

INTRODUCTION  
O

BIOLOGIE  
OOO

1<sup>ÈRE</sup> GÉNÉRATION  
OOOOOOOOOO  
OOOOO

2<sup>ÈME</sup> GÉNÉRATION  
OOOOOOOOOO  
OOOOOOOOOOOO

3<sup>ÈME</sup> GÉNÉRATION  
OOOOO  
OOOO

CONCLUSION  
OO

# COURS PSL

## RÉSEAUX DE NEURONES

DE LA BIOLOGIE À L'INTELLIGENCE ARTIFICIELLE

Jérémie Cabessa

Laboratoire DAVID, UVSQ

# INTRODUCTION

- ▶ Qu'est ce que l'intelligence artificielle (AI) et l'apprentissage automatique ou machine learning (ML)?
- ▶ Les réseaux de neurones désignent une *classe d'algorithmes* vaguement inspirés des réseaux de neurones biologiques.
- ▶ Survol des réseaux de neurones avec exemples d'applications:
  - ▶ Réseaux de neurones pour la reconnaissance d'images
  - ▶ Réseaux de neurones pour la reconnaissance de la parole
  - ▶ Réseaux de neurones de 3<sup>ème</sup> génération
- ▶ Compréhension générale des réseaux de neurones artificiels.
- ▶ Vers une démystification de l'intelligence artificielle et de l'apprentissage automatique.

# INTRODUCTION

- ▶ Qu'est ce que l'intelligence artificielle (AI) et l'apprentissage automatique ou machine learning (ML)?
- ▶ Les **réseaux de neurones** désignent une *classe d'algorithmes* vaguement inspirés des réseaux de neurones biologiques.
- ▶ Survol des réseaux de neurones avec exemples d'applications:
  - ▶ Réseaux de neurones de 1<sup>ère</sup> génération.
  - ▶ Réseaux de neurones de 2<sup>ème</sup> génération.
  - ▶ Compréhension générale des réseaux de neurones artificiels.
  - ▶ Vers une démystification de l'intelligence artificielle et de l'apprentissage automatique.

# INTRODUCTION

- ▶ Qu'est ce que l'intelligence artificielle (AI) et l'apprentissage automatique ou machine learning (ML)?
- ▶ Les **réseaux de neurones** désignent une *classe d'algorithmes* vaguement inspirés des réseaux de neurones biologiques.
- ▶ Survol des réseaux de neurones avec exemples d'applications:
  - ▶ Réseaux de neurones de 1<sup>ère</sup> génération.
  - ▶ Réseaux de neurones de 2<sup>ème</sup> génération.
  - ▶ Réseaux de neurones de 3<sup>ème</sup> génération.
- ▶ Compréhension générale des réseaux de neurones artificiels.
- ▶ Vers une démystification de l'intelligence artificielle et de l'apprentissage automatique.

# INTRODUCTION

- ▶ Qu'est ce que l'intelligence artificielle (AI) et l'apprentissage automatique ou machine learning (ML)?
- ▶ Les **réseaux de neurones** désignent une *classe d'algorithmes* vaguement inspirés des réseaux de neurones biologiques.
- ▶ Survol des réseaux de neurones avec exemples d'applications:
  - ▶ Réseaux de neurones de 1<sup>ère</sup> génération.
  - ▶ Réseaux de neurones de 2<sup>ème</sup> génération.
  - ▶ Réseaux de neurones de 3<sup>ème</sup> génération.
- ▶ Compréhension générale des réseaux de neurones artificiels.
- ▶ Vers une démystification de l'intelligence artificielle et de l'apprentissage automatique.

# INTRODUCTION

- ▶ Qu'est ce que l'**intelligence artificielle (AI)** et l'**apprentissage automatique ou machine learning (ML)**?
- ▶ Les **réseaux de neurones** désignent une *classe d'algorithmes* vaguement inspirés des réseaux de neurones biologiques.
- ▶ Survol des réseaux de neurones avec exemples d'applications:
  - ▶ Réseaux de neurones de 1<sup>ère</sup> génération.
  - ▶ Réseaux de neurones de 2<sup>ème</sup> génération.
  - ▶ Réseaux de neurones de 3<sup>ème</sup> génération.
- ▶ Compréhension générale des réseaux de neurones artificiels.
- ▶ Vers une démystification de l'intelligence artificielle et de l'apprentissage automatique.

# INTRODUCTION

- ▶ Qu'est ce que l'intelligence artificielle (AI) et l'apprentissage automatique ou machine learning (ML)?
- ▶ Les **réseaux de neurones** désignent une *classe d'algorithmes* vaguement inspirés des réseaux de neurones biologiques.
- ▶ Survol des réseaux de neurones avec exemples d'applications:
  - ▶ Réseaux de neurones de 1<sup>ère</sup> génération.
  - ▶ Réseaux de neurones de 2<sup>ème</sup> génération.
  - ▶ Réseaux de neurones de 3<sup>ème</sup> génération.
- ▶ Compréhension générale des réseaux de neurones artificiels.
- ▶ Vers une démystification de l'intelligence artificielle et de l'apprentissage automatique.

# INTRODUCTION

- ▶ Qu'est ce que l'**intelligence artificielle (AI)** et l'**apprentissage automatique ou machine learning (ML)**?
- ▶ Les **réseaux de neurones** désignent une *classe d'algorithmes* vaguement inspirés des réseaux de neurones biologiques.
- ▶ Survol des réseaux de neurones avec exemples d'applications:
  - ▶ Réseaux de neurones de 1<sup>ère</sup> génération.
  - ▶ Réseaux de neurones de 2<sup>ème</sup> génération.
  - ▶ Réseaux de neurones de 3<sup>ème</sup> génération.
- ▶ Compréhension générale des réseaux de neurones artificiels.
- ▶ Vers une démystification de l'intelligence artificielle et de l'apprentissage automatique.



# INTRODUCTION

- ▶ Qu'est ce que l'**intelligence artificielle (AI)** et l'**apprentissage automatique ou machine learning (ML)**?
- ▶ Les **réseaux de neurones** désignent une *classe d'algorithmes* vaguement inspirés des réseaux de neurones biologiques.
- ▶ Survol des réseaux de neurones avec exemples d'applications:
  - ▶ Réseaux de neurones de 1<sup>ère</sup> génération.
  - ▶ Réseaux de neurones de 2<sup>ème</sup> génération.
  - ▶ Réseaux de neurones de 3<sup>ème</sup> génération.
- ▶ Compréhension générale des réseaux de neurones artificiels.
- ▶ Vers une démystification de l'intelligence artificielle et de l'apprentissage automatique.

# NEURONE BIOLOGIQUE

- Corps cellulaire ou soma, axone, dendrites, et terminaisons axonales.

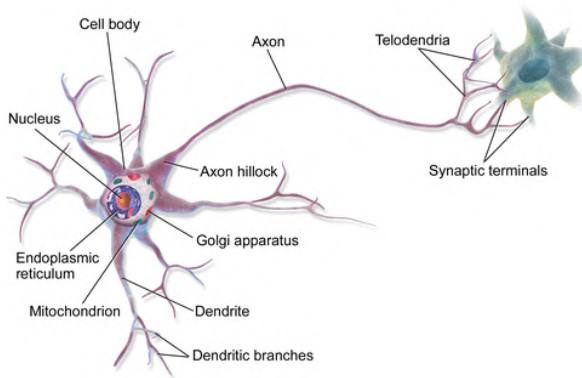


Figure taken from [Wikipedia](#)

# NEURONE BIOLOGIQUE

- Synapses: structures qui permettent de transférer un signal électrique ou chimique d'un neurone A à un autre B (depuis les terminaisons axonales de A vers les dendrites de B).

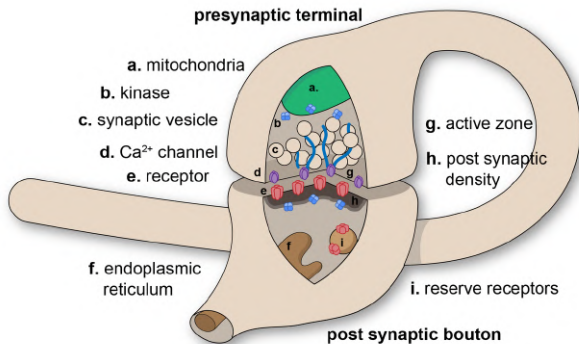


Figure taken from [Wikipedia](#)



# RÉSEAUX DE NEURONES DE 1<sup>ÈRE</sup> GÉNÉRATION

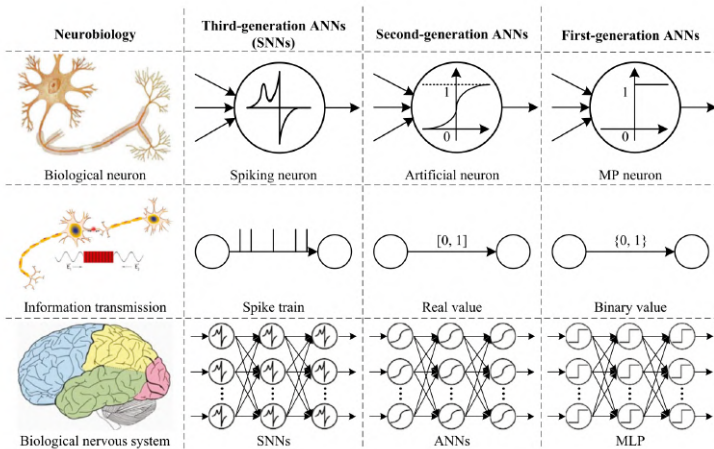
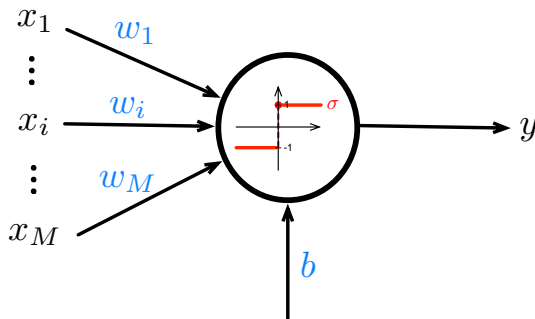


Figure taken from [Wang et al., 2020]

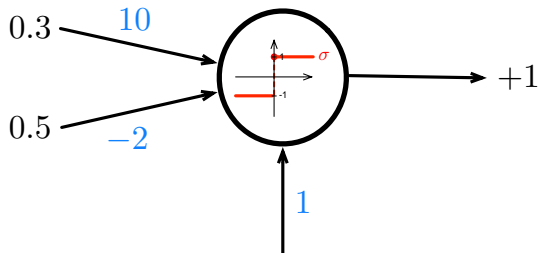
# PERCEPTRON

Le **perceptron** est un simple neurone qui agit comme un *classifieur binaire* [McCulloch and Pitts, 1943, Rosenblatt, 1957, Rosenblatt, 1958, Minsky and Papert, 1969].



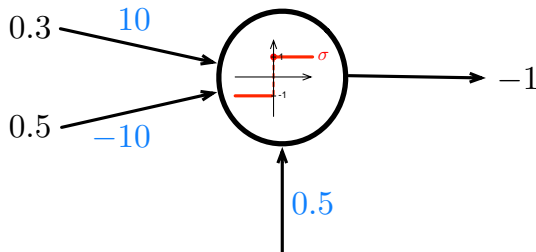
# PERCEPTRON

Exemples:



# PERCEPTRON

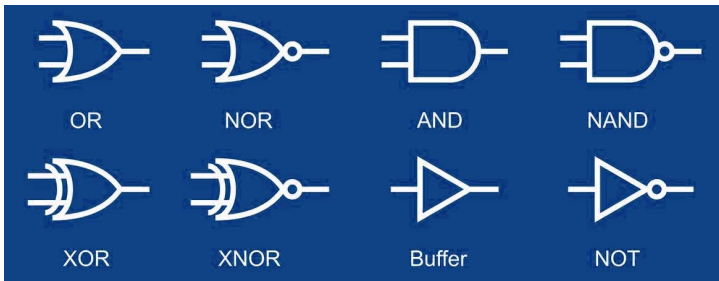
Exemples:





# PERCEPTRONS ET PORTES LOGIQUES

- Les perceptrons peuvent implémenter des *portes logiques*.



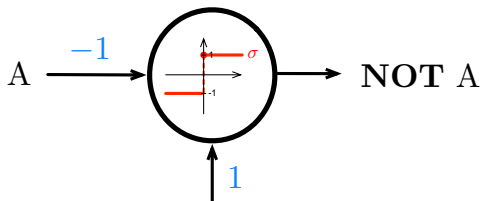
# PERCEPTRONS ET PORTES LOGIQUES

- **Exemple:** implémentation des portes logiques NOT, OR et AND.



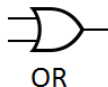
NOT

INPUT	OUTPUT
A	NOT A
0	1
1	0

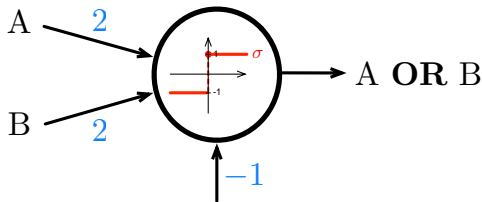


# PERCEPTRONS ET PORTES LOGIQUES

- **Exemple:** implémentation des portes logiques NOT, OR et AND.

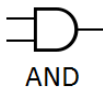


INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

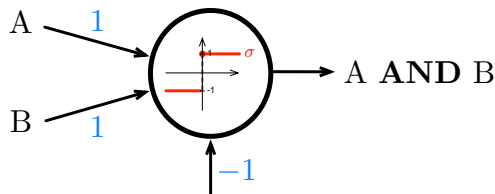


# PERCEPTRONS ET PORTES LOGIQUES

- **Exemple:** implémentation des portes logiques NOT, OR et AND.

