Neural networks

Diane Lingrand

UNIVERSITÉ
CÔTE D'AZUR

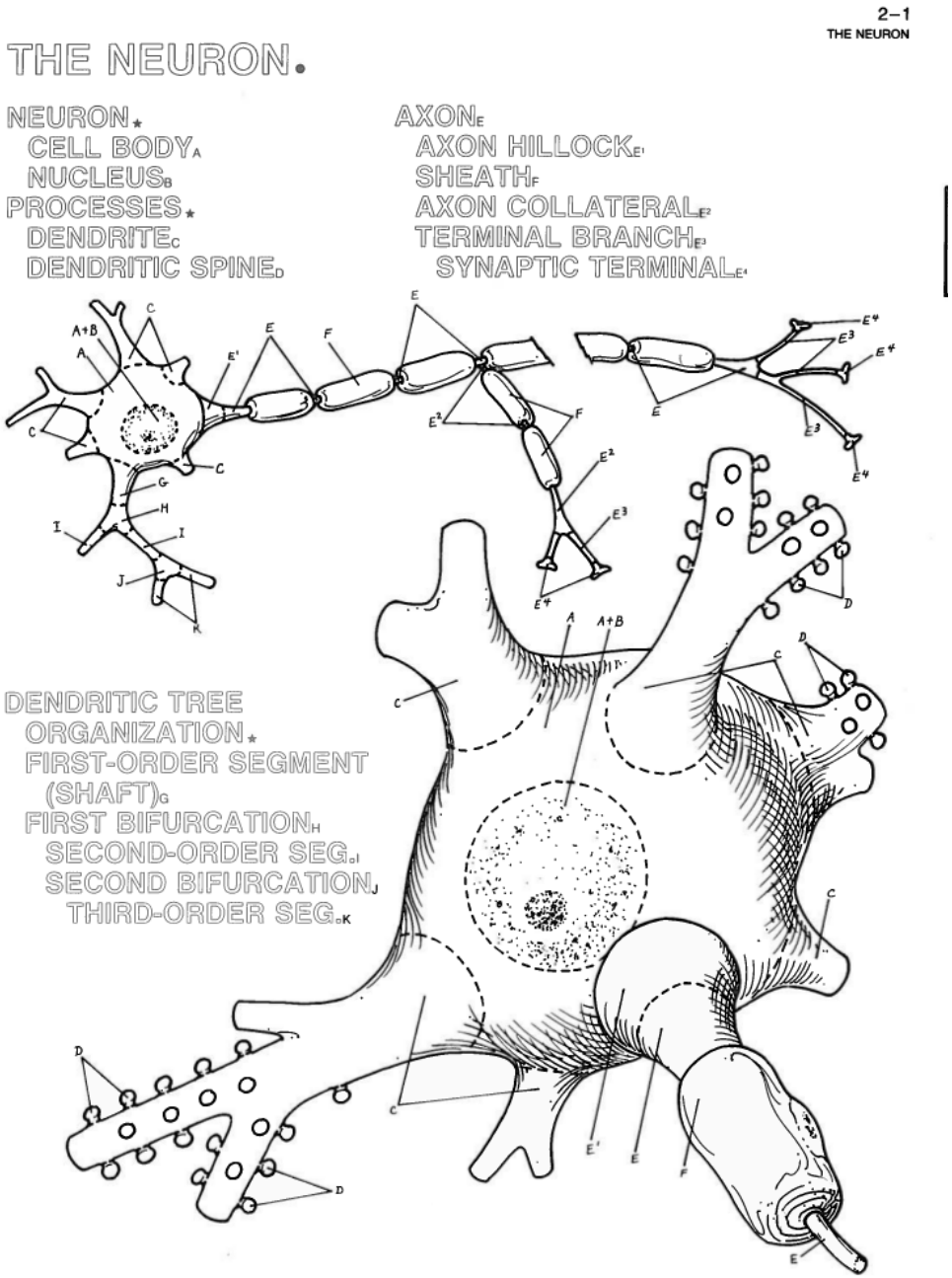
SI4

2022 - 2023

Outline

Outline

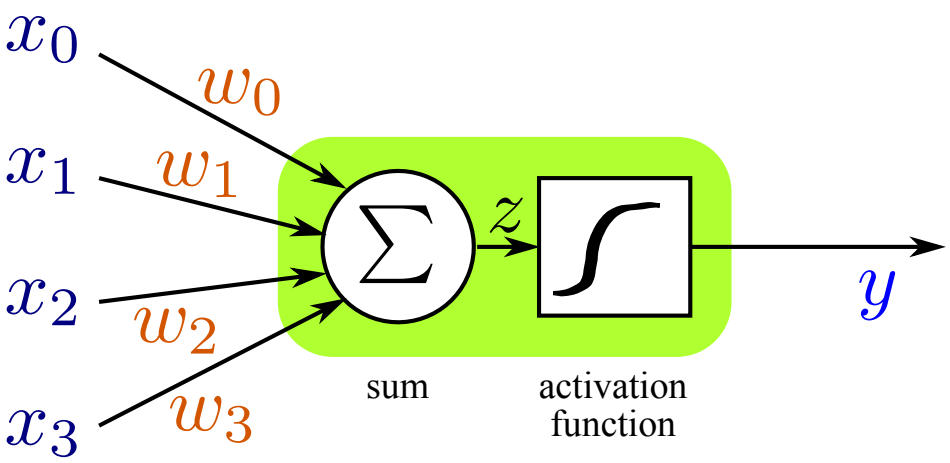
Biological neuron



- dentrites : input. multiplication with synaptic weight
- soma or cell body : summation function
- axon : activation function and output

The Human Brain Coloring Book
by Diamond etal, 1985

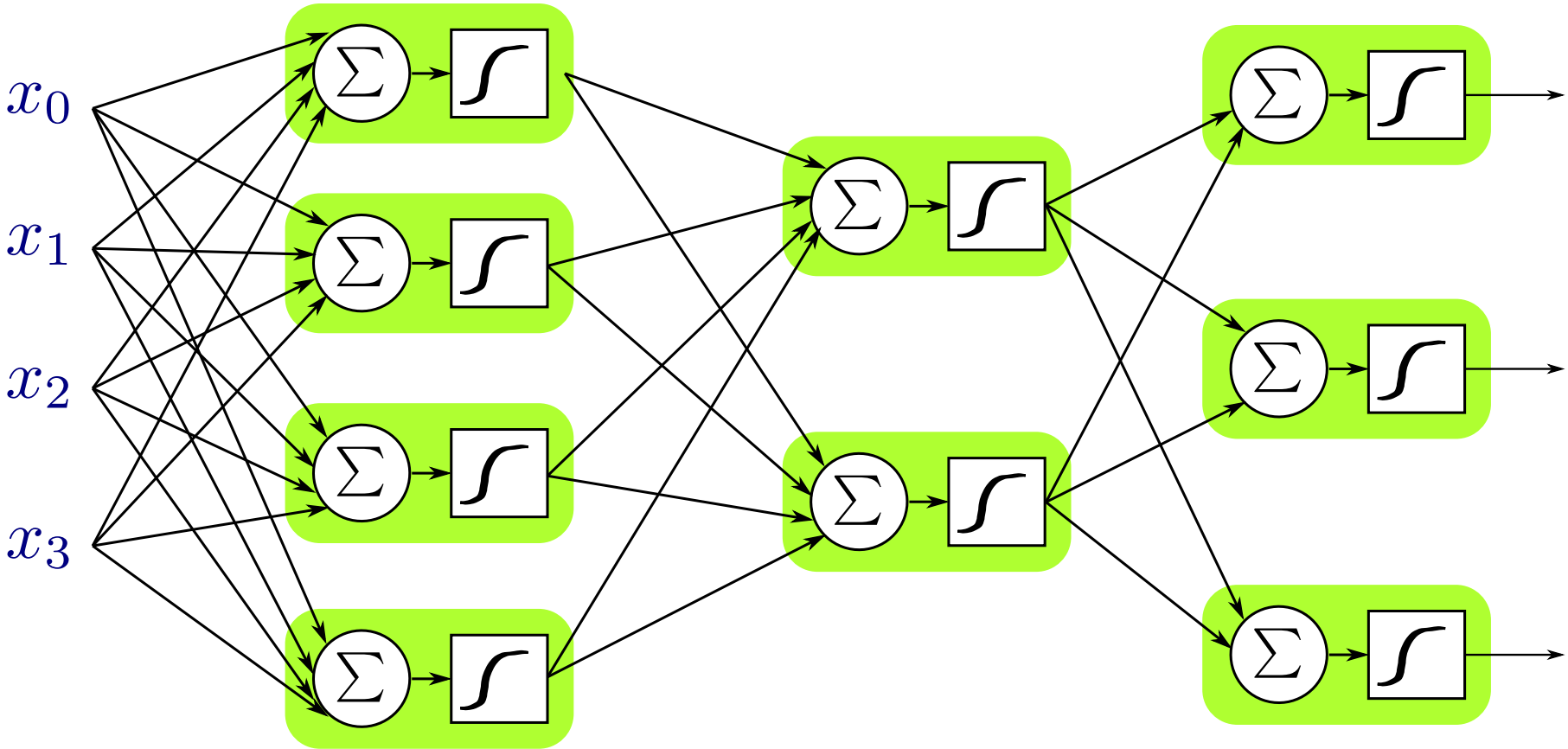
Artificial neuron



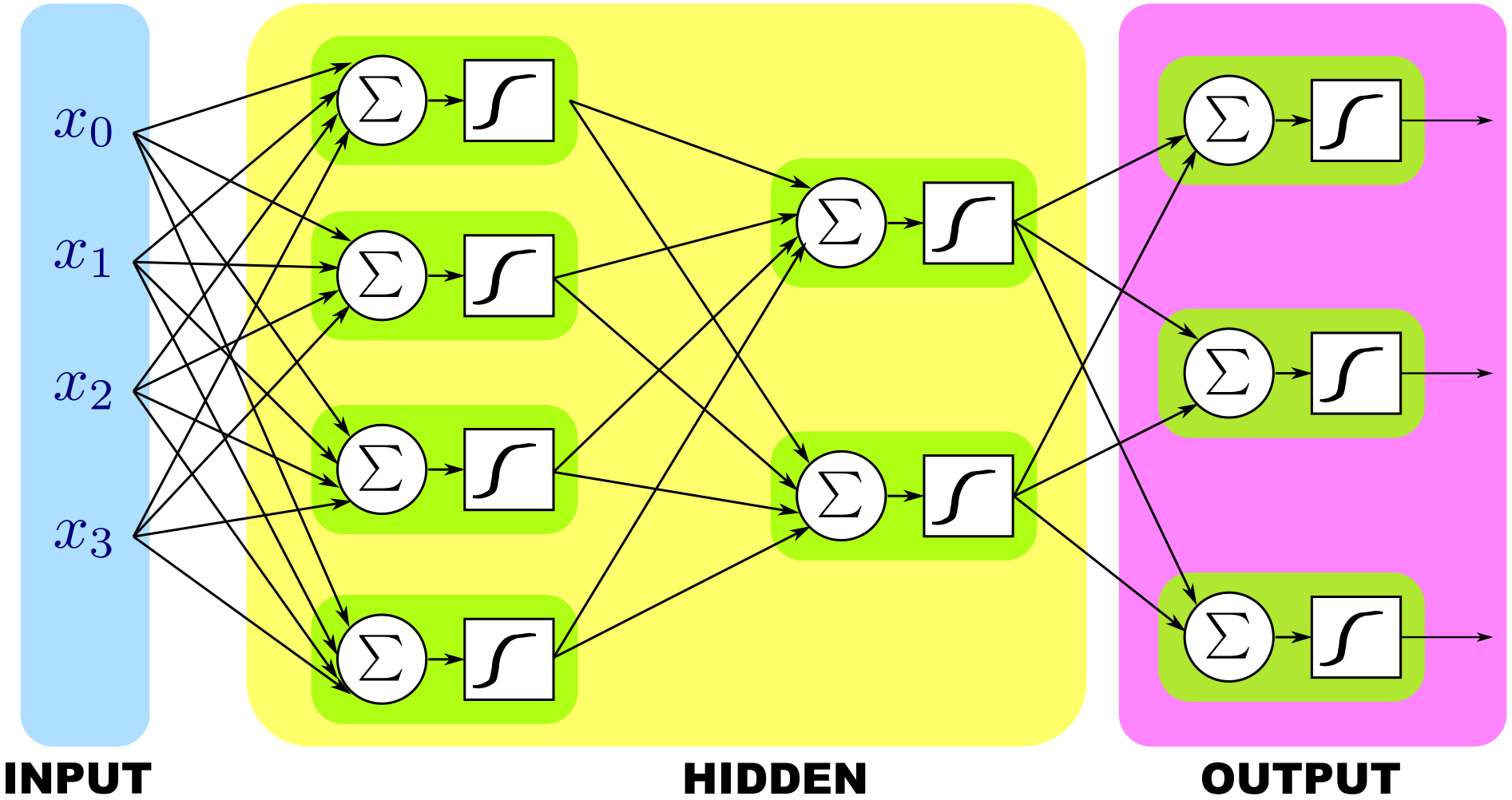
$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

- inputs x_i (also outputs of other neurons)
- associate weights w_i (synaptic modulation)
- bias : special input $x_0 = 1$ with weight w_0
- sum of the weighted inputs (cell body)
- activation function (axon hillock)
- output y

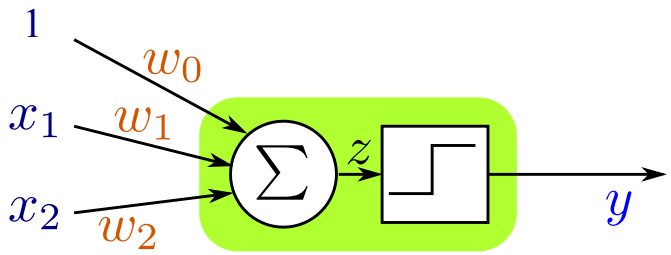
Feed forward neural network



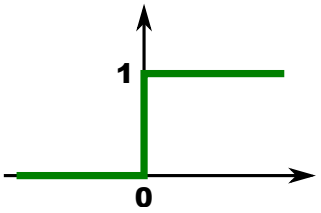
Feed forward neural network



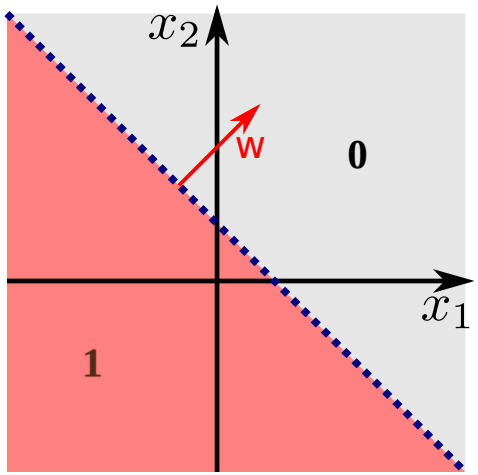
A simple single neuron



- single neuron with 3 inputs :
- activation function :

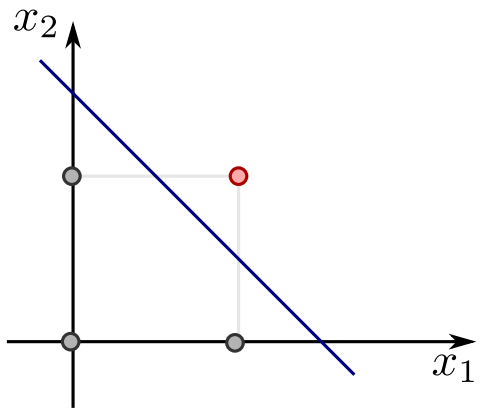


- output : $y = s(z) = s(w_0 + w_1 x_1 + w_2 x_2)$
- equivalent to divide the 2D plane by a line of equation : $w_0 + w_1 x_1 + w_2 x_2 = 0$ or w.x $= 0$



