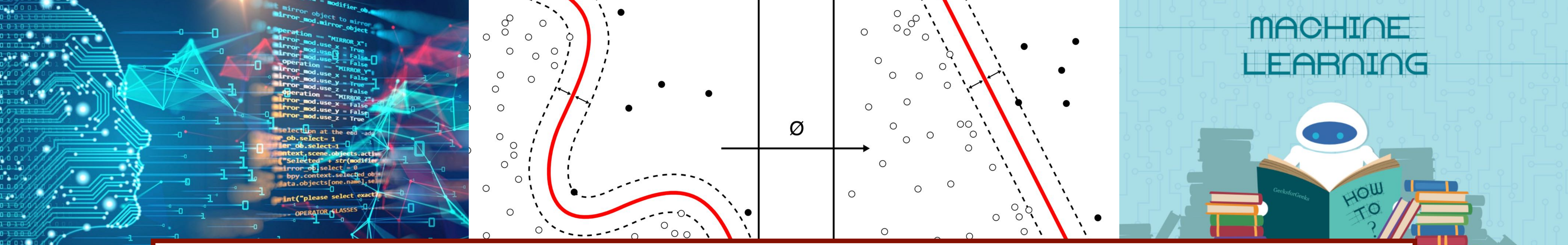


Introduction to Machine Learning

SCP8084699 - LT Informatica

Evaluation, Learning Curves & Babysitting

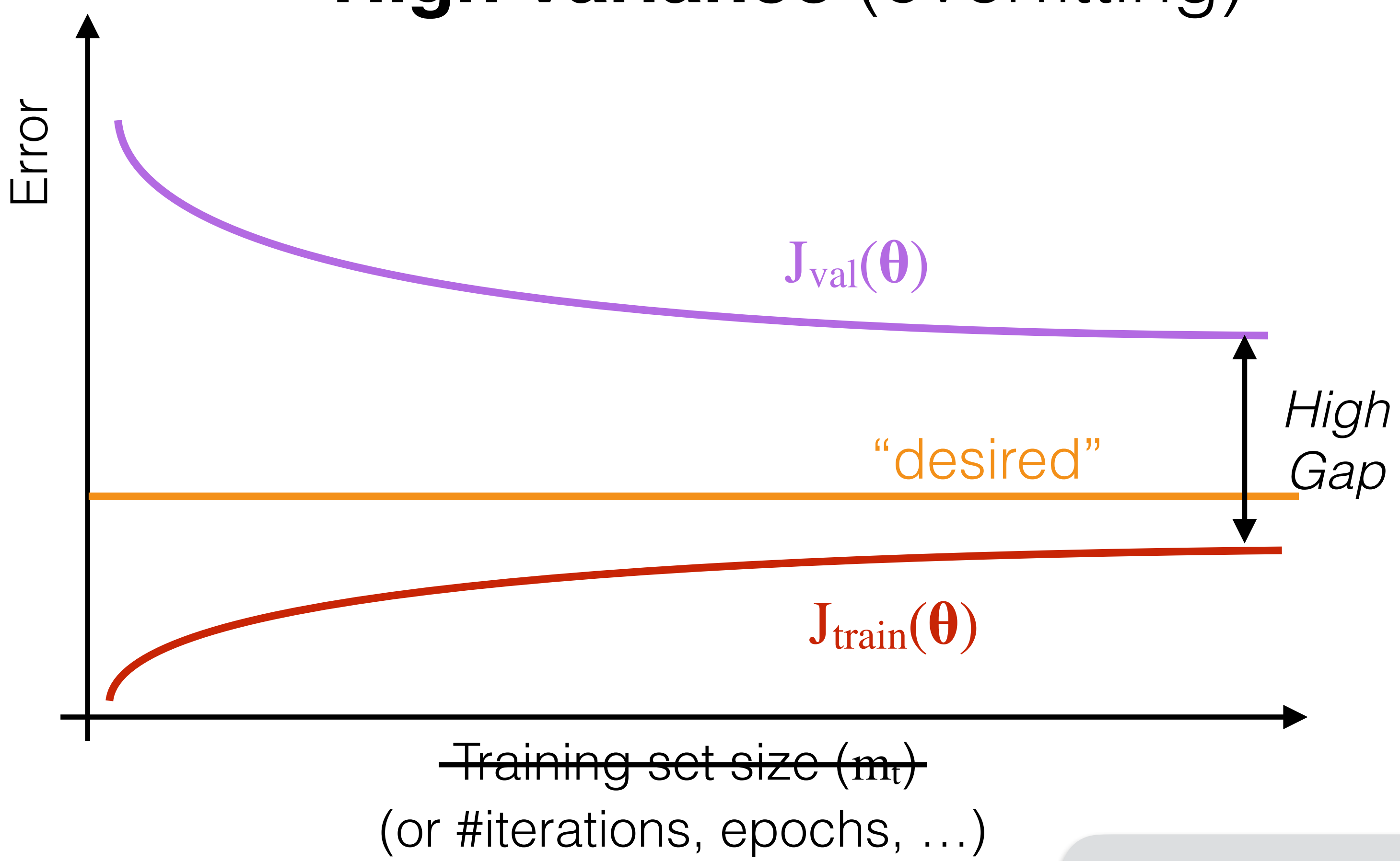
Prof. Lamberto Ballan

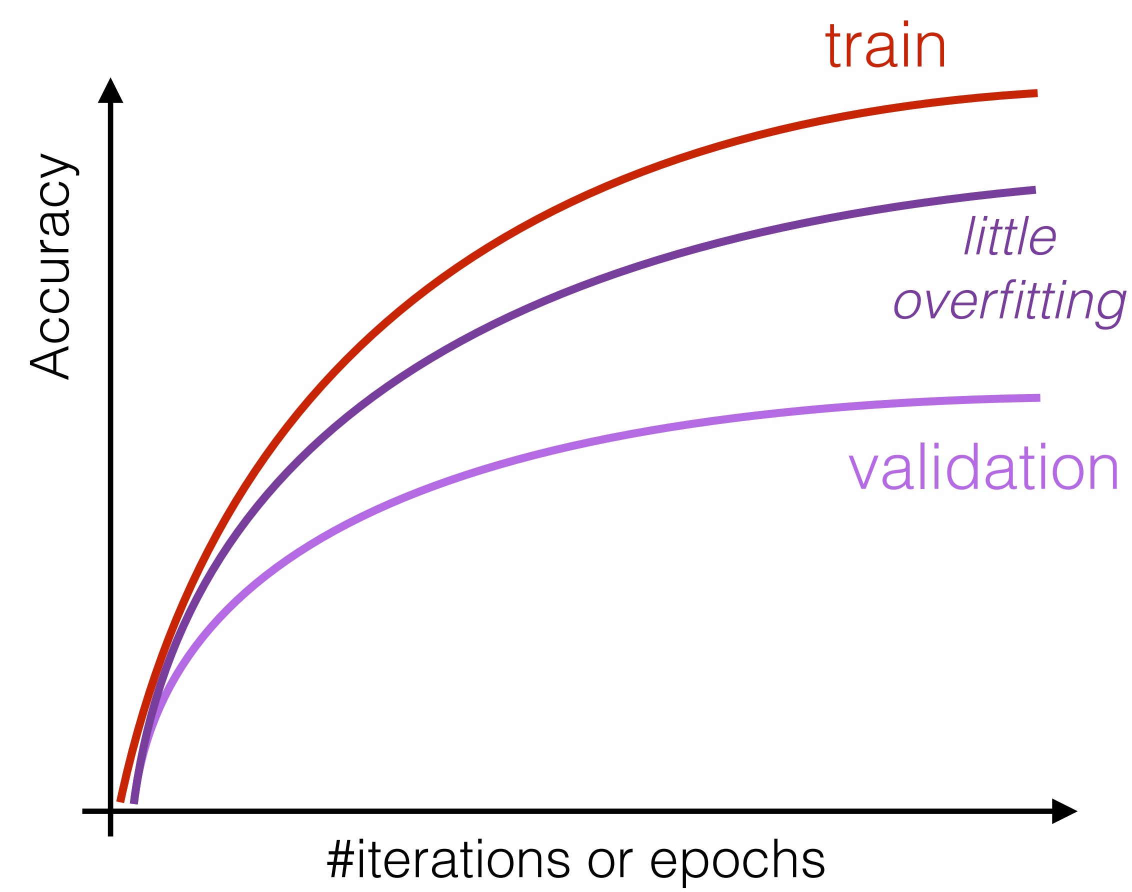
DIPARTIMENTO
MATEMATICA
Department of Mathematics "Tullio Levi-Civita"

Recap: Learning Curves

- You can compute learning curves w.r.t. different "dimensions" (e.g. evaluation measures, no. samples)

High variance (overfitting)