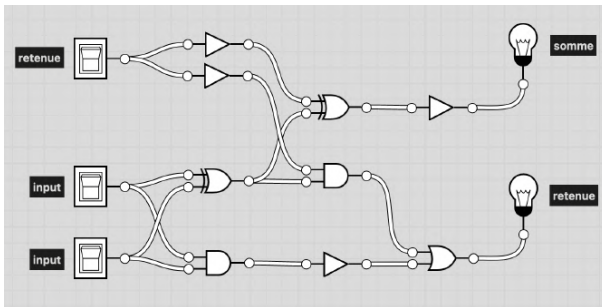


INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1



# PERCEPTRONS ET CIRCUITS LOGIQUES

- ▶ Ainsi, les perceptrons peuvent implémenter des *circuits logiques*, et donc des petits ordinateurs appelés *automates finis*.

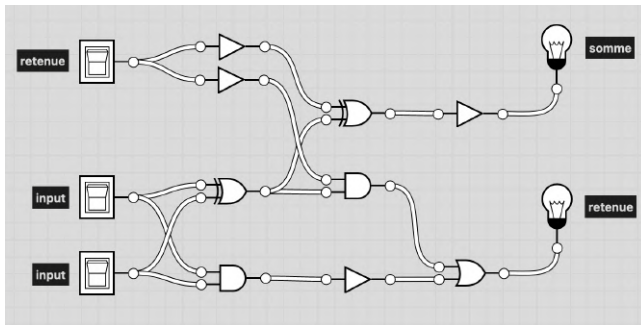


<https://logic.ly/demo/>

## PERCEPTRONS ET CIRCUITS LOGIQUES

- **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{ccccccc}
 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$

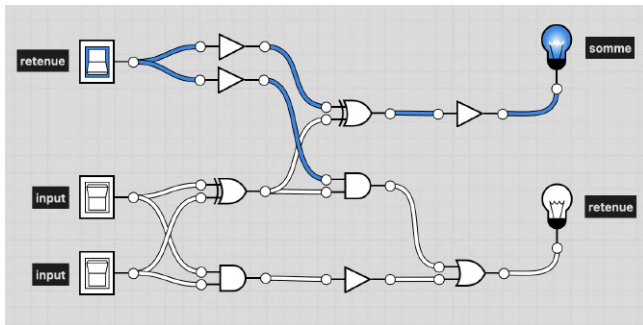




## PERCEPTRONS ET CIRCUITS LOGIQUES

► **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{ccccccc}
 & 1 & 1 & 0 & 1 & 1 & 1 \\
 + & 1 & 1 & 0 & 1 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 0
 \end{array}$$





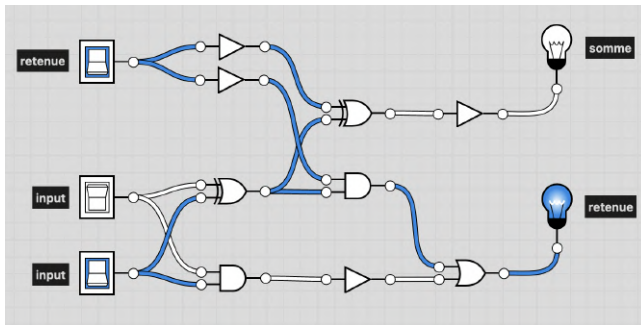




# PERCEPTRONS ET CIRCUITS LOGIQUES

► **Exemple:** implémentation de l'addition en binaire.

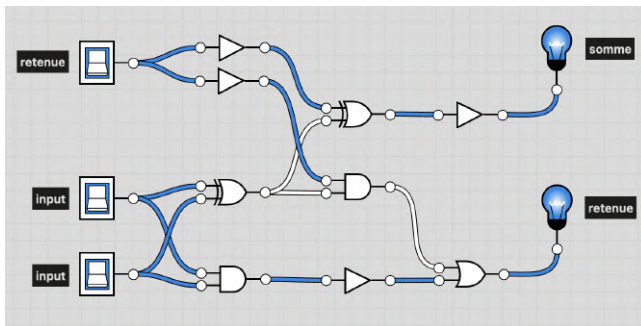
$$\begin{array}{ccccccc}
 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



## PERCEPTRONS ET CIRCUITS LOGIQUES

- **Exemple:** implémentation de l'addition en binaire.

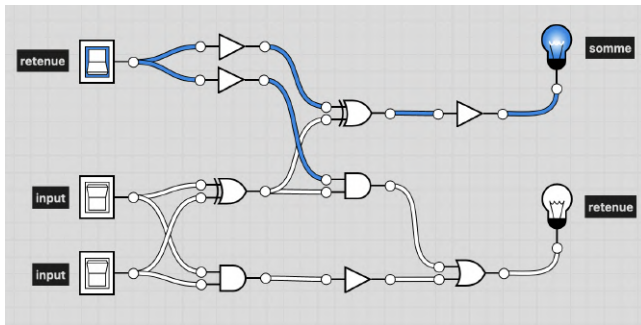
$$\begin{array}{ccccccc}
 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



## PERCEPTRONS ET CIRCUITS LOGIQUES

- **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{ccccccc}
 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$

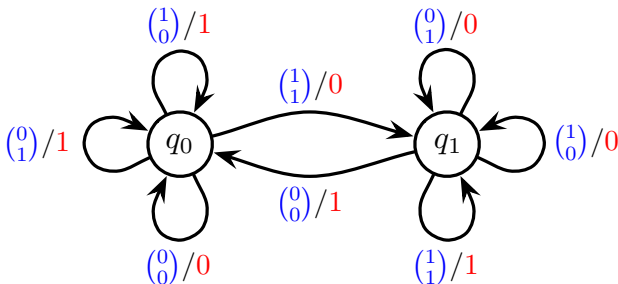


## JÉRÉMIE CABESSA

## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

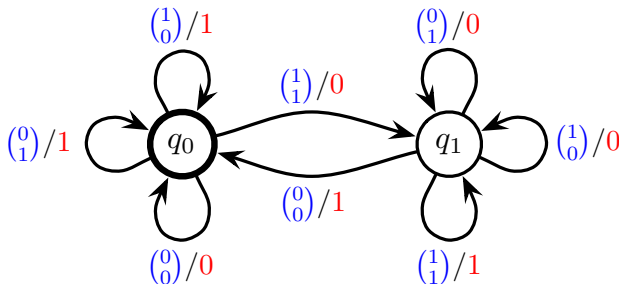
$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



# PERCEPTRONS ET AUTOMATES

► **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{r} \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ + \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \hline \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \end{array}$$





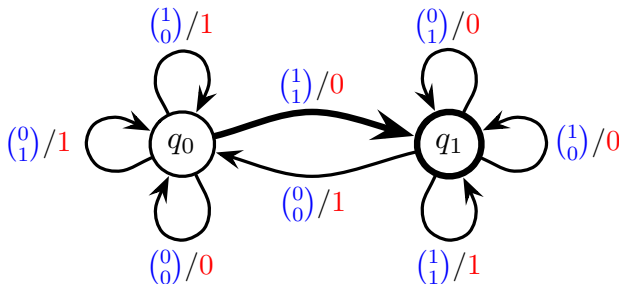
# PERCEPTRONS ET AUTOMATES

► **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{r} \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ + \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \hline \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \end{array}$$

1 1<sup>1</sup> 0<sup>1</sup> 1 1 0<sup>1</sup> 1

1 1 0 0 1 1 0



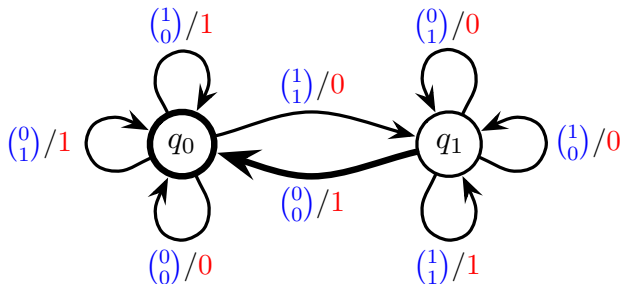
# PERCEPTRONS ET AUTOMATES

► **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{r} \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ + \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \hline \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \end{array}$$

1 1<sup>1</sup> 0<sup>1</sup> 1 1 0<sup>1</sup> 1

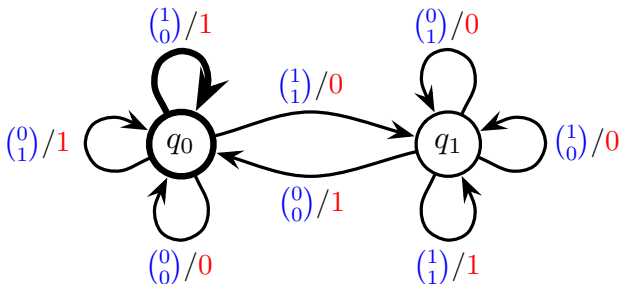
1 1 0 0 1 1 0



## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



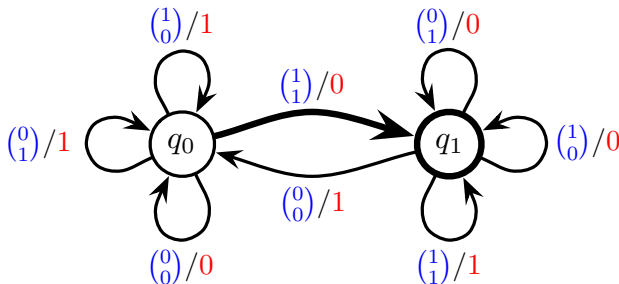
# PERCEPTRONS ET AUTOMATES

► **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{r} \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ + \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \hline \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \end{array}$$

1 1<sup>1</sup> 0<sup>1</sup> 1 1 0<sup>1</sup> 1

1 1 0 0 1 1 0



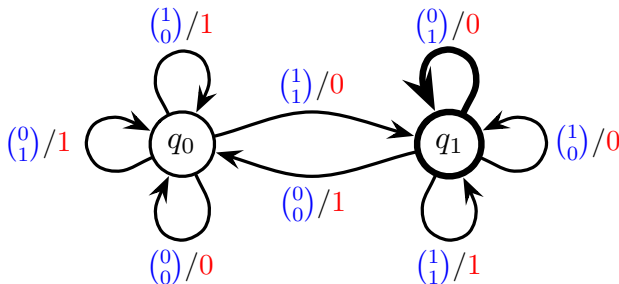
# PERCEPTRONS ET AUTOMATES

► **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{r} \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ + \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \hline \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \end{array}$$

1 1<sup>1</sup> 0<sup>1</sup> 1 1 0<sup>1</sup> 1

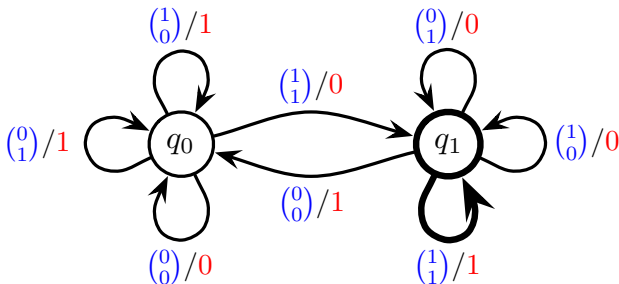
1 1 0 0 1 1 0



## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



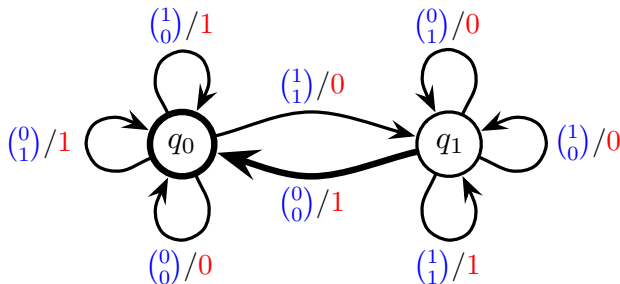
# PERCEPTRONS ET AUTOMATES

► **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{r} \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \phantom{+} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ + \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\ \hline \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \end{array}$$

1 1<sup>1</sup> 0<sup>1</sup> 1 1 0<sup>1</sup> 1

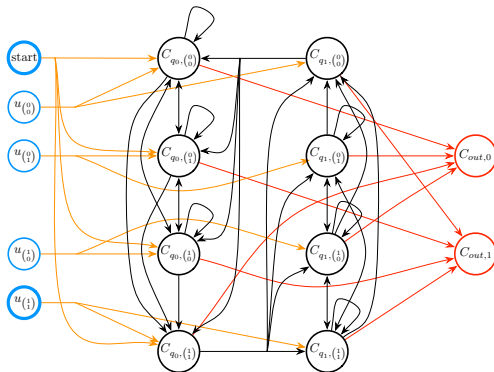
1 1 0 0 1 1 0



## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$

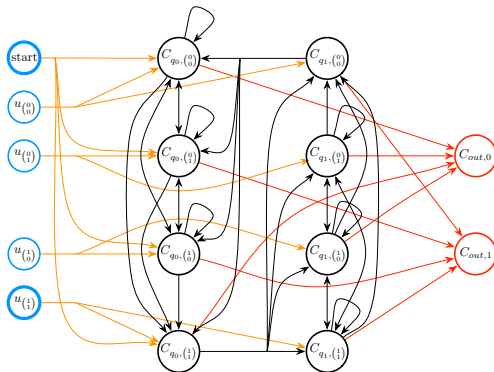




## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

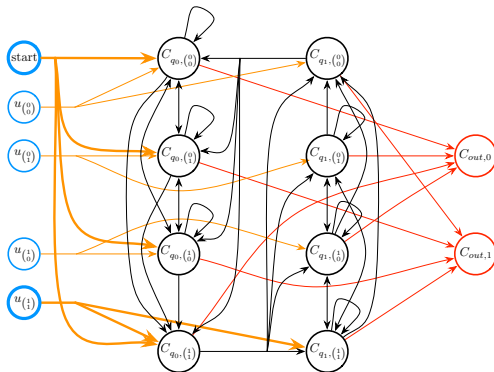
$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

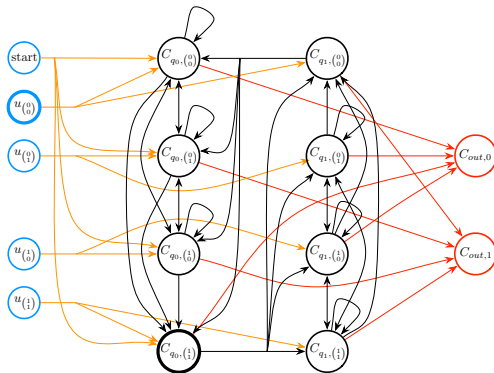
$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$