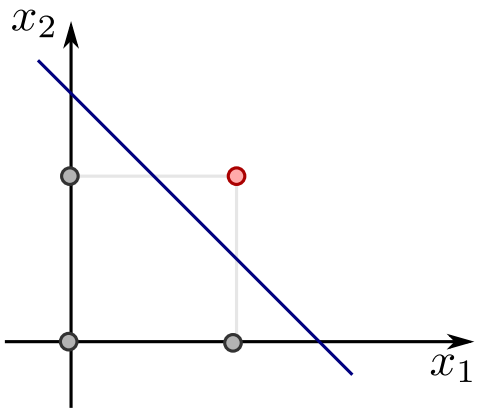
Boolean operation AND

AND	0	1
0	0	0
1	0	1

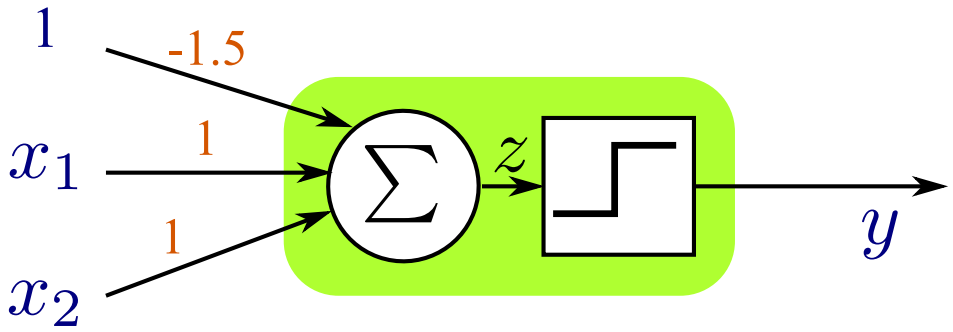


Boolean operation AND

AND	0	1
0	0	0
1	0	1

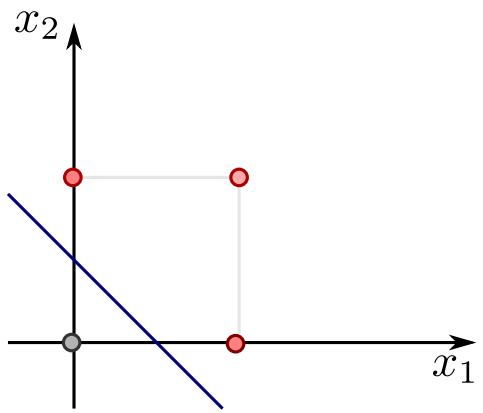


- A solution : $-1.5 + x_1 + x_2 = 0$



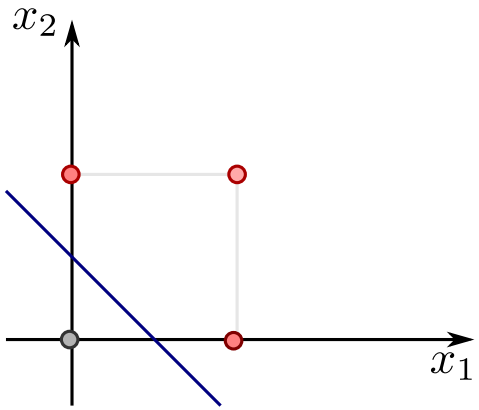
Boolean operation OR

OR	0	1
0	0	1
1	1	1

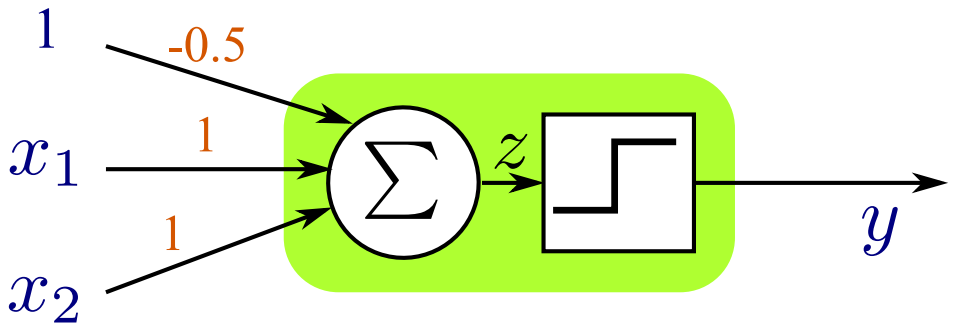


Boolean operation OR

OR	0	1
0	0	1
1	1	1

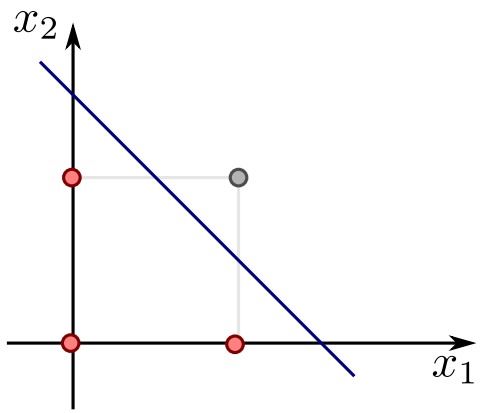


- A solution : $-0.5 + x_1 + x_2 = 0$



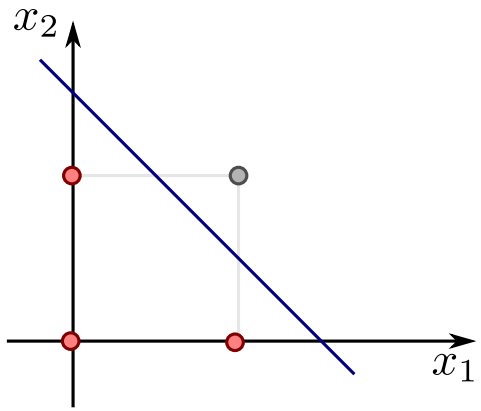
Boolean operation NAND

NAND	0	1
0	1	1
1	1	0

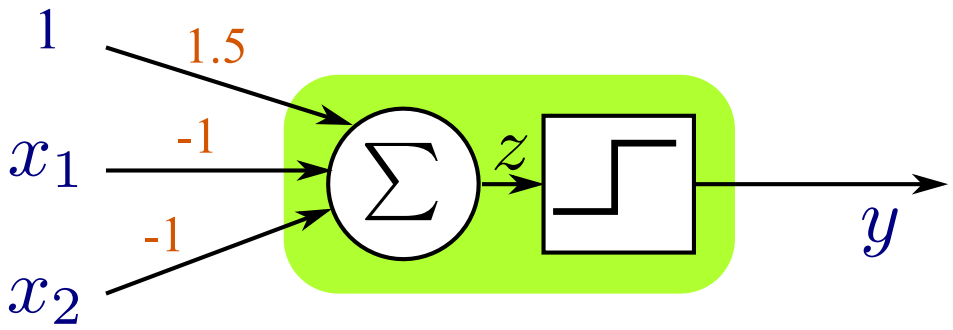


Boolean operation NAND

NAND	0	1
0	1	1
1	1	0

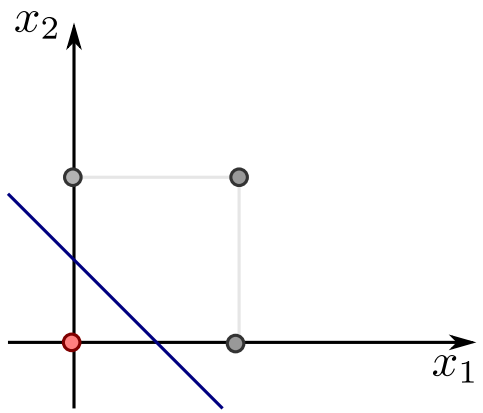


- A solution : $1.5 - x_1 - x_2 = 0$



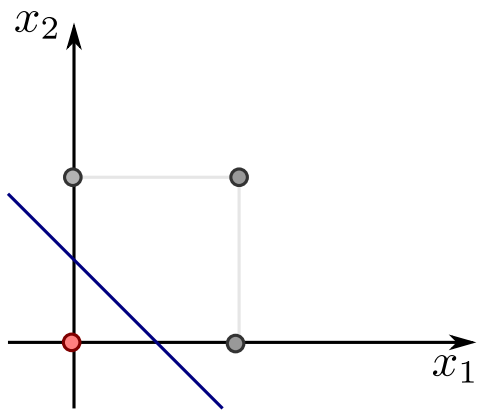
Boolean operation NOR

NOR	0	1
0	0	0
1	1	0

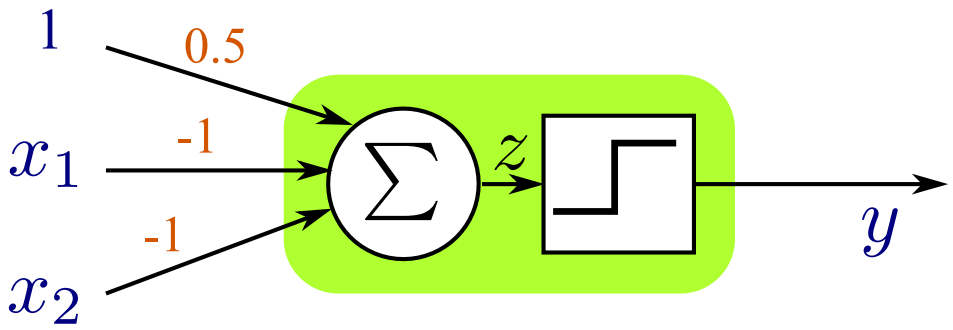


Boolean operation NOR

NOR	0	1
0	0	0
1	1	0

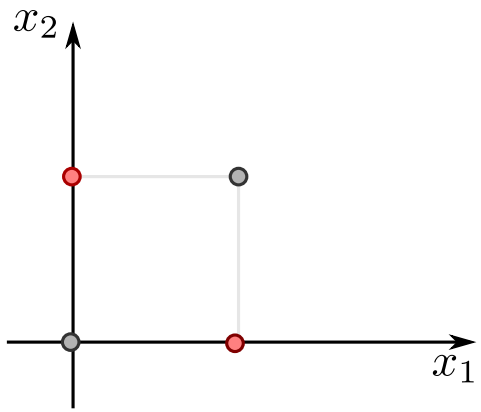


- A solution : $0.5 - x_1 - x_2 = 0$



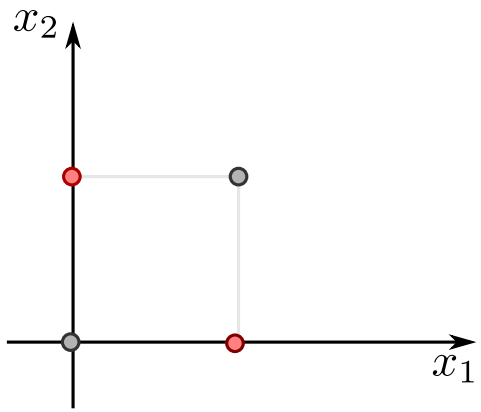
Boolean operation XOR

XOR	0	1
0	0	1
1	1	0



Boolean operation XOR

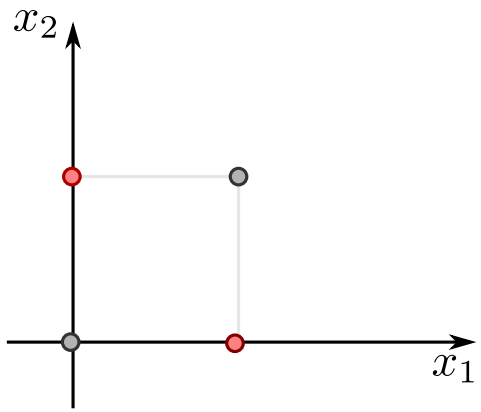
XOR	0	1
0	0	1
1	1	0



- $\Rightarrow$ Impossible with a single neuron !
- { NAND } and { NOR } are functionally complete
- Solution : express XOR using a combination of boolean operators :
  - `[ A NAND ( A NAND B ) ] NAND [ B NAND ( A NAND B ) ]`
  - `[ ( A NOR A ) NOR ( B NOR B ) ] NOR ( A NOR B )`

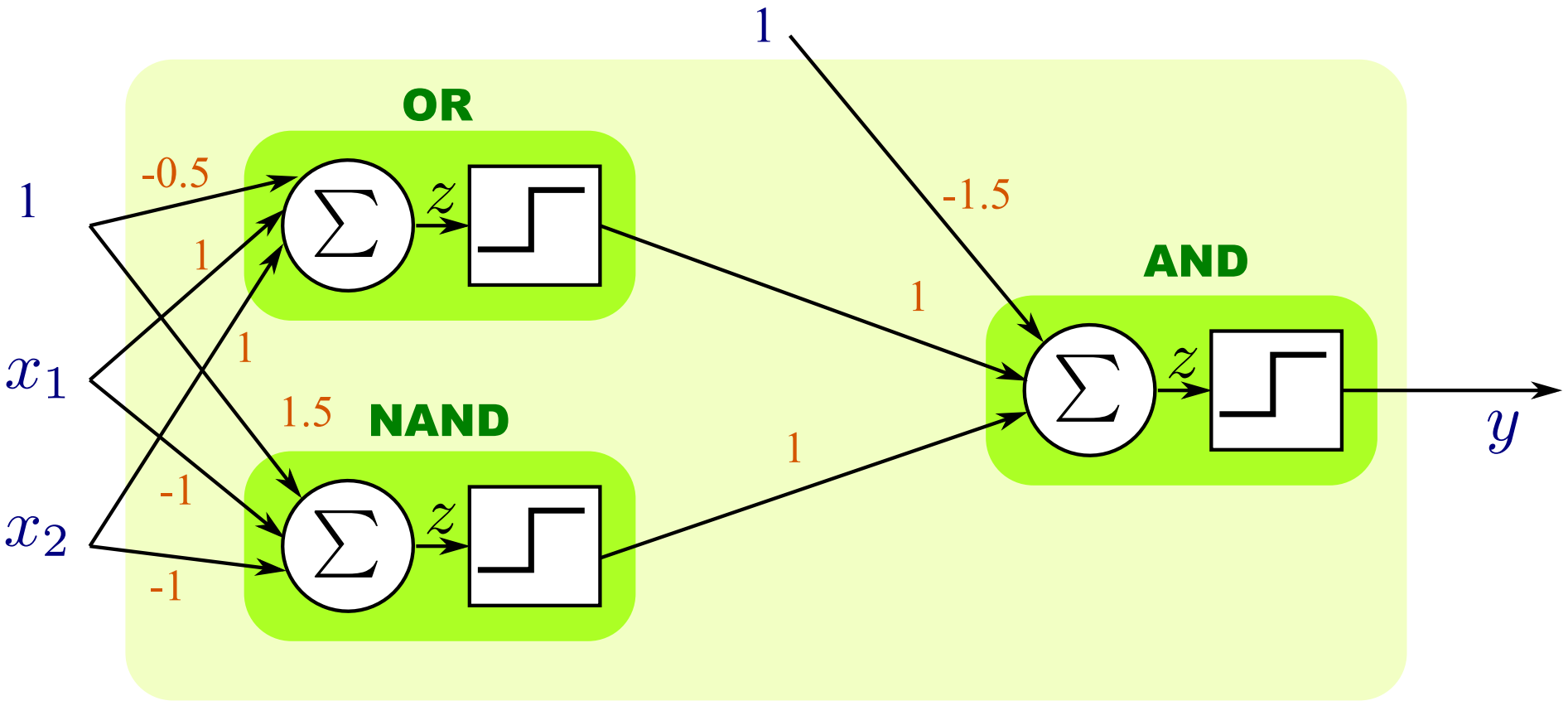
Boolean operation XOR

XOR	0	1
0	0	1
1	1	0



- $\Rightarrow$ Impossible with a single neuron !
- { NAND } and { NOR } are functionally complete
- Solution : express XOR using a combination of boolean operators :
  - `[ A NAND ( A NAND B ) ] NAND [ B NAND ( A NAND B ) ]`
  - `[ ( A NOR A ) NOR ( B NOR B ) ] NOR ( A NOR B )`
  - `A XOR B = (A NOR B) AND (A OR B)`

A XOR B = (A NOR B) AND (A OR B)

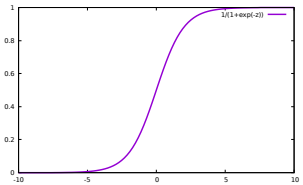


- It is proven that (Cybenko and followers) :
  - all boolean functions can be represented using a single hidden layer
  - all borned continuous functions can be represented using a single hidden layer, with an arbitrary precision
  - all functions can be represented using 2 hidden layers, with an arbitrary precision
- But :
  - How to determine the topology of the neural network ?
  - How many neurons per layer ?
  - How to tune the weights ? (learning phase)

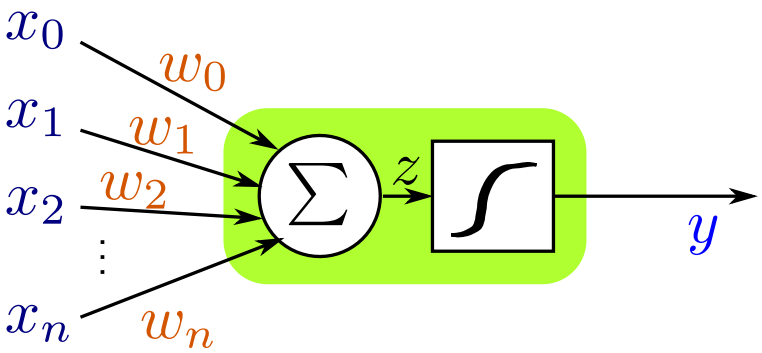
Outline

Activation function

- For the use of gradient, the activation function must be continuous and differentiable.
- Sigmoids (or S-like)
  - logistic function $s(z) = \frac{1}{1+e^{-z}}$ and $s'(z) = \frac{ds}{dz}(z) = \frac{e^{-z}}{(1+e^{-z})^2} = s(z)(1 - s(z))$



Back to the single neuron



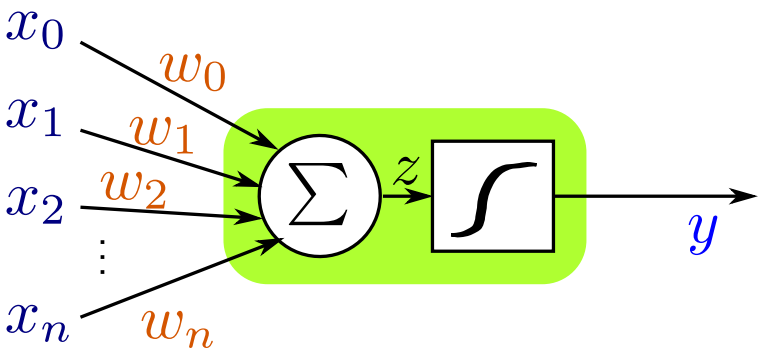
with

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

For a given data j, the output of the neuron $y(\mathrm{w}, \mathrm{x}^j)$ should be equal to y^j.

