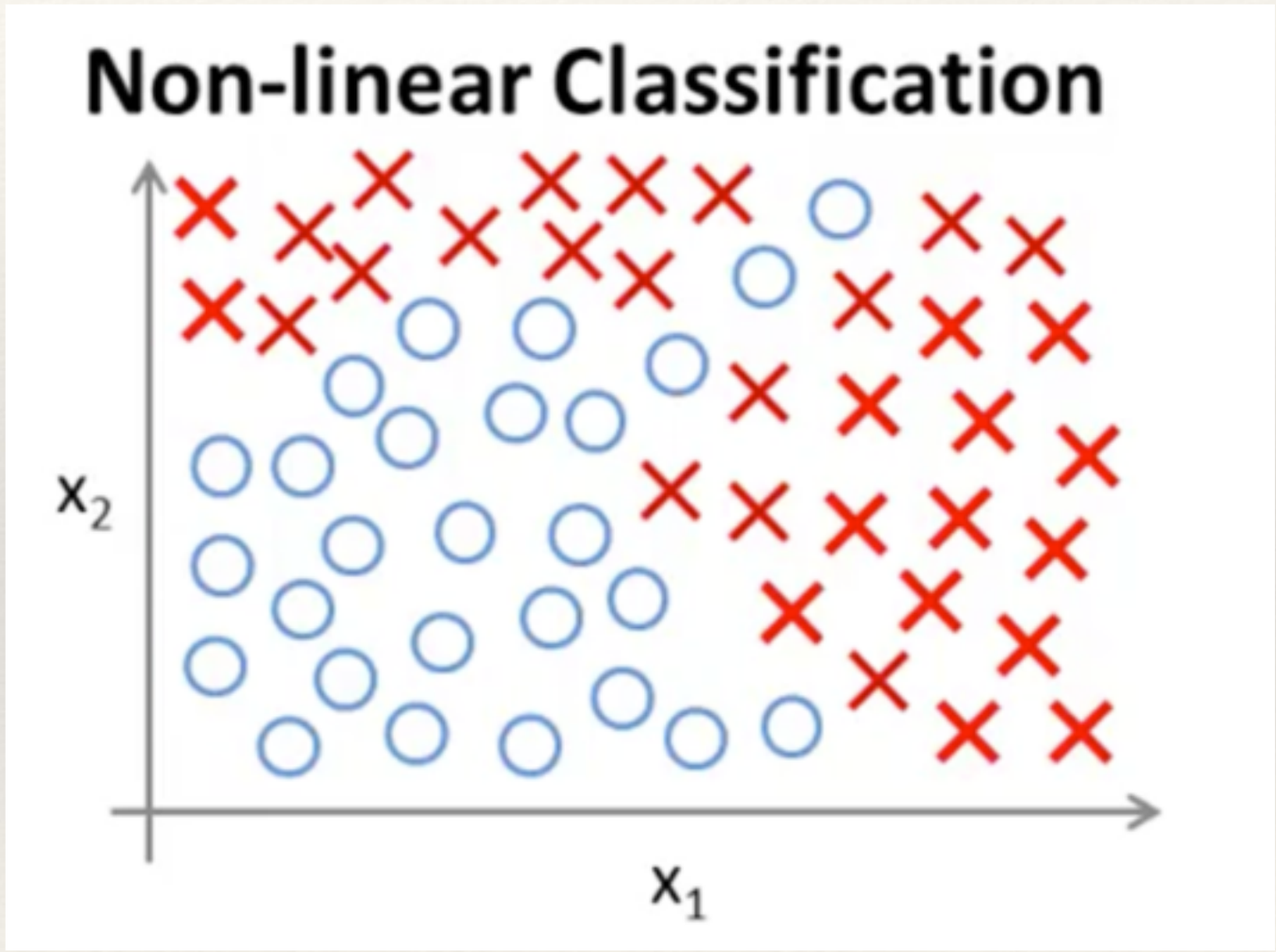
Biological basis

- Biologically inspired
- … but not very realistic

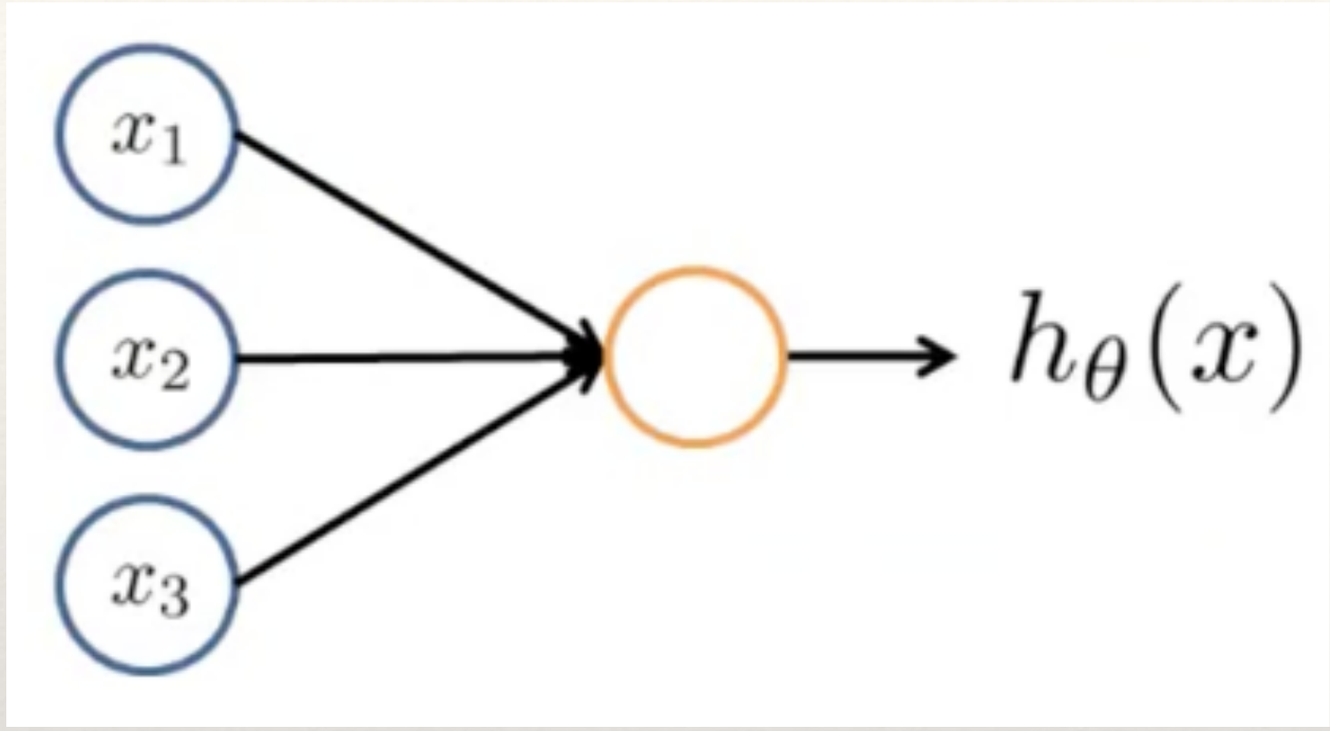


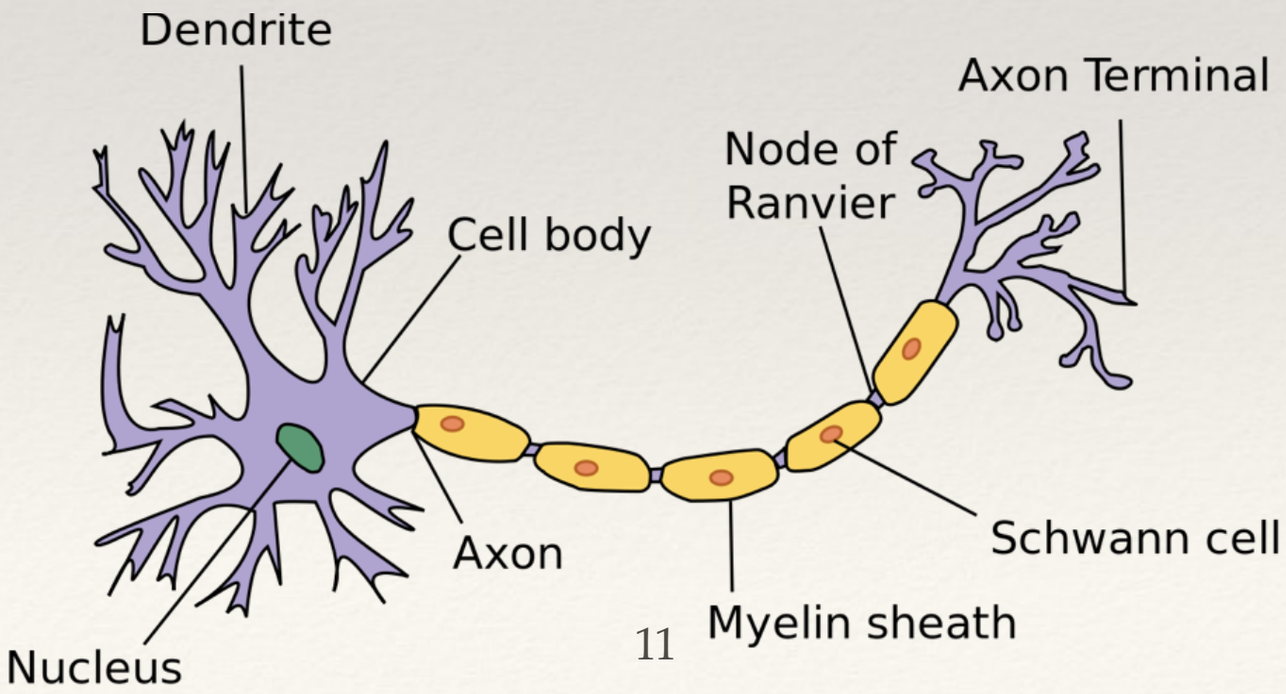
Why Neural Networks?