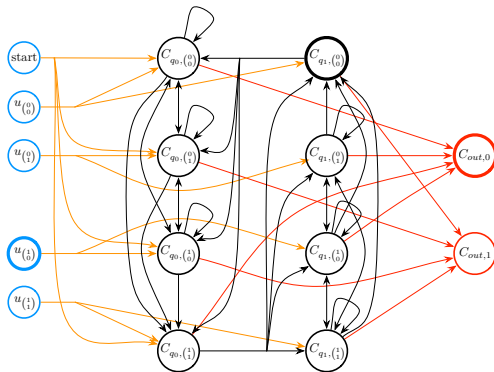




## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

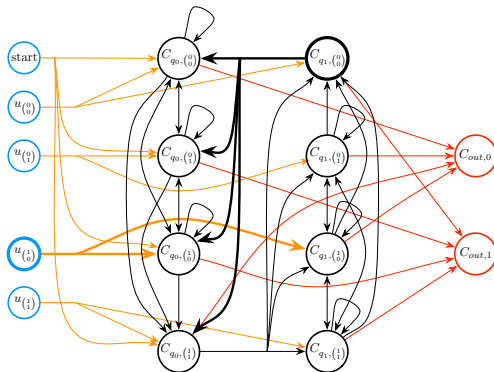
$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

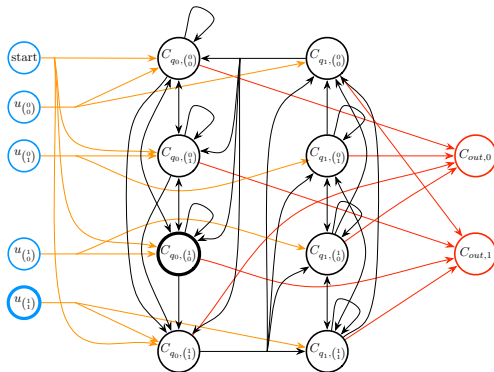
$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

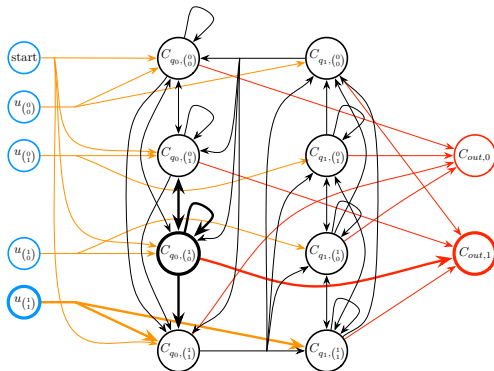
$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$



## PERCEPTRONS ET AUTOMATES

- **Exemple:** implémentation de l'addition en binaire.

$$\begin{array}{ccccccc}
 & 1 & 1^1 & 0^1 & 1 & 1 & 0^1 & 1 \\
 + & & 1 & 1 & 1 & 0 & 0 & 1 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$





# PERCEPTRONS ET AUTOMATES

Cette équivalence entre circuits logiques ou automates finis et réseaux de perceptrons est valable pour tout circuit logique ou tout automate fini.

## THEOREM ([KLEENE, 1956, MINSKY, 1967])

- ▶ *Tout automate fini peut-être simulé par un réseau de neurones composé de perceptrons.*
- ▶ *Tout réseau de neurones composé de perceptrons peut-être simulé par automate fini.*

# PERCEPTRONS ET AUTOMATES

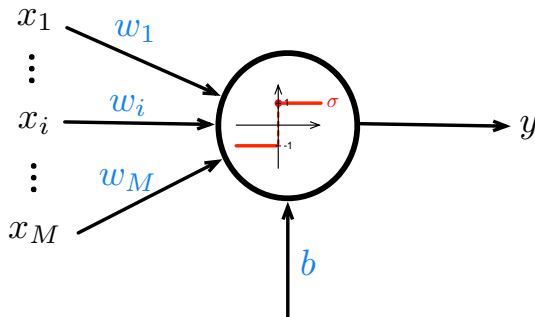
Cette équivalence entre circuits logiques ou automates finis et réseaux de perceptrons est valable pour tout circuit logique ou tout automate fini.

## THEOREM ([KLEENE, 1956, MINSKY, 1967])

- ▶ *Tout automate fini peut-être simulé par un réseau de neurones composé de perceptrons.*
- ▶ *Tout réseau de neurones composé de perceptrons peut-être simulé par automate fini.*

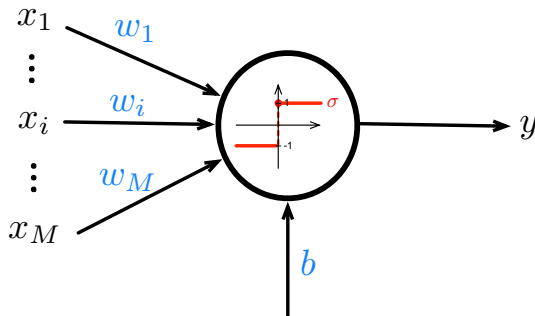
# APPRENTISSAGE

- Le **perceptron** est un *classifieur binaire*.
- Étant donné un input, il répond par oui (+1) ou non (-1).



# APPRENTISSAGE

- ▶ Le **perceptron** est un *classifieur binaire*.
- ▶ Étant donné un input, il répond par oui (+1) ou non (-1).



# APPRENTISSAGE

Soit un dataset formé de 2 types de de data.

- ▶ Des images violentes et d'autres non-violentes;
- ▶ Des mails qui sont des spams et d'autres non;
- ▶ Des produits défectueux et d'autres non.

Apprentissage ou entraînement du perceptron: déterminer les *poids synaptiques* et le *biais* du perceptron de telle sorte que:

- ▶ si on lui donne des inputs de la classe 1, il réponde 1 (et non 0)
- ▶ si on lui donne des inputs de la classe 2, il réponde 0 (et non 1)
- ▶ si on se trompe, il se corrige lui-même.

# APPRENTISSAGE

Soit un dataset formé de 2 types de data.

- ▶ Des images violentes et d'autres non-violentes;
- ▶ Des mails qui sont des spams et d'autres non;
- ▶ Des produits défectueux et d'autres non.

Apprentissage ou entraînement du perceptron: déterminer les *poids synaptiques* et le *biais* du perceptron de telle sorte que:

si on lui donne des inputs de la classe 1, il réponde 1 (et non 0)

et si on lui donne des inputs de la classe 0, il réponde 0 (et non 1)

Il y a une seule solution possible.

# APPRENTISSAGE

Soit un dataset formé de 2 types de de data.

- ▶ Des images violentes et d'autres non-violentes;
- ▶ Des mails qui sont des spams et d'autres non;
- ▶ Des produits défectueux et d'autres non.

Apprentissage ou entraînement du perceptron: déterminer les *poids synaptiques* et le *biais* du perceptron de telle sorte que:

- ▶ Pour une donnée *test* appartenant à la classe 1, il réponde "oui".
- ▶ Pour une donnée *test* appartenant à la classe 2, il réponde "non".
- ▶ Pour une donnée *test* appartenant à la classe 3, il réponde "oui".

# APPRENTISSAGE

Soit un dataset formé de 2 types de de data.

- ▶ Des images violentes et d'autres non-violentes;
- ▶ Des mails qui sont des spams et d'autres non;
- ▶ Des produits défectueux et d'autres non.

Apprentissage ou entraînement du perceptron: déterminer les *poids synaptiques* et le *biais* du perceptron de telle sorte que:

- ✱ si on lui donne des inputs de la classe 1, il réponde +1 et
- ✱ si on lui donne des inputs de la classe 2, il réponde -1



# APPRENTISSAGE

Soit un dataset formé de 2 types de de data.

- ▶ Des images violentes et d'autres non-violentes;
- ▶ Des mails qui sont des spams et d'autres non;
- ▶ Des produits défectueux et d'autres non.

**Apprentissage** ou **entraînement** du perceptron: déterminer les *poids synaptiques* et le *biais* du perceptron de telle sorte que:

- ✱ si on lui donne des inputs de la classe 1, il réponde  $+1$  et
- ✱ si on lui donne des inputs de la classe 2, il réponde  $-1$

# APPRENTISSAGE

Soit un dataset formé de 2 types de de data.

- ▶ Des images violentes et d'autres non-violentes;
- ▶ Des mails qui sont des spams et d'autres non;
- ▶ Des produits défectueux et d'autres non.

**Apprentissage** ou **entraînement** du perceptron: déterminer les *poids synaptiques* et le *biais* du perceptron de telle sorte que:

- ▶ si on lui donne des inputs de la classe 1, il réponde  $+1$  et
- ▶ si on lui donne des inputs de la classe 2, il réponde  $-1$
- ▶ en se trompant le moins possible...

# APPRENTISSAGE

Soit un dataset formé de 2 types de data.

- ▶ Des images violentes et d'autres non-violentes;
- ▶ Des mails qui sont des spams et d'autres non;
- ▶ Des produits défectueux et d'autres non.

**Apprentissage** ou **entraînement** du perceptron: déterminer les *poids synaptiques* et le *biais* du perceptron de telle sorte que:

- ▶ si on lui donne des inputs de la classe 1, il réponde  $+1$  et
- ▶ si on lui donne des inputs de la classe 2, il réponde  $-1$
- ▶ en se trompant le moins possible...

# APPRENTISSAGE

Soit un dataset formé de 2 types de de data.

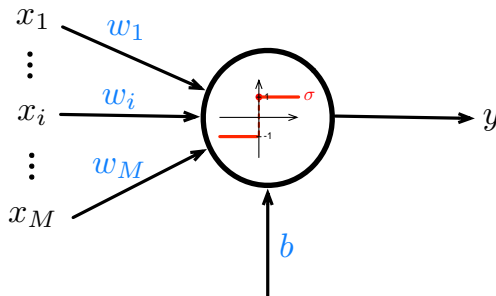
- ▶ Des images violentes et d'autres non-violentes;
- ▶ Des mails qui sont des spams et d'autres non;
- ▶ Des produits défectueux et d'autres non.

**Apprentissage** ou **entraînement** du perceptron: déterminer les *poids synaptiques* et le *biais* du perceptron de telle sorte que:

- ▶ si on lui donne des inputs de la classe 1, il réponde  $+1$  et
- ▶ si on lui donne des inputs de la classe 2, il réponde  $-1$
- ▶ en se trompant le moins possible...

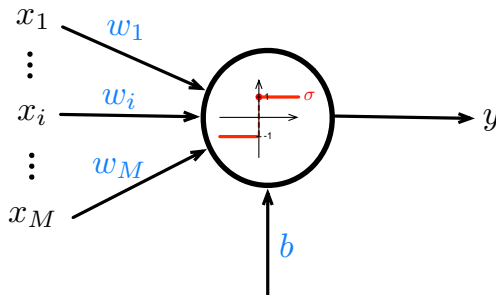
# APPRENTISSAGE

- ▶ Apprentissage: modification des poids synaptiques et du biais.
- ▶ Vaguement inspiré du concept crucial de **plasticité synaptique** en biologie.



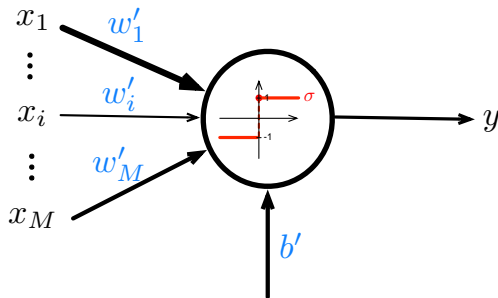
# APPRENTISSAGE

- ▶ Apprentissage: modification des poids synaptiques et du biais.
- ▶ Vaguement inspiré du concept crucial de **plasticité synaptique** en biologie.



# APPRENTISSAGE

- Apprentissage: modification des poids synaptiques et du biais.
- Vaguement inspiré du concept crucial de **plasticité synaptique** en biologie.



# APPRENTISSAGE

---

## Algorithm 1: Training algorithm of the perceptron

---

**Data:**

dataset =  $\{(\mathbf{x}_i, y_i) : i = 1, \dots, N\}$ ;

Perceptron  $P$  with random weights (and bias)  $\mathbf{w}$ ;

Learning rate  $\lambda > 0$ .

```
for  $e = 1$  to  $nb\_epochs$  do
  for  $k = 1$  to  $N$  do
    Process sample  $\mathbf{x}_k$  into perceptron  $P$ 
    if  $\mathbf{x}_k$  is ill-classified then
       $\mathbf{w} := \mathbf{w} + \lambda \cdot y_k \cdot \mathbf{x}_k$            // update weights and bias
    end
  end
end
return trained weights  $\mathbf{w}$ 
```

---

On parcourt l'ensemble des échantillons de notre table d'entraînement. Pour chaque échantillon, on passe les poids à un perceptron.

Si l'échantillon n'est pas bien classé, on met à jour les poids.



# APPRENTISSAGE

---

## Algorithm 1: Training algorithm of the perceptron

---

**Data:** $\text{dataset} = \{(\mathbf{x}_i, y_i) : i = 1, \dots, N\};$ Perceptron  $P$  with random weights (and bias)  $\mathbf{w}$ ;Learning rate  $\lambda > 0$ .**for**  $e = 1$  **to**  $nb\_epochs$  **do**    **for**  $k = 1$  **to**  $N$  **do**        Process sample  $\mathbf{x}_k$  into perceptron  $P$         **if**  $\mathbf{x}_k$  is ill-classified **then**             $\mathbf{w} := \mathbf{w} + \lambda \cdot y_k \cdot \mathbf{x}_k$  // update weights and bias        **end****end****return** trained weights  $\mathbf{w}$ 

---

On peut imaginer qu'on ajoute à notre vecteur de poids initial  $\mathbf{w}$  un vecteur  $\mathbf{x}_k$  pondéré par le poids  $\lambda \cdot y_k$ . On peut aussi imaginer qu'on ajoute à notre vecteur de poids initial  $\mathbf{w}$  un vecteur  $\mathbf{x}_k$  pondéré par le poids  $\lambda$ .

