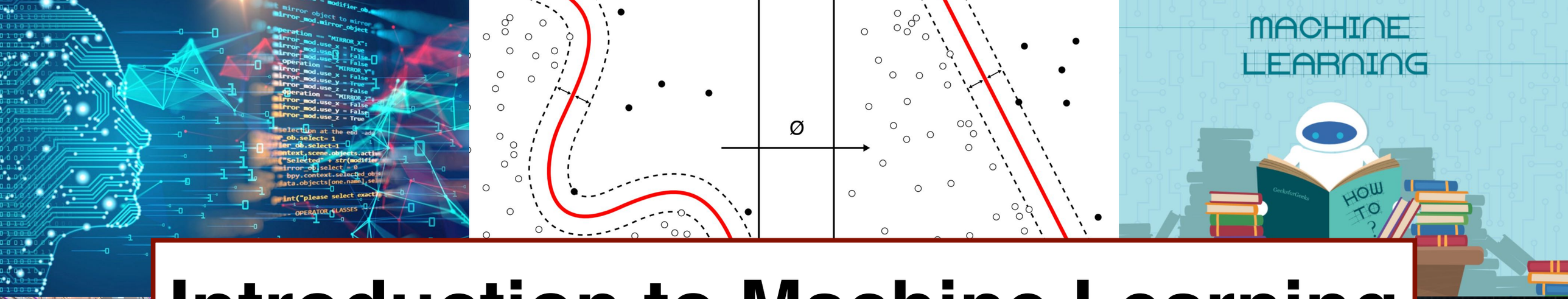


Introduction to Machine Learning

SCP8084699 - LT Informatica

Artificial Neural Networks II

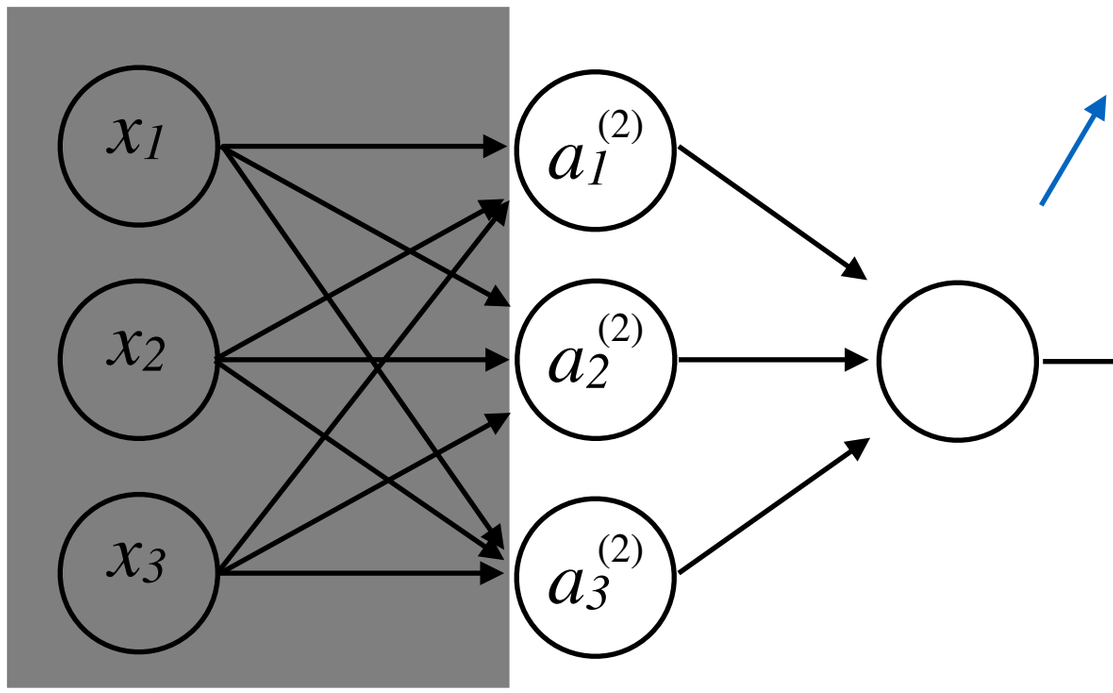
Prof. Lamberto Ballan

University of Padova, A.Y. 2021/22 April 7, 2022

DIPARTIMENTO MATEMATICA
Department of Mathematics “Tullio Levi-Civita”

Recap: Feed-forward Neural Net

- 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

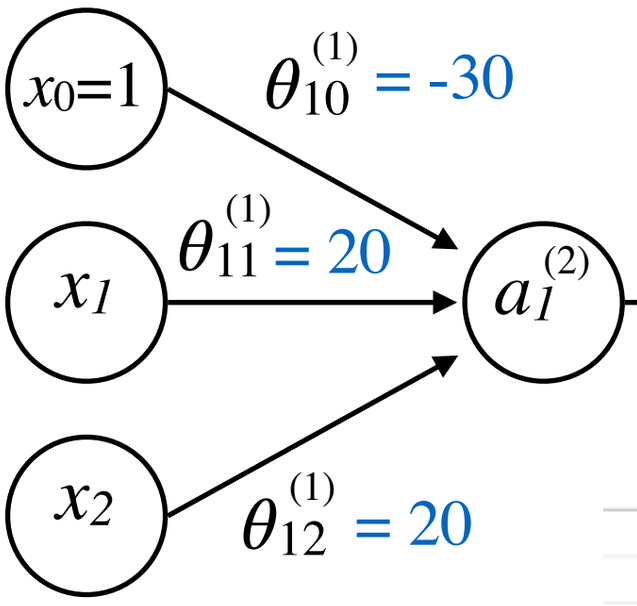
A few examples

- Let's try to compute logical functions

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

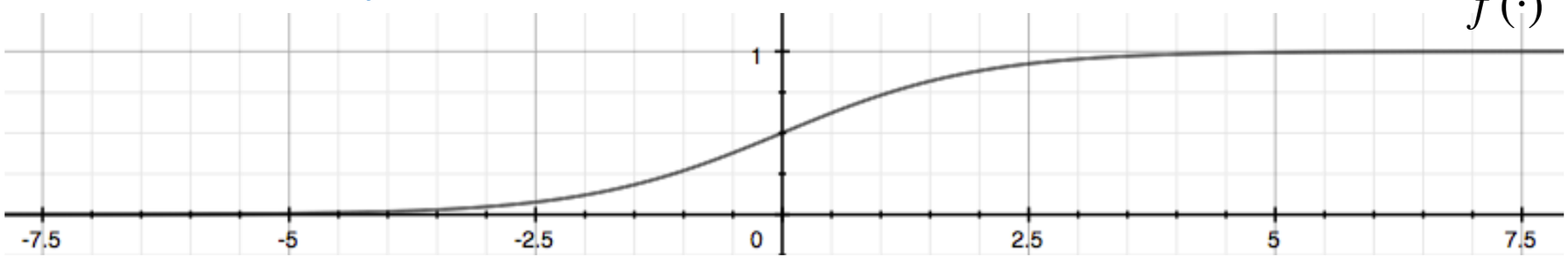
$y = x_1$ AND x_2

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	$= f(10) \approx 1$



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

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



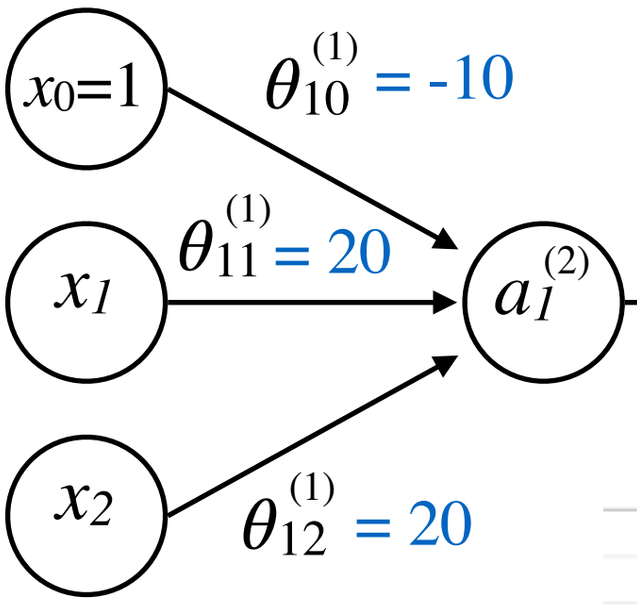
A few examples

- Let's try to compute logical functions

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

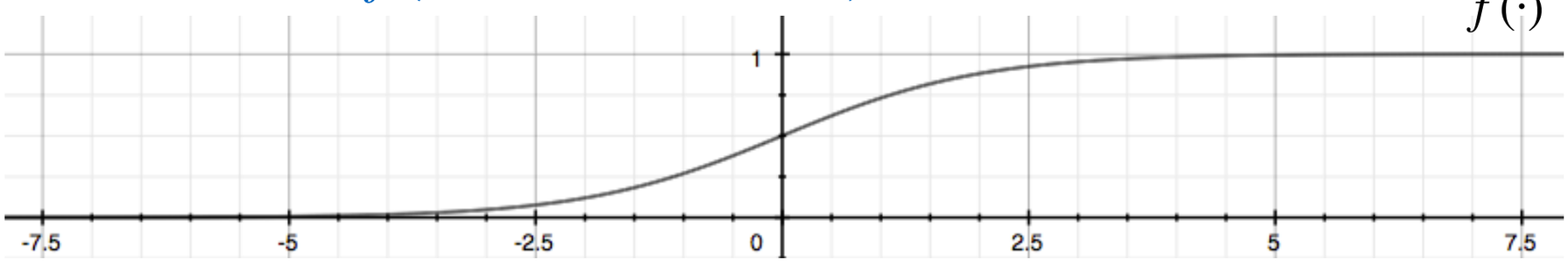
$y = x_1$ OR x_2

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	$= f(30) \approx 1$



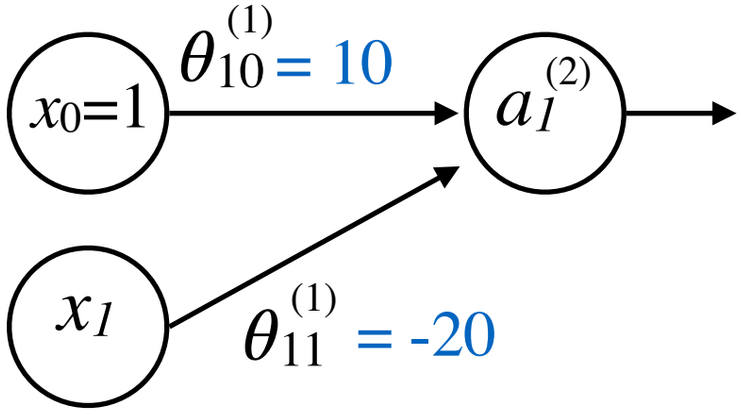
$$h_\theta(\mathbf{x}) = a_1^{(2)} = f(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$$

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



Representing Boolean functions

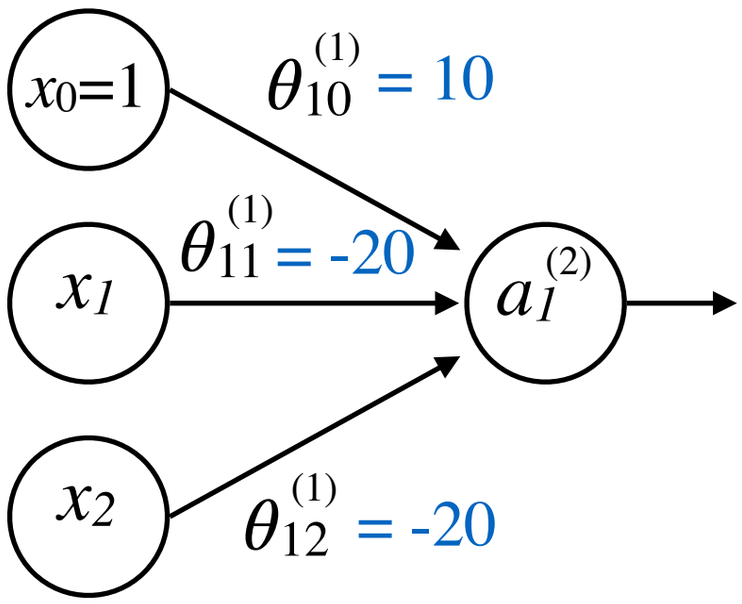
$y = (\text{NOT } x_1)$



$$h_\theta(\mathbf{x}) = f(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1)$$
$$= f(10 - 20\, x_1)$$

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

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



$$h_\theta(\mathbf{x}) = f(\theta_{10}^{(1)} x_0 + + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$$
$$= f(10 - 20\, x_1 - 20\, x_2)$$