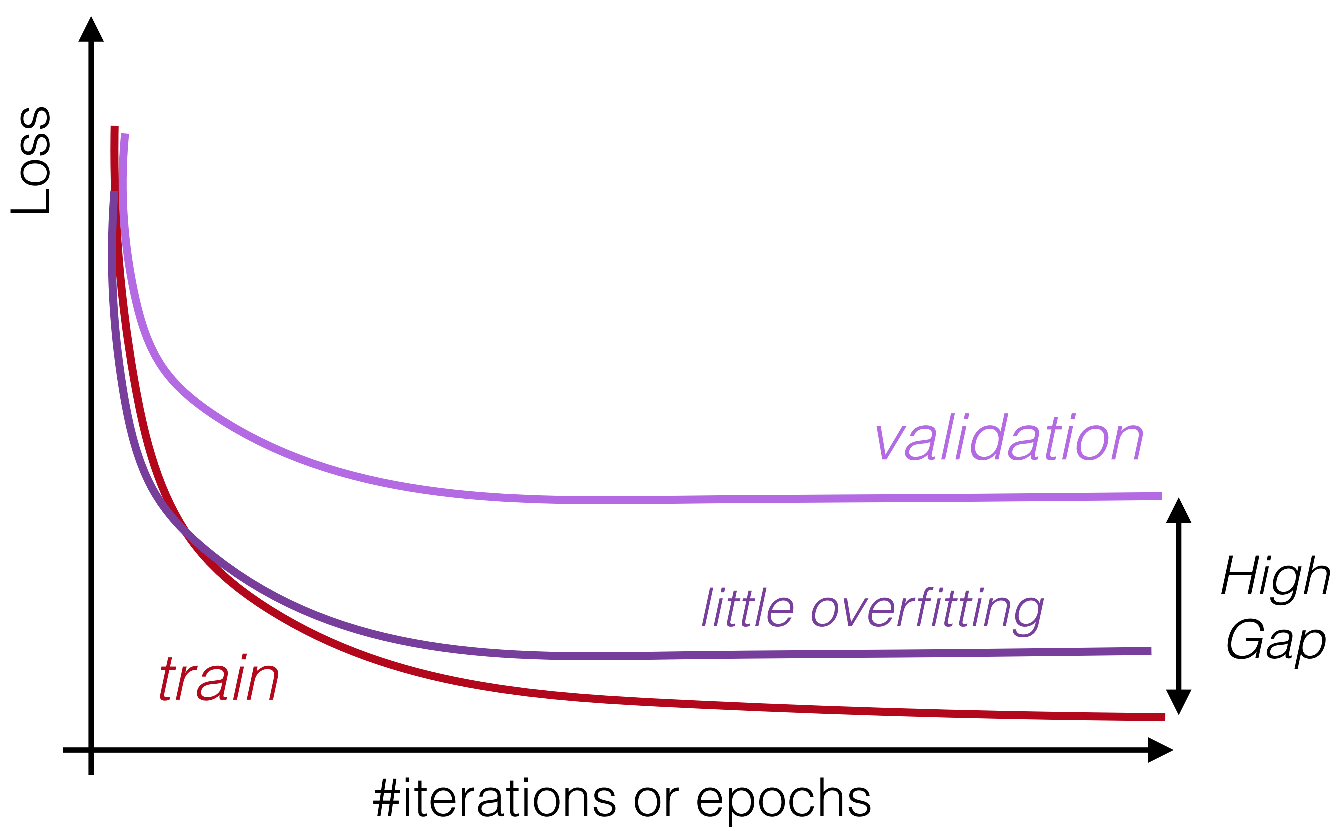


epoch = a pass of the entire set of examples

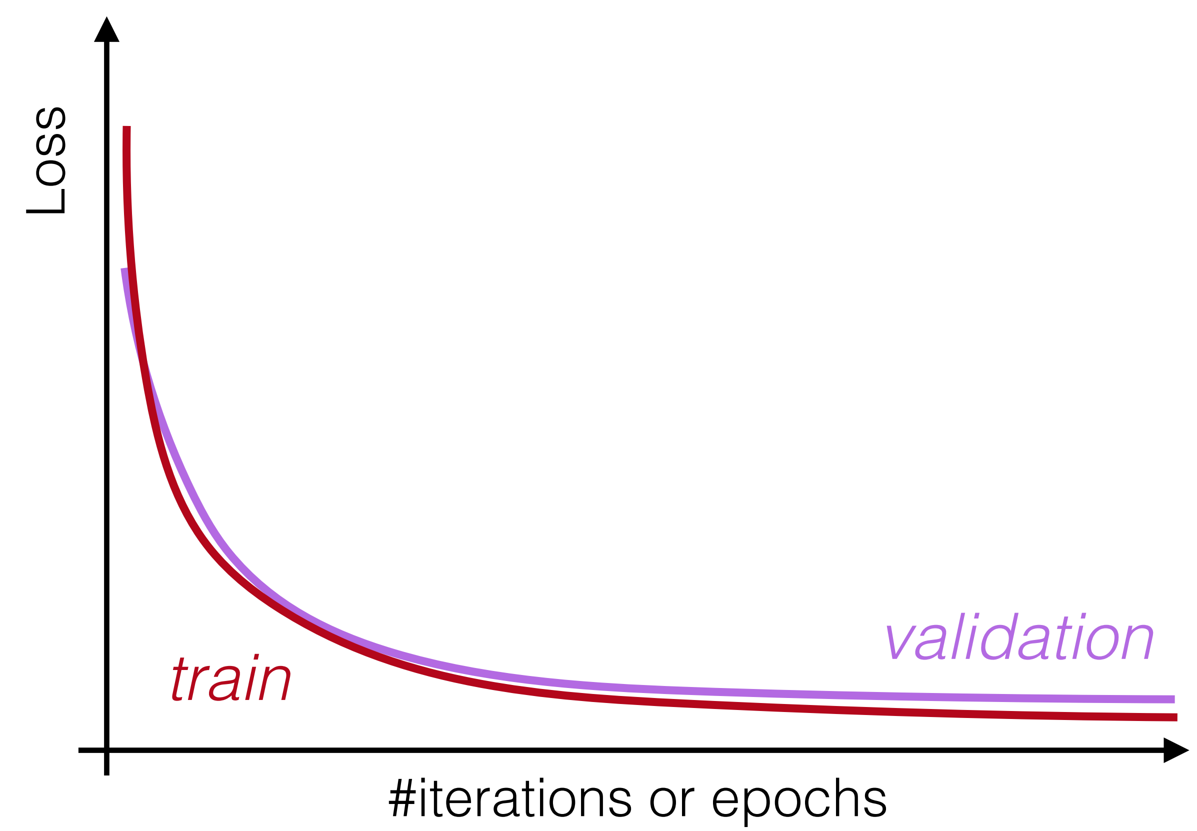
Learning Curves

- Often learning curves are plotted w.r.t. the loss

(overfitting)



(great fit)



What to do next

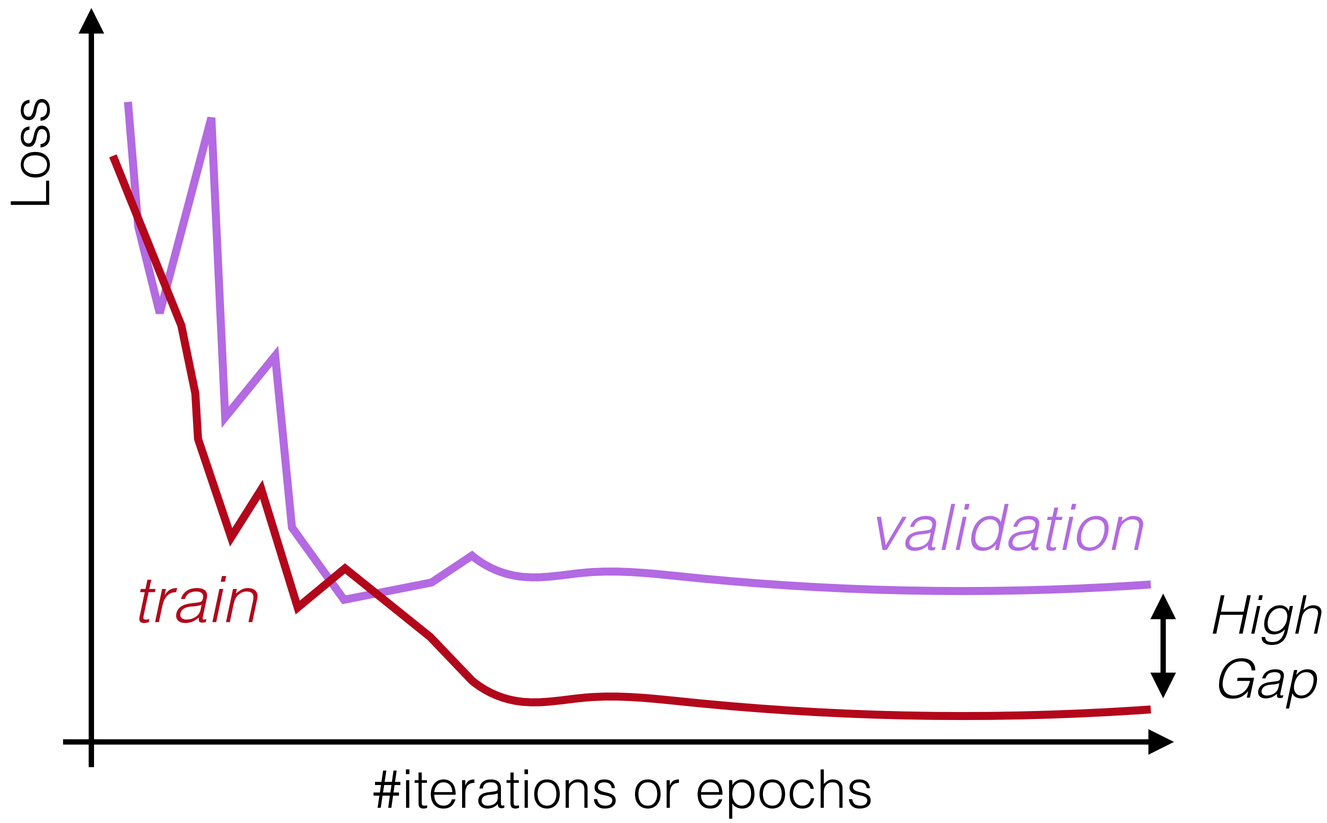
- Debugging (and babysitting) a learning algorithm:
  - Suppose you have implemented a regularized linear regression model for predicting housing prices
  - It doesn't work on new data; what should you do next?
    - You can get more training data → *Fixes high variance*
    - Try smaller set of features → *Fixes high variance*
    - Try getting more features → *Fixes high bias*
    - Try adding complexity to the model (e.g. polynomial features) → *Fixes high bias*
    - Try decreasing λ → *Fixes high bias*
    - Try increasing λ → *Fixes high variance*

Adapted from slide by A. Ng

Diagnosing our datasets

- Learning curves can be also used to diagnose the quality of our training/validation sets

Unrepresentative Training Set

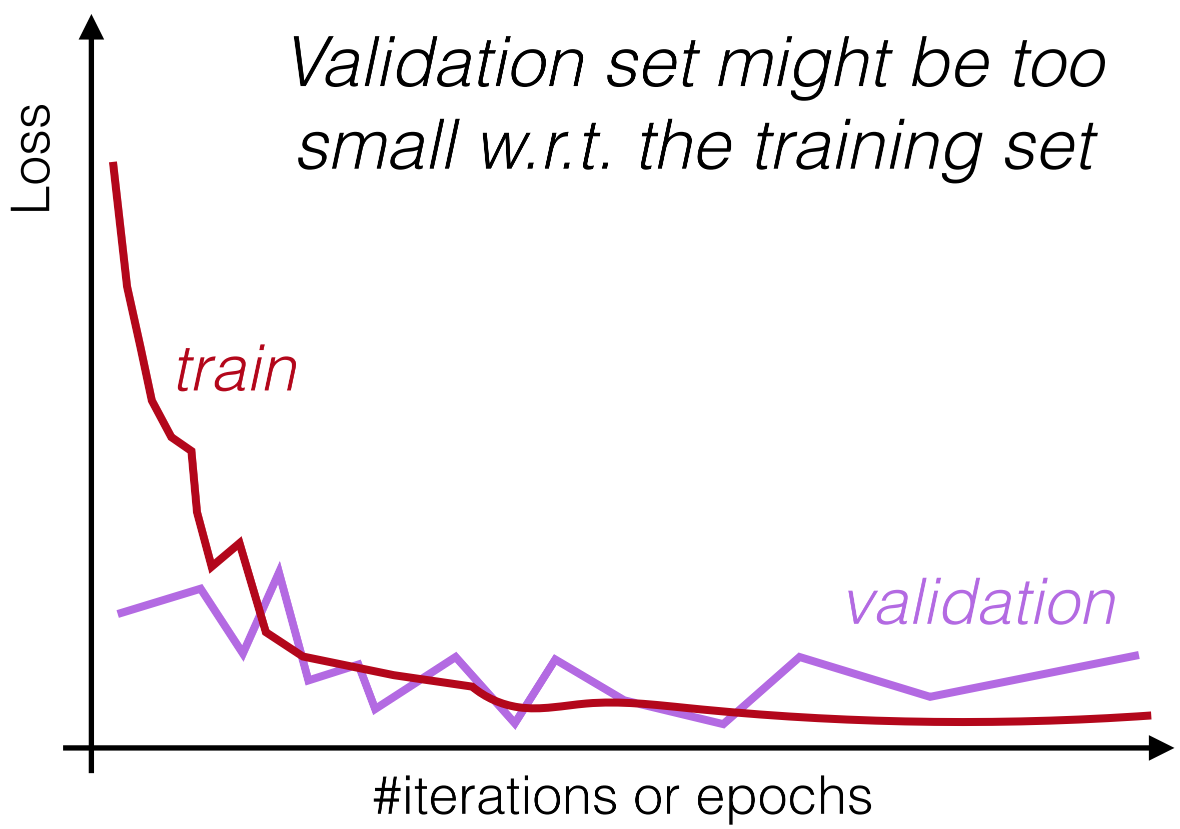


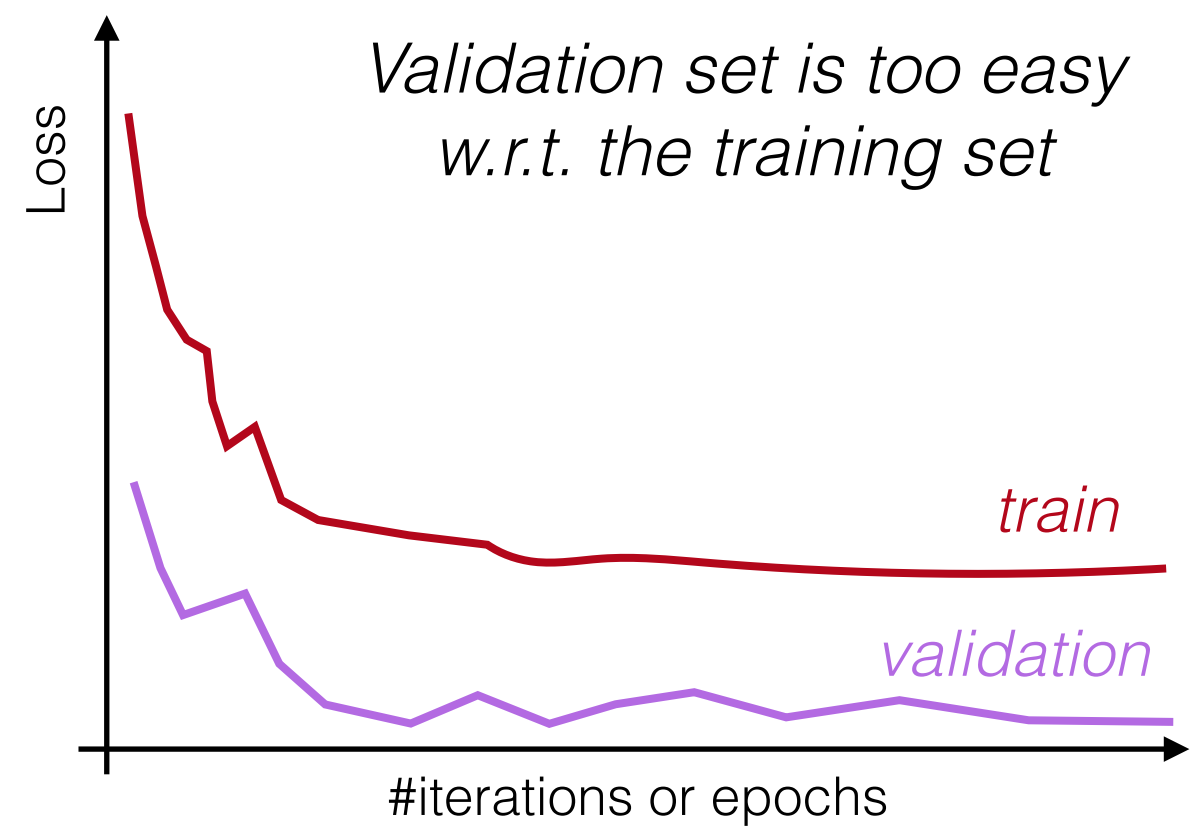
- The training set does not provide sufficient information to learn the problem
- It may occur if the training set has too few examples as compared to the validation set