$$E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$$

Back to the single neuron



with

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

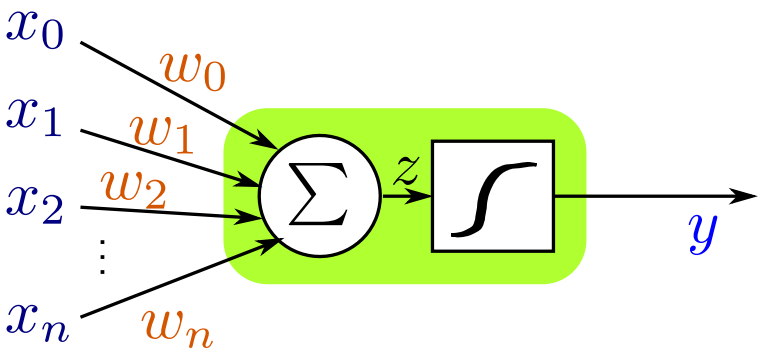
For a given data j, the output of the neuron $y(\mathrm{w}, \mathrm{x}^j)$ should be equal to y^j.

$$E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$$

Gradient descent :

$$\nabla_{\mathrm{w}} E = \left[\frac{\partial E}{\partial w_0} \; \frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_n}\right]$$

Back to the single neuron



with

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i\right)$$

For a given data j, the output of the neuron $y(\mathrm{w}, \mathrm{x}^j)$ should be equal to y^j.

$$E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$$

Gradient descent :

$$\nabla_{\mathrm{w}} E = \left[\frac{\partial E}{\partial w_0} \; \frac{\partial E}{\partial w_1} \cdots \frac{\partial E}{\partial w_n}\right] = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j)$$

where

$$\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \; \frac{\partial y}{\partial w_1} \cdots \frac{\partial y}{\partial w_n}\right]$$

Back to the single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n} \right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i \right)$$

Back to the single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n} \right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i \right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) =$$

Back to the single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n} \right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i \right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}(\mathrm{w}, \mathrm{x}^j)$$

Back to the single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n} \right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i \right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = s'(z) \underbrace{\frac{\partial \left(\sum_{k=0}^{n} w_k x_k^j \right)}{\partial w_i}}_{x_i^j}$$

Back to the single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n} \right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i \right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = s'(z) \underbrace{\frac{\partial \left(\sum_{k=0}^{n} w_k x_k^j \right)}{\partial w_i}}_{x_i^j} = y(1-y) x_i^j$$

with $y = y(\mathrm{w}, \mathrm{x}^j)$.

Back to the single neuron (2)

We compute :

$$\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j) \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) \text{ where } \nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j) = \left[\frac{\partial y}{\partial w_0} \ \frac{\partial y}{\partial w_1} \dots \frac{\partial y}{\partial w_n} \right]$$

knowing that :

$$y = s(z) = s\left(\sum_{i=0}^{n} w_i x_i \right)$$

$$\frac{\partial y}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}(\mathrm{w}, \mathrm{x}^j) = s'(z) \underbrace{\frac{\partial \left(\sum_{k=0}^{n} w_k x_k^j \right)}{\partial w_i}}_{x_i^j} = y(1-y) x_i^j$$

with $y = y(\mathrm{w}, \mathrm{x}^j)$. Thus :

$$\nabla_{\mathrm{w}} y = y(1-y)\mathrm{x}^{\mathrm{j}}$$

Back to the single neuron (3)

We obtained :

$$\nabla_{\mathrm{w}} E = (y - y^j) y (1 - y) \mathrm{x}^{\mathrm{j}}$$

Back to the single neuron (3)

We obtained :

$$\nabla_{\mathrm{w}} E = (y - y^j) y (1 - y) \mathrm{x}^{\mathrm{j}}$$

Let us note :

$$\delta = (y - y^j) y (1 - y)$$

Gradient descent rule :

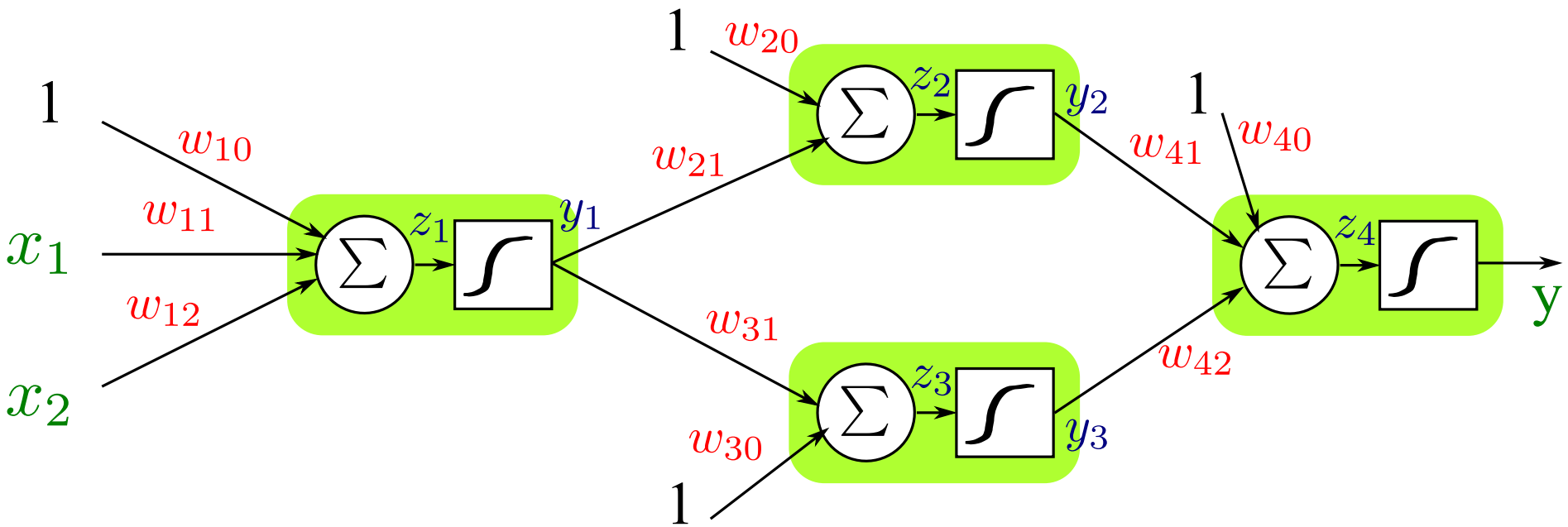
$$\mathrm{w} \leftarrow \mathrm{w} - \eta \, \delta \, \mathrm{x}^{\mathrm{j}}$$

Outline

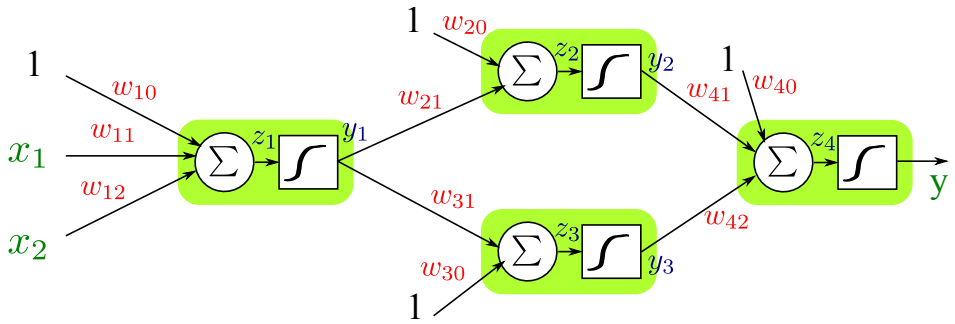
Multilayer feed-forward neural network as a universal approximator

- It is proven that (Cybenko and followers) :
  - all boolean functions can be represented using a single hidden layer
  - all borned continuous functions can be represented using a single hidden layer, with an arbitrary precision
  - all functions can be represented using 2 hidden layers, with an arbitrary precision
- But :
  - How to determine the topology of the neural network ?
  - How many neurons per layer ?
  - How to tune the weights ? (learning phase)

Training a multilayer perceptron

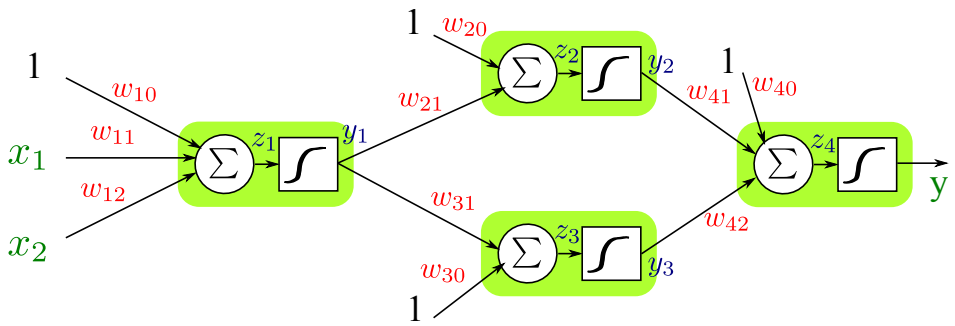


Forward propagation



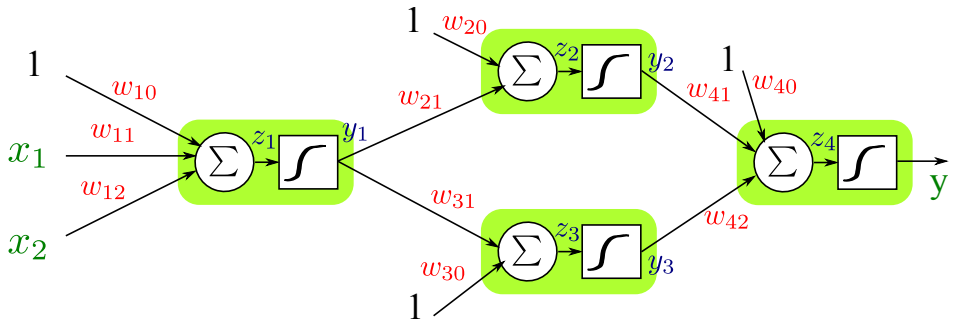
- $y_1 = s(z_1) = s(w_{10} + w_{11}x_1 + w_{12}x_2)$

Forward propagation



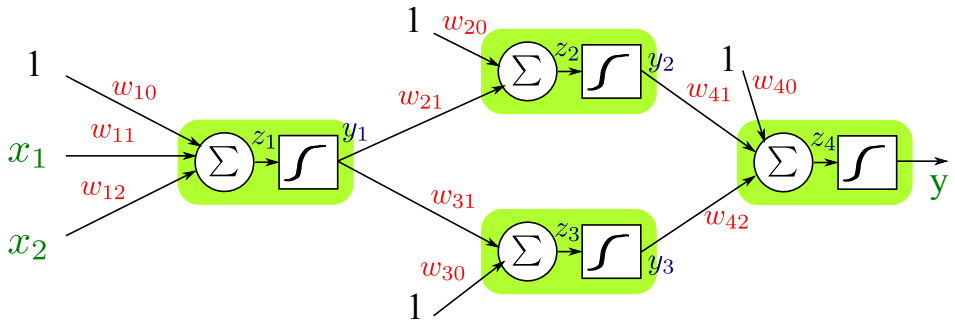
- $y_1 = s(z_1) = s(w_{10} + w_{11}x_1 + w_{12}x_2)$
- $y_2 = s(z_2) = s(w_{20} + w_{21}y_1)$

Forward propagation



- $y_1 = s(z_1) = s(w_{10} + w_{11}x_1 + w_{12}x_2)$
- $y_2 = s(z_2) = s(w_{20} + w_{21}y_1)$
- $y_3 = s(z_3) = s(w_{30} + w_{31}y_1)$

Forward propagation



- $y_1 = s(z_1) = s(w_{10} + w_{11}x_1 + w_{12}x_2)$
- $y_2 = s(z_2) = s(w_{20} + w_{21}y_1)$
- $y_3 = s(z_3) = s(w_{30} + w_{31}y_1)$
- $y = s(z_4) = s(w_{40} + w_{41}y_2 + w_{42}y_3)$

Error minimization

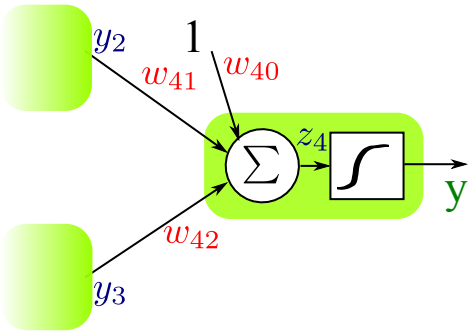
- Error : $E = \frac{1}{2}(y(\mathrm{w}, x^j) - y^j)^2$ with

  $\mathrm{w} = [w_{10}\ w_{11}\ w_{12}\ w_{20}\ w_{21}\ w_{30}\ w_{31}\ w_{40}\ w_{41}\ w_{42}]$ and $x^j = [x_1\ x_2]$
- Minimization : computation of the gradient

  $\nabla_{\mathrm{w}} E = (y(\mathrm{w}, x^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, x^j)$ where $\nabla_{\mathrm{w}} y = \left[\frac{\partial y}{\partial w_{10}}\ \frac{\partial y}{\partial w_{11}} \cdots \frac{\partial y}{\partial w_{42}}\right]$

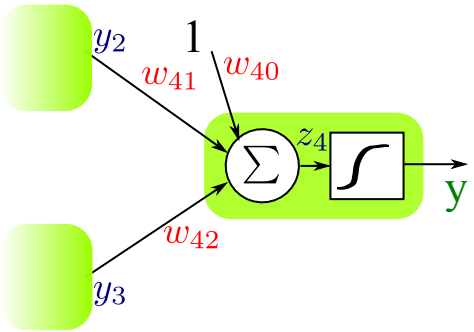
Error minimization

- Error : $E = \frac{1}{2}(y(\mathrm{w}, x^j) - y^j)^2$ with
  $\mathrm{w} = [w_{10}\ w_{11}\ w_{12}\ w_{20}\ w_{21}\ w_{30}\ w_{31}\ w_{40}\ w_{41}\ w_{42}]$ and $x^j = [x_1\ x_2]$
- Minimization : computation of the gradient
  $\nabla_{\mathrm{w}} E = (y(\mathrm{w}, x^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, x^j)$ where $\nabla_{\mathrm{w}} y = \left[\frac{\partial y}{\partial w_{10}}\ \frac{\partial y}{\partial w_{11}} \cdots \frac{\partial y}{\partial w_{42}}\right]$
- Output layer :



Error minimization

- Error : $E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$ with
  $\mathrm{w} = [w_{10}\ w_{11}\ w_{12}\ w_{20}\ w_{21}\ w_{30}\ w_{31}\ w_{40}\ w_{41}\ w_{42}]$ and $\mathrm{x}^j = [x_1\ x_2]$
- Minimization : computation of the gradient
  $\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j)$ where $\nabla_{\mathrm{w}} y = \left[\frac{\partial y}{\partial w_{10}}\ \frac{\partial y}{\partial w_{11}} \cdots \frac{\partial y}{\partial w_{42}}\right]$
- Output layer :



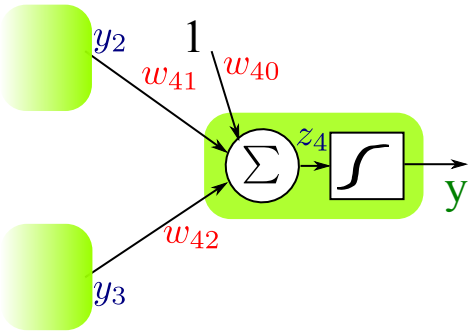
$$\frac{\partial y}{\partial w_{42}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{42}} = y(1-y)y_3$$
$$\frac{\partial y}{\partial w_{41}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{41}} = y(1-y)y_2$$
$$\frac{\partial y}{\partial w_{40}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{40}} = y(1-y)$$
$$\Rightarrow \delta_4 = (y - y^j)y(1-y)$$

Error minimization

- Error : $E = \frac{1}{2}(y(\mathrm{w}, \mathrm{x}^j) - y^j)^2$ with

  $\mathrm{w} = [w_{10}\ w_{11}\ w_{12}\ w_{20}\ w_{21}\ w_{30}\ w_{31}\ w_{40}\ w_{41}\ w_{42}]$ and $\mathrm{x}^j = [x_1\ x_2]$
- Minimization : computation of the gradient

  $\nabla_{\mathrm{w}} E = (y(\mathrm{w}, \mathrm{x}^j) - y^j)\nabla_{\mathrm{w}} y(\mathrm{w}, \mathrm{x}^j)$ where $\nabla_{\mathrm{w}} y = \left[\frac{\partial y}{\partial w_{10}}\ \frac{\partial y}{\partial w_{11}} \cdots \frac{\partial y}{\partial w_{42}}\right]$
- Output layer :

