Fouille de Données

Data Mining

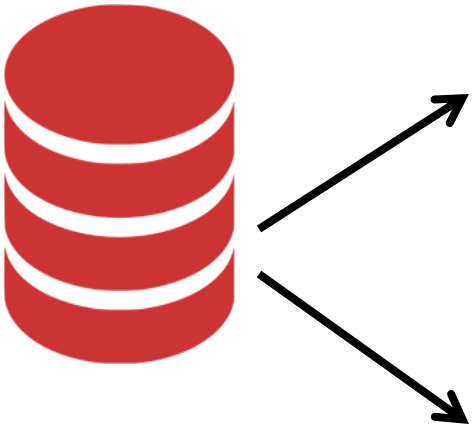
Classification - Partie 4

Plan du cours

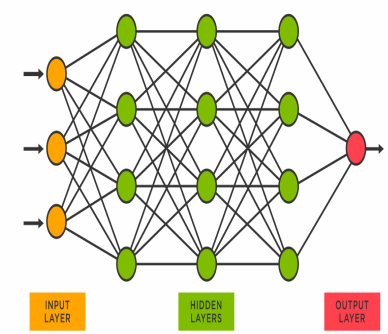
1. Les réseaux de Neurones
2. Deep Learning
3. Neurone Formel
4. Les réseaux de Neurones Artificiels
5. Perceptron et MLP
6. L'algorithme de Backpropagation

Classification

SAVOIR - PREDIRE - DECIDER



Données



Connaissances

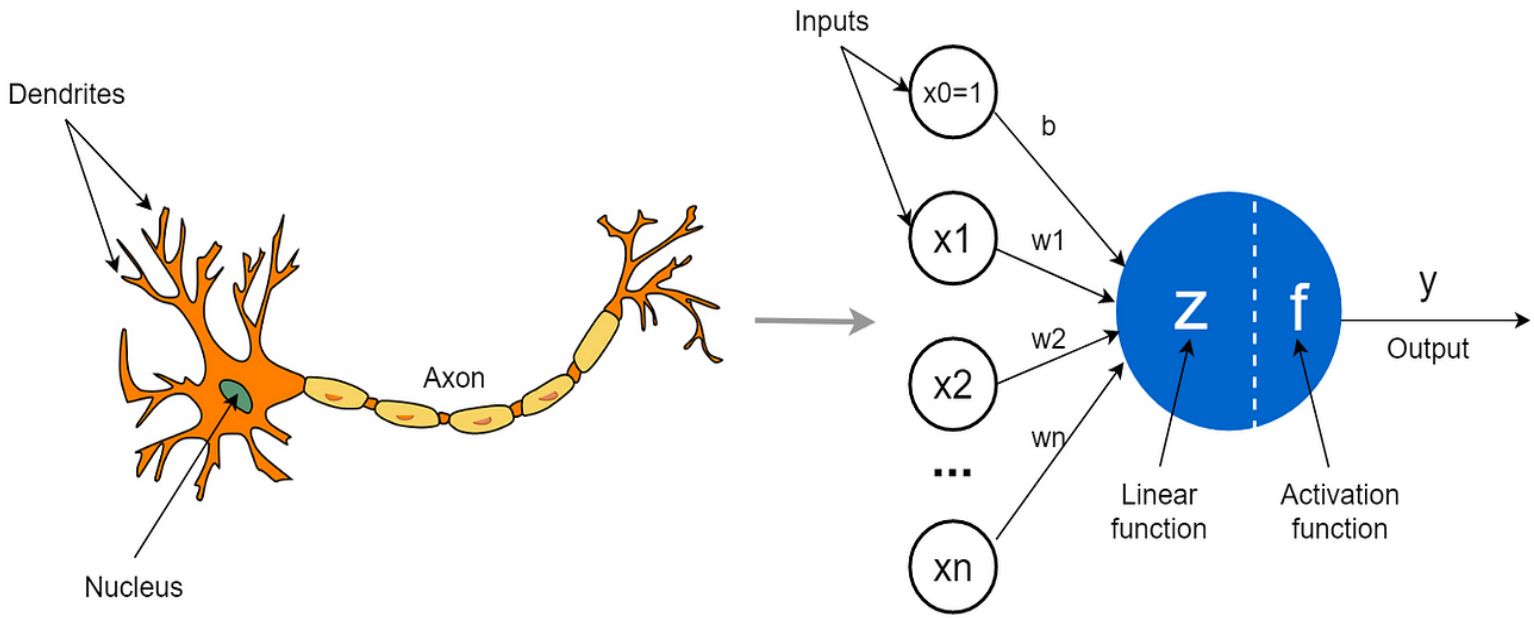
Réseaux de Neurones

Réseaux de Neurones

- Bio mimétisme.
- *Biomimicry: innovation inspired by nature*, Jeanine Benyus.
- « A chaque fois que vous rencontrez un problème, observez la nature. »

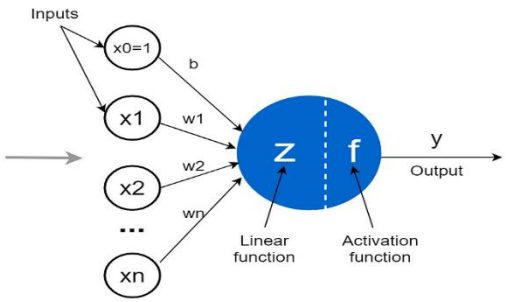
Réseaux de Neurones

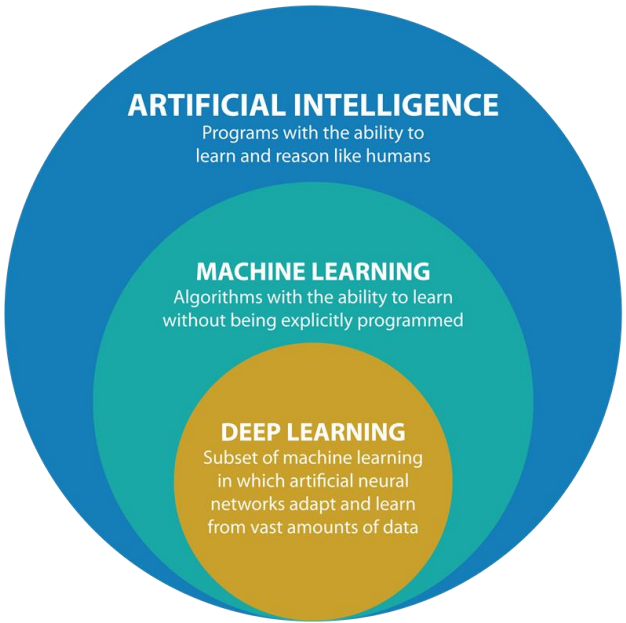
- Comment l'homme fait-il pour raisonner, parler, calculer, apprendre, ...?
- S'inspirer du fonctionnement du cerveau humain. ***Mais pas tellement.***
- Réseaux de neurones artificiels. ANN / RNA.



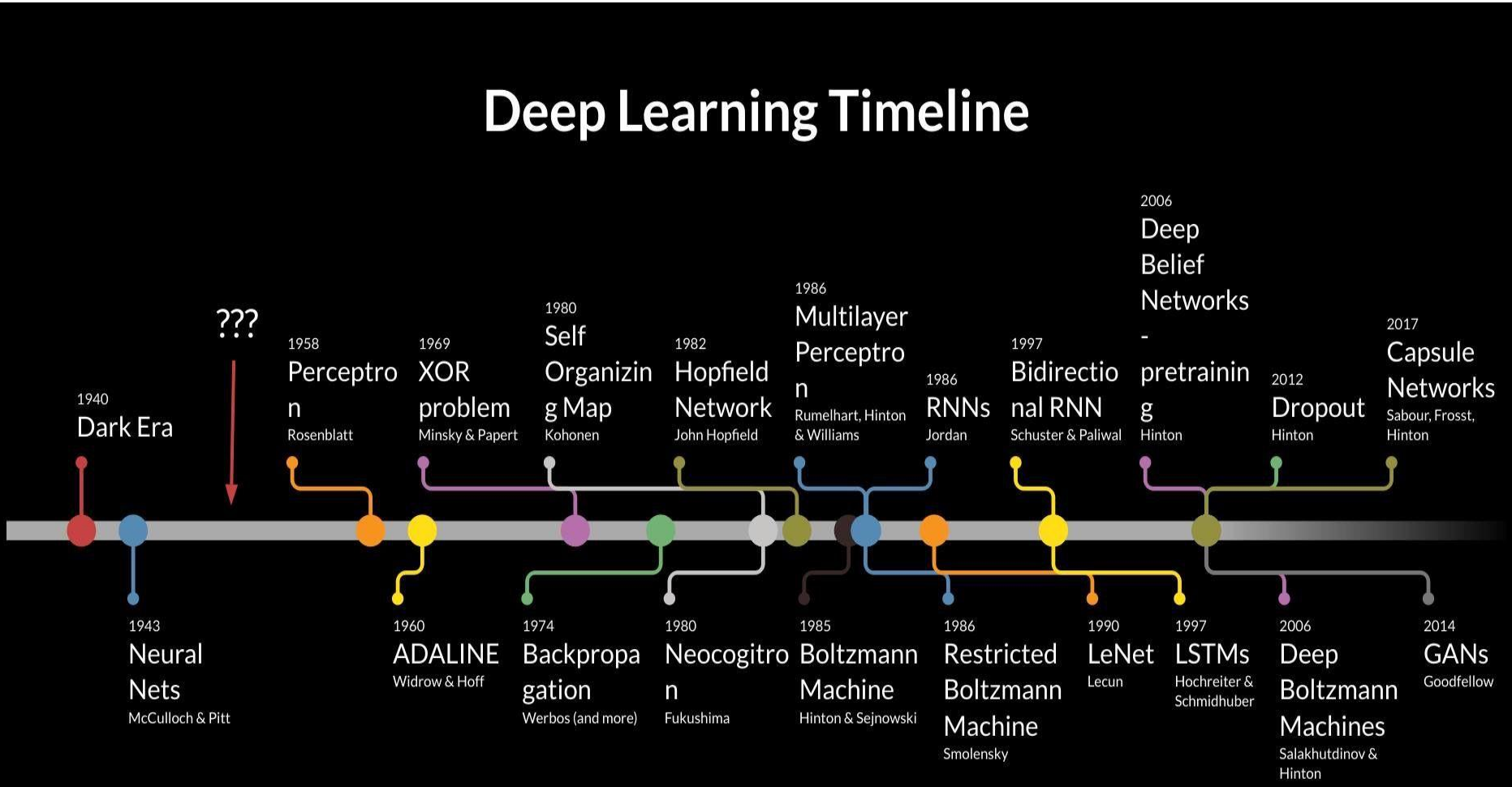
Réseaux de Neurones

- Mac Culloch et Pitts (**1943**) : définition d ’un **neurone formel**.
- Rosenblatt (**1958**), Widrow et Hoff : modèle avec processus d’apprentissage, **perceptron**.
- Minsky et Papert (**1969**) : limites des perceptrons.
- Rumelhart, Hinton, Williams (**1986**): **perceptron multi-couches (MLP)**, mécanismes d ’apprentissage performants (backpropagation du gradient).
- LeCun (**1990**): LeNet.
- **Deep Learning : Apprentissage Profond.**