- Non-linearity
- Power
  - Can represent *any* function

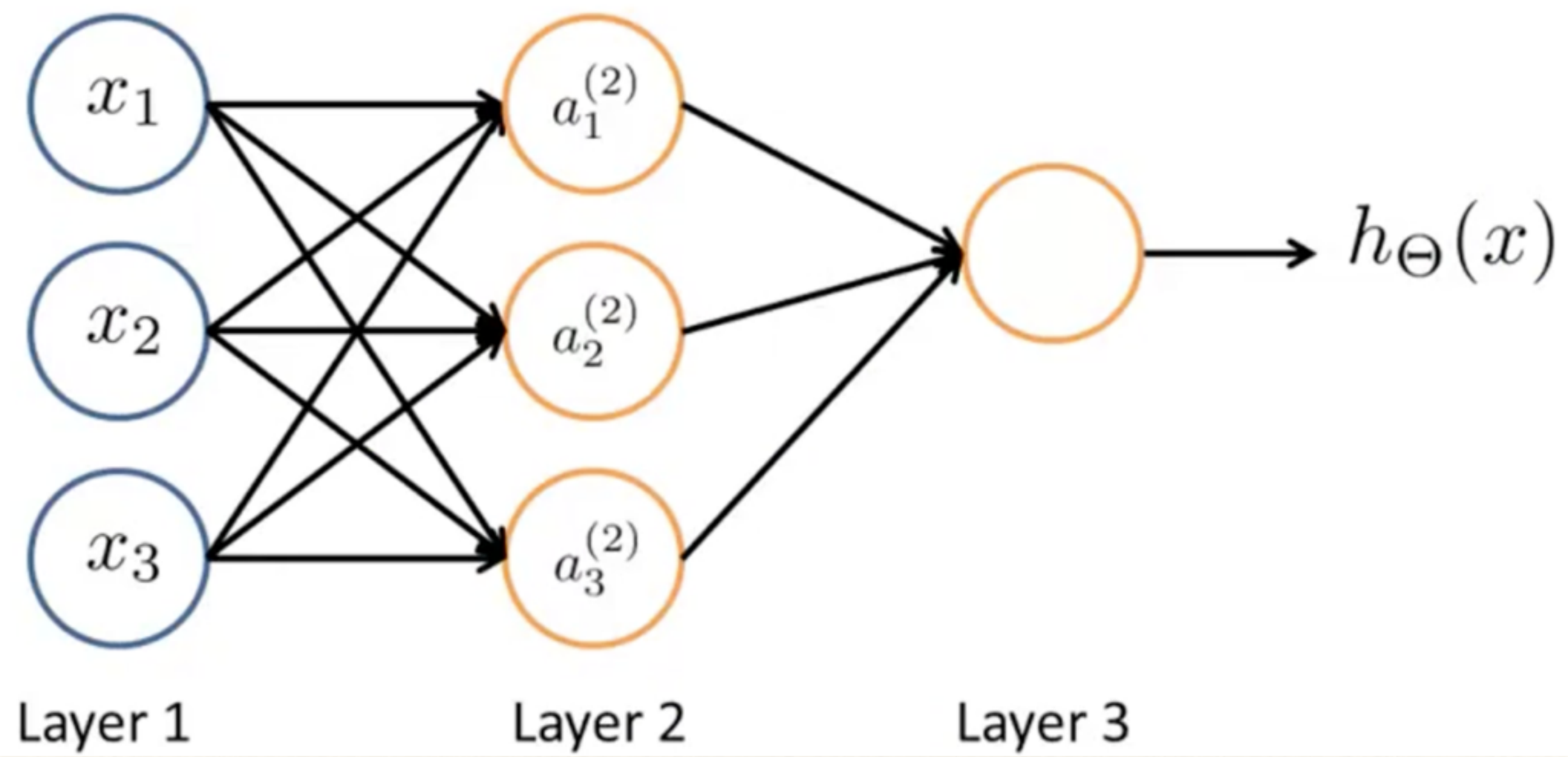


Neuron model

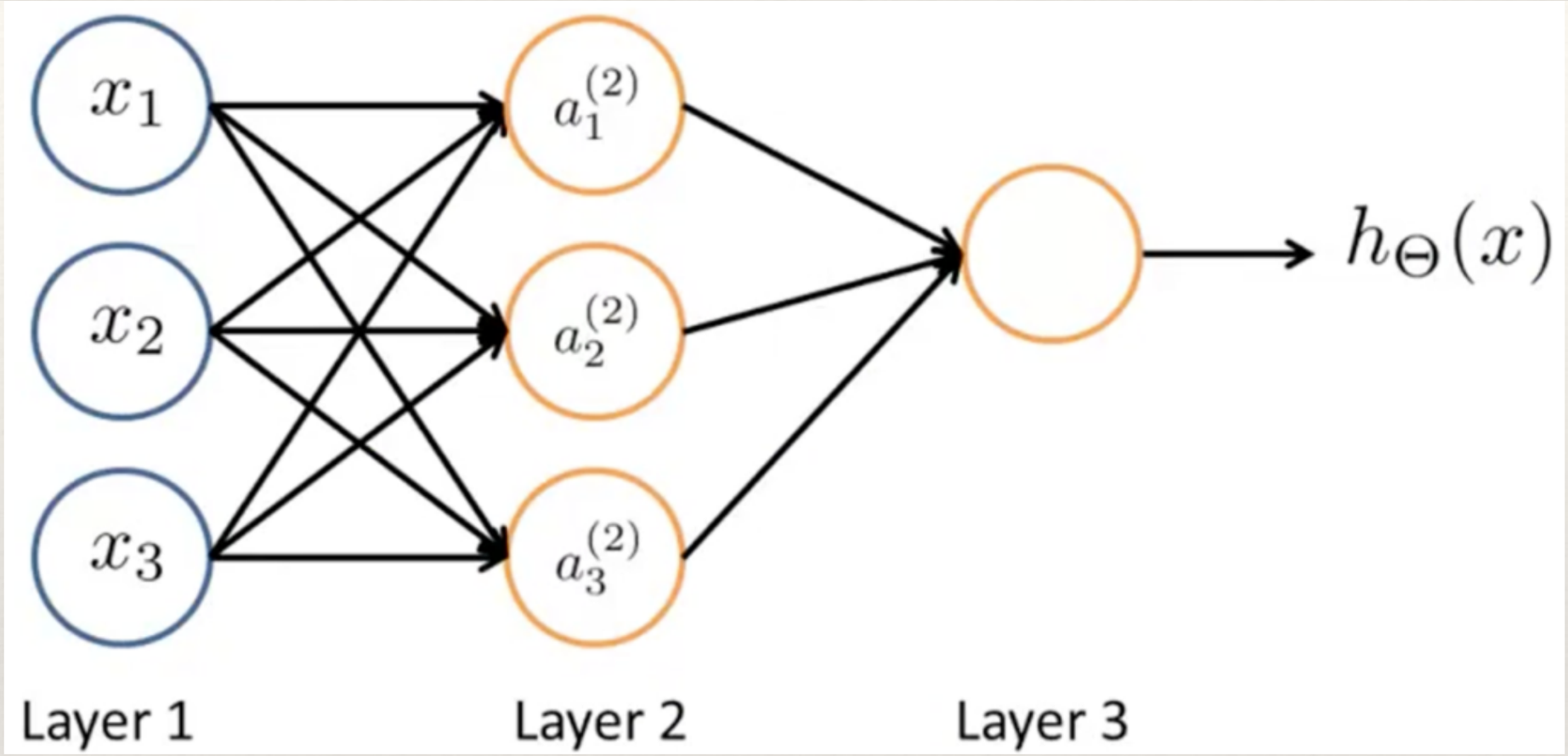




Neural Network - Representation

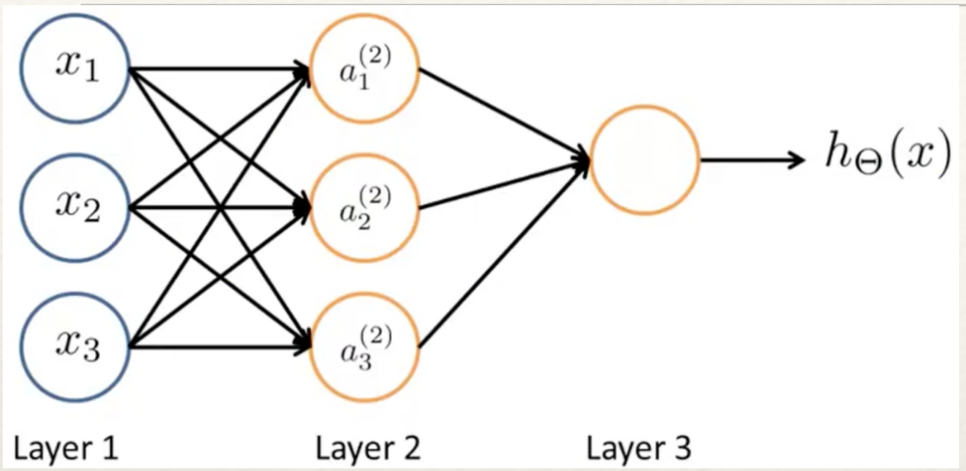


Neural Network - Representation



- $a_i^{(j)}$ = activation of unit i in layer j
- Θ = all activation weights
- $\Theta^{(j)}$ = weights for mapping from layer j to $j + i$

