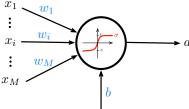
# PERCEPTRON MULTICOUCHES (MULTILAYER PERCEPTRON)

Jérémie Cabessa Laboratoire DAVID, UVSQ •0000000

Le **percepton** est un neurone qui agit comme un classifieur binaire. Sa dynamique est donnée par:

$$a = \sigma \left( \boldsymbol{w}^T \boldsymbol{x} + b \right)$$

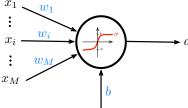
- $\mathbf{x} = (x_1, \dots, x_M) \in \mathbb{R}^M$  sont les inputs;



Le **percepton** est un neurone qui agit comme un classifieur binaire. Sa dynamique est donnée par:

$$a = \sigma \left( \boldsymbol{w}^T \boldsymbol{x} + b \right)$$

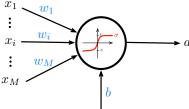
- $\mathbf{x} = (x_1, \dots, x_M) \in \mathbb{R}^M$  sont les inputs;
- $w = (w_1, \dots, w_M) \in \mathbb{R}^M$  sont les poids synaptiques;
- $b \in \mathbb{R}$  est le biais
- $\sigma: \mathbb{R} \to \mathbb{R}$  est une fonction d'activation sigmoïdale.
- a est l'activation du neurone



Le **percepton** est un neurone qui agit comme un classifieur binaire. Sa dynamique est donnée par:

$$a = \sigma \left( \boldsymbol{w}^T \boldsymbol{x} + b \right)$$

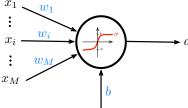
- $\mathbf{x} = (x_1, \dots, x_M) \in \mathbb{R}^M$  sont les inputs;
- $w = (w_1, \dots, w_M) \in \mathbb{R}^M$  sont les poids synaptiques;
- ▶  $b \in \mathbb{R}$  est le biais;
- $\sigma: \mathbb{R} \to \mathbb{R}$  est une fonction d'activation sigmoïdale.
- a est l'activation du neurone.



Le **percepton** est un neurone qui agit comme un classifieur binaire. Sa dynamique est donnée par:

$$a = \sigma \left( \boldsymbol{w}^T \boldsymbol{x} + b \right)$$

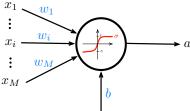
- $\mathbf{x} = (x_1, \dots, x_M) \in \mathbb{R}^M$  sont les inputs;
- $w = (w_1, \dots, w_M) \in \mathbb{R}^M$  sont les poids synaptiques;
- ▶  $b \in \mathbb{R}$  est le biais;
- $lackbox{\sigma}: \mathbb{R} \to \mathbb{R}$  est une fonction d'activation sigmoïdale.
- a est l'activation du neurone



Le **percepton** est un neurone qui agit comme un classifieur binaire. Sa dynamique est donnée par:

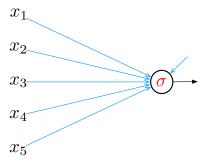
$$a = \sigma \left( \boldsymbol{w}^T \boldsymbol{x} + b \right)$$

- $\mathbf{x} = (x_1, \dots, x_M) \in \mathbb{R}^M$  sont les inputs;
- $w = (w_1, \dots, w_M) \in \mathbb{R}^M$  sont les poids synaptiques;
- ▶  $b \in \mathbb{R}$  est le biais;
- $ightharpoonup \sigma: \mathbb{R} o \mathbb{R}$  est une fonction d'activation sigmoïdale.
- ▶ a est l'activation du neurone.



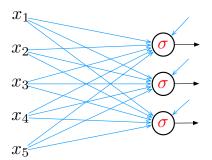
## PERCEPTRON: 1 NEURONE

$$a = \sigma\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$$
 où  $\boldsymbol{w} = (w_1, \dots, w_M)$ 



#### PERCEPTRON: 1 COUCHE

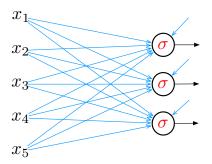
$$a_i = \sigma_i (\mathbf{w_i}^T \mathbf{x} + b_i)$$
 où  $\mathbf{w_i} = (w_{i1}, \dots, w_{iM}), i = 1, 2, 3.$ 