Deep Learning



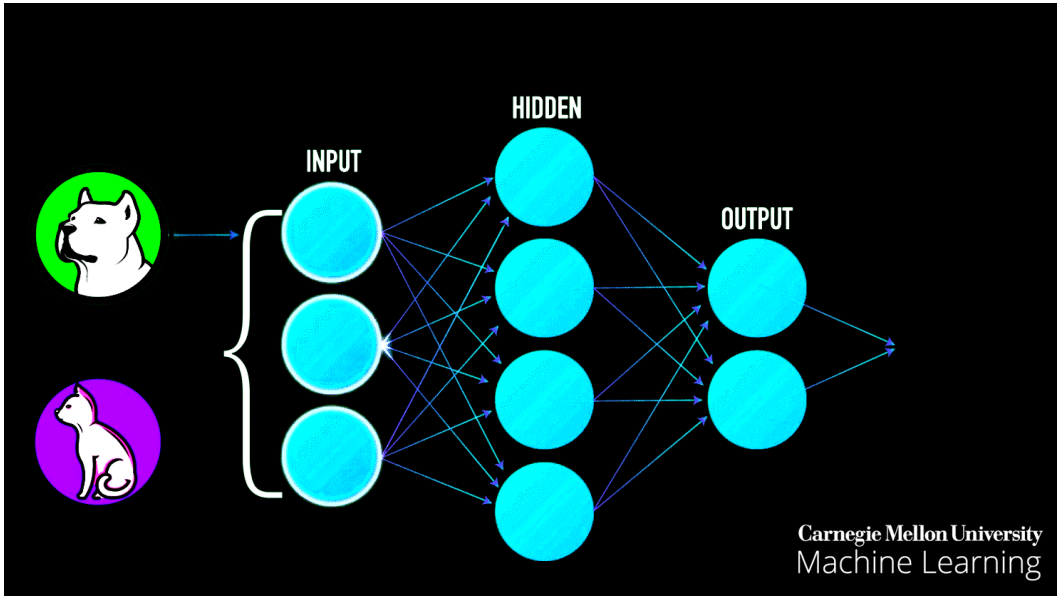
Made by Favio Vázquez

Deep Learning

- Deep Learning ou **Apprentissage Profond** : la machine est capable d'apprendre par elle-même.

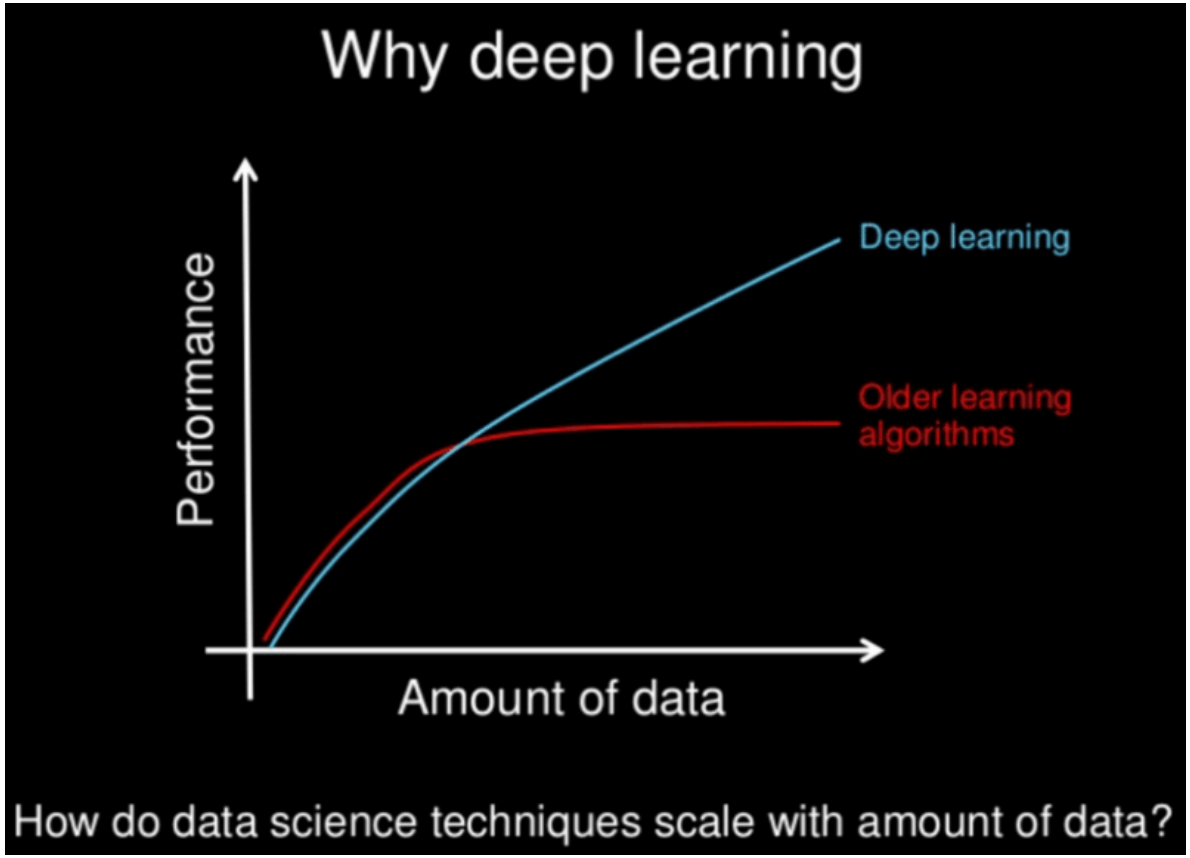
- Une technique, sous domaine, du Machine Learning reposant sur le modèle des réseaux neurones.

- Des dizaines voire des centaines de couches de neurones sont empilées pour apporter une plus grande complexité permettant un meilleur apprentissage.



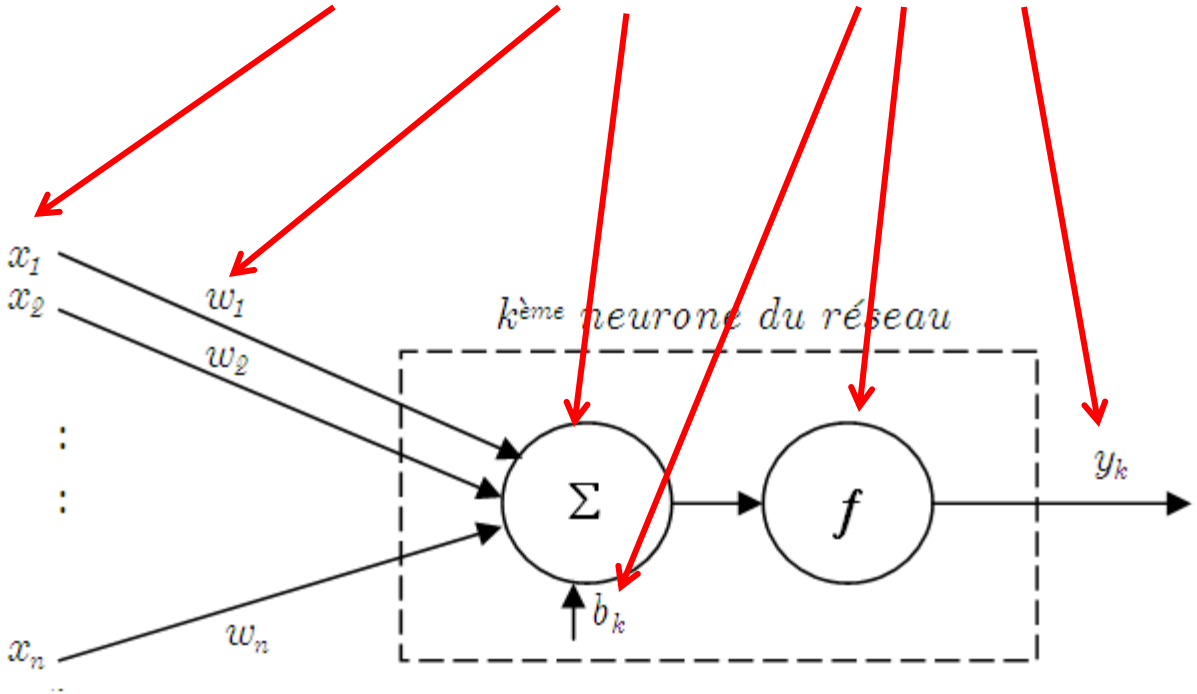
Deep Learning

- Les modèles de Deep Learning ont tendance à bien fonctionner avec une grande quantité de données alors que les modèles d'apprentissage automatique plus classique cessent de s'améliorer après un point de saturation.