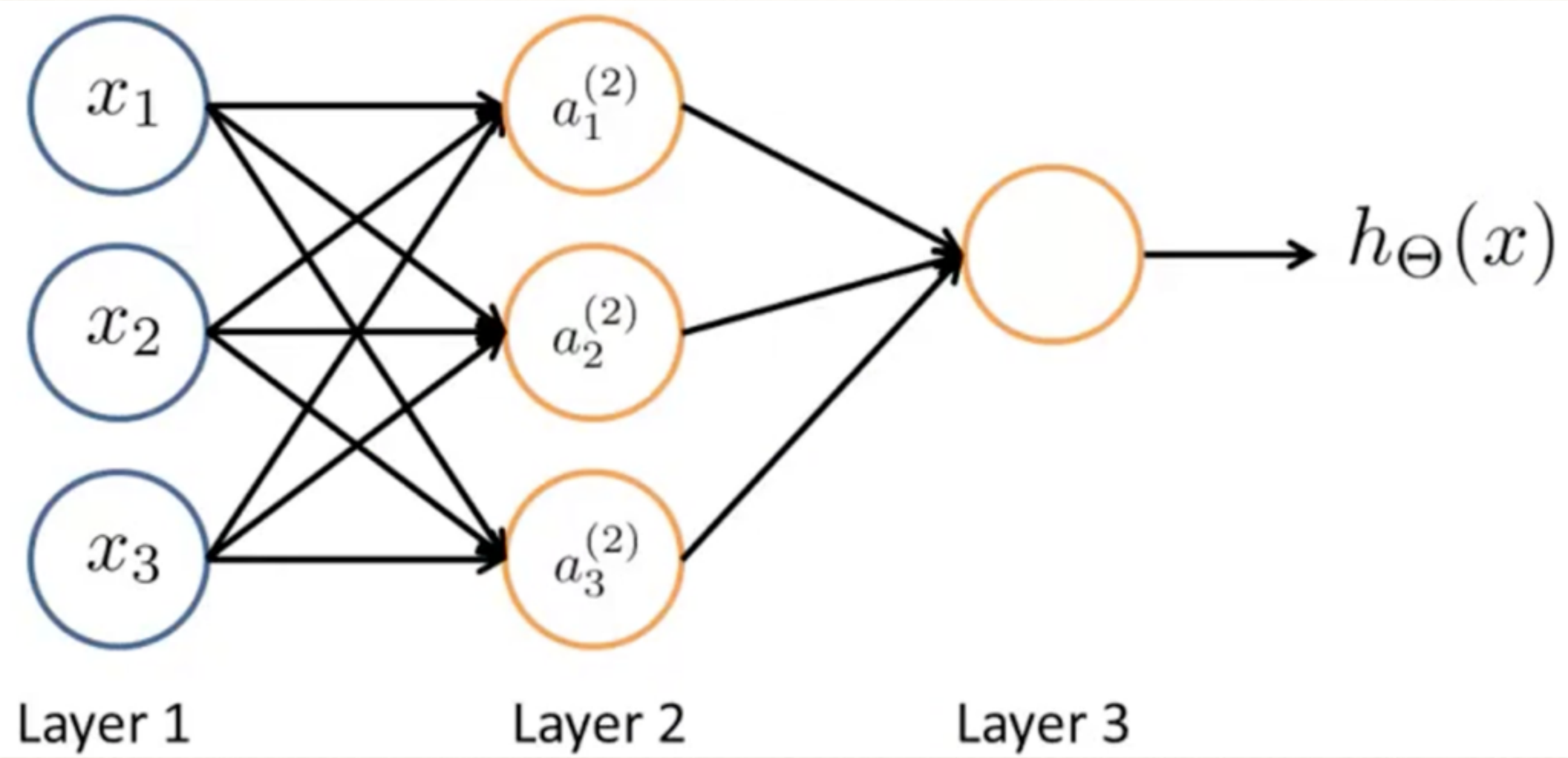
Forward propagation



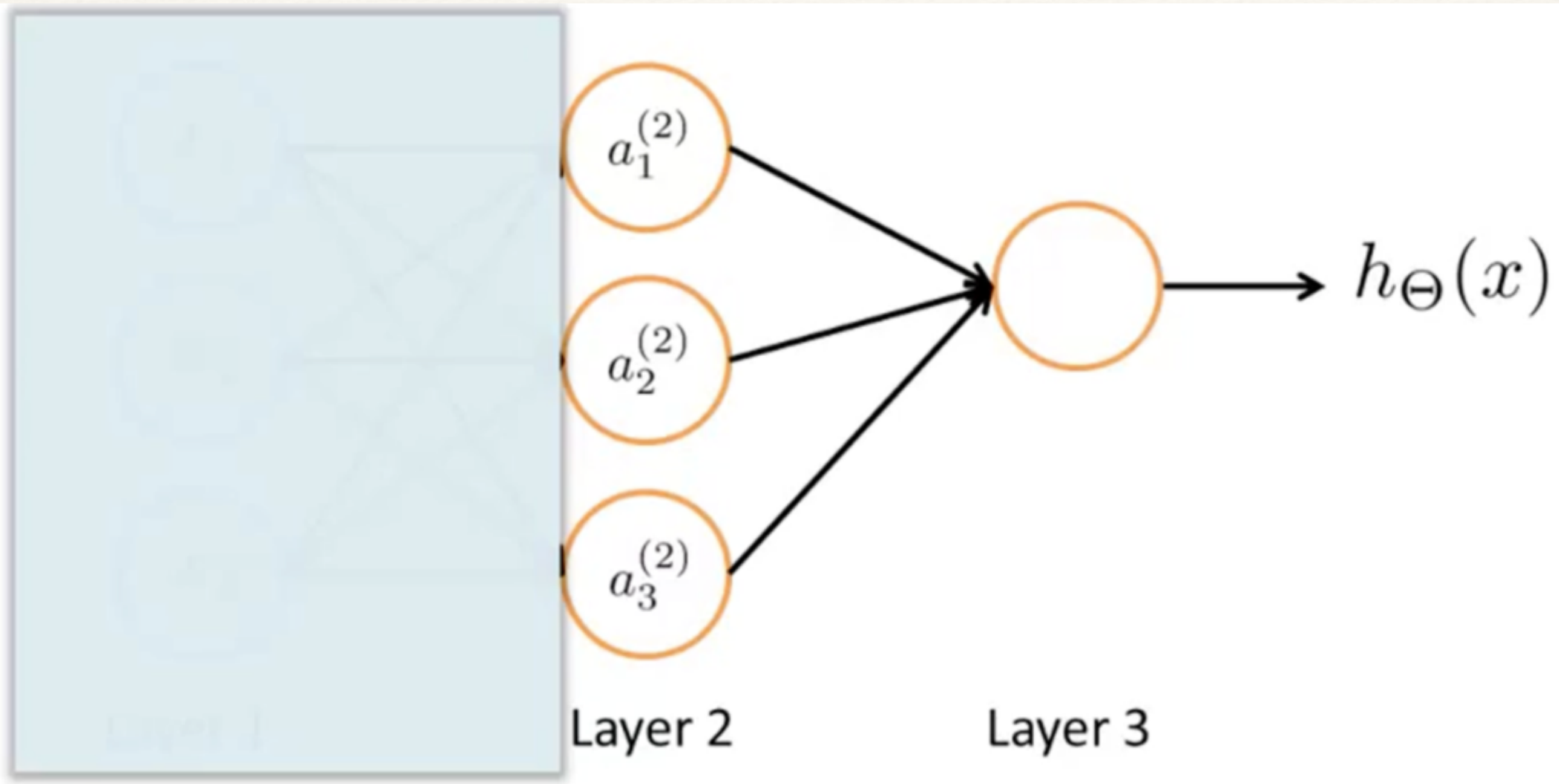
- $a_i^{(j)}$ = activation of unit i in layer j
- Θ = all weights
- $\Theta^{(j)}$ = weights for mapping from layer j to $j + i$