#### Perceptron: 1 couche

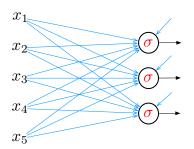
$$\boldsymbol{a} = \boldsymbol{\sigma} \left( \boldsymbol{W} \boldsymbol{x} + \boldsymbol{b} \right) \quad \text{où} \quad \boldsymbol{W} = \begin{pmatrix} \cdots \boldsymbol{w_1}^T \cdots \\ \cdots \boldsymbol{w_2}^T \cdots \\ \cdots \boldsymbol{w_3}^T \cdots \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1M} \\ w_{21} & \cdots & w_{2M} \\ w_{31} & \cdots & w_{3M} \end{pmatrix}, \ \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

et  $\sigma = (\sigma_1, \sigma_2, \sigma_3)$  appliquée composante par composante.



#### PERCEPTRON: 1 COUCHE

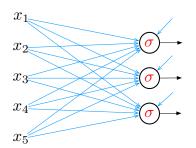
$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \pmb{\sigma} \left[ \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \right]$$



 $\blacktriangleright w_{ij}$ : poids de l'input j vers neurone i (et non de i vers j).

#### PERCEPTRON: 1 COUCHE

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \pmb{\sigma} \left[ \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \right]$$



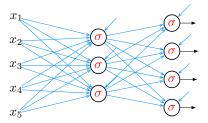
▶  $w_{ij}$ : poids de l'input j vers neurone i (et non de i vers j).



#### PERCEPTRON: 2 COUCHES

$$egin{array}{lcl} a^{[1]} & = & \sigma \left( W^{[1]} x + b^{[1]} 
ight) & := & \sigma \left( z^{[1]} 
ight) \ a^{[2]} & = & \sigma \left( W^{[2]} a^{[1]} + b^{[2]} 
ight) & := & \sigma \left( z^{[2]} 
ight) \end{array}$$

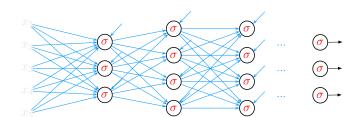
$$\begin{array}{llll} \text{O\`{u}} & \boldsymbol{W^{[i]}} = \begin{pmatrix} \cdots \boldsymbol{w_{1}^{[i]^{T}}} & \cdots \\ \cdots & \cdots & \cdots \\ \cdots \boldsymbol{w_{l_{i}}^{[i]^{T}}} & \cdots \end{pmatrix} = \begin{pmatrix} w_{11}^{[i]} & \cdots & w_{1l_{i-1}}^{[i]} \\ \cdots & \cdots & \cdots \\ w_{l_{i}1}^{[i]} & \cdots & w_{l_{i}l_{i-1}}^{[i]} \end{pmatrix} & \text{et} & \boldsymbol{b} = \begin{pmatrix} b_{1}^{[i]} \\ \vdots \\ b_{l_{i}}^{[i]} \end{pmatrix} & i = 1, 2 \end{array}$$



$$egin{cases} m{a^{[0]}} &= m{x} \ m{z^{[l]}} &= m{W^{[l]}}m{a^{[l-1]}} + m{b^{[l]}}, \ m{a^{[l]}} &= m{\sigma\left(m{z^{[l]}}
ight)} \end{cases}$$

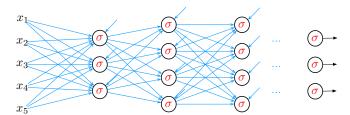
 $\triangleright x$  est l'input et  $a^{\lfloor L \rfloor}$  est l'output

lacksquare  $W^{[l]} \in \mathbb{R}^{\lambda_l} imes \mathbb{R}^{\lambda_{l-1}}$  et  $b^{[l]} \in \mathbb{R}^{\lambda_l}$ , où  $\lambda_l$  taille de la couche l



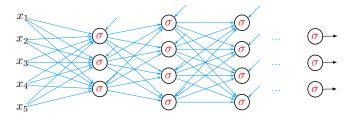
$$egin{cases} m{a^{[0]}} &= m{x} \ m{z^{[l]}} &= m{W^{[l]}}m{a^{[l-1]}} + m{b^{[l]}}, \ m{a^{[l]}} &= m{\sigma\left(m{z^{[l]}}
ight)} \end{cases}$$