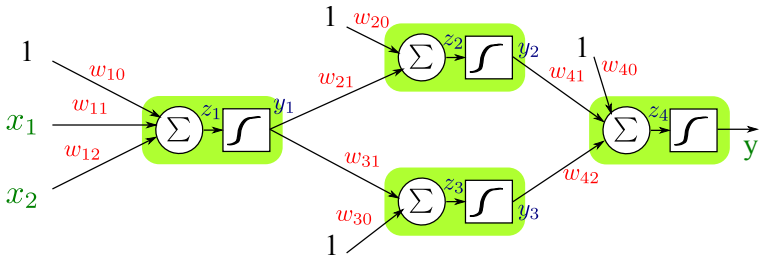


$$\frac{\partial y}{\partial w_{42}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{42}} = y(1-y)y_3$$
$$\frac{\partial y}{\partial w_{41}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{41}} = y(1-y)y_2$$
$$\frac{\partial y}{\partial w_{40}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{40}} = y(1-y)$$
$$\Rightarrow \delta_4 = (y - y^j)y(1-y)$$

- Gradient descent :

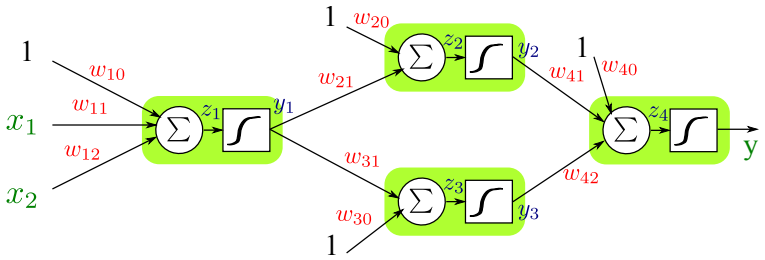
$$w_{42} \leftarrow w_{42} - \eta\delta_4 y_3$$
$$w_{41} \leftarrow w_{41} - \eta\delta_4 y_2$$
$$w_{40} \leftarrow w_{40} - \eta\delta_4$$

Gradient descent on hidden layers



$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

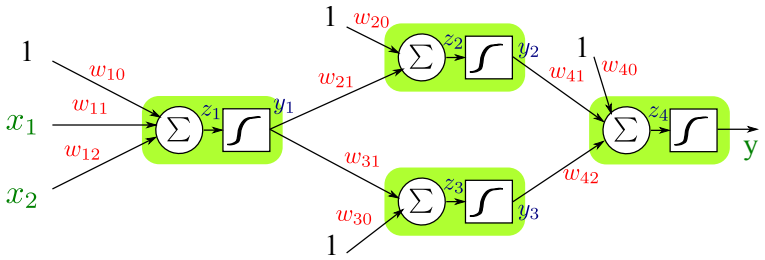
Gradient descent on hidden layers



$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

- Then $\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial w_{31}} = y(1-y)w_{42}y_3(1-y_3)y_1$

Gradient descent on hidden layers

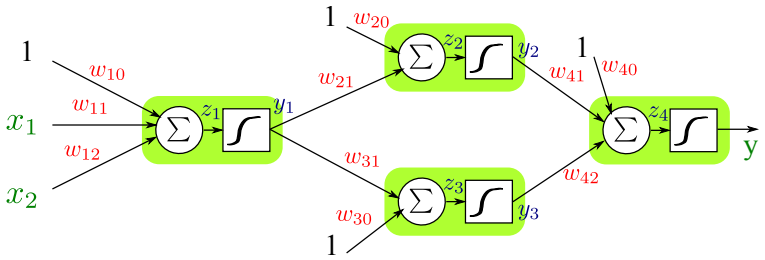


$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

- Then $\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial w_{31}} = y(1-y)w_{42}y_3(1-y_3)y_1$
- And, similarly :

$$\frac{\partial y}{\partial w_{30}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{30}} = y(1-y)w_{42}y_3(1-y_3)$$

Gradient descent on hidden layers



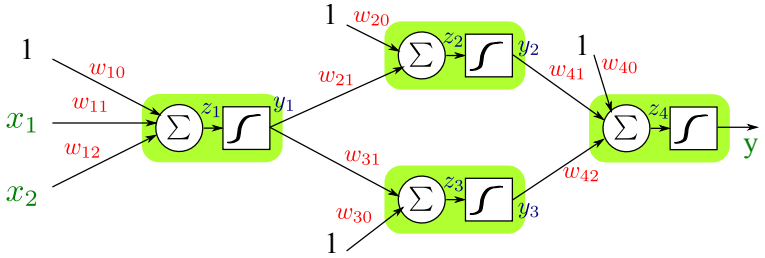
$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

- Then $\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial w_{31}} = y(1-y)w_{42}y_3(1-y_3)y_1$
- And, similarly :

$$\frac{\partial y}{\partial w_{30}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{30}} = y(1-y)w_{42}y_3(1-y_3)$$

- Thus : $\delta_3 = \delta_4 w_{42} y_3(1-y_3)$ (recursion !)

Gradient descent on hidden layers



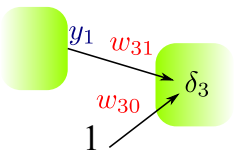
$$\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{31}} \text{ with } \begin{cases} z_4 = w_{40} + w_{41}y_2 + w_{42}y_3 \\ y_3 = s(z_3) = s(w_{30} + w_{31}y_1) \end{cases}$$

- Then $\frac{\partial y}{\partial w_{31}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial w_{31}} = y(1-y)w_{42}y_3(1-y_3)y_1$
- And, similarly :

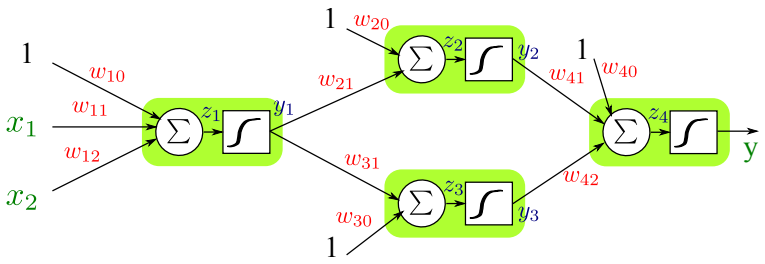
$$\frac{\partial y}{\partial w_{30}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{30}} = y(1-y)w_{42}y_3(1-y_3)$$

- Thus : $\delta_3 = \delta_4 w_{42} y_3(1-y_3)$ (recursion !)
- Gradient descent :

$$w_{30} \leftarrow w_{30} - \eta\delta_3$$
$$w_{31} \leftarrow w_{31} - \eta\delta_3 y_1$$



Gradient descent on hidden layers (2) (Students have to fill the answers)



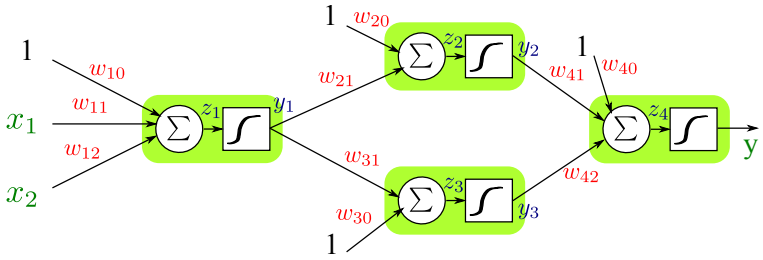
- Similarly :

$$\frac{\partial y}{\partial w_{21}} =$$

- and

$$\frac{\partial y}{\partial w_{20}} =$$

Gradient descent on hidden layers (2) (Students have to fill the answers)



- Similarly :

$$\frac{\partial y}{\partial w_{21}} =$$

- and

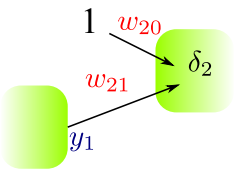
$$\frac{\partial y}{\partial w_{20}} =$$

- Thus :

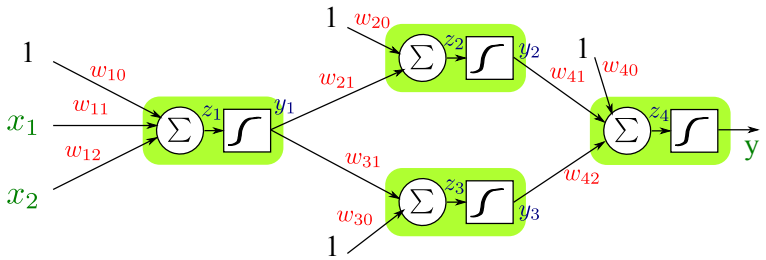
$$\delta_2 = \delta_4 w_{41} y_2 (1 - y_2)$$

- Gradient descent :

$$w_{20} \leftarrow w_{20} - \eta \delta_2$$
$$w_{21} \leftarrow w_{21} - \eta \delta_2 y_1$$



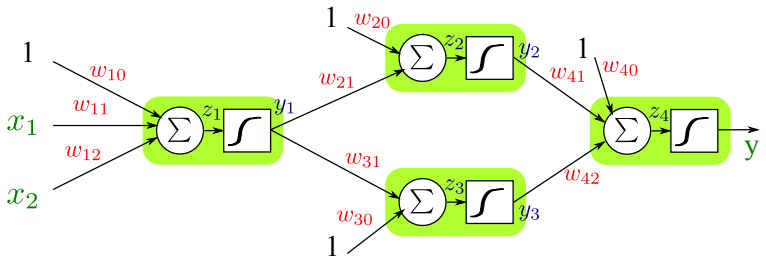
Gradient descent on first hidden layer



$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{12}}$$

with $z_4 = w_{40} + w_{41}y_2 + w_{42}y_3$

Gradient descent on first hidden layer

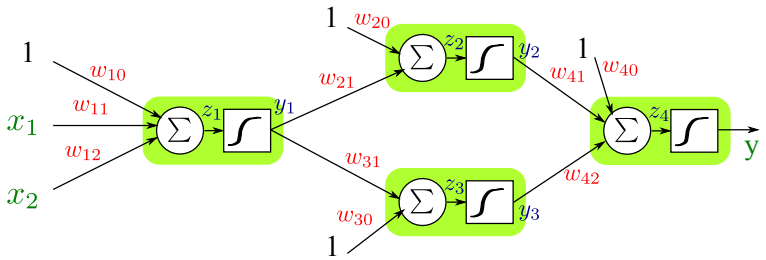


$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{12}}$$
with $z_4 = w_{40} + w_{41}y_2 + w_{42}y_3$

Thus :

$$\begin{aligned}
\frac{\partial y}{\partial w_{12}} &= \frac{\partial y}{\partial z_4}\left(\frac{\partial z_4}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}} + \frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}}\right) \\
&= y(1-y)\left(w_{41}y_2(1-y_2)w_{21}y_1(1-y_1)x_2\right. \\
&\quad \left. + w_{42}y_3(1-y_3)w_{31}y_1(1-y_1)x_2\right) \\
&= y(1-y)\left(w_{41}y_2(1-y_2)w_{21} + w_{42}y_3(1-y_3)w_{31}\right)y_1(1-y_1)x_2
\end{aligned}$$

Gradient descent on first hidden layer



$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{12}}$$
with $z_4 = w_{40} + w_{41}y_2 + w_{42}y_3$

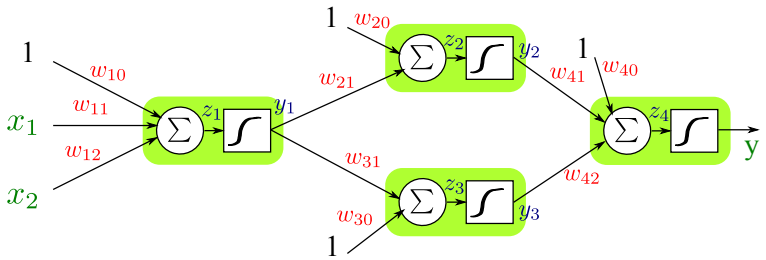
Thus :

$$\begin{aligned}
\frac{\partial y}{\partial w_{12}} &= \frac{\partial y}{\partial z_4}\left(\frac{\partial z_4}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}} + \frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}}\right) \\
&= y(1-y)\left(w_{41}y_2(1-y_2)w_{21}y_1(1-y_1)x_2\right. \\
&\quad \left. + w_{42}y_3(1-y_3)w_{31}y_1(1-y_1)x_2\right) \\
&= y(1-y)\left(w_{41}y_2(1-y_2)w_{21} + w_{42}y_3(1-y_3)w_{31}\right)y_1(1-y_1)x_2
\end{aligned}$$

And :

$$\delta_1 = (\delta_2 w_{21} + \delta_3 w_{31})y_1(1-y_1)$$

Gradient descent on first hidden layer



$$\frac{\partial y}{\partial w_{12}} = \frac{\partial y}{\partial z_4}\frac{\partial z_4}{\partial w_{12}}$$
with $z_4 = w_{40} + w_{41}y_2 + w_{42}y_3$

Thus :

$$\begin{aligned}
\frac{\partial y}{\partial w_{12}} &= \frac{\partial y}{\partial z_4}\left(\frac{\partial z_4}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}} + \frac{\partial z_4}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial y_1}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{12}}\right) \\
&= y(1-y)\left(w_{41}y_2(1-y_2)w_{21}y_1(1-y_1)x_2\right. \\
&\quad \left. + w_{42}y_3(1-y_3)w_{31}y_1(1-y_1)x_2\right) \\
&= y(1-y)\left(w_{41}y_2(1-y_2)w_{21} + w_{42}y_3(1-y_3)w_{31}\right)y_1(1-y_1)x_2
\end{aligned}$$

And :

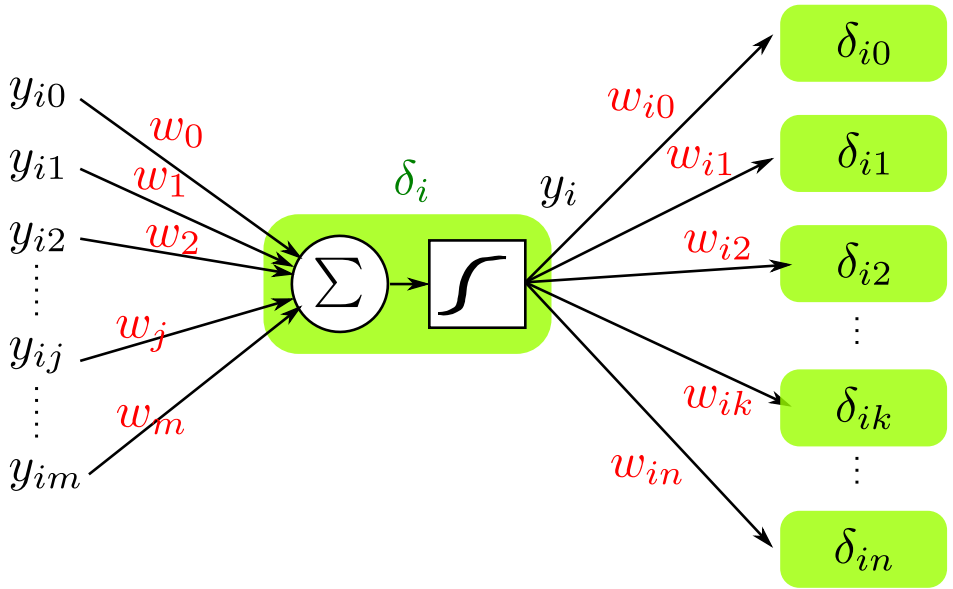
$$\delta_1 = (\delta_2 w_{21} + \delta_3 w_{31})y_1(1-y_1)$$

Gradient descent :

$$\begin{aligned}
w_{10} &\leftarrow w_{10} - \eta\delta_1 \\
w_{11} &\leftarrow w_{11} - \eta\delta_1 x_1 \\
w_{12} &\leftarrow w_{12} - \eta\delta_1 x_2
\end{aligned}$$

Generalized δ rule

Considering neuron i connected to other neurons :



- Delta rule :

$$\delta_i = y_i(1 - y_i) \sum_{k=0}^{n} w_{ik}\delta_{ik}$$

- Gradient descent :

$$\forall j \in [0\ m]\ w_j \leftarrow w_j - \eta\delta_i y_{ij}$$

Backpropagation Algorithm (*Retropropagation*)

1. Initialize weights to small random values
2. Choose a random sample training item, say (x^j, y^j)
3. Compute total input z_i and output y_i for each unit (**forward prop**)
4. Compute δ_p for output layer $\delta_p = y_p(1 - y_p)(y_p - y_j)$
5. Compute δ_i for all preceding layers by **backprop rule**
6. Compute weight change by **descent rule** (repeat for all weights)

- Note that each expression involves data local to a particular unit, we do not have to look around summing things over the whole network.
- It is for this reason, simplicity, locality and, therefore, efficiency that backpropagation has become the dominant paradigm for training neural nets.

