Perceptron multicouches (Multilayer Perceptron)

Jérémie Cabessa

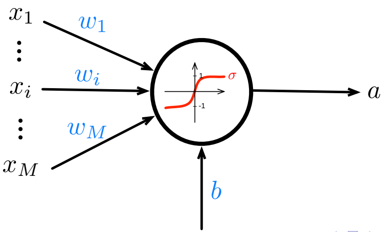
Laboratoire DAVID, UVSQ

Perceptron

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

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

- $\boldsymbol{x} = (x_1, \ldots, x_M) \in \mathbb{R}^M$ sont les inputs;
- $\boldsymbol{w} = (w_1, \ldots, w_M) \in \mathbb{R}^M$ 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.

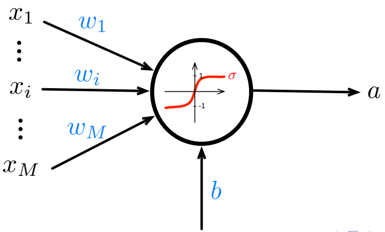


Perceptron

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

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

- $\boldsymbol{x} = (x_1, \ldots, x_M) \in \mathbb{R}^M$ sont les inputs;
- $\boldsymbol{w} = (w_1, \ldots, w_M) \in \mathbb{R}^M$ 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.

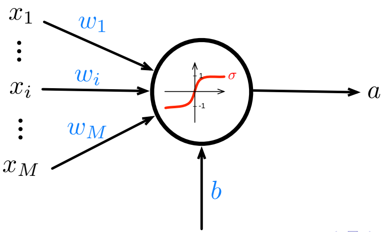


Perceptron

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

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

- $\boldsymbol{x} = (x_1, \ldots, x_M) \in \mathbb{R}^M$ sont les inputs;
- $\boldsymbol{w} = (w_1, \ldots, w_M) \in \mathbb{R}^M$ 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.

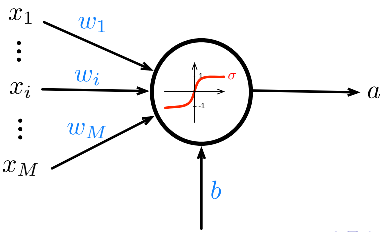


Perceptron

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

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

- $\boldsymbol{x} = (x_1, \ldots, x_M) \in \mathbb{R}^M$ sont les inputs;
- $\boldsymbol{w} = (w_1, \ldots, w_M) \in \mathbb{R}^M$ 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.

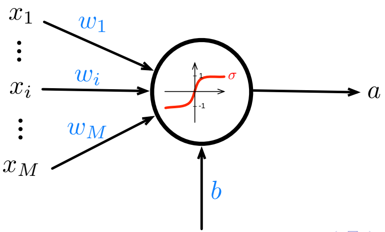


Perceptron

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

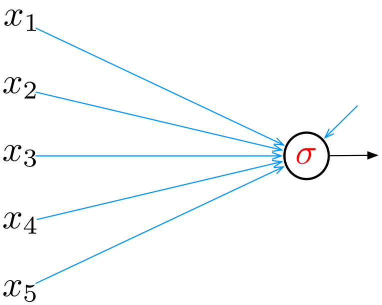
$$a = \sigma \left(\boldsymbol{w}^T \boldsymbol{x} + b \right)$$

- $\boldsymbol{x} = (x_1, \ldots, x_M) \in \mathbb{R}^M$ sont les inputs;
- $\boldsymbol{w} = (w_1, \ldots, w_M) \in \mathbb{R}^M$ 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.