Neurone Formel

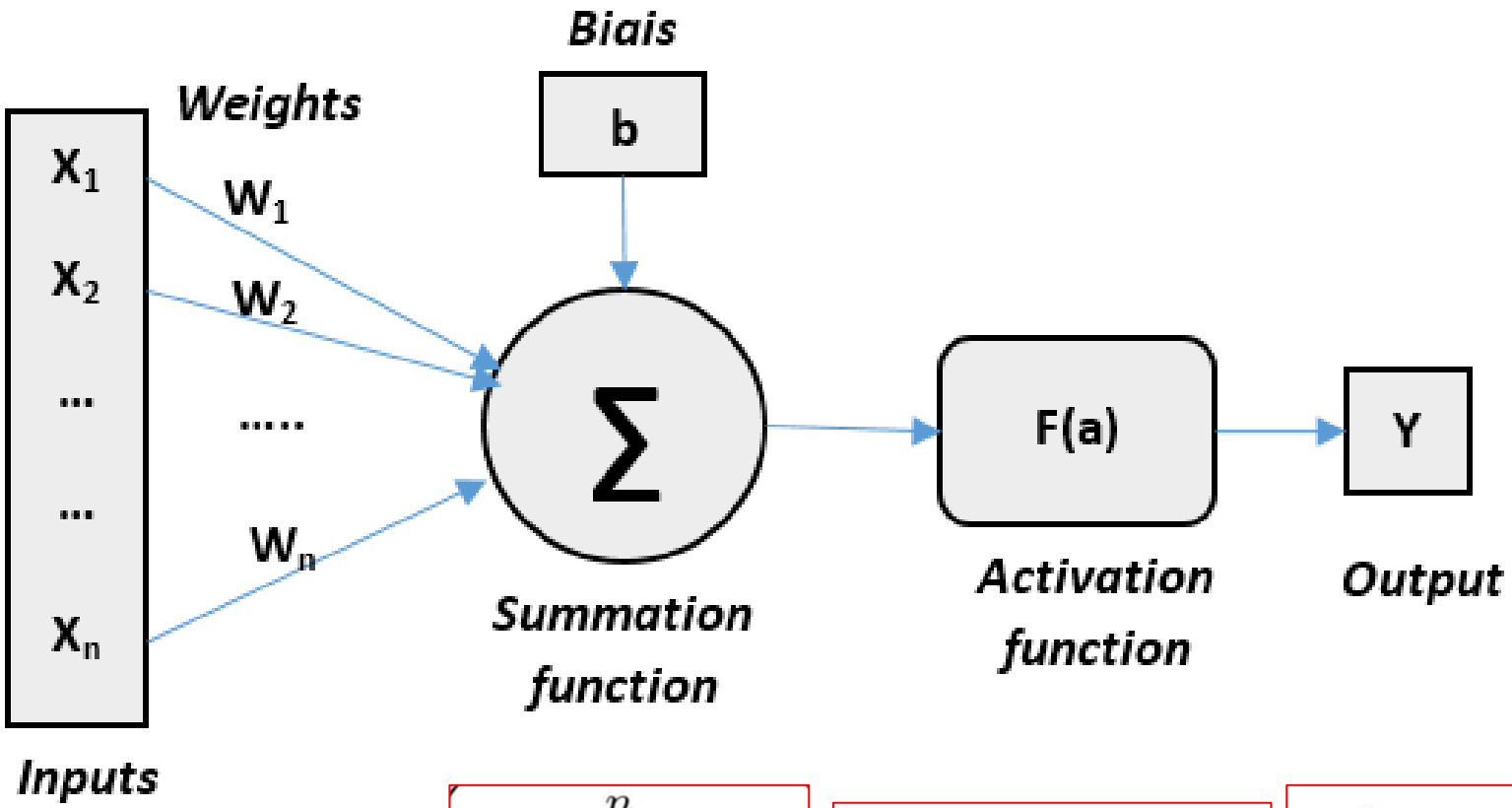
- McCulloch et Pitts (1943)
- Model mathématique d'un neurone formel très simple.
- Réseau : Nœuds interconnectés par des liaisons directionnelles.
- Il se constitue de **quelques éléments basiques** :



Entrées - Poids - Additionneur - Biais - Fonction d'activation - Sortie

Neurone Formel

- Un poids **Wi** est associé à chaque entrée **Xi**.



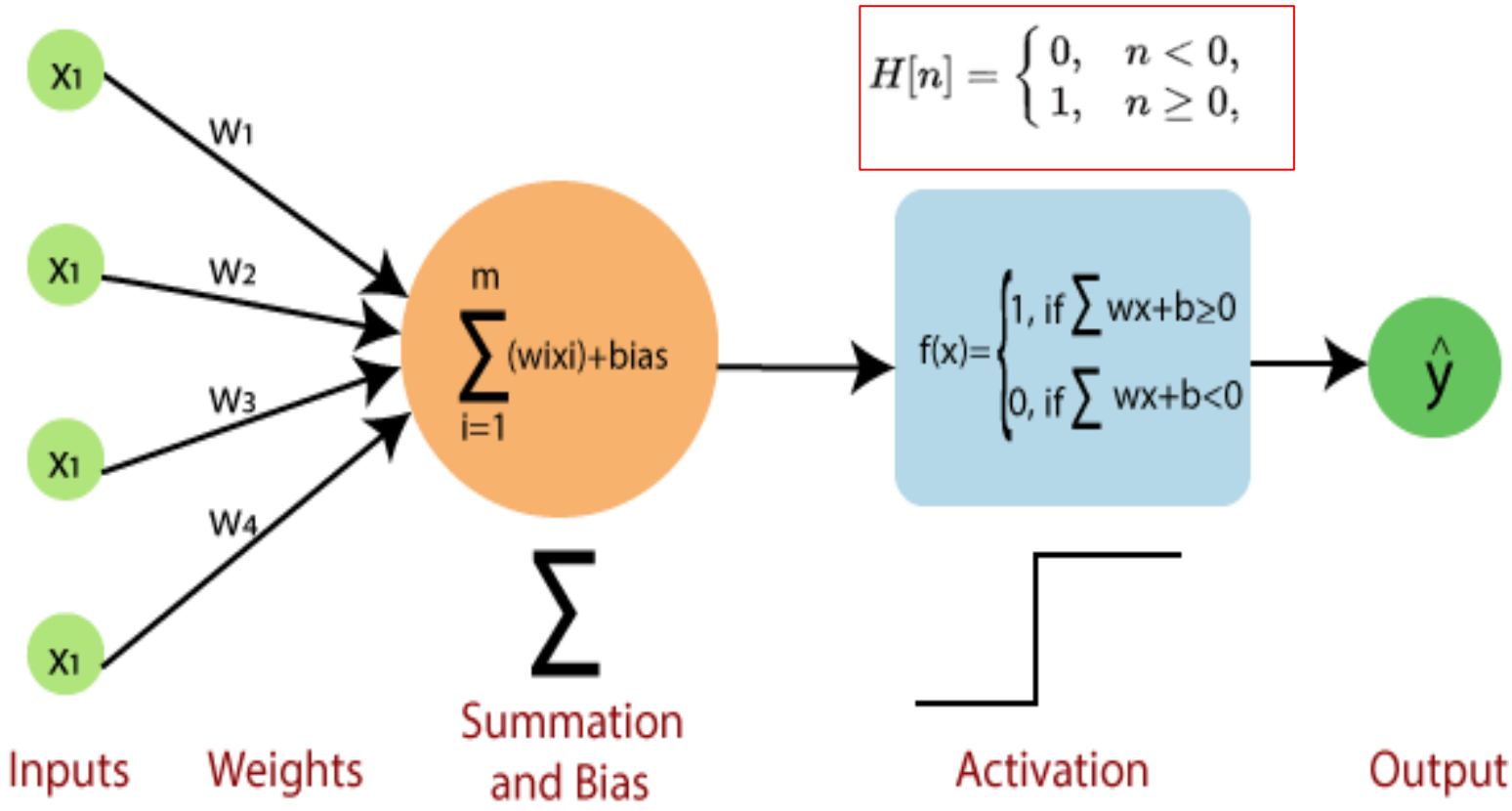
$$b + \sum_{i=1}^{n} x_i w_i$$

$$H[n] = \begin{cases} 0, & n < 0, \\ 1, & n \geq 0, \end{cases}$$

$$f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

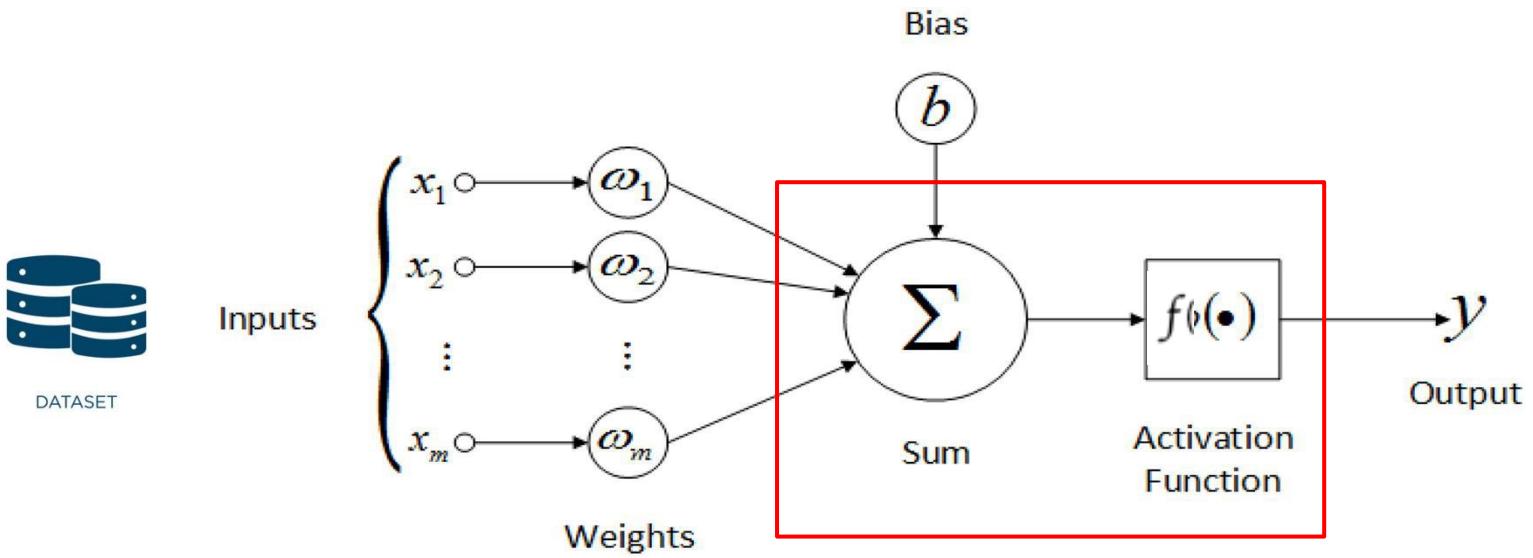
Neurone Formel

- Un poids **Wi** est associé à chaque entrée **Xi**.



Neurone Formel

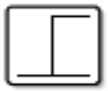
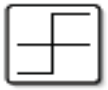
- Biais – Fonction d'activation, pourquoi ?
  - Introduire la non-linéarité dans la sortie du neurone.
  - La plupart des données du monde réel ne sont pas linéaires >> Apprendre aux neurones ces représentations non linéaires.