- ightharpoonup x est l'input et  $a^{[L]}$  est l'output
- $m{W}^{[l]} \in \mathbb{R}^{\lambda_l} imes \mathbb{R}^{\lambda_{l-1}}$  et  $m{b}^{[l]} \in \mathbb{R}^{\lambda_l}$ , où  $\lambda_l$  taille de la couche l
- $w_{ij}^{[l]}$ : poids du neurone j (couche l-1) vers neurone i (couche l)



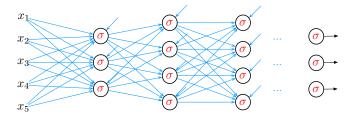
$$egin{cases} oldsymbol{a^{[0]}} &= oldsymbol{x} \ oldsymbol{z^{[l]}} &= oldsymbol{W^{[l]}} oldsymbol{a^{[l-1]}} + oldsymbol{b^{[l]}}, \ oldsymbol{a^{[l]}} &= oldsymbol{\sigma\left(oldsymbol{z^{[l]}}
ight)} \end{cases}$$

- lacksquare x est l'input et  $a^{[L]}$  est l'output
- $m{W}^{[l]} \in \mathbb{R}^{\lambda_l} imes \mathbb{R}^{\lambda_{l-1}}$  et  $m{b}^{[l]} \in \mathbb{R}^{\lambda_l}$ , où  $\lambda_l$  taille de la couche l
- $w_{ij}^{[l]}$ : poids du neurone j (couche l-1) vers neurone i (couche l)

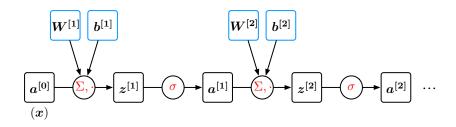


$$egin{cases} oldsymbol{a^{[0]}} &= oldsymbol{x} \ oldsymbol{z^{[l]}} &= oldsymbol{W^{[l]}} oldsymbol{a^{[l-1]}} + oldsymbol{b^{[l]}}, \ oldsymbol{a^{[l]}} &= oldsymbol{\sigma\left(oldsymbol{z^{[l]}}
ight)} \end{cases}$$

- lacksquare x est l'input et  $a^{[L]}$  est l'output
- $m{W}^{[l]} \in \mathbb{R}^{\lambda_l} imes \mathbb{R}^{\lambda_{l-1}}$  et  $m{b}^{[l]} \in \mathbb{R}^{\lambda_l}$ , où  $\lambda_l$  taille de la couche l
- $lacksquare w_{ij}^{[l]}$ : poids du neurone j (couche l-1) vers neurone i (couche l)