On peut aussi imaginer qu'on ajoute à notre vecteur de poids initial  $\mathbf{w}$  un vecteur  $\mathbf{x}_k$  pondéré par le poids  $\lambda$ .

# APPRENTISSAGE

---

## Algorithm 1: Training algorithm of the perceptron

---

**Data:** $\text{dataset} = \{(\mathbf{x}_i, y_i) : i = 1, \dots, N\};$ Perceptron  $P$  with random weights (and bias)  $\mathbf{w}$ ;Learning rate  $\lambda > 0$ .**for**  $e = 1$  *to*  $nb\_epochs$  **do**    **for**  $k = 1$  *to*  $N$  **do**        Process sample  $\mathbf{x}_k$  into perceptron  $P$         **if**  $\mathbf{x}_k$  *is ill-classified* **then**             $\mathbf{w} := \mathbf{w} + \lambda \cdot y_k \cdot \mathbf{x}_k$  // update weights and bias        **end****end****return** trained weights  $\mathbf{w}$ 

---

On peut imaginer qu'on ajoute à notre vecteur de poids une "biase"  $b$ . On a

alors  $\mathbf{w} = [w_1, w_2, \dots, w_n, b]$ . Mais pour des raisons techniques, on va préférer écrire  $b$  à part.

On va donc écrire  $\mathbf{w} = [w_1, w_2, \dots, w_n]$  et  $b$  à part.

On va donc écrire  $\mathbf{w} = [w_1, w_2, \dots, w_n]$  et  $b$  à part.

# APPRENTISSAGE

---

## Algorithm 1: Training algorithm of the perceptron

---

**Data:**

dataset =  $\{(\mathbf{x}_i, y_i) : i = 1, \dots, N\}$ ;

Perceptron  $P$  with random weights (and bias)  $\mathbf{w}$ ;

Learning rate  $\lambda > 0$ .

**for**  $e = 1$  *to*  $nb\_epochs$  **do**

**for**  $k = 1$  *to*  $N$  **do**

        Process sample  $\mathbf{x}_k$  into perceptron  $P$

**if**  $\mathbf{x}_k$  *is ill-classified* **then**

$\mathbf{w} := \mathbf{w} + \lambda \cdot y_k \cdot \mathbf{x}_k$

            // update weights and bias

**end**

**end**

**return** trained weights  $\mathbf{w}$

---

- Remarque:  $\mathbf{w}$  et  $\mathbf{x}_k$  sont deux vecteurs de même taille. Ainsi, la mise à jour des poids est une somme vectorielle:

$$\mathbf{w} := \mathbf{w} + \lambda \cdot y_k \cdot \mathbf{x}_k$$



# APPRENTISSAGE

---

**Algorithm 1:** Training algorithm of the perceptron

---

**Data:**dataset =  $\{(\mathbf{x}_i, y_i) : i = 1, \dots, N\}$ ;Perceptron  $P$  with random weights (and bias)  $\mathbf{w}$ ;Learning rate  $\lambda > 0$ .**for**  $e = 1$  *to*  $nb\_epochs$  **do**    **for**  $k = 1$  *to*  $N$  **do**        Process sample  $\mathbf{x}_k$  into perceptron  $P$         **if**  $\mathbf{x}_k$  *is ill-classified* **then**             $\mathbf{w} := \mathbf{w} + \lambda \cdot y_k \cdot \mathbf{x}_k$  // update weights and bias        **end****end****return** trained weights  $\mathbf{w}$ 

---

- **Remarque:**  $\mathbf{w}$  et  $\mathbf{x}_k$  sont deux vecteurs de même taille. Ainsi, la mise à jour des poids est une somme vectorielle:

$$\mathbf{w} := \mathbf{w} + \lambda \cdot y_k \cdot \mathbf{x}_k$$

## EXEMPLE

Démo notebook: entraînement du perceptron sur le dataset MNIST.



# RÉSEAUX DE NEURONES DE 2<sup>ÈME</sup> GÉNÉRATION

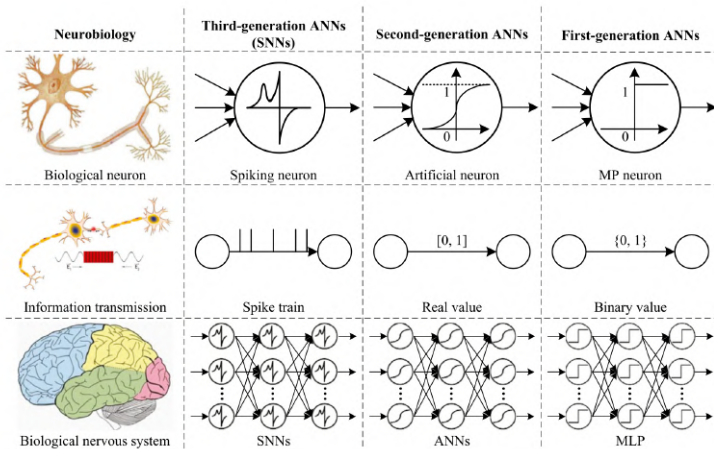
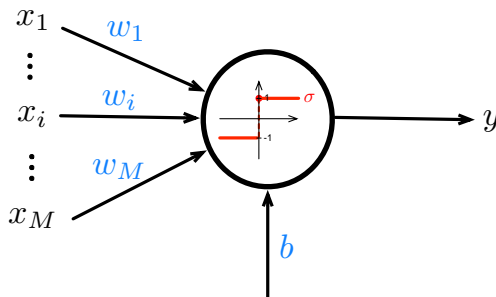


Figure taken from [Wang et al., 2020]

# NEURONES CONTINUS

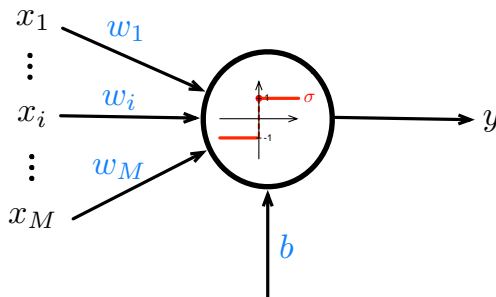
- ▶ Le perceptron est un simple **neurone discret**: la fonction d'activation est discrète.
- ▶ En deep learning, on utilise des **neurones continus**: la fonction d'activation est continue (tanh, ReLU, etc.).





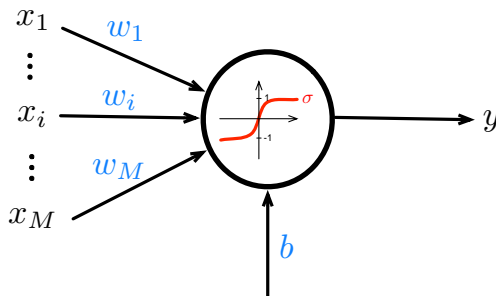
# NEURONES CONTINUS

- ▶ Le perceptron est un simple **neurone discret**: la fonction d'activation est discrète.
- ▶ En deep learning, on utilise des **neurones continus**: la fonction d'activation est continue (tanh, ReLU, etc.).



# NEURONES CONTINUS

- ▶ Le perceptron est un simple **neurone discret**: la fonction d'activation est discrète.
- ▶ En deep learning, on utilise des **neurones continus**: la fonction d'activation est continue (tanh, ReLU, etc.).



# NEURONES CONTINUS

## Fonctions d'activations

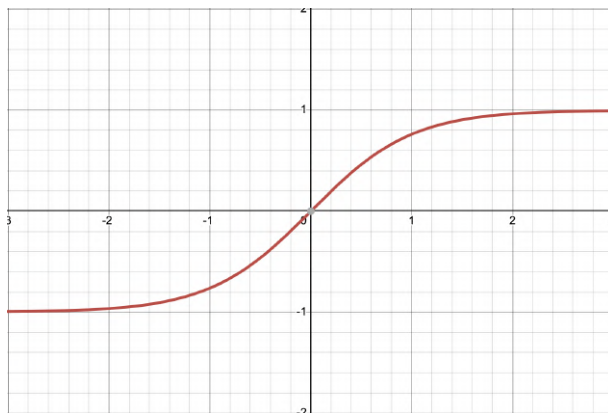
- discrète, tanh et ReLU(rectified linear unit)



# NEURONES CONTINUS

## Fonctions d'activations

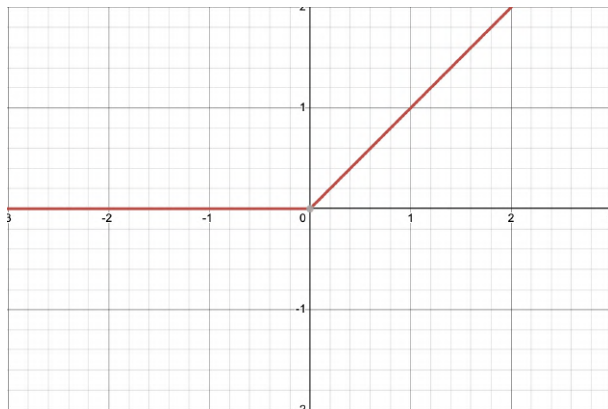
- ▶ discrète, tanh et ReLU(rectified linear unit)



# NEURONES CONTINUS

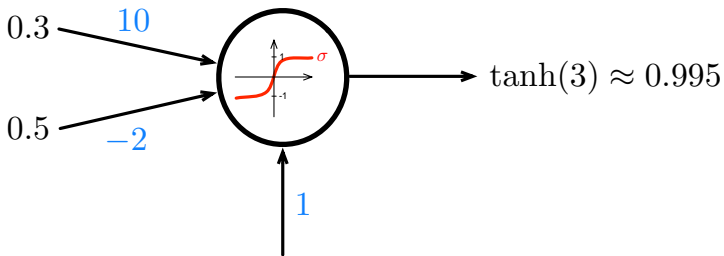
## Fonctions d'activations

- ▶ discrète, tanh et ReLU(rectified linear unit)



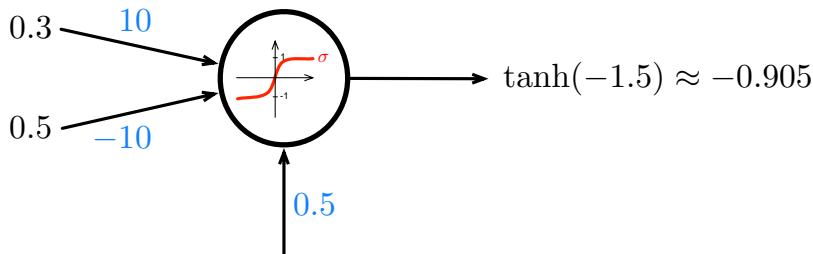
# NEURONES CONTINUS

Exemples: fonction d'activation tanh



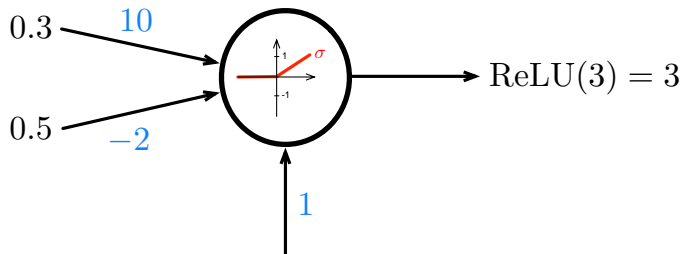
# NEURONES CONTINUS

Exemples: fonction d'activation tanh



# NEURONES CONTINUS

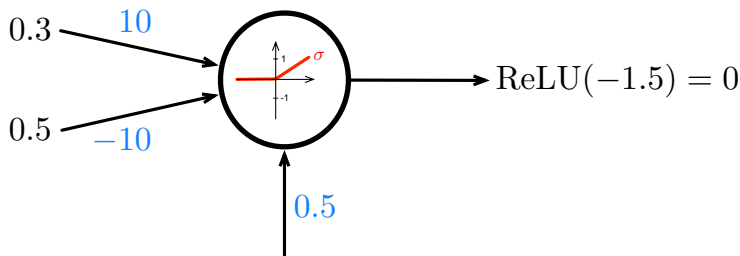
Exemples: fonction d'activation ReLU





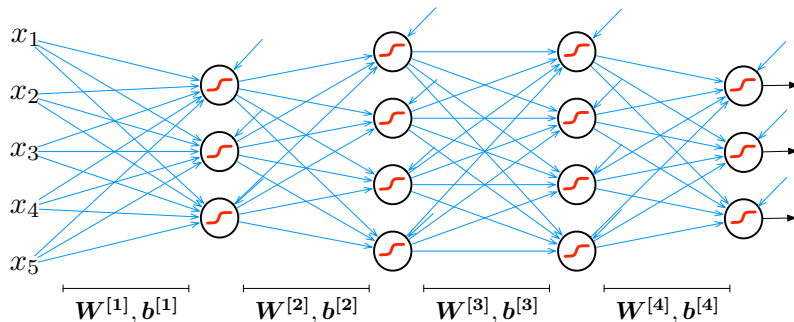
# NEURONES CONTINUS

**Exemples:** fonction d'activation ReLU



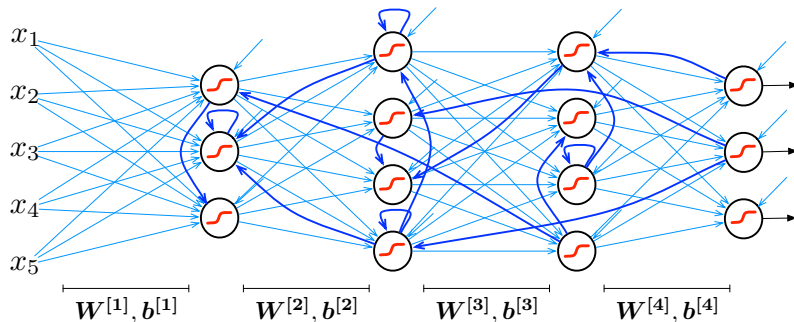
# RÉSEAU DE NEURONES

## Réseau de neurones feedforward



# RÉSEAU DE NEURONES

## Réseau de neurones récurrents



# RÉSEAU DE NEURONES

- ▶ Les réseaux de neurones continus sont beaucoup plus puissants que les réseaux de perrceptron.
- ▶ Les réseaux de neurones *récurrents* sont computationnellement équivalents à des machines de Turing.
- ▶ Ils peuvent donc simuler d'importe quel algorithme (thèse de Church-Turing).
- ▶ Et ils peuvent donc apprendre des problèmes beaucoup plus complexes!