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)*

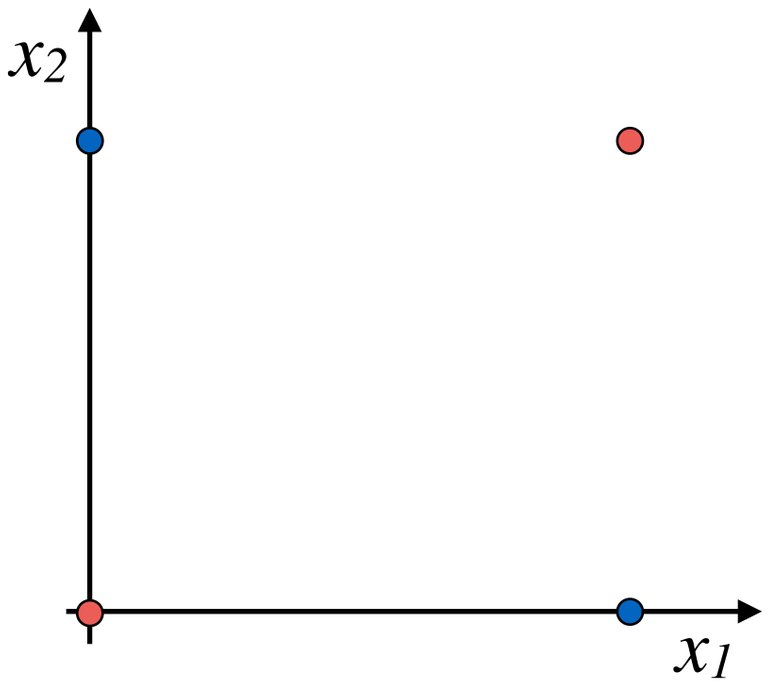
Representing Boolean functions

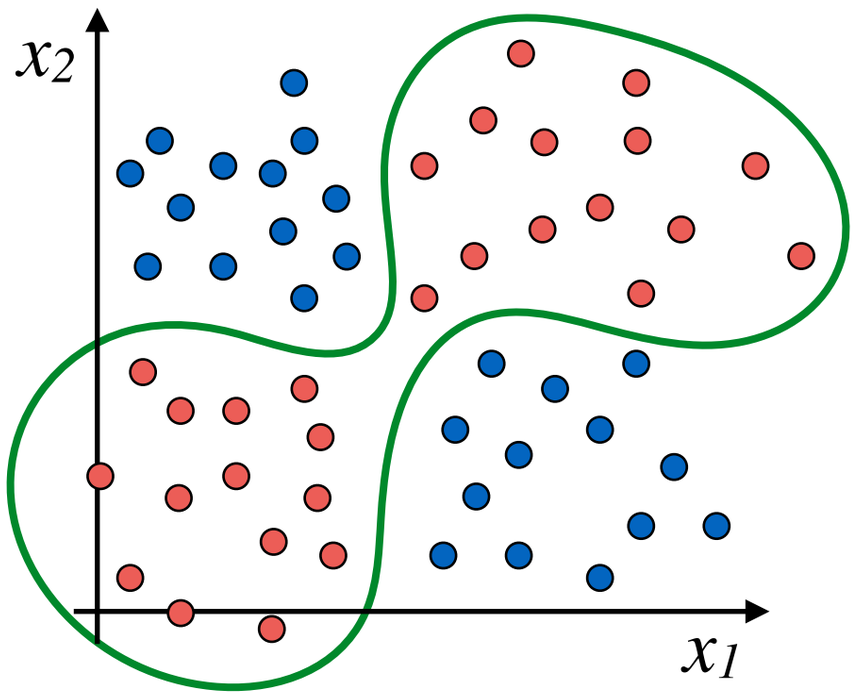
- You can think at this as simplified version of a more complex (non-linear) classification problem

$x_1, x_2 \in \{0, 1\}$ (binary)

$y = x_1 \text{ XOR } x_2$ / $y = x_1 \text{ XNOR } x_2$

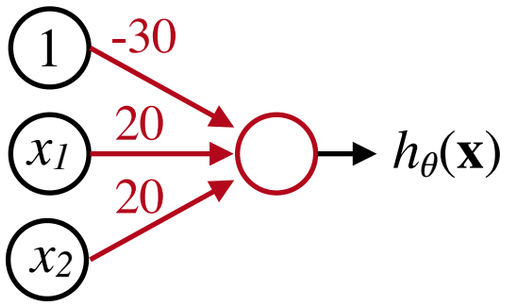
- Positive class
- Negative class



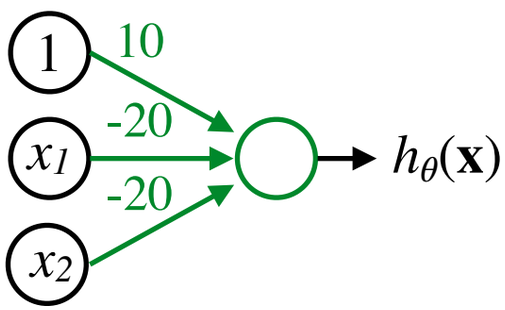


Representing Boolean functions

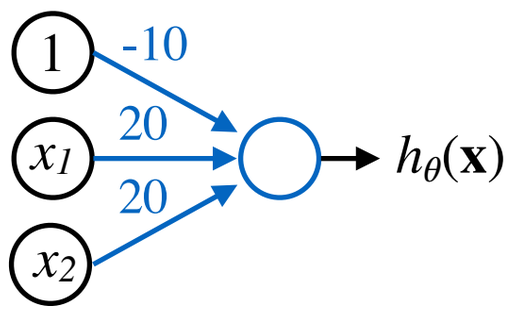
- Combining representations for non-linear functions



x_1 AND x_2

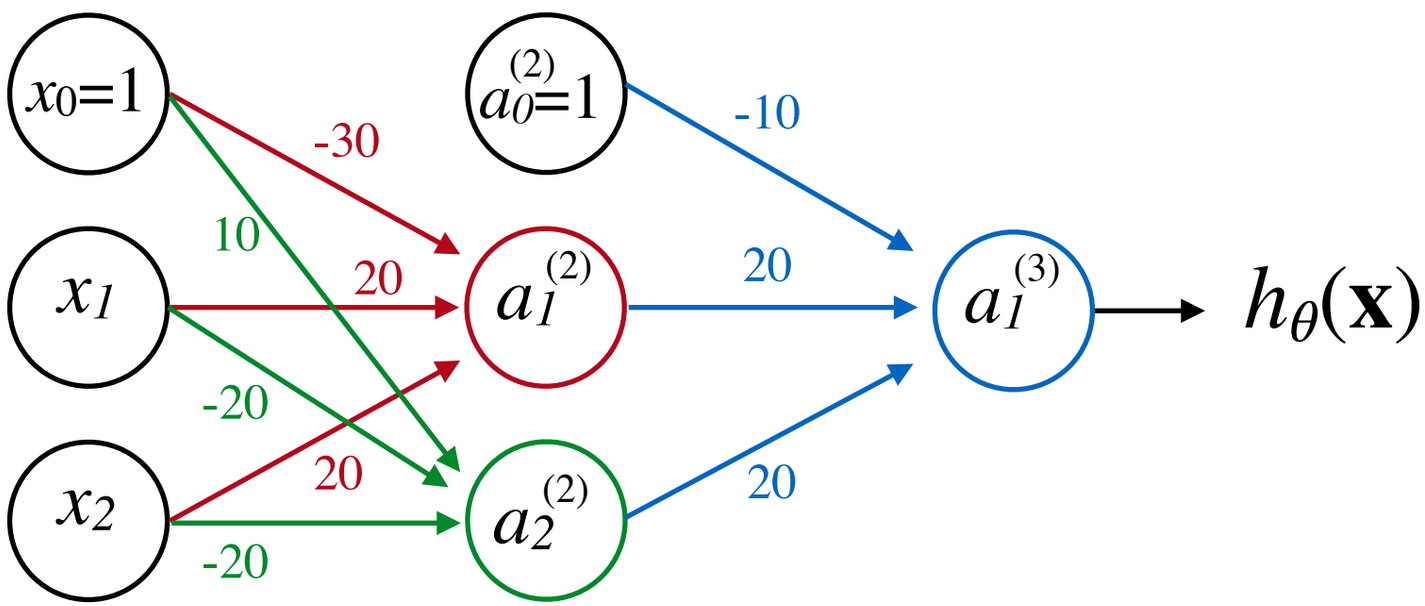


(NOT x_1) AND (NOT x_2)



x_1 OR x_2

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	y	$h_\theta(\mathbf{x})$
0	0	≈ 0	≈ 1	1	≈ 1
0	1	≈ 0	≈ 0	0	≈ 0
1	0	≈ 0	≈ 0	0	≈ 0
1	1	≈ 1	≈ 0	1	≈ 1

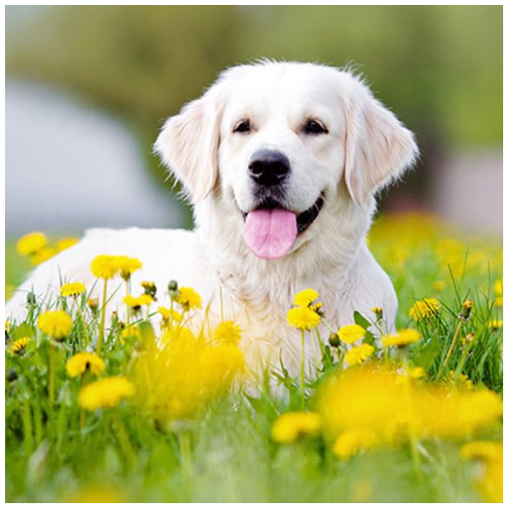


$y = x_1$ XNOR x_2

Another example

- We introduce a binary classifier to recognised cats and dogs (based on two simple features)

Classes



“dog”

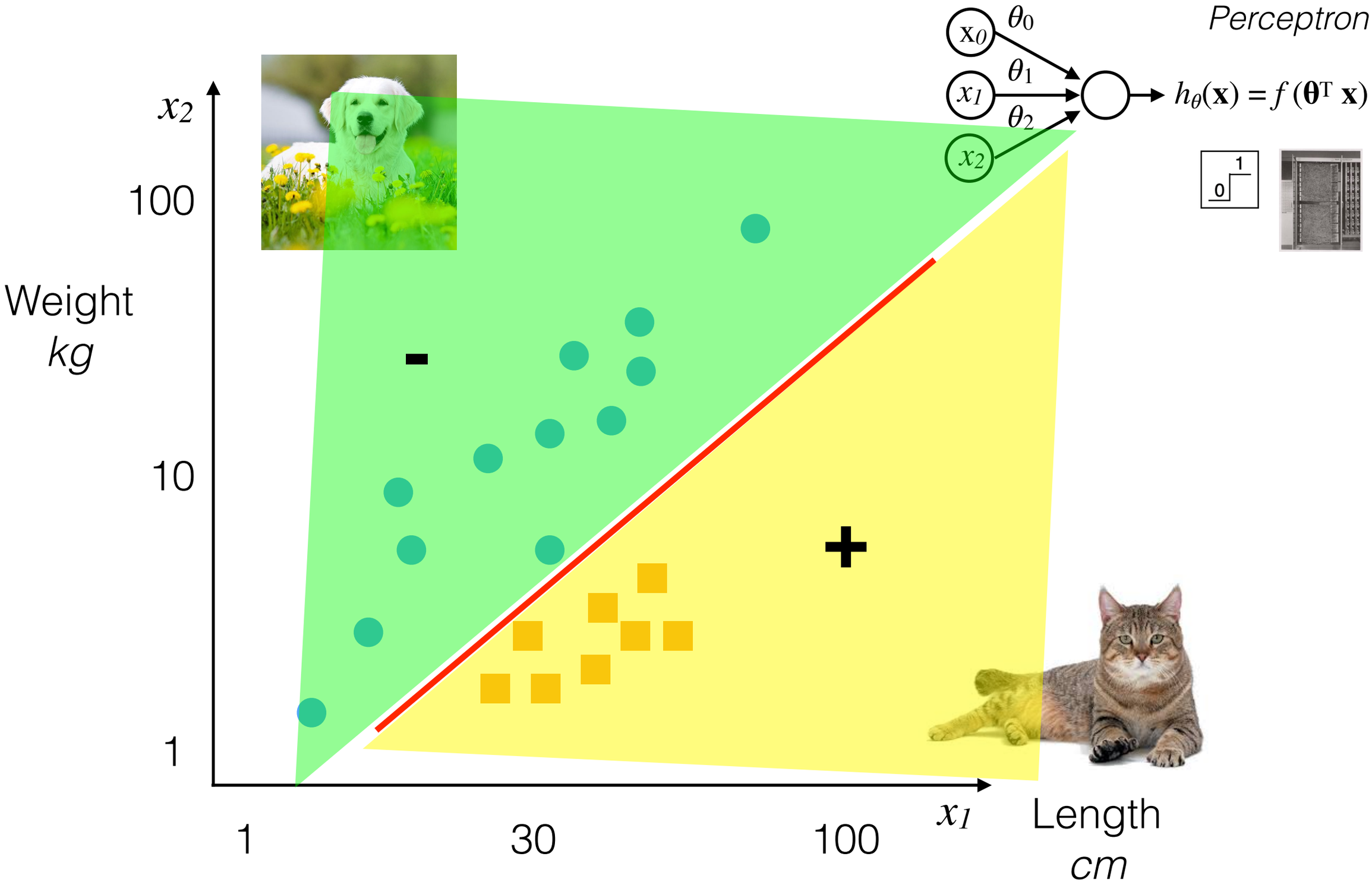
“cat”