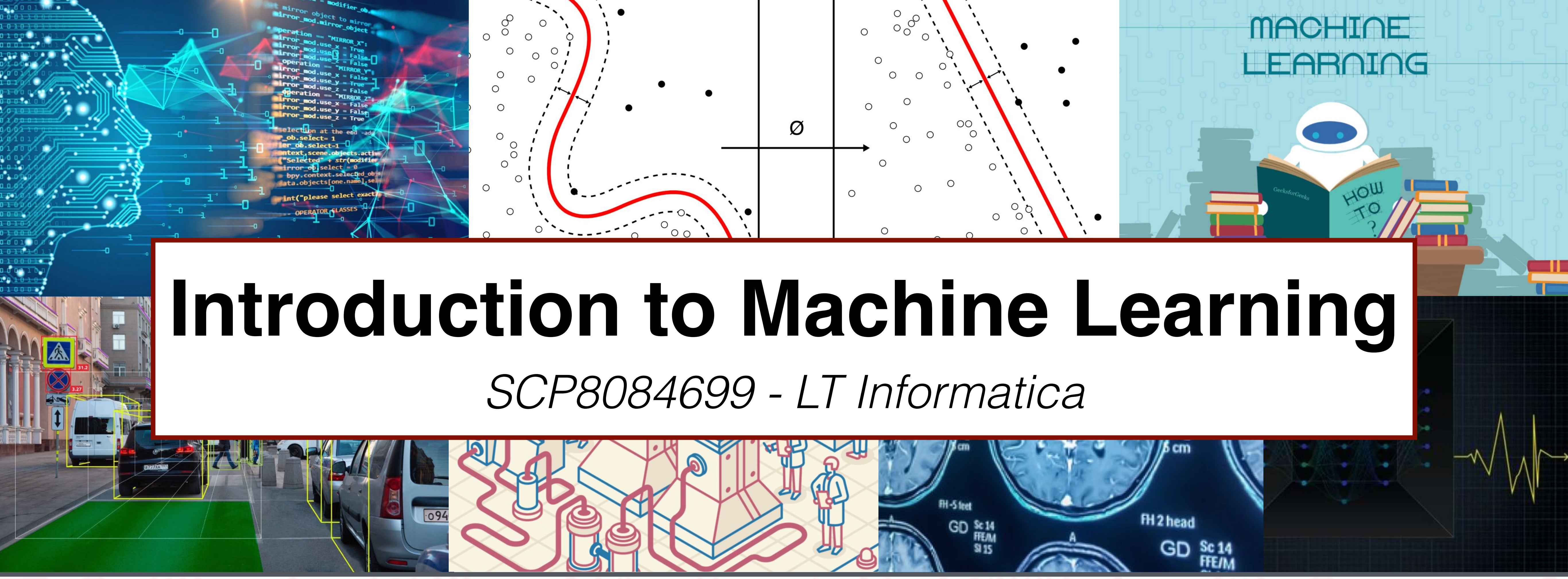
Diagnosing our datasets

- Learning curves can be also used to diagnose the quality of our training/validation sets

Unrepresentative Validation Set







Introduction to Machine Learning

SCP8084699 - LT Informatica

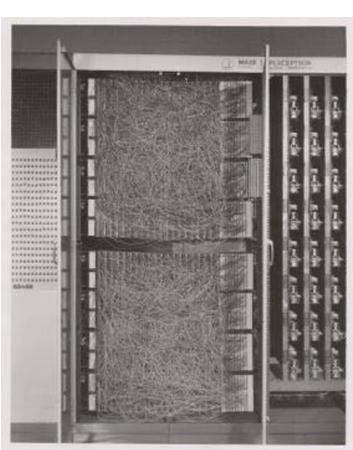
Artificial Neural Networks I

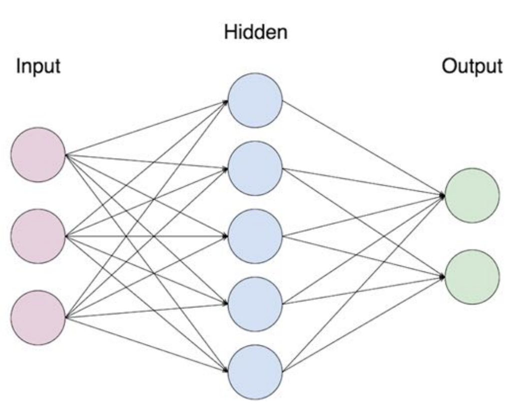
Prof. Lamberto Ballan

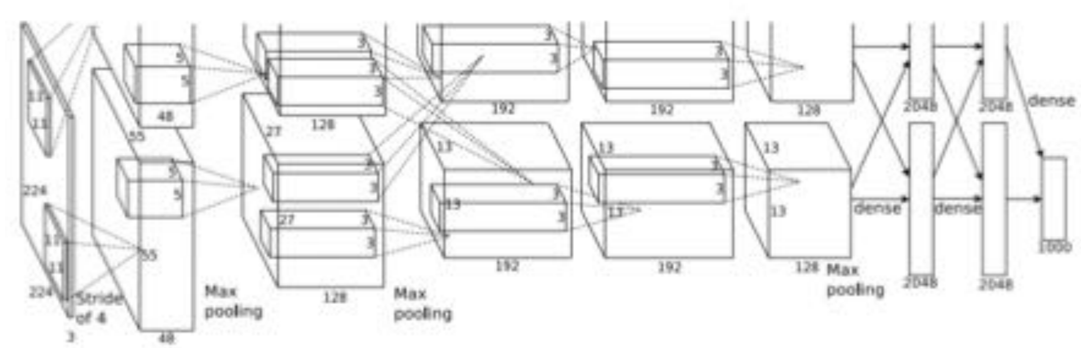
DIPARTIMENTO
MATEMATICA
Department of Mathematics "Tullio Levi-Civita"

Neural Networks: origins

- Algorithms that are modelled (loosely) after the brain
  - Neural networks have been introduced in late 1950s
  - They were widely used in 1980s and early 1990s; their popularity largely diminished in late 1990s
  - Recent “resurgence”: late 2000s/early 2010s (deep learning revolution)
  - Artificial neural networks are not nearly as complex or intricate as the actual brain structure

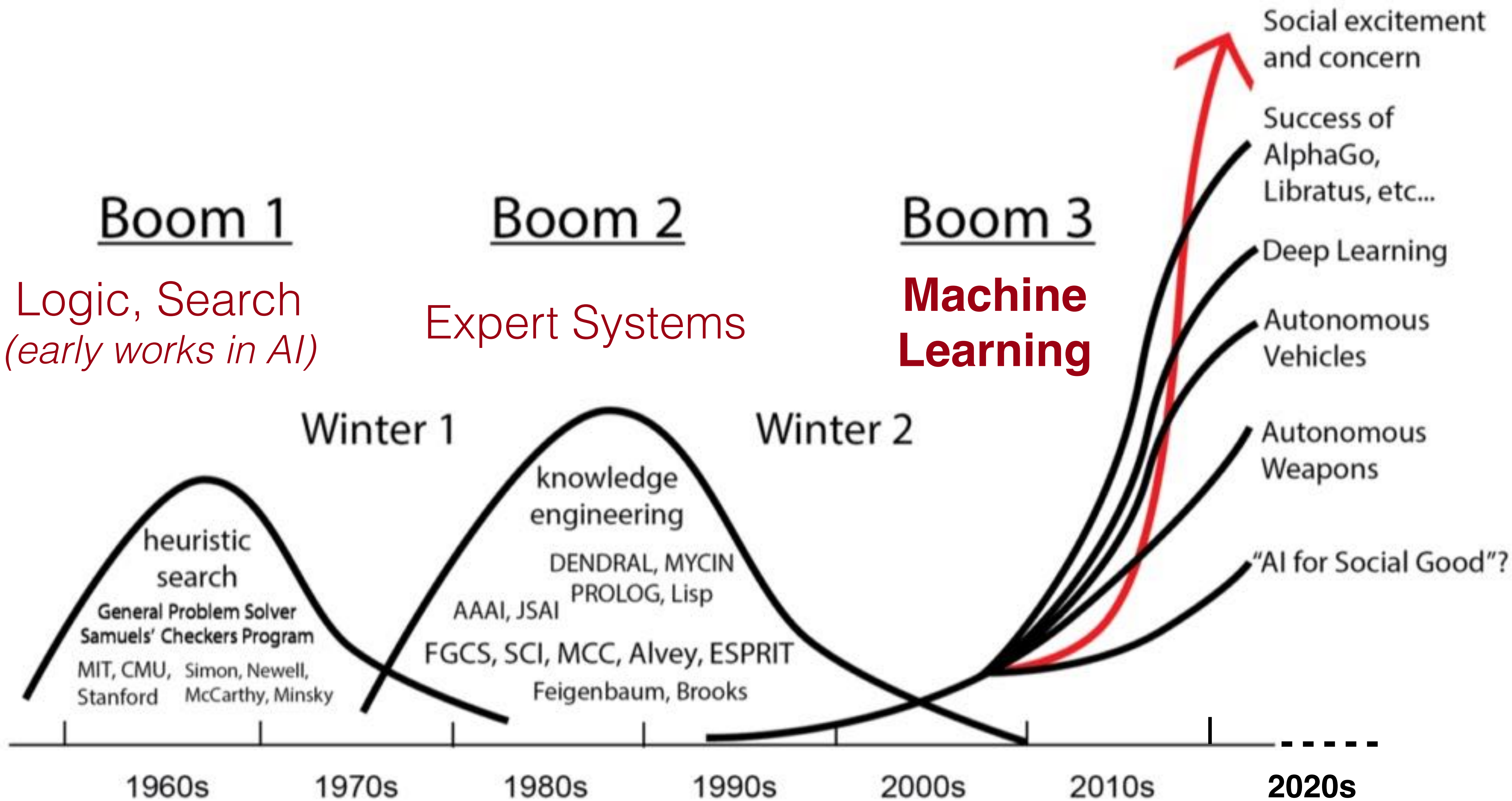






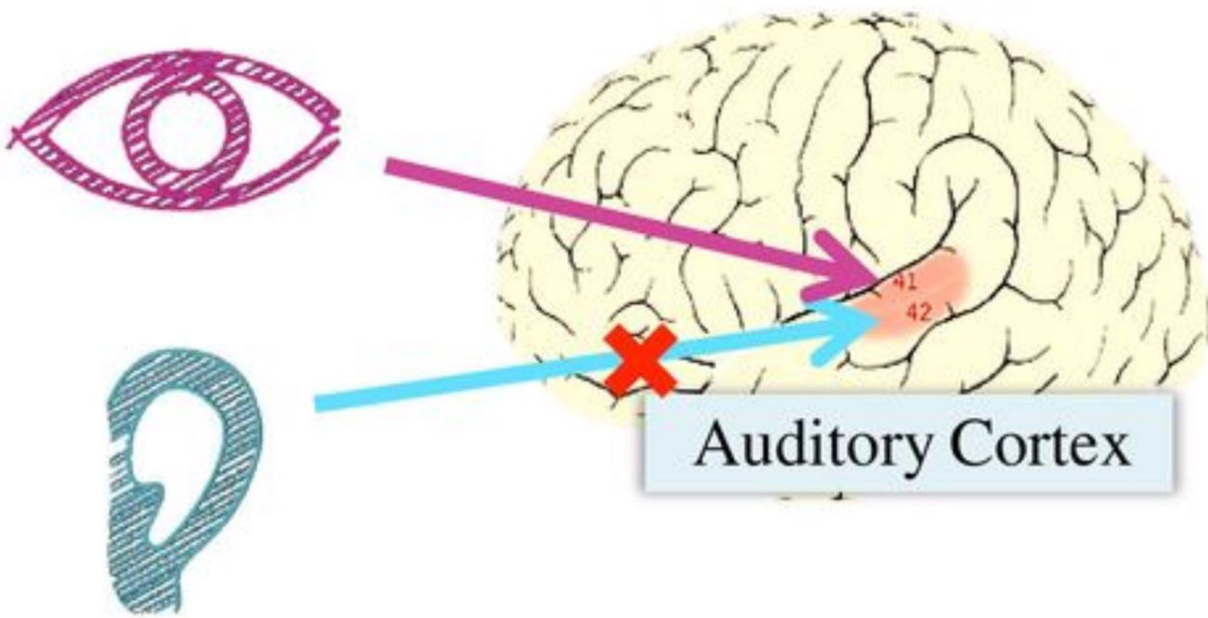
Neural Networks: origins

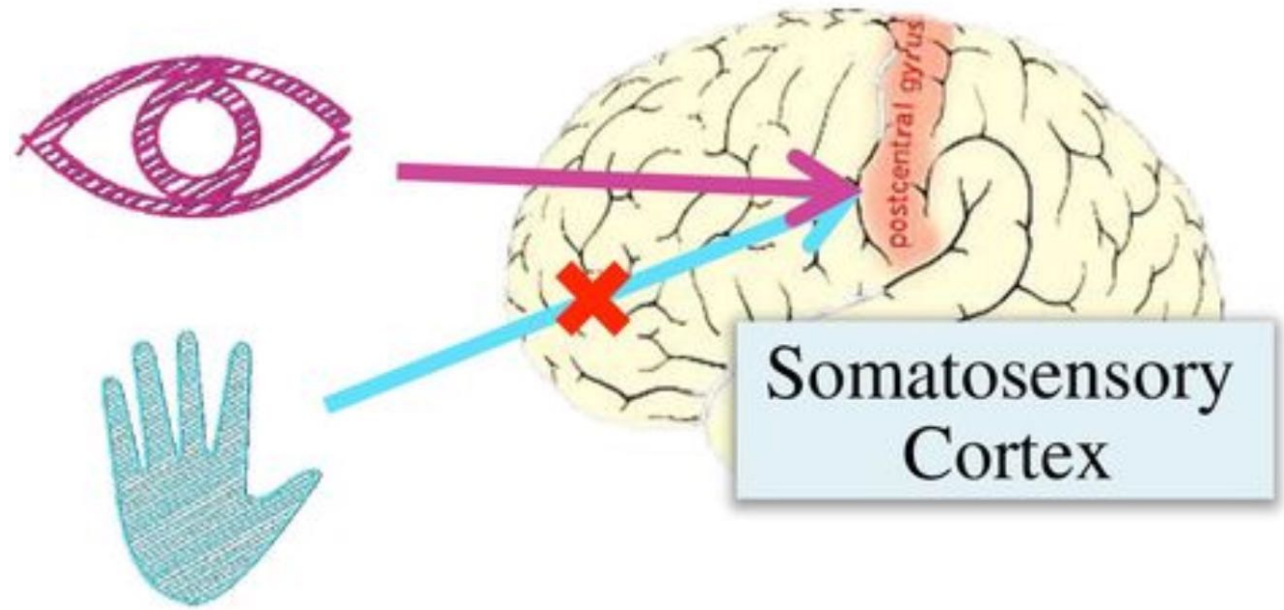
- Neural Networks and AI winters



Neural Networks: origins

- The “one learning algorithm” hypothesis





Auditory cortex learns to see

[Roe et al., 1992]