# RÉSEAU DE NEURONES

- ▶ Les réseaux de neurones continus sont beaucoup plus puissants que les réseaux de perceptron.
- ▶ Les réseaux de neurones *récurrents* sont computationnellement équivalents à des **machines de Turing**.
- ▶ Ils peuvent donc simuler d'importe quel algorithme (thèse de Church-Turing).
- ▶ Et ils peuvent donc apprendre des problèmes beaucoup plus complexes!

# RÉSEAU DE NEURONES

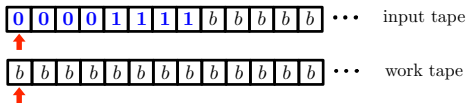
- ▶ Les réseaux de neurones continus sont beaucoup plus puissants que les réseaux de perceptron.
- ▶ Les réseaux de neurones *récurrents* sont computationnellement équivalents à des **machines de Turing**.
- ▶ Ils peuvent donc simuler d'importe quel algorithme (thèse de Church-Turing).
- ▶ Et ils peuvent donc apprendre des problèmes beaucoup plus complexes!

# RÉSEAU DE NEURONES

- ▶ Les réseaux de neurones continus sont beaucoup plus puissants que les réseaux de perceptron.
- ▶ Les réseaux de neurones *récurrents* sont computationnellement équivalents à des **machines de Turing**.
- ▶ Ils peuvent donc simuler d'importe quel algorithme (thèse de Church-Turing).
- ▶ Et ils peuvent donc apprendre des problèmes beaucoup plus complexes!

# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$



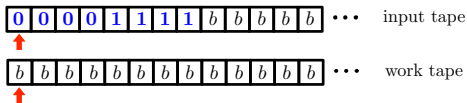
Program *current state:  $q_{in}$*

$(q_{in}, b, b) \mapsto (q_{acc}, b, b, S, S)$	
$(q_{in}, 0, b) \mapsto (q_0, 0, b, R, S)$	
$(q_{in}, 1, b) \mapsto (q_{rej}, 1, b, S, S)$	
$(q_0, 0, b) \mapsto (q_{0bis}, 0, 0, R, R)$	$(q_{bis}, 0, b) \mapsto (q_0, 0, 0, R, R)$
$(q_0, 1, b) \mapsto (q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b) \mapsto (q_1, 1, 1, R, L)$
$(q_0, b, b) \mapsto (q_{rej}, b, b, S, S)$	$(q_{0bis}, b, b) \mapsto (q_{rej}, b, b, S, S)$
$(q_1, 1, 0) \mapsto (q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0) \mapsto (q_1, 1, 1, R, L)$
$(q_1, b, 1) \mapsto (q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1) \mapsto (q_{acc}, b, 1, S, S)$
$(q_1, b, 0) \mapsto (q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0) \mapsto (q_{rej}, b, 0, S, S)$
$(q_1, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$



# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$

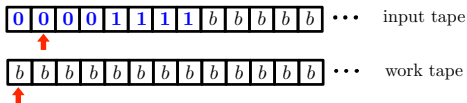


Program    *current state:  $q_{in}$*

$(q_{in}, b, b)$	$\mapsto$	$(q_{acc}, b, b, S, S)$			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_0, b, b)$	$\mapsto$	$(q_{rej}, b, b, S, S)$	$(q_{0bis}, b, b)$	$\mapsto$	$(q_{rej}, b, b, S, S)$
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$

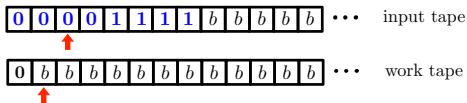


Program    current state:  $q_0$

$(q_{in}, b, b) \mapsto (q_{acc}, b, b, S, S)$	
$(q_{in}, 0, b) \mapsto (q_0, 0, b, R, S)$	
$(q_{in}, 1, b) \mapsto (q_{rej}, 1, b, S, S)$	
$(q_0, 0, b) \mapsto (q_{0bis}, 0, 0, R, R)$	$(q_{bis}, 0, b) \mapsto (q_0, 0, 0, R, R)$
$(q_0, 1, b) \mapsto (q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b) \mapsto (q_1, 1, 1, R, L)$
$(q_0, b, b) \mapsto (q_{rej}, b, b, S, S)$	$(q_{0bis}, b, b) \mapsto (q_{rej}, b, b, S, S)$
$(q_1, 1, 0) \mapsto (q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0) \mapsto (q_1, 1, 1, R, L)$
$(q_1, b, 1) \mapsto (q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1) \mapsto (q_{acc}, b, 1, S, S)$
$(q_1, b, 0) \mapsto (q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0) \mapsto (q_{rej}, b, 0, S, S)$
$(q_1, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$

# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$

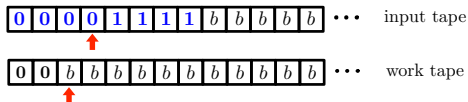


Program    current state: *q0bis*

$(q_{in}, b, b) \mapsto (q_{acc}, b, b, S, S)$	
$(q_{in}, 0, b) \mapsto (q_0, 0, b, R, S)$	
$(q_{in}, 1, b) \mapsto (q_{rej}, 1, b, S, S)$	
$(q_0, 0, b) \mapsto (q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b) \mapsto (q_0, 0, 0, R, R)$
$(q_0, 1, b) \mapsto (q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b) \mapsto (q_1, 1, 1, R, L)$
$(q_0, b, b) \mapsto (q_{rej}, b, b, S, S)$	$(q_{0bis}, b, b) \mapsto (q_{rej}, b, b, S, S)$
$(q_1, 1, 0) \mapsto (q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0) \mapsto (q_1, 1, 1, R, L)$
$(q_1, b, 1) \mapsto (q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1) \mapsto (q_{acc}, b, 1, S, S)$
$(q_1, b, 0) \mapsto (q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0) \mapsto (q_{rej}, b, 0, S, S)$
$(q_1, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$

# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$

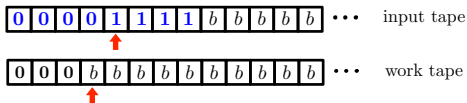


Program    current state:  $q_0$

$(q_{in}, b, b) \mapsto (q_{acc}, b, b, S, S)$	
$(q_{in}, 0, b) \mapsto (q_0, 0, b, R, S)$	
$(q_{in}, 1, b) \mapsto (q_{rej}, 1, b, S, S)$	
$(q_0, 0, b) \mapsto (q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b) \mapsto (q_0, 0, 0, R, R)$
$(q_0, 1, b) \mapsto (q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b) \mapsto (q_1, 1, 1, R, L)$
$(q_0, b, b) \mapsto (q_{rej}, b, b, S, S)$	$(q_{0bis}, b, b) \mapsto (q_{rej}, b, b, S, S)$
$(q_1, 1, 0) \mapsto (q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0) \mapsto (q_1, 1, 1, R, L)$
$(q_1, b, 1) \mapsto (q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1) \mapsto (q_{acc}, b, 1, S, S)$
$(q_1, b, 0) \mapsto (q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0) \mapsto (q_{rej}, b, 0, S, S)$
$(q_1, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$

# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$

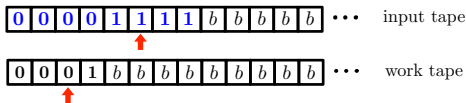


Program    current state: *q<sub>0bis</sub>*

$(q_{in}, b, b) \mapsto (q_{acc}, b, b, S, S)$	
$(q_{in}, 0, b) \mapsto (q_0, 0, b, R, S)$	
$(q_{in}, 1, b) \mapsto (q_{rej}, 1, b, S, S)$	
$(q_0, 0, b) \mapsto (q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b) \mapsto (q_0, 0, 0, R, R)$
$(q_0, 1, b) \mapsto (q_1, 1, 1, R, L)$	<i><math>(q_{0bis}, 1, b) \mapsto (q_1, 1, 1, R, L)</math></i>
$(q_0, b, b) \mapsto (q_{rej}, b, b, S, S)$	$(q_{0bis}, b, b) \mapsto (q_{rej}, b, b, S, S)$
$(q_1, 1, 0) \mapsto (q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0) \mapsto (q_1, 1, 1, R, L)$
$(q_1, b, 1) \mapsto (q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1) \mapsto (q_{acc}, b, 1, S, S)$
$(q_1, b, 0) \mapsto (q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0) \mapsto (q_{rej}, b, 0, S, S)$
$(q_1, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$

# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$

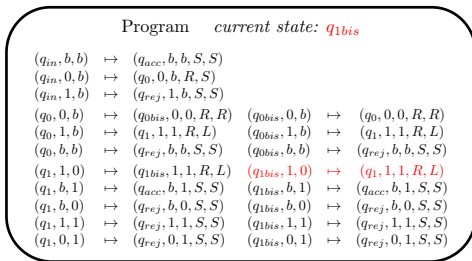
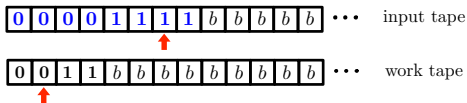


Program current state:  $q_1$

$(q_{in}, b, b) \mapsto (q_{acc}, b, b, S, S)$	
$(q_{in}, 0, b) \mapsto (q_0, 0, b, R, S)$	
$(q_{in}, 1, b) \mapsto (q_{rej}, 1, b, S, S)$	
$(q_0, 0, b) \mapsto (q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b) \mapsto (q_0, 0, 0, R, R)$
$(q_0, 1, b) \mapsto (q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b) \mapsto (q_1, 1, 1, R, L)$
$(q_0, b, b) \mapsto (q_{rej}, b, b, S, S)$	$(q_{0bis}, b, b) \mapsto (q_{rej}, b, b, S, S)$
$(q_1, 1, 0) \mapsto (q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0) \mapsto (q_1, 1, 1, R, L)$
$(q_1, b, 1) \mapsto (q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1) \mapsto (q_{acc}, b, 1, S, S)$
$(q_1, b, 0) \mapsto (q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0) \mapsto (q_{rej}, b, 0, S, S)$
$(q_1, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$

# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$



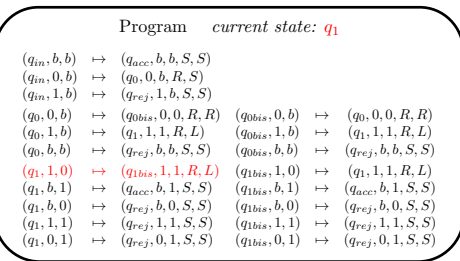
# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$

0 0 0 0 1 1 1 1 b b b b b ... input tape



0 1 1 1 b b b b b b b b b ... work tape





# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$

0 0 0 0 1 1 1 1 b b b b b ... input tape

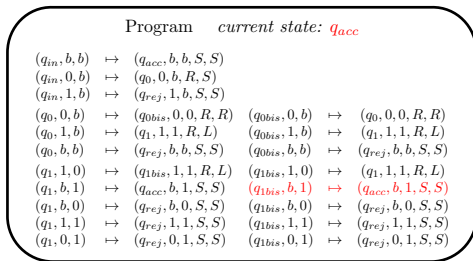
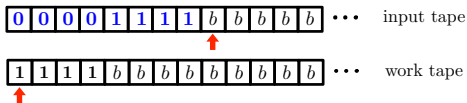


1 1 1 1 b b b b b b b b b ... work tape

Program		current state: <i>q<sub>1bis</sub></i>	
$(q_{in}, b, b) \mapsto (q_{acc}, b, b, S, S)$			
$(q_{in}, 0, b) \mapsto (q_0, 0, b, R, S)$			
$(q_{in}, 1, b) \mapsto (q_{rej}, 1, b, S, S)$			
$(q_0, 0, b) \mapsto (q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b) \mapsto (q_0, 0, 0, R, R)$		
$(q_0, 1, b) \mapsto (q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b) \mapsto (q_1, 1, 1, R, L)$		
$(q_0, b, b) \mapsto (q_{rej}, b, b, S, S)$	$(q_{0bis}, b, b) \mapsto (q_{rej}, b, b, S, S)$		
$(q_1, 1, 0) \mapsto (q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0) \mapsto (q_1, 1, 1, R, L)$		
$(q_1, b, 1) \mapsto (q_{acc}, b, 1, S, S)$	<i><math>(q_{1bis}, b, 1) \mapsto (q_{acc}, b, 1, S, S)</math></i>		
$(q_1, b, 0) \mapsto (q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0) \mapsto (q_{rej}, b, 0, S, S)$		
$(q_1, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1) \mapsto (q_{rej}, 1, 1, S, S)$		
$(q_1, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1) \mapsto (q_{rej}, 0, 1, S, S)$		

# MACHINE DE TURING

- Turing machine qui reconnaît le langage  $\{0^n 1^n : n \geq 0\}$



# RÉSEAUX DE NEURONES ET MACHINE DE TURING

Universalité computationnelle des réseaux de neurones récurrents  
(avec neurones sigmoïdaux continus).

THEOREM ([SIEGELMANN AND SONTAG,  
1994, SIEGELMANN AND SONTAG, 1995])

- ▶ *Si on suppose des valeurs d'activation rationnelles, toute machine de Turing peut être simulée par un réseau de neurones récurrent.*
- ▶ *Si on suppose des valeurs d'activation réelles, les réseaux de neurones récurrents sont super-Turing.*

# RÉSEAUX DE NEURONES ET MACHINE DE TURING

Universalité computationnelle des réseaux de neurones récurrents (avec neurones sigmoïdaux continus).

THEOREM ([SIEGELMANN AND SONTAG, 1994, SIEGELMANN AND SONTAG, 1995])

- ▶ *Si on suppose des valeurs d'activation rationnelles, toute machine de Turing peut être simulée par un réseau de neurones récurrent.*
- ▶ *Si on suppose des valeurs d'activation réelles, les réseaux de neurones récurrents sont super-Turing.*

# RÉSEAUX DE NEURONES

## Movie 2

Video taken from the “3 Blue 1 Braun” YouTube channel

# APPRENTISSAGE: CLASSIFICATION

Soit un dataset  $S = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, N\}$ .