Features / Representations

Weight

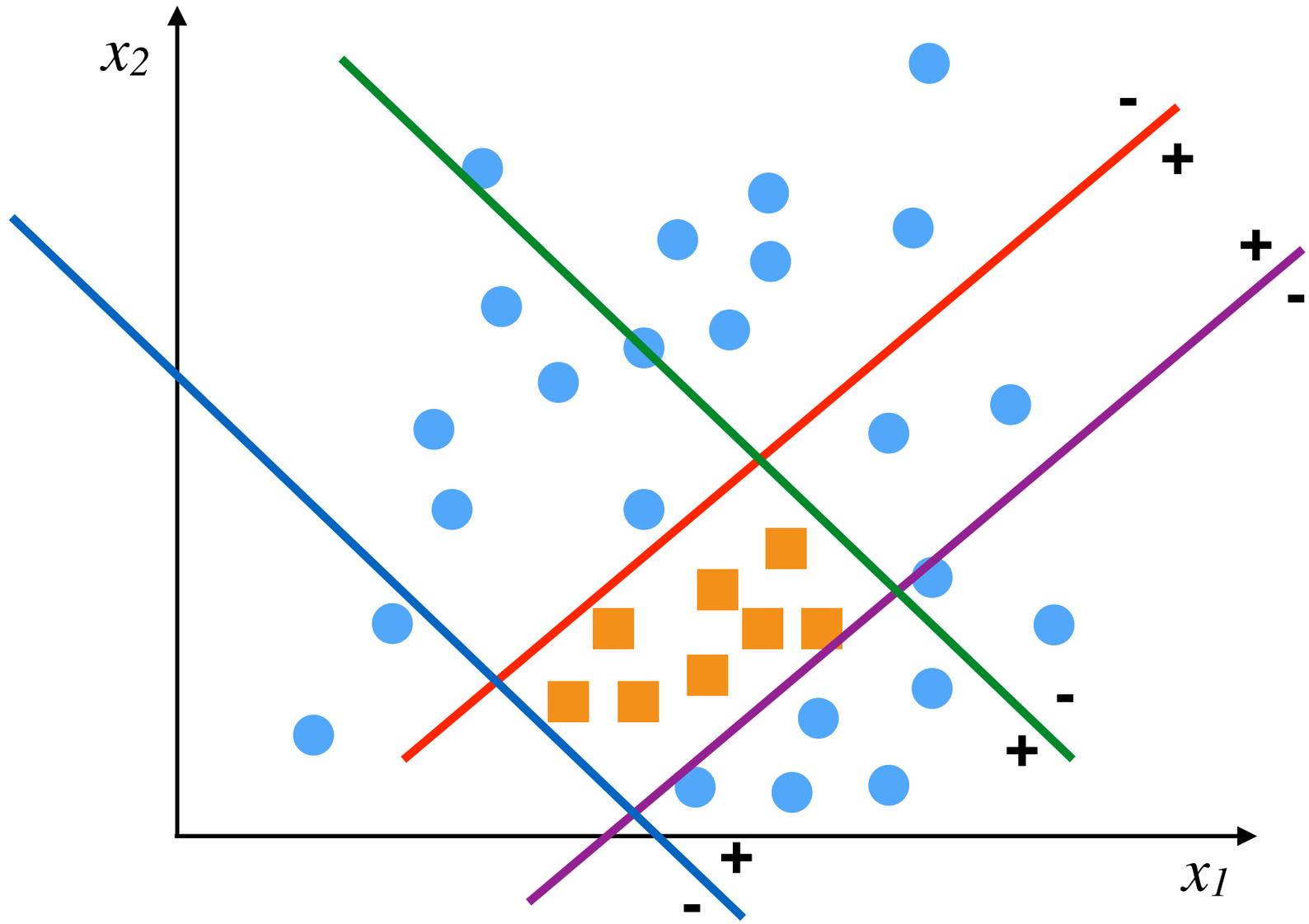
Length

Another example



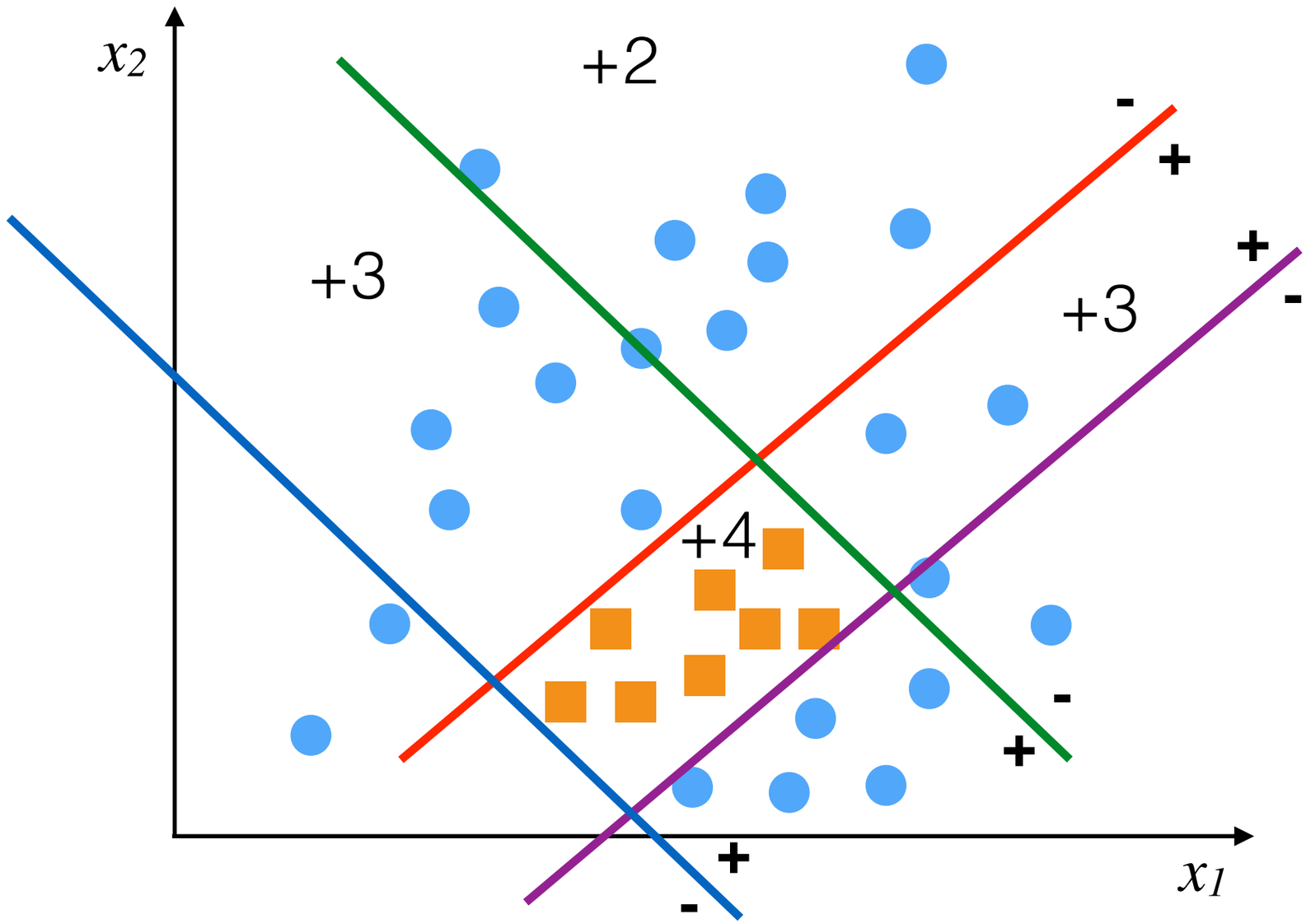
Another example

- Let's generalize: now we look for "cat" vs "non-cat"



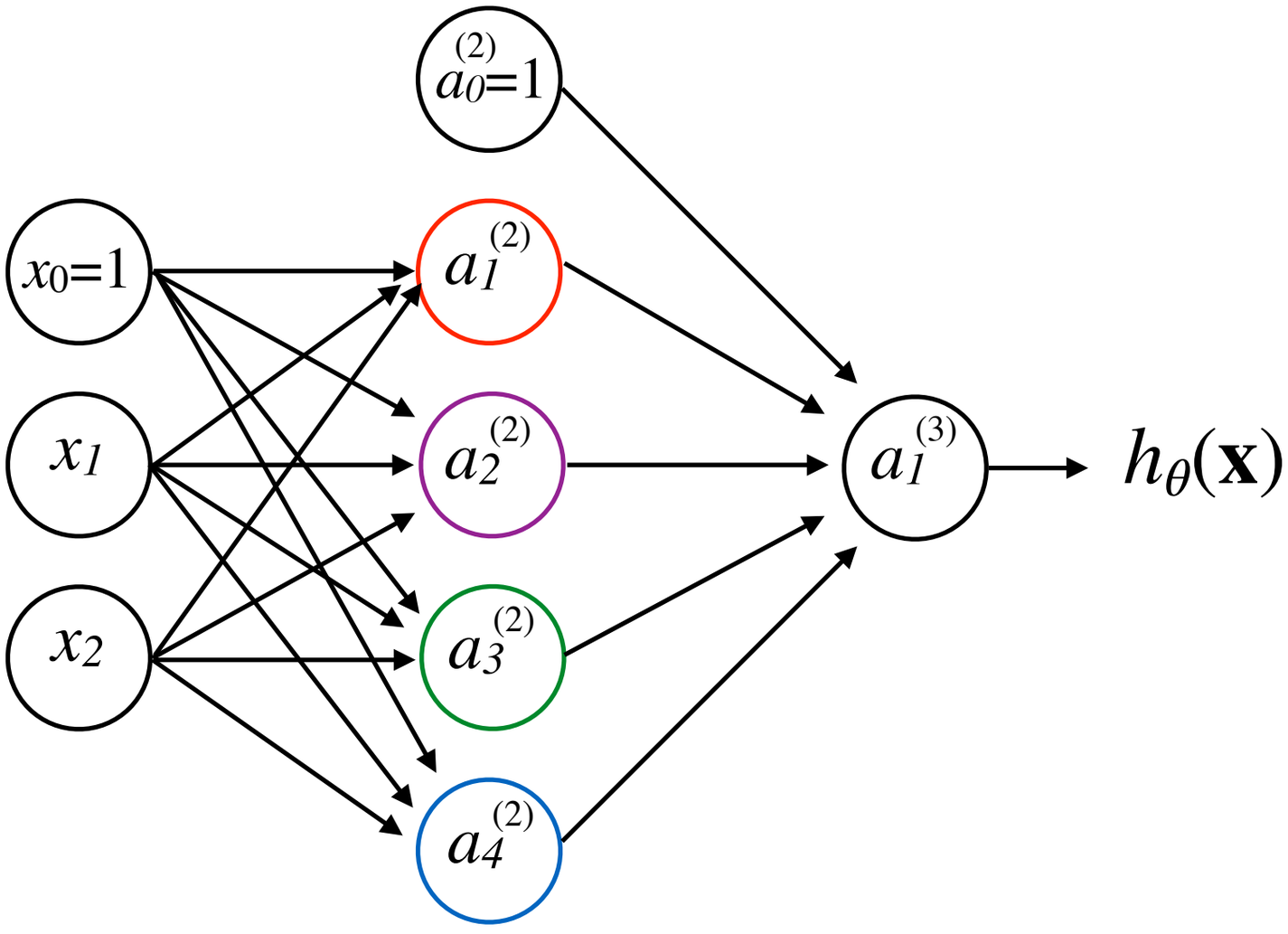
Another example

- Combining representations for non-linear functions



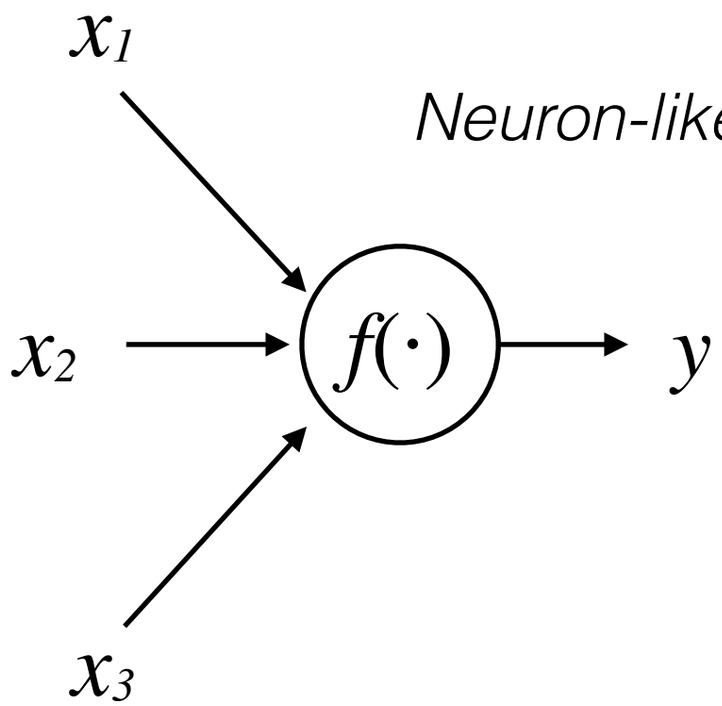
Another example

- Combining representations for non-linear functions



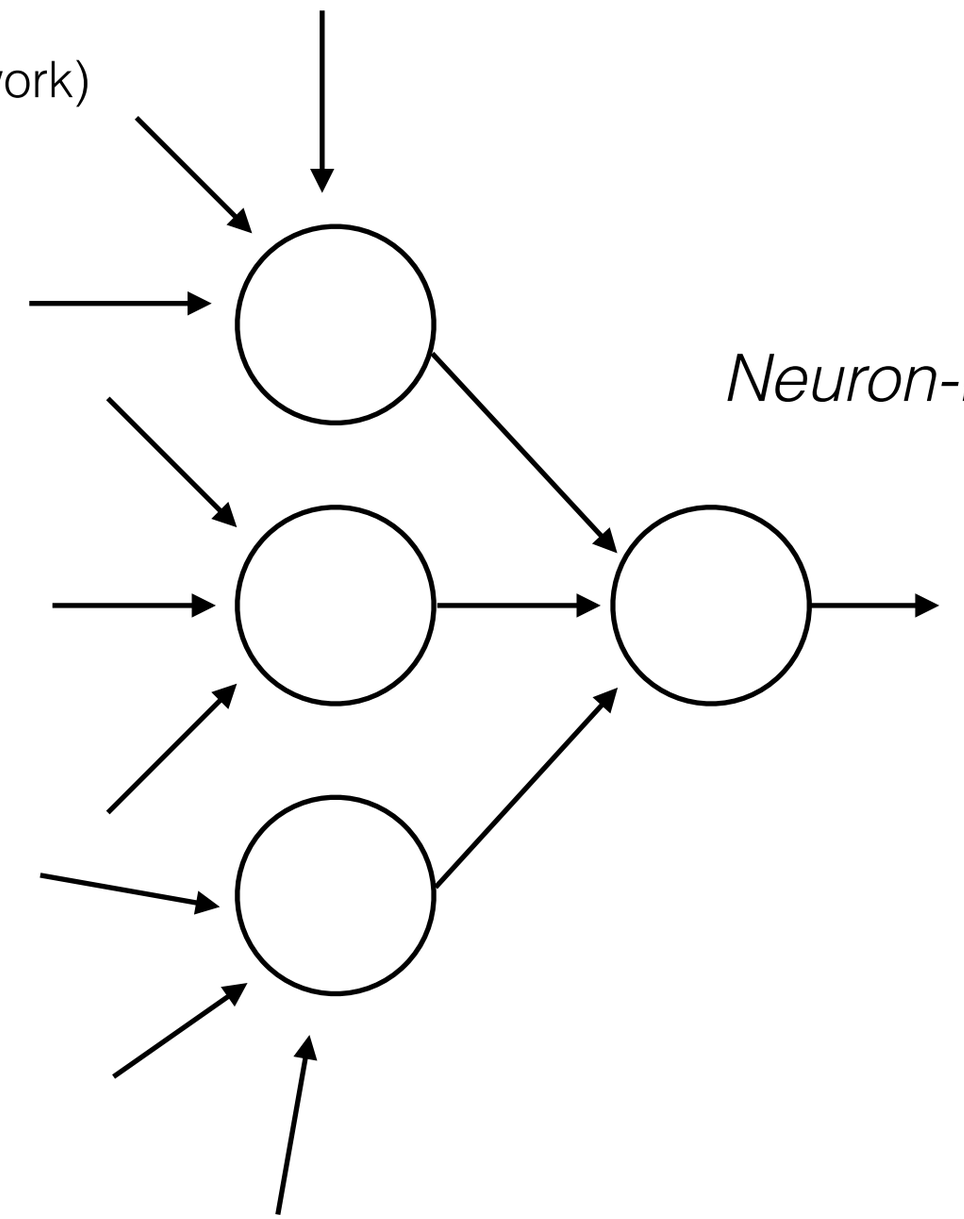
Deep Neural Networks

Deep Learning
(e.g. Convolutional Neural Network)