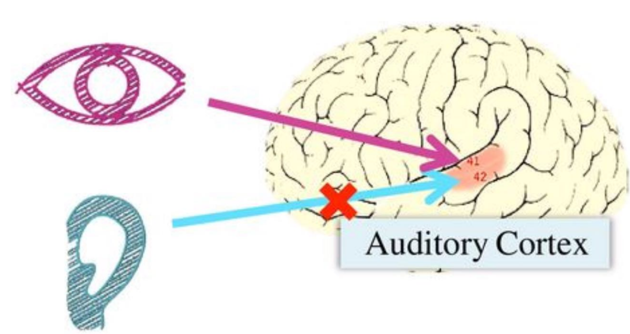
Somatosensory cortex learns to see

[Metin and Frost, 1989]

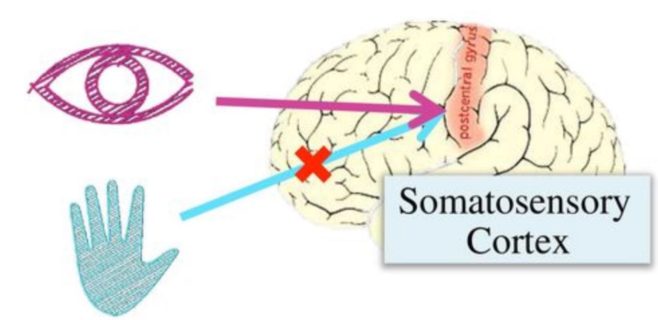
Adapted from slide by A. Ng

Neural Networks: origins

- The "one learning algorithm" hypothesis
  - Because of these experiments there is this sense that if the same part of the brain can process sight or sound or touch, then (maybe) there is *one learning algorithm*
  - Therefore, instead of designing hundreds of different algorithms, we need to approximate this one algorithm



[Roe et al., 1992]



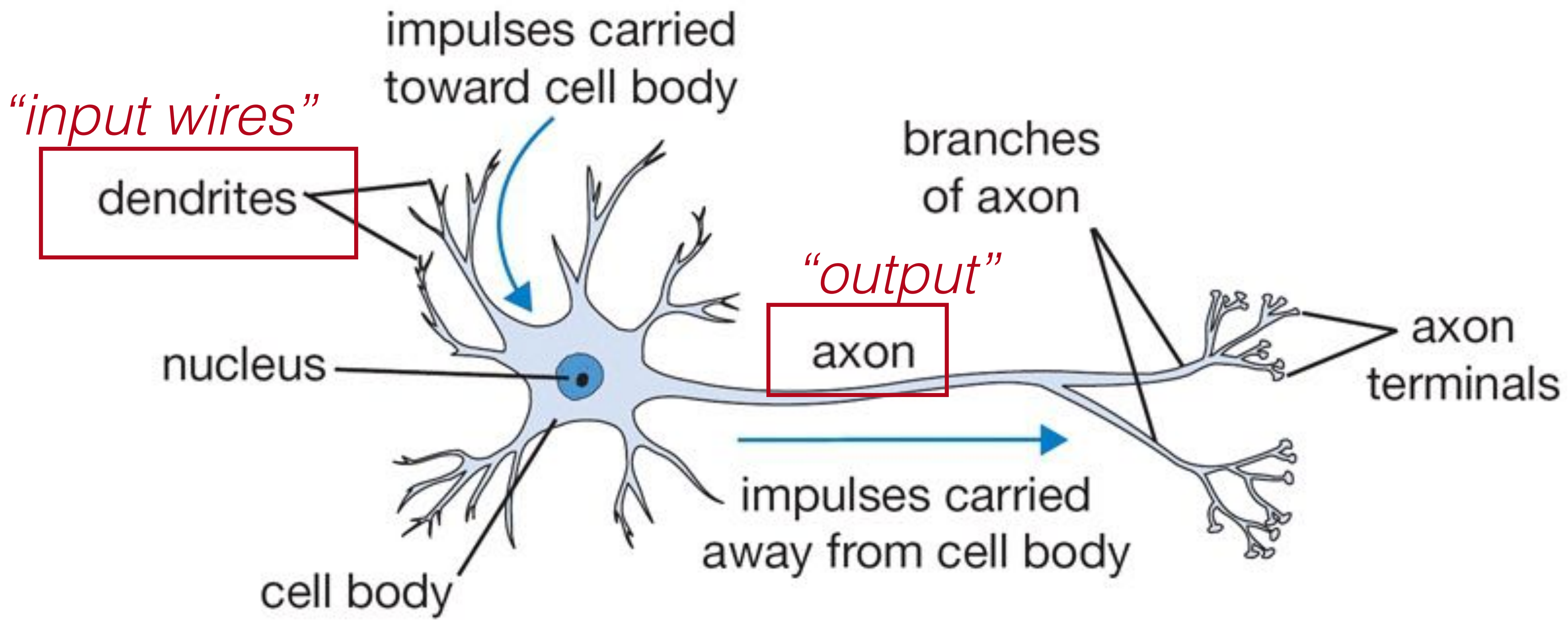
[Metin and Frost, 1989]

Neural Networks: applications

- Neural networks are everywhere…

Neural Networks

- Let’s start by looking at what a single neuron in the brain looks like

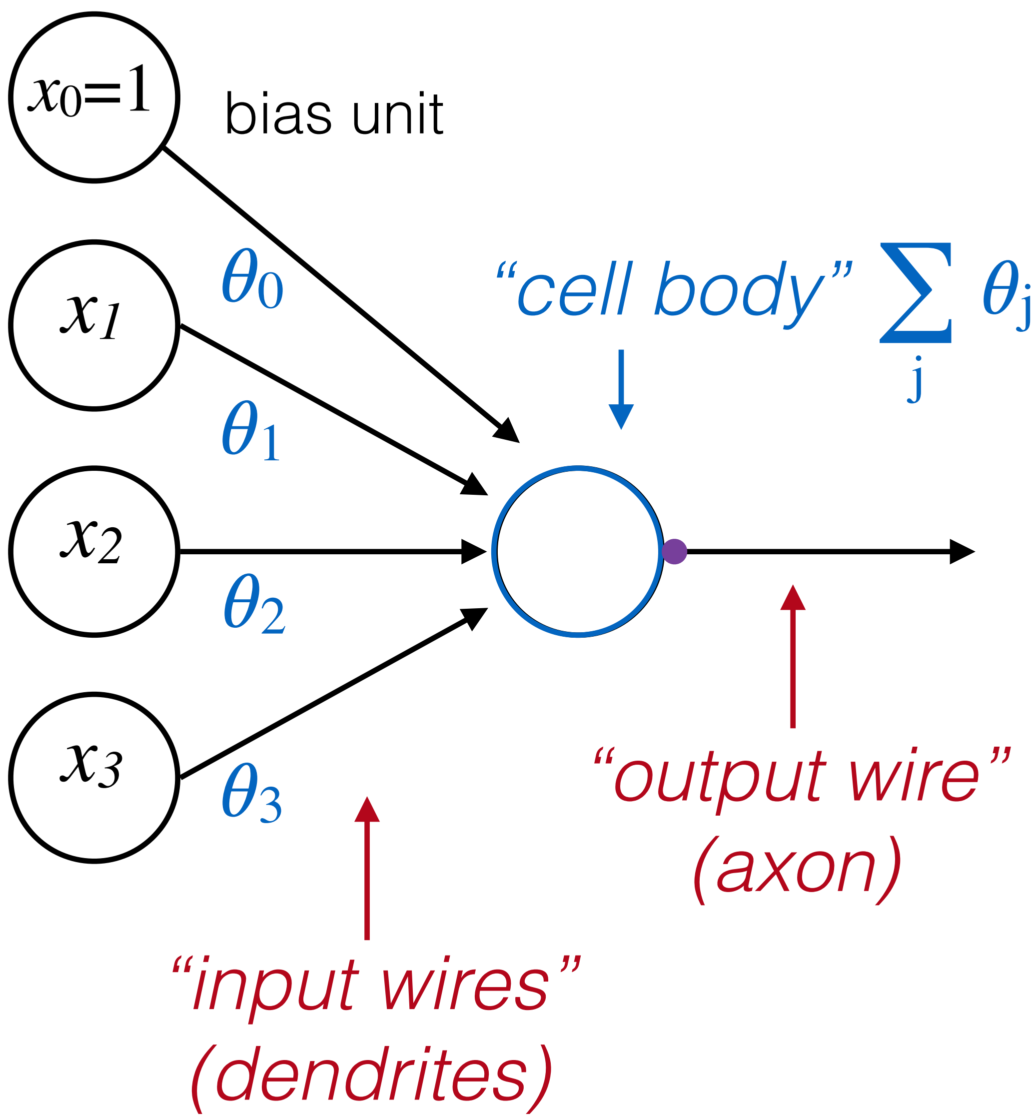


Artificial Neural Networks

- We define a simple model for an artificial neuron (computational unit)

$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix}$$

Parameters ("weights")



$$h_\theta(\mathbf{x}) = f\,(\boldsymbol{\theta}^T\,\mathbf{x})$$

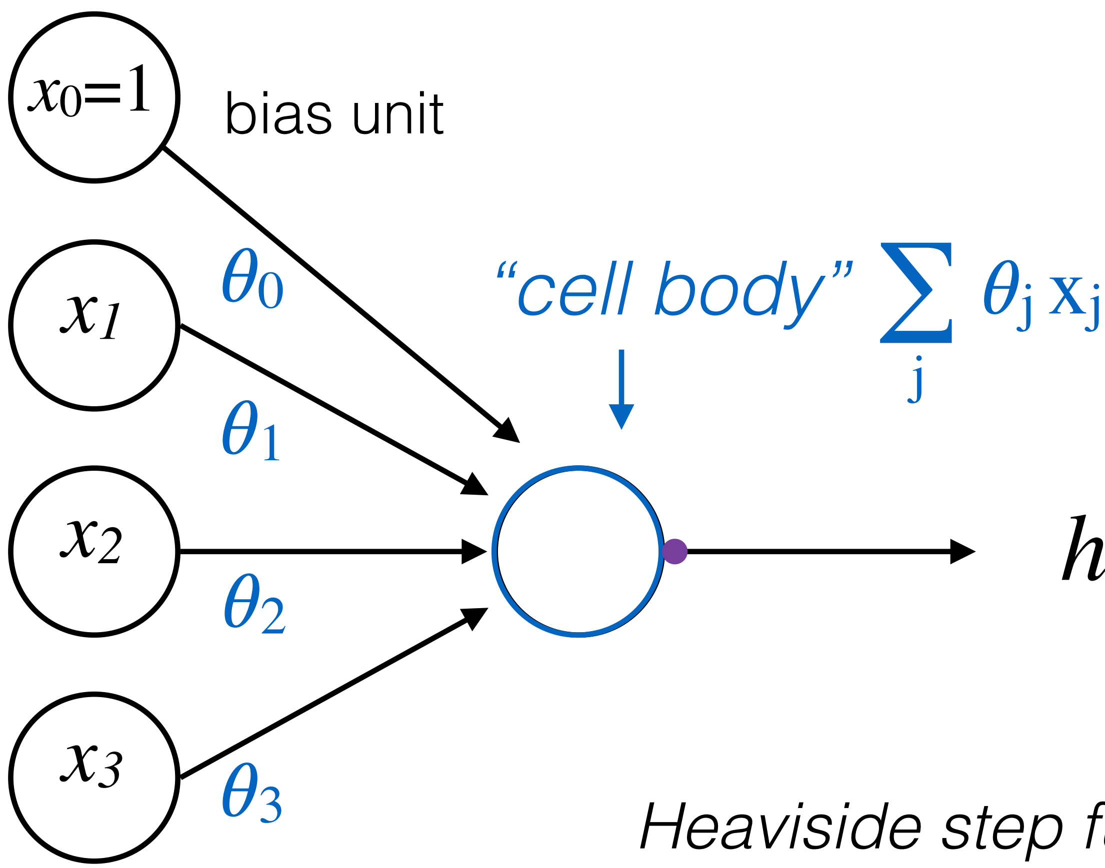
activation function f

Artificial Neural Networks

- We define a simple model for an artificial neuron (computational unit)