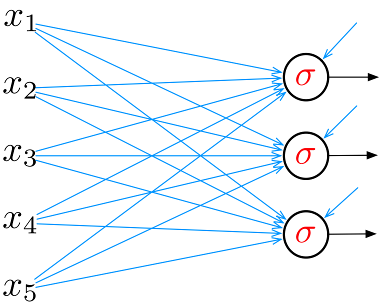
Perceptron: 1 neurone

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



Perceptron: 1 couche

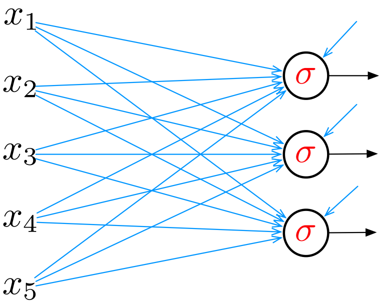
$$a_i = \sigma_i \left(\boldsymbol{w_i}^T \boldsymbol{x} + b_i \right) \text{ où } \boldsymbol{w_i} = (w_{i1}, \ldots, w_{iM}), \; i = 1, 2, 3.$$



Perceptron: 1 couche

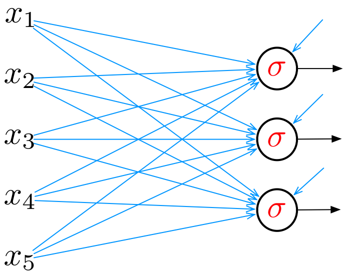
$$\boldsymbol{a} = \boldsymbol{\sigma}\left(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}\right) \quad \text{où} \quad \boldsymbol{W} = \begin{pmatrix} \cdots \boldsymbol{w_1}^T \cdots \\ \cdots \boldsymbol{w_2}^T \cdots \\ \cdots \boldsymbol{w_3}^T \cdots \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots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

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



Perceptron: 1 couche

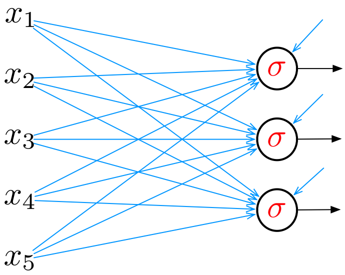
$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \boldsymbol{\sigma} \left[\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}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



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

Perceptron: 1 couche

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

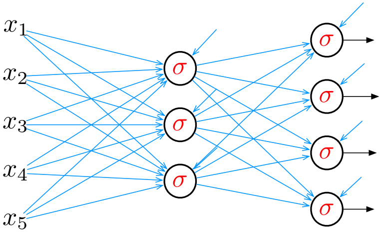


- w_{ij}: poids de l'input j vers neurone i (et non de i vers j).

Perceptron: 2 couches

$$
\begin{aligned}
\boldsymbol{a}^{[1]} &= \boldsymbol{\sigma}\left(\boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}\right) &:= \boldsymbol{\sigma}\left(\boldsymbol{z}^{[1]}\right) \\
\boldsymbol{a}^{[2]} &= \boldsymbol{\sigma}\left(\boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}\right) &:= \boldsymbol{\sigma}\left(\boldsymbol{z}^{[2]}\right)
\end{aligned}
$$

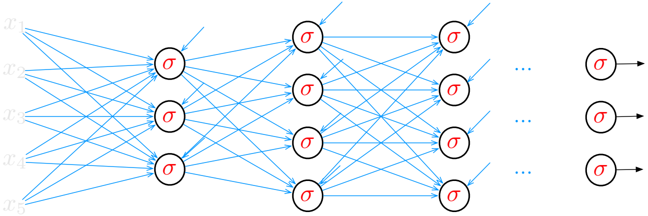
où $\quad \boldsymbol{W}^{[i]} = \begin{pmatrix} \cdots \boldsymbol{w}_{\boldsymbol{1}}^{[\boldsymbol{i}]^T} \cdots \\ \cdots \quad \cdots \quad \cdots \\ \cdots \boldsymbol{w}_{\boldsymbol{l_i}}^{[\boldsymbol{i}]^T} \cdots \end{pmatrix} = \begin{pmatrix} w_{11}^{[i]} & \cdots & w_{1l_{i-1}}^{[i]} \\ \cdots & \cdots & \cdots \\ w_{l_i1}^{[i]} & \cdots & w_{l_il_{i-1}}^{[i]} \end{pmatrix}$ et $\boldsymbol{b} = \begin{pmatrix} b_1^{[i]} \\ \vdots \\ b_{i_i}^{[i]} \end{pmatrix} \quad i = 1, 2$