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

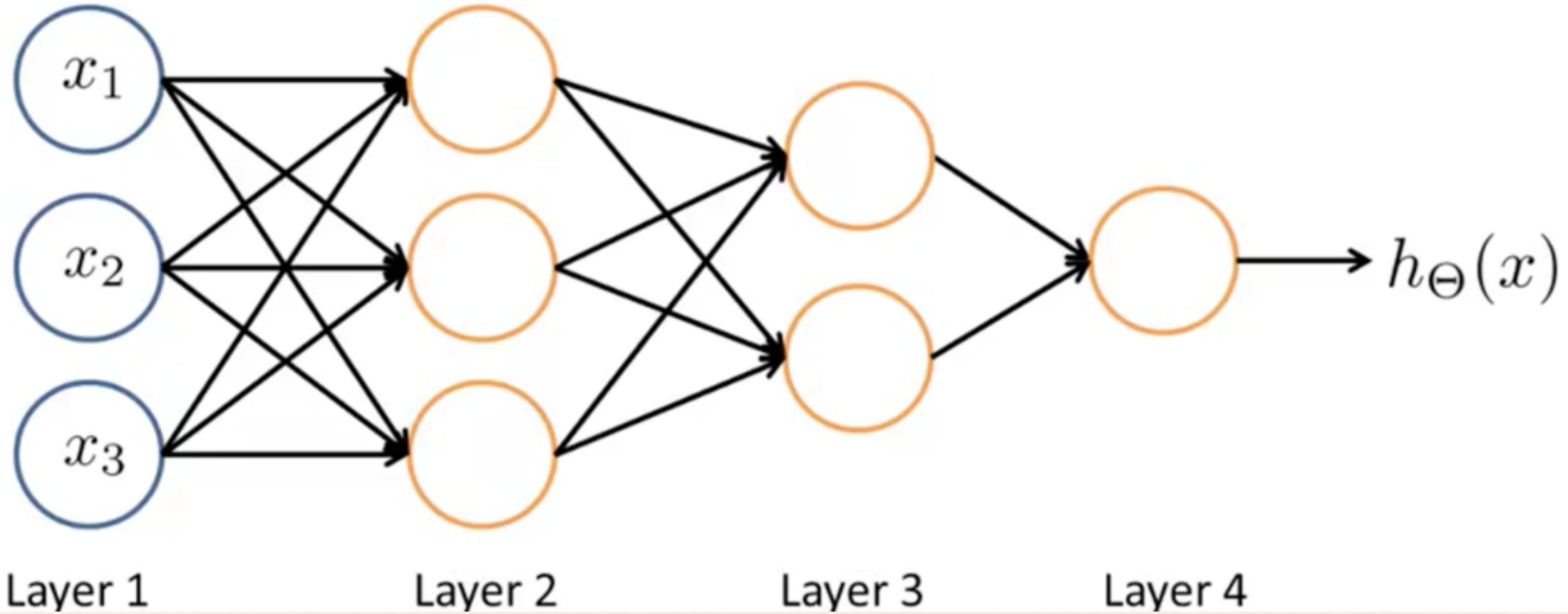
Automatic feature learning



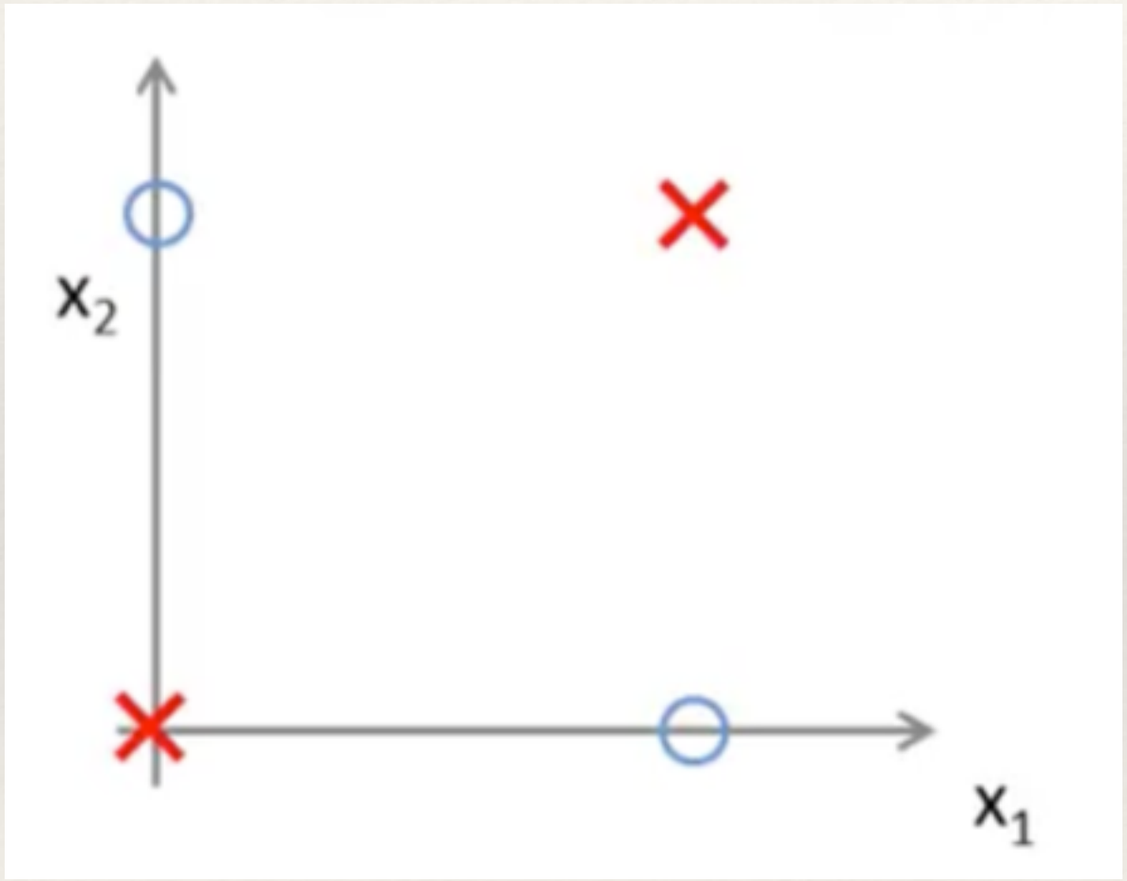
Automatic feature learning



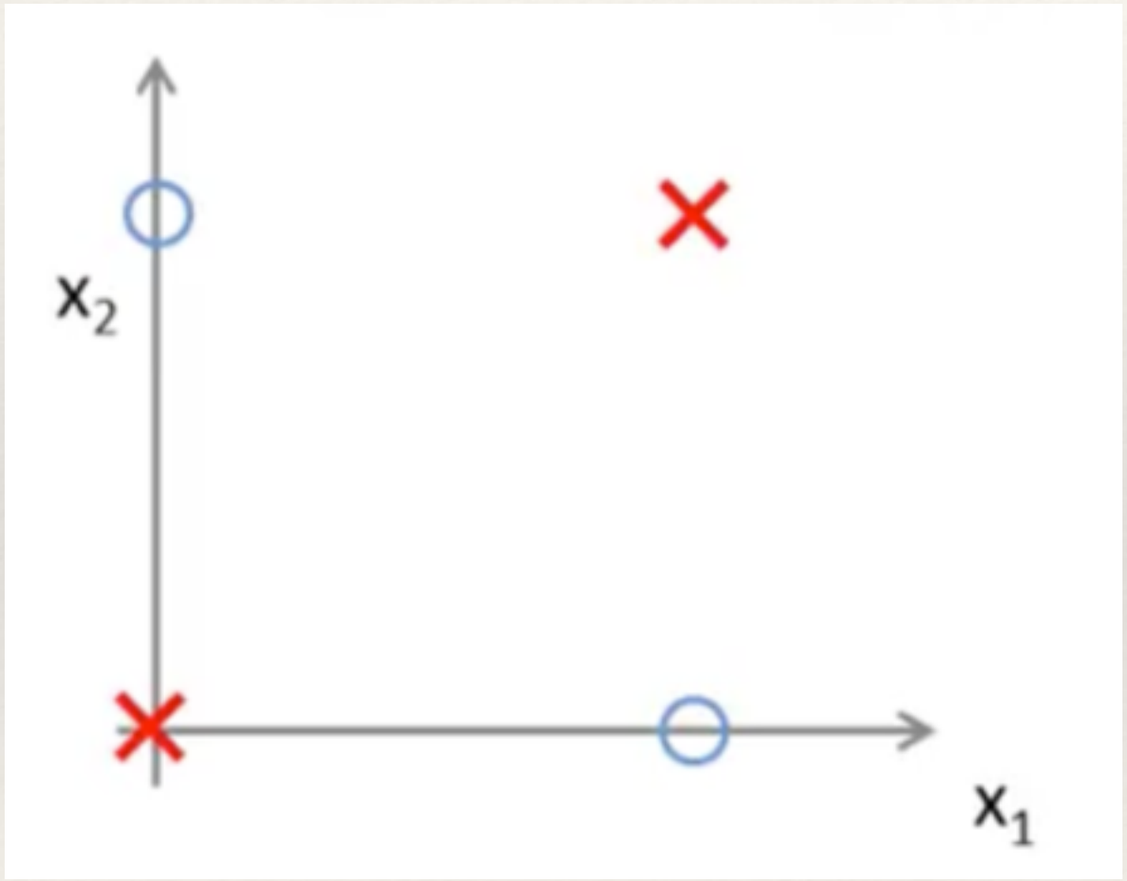
Network Architectures

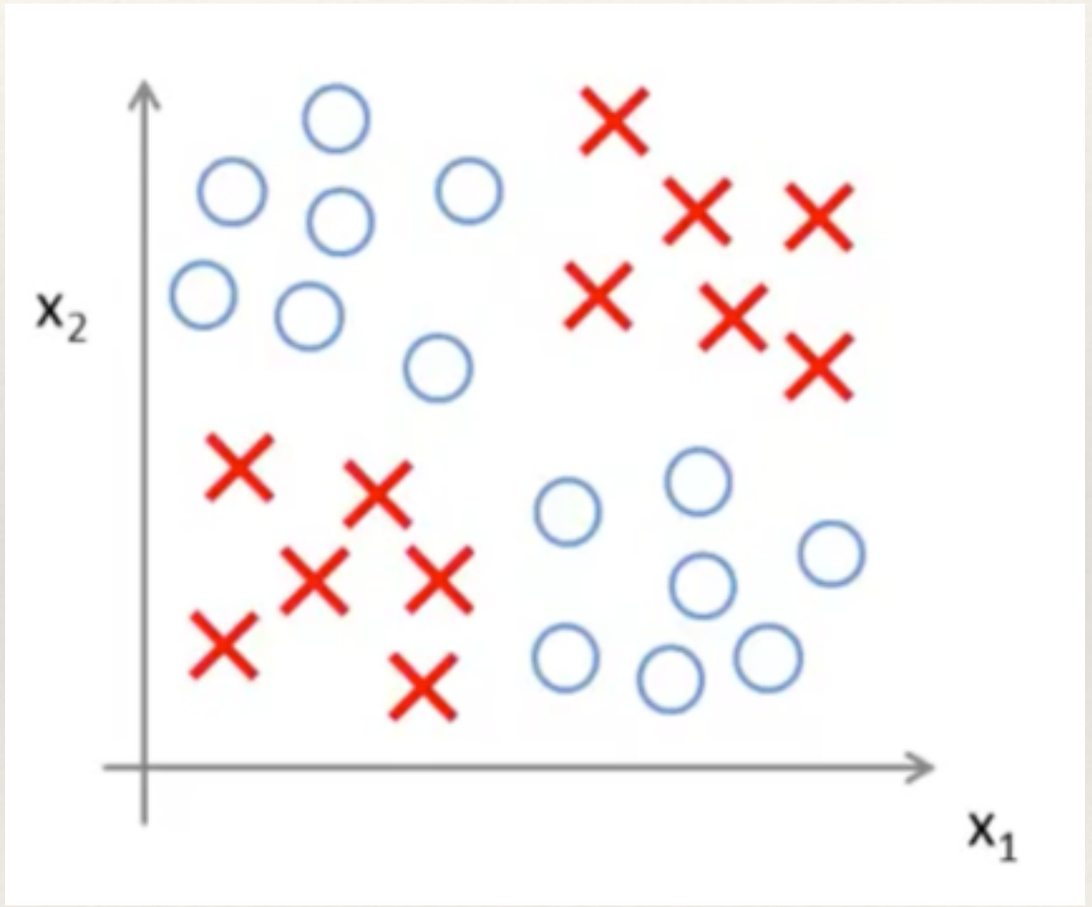


Intuitions

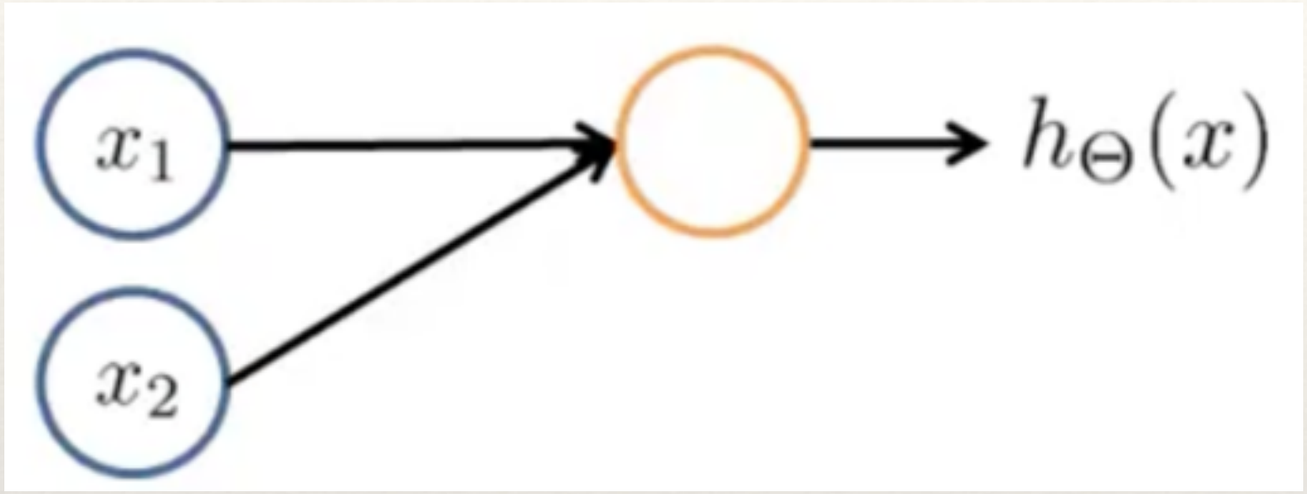