Input encoding

- All values are real values from [0 1].
- If not, scale the values : $x = \frac{u - u_{\min}}{u_{\max} - u_{\min}}$
- For discrete values, unary encoding form. As for example :

	x_1	x_2	x_3
flour	1	0	0
butter	0	1	0
sugar	0	0	1

Output encoding

- for a **regression** :
  - one output neuron if we need to predict a scalar value
  - n output neurons if the dimension of the vector to predict is n
- for a **binary classification** :
  - one output neuron
    - if $y < 0.5$: class 0
    - else class 1
- for a **multi-class classification** (n classes) :
  - one output neuron : divide [0 1] by n
  - n output neurons
    - each neuron corresponds to probability over classes
    - `tensorflow.keras.utils.to_categorical(yTest, nbClasses)`
    - e.g. : yTest=3, nbClasses = 10 lead to $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]^T$
    - `numpy.argmax(model.predict(xTest),axis=1)`
    - e.g. xTest = $[0.01, 0.02, 0.6, 0.05, 0.02, 0.1, 0.1, 0.03, 0.04, 0.03]^T$ leads to 2

Training

1. Split data set (randomly) into three subsets :
   - Training set : used for adjusting weights
   - Validation set : used to stop training
   - Test set : used to evaluate performance
2. Pick random, small weights as initial values
3. Perform iterative minimization of error over training set
4. Stop when error on validation set reaches a minimum (to avoir overfitting)
5. Repeat training (from step 2) several times (to avoir local minima)
6. Use best weights to compute error on the test set.

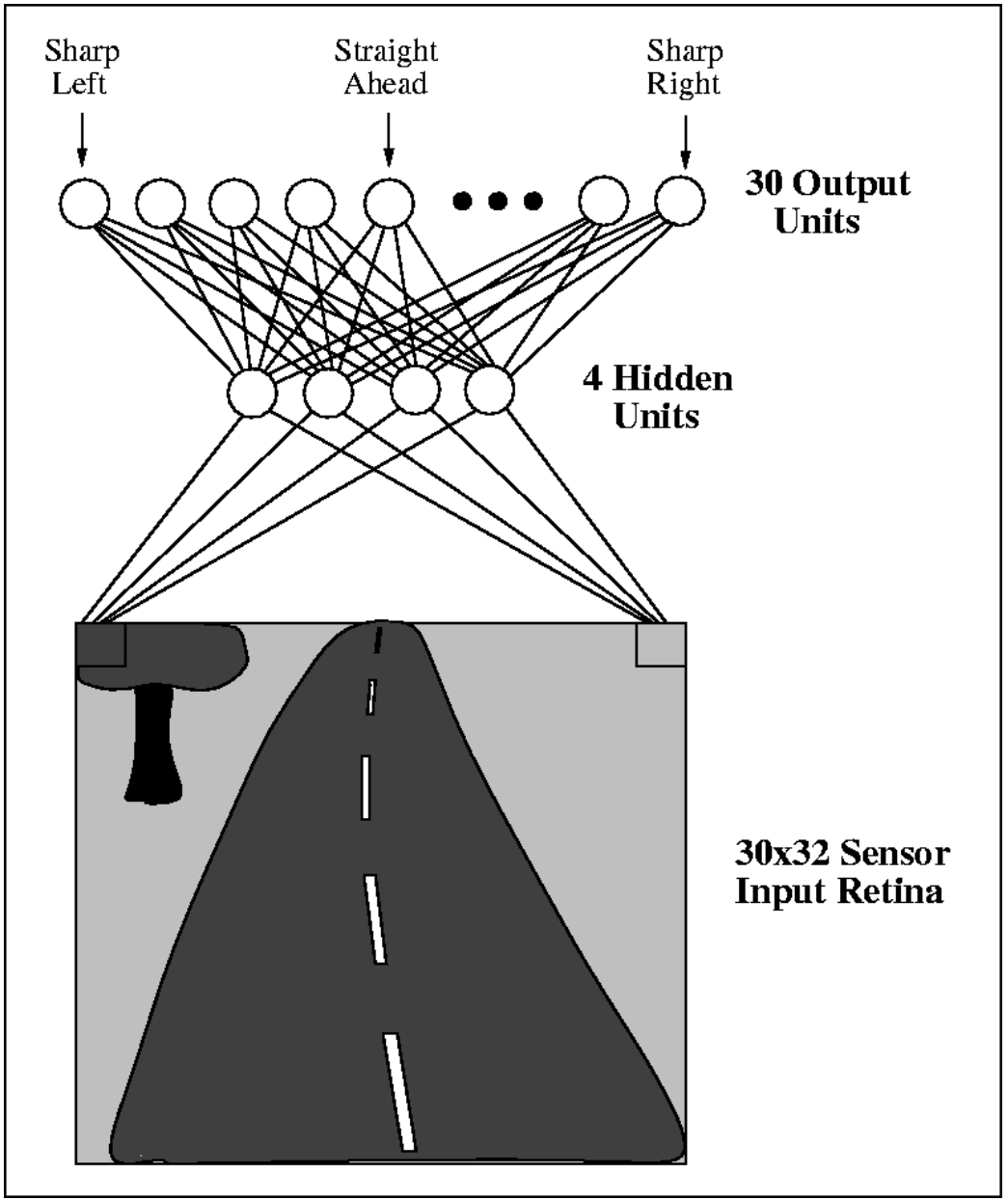
If the data set is too small to be divided into 3 subsets, use cross-validation.

ALVINN : Autonomous Land Vehicle In a Neural Network

- 960 inputs (a 30x32 array derived from the pixels of an image)
- 4 hidden units
- 30 output units (each representing a steering command)

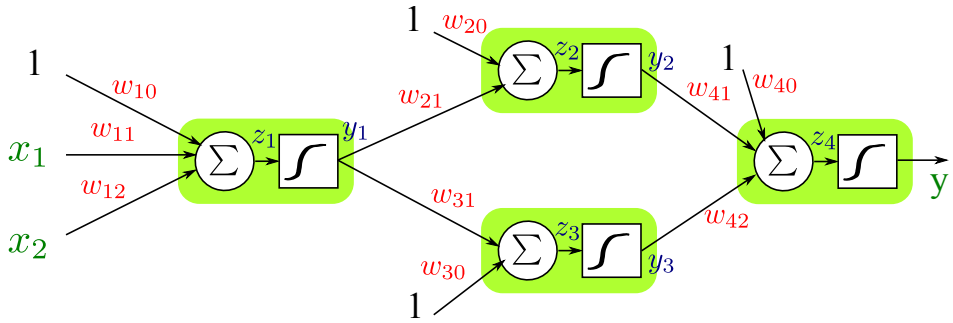
`https://youtu.be/ilP4aPDTBPE`

Dean Pomerleau, CMU, 1989



Exercice

- Here is the network for regression :



- Actual values of weights are :

w_{10}	w_{11}	w_{12}	w_{20}	w_{21}	w_{30}	w_{31}	w_{40}	w_{41}	w_{42}
0.1	0.1	0.1	0.15	0.12	0.09	0.07	0.2	0.2	0.1

- New incoming data is : $x_1 = 0.4$, $x_2 = 0.6$, $y = 0.2$
- Considering $\eta = 0.01$, compute the weights at next step :

w_{10}	w_{11}	w_{12}	w_{20}	w_{21}	w_{30}	w_{31}	w_{40}	w_{41}	w_{42}

Exercice

- Step 1 : forward propagation. Compute the output of each neuron.
  - $y_1 = s(0.1 + 0.04 + 0.06) = s(0.2) = 0.550$
  - $y_2 = s(0.15 + 0.12 * 0.550) = 0.554$
  - $y_3 = s(0.09 + 0.07 * 0.550) = 0.532$
  - $y = s(0.2 + 0.2 * 0.554 + 0.1 * 0.532) = 0.59$ higher than the expected value of 0.2
- Step 2 : compute the δ for each neuron, backward.
  - $\delta_4 = (0.59 - 0.2) * 0.59 * (1 - 0.59) = 0.094$
  - $\delta_2 = 0.094 * 0.1 * 0.532 * (1 - 0.532) = 0.0047$
  - $\delta_3 = 0.094 * 0.2 * 0.554 * (1 - 0.554) = 0.0023$
  - $\delta_1 = (0.12 * \delta_2 + 0.07 * \delta_3) * 0.55 * (1 - 0.55) = 0.00015$
- Step 3 : update weights

w_{10}	w_{11}	w_{12}	w_{20}	w_{21}
0.099998	0.099999	0.099999	0.14998	0.11999
w_{30}	w_{31}	w_{40}	w_{41}	w_{42}
0.08995	0.06997	0.1991	0.1995	0.0995

new y4 = 0.5896

Activation functions

- Main activation functions :
  - Sigmoid
  - ReLU : Rectified Linear Unit
  - Softmax : transforms output values into values that can be interpreted as probabilities
- Other activation functions :
  - tanh, LeakyReLU, PReLU, ELU, SELU, GELU ...

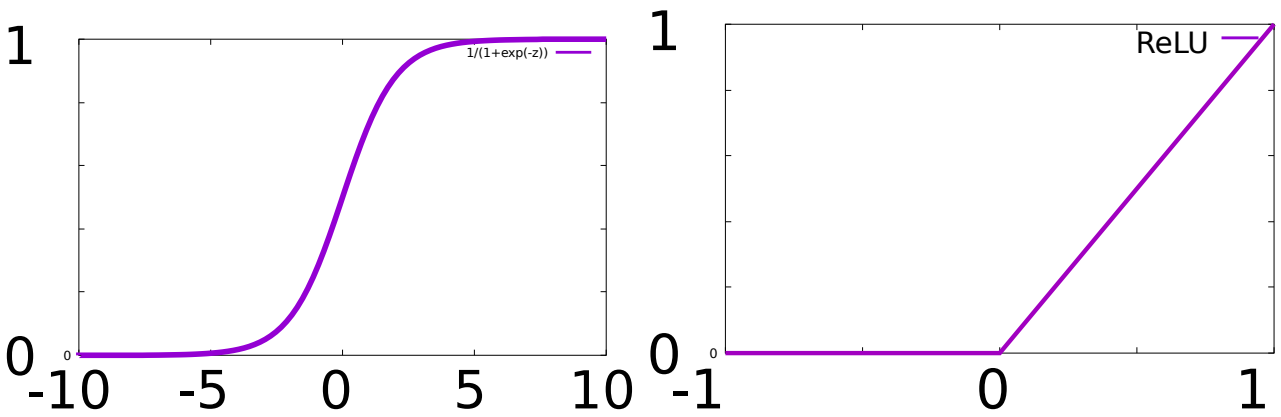
Why ReLU ?

- problem of vanishing gradient
  - remember the delta rule and gradient descent :

$$\delta_i = y_i(1 - y_i) \sum_{k=0}^{n} w_{ik}\delta_{ik}$$

and

$$\forall j \in [0\ m]\ w_j \leftarrow w_j - \eta\delta_i y_{ij}$$



- different solutions :
  - change to activation function to ReLU
  - residual networks
  - batch normalisation

Softmax layer

- smooth approximation to the arg max function
  - assign probabilities to indices of having the highest response $z = [z_0 z_1 z_2 ... z_n]$ for each i,

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

  - every value in [0 1] and the sum is equal to 1

Loss functions

- classification :
  - binary crossentropy (same as logistic regression) :
    - $-\left(y^{(i)} \log(h_\theta(\mathrm{x}^{(\mathrm{i})})) + (1 - y^{(i)}) \log(1 - h_\theta(\mathrm{x}^{(\mathrm{i})}))\right)$
  - categorical crossentropy :
    - $-\sum_{c=1}^{nbClasses} y^i \log(h_\theta(\mathrm{x}^{(\mathrm{i})})$
- regression :
  - mse, rmse ...
- see `https://www.tensorflow.org/api_docs/python/tf/keras/losses` for details
  - and this video : `https://www.youtube.com/watch?v=QBbC3Cjsnjg`

Outline

How to build a neural network in `tensorflow.keras` ?

- 3 modes :
  - Sequential : the easiest but limited

```
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Dense(20, input_dim=784, activation='sigmoid'))
```

How to build a neural network in `tensorflow.keras` ?

- 3 modes :
  - Sequential : the easiest but limited

```
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Dense(20, input_dim=784, activation='sigmoid'))
```

  - Functional API : for creating more complex networks (branches, multiple inputs or outputs, DAGs, shared layers)

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
inputImage = Input(shape=(784,))
myNet = Dense(20, activation='sigmoid')(inputImage)
model = Model(inputImage, myNet)
```

How to build a neural network in tensorflow.keras ?

- 3 modes :
  - Sequential : the easiest but limited

```
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Dense(20, input_dim=784, activation='sigmoid'))
```

  - Functional API : for creating more complex networks (branches, multiple inputs or outputs, DAGs, shared layers)

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
inputImage = Input(shape=(784,))
myNet = Dense(20, activation='sigmoid')(inputImage)
model = Model(inputImage, myNet)
```

  - Model Subclassing : new from tensorflow 2.0

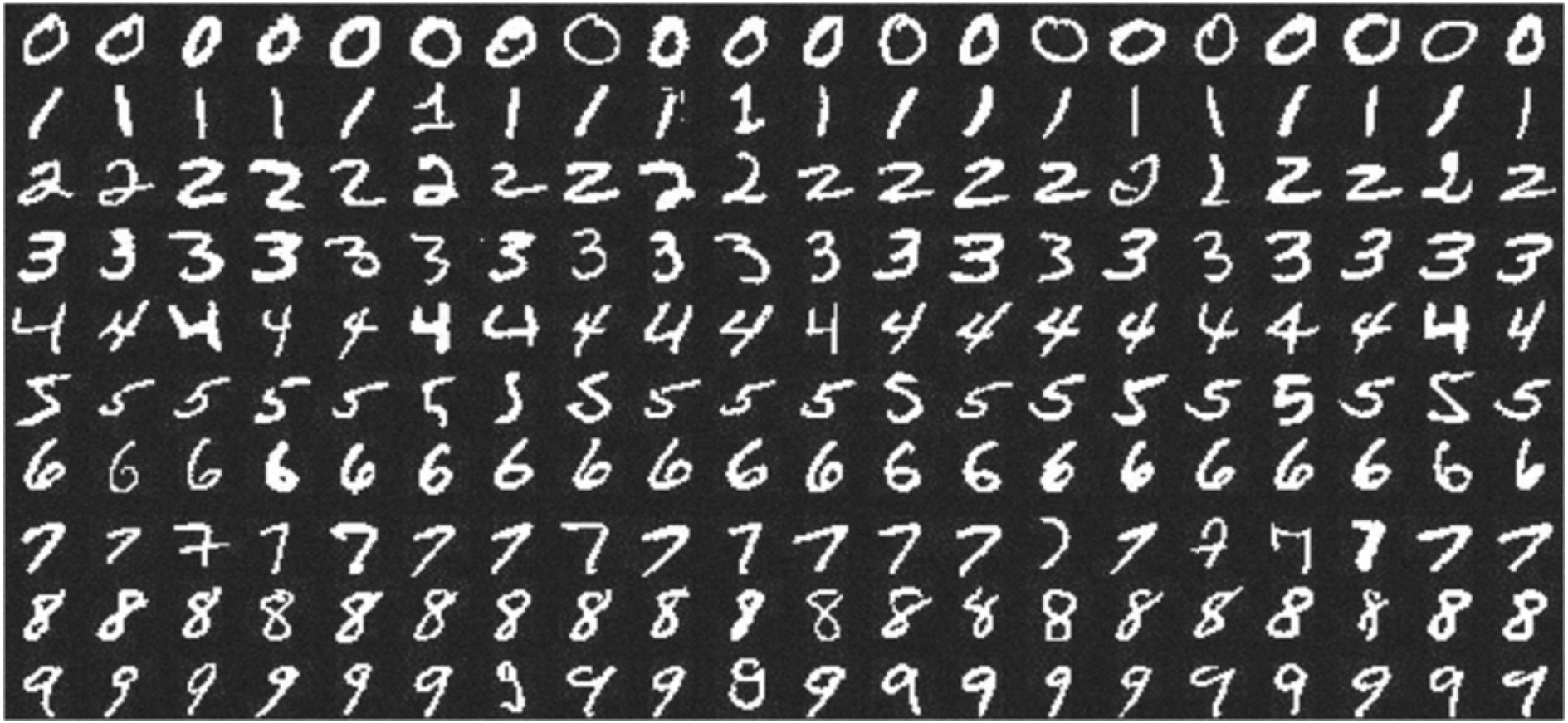
```
tensorflow.keras.Model class myFirstModel(Model): ...
```