Input du neurone = Somme (Xi*Wi + b) **Output** du neurone = F(Input du neurone)

Neurone Formel

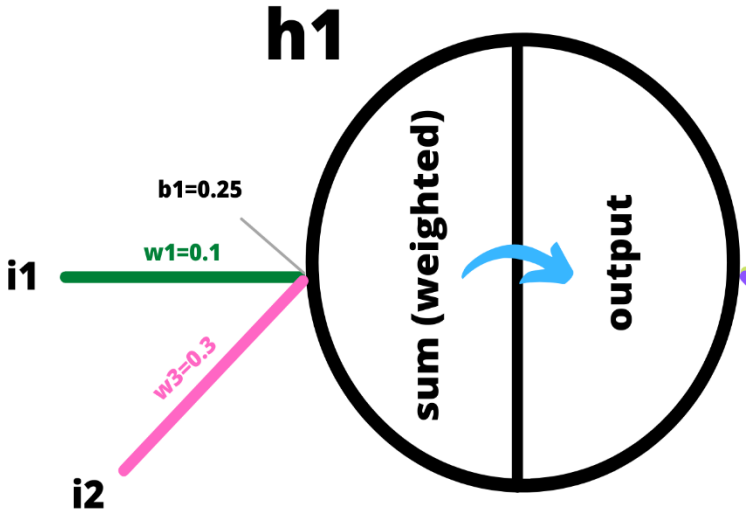
- **Fonctions d'activation**:

Nom de la fonction	Relation d'entrée/sortie	Icône
seuil Heaviside	$a = 0$ si $n < 0$ $a = 1$ si $n \geq 0$	
seuil symétrique	$a = -1$ si $n < 0$ $a = 1$ si $n \geq 0$	
linéaire	$a = n$	
linéaire saturée	$a = 0$ si $n < 0$ $a = n$ si $0 \leq n \leq 1$ $a = 1$ si $n > 1$	
linéaire saturée symétrique	$a = -1$ si $n < -1$ $a = n$ si $-1 \leq n \leq 1$ $a = 1$ si $n > 1$	
linéaire positive	$a = 0$ si $n < 0$ $a = n$ si $n \geq 0$	
sigmoïde	$a = \frac{1}{1+\exp^{-n}}$	
tangente hyperbolique	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$	
compétitive	$a = 1$ si n maximum $a = 0$ autrement	

Neurone Formel

- **Exemple :**

i1	i2	W1	W3	b1
0,1	0,5	0,1	0,3	0,25



- La **somme** du neurone h1:

$$sum_{h1} = i_1 * w_1 + i_2 * w_3 + b_1$$

$$sum_{h1} = 0.1 * 0.1 + 0.5 * 0.3 + 0.25 = 0.41$$

- La **sortie** du neurone h1 avec la **fonction d'activation** Sigmoide :

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

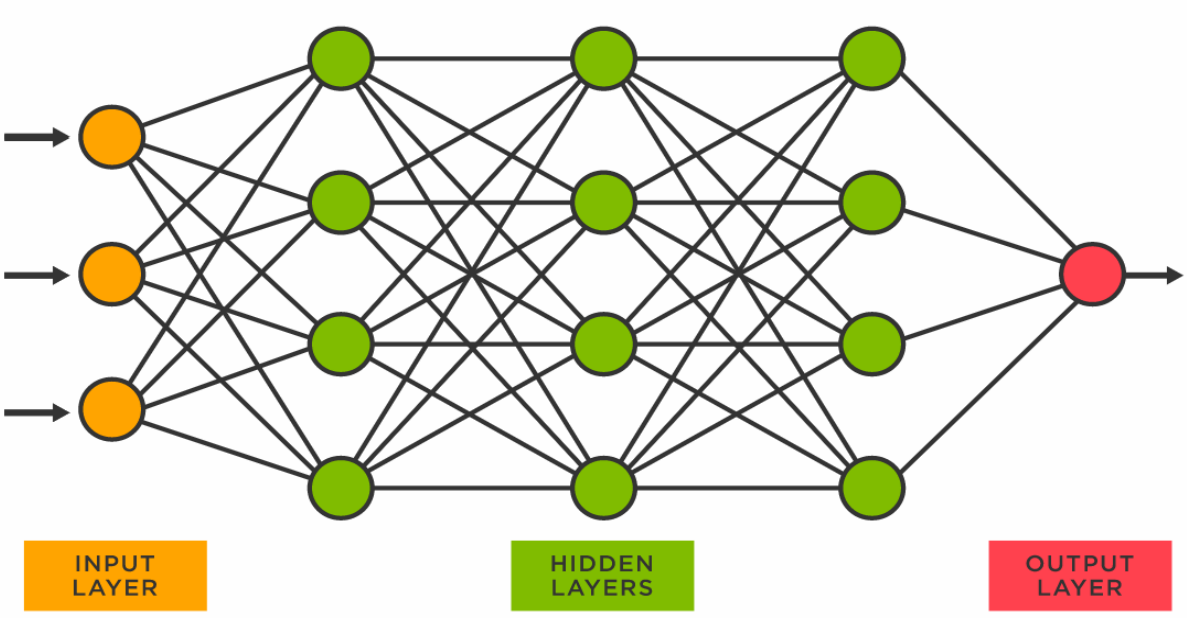
$$output_{h1} = \frac{1}{1 + e^{-sum_{h1}}}$$

$$output_{h1} = \frac{1}{1 + e^{-0.41}} = 0.60108$$

https://theneuralblog.com/forward-pass-backpropagation-example/

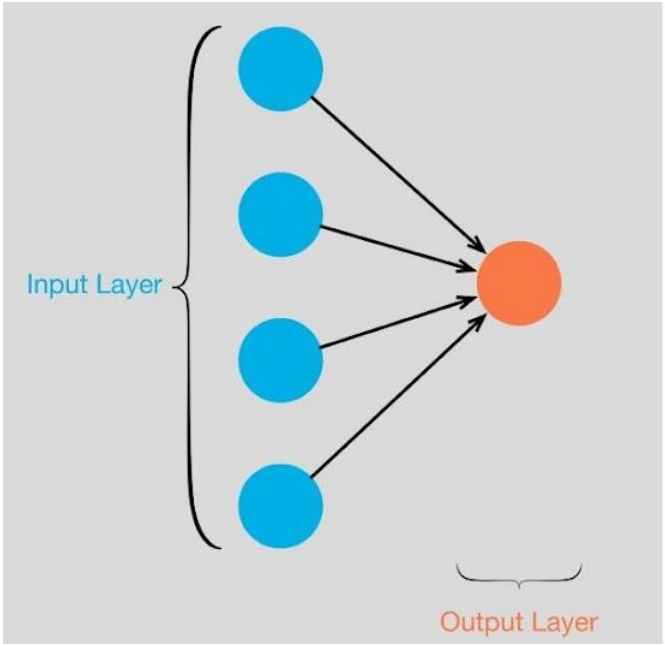
Réseaux de Neurones Artificiels

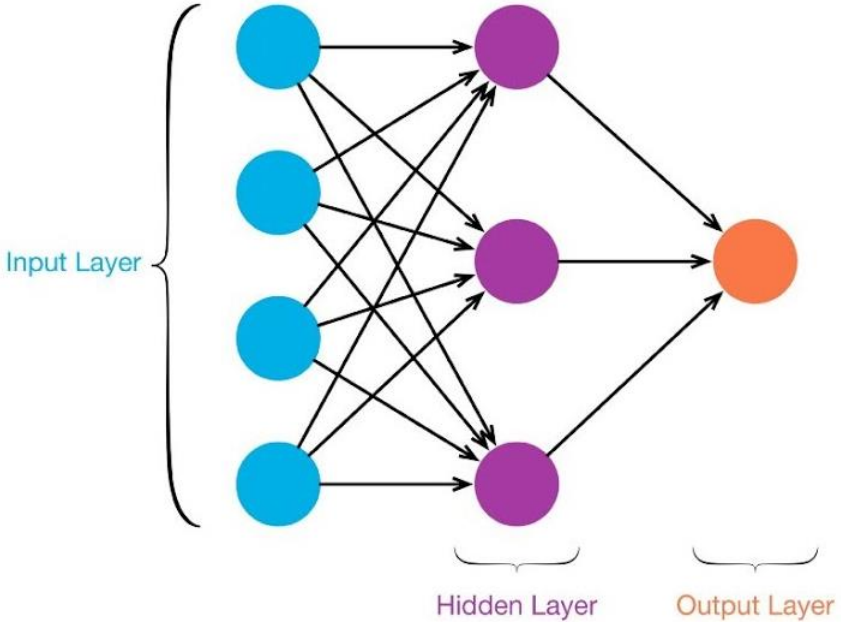
- Artificial Neural Networks : **ANN**. Neural networks (**NN**).
- Réseau de neurones artificiel : Plusieurs neurones/nœuds interconnectés par des liaisons directionnelles.
- Organisé en **trois parties** : Couche d'Entrée (Input Layer), Couches Cachées (Hidden Layers), et couche de Sortie (Output Layer).



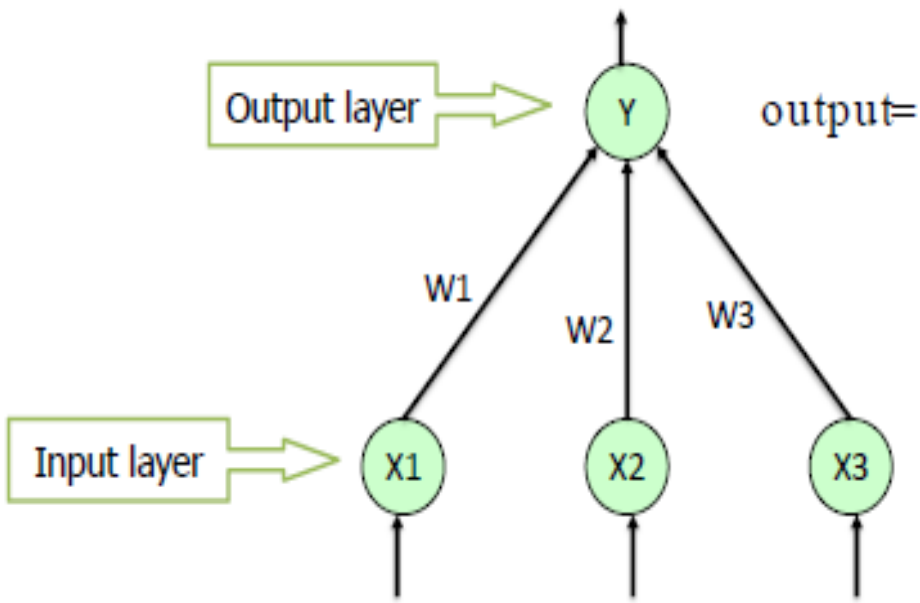
Réseaux de Neurones Artificiels

- Artificial Neural Networks : ANN.
- Nombreux modèles mathématiques ont été développés dans la littérature.
- Parmi eux: Le **Perceptron** de Rosenblatt (1958) et le **Multi Layer Perceptron** de Rumelhart, Hinton, Williams (1986).