Intuitions - XNOR

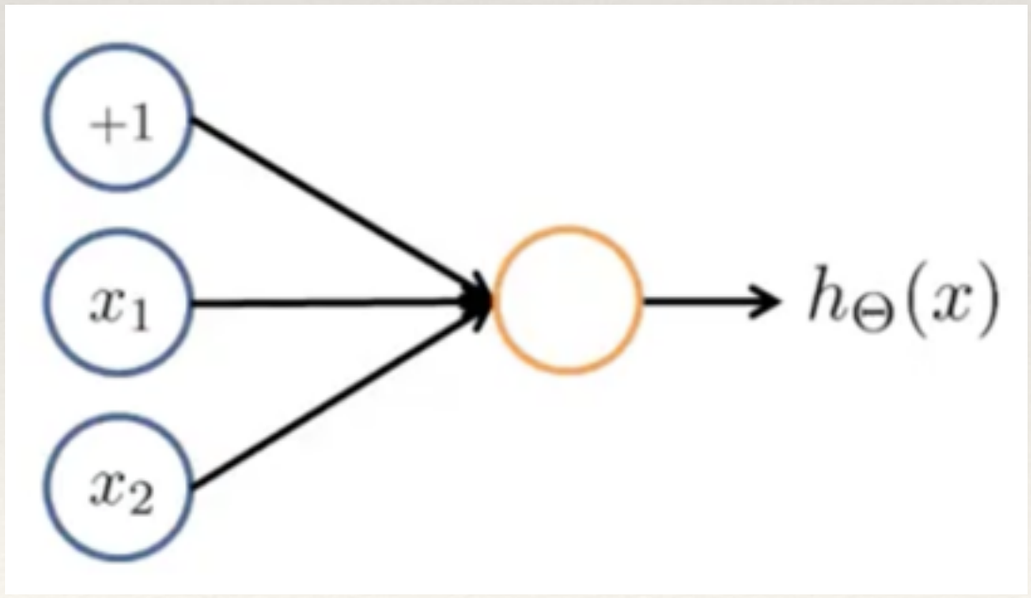




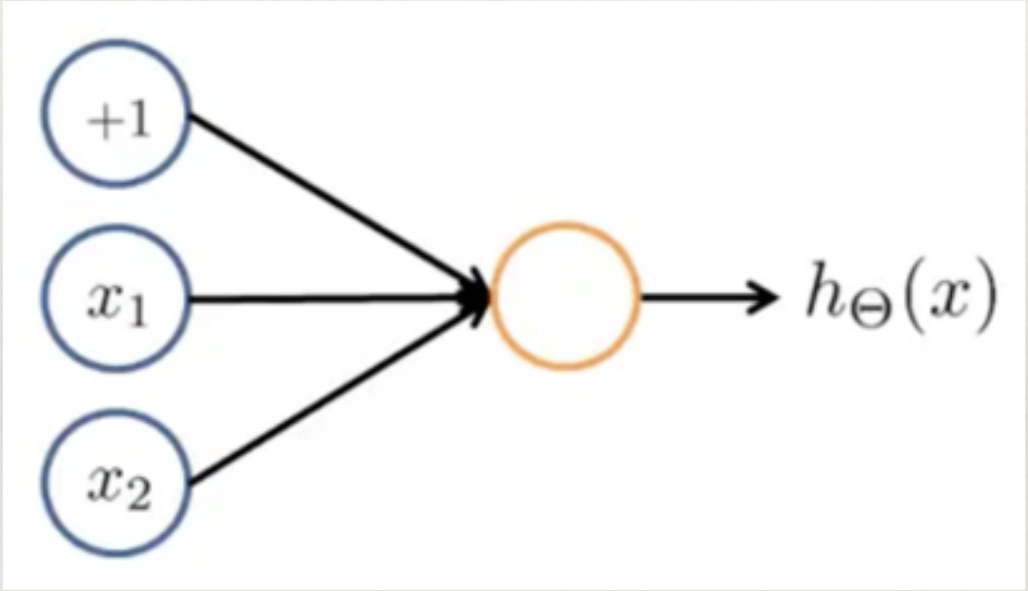
Intuitions - AND function



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

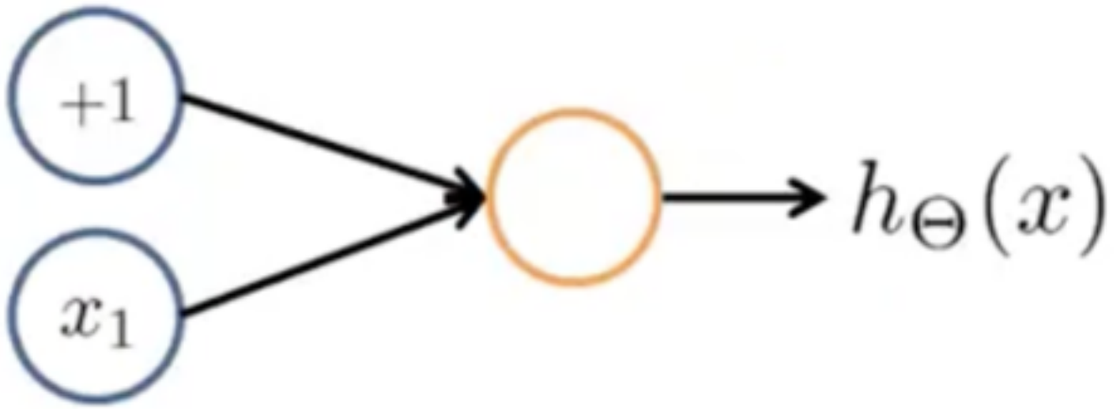


Intuitions - OR function



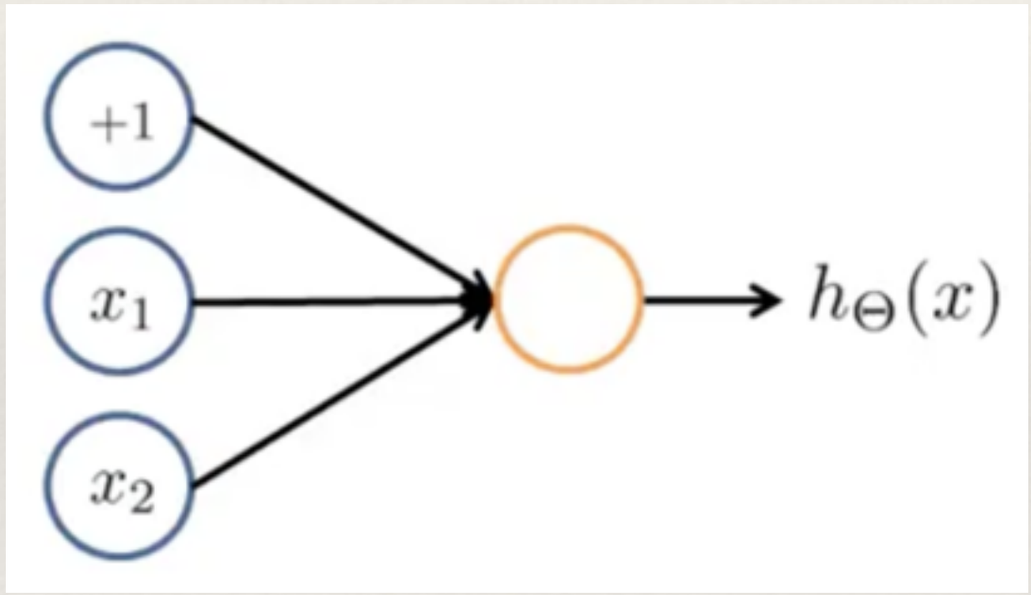
x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