Multi Layer Perceptron (MLP)

$$\begin{cases} \boldsymbol{a}^{[0]} &= \boldsymbol{x} \\ \boldsymbol{z}^{[l]} &= \boldsymbol{W}^{[l]} \boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}, \\ \boldsymbol{a}^{[l]} &= \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right) \end{cases} \qquad l = 1, \ldots, L$$

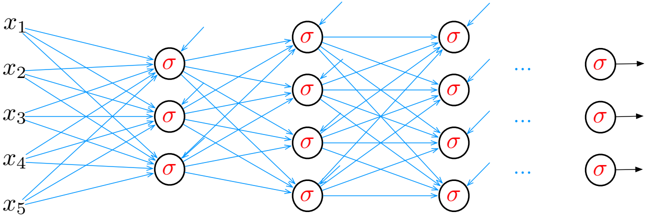
- $\boldsymbol{x}$ est l'input et $\boldsymbol{a}^{[L]}$ est l'output
- $\boldsymbol{W}^{[l]} \in \mathbb{R}^{\lambda_l} \times \mathbb{R}^{\lambda_{l-1}}$ et $\boldsymbol{b}^{[l]} \in \mathbb{R}^{\lambda_l}$, où λ_l taille de la couche l
- $w_{ij}^{[l]}$: poids du neurone j (couche $l-1$) vers neurone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



Multi Layer Perceptron (MLP)

$$\begin{cases} \boldsymbol{a}^{[0]} &= \boldsymbol{x} \\ \boldsymbol{z}^{[l]} &= \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}, \\ \boldsymbol{a}^{[l]} &= \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right) \end{cases} \quad l = 1, \ldots, L$$

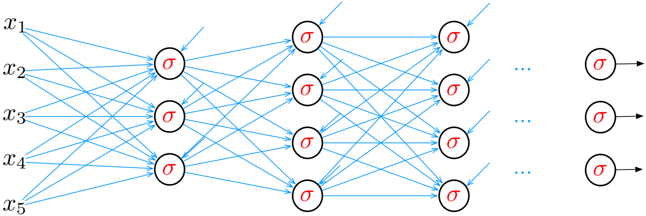
- $\boldsymbol{x}$ est l'input et $\boldsymbol{a}^{[L]}$ est l'output
- $\boldsymbol{W}^{[l]} \in \mathbb{R}^{\lambda_l} \times \mathbb{R}^{\lambda_{l-1}}$ et $\boldsymbol{b}^{[l]} \in \mathbb{R}^{\lambda_l}$, où λ_l taille de la couche l
- $w_{ij}^{[l]}$: poids du neurone j (couche $l-1$) vers neurone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



Multi Layer Perceptron (MLP)

$$\begin{cases} \boldsymbol{a}^{[0]} & = \quad \boldsymbol{x} \\ \boldsymbol{z}^{[l]} & = \quad \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}, \\ \boldsymbol{a}^{[l]} & = \quad \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right) \end{cases} \qquad l = 1, \ldots, L$$

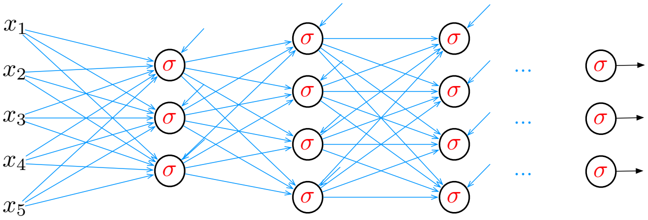
- $\boldsymbol{x}$ est l'input et $\boldsymbol{a}^{[L]}$ est l'output
- $\boldsymbol{W}^{[l]} \in \mathbb{R}^{\lambda_l} \times \mathbb{R}^{\lambda_{l-1}}$ et $\boldsymbol{b}^{[l]} \in \mathbb{R}^{\lambda_l}$, où λ_l taille de la couche l
- $w_{ij}^{[l]}$: poids du neurone j (couche $l-1$) vers neurone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