- **Classification:** les  $\mathbf{y}_i$  sont des (1-hot encodages de) classes discrètes  $1, \dots, C$ .
- Exemples: classification d'images, de documents, de produits, détection de maladies, etc.

Apprentissage ou entraînement du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

pour un donné  $\mathbf{x}$ , nous ayons un impact  $\sigma$  de la classe  $c$ , alors que  $\sigma$  est le plus petit possible pour les autres classes.

→ *Optimisation des paramètres du réseau de neurones*

→ *Algorithme de descente de gradient (backpropagation)*

→ *Algorithme de descente de gradient stochastique*

# APPRENTISSAGE: CLASSIFICATION

Soit un dataset  $S = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, N\}$ .

- **Classification:** les  $\mathbf{y}_i$  sont des (1-hot encodages de) classes discrètes  $1, \dots, C$ .
- Exemples: classification d'images, de documents, de produits, détection de maladies, etc.

Apprentissage ou entraînement du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

pour tout  $(\mathbf{x}_i, \mathbf{y}_i) \in S$ , le réseau de neurones associe à  $\mathbf{x}_i$  le vecteur  $\mathbf{y}_i$ .

# APPRENTISSAGE: CLASSIFICATION

Soit un dataset  $S = \{(x_i, y_i) : i = 1, \dots, N\}$ .

- **Classification:** les  $y_i$  sont des (1-hot encodages de) classes discrètes  $1, \dots, C$ .
- Exemples: classification d'images, de documents, de produits, détection de maladies, etc.

Apprentissage ou entraînement du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- ✱ si on donne au réseau un input  $x$  de la classe  $c$ , alors c'est le  $c$ -ème neurone d'output qui devra avoir la plus forte activation, en se trompant le moins possible.

➤ On va donc apprendre à partir d'un dataset  $S$  à classer des exemples  $(x_i, y_i)$ .

➤ On va donc apprendre à partir d'un dataset  $S$  à classer des exemples  $(x_i, y_i)$ .



# APPRENTISSAGE: CLASSIFICATION

Soit un dataset  $S = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, N\}$ .

- **Classification:** les  $\mathbf{y}_i$  sont des (1-hot encodages de) classes discrètes  $1, \dots, C$ .
- Exemples: classification d'images, de documents, de produits, détection de maladies, etc.

**Apprentissage** ou **entraînement** du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- ✱ si on donne au réseau un input  $\mathbf{x}$  de la classe  $c$ , alors c'est le  $c$ -ème neurone d'output qui devra avoir la plus forte activation, en se trompant le moins possible.
- ✱ Minimiser l'entropie croisée (categorical cross entropy, CCE) entre classes réelles et prédictions.

# APPRENTISSAGE: CLASSIFICATION

Soit un dataset  $S = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, N\}$ .

- ▶ **Classification:** les  $\mathbf{y}_i$  sont des (1-hot encodages de) classes discrètes  $1, \dots, C$ .
- ▶ Exemples: classification d'images, de documents, de produits, détection de maladies, etc.

**Apprentissage** ou **entraînement** du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- ▶ si on donne au réseau un input  $\mathbf{x}$  de la classe  $c$ , alors c'est le  $c$ -ème neurone d'output qui devra avoir la plus forte activation, en se trompant le moins possible.
- ▶ Minimiser l'entropie croisée (categorical cross entropy, CCE) entre classes réelles et prédictions.

# APPRENTISSAGE: CLASSIFICATION

Soit un dataset  $S = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, N\}$ .

- **Classification:** les  $\mathbf{y}_i$  sont des (1-hot encodages de) classes discrètes  $1, \dots, C$ .
- Exemples: classification d'images, de documents, de produits, détection de maladies, etc.

**Apprentissage** ou **entraînement** du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- si on donne au réseau un input  $\mathbf{x}$  de la classe  $c$ , alors c'est le  $c$ -ème neurone d'output qui devra avoir la plus forte activation, en se trompant le moins possible.
- Minimiser l'**entropie croisée** (categorical cross entropy, CCE) entre classes réelles et prédictions.

# APPRENTISSAGE: RÉGRESSION

Soit un dataset  $S = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, N\}$ .

- ▶ **Régression:** les  $\mathbf{y}_i$  sont des valeurs/vecteurs continus.
- ▶ Exemples: prédiction de prix, consommation, température, pollution, etc.

Apprentissage ou entraînement du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

lorsqu'on passe au réseau un *support vector*  $\mathbf{x}$ , la valeur  $\mathbf{y}$  obtenue à la sortie (l'output) du réseau donne un *vecteur*  $\mathbf{y}$  dont l'écart avec le *vecteur cible*  $\mathbf{y}_i$  est le plus petit.

On minimise l'erreur quadratique moyenne (mean squared error, MSE) entre le vecteur obtenu  $\mathbf{y}$  et le vecteur cible  $\mathbf{y}_i$ .

# APPRENTISSAGE: RÉGRESSION

Soit un dataset  $S = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, N\}$ .

- ▶ **Régression:** les  $\mathbf{y}_i$  sont des valeurs/vecteurs continus.
- ▶ Exemples: prédiction de prix, consommation, température, pollution, etc.

Apprentissage ou entraînement du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- ▶ Le réseau de neurones soit capable de généraliser à des données jamais vues.
- ▶ Le réseau de neurones soit capable de reconnaître des motifs complexes.
- ▶ Le réseau de neurones soit capable de reconnaître des motifs récurrents.
- ▶ Le réseau de neurones soit capable de reconnaître des motifs spatiaux.
- ▶ Le réseau de neurones soit capable de reconnaître des motifs temporels.

# APPRENTISSAGE: RÉGRESSION

Soit un dataset  $S = \{(x_i, y_i) : i = 1, \dots, N\}$ .

- **Régression:** les  $y_i$  sont des valeurs/vecteurs continus.
- Exemples: prédiction de prix, consommation, température, pollution, etc.

Apprentissage ou entraînement du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- si on donne au réseau un input  $x$  associé à la valeur  $y$ , alors la couche d'output du réseau donne un vecteur  $\hat{y}$  devra être le plus proche possible de  $y$ .

- minimiser l'erreur quadratique (mean squared error)  $\frac{1}{2} \|y - \hat{y}\|^2$  (la distance au carré entre  $y$  et  $\hat{y}$ )

# APPRENTISSAGE: RÉGRESSION

Soit un dataset  $S = \{(x_i, y_i) : i = 1, \dots, N\}$ .

- ▶ **Régression:** les  $y_i$  sont des valeurs/vecteurs continus.
- ▶ Exemples: prédiction de prix, consommation, température, pollution, etc.

**Apprentissage ou entraînement** du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- ▶ si on donne au réseau un input  $x$  associé à la valeur  $y$ , alors la couche d'output du réseau donne un vecteur  $\hat{y}$  devra être le plus proche possible de  $y$ .
- ▶ Minimiser l'erreur quadratique moyenne (mean squared error, MSE) entre valeurs réelles et prédictions.

# APPRENTISSAGE: RÉGRESSION

Soit un dataset  $S = \{(x_i, y_i) : i = 1, \dots, N\}$ .

- ▶ **Régression:** les  $y_i$  sont des valeurs/vecteurs continus.
- ▶ Exemples: prédiction de prix, consommation, température, pollution, etc.

**Apprentissage** ou **entraînement** du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- ▶ si on donne au réseau un input  $x$  associé à la valeur  $y$ , alors la couche d'output du réseau donne un vecteur  $\hat{y}$  devra être le plus proche possible de  $y$ .
- ▶ Minimiser l'erreur quadratique moyenne (mean squared error, MSE) entre valeurs réelles et prédictions.



# APPRENTISSAGE: RÉGRESSION

Soit un dataset  $S = \{(x_i, y_i) : i = 1, \dots, N\}$ .

- ▶ **Régression:** les  $y_i$  sont des valeurs/vecteurs continus.
- ▶ Exemples: prédiction de prix, consommation, température, pollution, etc.

**Apprentissage** ou **entraînement** du réseau de neurones: déterminer les *poids synaptiques* et les *biais* du réseau de telle sorte que:

- ▶ si on donne au réseau un input  $x$  associé à la valeur  $y$ , alors la couche d'output du réseau donne un vecteur  $\hat{y}$  devra être le plus proche possible de  $y$ .
- ▶ Minimiser l'**erreur quadratique moyenne (mean squared error, MSE)** entre valeurs réelles et prédictions.

# APPRENTISSAGE

Features

X

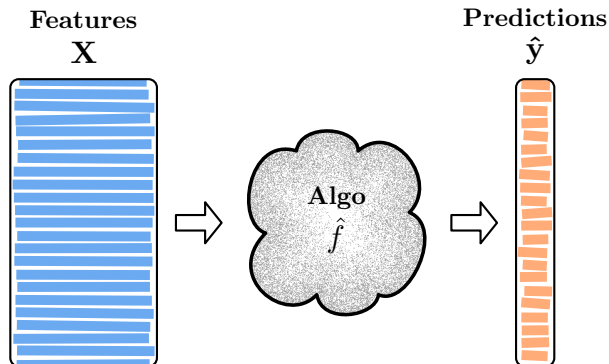


# APPRENTISSAGE

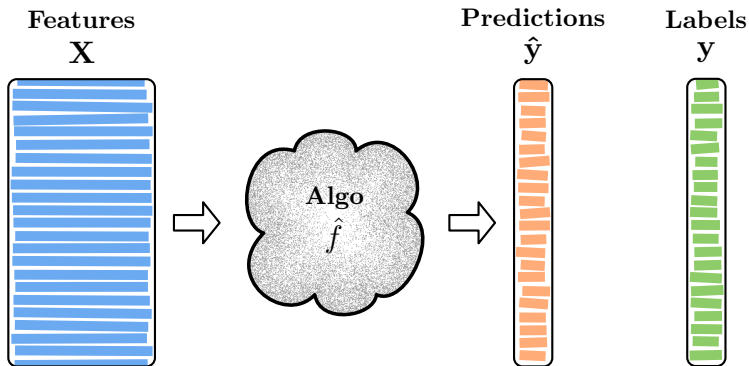
Features  
**X**



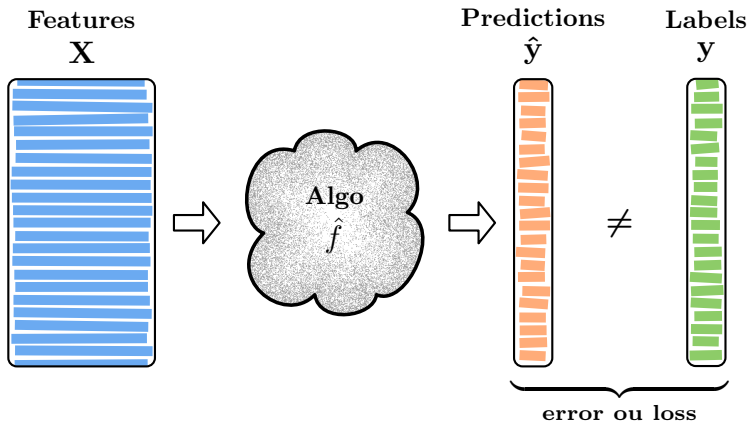
# APPRENTISSAGE



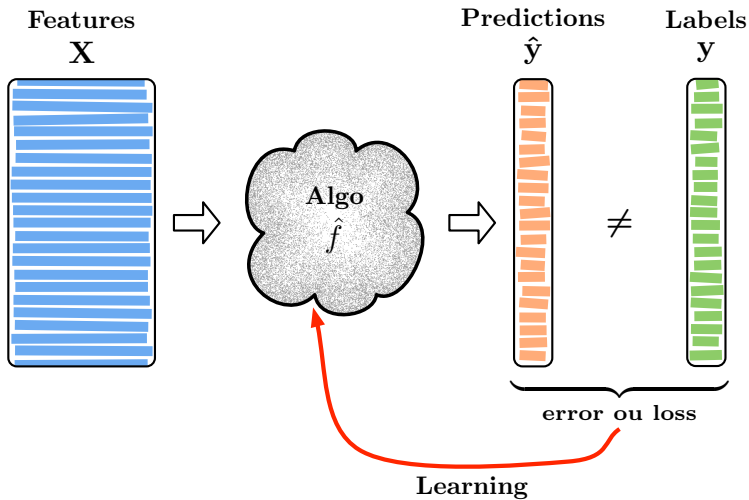
# APPRENTISSAGE



# APPRENTISSAGE



# APPRENTISSAGE



# FONCTION DE COÛT

Movie 3

Movie 4

Videos taken from the “3 Blue 1 Braun” YouTube channel



# DESCENTE DE GRADIENT

- ▶ Si on varie les paramètres (poids et biais) du réseau, on obtient des erreurs (loss ou cost) plus ou moins grandes.
- ▶ Minimisation de la fonction d'erreur: trouver les paramètres qui donnent une erreur minimale.
- ▶ L'inverse du gradient de la fonction d'erreur donne la direction de la plus grande pente descendante.
- ▶ On avance pas à pas dans la direction inverse du gradient, vers un minimum local/global.

# DESCENTE DE GRADIENT

- ▶ Si on varie les paramètres (poids et biais) du réseau, on obtient des erreurs (loss ou cost) plus ou moins grandes.
- ▶ **Minimisation de la fonction d'erreur:** trouver les paramètres qui donnent une erreur minimale.
- ▶ L'inverse du gradient de la fonction d'erreur donne la direction de la plus grande pente descendante.
- ▶ On avance pas à pas dans la direction inverse du gradient, vers un minimum local/global.

# DESCENTE DE GRADIENT

- ▶ Si on varie les paramètres (poids et biais) du réseau, on obtient des erreurs (loss ou cost) plus ou moins grandes.
- ▶ **Minimisation de la fonction d'erreur:** trouver les paramètres qui donnent une erreur minimale.
- ▶ L'inverse du **gradient de la fonction d'erreur** donne la **direction de la plus grande pente descendante**.
- ▶ On avance pas à pas dans la direction inverse du gradient, vers un minimum local/global.