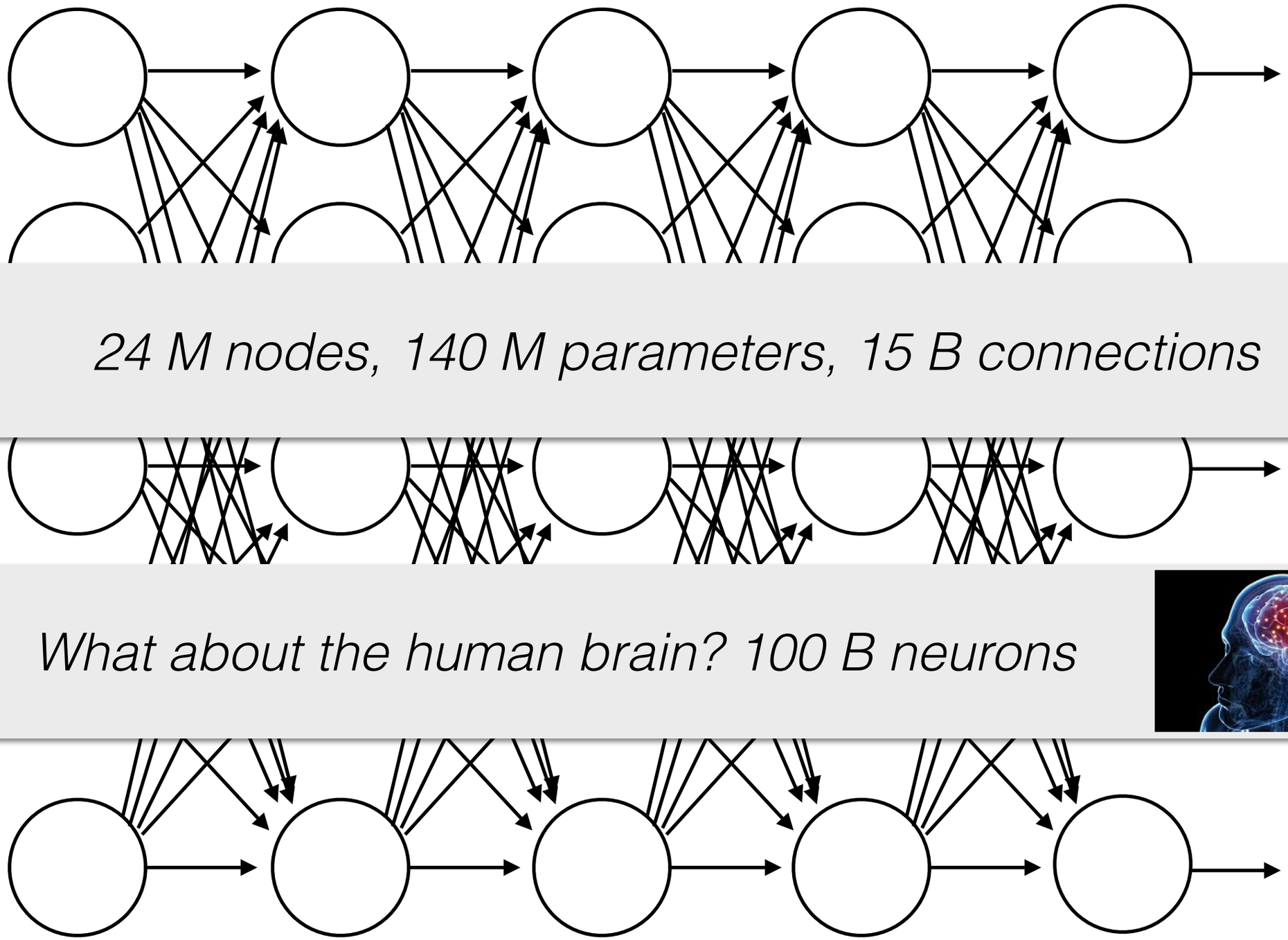
Deep Neural Networks

Deep Learning
(e.g. Convolutional Neural Network)



Neuron-like unit

Deep Neural Networks



24 M nodes, 140 M parameters, 15 B connections

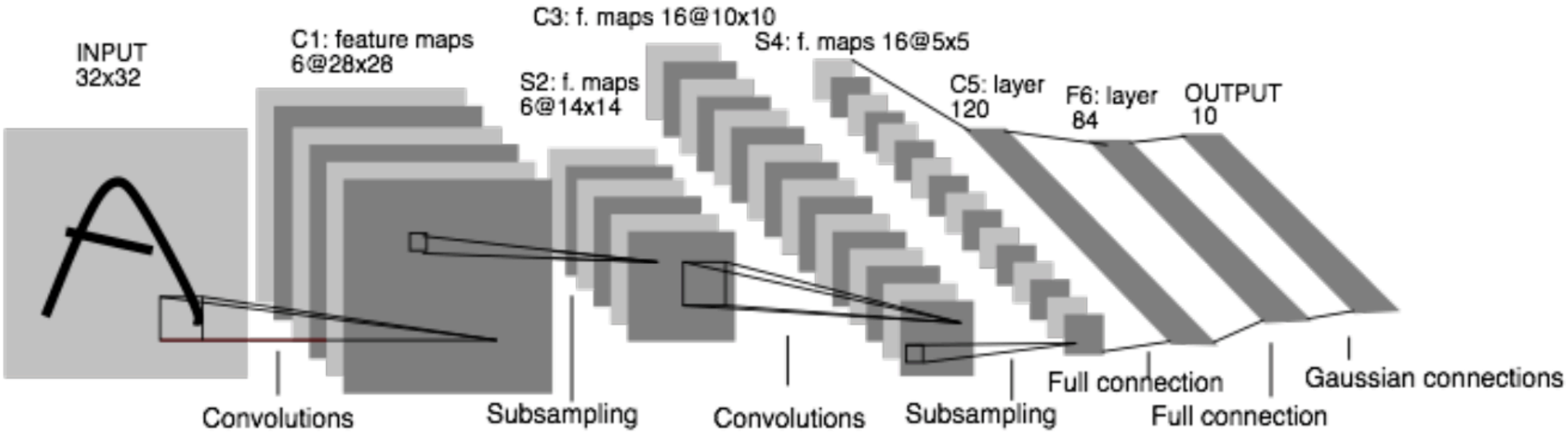
What about the human brain? 100 B neurons

Deep Neural Networks

- CNNs: the assumption that the inputs are images allows us to encode certain local properties into the architecture

Convolutional Neural Network

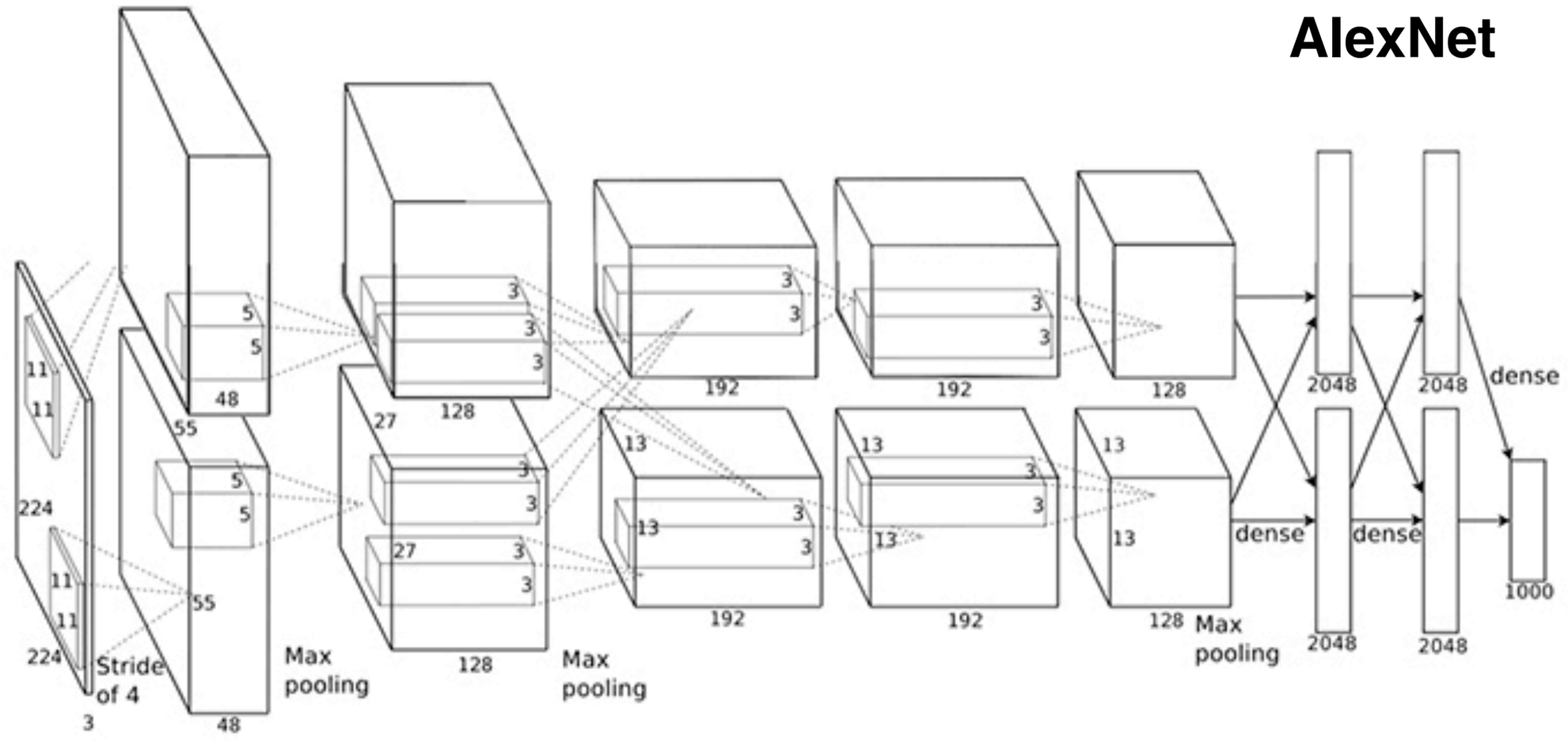
LeNet-5



Y.LeCun, L.Bottou, Y.Bengio, P.Haffner, “Gradient-based learning applied to document recognition”, Proc.IEEE 1998

Deep Neural Networks

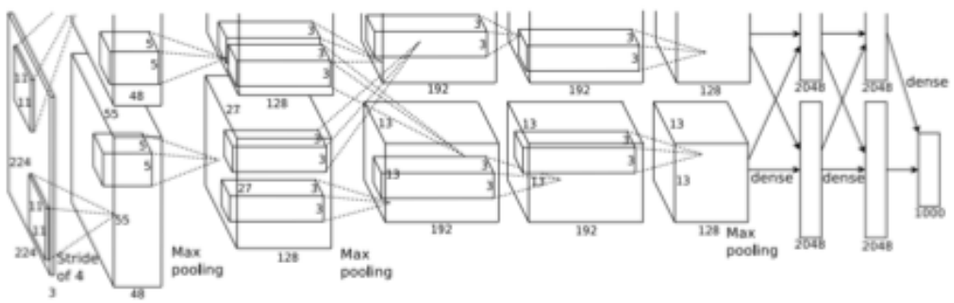
- Return of the CNN: first strong “modern” results