- Neural Networks - Single Layer **Perceptron**



$$\text{output} = \begin{cases} 1 & \text{if } \sum w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

Activation – Seuil - Heaviside

Output

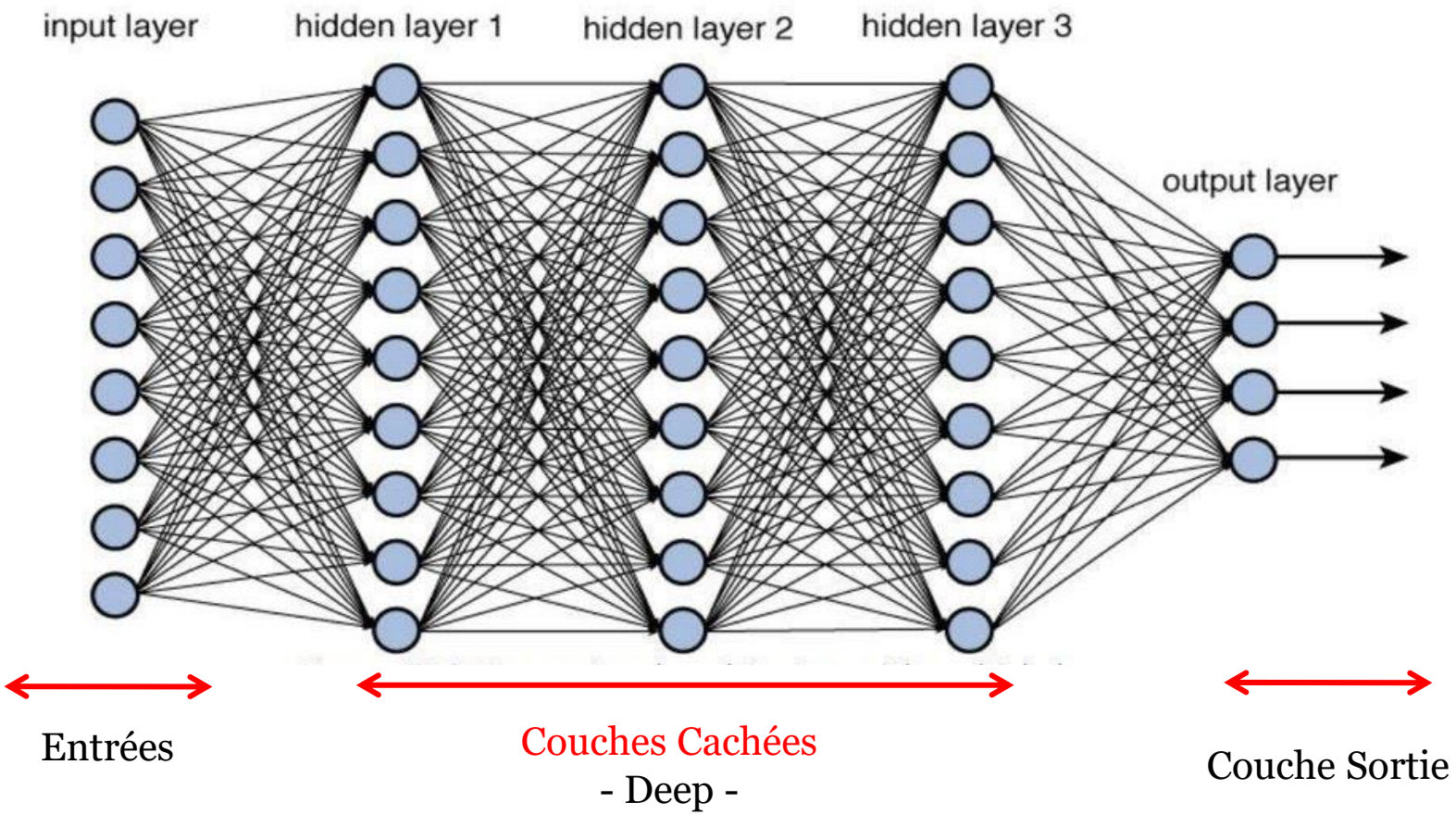
$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta \Rightarrow 1$$

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \leq \theta \Rightarrow 0$$

Réseaux de neurones artificiels

- Neural Networks - **Multi Layer Perceptron (MLP)**

Activation - Sigmoïde



- Neural Networks - **Multi Layer Perceptron (MLP)**

Type de Couche	**Nombre de Neurones**	**Règle / Déterminant**
Couche d'Entrée	Égal au nombre d'attributs (features)	Fixé par la dimension des données (ex: nombre de colonnes dans un tableau).
Couches Cachées	Variable (Hyperparamètre)	On commence généralement petit et on augmente si le modèle "sous-apprend" (underfitting).
Couche de Sortie	Dépend de la tâche	Fixé par le type de problème à résoudre (Régression, Classification binaire ou Classification multiple).

- Neural Networks - **Multi Layer Perceptron (MLP)**

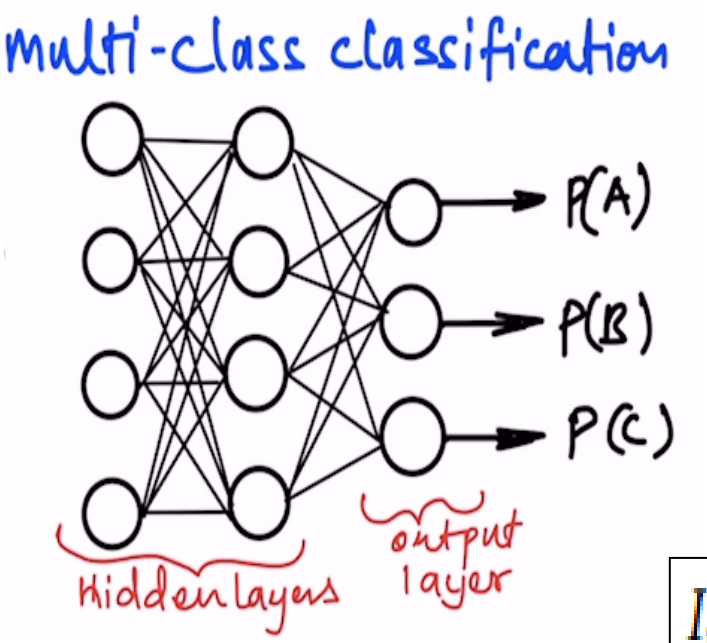
Type de Problème	**Nombre de Neurones de Sortie**	**Fonction d'activation de sortie suggérée**
Classification Binaire (Oui/Non, Spam/Ham)	**1**	Sigmoide $\frac{1}{1+e^{-x}}$
Classification Multi-classes (Chien/Chat/Oiseau)	**N** (Nombre de classes)	Softmax $\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
Régression (Prédire un prix, une température)	**1**	Linéaire (Pas d'activation) ou parfois ReLU (Rectified Linear Unit) $max\,(0, x)$

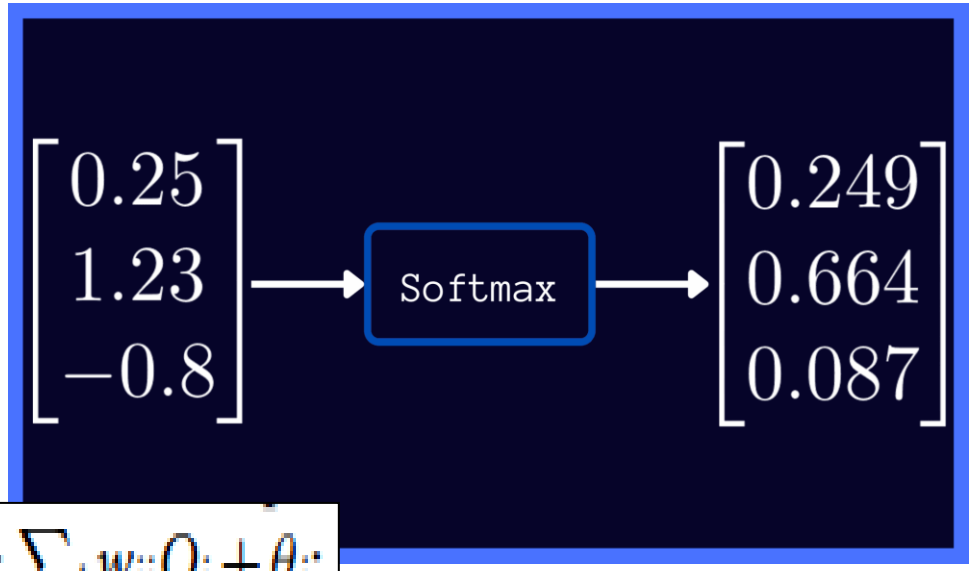
Réseaux de neurones artificiels

- Neural Networks - **Multi Layer Perceptron (MLP)**

The Softmax Activation Function

$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$





$$I_j = \sum_i w_{ij} O_i + \theta_j;$$

- Neural Networks - **Multi Layer Perceptron (MLP)**

The Softmax Activation Function

$$\begin{bmatrix} 0.25 \\ 1.23 \\ -0.8 \end{bmatrix}$$

$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$Pr[y_i = seal] = softmax(\mathbf{z})_0 = \frac{e^{0.25}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.249$$

$$Pr[y_i = panda] = softmax(\mathbf{z})_1 = \frac{e^{1.23}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.664$$

$$Pr[y_i = duck] = softmax(\mathbf{z})_2 = \frac{e^{-0.8}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.087$$

https://www.pinecone.io/learn/softmax-activation/

- Neural Networks - **Multi Layer Perceptron (MLP)**

- Dans les RNA, tout est traité **sous forme numérique**, aussi bien : les attributs (features) que les classes (labels).