# DESCENTE DE GRADIENT

- ▶ Si on varie les paramètres (poids et biais) du réseau, on obtient des erreurs (loss ou cost) plus ou moins grandes.
- ▶ **Minimisation de la fonction d'erreur:** trouver les paramètres qui donnent une erreur minimale.
- ▶ L'inverse du **gradient de la fonction d'erreur** donne la **direction de la plus grande pente descendante**.
- ▶ On avance pas à pas dans la direction inverse du gradient, **vers un minimum local/global**.

# DESCENTE DE GRADIENT

$$\ell(\dots; \Theta) \text{ or } \mathcal{L}(\dots; \Theta)$$

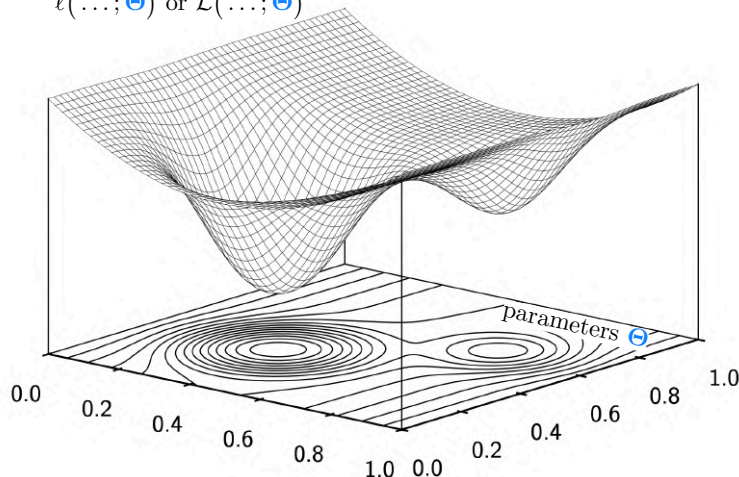


Figure adapted from [Fleuret, 2022]

# DESCENTE DE GRADIENT

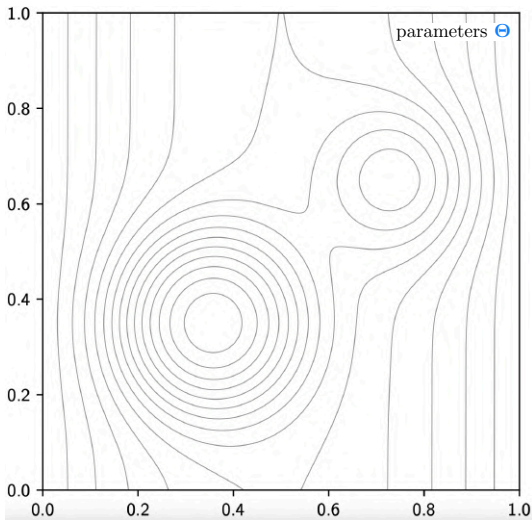


Figure adapted from [Fleuret, 2022]

# DESCENTE DE GRADIENT

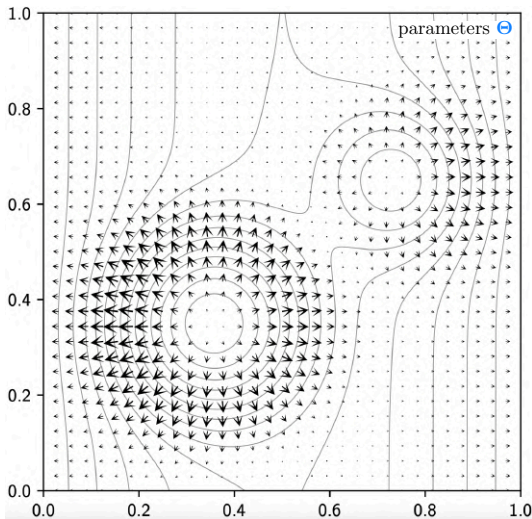


Figure adapted from [Fleuret, 2022]

# DESCENTE DE GRADIENT

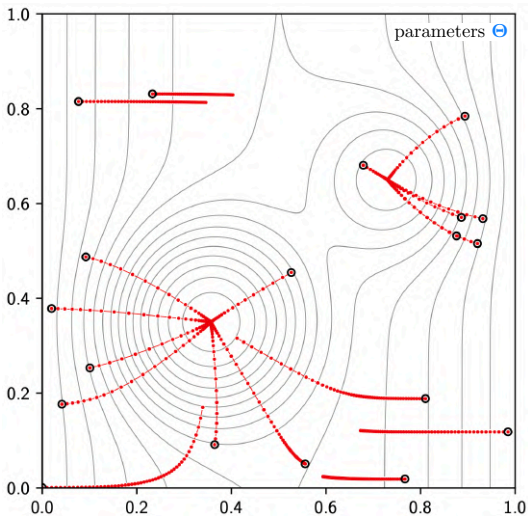


Figure adapted from [Fleuret, 2022]



## DESCENTE DE GRADIENT

## Movie 5

## Movie 6

## Movie 7

## Movie 8

Videos taken from the “3 Blue 1 Braun” YouTube channel

# BACKPROPAGATION

- ▶ La **backpropagation** est un algorithme qui implémente une descente de gradient de manière efficace.
- ▶ En résumé, on utilise le calcul des gradients de la couche  $l + 1$  pour calculer les gradients de la couche  $l$ .
- ▶ On calcule donc tous les gradients. depuis la dernière couche, jusqu'à la première couche: **backward pass**.
- ▶ On omet les équation ici...

# BACKPROPAGATION

- ▶ La **backpropagation** est un algorithme qui implémente une descente de gradient de manière efficace.
- ▶ En résumé, on utilise le calcul des gradients de la couche  $l + 1$  pour calculer les gradients de la couche  $l$ .
- ▶ On calcule donc tous les gradients. depuis la dernière couche, jusqu'à la première couche: **backward pass**.
- ▶ On omet les équation ici...

# BACKPROPAGATION

- ▶ La **backpropagation** est un algorithme qui implémente une descente de gradient de manière efficace.
- ▶ En résumé, on utilise le calcul des gradients de la couche  $l + 1$  pour calculer les gradients de la couche  $l$ .
- ▶ On calcule donc tous les gradients. depuis la dernière couche, jusqu'à la première couche: **backward pass**.
- ▶ On omet les équation ici...

# BACKPROPAGATION

- ▶ La **backpropagation** est un algorithme qui implémente une descente de gradient de manière efficace.
- ▶ En résumé, on utilise le calcul des gradients de la couche  $l + 1$  pour calculer les gradients de la couche  $l$ .
- ▶ On calcule donc tous les gradients. depuis la dernière couche, jusqu'à la première couche: **backward pass**.
- ▶ On omet les équation ici...

INTRODUCTION  
○

BIOLOGIE  
○○○

1<sup>ÈRE</sup> GÉNÉRATION  
○○○○○○○○○○  
○○○○○

2<sup>ÈME</sup> GÉNÉRATION  
○○○○○○○○○○  
○○○○○○○○○●○○

3<sup>ÈME</sup> GÉNÉRATION  
○○○○○  
○○○○

CONCLUSION  
○○

# BACKPROPAGATION

Movie 9

Movie 10

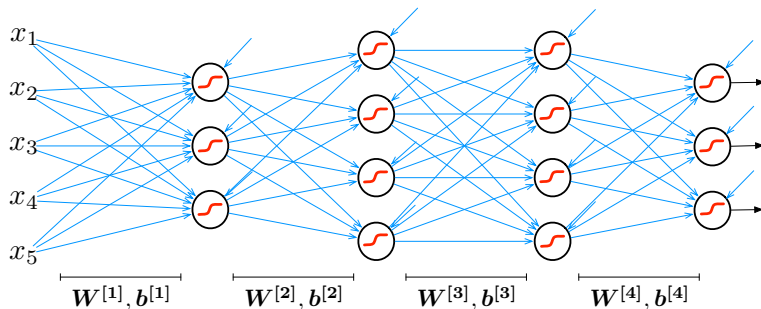
Movie 11

Movie 12

Videos taken from the “3 Blue 1 Braun” YouTube channel

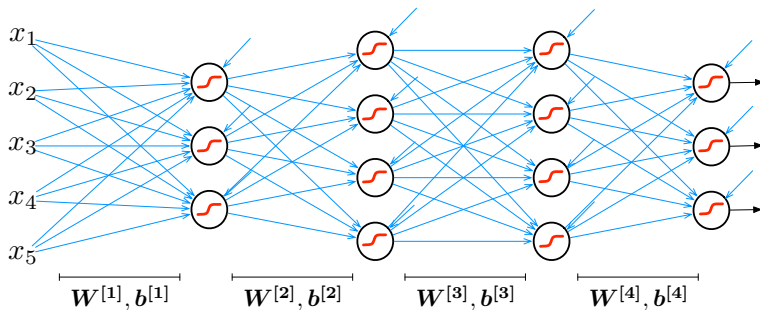
# BACKPROPAGATION

- ▶ Apprentissage: modification des poids synaptiques et des biais du réseau entier.
- ▶ Vaguement inspiré du concept crucial de **plasticité synaptique** en biologie.



# BACKPROPAGATION

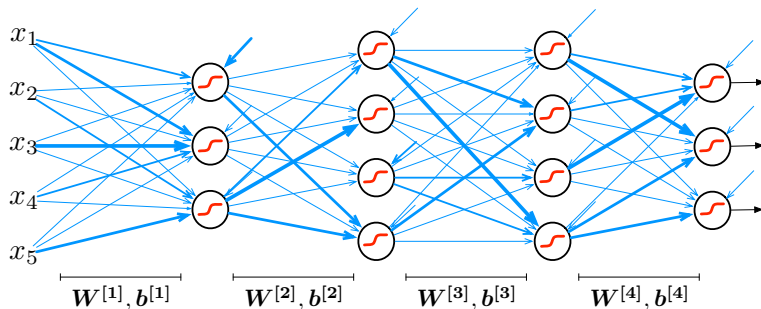
- ▶ Apprentissage: modification des poids synaptiques et des biais du réseau entier.
- ▶ Vaguement inspiré du concept crucial de **plasticité synaptique** en biologie.





# BACKPROPAGATION

- ▶ Apprentissage: modification des poids synaptiques et des biais du réseau entier.
- ▶ Vaguement inspiré du concept crucial de **plasticité synaptique** en biologie.



## EXEMPLE

**Démo notebook:** entraînement d'un réseau de neurones feedforward sur le dataset MNIST.



# RÉSEAUX DE NEURONES DE 3<sup>ÈME</sup> GÉNÉRATION

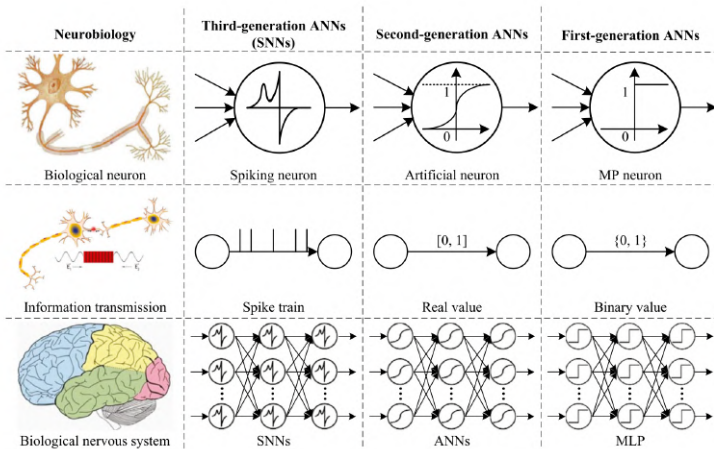
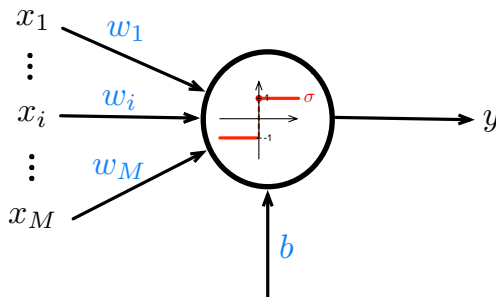


Figure taken from [Wang et al., 2020]

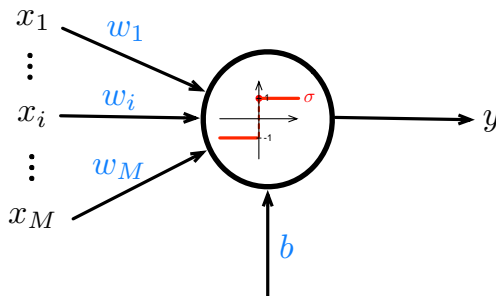
# NEURONES BIO-INSPIRÉS

- ▶ En deep learning, on utilise des neurones **neurones discrets** puis des **neurones continus**.
- ▶ Les réseaux de neurones à impulsion (spiking neural networks) utilisent des neurones bio-inspurés.



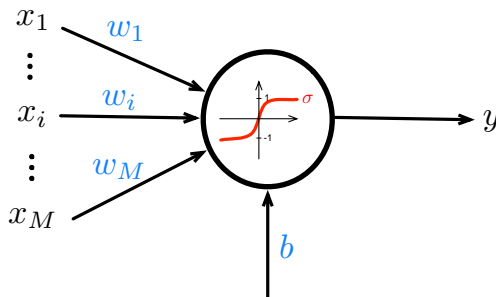
# NEURONES BIO-INSPIRÉS

- ▶ En deep learning, on utilise des neurones **neurones discrets** puis des **neurones continus**.
- ▶ Les **réseaux de neurones à impulsion (spiking neural networks)** utilisent des **neurones bio-inspurés**.



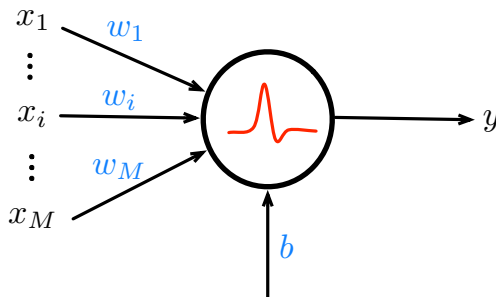
# NEURONES BIO-INSPIRÉS

- ▶ En deep learning, on utilise des neurones **neurones discrets** puis des **neurones continus**.
- ▶ Les **réseaux de neurones à impulsion (spiking neural networks)** utilisent des **neurones bio-inspurés**.