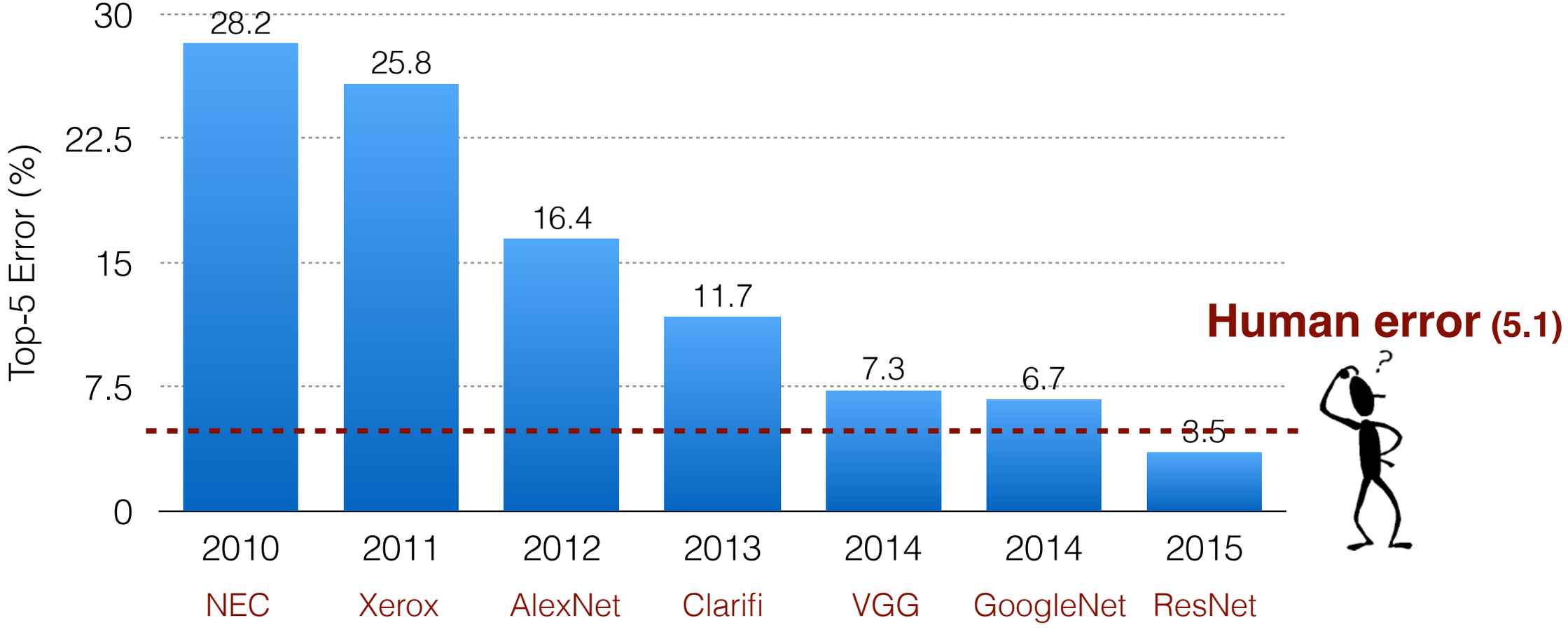


A.Krizhevsky, I.Sutskever, G.Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012

Neural networks are cool again

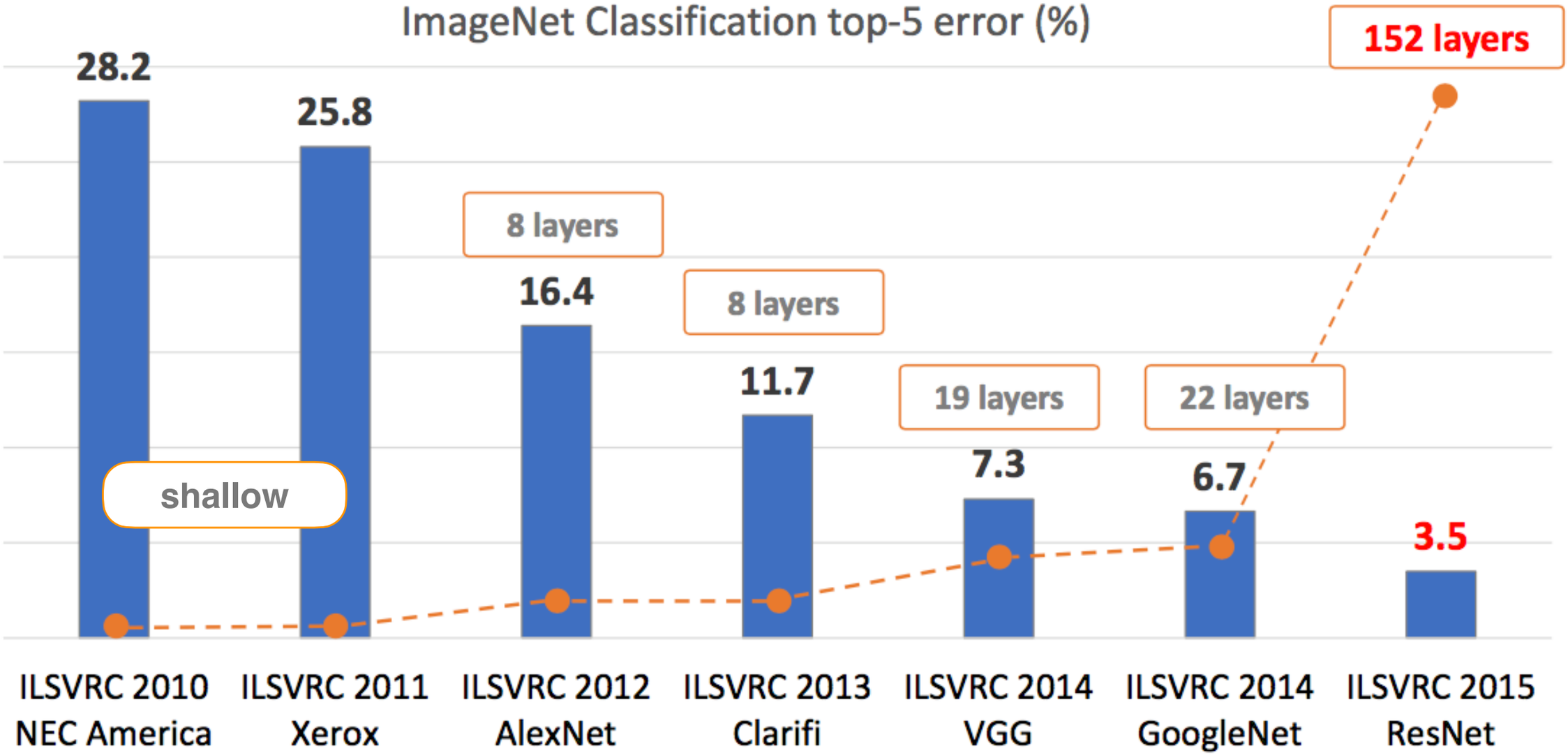
- ImageNet-ILSVRC (classification) over the years





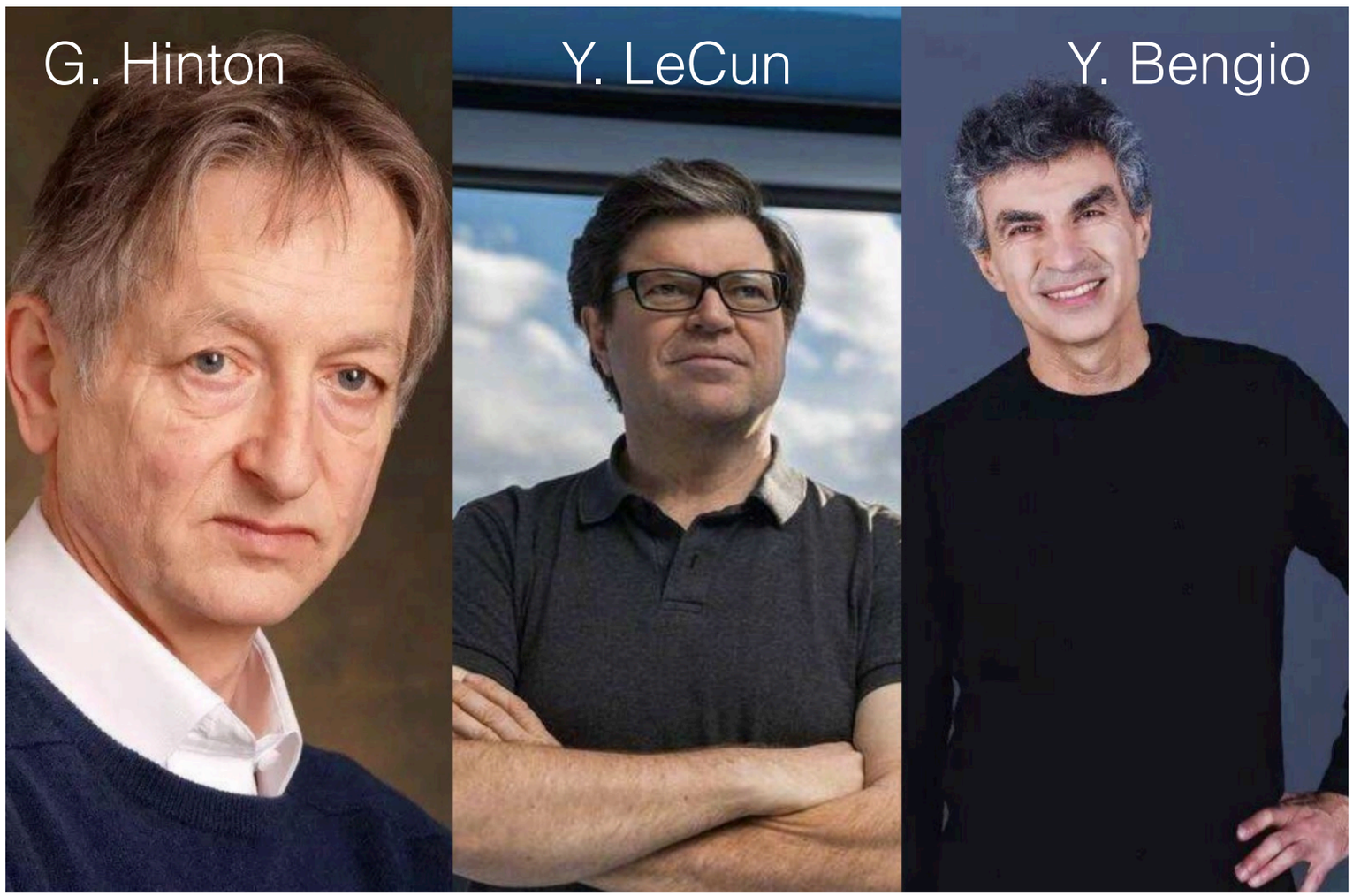
Neural networks are cool again

- Revolution of depth: how deep is deep?



Deep Learning revolution

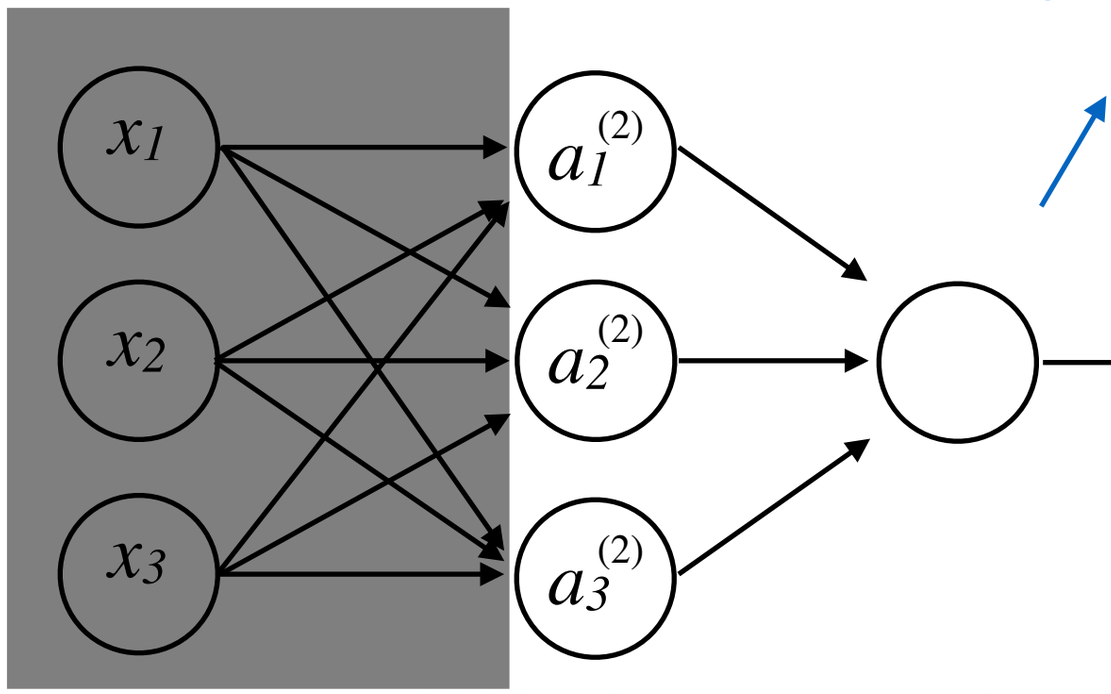
- ACM Turing Award (*i.e. the "Nobel prize" in CS*) 2018:
  "*For conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.*"



https://awards.acm.org/about/2018-turing

Recap: Feed-forward Neural Net

- 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

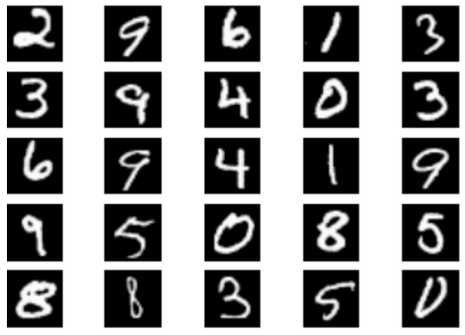
Multiple Classes

- In general, we can use two strategies to tackle a K-way multiclass classification problems
- **One-vs-one**: we train $K \cdot (K\text{-}1)/2$ binary classifiers
  - Each binary classifier is trained to discriminate between two classes; at prediction time we apply a voting scheme
- **One-vs-all** (one-vs-rest): we train a classifier per class
  - Each binary classifier is trained using its positive samples and samples belonging to all other classes as negative
  - This strategy requires each classifier to produce a real-valued confidence score for its decision; at prediction time we select the classifier with the highest confidence score