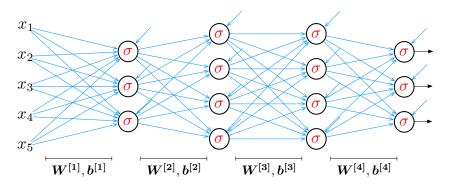


## MLP: REPRÉSENTATION GRAPHIQUE



carrés = variables ronds = opérations

$$egin{cases} m{a^{[0]}} &= m{x} \ m{z^{[l]}} &= m{W^{[l]}}m{a^{[l-1]}} + m{b^{[l]}}, \ m{a^{[l]}} &= m{\sigma\left(m{z^{[l]}}
ight)} \end{cases}$$



```
Algorithm 1: MLP: forward pass
```

```
\begin{aligned} & \mathsf{Data:} \; \mathsf{dataset} = \{(x_i, y_i) : i = 1, \dots, N\} \\ & \mathsf{Inputs:} \; \mathsf{MLP} = \left\{ \begin{pmatrix} \boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]} \end{pmatrix} : l = 1, \dots, L \right\} \\ & \mathsf{predictions} = [] \\ & \mathsf{for} \; i = 1 \; to \; N \; \mathsf{do} \\ & & | \; a^{[0]} = x_i \\ & \mathsf{for} \; l = 1 \; to \; L \; \mathsf{do} \\ & & | \; z^{[l]} = \boldsymbol{W}^{[l]} a^{[l-1]} + b^{[l]} \\ & & | \; a^{[l]} = \sigma \left( z^{[l]} \right) \\ & \mathsf{end} \\ & \mathsf{predictions.append}(\boldsymbol{a}^{[L]}) \end{aligned}
```

```
Algorithm 1: MLP: forward pass
```

```
Algorithm 1: MLP: forward pass
```

```
\begin{array}{l} \textbf{Data: } \operatorname{dataset} = \{(x_i,y_i): i=1,\dots,N\} \\ \textbf{Inputs: } \operatorname{MLP} = \left\{ \begin{pmatrix} \boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]} \end{pmatrix}: l=1,\dots,L \right\} \\ \textbf{predictions} = [] \\ \textbf{for } i=1 \ to \ N \ \textbf{do} \\ & \begin{vmatrix} a^{[0]} = x_i \\ \text{for } l=1 \ to \ L \ \textbf{do} \\ & a^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \\ & a^{[l]} = \sigma \left( z^{[l]} \right) \\ \textbf{end} \\ & \text{predictions.append}(a^{[L]}) \\ \textbf{end} \\ \end{array}
```

```
Data: dataset = \{(x_i, y_i) : i = 1, ..., N\}
Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, ..., L\}
predictions = []
for i = 1 to N do
      a^{[0]} = x_i
      for l=1 to L do
                                                                                     // forward pass
            z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}
           oldsymbol{a^{[l]}} = oldsymbol{\sigma}\left(oldsymbol{z^{[l]}}
ight)
      end
```

```
Data: dataset = \{(x_i, y_i) : i = 1, ..., N\}
Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, ..., L\}
predictions = []
for i = 1 to N do
      a^{[0]} = x_i
      for l=1 to L do
                                                                                   // forward pass
           z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}
          oldsymbol{a^{[l]}} = oldsymbol{\sigma}\left(oldsymbol{z^{[l]}}
ight)
      end
      predictions.append(a^{[L]})
end
```

```
Algorithm 1: MLP: forward pass
Data: dataset = \{(x_i, y_i) : i = 1, ..., N\}
Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, ..., L\}
predictions = []
for i = 1 to N do
     a^{[0]} = x_i
     for l=1 to L do
                                                                               // forward pass
           z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]}
          oldsymbol{a^{[l]}} = oldsymbol{\sigma}\left(oldsymbol{z^{[l]}}
ight)
     end
      predictions.append(a^{[L]})
end
return predictions
```

- ▶ Soit le dataset  $S = \{(\boldsymbol{x_i}, \boldsymbol{y_i}) : i = 1, \dots, N\}$
- On peut *paralléliser* la forward pass en passant les data "batch par batch" (batches de taille  $B = 32, 64, \dots$ ).
- Le i-ème batch  $B_i=(X_i,Y_i)$  est composé de B inputs et outputs  $m{x_k}$  et  $m{y_k}$  alignés en deux matrices:

$$egin{aligned} X_i = egin{pmatrix} dots & dots & \ldots & dots \ x_1 & x_2 & \cdots & x_B \ dots & dots & \ldots & dots \end{pmatrix} ext{ et } Y_i = egin{pmatrix} dots & dots & \ldots & dots \ y_1 & y_2 & \cdots & y_B \ dots & dots & \ldots & dots \end{pmatrix} \end{aligned}$$

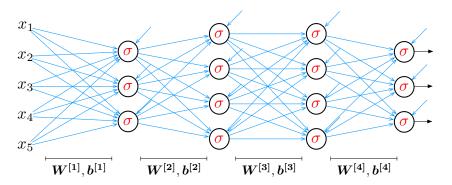
- ► Soit le dataset  $S = \{(x_i, y_i) : i = 1, ..., N\}$
- On peut *paralléliser* la forward pass en passant les data "batch par batch" (batches de taille  $B=32,64,\ldots$ ).
- Le *i*-ème batch  $B_i = (X_i, Y_i)$  est composé de B inputs et outputs  $x_k$  et  $y_k$  alignés en deux matrices:

$$m{X_i} = egin{pmatrix} dots & dots & \cdots & dots \ m{x_1} & m{x_2} & \cdots & m{x_B} \ dots & dots & \cdots & dots \end{pmatrix} ext{ et } m{Y_i} = egin{pmatrix} dots & dots & \cdots & dots \ m{y_1} & m{y_2} & \cdots & m{y_B} \ dots & dots & \cdots & dots \end{pmatrix}$$