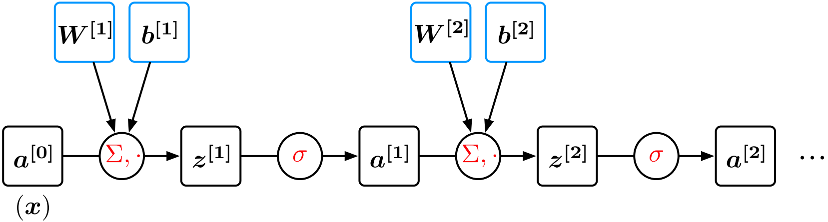
Multi Layer Perceptron (MLP)

$$\begin{cases} \boldsymbol{a}^{[0]} &= \boldsymbol{x} \\ \boldsymbol{z}^{[l]} &= \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}, \\ \boldsymbol{a}^{[l]} &= \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right) \end{cases} \quad l = 1, \ldots, L$$

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



MLP: représentation graphique

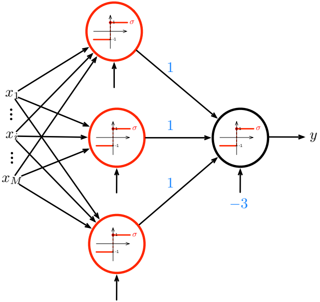


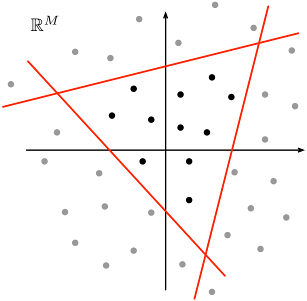
carrés = variables

ronds = opérations

Séparabilité non-linéaire

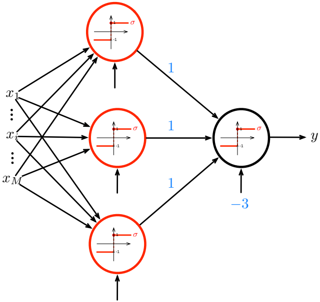
- **Rappel:** dans le cas de perceptrons avec fonction d'activation heaviside, chaque neurone représente un hyperplan dans l'espace de ses inputs.
- **Remarque importante:** un réseau de perceptron à 2 couches peut alors implémenter une frontière de décision non-linéaire!

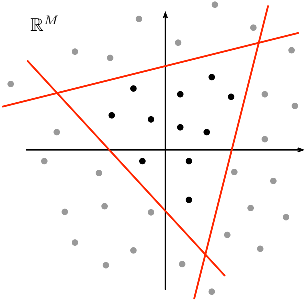




Séparabilité non-linéaire

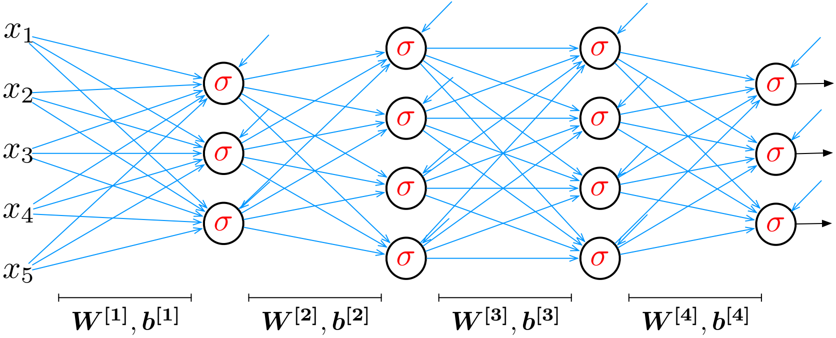
- **Rappel:** dans le cas de perceptrons avec fonction d'activation heaviside, chaque neurone représente un hyperplan dans l'espace de ses inputs.
- **Remarque importante:** un réseau de perceptron à 2 couches peut alors implémenter une frontière de décision non-linéaire!