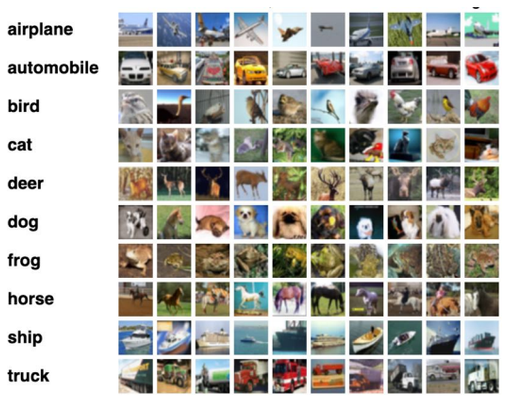
Multiple Classes

- Multiclass **is not** *multi-label* classification
  - In multi-label classification problems, multiple labels are to be predicted for each instance

MNIST



CIFAR-10

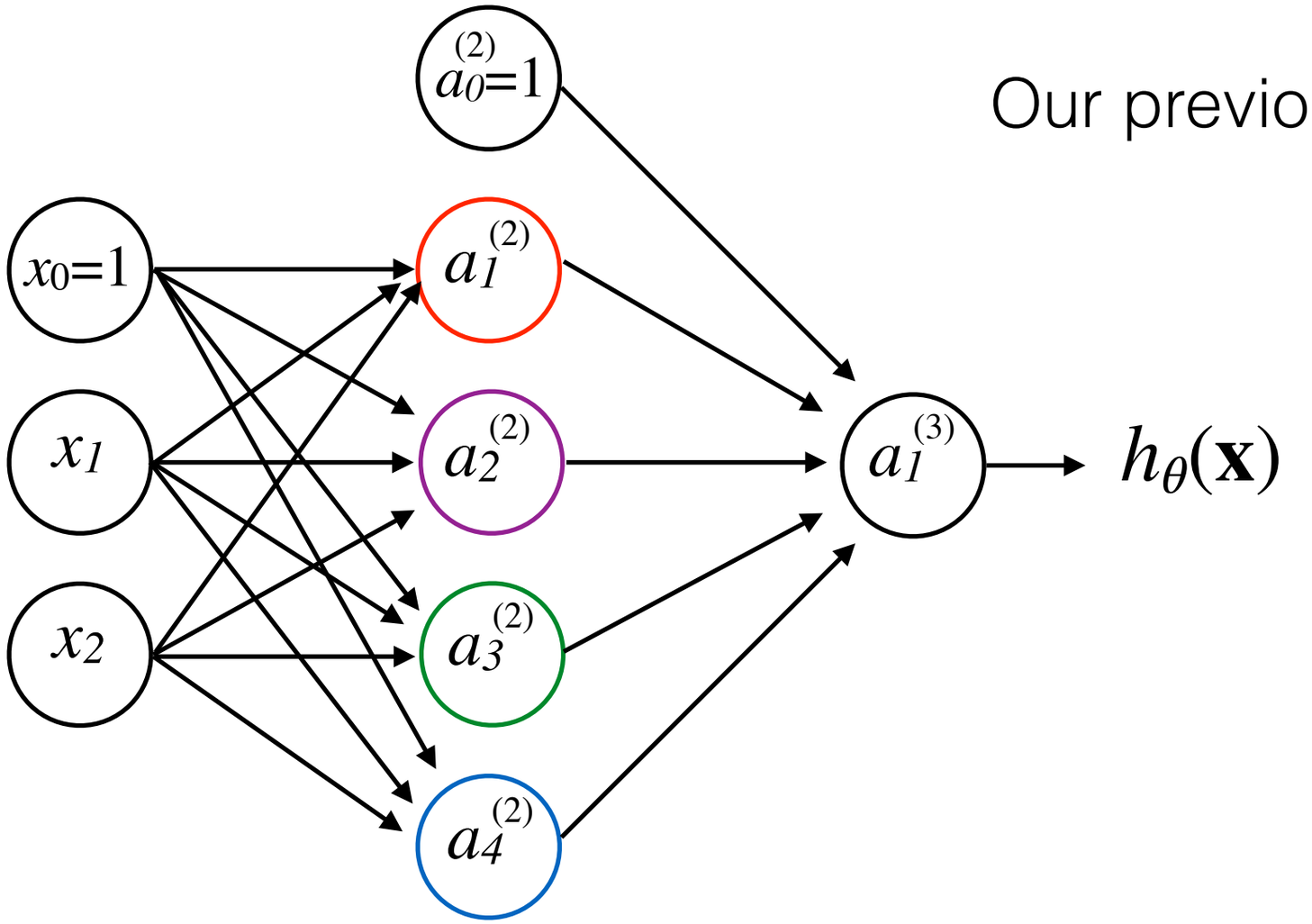


ImageNet-ILSVRC (1000 classes)

multi-label classification

Multiple Classes

- The way we do multiclass classification in neural networks is essentially one-vs-all



Our previous example:

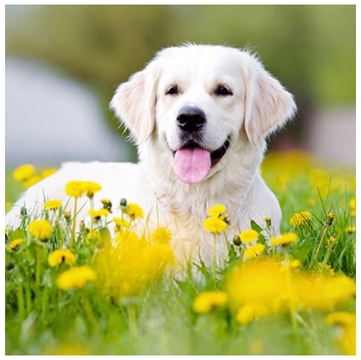
vs

“non-cat”

Multiple Classes

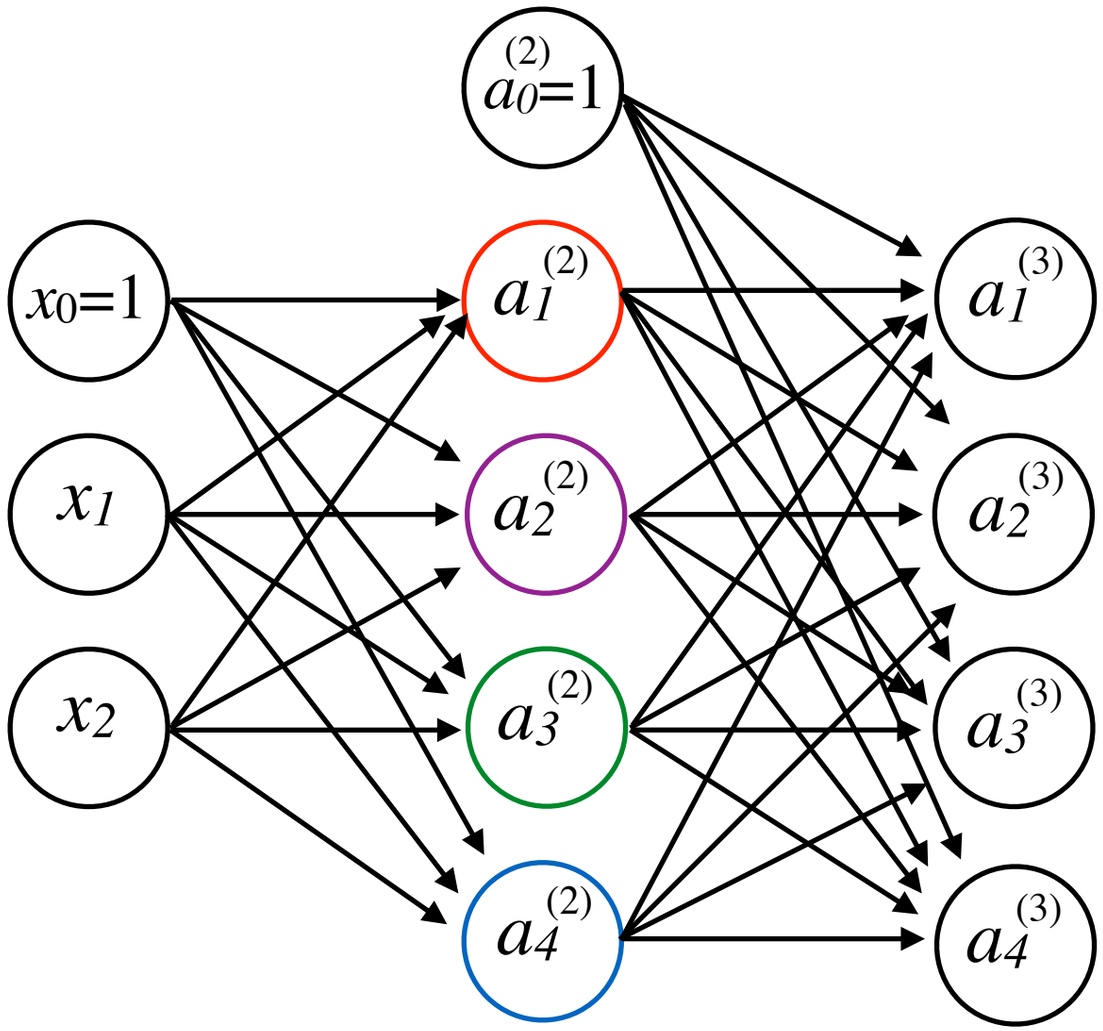
“cat”

“dog”



“fox”

“bird”



$h_\theta(\mathbf{x}) \in \mathbb{R}^4$, we want:

“cat” “dog”

$$h_\theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad h_\theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

“fox” “bird”

$$h_\theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad h_\theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

(one-hot encoding)

Neural Networks - Loss Function

- Logistic Regression cost function:

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

Our cost function is going to be a generalization of this, where instead of having just one logistic regression output unit, we have K of them

- Neural Network cost function:

$$h_\theta(\mathbf{x}) \in \mathbb{R}^K, \text{ where } (h_\theta(\mathbf{x}))_k = k^{th} \text{ output}$$

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

softmax function

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{u_l}\sum_{j=1}^{u_{l+1}} (\theta_{ji}^{(l)})^2$$

L : number of layers in the network

u_l : number of units in layer l

Parameter Learning

- We can learn our parameters with gradient descent

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \cdot \log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) \cdot \log(1 - (h_\theta(x^{(i)}))_k)\right] + \\ + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{u_l}\sum_{j=1}^{u_{l+1}} (\theta_{ji}^{(l)})^2$$

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- We need to compute: $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\boldsymbol{\theta})$

Remember, $\theta_{ij}^{(l)} \in \mathbb{R}$ connects unit j in layer l to unit i in layer l+1

Parameter Learning

- Feed-forward computation:

"forward propagation"

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

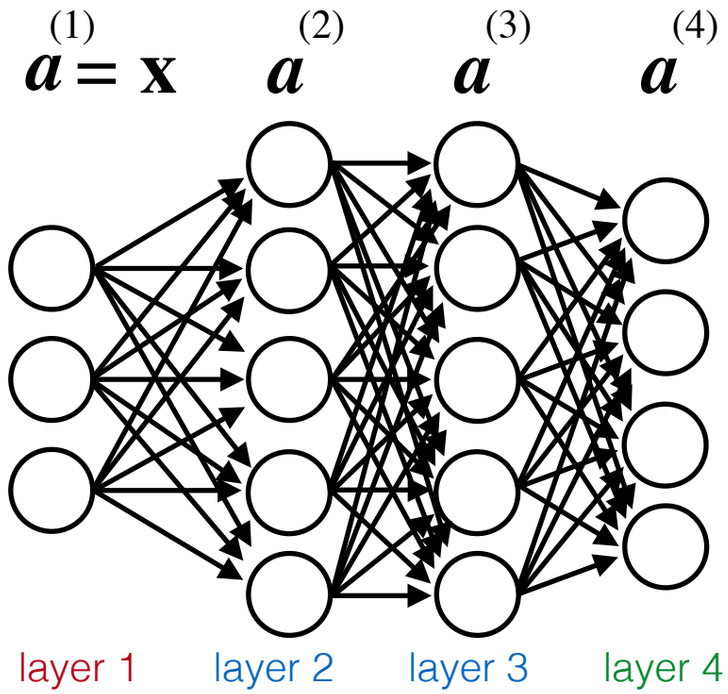
$$\boldsymbol{a}^{(2)} = f(\mathbf{z}^{(2)}) \qquad \text{(add bias: } a_0^{(2)} = 1\text{)}$$

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

$$\boldsymbol{a}^{(3)} = f(\mathbf{z}^{(3)}) \qquad \text{(add bias: } a_0^{(3)} = 1\text{)}$$

$$\mathbf{z}^{(4)} = \boldsymbol{\theta}^{(3)} \boldsymbol{a}^{(3)}$$

$$\boldsymbol{a}^{(4)} = f(\mathbf{z}^{(4)}) = h_\theta(\mathbf{x})$$



Next, in order to compute the partial derivatives, we are going to use an algorithm called "backpropagation"

Parameter Learning

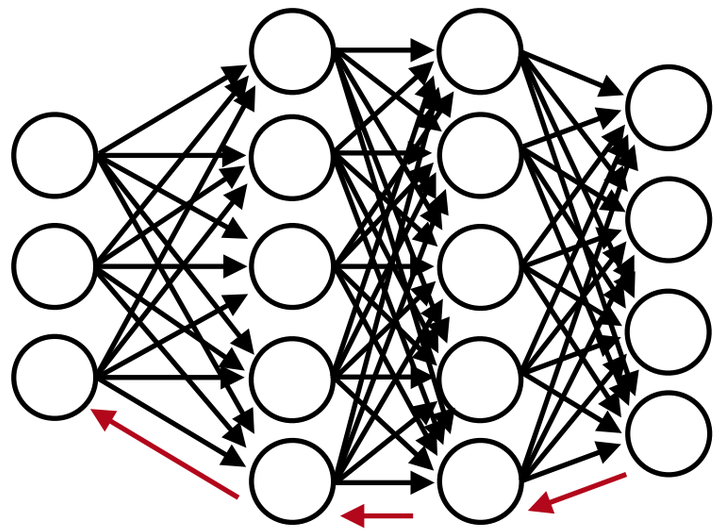
- Gradient computation:

"backpropagation"

Intuition: we shall compute $\delta_j^{(l)}$

("error" of unit j in layer l)

$a_j^{(l)}$: activation of unit j in layer l



$\delta^{(2)} \leftarrow \delta^{(3)} \leftarrow \delta^{(4)}$

Therefore, for each output unit:

(e.g. l=4 in this example)

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

Then we compute the error for the previous layers:

$$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} \; .* \; f'(z^{(3)}) \longrightarrow a^{(3)} \; .* \; (1 - a^{(3)})$$

$$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} \; .* \; f'(z^{(2)}) \quad \text{(no } \delta^{(1)}\text{)}$$

. is the element-wise multiplication*

Parameter Learning

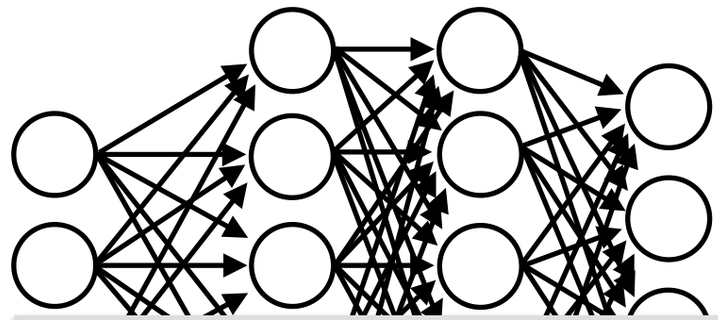
- Gradient computation:

"backpropagation"

Intuition: we shall compute $\delta_j^{(l)}$

("error" of unit j in layer l)

$a_j^{(l)}$: activation of unit j in layer l



Why is that?