Intuitions - NOT function



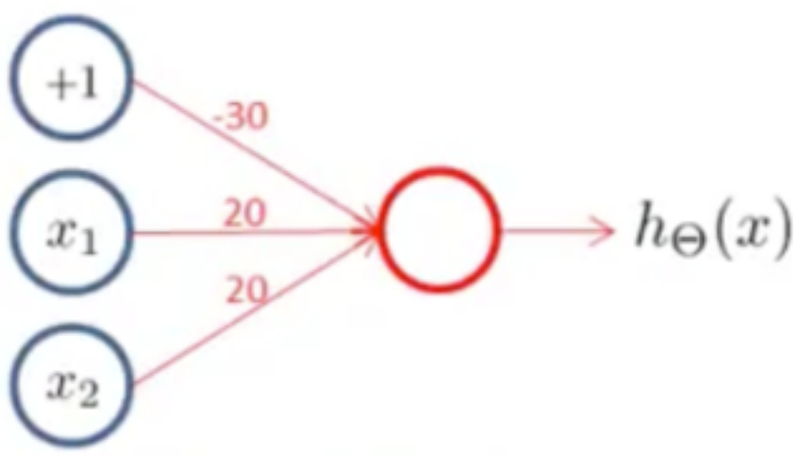
x_1	$h_\Theta(x)$
0	
1	

Intuitions - NOT + AND

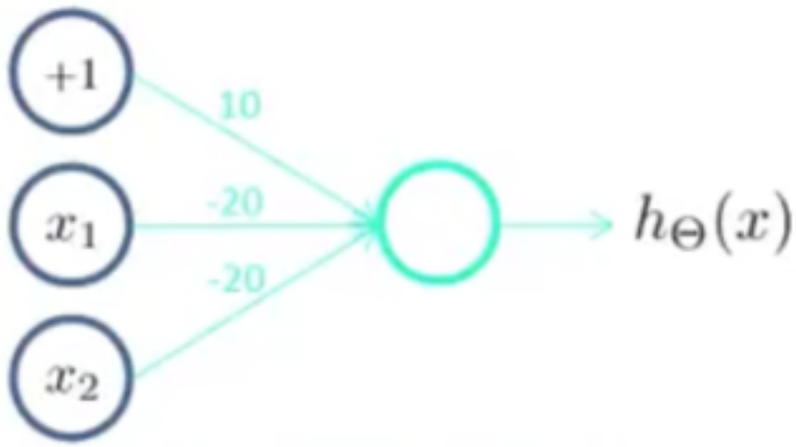


x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

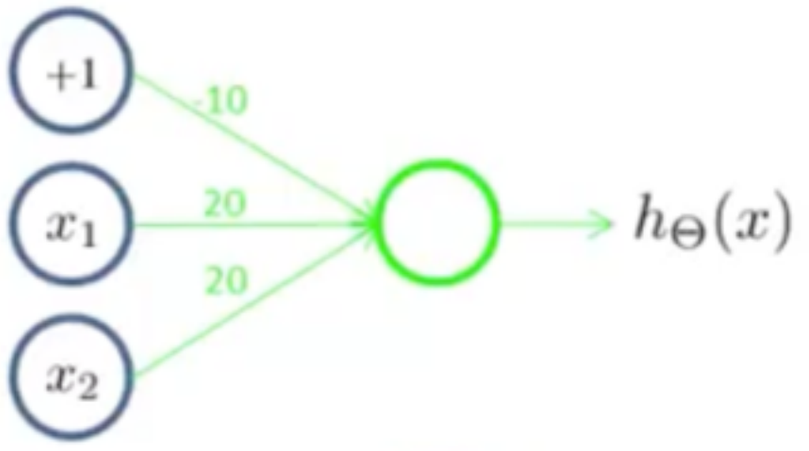
Putting it together: XNOR



x_1 AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2

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

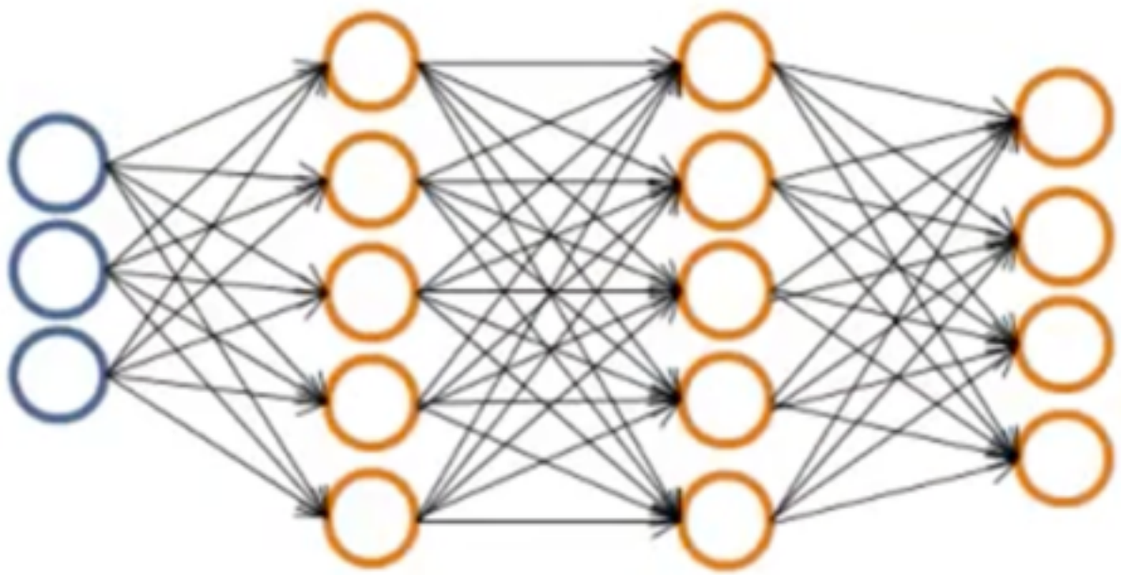
Multi-class classification

Pedestrian

Car

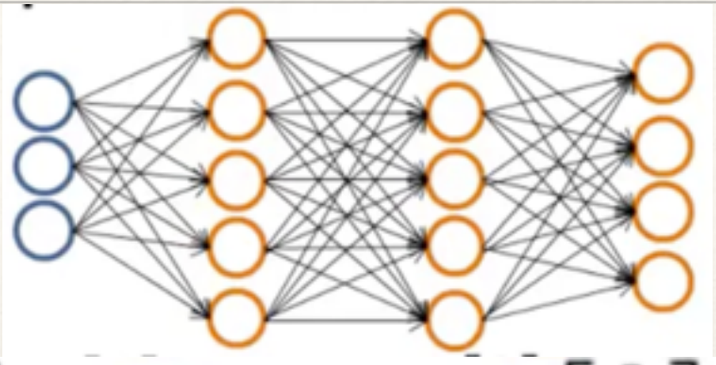
Motorcycle

Truck



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

Multi-class representation



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.

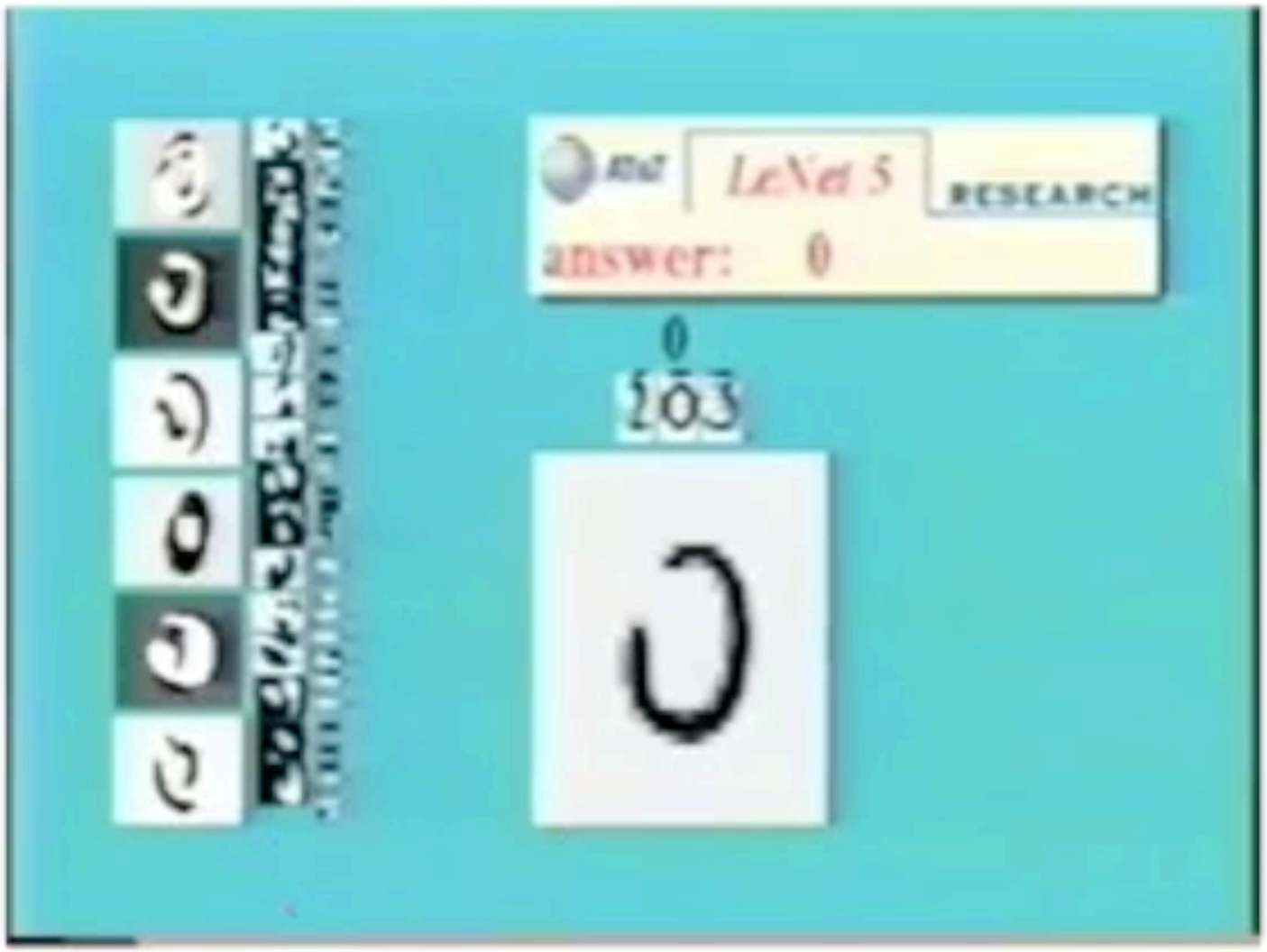