MLP: forward pass

$$\begin{cases} \boldsymbol{a}^{[0]} & = & \boldsymbol{x} \\ \boldsymbol{z}^{[l]} & = & \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}, \\ \boldsymbol{a}^{[l]} & = & \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right) \end{cases} \quad l = 1, \ldots, L$$



MLP: forward pass

Algorithm 1: MLP: forward pass

Data: dataset $= \{(\boldsymbol{x_i}, \boldsymbol{y_i}) : i = 1, \ldots, N\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \ldots, L\right\}$

```
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]
        a^[l] = σ(z^[l])
    end
    predictions.append(a^[L])
end
return predictions
```

MLP: forward pass

Algorithm 1: MLP: forward pass

Data: dataset $= \{(\boldsymbol{x_i}, \boldsymbol{y_i}) : i = 1, \dots, N\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \dots, L\right\}$

predictions = [ ]
for $i = 1$ *to* N **do**
  $\boldsymbol{a}^{[0]} = \boldsymbol{x}_i$
  for $l = 1$ *to* L **do** // forward pass
    $\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}$
    $\boldsymbol{a}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right)$
  end
  predictions.append($\boldsymbol{a}^{[L]}$)
end
return predictions

MLP: forward pass

Algorithm 1: MLP: forward pass

Data: dataset $= \{(\boldsymbol{x_i}, \boldsymbol{y_i}) : i = 1, \ldots, N\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \ldots, L\right\}$

predictions $=$ [ ]
for $i = 1$ *to* N **do**
  $\boldsymbol{a}^{[0]} = \boldsymbol{x}_i$
  for $l = 1$ *to* L **do** `// forward pass`
    $\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]} \boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}$
    $\boldsymbol{a}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right)$
  end
  predictions.append($\boldsymbol{a}^{[L]}$)
end
return predictions

MLP: forward pass

Algorithm 1: MLP: forward pass

Data: dataset $= \{(\boldsymbol{x_i}, \boldsymbol{y_i}) : i = 1, \ldots, N\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \ldots, L\right\}$

predictions = [ ]
for $i = 1$ *to* N **do**
  $\boldsymbol{a}^{[0]} = \boldsymbol{x_i}$
  for $l = 1$ *to* L **do** `// forward pass`
    $\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]} \boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}$
    $\boldsymbol{a}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right)$
  end
  predictions.append($\boldsymbol{a}^{[L]}$)
end
return predictions

MLP: forward pass

Algorithm 1: MLP: forward pass

Data: dataset $= \{(\boldsymbol{x_i}, \boldsymbol{y_i}) : i = 1, \ldots, N\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \ldots, L\right\}$

predictions $=$ [ ]
for $i = 1$ *to* N **do**
  $\boldsymbol{a}^{[0]} = \boldsymbol{x_i}$
  for $l = 1$ *to* L **do** // forward pass
    $\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}$
    $\boldsymbol{a}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right)$
  end
  predictions.append($\boldsymbol{a}^{[L]}$)
end
return predictions

MLP: forward pass

Algorithm 1: MLP: forward pass

Data: dataset $= \{(\boldsymbol{x_i}, \boldsymbol{y_i}) : i = 1, \dots, N\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \dots, L\right\}$

predictions $=$ [ ]
for $i = 1$ *to* N **do**
    $\boldsymbol{a}^{[0]} = \boldsymbol{x_i}$
    for $l = 1$ *to* L **do** `// forward pass`
        $\boldsymbol{z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}$
        $\boldsymbol{a}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{z}^{[l]}\right)$
    end
    predictions.append($\boldsymbol{a}^{[L]}$)
end
return predictions

MLP: forward pass (batched)

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

$$\boldsymbol{X_i} = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_B} \\ \vdots & \vdots & \cdots & \vdots \end{pmatrix} \text{ et } \boldsymbol{Y_i} = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{y_1} & \boldsymbol{y_2} & \cdots & \boldsymbol{y_B} \\ \vdots & \vdots & \cdots & \vdots \end{pmatrix}$$

MLP: forward pass (batched)