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

$$f'(z) = \sigma(z)\,(1 - \sigma(z))$$

Therefore, for each output unit:

(e.g. l=4 in this example)

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

Then we compute the error for the prev

$$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} \;.*\; f'(z^{(3)}) \longrightarrow a^{(3)} \;.*\; (1 - a^{(3)})$$

$$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} \;.*\; f'(z^{(2)}) \quad \text{(no } \delta^{(1)}\text{)}$$

Backpropagation

Training examples $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Initialization: `set` $\Delta_{ij}^{(l)} = 0$ (for all i, j, l)

"Accumulators" used to compute → $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$

`for` k=1 `to` m

`set` $a^{(1)} = x^{(k)}$

`for` l=2 `to` L `compute` $a^{(l)}$ *"forward propagation"*

`compute` $\delta^{(L)} = a^{(L)} - y^{(k)}$

`compute` $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ *"backpropagation"*

The values $\delta^{(l)}$ are calculated by multiplying the delta values in the next layer with the theta matrix of layer l:

$$\delta^{(l)} = ((\theta^{(l)})^T \delta^{(l+1)}) \; .* \boxed{a^{(l)} \; .*(1 - a^{(l)})}$$

$$f'(z^{(l)})$$

Backpropagation

Training examples $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Initialization: set $\Delta_{ij}^{(l)} = 0$ (for all i, j, l)

"Accumulators" used to compute $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$

for k=1 to m

set $a^{(1)} = x^{(k)}$

for l=2 to L compute $a^{(l)}$ *"forward propagation"*

compute $\delta^{(L)} = a^{(L)} - y^{(k)}$

compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ *"backpropagation"*

compute gradients $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ *vectorization* $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

compute $D_{ij}^{(l)} := \begin{cases} \frac{1}{m}(\Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)}) & \text{, if } j \neq 0 \\ \frac{1}{m} \Delta_{ij}^{(l)} & \text{, if } j = 0 \end{cases}$

where $D_{ij}^{(l)} = \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$

Gradient Descent with Backprop

Training examples $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Initialize all $\boldsymbol{\theta}^{(l)}$ randomly (not to zero!)

`Loop/iterate:` *each iteration is called an epoch*

Initialization: `set` $\Delta_{ij}^{(l)} = 0$ (for all i, j, l)

`for` $k=1$ `to` m

`set` $\boldsymbol{a}^{(1)} = \mathbf{x}^{(k)}$, `for` $l=2$ `to` L `compute` $\boldsymbol{a}^{(l)}$ *"forward propagation"*

`compute` $\boldsymbol{\delta}^{(L)} = \boldsymbol{a}^{(L)} - y^{(k)}$

`compute` $\boldsymbol{\delta}^{(L-1)}, \boldsymbol{\delta}^{(L-2)}, \ldots, \boldsymbol{\delta}^{(2)}$ *"backpropagation"*

`compute gradients` $\boldsymbol{\Delta}^{(l)} := \boldsymbol{\Delta}^{(l)} + \boldsymbol{\delta}^{(l+1)}(\boldsymbol{a}^{(l)})^T$

`compute` $D_{ij}^{(l)} := \{ \frac{1}{m}(\Delta_{ij}^{(l)} + \lambda\, \theta_{ij}^{(l)}) \text{ , if } j \neq 0 \mid \frac{1}{m}\, \Delta_{ij}^{(l)} \text{ , if } j = 0 \}$

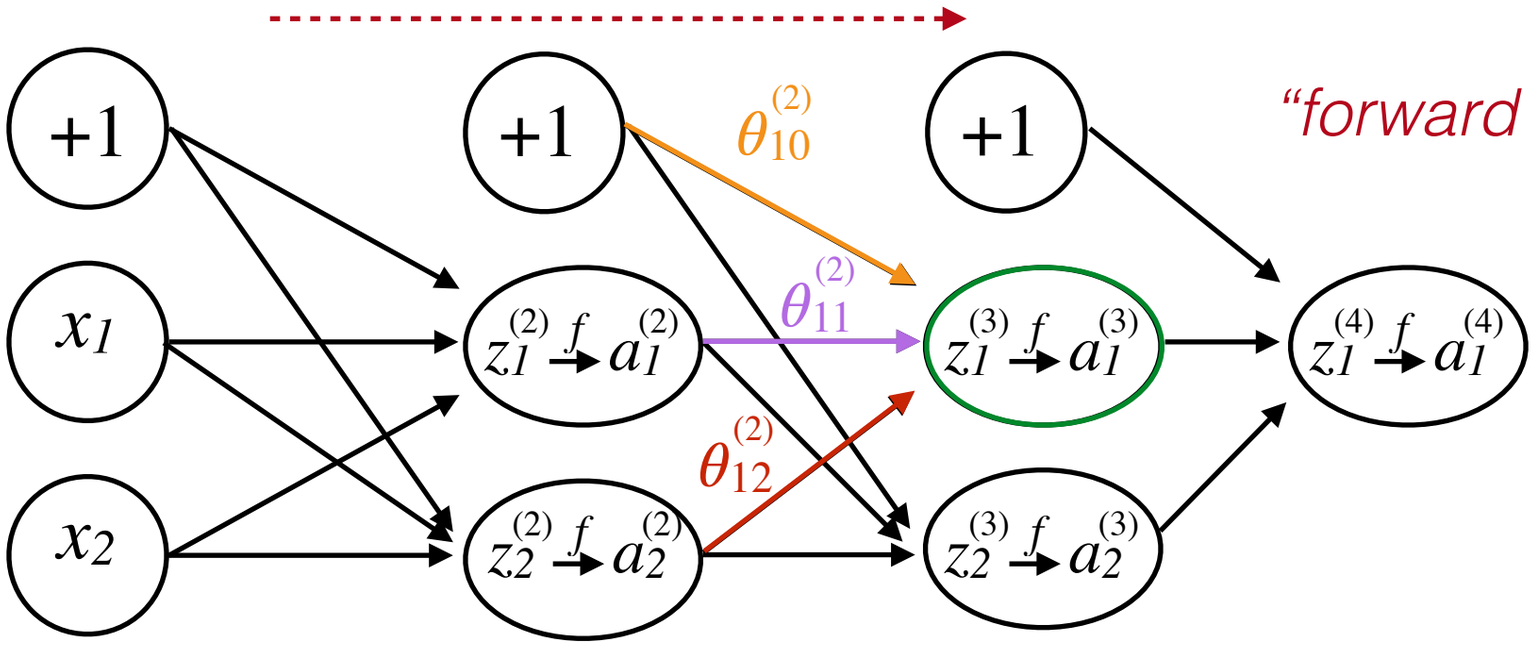
`Update weights:` $\theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \eta\, D_{ij}^{(l)}$

Backpropagation

Backpropagation Intuition

- Each unit *j* is *responsible* for a fraction of the error $\delta_{\mathrm{j}}^{(l)}$ in each of the output units to which it connects
- $\delta_{\mathrm{j}}^{(l)}$ is divided according to the strength of the connection between hidden and output units
- Then, the “blame” is propagated back to provide the error values for the hidden layer

Backpropagation Intuition



"forward propagation"

"backpropagation"

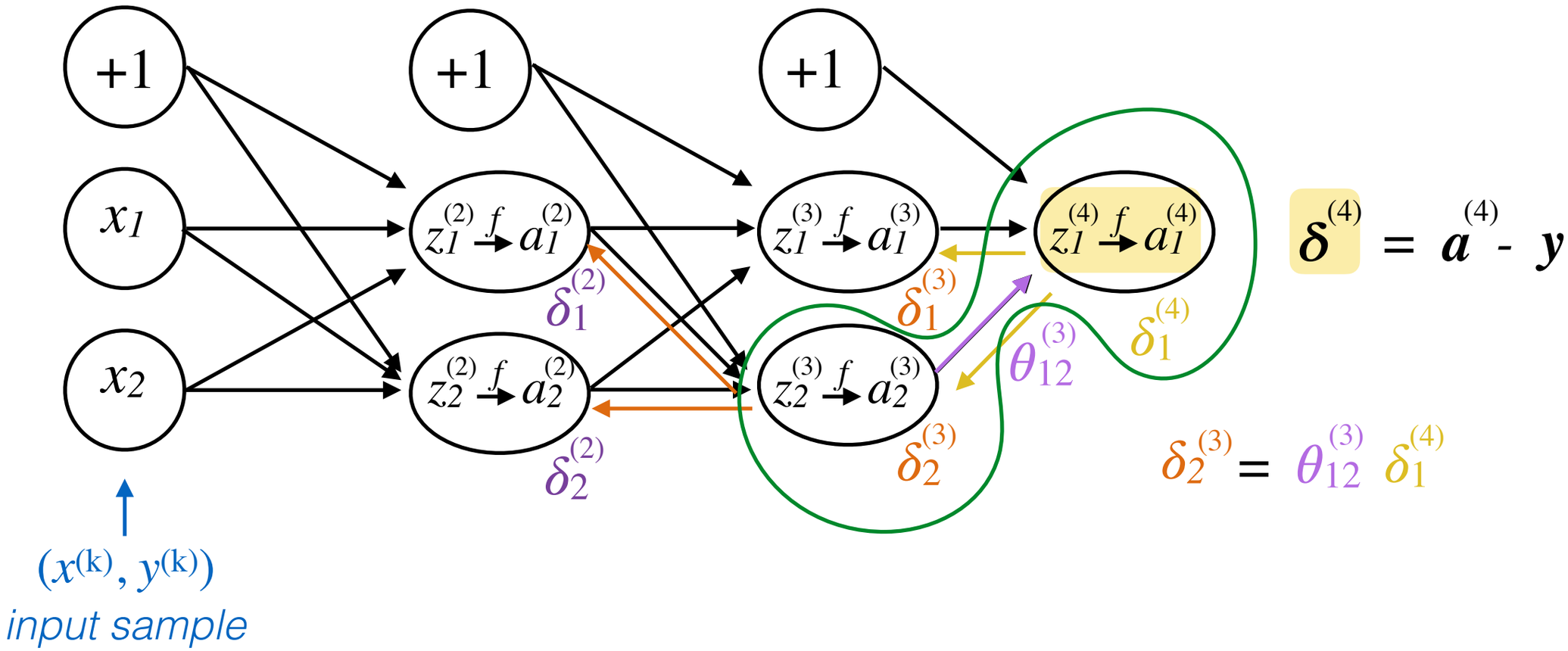
$(x^{(k)}, y^{(k)})$
input sample

$$z_1^{(3)} = \theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} \quad \rightarrow \quad a_1^{(3)} = f(z_1^{(3)})$$

Backprop intuition: we shall compute $\delta_j^{(l)}$ (error of cost for $a_j^{(l)}$, i.e. unit j in layer l)

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(k)$, $\text{cost}(k) = y^{(k)} \log h_\theta(x^{(k)}) + (1-y^{(k)}) \log(1-h_\theta(x^{(k)}))$
(for $j \geq 0$)

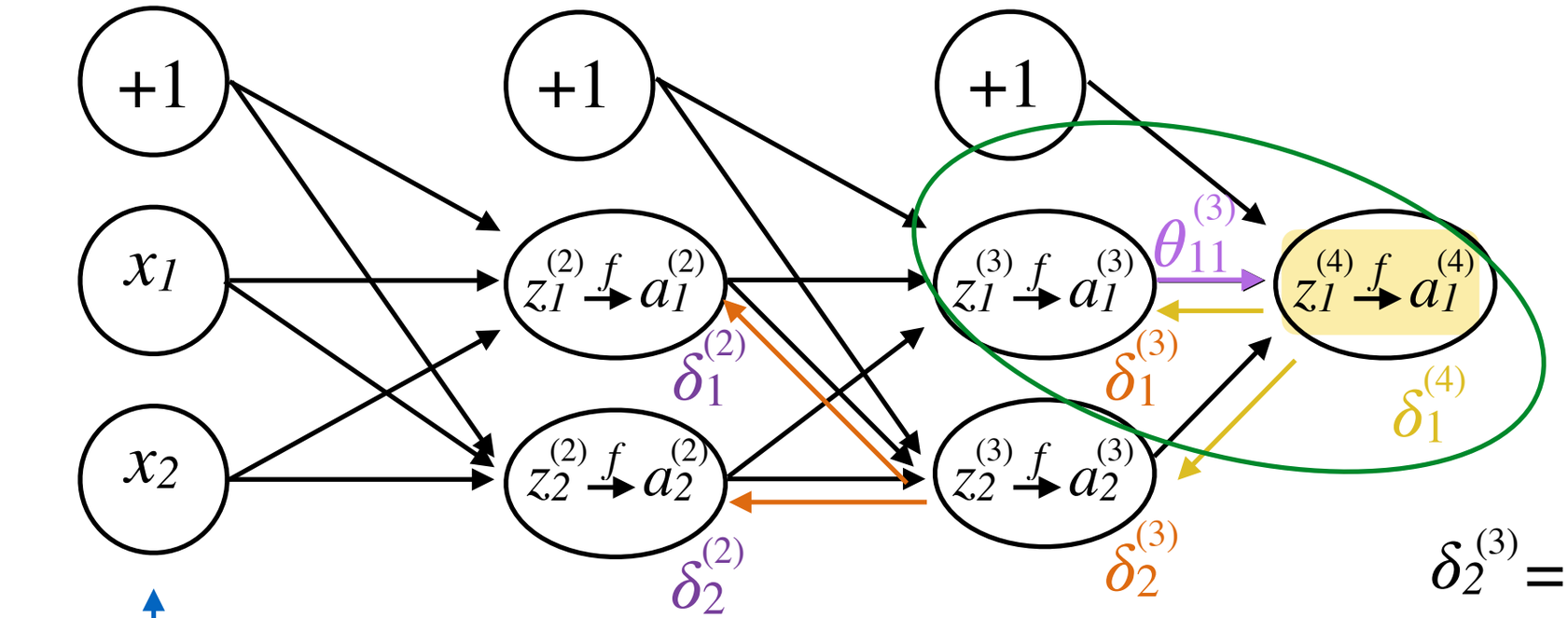
Backpropagation Intuition



Backprop intuition: we shall compute $\delta_j^{(l)}$ (error of cost for $a_j^{(l)}$, i.e. unit j in layer l)