when pedestrian when car when motorcycle

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

Handwritten digit classification



[Courtesy of Yann LeCun]

Andrew Ng

Break!

Afterwards:

- Learning weights (backpropagation)
- Deep learning and its prerequisites

Backprop algorithm

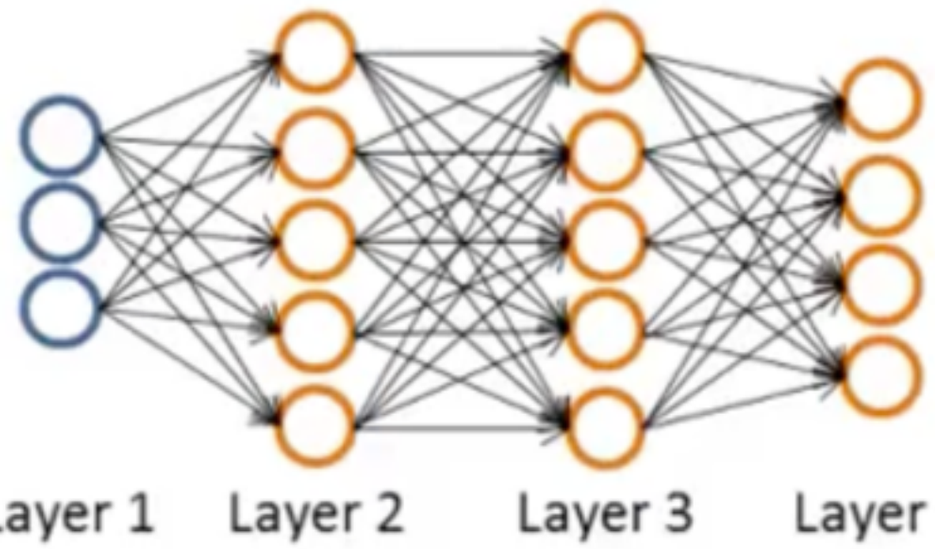
- Algorithm to ‘learn weights’
  - Actually, to calculate the derivative of the ‘error’ of a neural network…
- Intuition: calculate how *wrong* each neuron in each layer is

Before backprop: forward prop

Given one training example (x, y):

Forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$

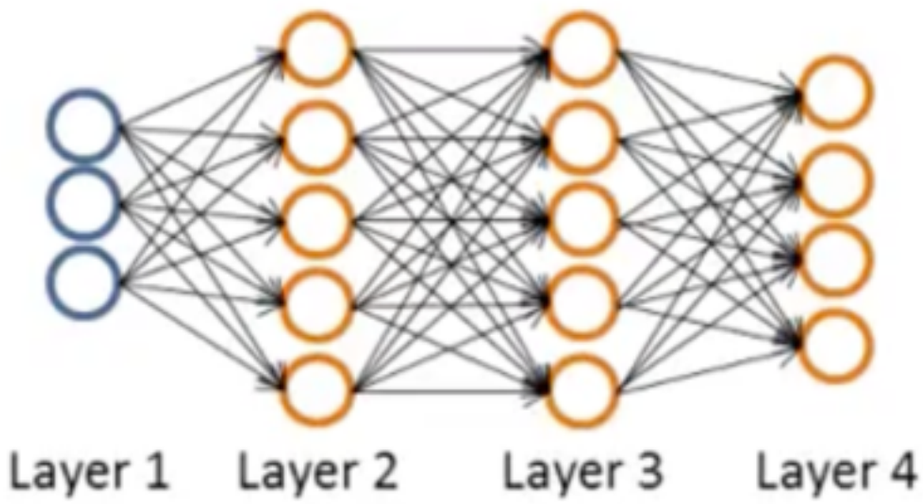


‘Error’ as compared to gold labels

Intuition: $\delta_j^{(l)}$ = “error” of node j in layer l.