- 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 $\boldsymbol{B_i} = (\boldsymbol{X_i}, \boldsymbol{Y_i})$ est composé de B inputs et outputs $\boldsymbol{x_k}$ et $\boldsymbol{y_k}$ alignés en deux matrices:

$$\boldsymbol{X_i} = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_B} \\ \vdots & \vdots & \cdots & \vdots \end{pmatrix} \text{ et } \boldsymbol{Y_i} = \begin{pmatrix} \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{y_1} & \boldsymbol{y_2} & \cdots & \boldsymbol{y_B} \\ \vdots & \vdots & \cdots & \vdots \end{pmatrix}$$

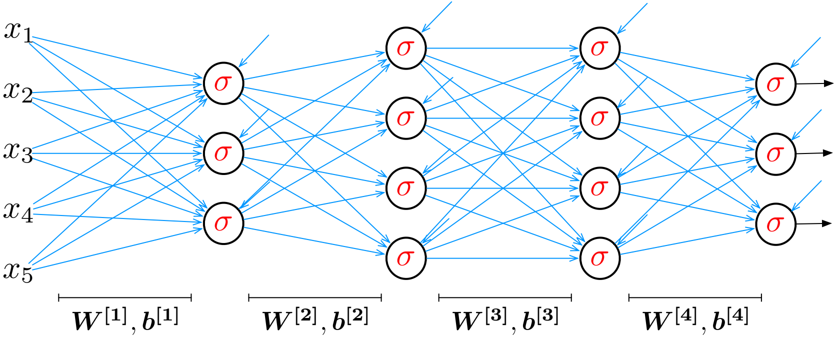
MLP: forward pass (batched)

- 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 $\boldsymbol{B_i} = (\boldsymbol{X_i}, \boldsymbol{Y_i})$ est composé de B inputs et outputs $\boldsymbol{x_k}$ et $\boldsymbol{y_k}$ alignés en deux matrices:

$$\boldsymbol{X_i} = \begin{pmatrix} \vdots & \vdots & \dots & \vdots \\ \boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_B} \\ \vdots & \vdots & \dots & \vdots \end{pmatrix} \text{ et } \boldsymbol{Y_i} = \begin{pmatrix} \vdots & \vdots & \dots & \vdots \\ \boldsymbol{y_1} & \boldsymbol{y_2} & \cdots & \boldsymbol{y_B} \\ \vdots & \vdots & \dots & \vdots \end{pmatrix}$$

MLP: forward pass (batched)

$$\begin{cases} \boldsymbol{A}^{[0]} &= \boldsymbol{X}_i \\ \boldsymbol{Z}^{[l]} &= \boldsymbol{W}^{[l]}\boldsymbol{A}^{[l-1]} \oplus \boldsymbol{b}^{[l]}, \\ \boldsymbol{A}^{[l]} &= \boldsymbol{\sigma}\left(\boldsymbol{Z}^{[l]}\right) \end{cases} \quad l = 1, \ldots, L$$



MLP: forward pass (batched)

Algorithm 2: MLP: forward pass (batched)

Data: dataloader = $\{\boldsymbol{B_i} = (\boldsymbol{X_i}, \boldsymbol{Y_i}) : i = 1, \ldots, B\}$
Inputs: MLP = $\left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \ldots, L\right\}$

```
predictions = [ ]
for i = 1 to B 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] = σ(Z^[l])
    end
    predictions.append(A^[L])
end
return concat(predictions)
```

MLP: forward pass (batched)

Algorithm 2: MLP: forward pass (batched)

Data: dataloader = $\{\boldsymbol{B_i} = (\boldsymbol{X_i}, \boldsymbol{Y_i}) : i = 1, \ldots, B\}$
Inputs: MLP = $\{(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}) : l = 1, \ldots, L\}$

predictions = [ ]
for $i = 1$ *to* B **do** `// loop over batches`
    $\boldsymbol{A}^{[0]} = \boldsymbol{X}_i$
    for $l = 1$ *to* L **do** `// forward pass`
        $\boldsymbol{Z}^{[l]} = \boldsymbol{W}^{[l]} \boldsymbol{A}^{[l-1]} + \boldsymbol{b}^{[l]}$
        $\boldsymbol{A}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{Z}^{[l]}\right)$
    end
    predictions.append($\boldsymbol{A}^{[L]}$)
end
return concat(predictions)