Toy example in python

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
import numpy as np

X = np.array([[0, 0], [1, 1]])
y = np.array([0,1])

model = Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(X, y, epochs=100, batch_size=2)
```

Classify handwritten digits using MNIST dataset

Dataset of 60000+10000 grayscale squared images (28x28).

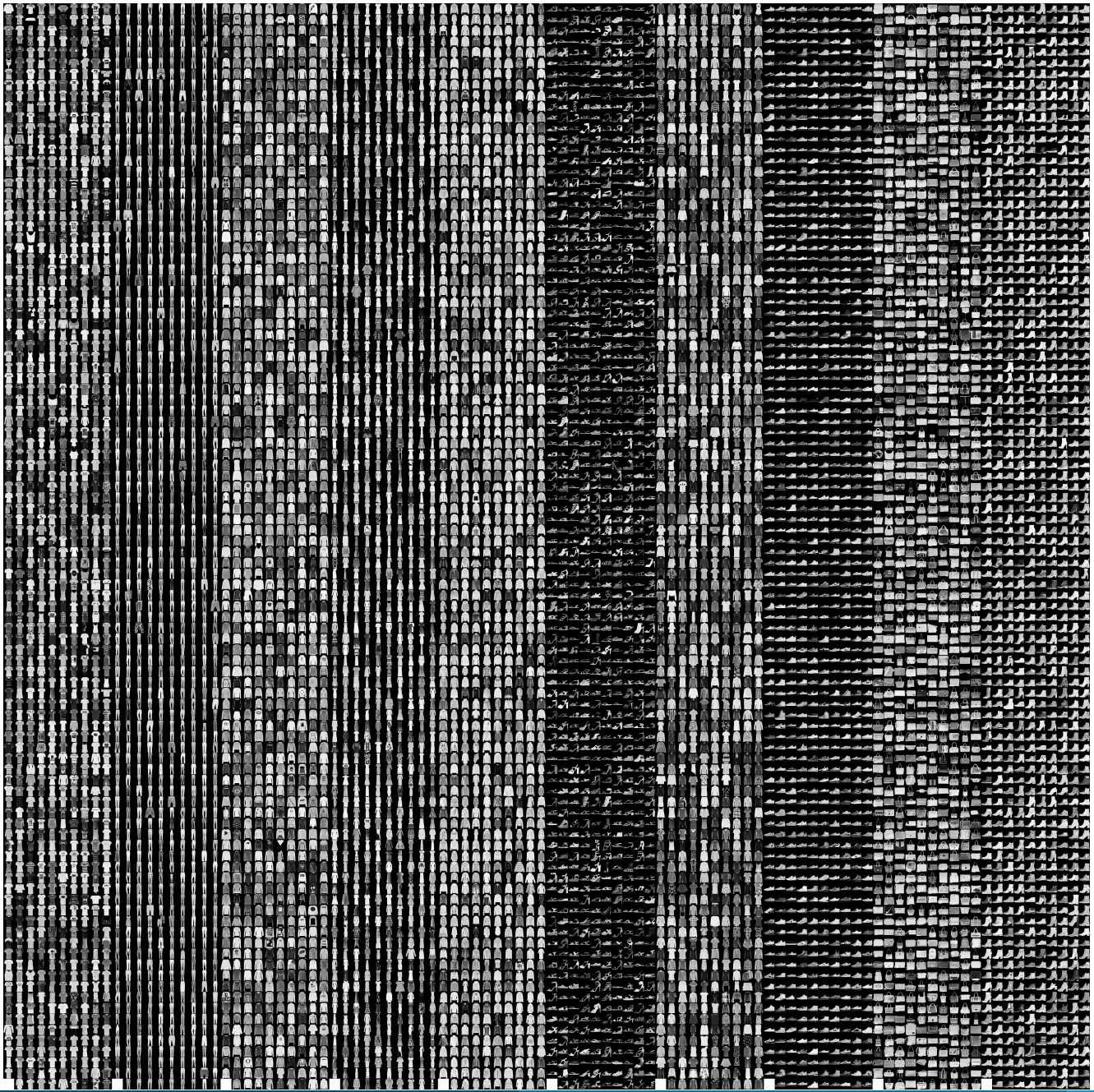


```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
import numpy as np
from tensorflow.keras.datasets import mnist
```

Classify handwritten digits using MNIST dataset : code

```
(xTrain,yTrain),(xTest,yTest) = mnist.load_data()
nbClasses = 10
nbNeuronsHL = 20 # nb of neurons on the hidden layer
xTrain = xTrain.reshape(60000, 784)
xTest = xTest.reshape(10000, 784)
xTrain = xTrain.astype('float32')
xTest = xTest.astype('float32')
xTrain /= 255
xTest /= 255
yTrain = tf.keras.utils.to_categorical(yTrain, nbClasses)
yTest = tf.keras.utils.to_categorical(yTest, nbClasses)
model = Sequential()
model.add(Dense(nbNeuronsHL, input_dim=784, activation='sigmoid'))
model.add(Dense(nbClasses, activation='softmax'))

model.compile(optimizer='rmsprop',loss='categorical_crossentropy',
    metrics=['accuracy'])