For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$



Backprop algorithm

Calculate the error of each unit in each layer

1. Perform forward propagation to calculate activations
2. Using gold labels, calculate δ^{L}
3. Calculate δ^{L-1} etc.

Backprop – Step by step

Backprop – Step by step

- Weights update after each step
- A step is either:
  - One training example
  - One ‘batch’ of multiple training examples

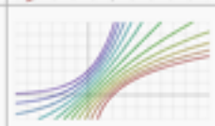
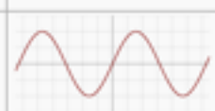
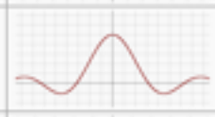
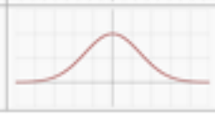
Weight initialisation

- Zero initialisation?
- Initialise all weights to 1?
- Random initialisation?

Dropout

- Weights are randomly ‘forgotten’
  - Probabilities per layer
- Increases the robustness of a neural network
- A form of regularisation

Activation functions

Name	Plot	Equation	Derivative	Monotonic	$f(x) \approx x$ when $x \approx 0$	Range
Identity		$f(x) = x$	$f'(x) = 1$	Yes	Yes	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	Yes	No	$\{0, 1\}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	Yes	No	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	Yes	Yes	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	Yes	Yes	$(-\frac{\pi}{2}, \frac{\pi}{2})$
Rectified Linear		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	Yes	No	$[0, \infty)$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	Yes	No	$(0, \infty)$
Bent identity		$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2 + 1}} + 1$	Yes	Yes	$(-\infty, \infty)$
SoftExponential		$f(\alpha, x) = \begin{cases} -\frac{\log_e(1 - \alpha(x + \alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x} - 1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1 - \alpha(\alpha + x)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$	Yes	Yes iff $\alpha \approx 0$	$(-\infty, \infty)$
Sinusoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$	No	Yes	$[-1, 1]$
Sinc		$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$	No	No	$[\approx -.217234, 1]$
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$	No	No	$(0, 1]$

https://en.wikipedia.org/wiki/Activation_function

Activation functions

Traditional:

- Sigmoid, Tangent

Modern (Deep Learning!):

- Rectifier (aka. ReLU — Rectified Linear Unit)
- Maxout
  - Learns its own activation function per layer!

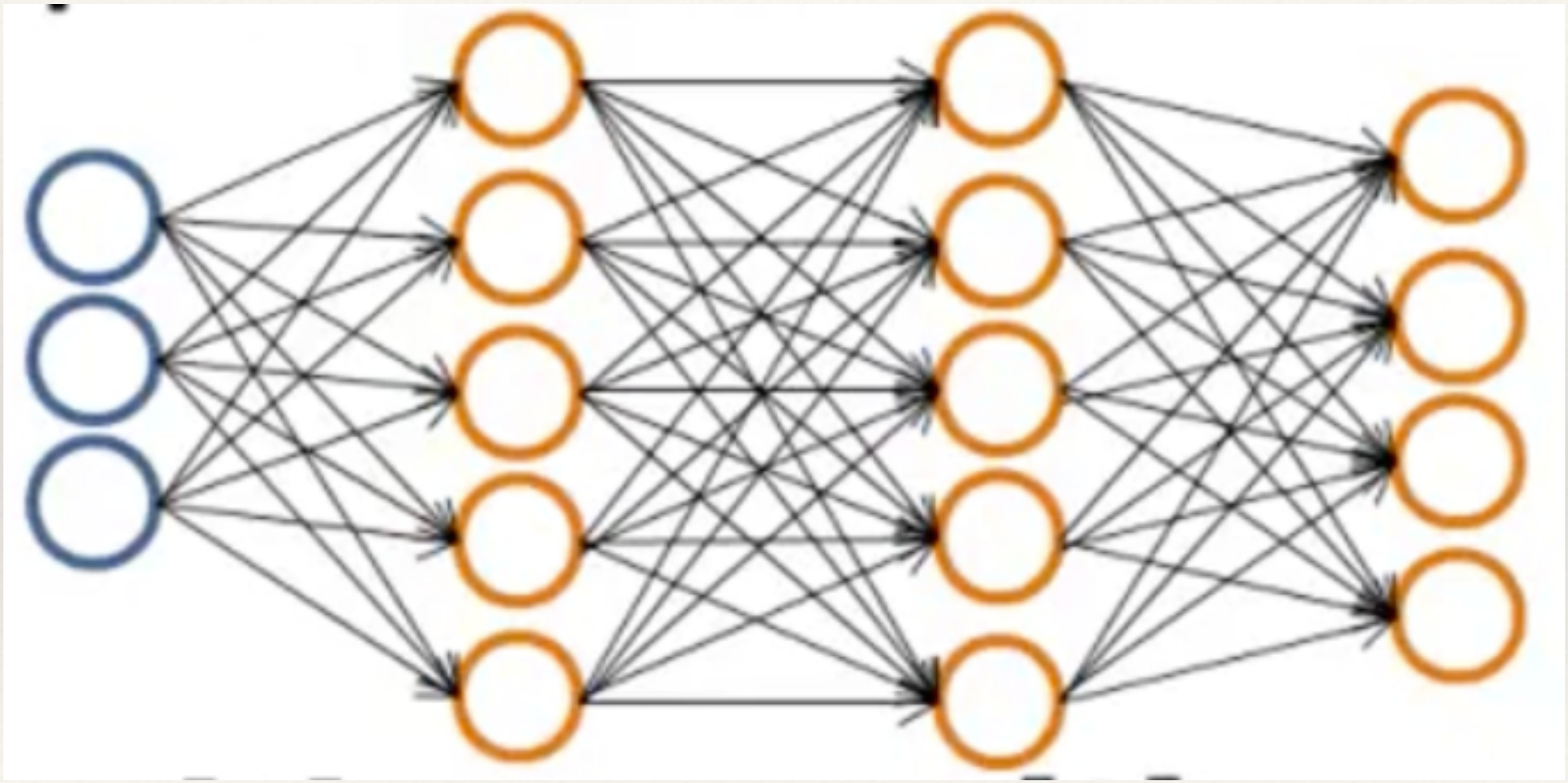
"I've worked all my life in Machine Learning, and I've never seen one algorithm knock over benchmarks like Deep Learning"

Andrew Ng

Deep Learning

- Resurgence of Neural Network popularity!
- Catchy name…
- But what is it?

Deep Learning



Deep Learning

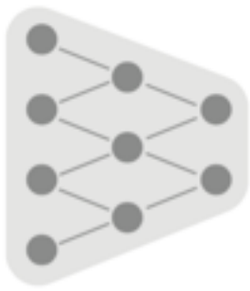
Why now?

- More computing power
- Better ways of training neural nets
  - Dropout
  - Rectifiers (ReLU)

Deep Learning

Why?

- Each layer is another level of abstraction
- Much more complex problems can be handled
  - Image recognition
  - Facial recognition
  - From images to text



A group of people shopping at an outdoor market.

There are many vegetables at the fruit stand.

A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

Figure 3 | From image to text. Captions generated by a recurrent neural network (RNN) taking, as extra input, the representation extracted by a deep convolution neural network (CNN) from a test image, with the RNN trained to 'translate' high-level representations of images into captions (top). Reproduced with permission from ref. 102. When the RNN is given the ability to focus its attention on a different location in the input image (middle and bottom; the lighter patches were given more attention) as it generates each word (bold), we found[86] that it exploits this to achieve better 'translation' of images into captions.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

Putting it together

- Picking architecture:
  - Number of layers, units per layer
  - Activation function
  - Input units?
  - Output units?

Assignment preparation

- Install Keras (http://keras.io/#installation)
  - May not be possible on Windows — Use LWP computers
- Neural Network videos by Andrew Ng on Coursera
  - https://www.coursera.org/learn/machine-learning/home/week/4 (especially 'Model Representation')
  - https://www.coursera.org/learn/machine-learning/home/week/5 (especially Backprop)

Assignment

Two main parts:

- Theory
- Practice — Competition!
  - Noun compound classification
  - Best performing system wins a prize!