MLP: forward pass (batched)

Algorithm 2: MLP: forward pass (batched)

Data: dataloader $= \{\boldsymbol{B_i} = (\boldsymbol{X_i}, \boldsymbol{Y_i}) : i = 1, \ldots, B\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \ldots, L\right\}$

predictions = [ ]
for $i = 1$ *to* B **do** // loop over batches
  $\boldsymbol{A}^{[0]} = \boldsymbol{X}_i$
  for $l = 1$ *to* L **do** // forward pass
    $\boldsymbol{Z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{A}^{[l-1]} + \boldsymbol{b}^{[l]}$
    $\boldsymbol{A}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{Z}^{[l]}\right)$
  end
  predictions.append($\boldsymbol{A}^{[L]}$)
end
return concat(predictions)

MLP: forward pass (batched)

Algorithm 2: MLP: forward pass (batched)

Data: dataloader $= \{\boldsymbol{B_i} = (\boldsymbol{X_i}, \boldsymbol{Y_i}) : i = 1, \ldots, B\}$
Inputs: MLP $= \{(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}) : l = 1, \ldots, L\}$

predictions $=$ [ ]
for $i = 1$ *to* B **do** // loop over batches
    $\boldsymbol{A}^{[0]} = \boldsymbol{X_i}$
    for $l = 1$ *to* L **do** // forward pass
        $\boldsymbol{Z}^{[l]} = \boldsymbol{W}^{[l]} \boldsymbol{A}^{[l-1]} + \boldsymbol{b}^{[l]}$
        $\boldsymbol{A}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{Z}^{[l]}\right)$
    end
    predictions.append($\boldsymbol{A}^{[L]}$)
end
return concat(predictions)

MLP: forward pass (batched)

Algorithm 2: MLP: forward pass (batched)

Data: dataloader $= \{\boldsymbol{B_i} = (\boldsymbol{X_i}, \boldsymbol{Y_i}) : i = 1, \ldots, B\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \ldots, L\right\}$

predictions = [ ]
for $i = 1$ *to* B **do** // loop over batches
  $\boldsymbol{A}^{[0]} = \boldsymbol{X_i}$
  for $l = 1$ *to* L **do** // forward pass
    $\boldsymbol{Z}^{[l]} = \boldsymbol{W}^{[l]}\boldsymbol{A}^{[l-1]} + \boldsymbol{b}^{[l]}$
    $\boldsymbol{A}^{[l]} = \boldsymbol{\sigma}\left(\boldsymbol{Z}^{[l]}\right)$
  end
  predictions.append($\boldsymbol{A}^{[L]}$)
end
return concat(predictions)

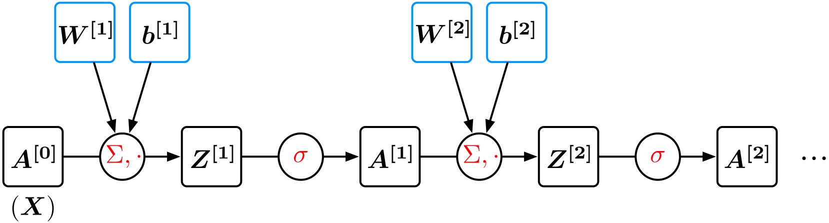
MLP: forward pass (batched)

Algorithm 2: MLP: forward pass (batched)

Data: dataloader $= \{\boldsymbol{B_i} = (\boldsymbol{X_i}, \boldsymbol{Y_i}) : i = 1, \ldots, B\}$
Inputs: MLP $= \left\{\left(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}\right) : l = 1, \ldots, L\right\}$

```
predictions = [ ]
for i = 1 to B 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] = σ(Z^[l])
    end
    predictions.append(A^[L])
end
return concat(predictions)
```

MLP: représentation graphique (batched)



carrés = variables

ronds = opérations

Bibliographie

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