$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix}$$

Parameters (“weights”)



$$h_\theta(\mathbf{x}) = f\,(\boldsymbol{\theta}^T \mathbf{x})$$

Heaviside step function

$$h_\theta(\mathbf{x}) = \begin{cases} 1 & \text{if } \boldsymbol{\theta}^T \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron

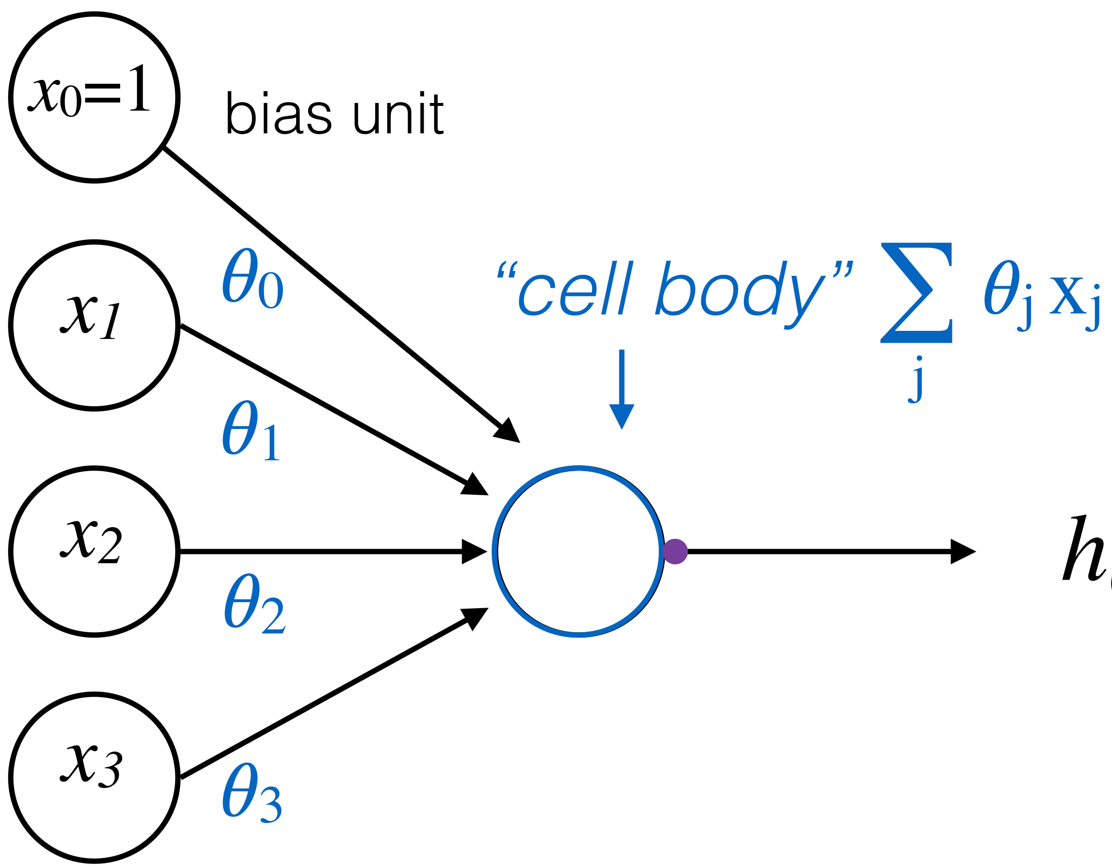
F. Rosenblatt, 1958

Artificial Neural Networks

- We define a simple model for an artificial neuron (computational unit)

$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix}$$

Parameters (“weights”)



$$h_\theta(\mathbf{x}) = f\,(\boldsymbol{\theta}^T\,\mathbf{x})$$

Logistic Unit

$$f\,(z) = \sigma\,(z) = \frac{1}{1 + e^{-z}}$$

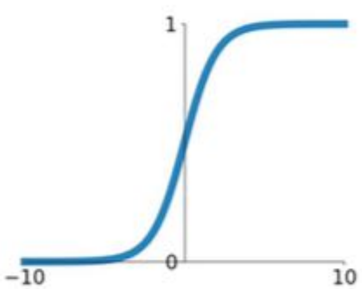
“activation function”
“non-linearity”

Neural Networks: activation functions

- Nowadays there are several activation functions you may encounter:

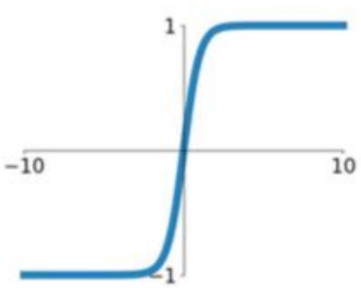
Sigmoid

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



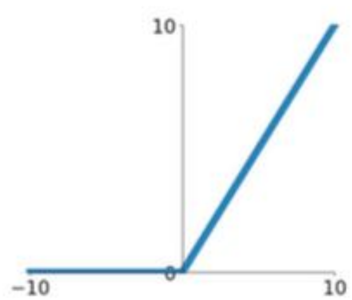
tanh

$\tanh(x)$



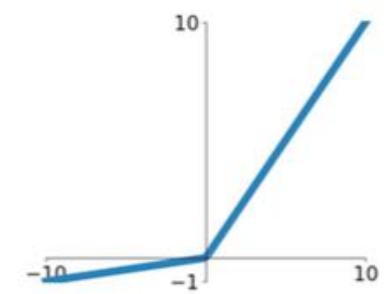
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

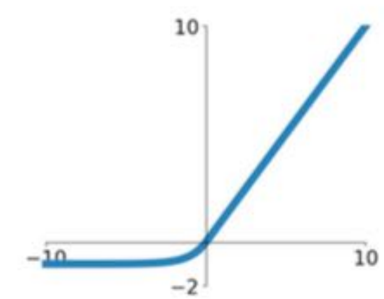


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

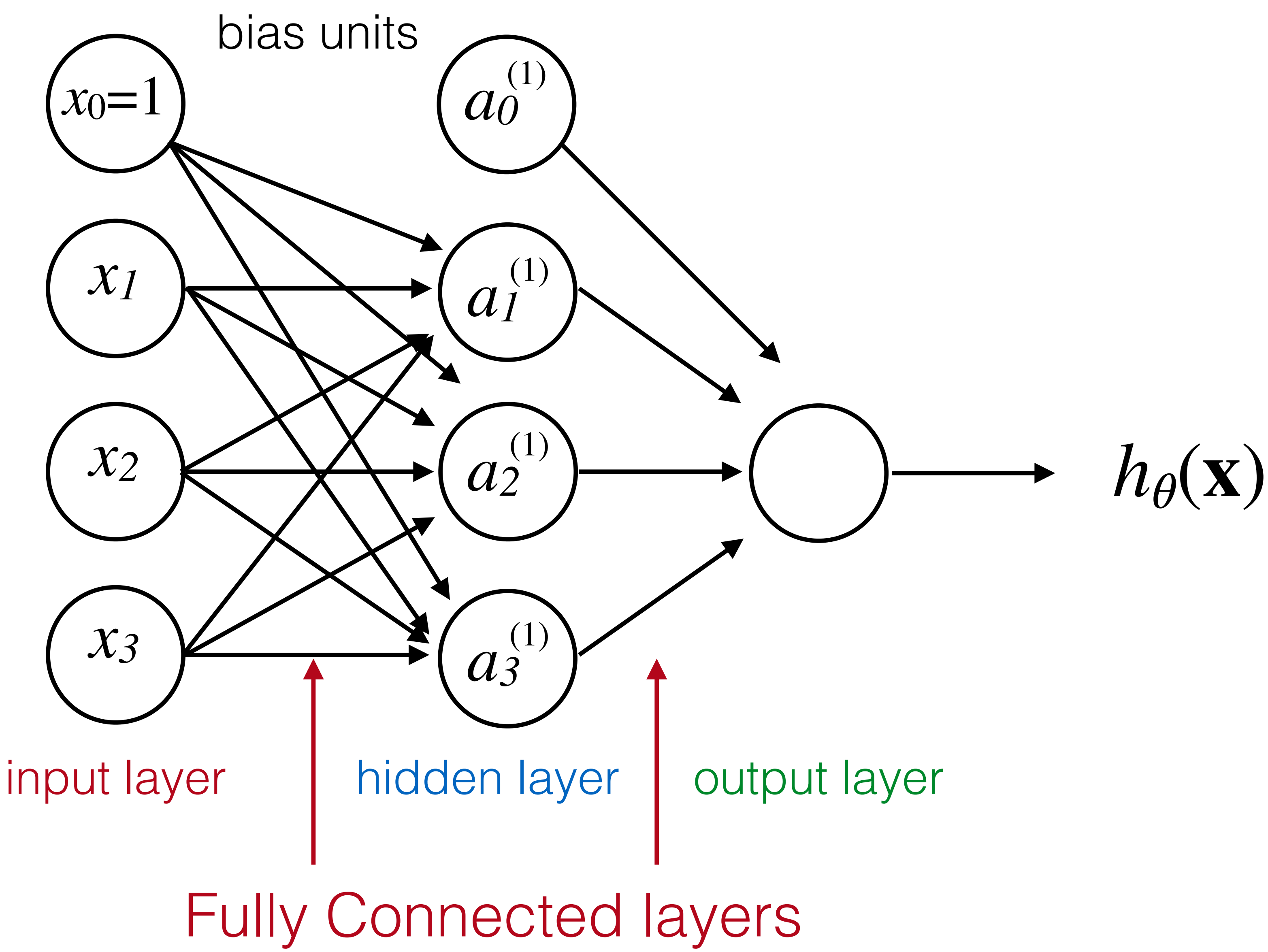
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural Networks: definition

- A (artificial) neural network is just a group of this different neurons strong together



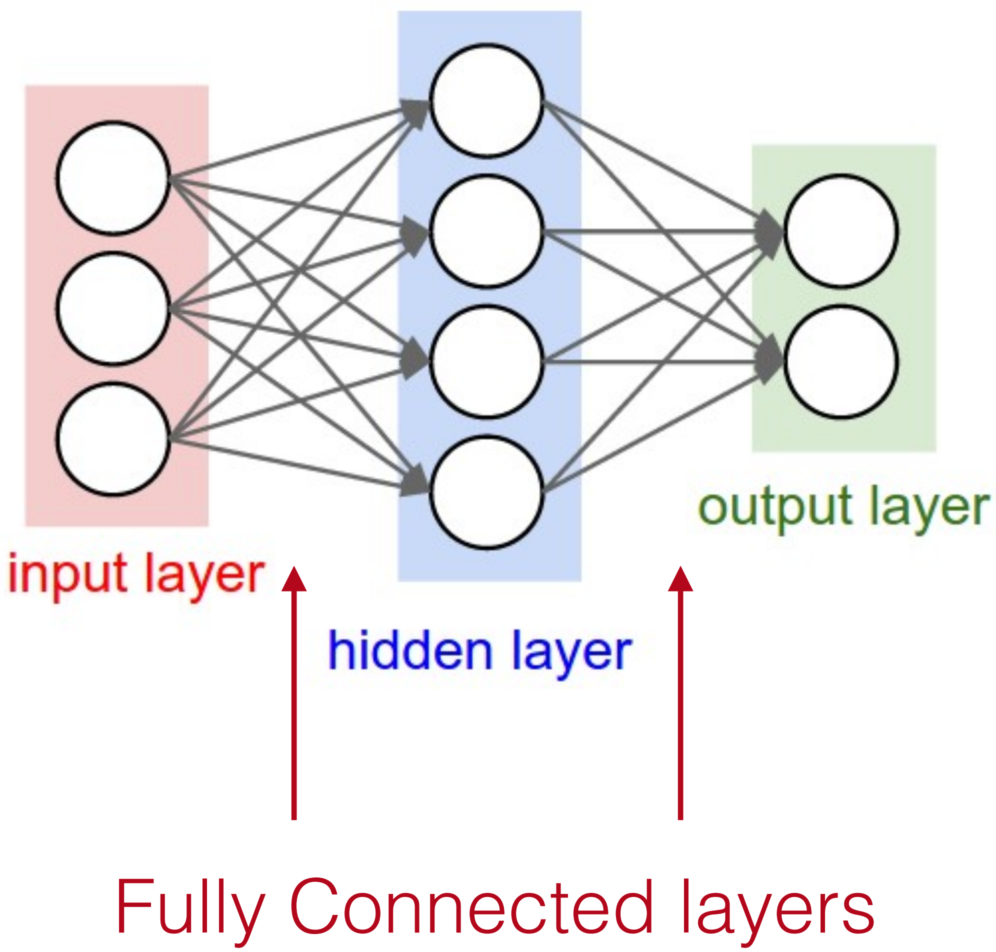
Adapted from slide by A. Ng

Neural Networks: architectures

- Here the term *architecture* refers to how the different neurons are connected to each other