Type Problème	**Encodage classe**	**Couche sortie**	**Activation**
Binaire	0 ou 1	1 neurone	Sigmoid
Multi-classes (One-Hot)	[1,0,0], [0,1,0], [0, 0, 1] …	N neurones	Softmax
Multi-classes (label encoding)	0, 1, 2, … => [1,0,0], [0,1,0], [0, 0, 1] …	N neurones	Softmax

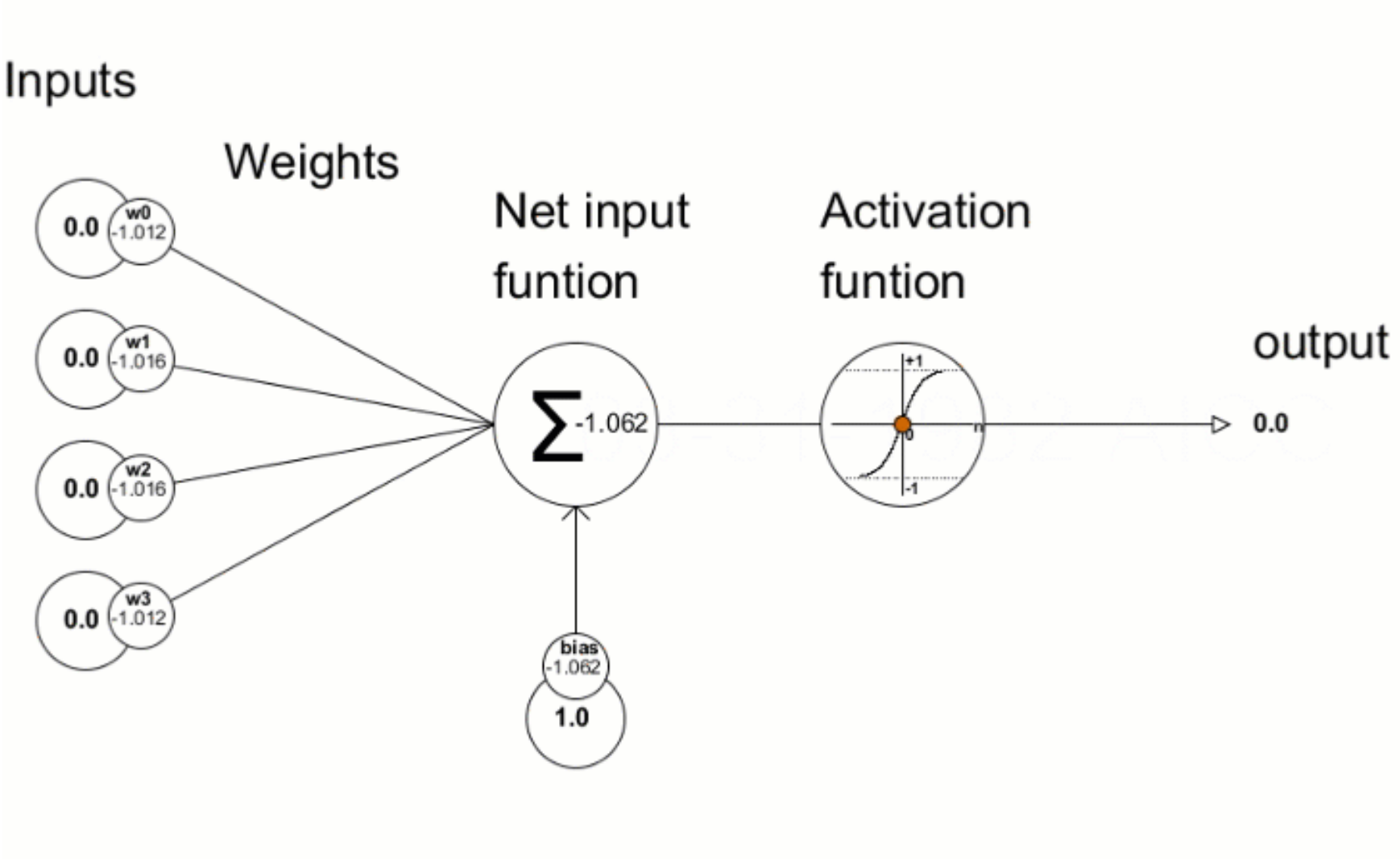
Réseaux de neurones artificiels

- Deux phases – états d'un ANN:
  - Apprentissage - Entrainement
  - Fonctionnement optimal - Utilisation

- Apprentissage :
  - Apprendre via un processus **itératif** et répétitif **d'ajustement des poids Wi** ;
  - Les valeurs des poids sont initialisées aléatoirement, puis **corrigées selon les erreurs** entres les valeurs de sortie attendues et obtenues: Forward propagation.
  - La correction se fait dans un sens inverse du sens de propagation des données : Backpropagation.

- Fonctionnement optimal : Une fois le réseau suffisamment calibré, il atteint un niveau où il n'est plus nécessaire de le superviser.

Réseaux de neurones artificiels



L'algorithme Backpropagation:

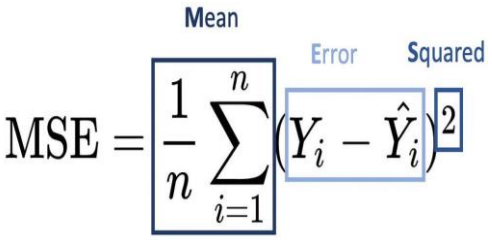
- À chaque présentation d'un exemple d'apprentissage au réseau, on passe par deux étapes :
  - ✓ Forward Propagation
  - ✓ Back Propagation

1. En Forward Propagation, on applique un ensemble de poids aux données d'entrée et on calcule une sortie. Pour la première propagation, l'ensemble des poids est sélectionné de manière aléatoire.

2. Dans la Back Propagation, on mesure la marge d'erreur de la sortie et on ajuste les poids en conséquence pour diminuer l'erreur.

3. Les réseaux neuronaux répètent les deux propagations jusqu'à ce que les poids soient calibrés pour prédire avec précision une sortie.

L'algorithme Backpropagation: MLP Backprop (Rumelhart/Hinton, 1986)

Composant	Original MLP Backprop (Rumelhart/Hinton)	Standard Aujourd'hui
Activation Cachée	**Sigmoïde**	**ReLU**
Activation Sortie	**Sigmoïde** (même pour multi-classes)	**Sigmoïde** (binaire) ou **Softmax** (multi-classes) ou **Linéaire** (régression)
Fonction Loss	**MSE** (Mean Squared Error) 	**Cross-Entropy** (Binary, Categorical, Sparse Categorical)

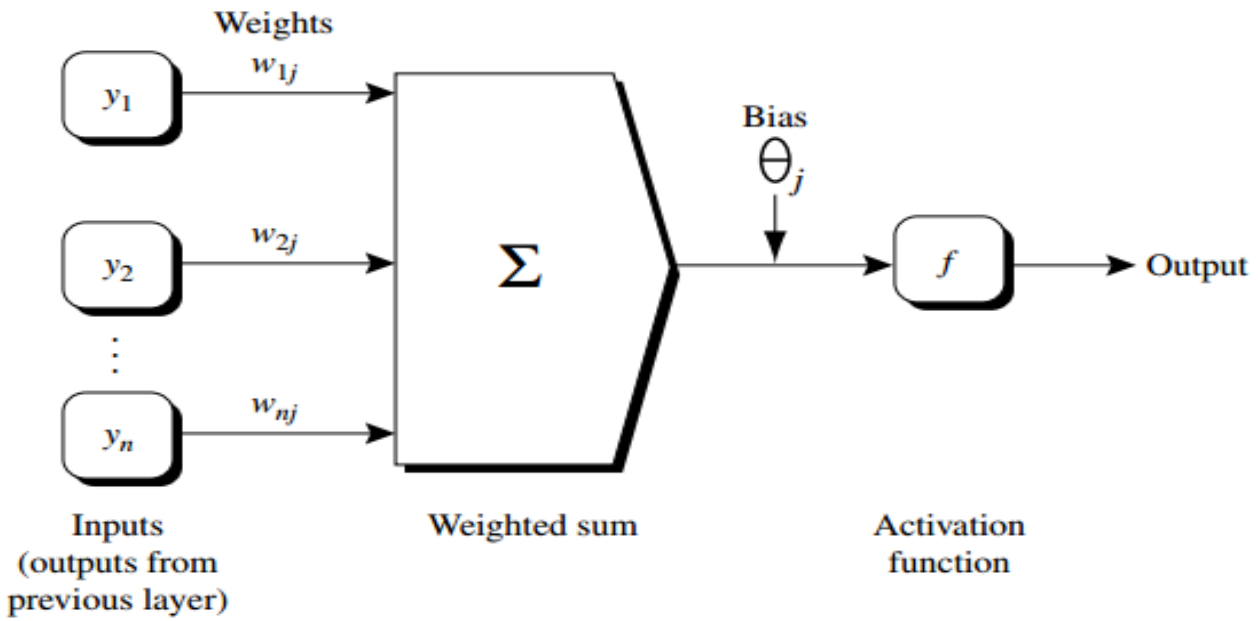
L'algorithme Backpropagation: Pseudo-code