- ► Soit le dataset  $S = \{(x_i, y_i) : i = 1, ..., N\}$
- ▶ On peut *paralléliser* la forward pass en passant les data "batch par batch" (batches de taille B = 32, 64, ...).
- Le i-ème batch  $B_i = (X_i, Y_i)$  est composé de B inputs et outputs  $x_k$  et  $y_k$  alignés en deux matrices:

$$egin{aligned} egin{aligned} egin{aligned} egin{aligned} egin{aligned} dar{x_1} & egin{aligned} egin{aligned} dar{x_2} & \cdots & egin{aligned} egin{aligned} dar{x_2} & \cdots & egin{aligned} egin{aligned} dar{x_1} & egin{aligned} egin{aligned} dar{x_2} & \cdots & egin{aligned} egin{aligned} dar{x_1} & egin{aligned} egin{aligned} dar{x_2} & \cdots & egin{aligned} egin{aligned} dar{x_1} & egin{aligned} egin{aligned} dar{x_2} & \cdots & egin{aligned} egin{aligned} dar{x_1} & egin{aligned} egin{aligned} dar{x_1} & egin{aligned} ar{x_2} & \cdots & egin{aligned} ar{x_2} & \cdots & egin{aligned} ar{x_1} & egin{aligned} ar{x_1} & egin{aligned} ar{x_2} & \cdots & egin{aligned} ar{x_1} & egin{aligned} ar{x_1} & ar{x_2} & \cdots & egin{aligned} ar{x_2} & egin{aligned} ar{x_1} & ar{x_2} & \cdots & egin{aligned} ar{x_2} & \cdots & egin{alig$$

$$\begin{cases} A^{[0]} = X_i \\ Z^{[l]} = W^{[l]}A^{[l-1]} \oplus b^{[l]}, \\ A^{[l]} = \sigma(Z^{[l]}), \quad l = 1, \dots, L \end{cases} \quad l = 1, \dots, L$$



```
Algorithm 2: MLP: forward pass (batched)
```

```
predictions.append(A^{[L]}
```

return concat(predictions)

```
Data: dataloader = \{B_i = (X_i, Y_i) : i = i, \dots, nb\_batches\}

Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, \dots, L\}

predictions = []

for i = 1 to nb\_batches do // loop over batches \begin{vmatrix} A^{[0]} = X_i \\ for \ l = 1 \ to \ L \ do \\ Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]} \\ A^{[l]} = \sigma \left(Z^{[l]}\right)
end predictions.append(A^{[L]})
```

```
Algorithm 2: MLP: forward pass (batched)
```

```
 \begin{aligned} \mathbf{Data:} & \text{ dataloader} = \{B_i = (X_i, Y_i) : i = i, \dots, nb\_batches\} \\ \mathbf{Inputs:} & \text{ MLP} = \left\{ \left( \boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]} \right) : l = 1, \dots, L \right\} \\ \text{predictions} &= [] \\ \mathbf{for} & i = 1 \ to \ nb\_batches \ \mathbf{do} \\ & A^{[0]} = X_i \\ & \mathbf{for} & l = 1 \ to \ L \ \mathbf{do} \\ & & Z^{[l]} = \boldsymbol{W}^{[l]} \boldsymbol{A}^{[l-1]} + \boldsymbol{b}^{[l]} \\ & & A^{[l]} = \sigma \left( \boldsymbol{Z}^{[l]} \right) \\ & \mathbf{end} \\ & \mathbf{predictions.append}(\boldsymbol{A}^{[L]}) \end{aligned}
```

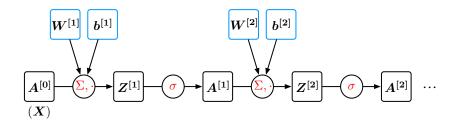
return concat(prediction

```
Algorithm 2: MLP: forward pass (batched)
```

4□ > 4□ > 4 = > 4 = > = 900

```
Algorithm 2: MLP: forward pass (batched)
Data: dataloader = {B_i = (X_i, Y_i) : i = i, ..., nb \ batches}
Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, ..., L\}
predictions = []
for i = 1 to nb batches do
                                                              // loop over batches
     A^{[0]} = X_i
     for l=1 to L do
                                                                    // forward pass
         Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
        A^{[l]} = \sigma\left(Z^{[l]}
ight)
     end
     predictions.append(A^{[L]})
end
return concat(predictions)
```

# MLP: REPRÉSENTATION GRAPHIQUE (BATCHED)



carrés = variables ronds = opérations

## **BIBLIOGRAPHIE**



Fleuret, F. (2022). Deep Learning Course.