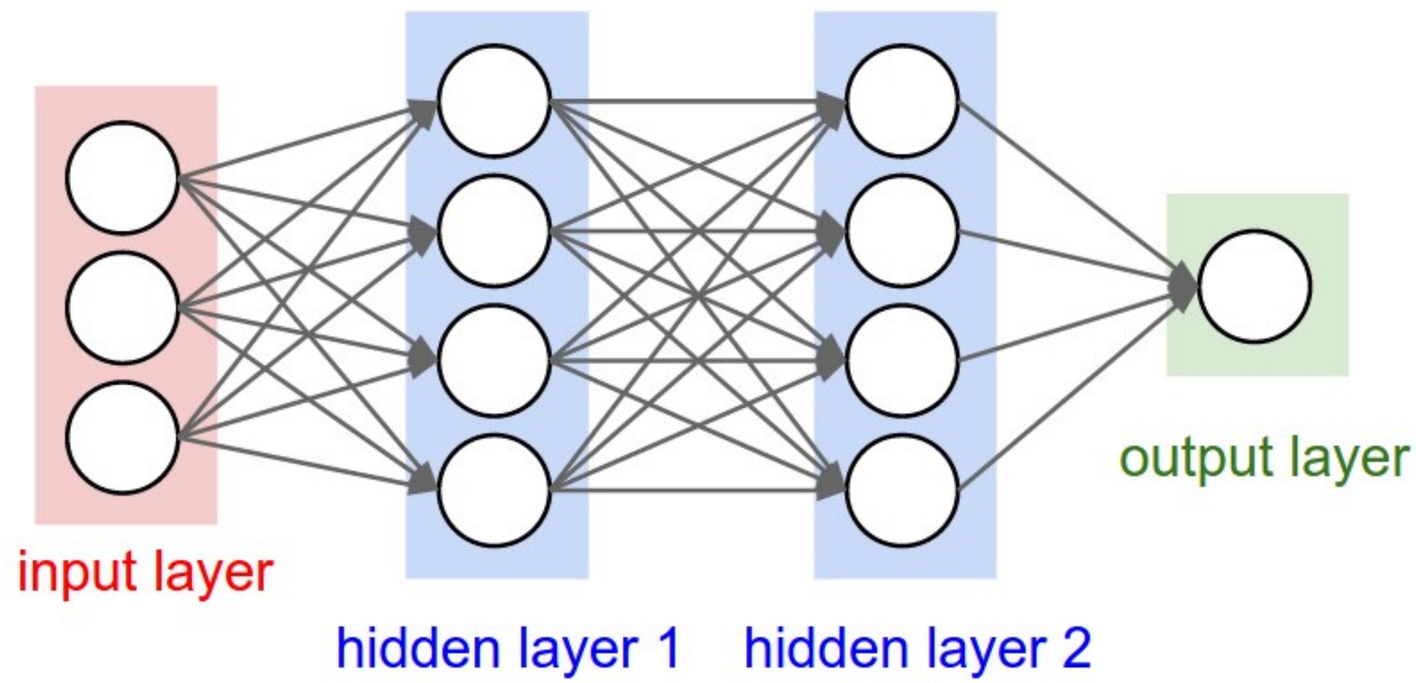
2-layer neural network

or 1-hidden-layer neural net



3-layer neural network

or 2-hidden-layer neural net

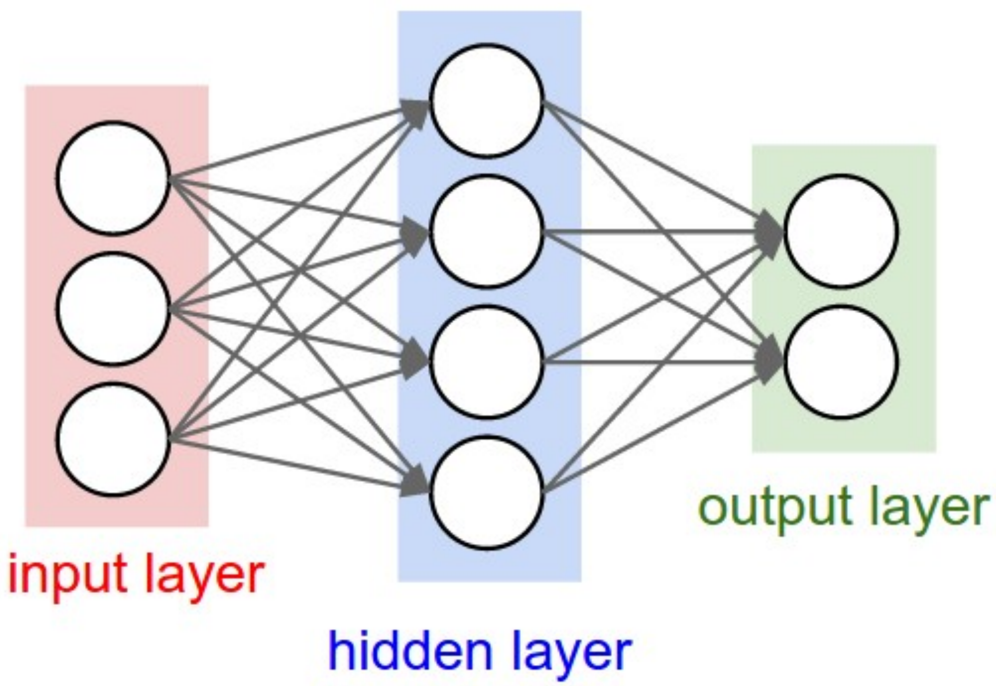


Note: when we say N-layer neural network, we do not count the input layer

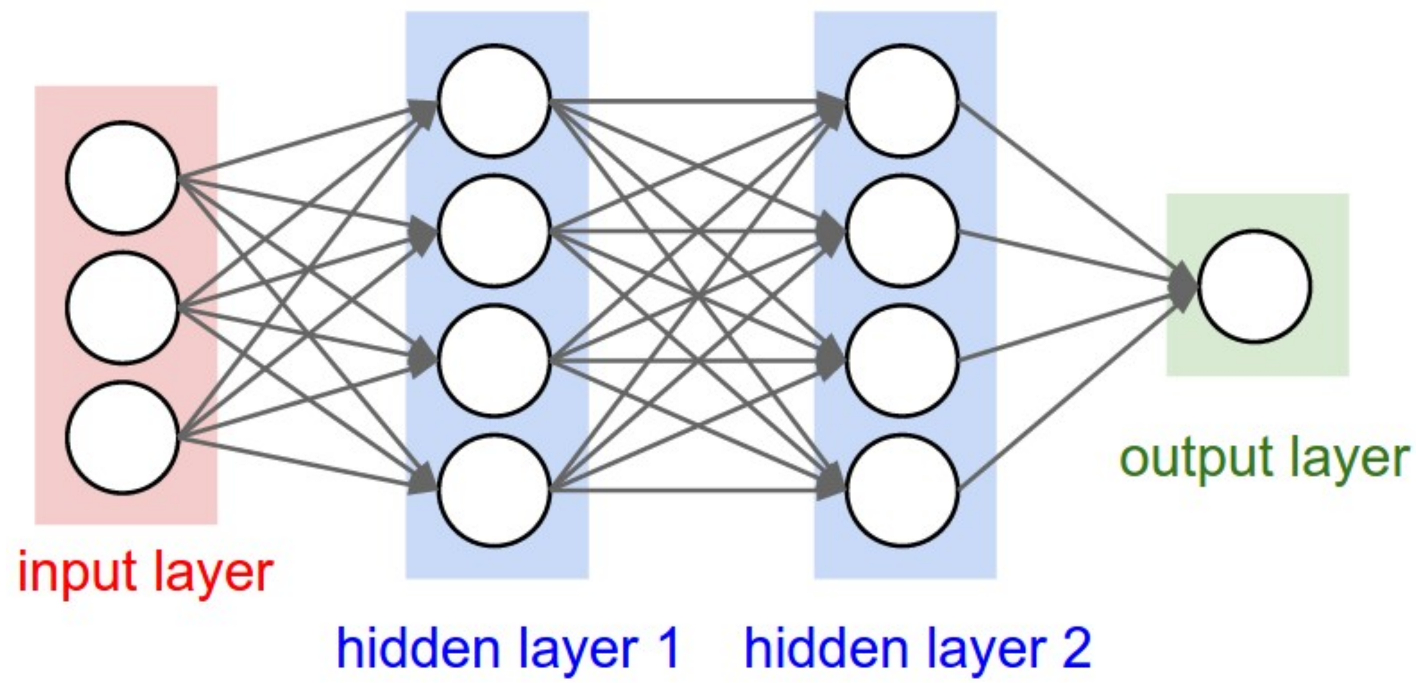
Neural Networks: architectures

- **Sizing neural networks:** the two metrics that are commonly used to measure the size of a NeuralNet are the #neurons or (more commonly) the #parameters

(a) **2-layer neural network**



(b) **3-layer neural network**

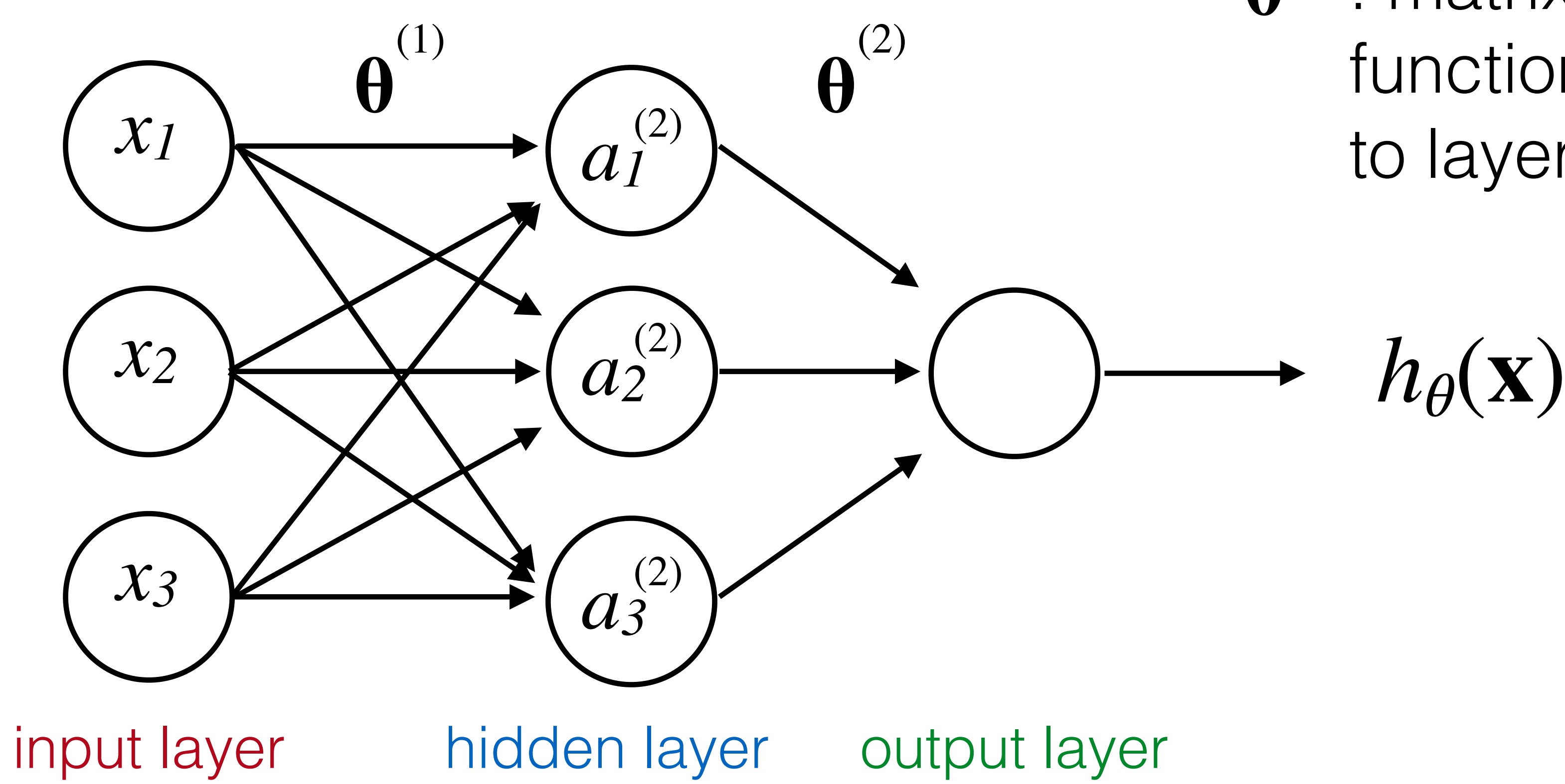


(b) has 4+4+1 neurons; how many (learnable) parameters?