// Propagate the inputs forward:

for each hidden or output layer unit j {

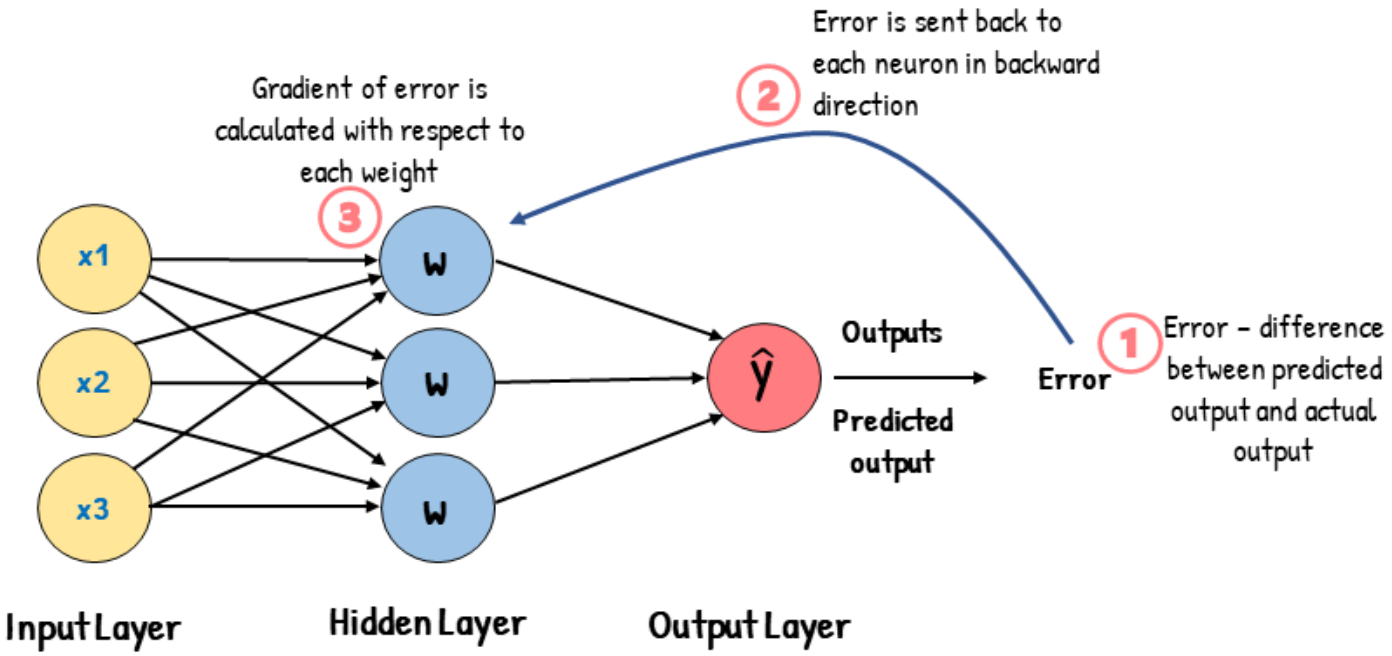
$I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i

$O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j



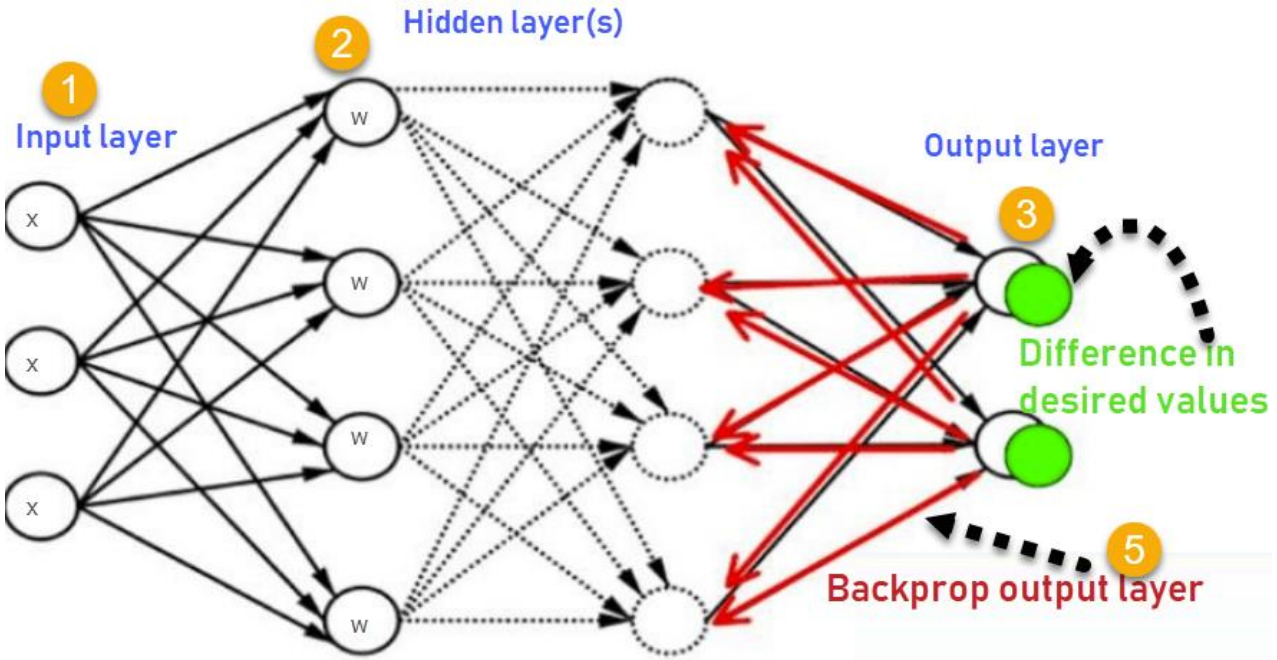
L'algorithme Backpropagation: Pseudo-code

// Backpropagate the errors:
for each unit j in the output layer
$Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error T: Target value
for each unit j in the hidden layers, from the last to the first hidden layer
$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k



L'algorithme Backpropagation: Pseudo-code

for each weight w_{ij} in *network* {
$\Delta w_{ij} = (l) Err_j O_i$; // weight increment
$w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
for each bias θ_j in *network* {
$\Delta \theta_j = (l) Err_j$; // bias increment
$\theta_j = \theta_j + \Delta \theta_j$; } // bias update
} }



L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

1 - Fixer **l'architecture** du ANN : nombre des neurones (couches cachées et couche sortie (selon le nombre des classes possibles).

2 - Initialisation aléatoire de l'ensemble des **poids** et des **biais** du ANN.

3 - Forward propagation : des couches cachées vers la couche des sorties:

- Calculer les inputs et les outputs pour chaque neurone :

```
1  Input = sum(weight_i * input_i) + bias
2  output = 1 / (1 + exp(-input))
```

Activation - Sigmoïde

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

$$O_j = \frac{1}{1+e^{-I_j}}$$

L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

4 – Backpropagate Error :

a - Calcul de Transfer Derivative : *Dérivée de la Sigmoïde*

```
1 transfer_derivative = output * (1 - output)
```

b - Calcul d'erreur pour chaque neurone : *Signal Error - Delta*

-**Cas** - Neurone couche sorties : *Dérivée de la MSE*

```
2 error = (expected_target - output) * transfer_derivative(output)
```

-Cas – Neurone couche cachée : weight et error des neurones précédents

```
3 error = transfer_derivative(output) * somme(weight_k * error_k)
```

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

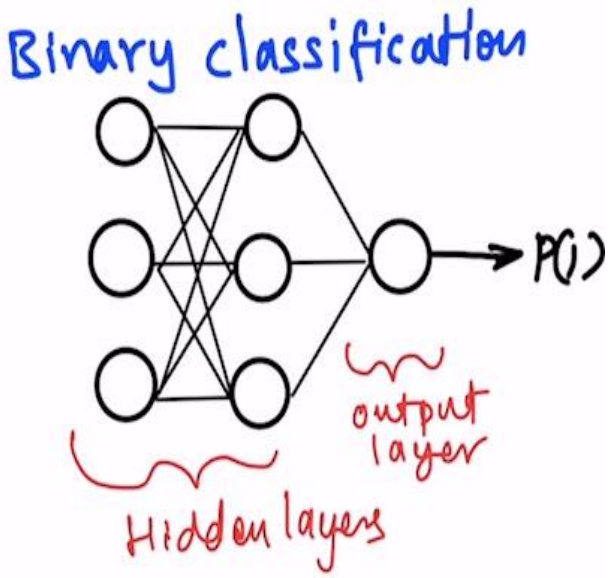
$$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$$

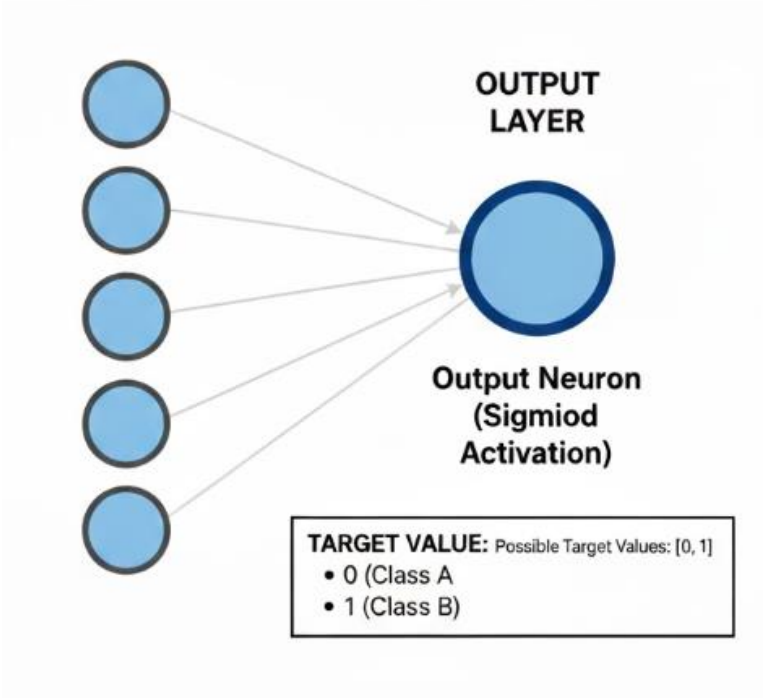
Réseaux de neurones artificiels

L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

$$Err_j = O_j(1 - O_j)(T_j - O_j);$$

Class	Target vector
Class A	0
Class B	1



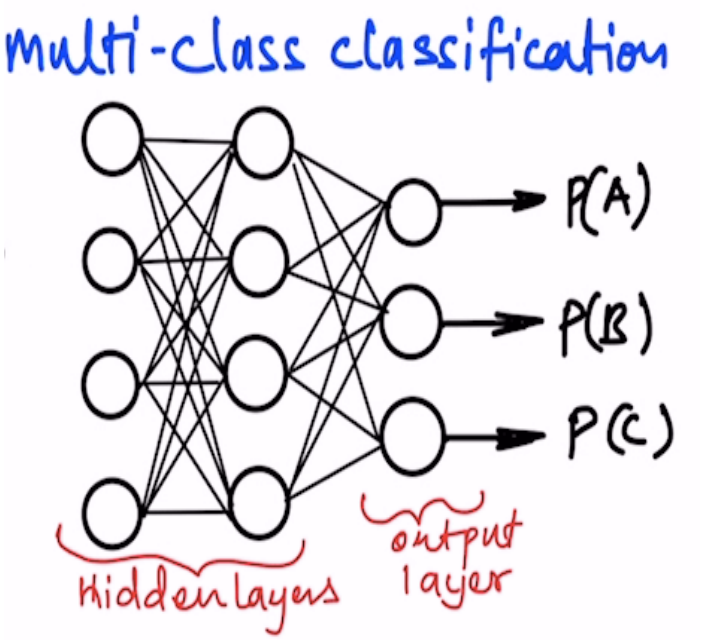


Réseaux de neurones artificiels

L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

Class	Target Value
Class A	[1, 0, 0]
Class B	[0, 1, 0]
Class C	[0, 0, 1]