model.fit(xTrain, yTrain, epochs=20, batch_size=128)
score = model.evaluate(xTest,yTest)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))
```

Fashion MNIST



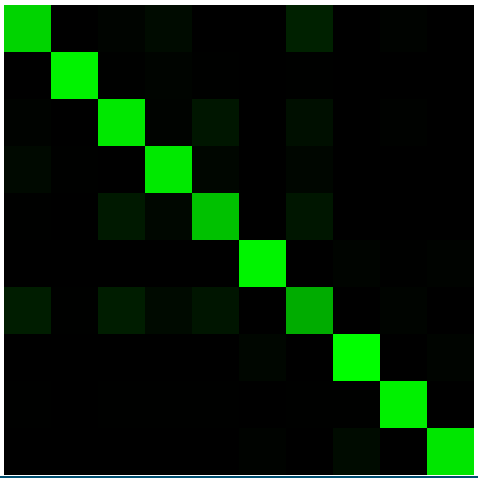
Dataset of 60000+10000 grayscale squared images (28x28).

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Tests on fmnist : confusion matrix computed on test data

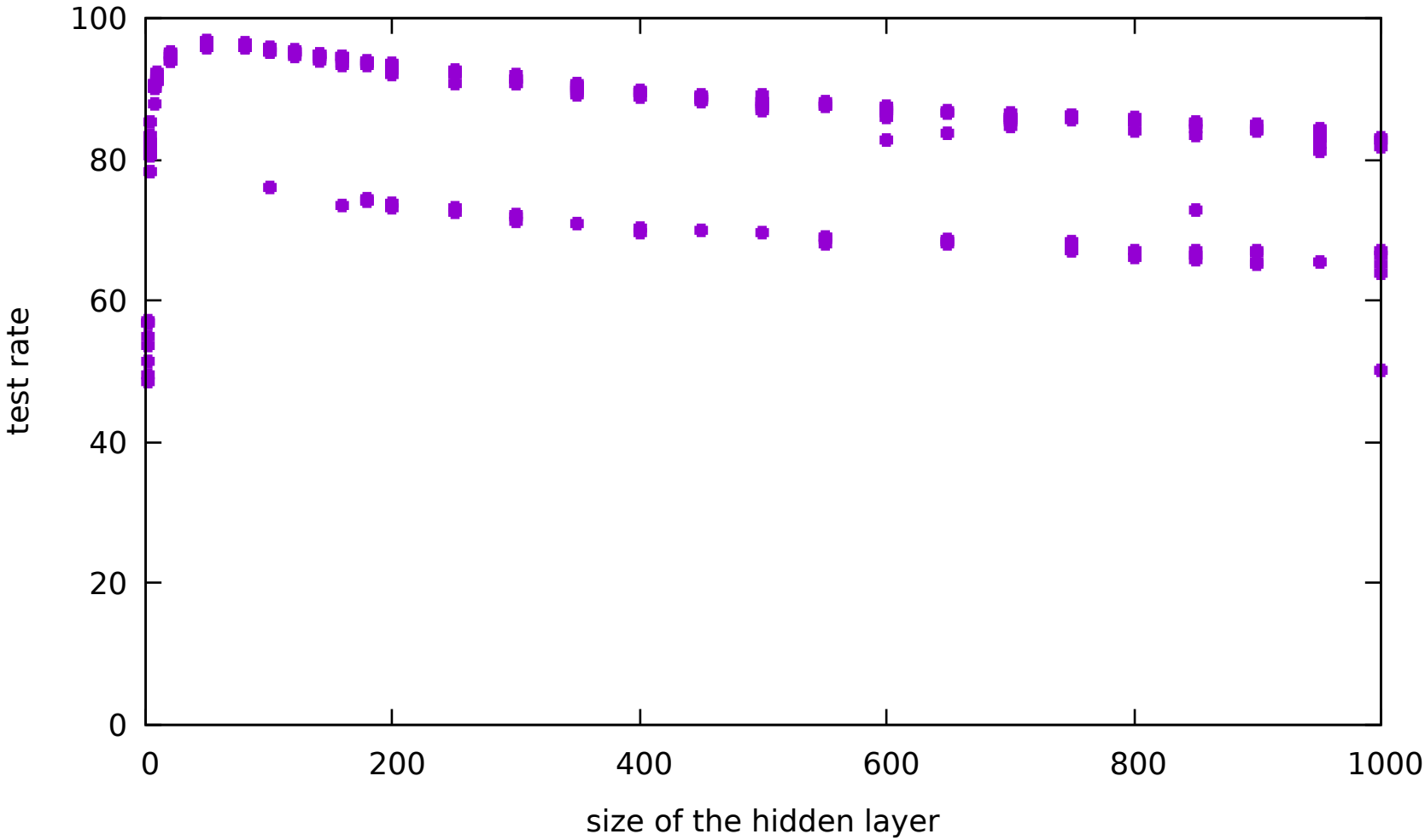
One hidden layer with 20 neurons.

	predicted class									
	0	1	2	3	4	5	6	7	8	9
0	686	3	14	37	3	1	109	1	11	0
1	1	788	6	13	4	0	5	0	1	0
2	11	2	752	11	73	1	49	0	8	0
3	32	6	3	754	21	0	21	0	0	1
4	4	0	81	23	625	0	73	0	2	0
5	0	0	0	0	0	790	0	15	5	11
6	96	4	95	35	71	0	557	2	13	1
7	0	0	0	0	0	20	0	825	0	14
8	5	1	5	5	4	2	6	4	781	0
9	0	0	0	0	0	12	0	38	0	747



One hidden layer : only 10000 data (test) 80%/20%

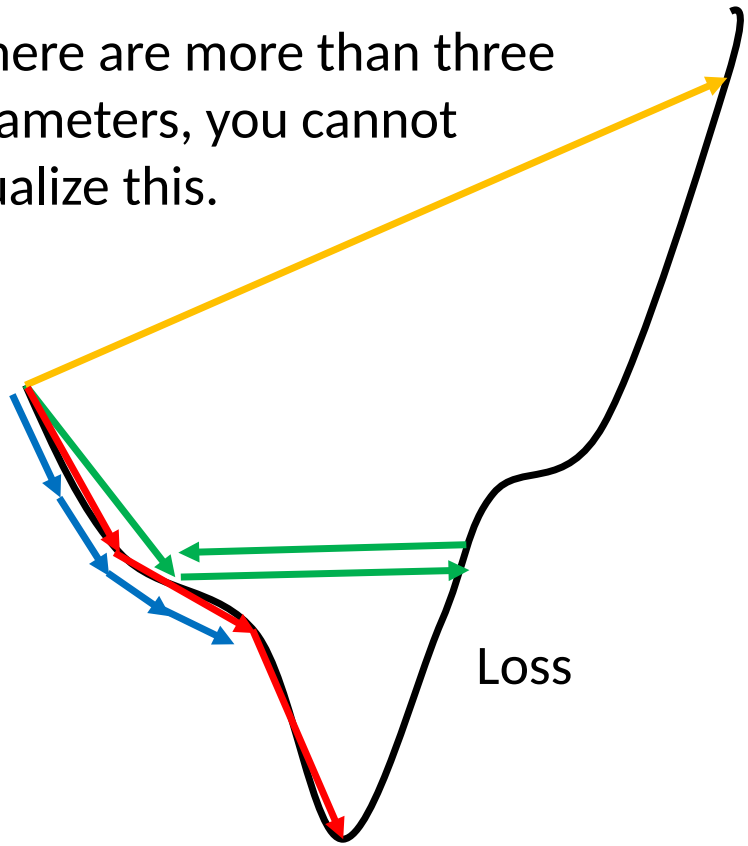
Many trials with random initialisation of weight (modified OpenCV code).



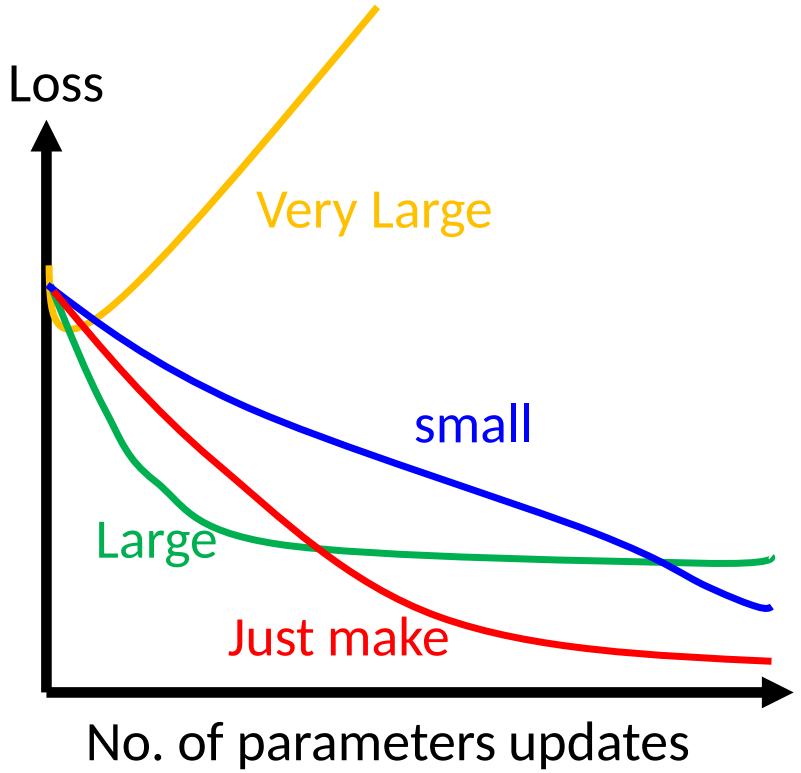
Local minima ...

Learning rate

If there are more than three parameters, you cannot visualize this.



Set the learning rate η carefully

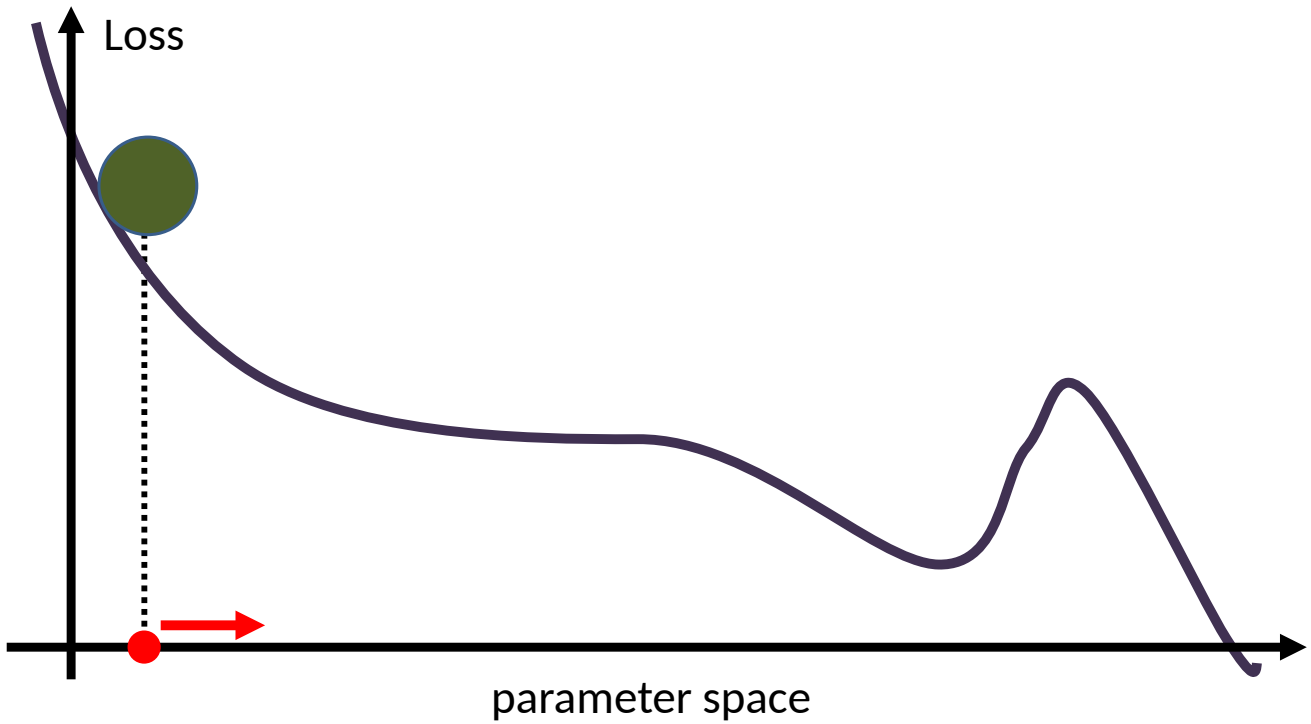


But you can always visualize this.

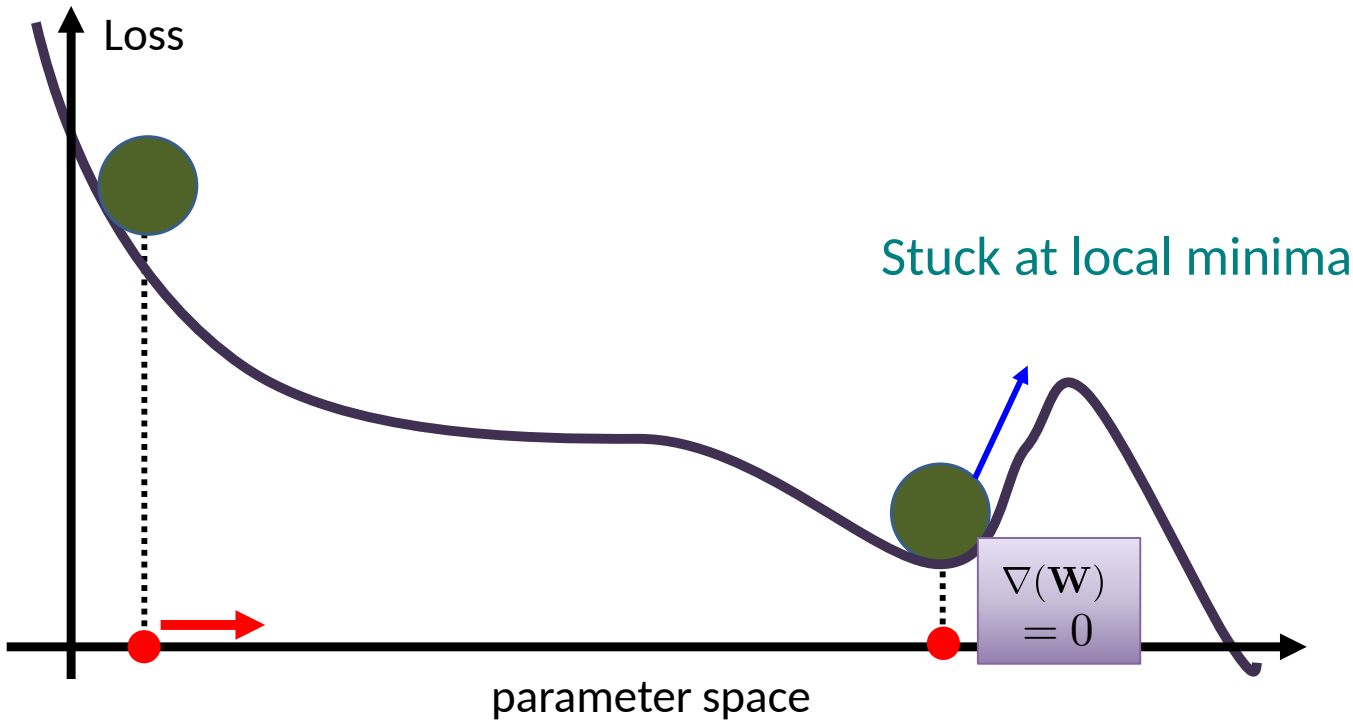
Rprop algorithm (1992)

- Only the sign of gradient does matter :
  - if the last 2 gradients have the same sign :
    - good direction, we can fasten the convergence by increasing the learning rate : $\alpha* = \eta_+$ with $\eta_+ > 1$, e.g. 1.2.
  - if the last 2 gradients have opposite sign :
    - the step was too large : we need to decrease the learning rate : $\alpha* = \eta_-$ with $\eta_- < 1$, e.g. 0.5.

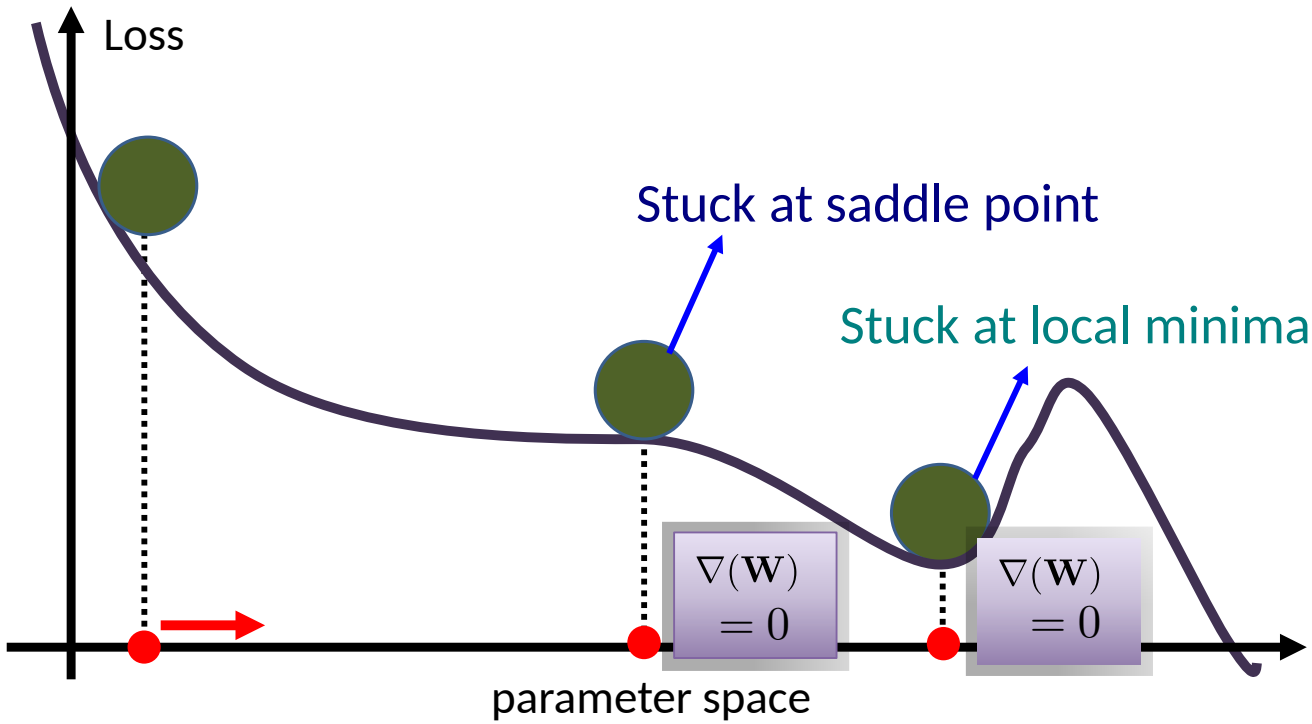
Momentum : avoid local minima



Momentum : avoid local minima

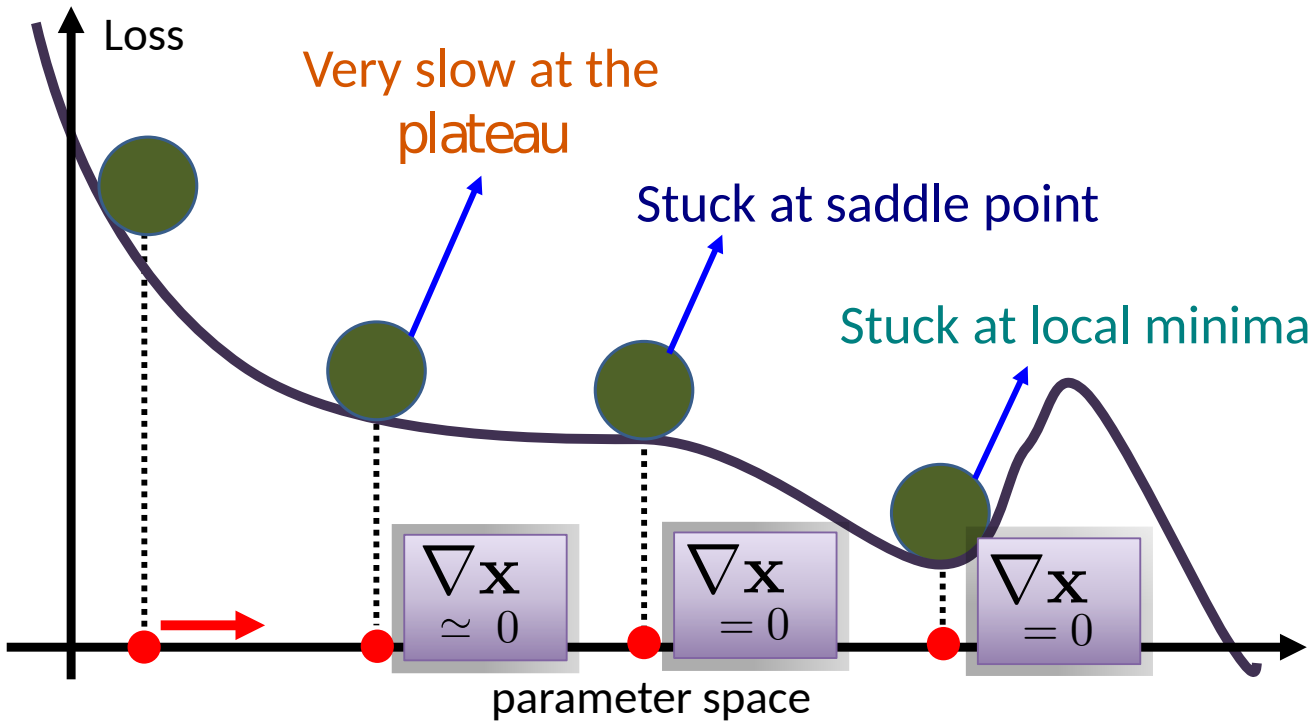


Momentum : avoid local minima

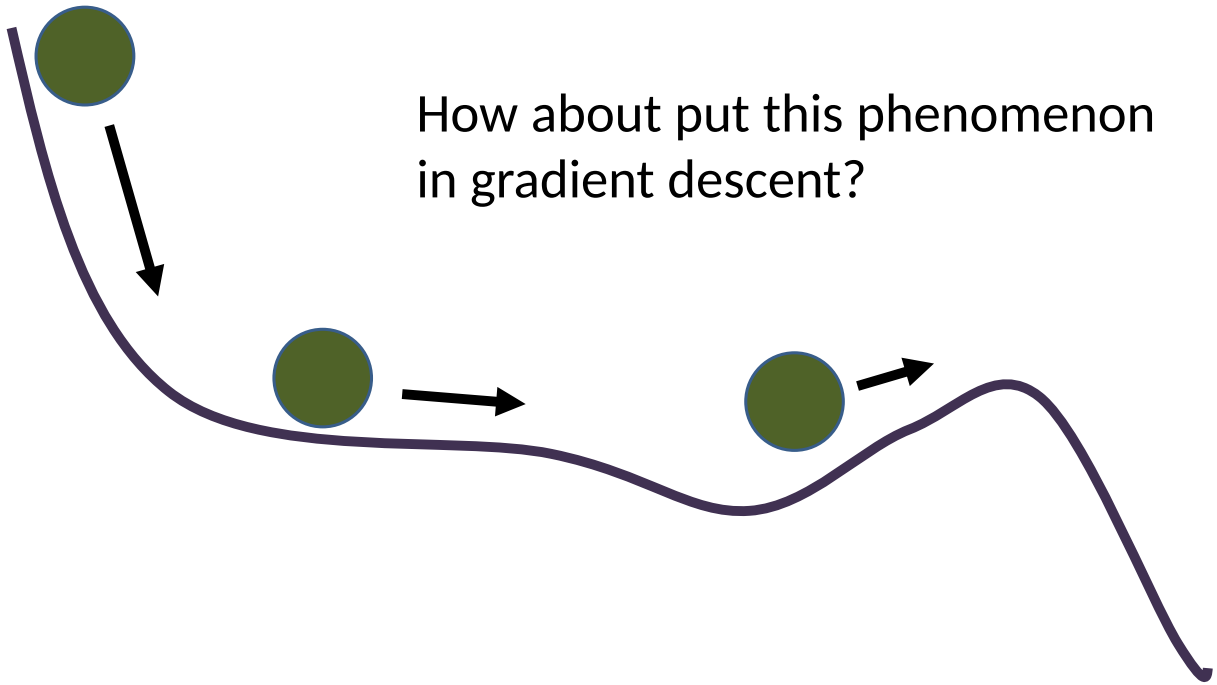


123

Momentum : avoid local minima

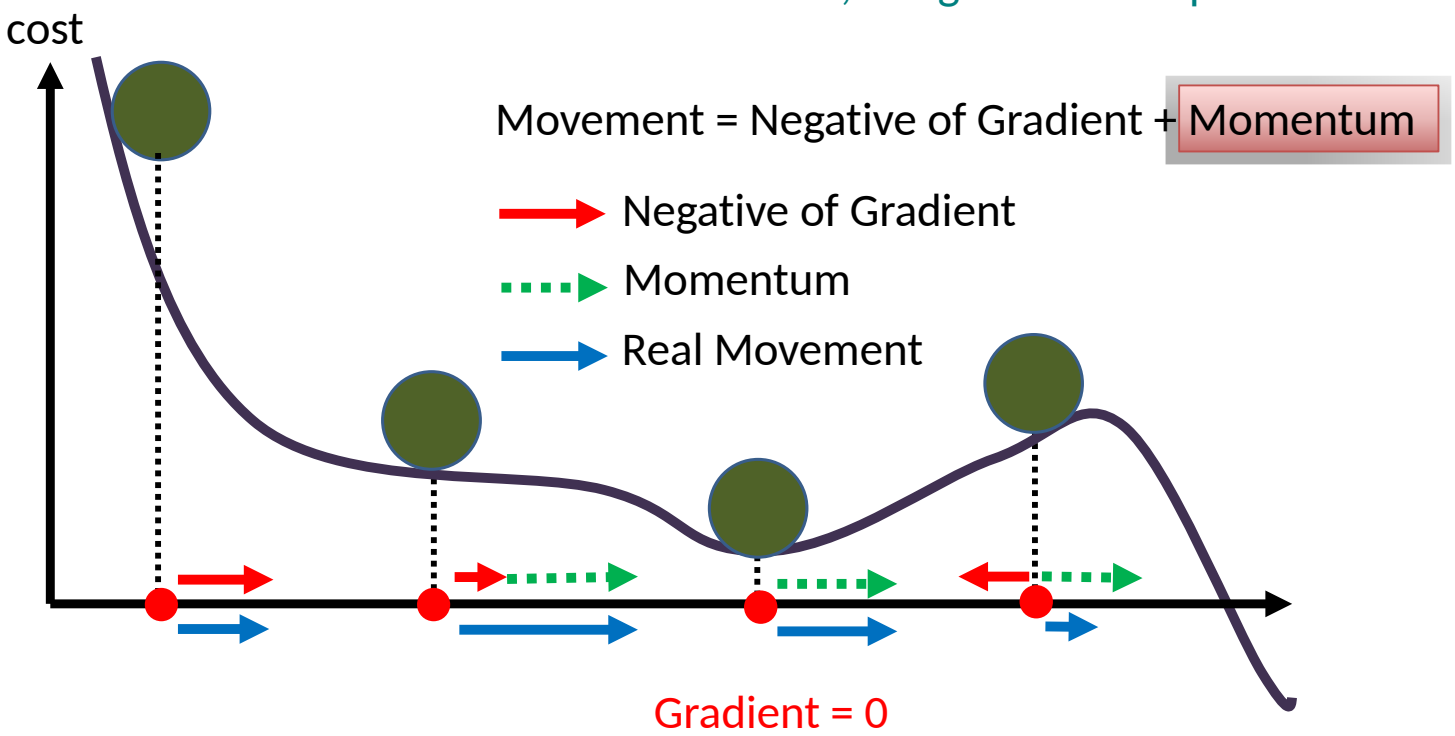


Momentum : avoid local minima



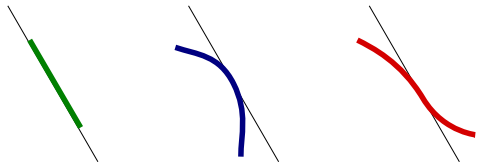
Momentum : avoid local minima

Still not guarantee reaching global minima, but give some hope .....



Local minima - Pathological curvatures

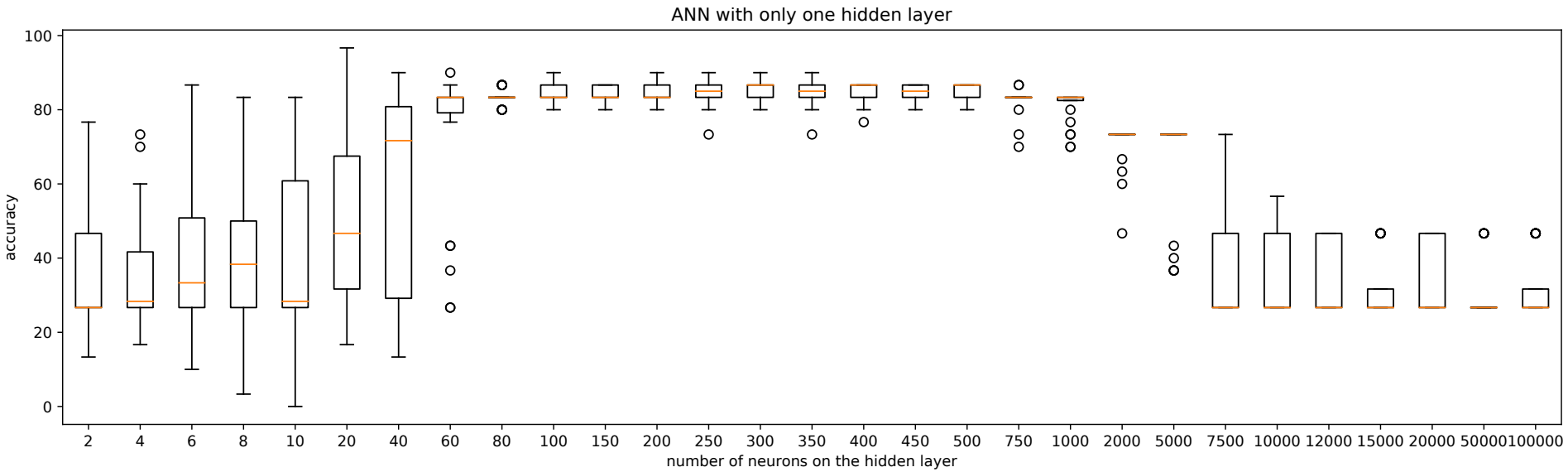