(b) #parameters: 3x4 + 4x4 + 4x1 = 32 + 9 *(biases)* = 41

Neural Networks: model representation

- A (artificial) neural network is just a group of this different neurons strong together



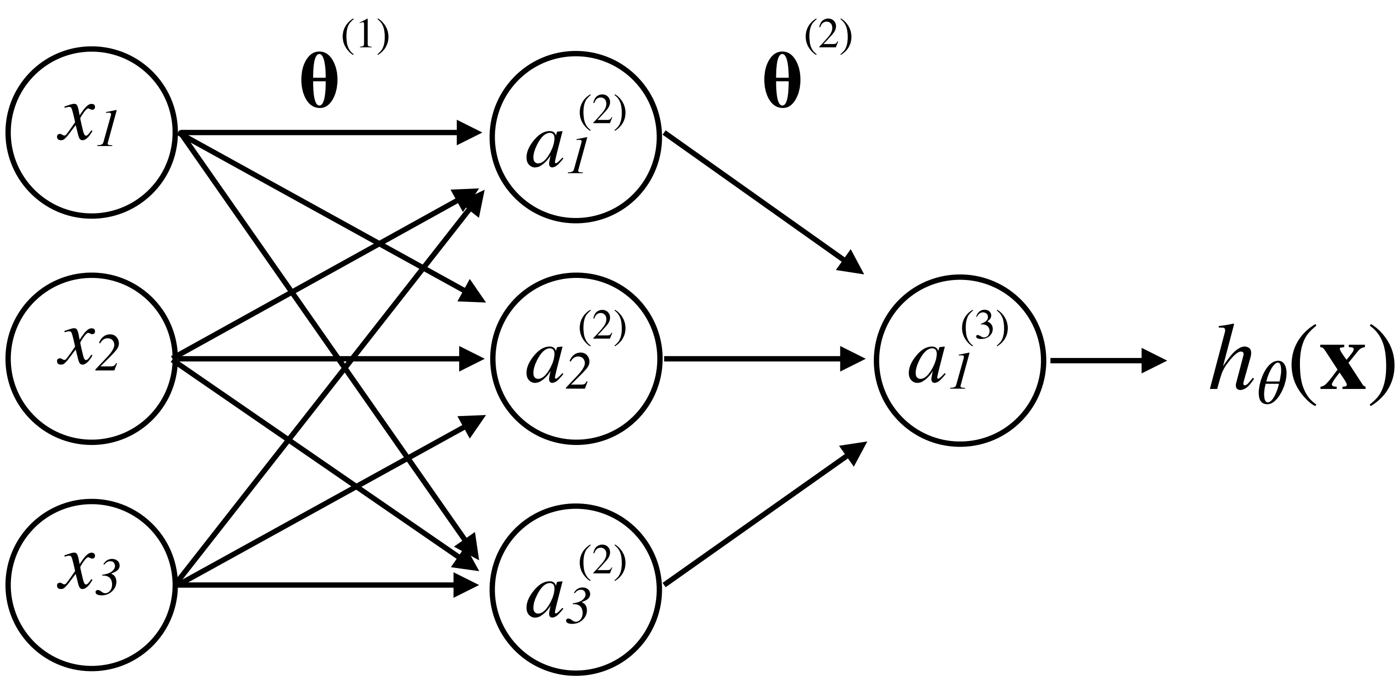
$a_i^{(j)}$: activation of unit i in layer j

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

Neural Networks: feed-forward computation

- Input units are set by some exterior function (it can be generated by sensors) which causes their output links to be activated at the specified level
- Working forward through the network, these outputs are going to be the input for the next layer
  - Each output is just the weighted sum of the activation on the links feeding into a node
  - The activation function transforms this linear combination: typically this is a non linear function (such a sigmoid)
  - This function corresponds to the “threshold” of that node

Neural Networks: feed-forward computation



$\boldsymbol{\theta}^{(1)} \in \mathbb{R}^{3\times4}$ $\quad$ $\boldsymbol{\theta}^{(2)} \in \mathbb{R}^{1\times4}$

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

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

If network has u_j units in layer j and u_{j+1} units in layer $j+1$, then $\boldsymbol{\theta}^{(j)}$ will be of dimension $u_{j+1} \times (u_j + 1)$

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

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

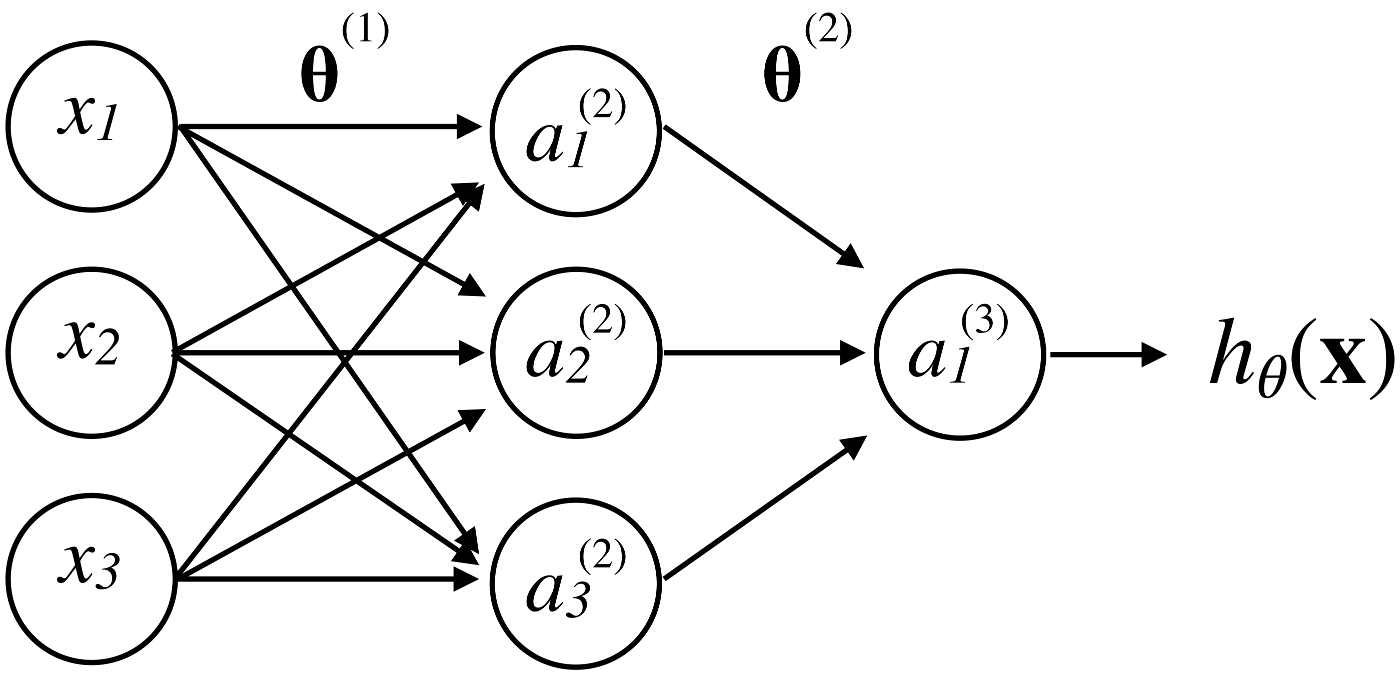
$$a_3^{(2)} = f(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$

$$h_\theta(\mathbf{x}) = a_1^{(3)} = f(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

Adapted from slide by A. Ng

Neural Networks: feed-forward computation

(vectorization)



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

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

$z_i^{(j)}$: linear combination of the input nodes

$$a_1^{(2)} = f\,\underbrace{\boxed{(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)}}_{z_1^{(2)}} = f\,(z_1^{(2)})$$

$$a_2^{(2)} = f\,(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3) = f\,(z_2^{(2)})$$

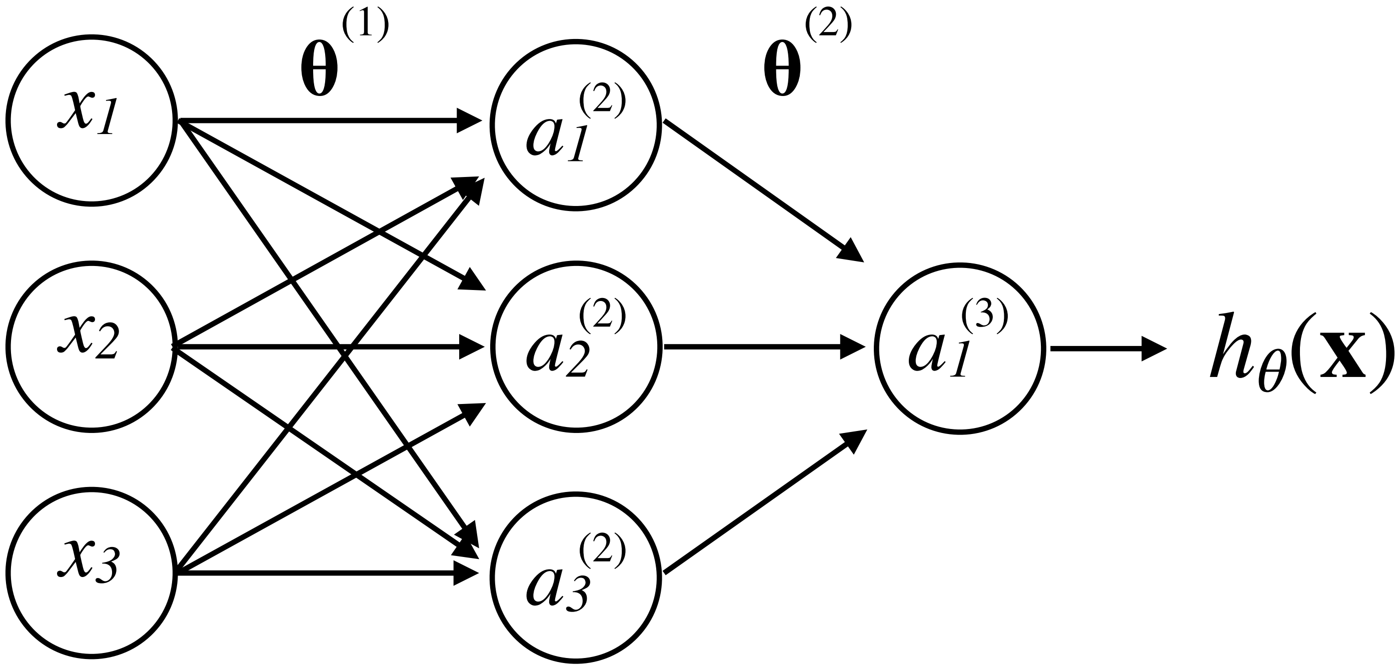
$$a_3^{(2)} = f\,(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3) = f\,(z_3^{(2)})$$

$$h_\theta(\mathbf{x}) = a_1^{(3)} = f\,(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}) = f\,(z_1^{(3)})$$

Adapted from slide by A. Ng

Neural Networks: feed-forward computation

(vectorization)



$$\boldsymbol{a}^{(1)} = \mathbf{x}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_3 \end{bmatrix} \qquad \mathbf{z}^{(2)} = \begin{bmatrix} z_1^{(2)} \\ \vdots \\ z_3^{(2)} \end{bmatrix}$$

$$a_1^{(2)} = f(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3) = f(z_1^{(2)})$$

$$a_2^{(2)} = f(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3) = f(z_2^{(2)})$$

$$a_3^{(2)} = f(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3) = f(z_3^{(2)})$$

$$\mathbf{z}^{(2)} = \boldsymbol{\theta}^{(1)} \boldsymbol{a}^{(1)}$$

$$\boldsymbol{a}^{(2)} = f(\mathbf{z}^{(2)})$$

Add bias: $a_0^{(2)} = 1$

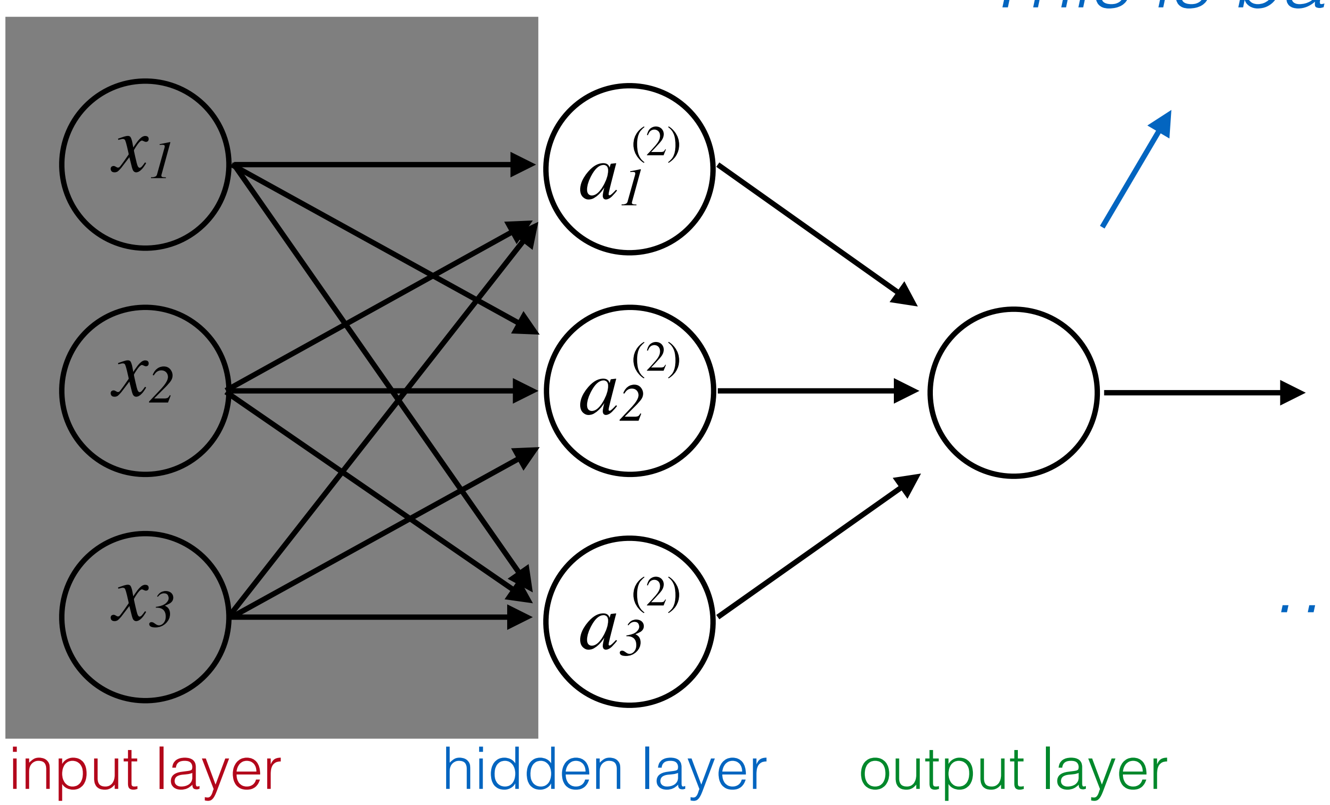
$$\mathbf{z}^{(3)} = \boldsymbol{\theta}^{(2)} \boldsymbol{a}^{(2)}$$

$$h_\theta(\mathbf{x}) = a_1^{(3)} = f(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}) = f(z_1^{(3)}) = \boldsymbol{a}^{(3)} = f(\mathbf{z}^{(3)})$$