# NEURONES BIO-INSPIRÉS

- ▶ En deep learning, on utilise des neurones **neurones discrets** puis des **neurones continus**.
- ▶ Les réseaux de neurones à impulsion (spiking neural networks) utilisent des **neurones bio-inspurés**.



# NEURONES BIO-INSPIRÉS: LEAKY-INTEGRATE-AND-FIRE (LIF)

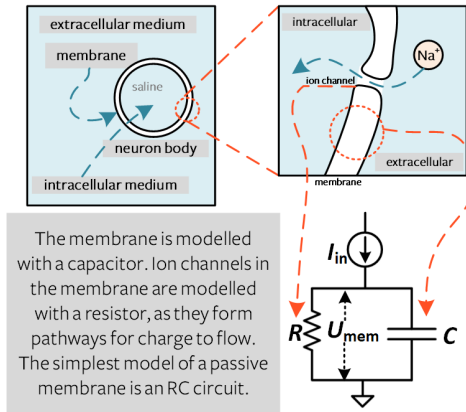
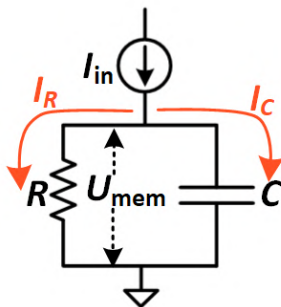


Figure taken from the `snnTorch` tutorial [Eshraghian et al., 2021].



# NEURONES BIO-INSPIRÉS: LEAKY-INTEGRATE-AND-FIRE (LIF)



$$I_{in}(t) = I_R + I_C$$

$$\Rightarrow I_{in}(t) = \frac{U_{\text{mem}}(t)}{R} + C \frac{dU_{\text{mem}}(t)}{dt}$$

$$\Rightarrow I_{in}(t)R = U_{\text{mem}}(t) + RC \frac{dU_{\text{mem}}(t)}{dt}$$

$$\Rightarrow U_{\text{mem}}(t) = I_{in}(t)R + c_1 e^{-t/RC}$$

$$\text{where at } t=0, U_{\text{mem}}(t) = U_0$$

$$\Rightarrow c_1 = U_0 - I_{in}(t)R$$

$$\Rightarrow U_{\text{mem}}(t) = I_{in}(t)R + [U_0 - I_{in}(t)R]e^{-t/RC}$$

Figure taken from the [snnTorch tutorial](#) [Eshraghian et al., 2021].

# NEURONES BIO-INSPIRÉS: LEAKY-INTEGRATE-AND-FIRE (LIF)

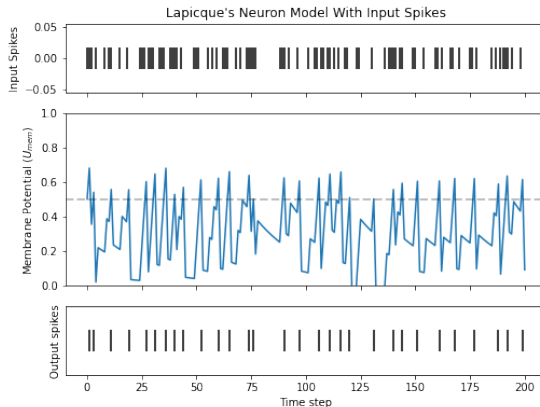
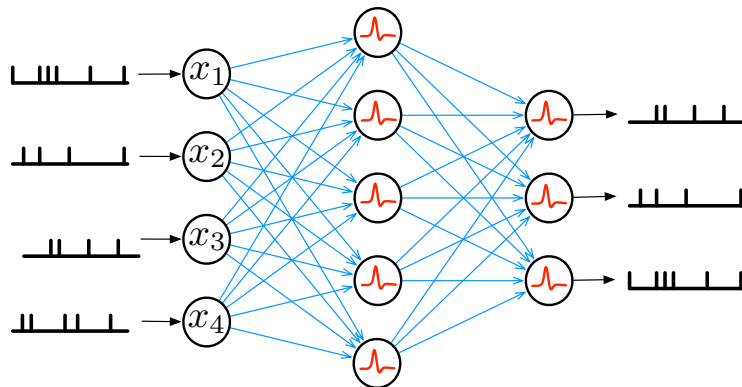


Figure taken from the `snnTorch` tutorial [Eshraghian et al., 2021].

# RÉSEAUX DE NEURONES À IMPULSIONS



# RÉSEAUX DE NEURONES À IMPULSIONS

Universalité computationnelle des réseaux de neurones à impulsions dans un paradigme de temporal coding.

## THEOREM ([MAASS, 1996])

- ▶ *Dans un paradigme de temporal coding (à valeurs rationnelles), toute machine de Turing peut être simulée par un réseau de neurones à impulsions.*
- ▶ *Dans un paradigme de temporal coding continu à valeurs réelles, les réseaux de neurones à impulsions sont super-Turing.*

# RÉSEAUX DE NEURONES À IMPULSIONS

Universalité computationnelle des réseaux de neurones à impulsions dans un paradigme de temporal coding.

## THEOREM ([MAASS, 1996])

- ▶ *Dans un paradigme de temporal coding (à valeurs rationnelles), toute machine de Turing peut être simulée par un réseau de neurones à impulsions.*
- ▶ *Dans un paradigme de temporal coding continu à valeurs réelles, les réseaux de neurones à impulsions sont super-Turing.*

# APPRENTISSAGE

- ▶ Généralisation de la “backpropagation through time”.
- ▶ Mécanismes Hebbiens de plasticité synaptique: spike-timing-dependent plasticity (STDP), etc.
- ▶ Combinaison des deux...

# APPRENTISSAGE

- ▶ Généralisation de la “backpropagation through time”.
- ▶ Mécanismes Hebbiens de plasticité synaptique: spike-timing-dependent plasticity (STDP), etc.
- ▶ Combinaison des deux...

# APPRENTISSAGE

- ▶ Généralisation de la “backpropagation through time”.
- ▶ Mécanismes Hebbiens de plasticité synaptique: spike-timing-dependent plasticity (STDP), etc.
- ▶ Combinaison des deux...



# COMPUTATION NEUROMORPHIQUE

- ▶ **Computation neuromorphique (neuromorphic computing):** implémentation des spiking neural networks sur des hardware analogiques: *computationnellement et énergétiquement très efficace (règle des trois S)!*
- ▶ **Communication par impulsions (spikes):** le hardware gère des valeurs discrètes (0 et 1) plutôt que continues: calculs plus rapides, moins de mémoire utilisée.
- ▶ **Activité sporadique (sparsity):** la plupart du temps, les neurones sont au repos: calculs plus rapides (opérations sparses), moins de mémoire utilisée.
- ▶ **Traitement dynamique de l'information (static-suppression):** une nouvelle information est traitée seulement si elle diffère de la précédente: moins de calculs, moins d'énergie consommée.

# COMPUTATION NEUROMORPHIQUE

- ▶ **Computation neuromorphique (neuromorphic computing):** implémentation des spiking neural networks sur des hardware analogiques: *computationnellement et énergétiquement très efficace (règle des trois S)!*
- ▶ **Communication par impulsions (spikes):** le hardware gère des valeurs discrètes (0 et 1) plutôt que continues: calculs plus rapides, moins de mémoire utilisée.
- ▶ **Activité sporadique (sparsity):** la plupart du temps, les neurones sont au repos: calculs plus rapides (opérations sparses), moins de mémoire utilisée.
- ▶ **Traitement dynamique de l'information (static-suppression):** une nouvelle information est traitée seulement si elle diffère de la précédente: moins de calculs, moins d'énergie consommée.

# COMPUTATION NEUROMORPHIQUE

- ▶ **Computation neuromorphique (neuromorphic computing):** implémentation des spiking neural networks sur des hardware analogiques: *computationnellement et énergétiquement très efficace (règle des trois S)!*
- ▶ **Communication par impulsions (spikes):** le hardware gère des valeurs discrètes (0 et 1) plutôt que continues: calculs plus rapides, moins de mémoire utilisée.
- ▶ **Activité sporadique (sparsity):** la plupart du temps, les neurones sont au repos: calculs plus rapides (opérations sparses), moins de mémoire utilisée.
- ▶ **Traitement dynamique de l'information (static-suppression):** une nouvelle information est traitée seulement si elle diffère de la précédente: moins de calculs, moins d'énergie consommée.

# COMPUTATION NEUROMORPHIQUE

- ▶ **Computation neuromorphique (neuromorphic computing):** implémentation des spiking neural networks sur des hardware analogiques: *computationnellement et énergétiquement très efficace (règle des trois S)!*
- ▶ **Communication par impulsions (spikes):** le hardware gère des valeurs discrètes (0 et 1) plutôt que continues: calculs plus rapides, moins de mémoire utilisée.
- ▶ **Activité sporadique (sparsity):** la plupart du temps, les neurones sont au repos: calculs plus rapides (opérations sparses), moins de mémoire utilisée.
- ▶ **Traitement dynamique de l'information (static-suppression):** une nouvelle information est traitée seulement si elle diffère de la précédente: moins de calculs, moins d'énergie consommée.

# COMPUTATION NEUROMORPHIQUE

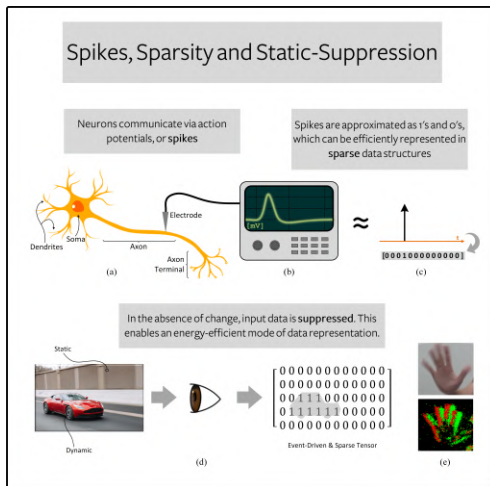


Figure taken from the `snnTorch` tutorial [Eshraghian et al., 2021].

## EXEMPLE

**Démo notebook:** entraînement d'un réseau de neurones à impulsions sur le dataset MNIST.



# CONCLUSION

- ▶ Les réseaux de neurones sont des algorithmes extrêmement efficaces de machine learning.
- ▶ Ils sont plus ou moins inspirés de la biologie: moins pour 1<sup>ère</sup> et 2<sup>ème</sup> générations et plus pour 3<sup>ème</sup> génération.
- ▶ Le cerveau possède des capacités computationnelles importantes à très faible énergie: les algos d'inspiration biologiques sont donc une piste de recherche très pertinente.
- ▶ La compréhension des algorithmes démystifie bien des illusions à propos de l'intelligence artificielle.

# CONCLUSION

- ▶ Les réseaux de neurones sont des algorithmes extrêmement efficaces de machine learning.
- ▶ Ils sont plus ou moins inspirés de la biologie: moins pour 1<sup>ère</sup> et 2<sup>ème</sup> générations et plus pour 3<sup>ème</sup> génération.
- ▶ Le cerveau possède des capacités computationnelles importantes à très faible énergie: les algos d'inspiration biologiques sont donc une piste de recherche très pertinente.
- ▶ La compréhension des algorithmes démystifie bien des illusions à propos de l'intelligence artificielle.



# CONCLUSION

- ▶ Les réseaux de neurones sont des algorithmes extrêmement efficaces de machine learning.
- ▶ Ils sont plus ou moins inspirés de la biologie: moins pour 1<sup>ère</sup> et 2<sup>ème</sup> générations et plus pour 3<sup>ème</sup> génération.
- ▶ Le cerveau possède des capacités computationnelles importantes à très faible énergie: les algos d'inspiration biologiques sont donc une piste de recherche très pertinente.
- ▶ La compréhension des algorithmes démystifie bien des illusions à propos de l'intelligence artificielle.

# CONCLUSION

- ▶ Les réseaux de neurones sont des algorithmes extrêmement efficaces de machine learning.
- ▶ Ils sont plus ou moins inspirés de la biologie: moins pour 1<sup>ère</sup> et 2<sup>ème</sup> générations et plus pour 3<sup>ème</sup> génération.
- ▶ Le cerveau possède des capacités computationnelles importantes à très faible énergie: les algos d'inspiration biologiques sont donc une piste de recherche très pertinente.
- ▶ La compréhension des algorithmes démystifie bien des illusions à propos de l'intelligence artificielle.

# BIBLIOGRAPHIE I



Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Benamoun, M., Jeong, D. S., and Lu, W. D. (2021).

Training spiking neural networks using lessons from deep learning.

*CoRR*, abs/2109.12894.



Fleuret, F. (2022).

Deep Learning Course.



Kleene, S. C. (1956).

Representation of events in nerve nets and finite automata.

In Shannon, C. and McCarthy, J., editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ.



Maass, W. (1996).

Lower bounds for the computational power of networks of spiking neurons.

*Neural Computation*, 8(1):1–40.



McCulloch, W. S. and Pitts, W. (1943).

A logical calculus of the ideas immanent in nervous activity.

*Bulletin of Mathematical Biophysic*, 5:115–133.

# BIBLIOGRAPHIE II



Minsky, M. L. (1967).  
*Computation: finite and infinite machines.*  
Prentice-Hall, Inc., Englewood Cliffs, N. J.



Minsky, M. L. and Papert, S. (1969).  
*Perceptrons: An Introduction to Computational Geometry.*  
MIT Press, Cambridge, MA, USA.



Rosenblatt, F. (1957).  
The perceptron: A perceiving and recognizing automaton.  
Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York.



Rosenblatt, F. (1958).  
The perceptron: A probabilistic model for information storage and organization in the brain.  
*Psychological Review*, 65(6):386–408.



Siegelmann, H. T. and Sontag, E. D. (1994).  
Analog computation via neural networks.  
*Theor. Comput. Sci.*, 131(2):331–360.

# BIBLIOGRAPHIE III



Siegelmann, H. T. and Sontag, E. D. (1995).  
On the computational power of neural nets.  
*J. Comput. Syst. Sci.*, 50(1):132–150.



Wang, X., Lin, X., and Dang, X. (2020).  
Supervised learning in spiking neural networks: A review of algorithms and evaluations.  
*Neural Networks*, 125:258–280.