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(k)$, $\text{cost}(k) = y^{(k)} \log h_\theta(x^{(k)}) + (1-y^{(k)}) \log(1-h_\theta(x^{(k)}))$
(for $j \geq 0$)

Backpropagation Intuition



$$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \boldsymbol{y}$$

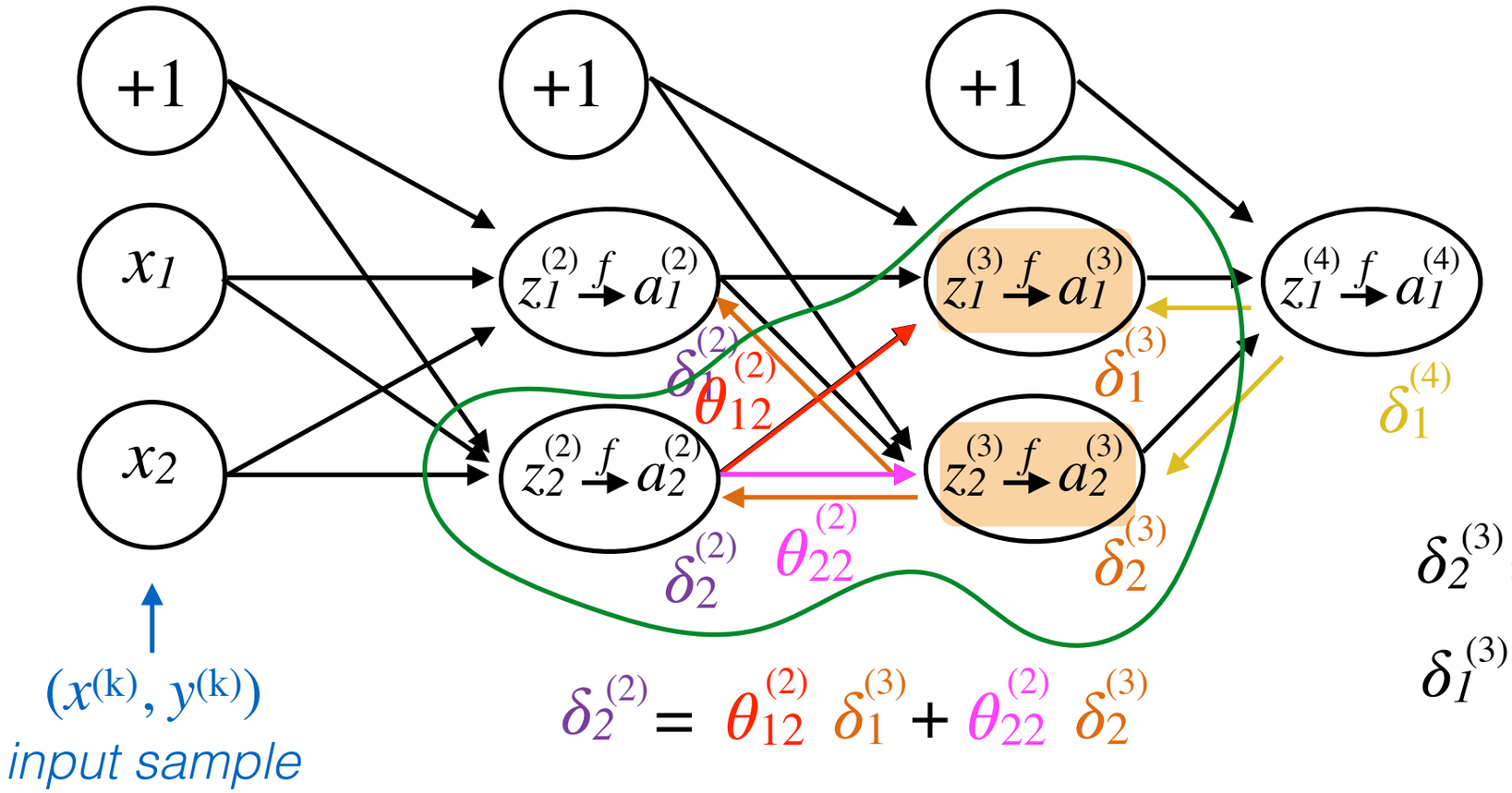
$$\delta_2^{(3)} = \theta_{12}^{(3)}\ \delta_1^{(4)}$$

$$\delta_1^{(3)} = \theta_{11}^{(3)}\ \delta_1^{(4)}$$

Backprop intuition: we shall compute $\delta_j^{(l)}$ (error of cost for $a_j^{(l)}$, i.e. unit j in layer l)

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(k)$, $\text{cost}(k) = y^{(k)} \log h_\theta(x^{(k)}) + (1-y^{(k)}) \log(1-h_\theta(x^{(k)}))$ (for $j \geq 0$)

Backpropagation Intuition



Backprop intuition: we shall compute $\delta_j^{(l)}$ (error of cost for $a_j^{(l)}$, i.e. unit j in layer l)

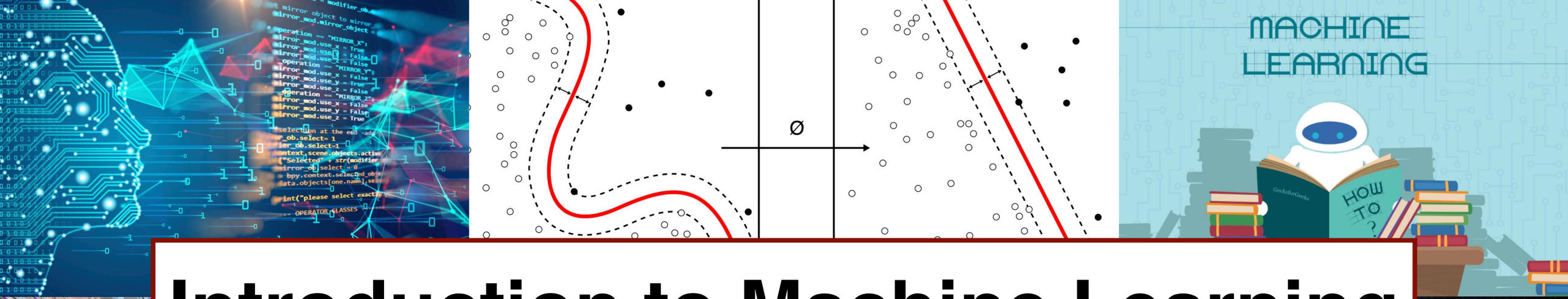
Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(k)$, $\text{cost}(k) = y^{(k)} \log h_\theta(x^{(k)}) + (1-y^{(k)}) \log(1-h_\theta(x^{(k)}))$ (for $j \geq 0$)

Backpropagation Intuition

- Hopefully that gives you a little better intuition about what backpropagation is doing

"Backprop is the cockroach of machine learning. It's ugly, and annoying, but you just can't get rid of it."

Geoffrey Hinton



Introduction to Machine Learning

SCP8084699 - LT Informatica

Artificial Neural Networks: implementation issues

Prof. Lamberto Ballan

University of Padova, A.Y. 2021/22

DIPARTIMENTO MATEMATICA
Department of Mathematics “Tullio Levi-Civita”

ANN: Implementation details

- A few practical suggestions:
  - Random initialization of the parameters
  - Unrolling the parameters (if needed), batch, activations, etc.
  - Debugging: apply gradient checking to validate your backprop implementation

- Babysitting the learning process:
  - Check the loss (training and validation curves)
  - Plot also train and val accuracy
    - Remember: a huge gap between train and val accuracy indicates that your model is overfitting

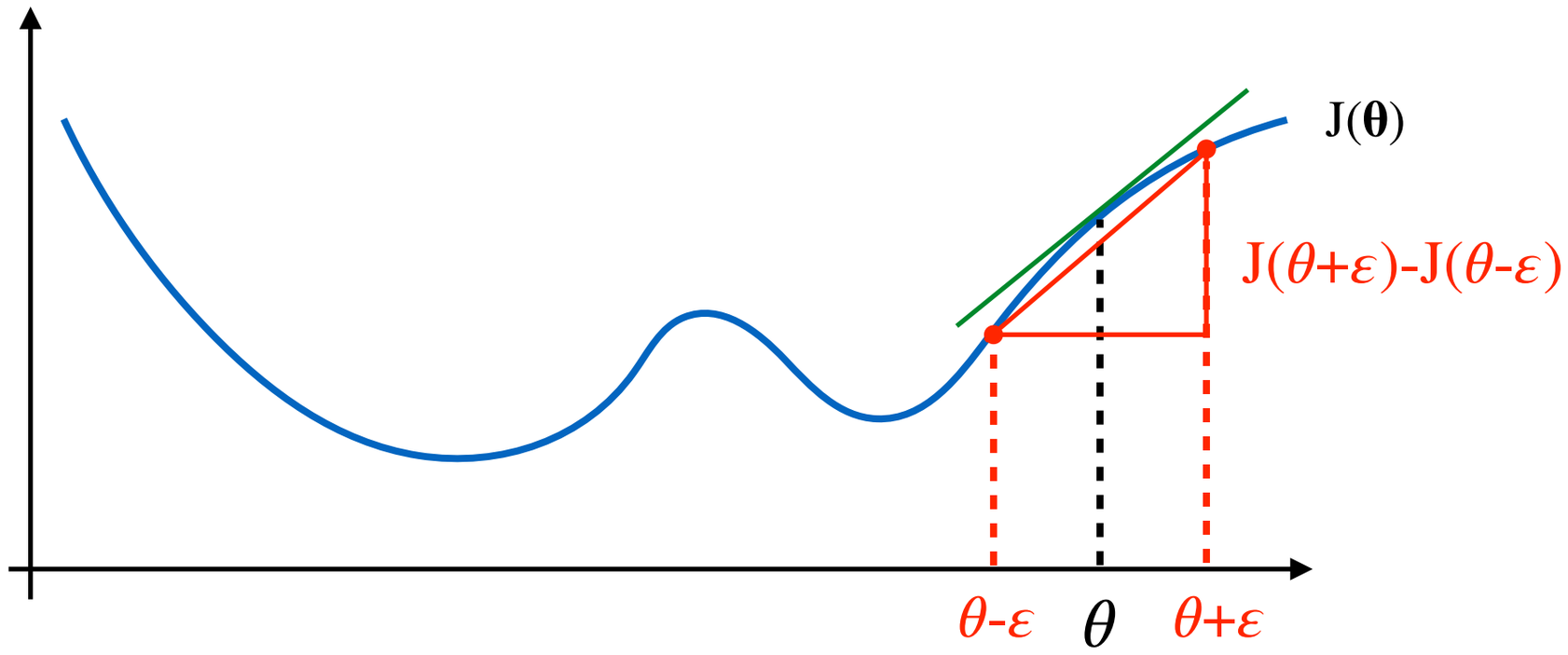
Random initialization

- Initializing all weights to zero does not work with neural networks
  - When we backpropagate, all nodes will update to the same value repeatedly
- "Symmetry breaking"
  - A random initialization of the weights solves the problem
  - We can develop a simple strategy such as:

Initialize all $\theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

Gradient checking

- It assures that backpropagation works as intended
- Numerical estimation of the gradient:



$$\frac{\mathrm{d}}{\mathrm{d}\theta} J(\boldsymbol{\theta}) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

e.g. $\varepsilon=10^{-4}$

Gradient checking

- It assures that backpropagation works as intended
- Numerical estimation of the gradient
  - Actually $\boldsymbol{\theta}$ is a matrix, we can approximate the derivative with respect to θ_j as follows:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) \approx \frac{J(\theta_1, \ldots, \theta_j + \varepsilon, \ldots, \theta_n) - J(\theta_1, \ldots, \theta_j - \varepsilon, \ldots, \theta_n)}{2\varepsilon}$$

  - We previously computed $\boldsymbol{\Delta}$, so we can check if the two gradients are very close to each other
  - Once we have verified that backprop is correct, we can remove gradient checking (it can be very slow)

References

- Stanford CS231n notes (tips and tricks):
  - https://cs231n.github.io/neural-networks-2/
  - https://cs231n.github.io/neural-networks-3/
- Y.Bengio's practical recommendation for gradient-based training of deep networks
  - https://arxiv.org/pdf/1206.5533v2.pdf
- L.Bottou's stochastic gradient descent tricks
  - https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/tricks-2012.pdf

Coming up

- **Next lecture(s)**:
  - Lab on artificial neural networks (the material has been already uploaded on Moodle)