"forward propagation"

Neural Networks: feed-forward computation

- This forward propagation view also help us to understand what neural networks might be doing



This is basically logistic regression…

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

$$h_\theta(\mathbf{x}) = f(\boldsymbol{\theta}^{(2)} \boldsymbol{a}^{(2)})$$

… but now features ($\boldsymbol{a}^{(2)}$) are learned by the network

Adapted from slide by A. Ng

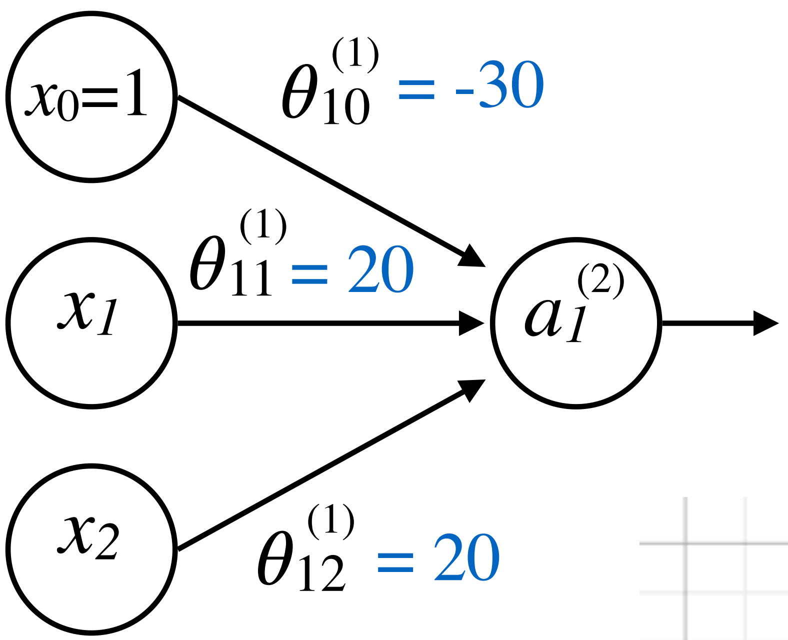
A few examples

- Let's try to compute logical functions

$x_1, x_2 \in \{0, 1\}$

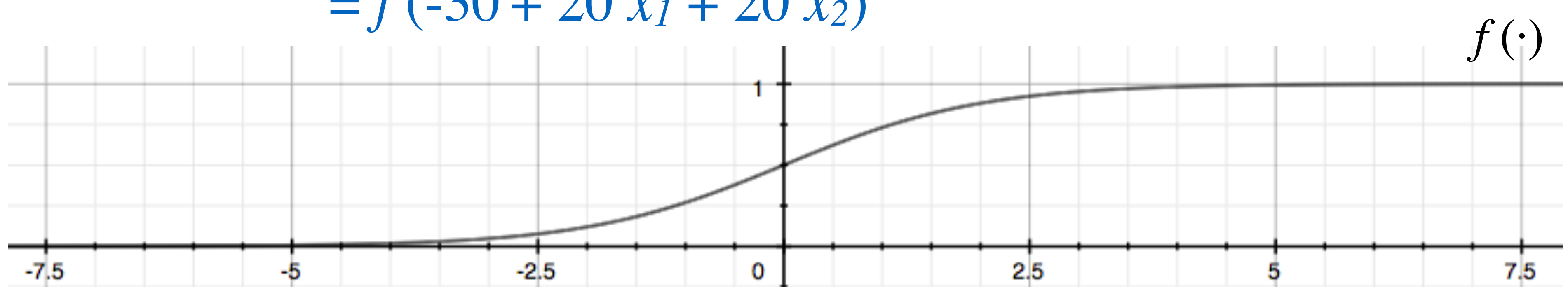
$y = x_1 \text{ AND } x_2$

x_1	x_2	y	$h_\theta(\mathbf{x})$
0	0	0	$= f(-30) \approx 0$
0	1	0	$= f(-10) \approx 0$
1	0	0	$= f(-10) \approx 0$
1	1	1	$= f(10) \approx 1$



$$h_\theta(\mathbf{x}) = a_1^{(2)} = f(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$$

$$= f(-30 + 20\, x_1 + 20\, x_2)$$



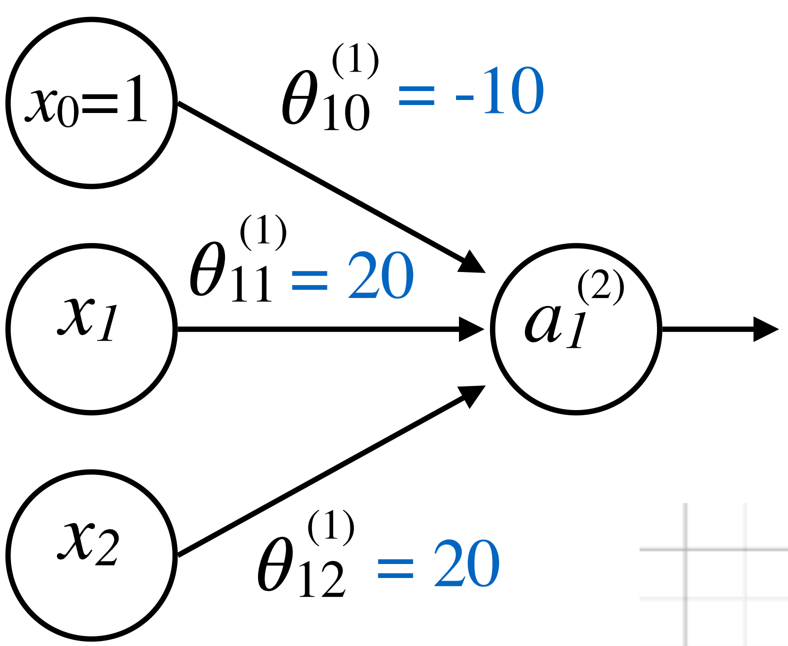
A few examples

- Let's try to compute logical functions

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ OR } x_2$

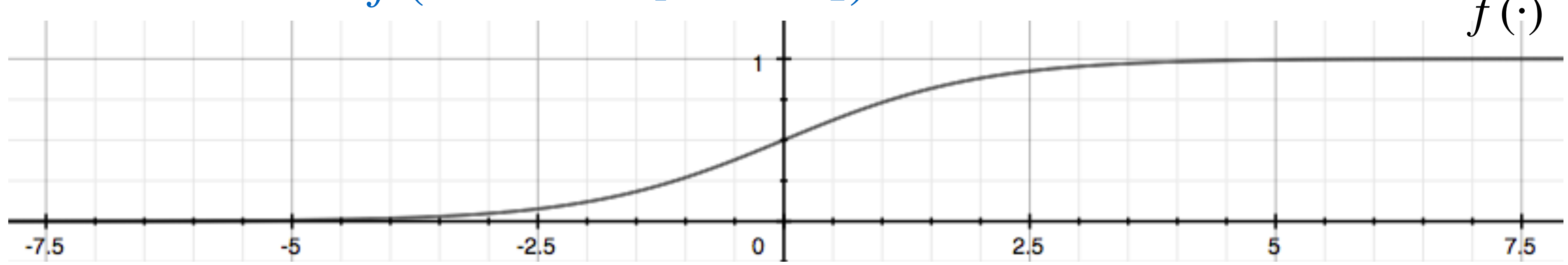
x_1	x_2	y	$h_\theta(\mathbf{x})$
0	0	0	$= f(-10) \approx 0$
0	1	1	$= f(10) \approx 1$
1	0	1	$= f(10) \approx 1$
1	1	1	$= f(30) \approx 1$



$$h_\theta(\mathbf{x}) = a_1^{(2)} = f(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$$

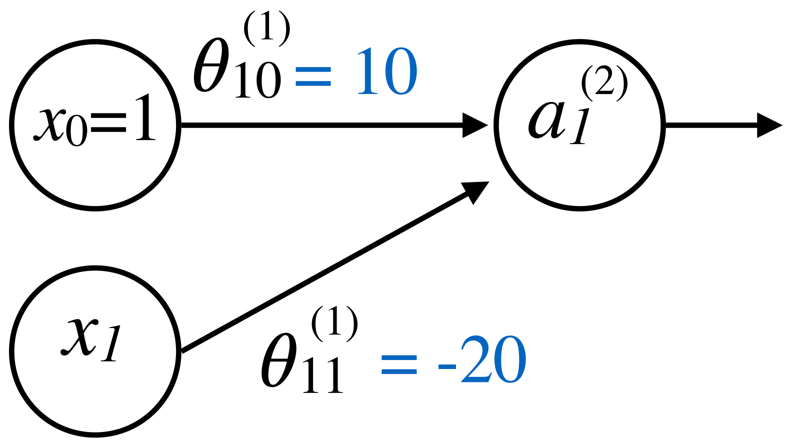
$$= f(-10 + 20\, x_1 + 20\, x_2)$$

$f(\cdot)$



Representing Boolean functions

y = (NOT x_1)

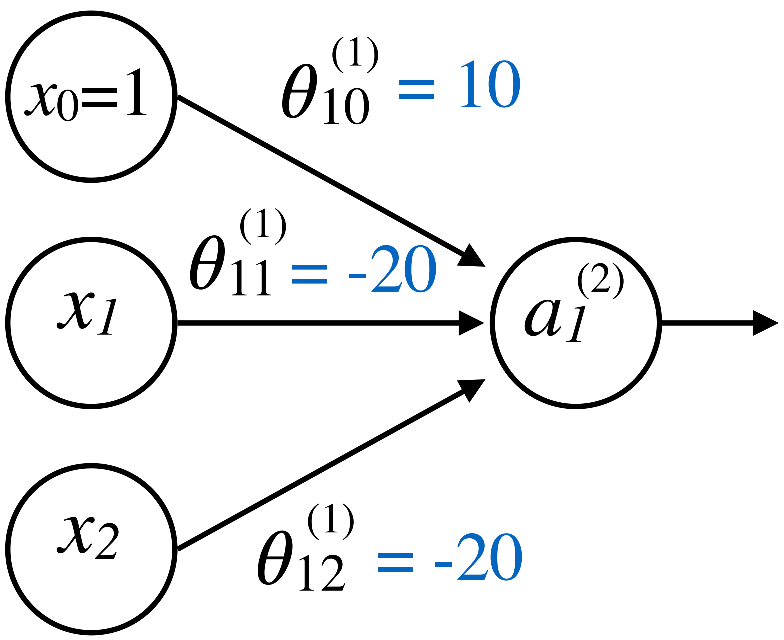


$$h_\theta(\mathbf{x}) = f\,(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1)$$

$$= f\,(10 - 20\, x_1)$$

x_1	y	$h_\theta(\mathbf{x})$
0	1	$= f\,(10) \approx 1$
1	0	$= f\,(-10) \approx 0$

y = (NOT x_1) AND (NOT x_2)



$$h_\theta(\mathbf{x}) = f\,(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$$

$$= f\,(10 - 20\, x_1 - 20\, x_2)$$

x_1	x_2	y	$h_\theta(\mathbf{x})$
0	0	1	$= f\,(10) \approx 1$
0	1	0	$= f\,(-10) \approx 0$
1	0	0	$= f\,(-10) \approx 0$
1	1	0	$= f\,(-30) \approx 0$

Representing Boolean functions

- In 1969 Minsky and Seymour showed that it was impossible for perceptrons to learn an XOR function
    - Often it's (incorrectly) reported that they also conjectured that a similar result would hold for multi-layer perceptrons
    - Nevertheless, it caused a significant decline in interest and funding of neural network research
    - It took ten more years until neural networks experienced a resurgence in the 1980s

M.Minsky, S.Papert, "Perceptrons: an introduction to computational geometry", MIT Press, 1969 *(expanded edition published in 1987)*

Contact

- **Office:** Torre Archimede 6CD, room 622
- **Office hours** (ricevimento)**:** Monday 11:00-13:00

✉ lamberto.ballan@unipd.it

⌂ http://www.lambertoballan.net

⌂ http://vimp.math.unipd.it

@ twitter.com/lambertoballan