Idea : gradient doesn't tell everything. What about variations of gradient (second derivatives) ?



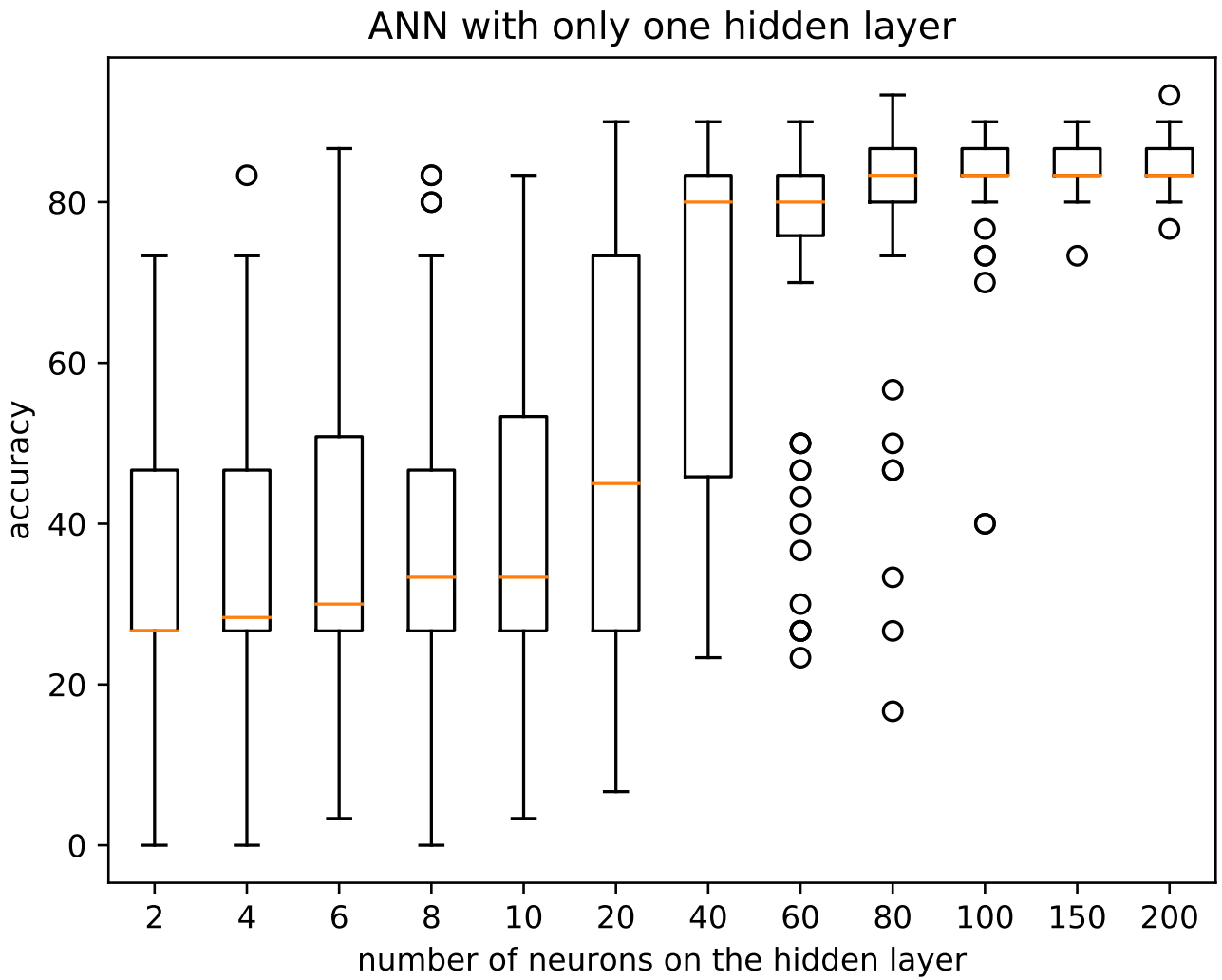
- Momentum
  - accumulates the gradient of the previous steps in order to compute the new weights
  - nice presentation by Andrew Ng
    `https://www.youtube.com/watch?v=k8fTYJPd3_I`
- RMSProp (Root Mean Square Propagation)
  - different learning rates for each weight
  - nice presentation by Andrew Ng
    `https://www.youtube.com/watch?v=_e-LFe_igno`
- AdaM (Adaptive Moment Optimization)
  - combining both Momentum and RMSProp

One hidden layer. Influence of the number of neurons

- Iris dataset (3 classes, 150 data, 4 features).
- ANN using one hidden layer. 24 trials for each set of parameters
- sigmoid activation for hidden layers, softmax for output layer, optimizer RMSprop
- early stopping on validation accuracy, $\min\Delta = 10^{-4}$, patience of 20

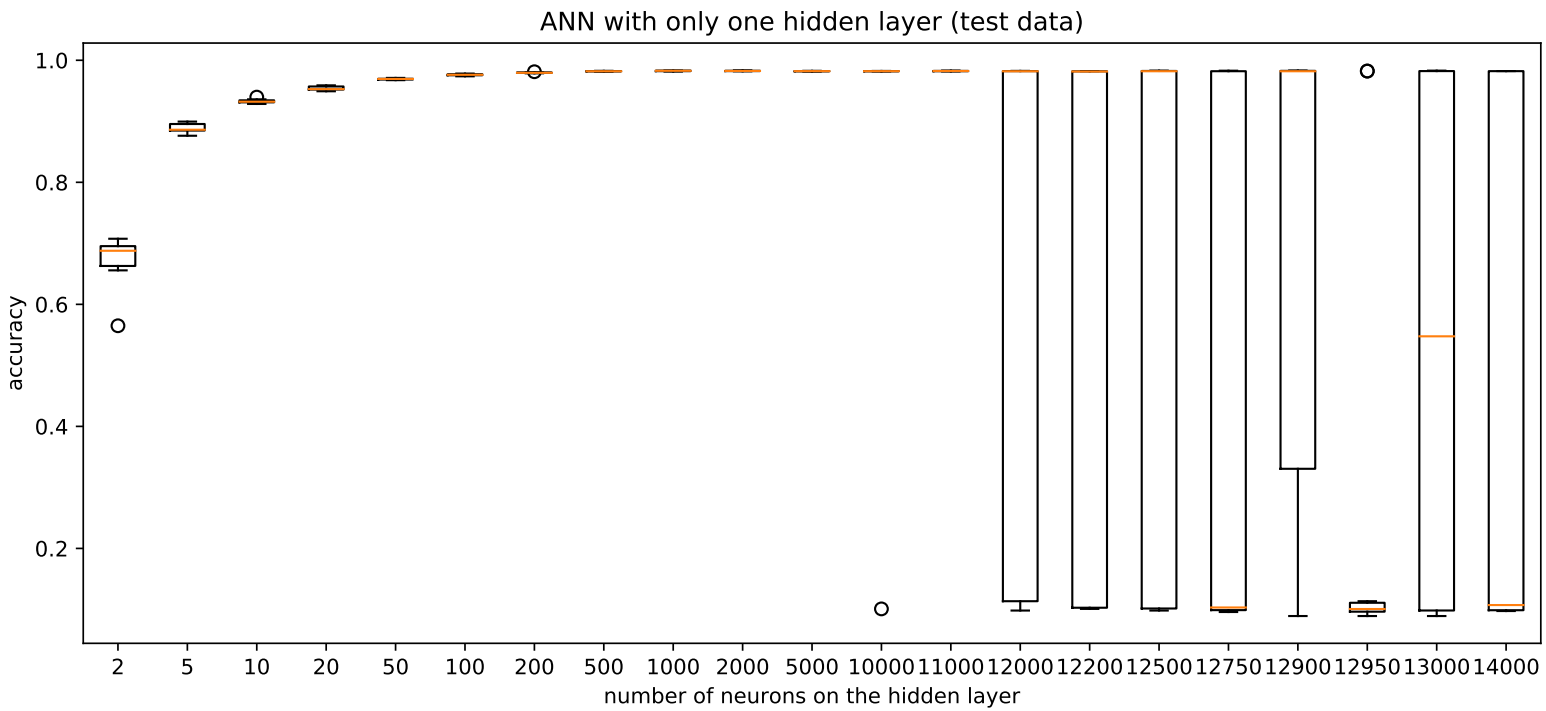


Close-up using 100 trials



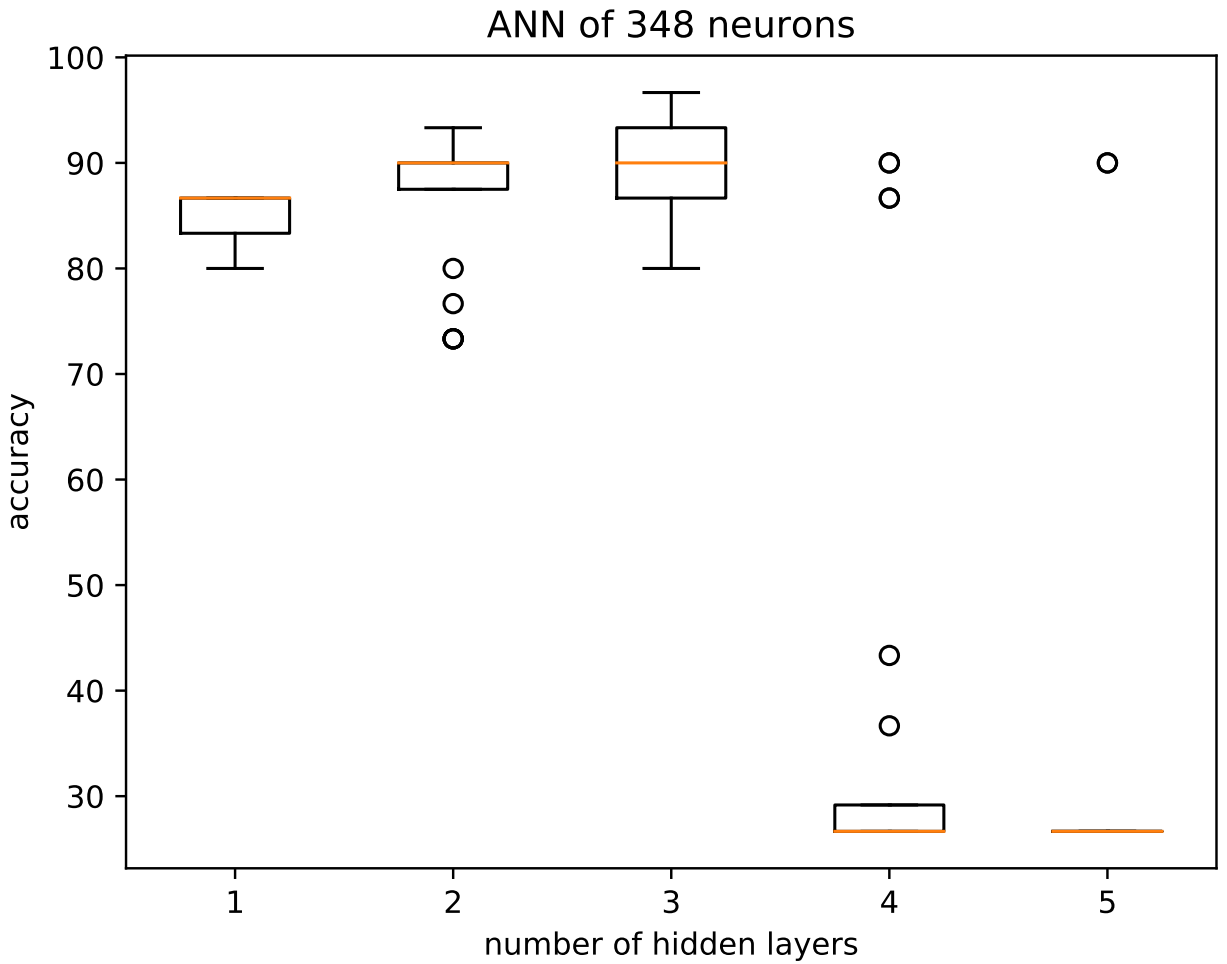
One hidden layer. Influence of the number of neurons

- MNIST data set (10 classes, 70000 data, 784 features)
- ANN using one hidden layer. 10 trials for each set of parameters
- sigmoid activation for hidden layers, softmax for output layer, optimizer RMSprop
- early stopping on validation accuracy, $\min\Delta = 10^{-4}$, patience of 20



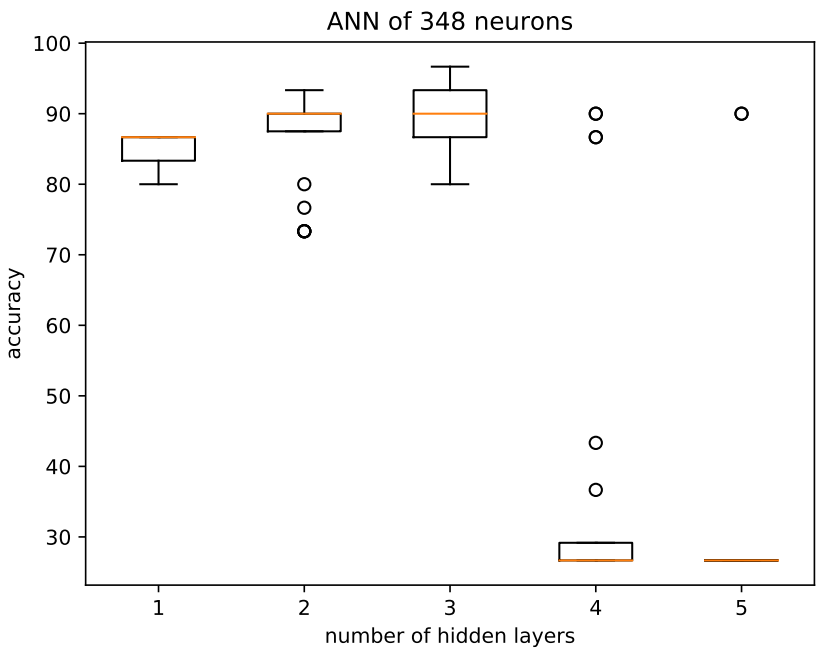
Experimenting the number of hidden layers

- Iris dataset (3 classes, 4 dimensionnal data).
- ANN using 348 neurons on 1 to 5 hidden layers. 24 trials for each set of parameters
- sigmoid activation for hidden layers, softmax for output layer, optimizer RMSprop
- early stopping on validation accuracy, $\min\Delta = 10^{-4}$, patience of 20

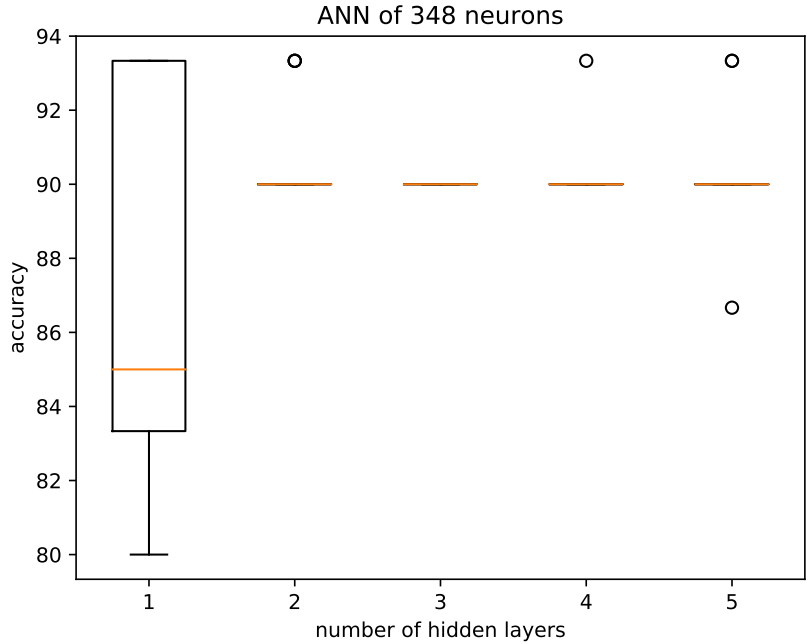


Same experiment comparing sigmoid to ReLu activation

- sigmoid activation



- ReLu activation



Tensorflow playground

http://playground.tensorflow.org

Conclusion

- Although Neural Networks kicked off the current phase of interest in Machine Learning, they are extremely problematic
  - Too many parameters (weights, learning rate, momentum, etc)
  - Hard to choose the architecture
  - Very slow to train
  - Easy to get stuck in local minima
- Other methods are of interest