Class	Value	One-Hot Encoding
0		[1, 0, 0]
1		[0, 1, 0]
2		[0, 0, 1]

L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

4 – Backpropagate Error :

a - Calcul de Transfer Derivative : *Dérivée de la Sigmoïde*

```
1 transfer_derivative = output * (1 - output)
```

b - Calcul d'erreur pour chaque neurone : *Signal Error - Delta*

-Cas - Neurone couche sorties : *Dérivée de la MSE*

```
2 error = (expected_target - output) * transfer_derivative(output)
```

-**Cas** – Neurone couche cachée : weight et error des neurones précédents

```
3 error = transfer_derivative(output) * somme(weight_k * error_k)
```

$$Err_j = O_j(1 - O_j)(T_j - O_j);$$

$$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk};$$

L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

4 – Backpropagate **Error** :

- Le calcul des erreurs est représenté par une fonction qu'on appelle **fonction de perte** : **Loss** function. Loss est l'erreur pour un seul exemple.
- La fonction de **coût** mesure l'ensemble des erreurs commises par le modèle.
- Il s'agit d'une mesure de performance sur la façon dont le ANN parvient à atteindre son objectif de générer des sorties aussi proches que possible des valeurs souhaitées (Expected Target).
- Avoir un **bon modèle** de classification (ANN ou autre), c'est avoir un modèle qui nous donne de **petites erreurs**, donc **une petite Fonction Coût/Perte**.
- Différentes fonctions coût/perte existent : Mean Squared Error, Mean Absolut Error, Cross Entropy, Sum of Squared Estimate of Errors, etc.

L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

4 – Backpropagate Error :

- L'objectif central de l'apprentissage est donc de trouver les paramètres du modèle (les poids wi) qui **minimisent la Fonction Coût/Perte**.
- Pour cela, on utilise généralement un **algorithme d'optimisation** permettant de converger et de mettre à jour les poids du ANN.
- L'exemple le plus courant étant l'algorithme de **Gradient Descent (Descente de Gradient).**
- Cet algorithme utilise une vitesse d'apprentissage, appelée **Learning Rate.** La valeur de cet hyperparamètre est fixée au départ.
- Bien choisir sa valeur. Car : Si la vitesse est trop lente (petite), le modèle peut mettre longtemps à être entraîné; si la vitesse est trop grande, alors la distance parcourue est trop longue et le modèle peut ne jamais converger.

L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

4 – Backpropagate Error :

c – **Ajuster** les poids (y compris les biais) selon les erreurs calculées, avec **Delta Rule** (Règle de **Widrow-Hoff**), optimisation de la fonction MSE.

```
1 weight = weight + (learning_rate * error * output)
2 Weight_bias = weight + (learning_rate * error)
```

$$\Delta w_{ij} = (l) Err_j O_i$$
$$w_{ij} = w_{ij} + \Delta w_{ij};$$

$$\Delta \theta_j = (l) Err_j;$$
$$\theta_j = \theta_j + \Delta \theta_j;$$

5 – Entrainement (Apprentissage) du réseau de neurones :

- **Itérer** et Répéter les 4 étapes précédentes pour un nombre d'itérations prédéfini, appelé : **epochs** number.

L'algorithme Backpropagation: Déconstruction des étapes de fonctionnement

5 – Entrainement (Apprentissage) du réseau de neurones :

- **Itérer** et Répéter les 4 étapes précédentes pour un nombre d'itérations prédéfini, appelé : **epochs** number.

👇 Visual timeline for 1 epoch (batch size = full dataset)

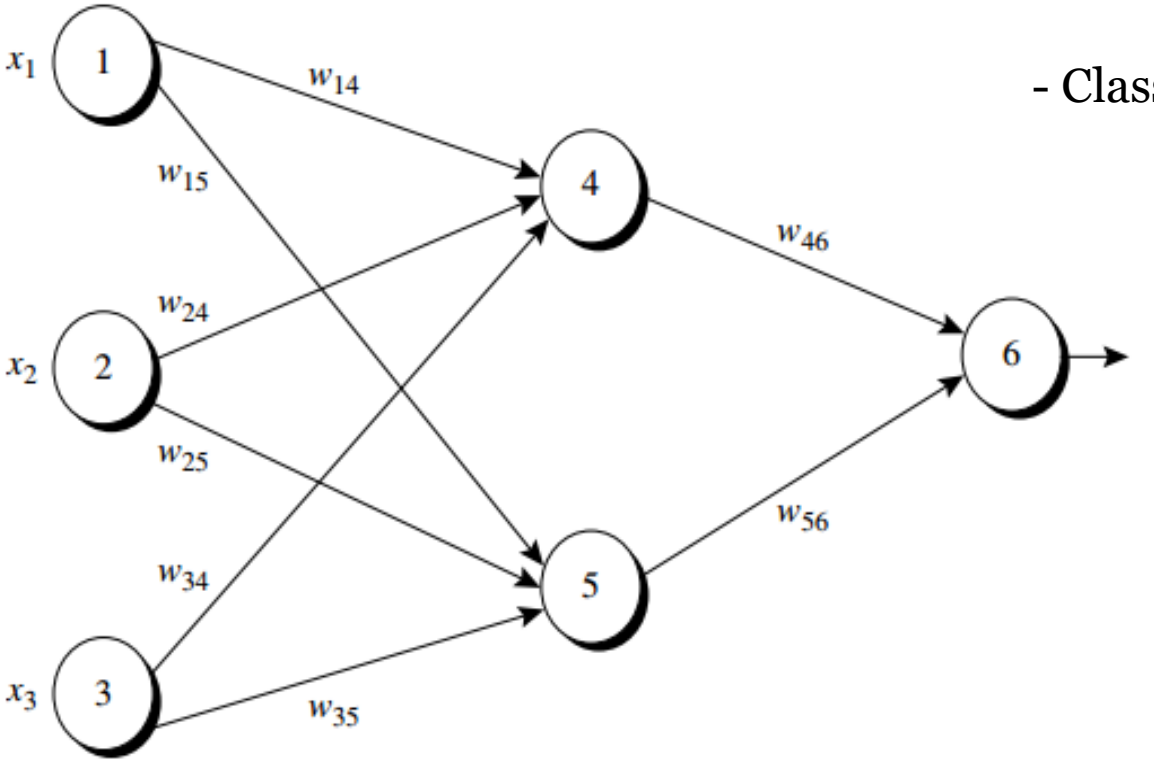
Sample	Forward	Delta	Accumulate gradient	Update weights
1	✔	✔	✔	✘
2	✔	✔	✔	✘
3	✔	✔	✔	✘
...	✔	✔	✔	✘
10	✔	✔	✔	✘
After sample 10	—	—	—	✔ (one update)

1	0	1	1

L'algorithme Backpropagation: Exemple

Learning Rate : **l = 0.9**

- Classification Binaire



Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

1	0	1	1

L'algorithme Backpropagation: Exemple

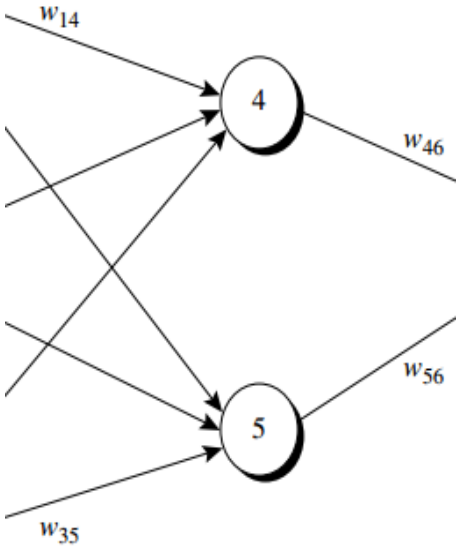
Learning Rate : l = 0.9

Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Net Input and Output Calculations

Unit, j	*Net Input, I_j*	*Output, O_j*
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$



$$I_j = \sum_i w_{ij} O_i + \theta_j;$$

$$O_j = \frac{1}{1+e^{-I_j}}$$

1	0	1	1

L'algorithme Backpropagation: Exemple

Learning Rate : l = 0.9

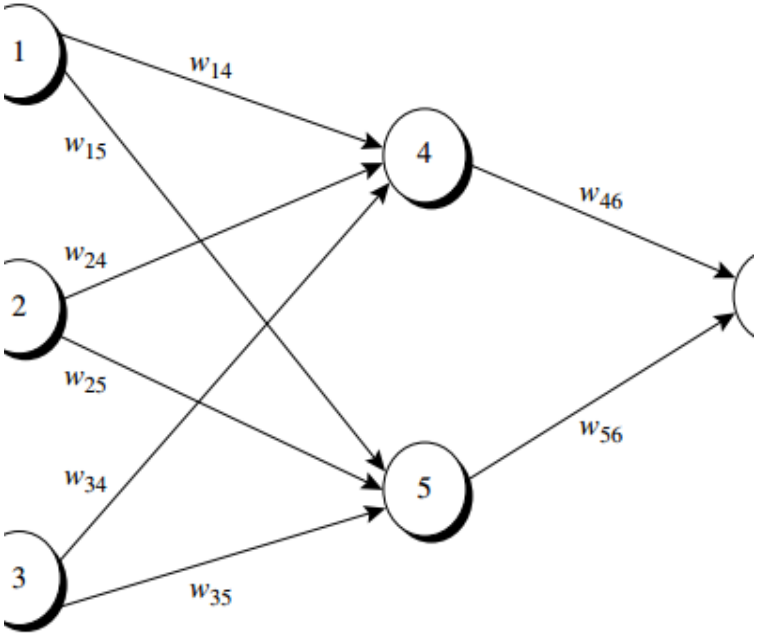
Net Input and Output Calculations

Unit, j	*Net Input, I_j*	*Output, O_j*
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Calculation of the Error at Each Node

Unit, j	*Err_j*
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$



1	0	1	1

L'algorithme Backpropagation: Exemple

Learning Rate : l = 0.9

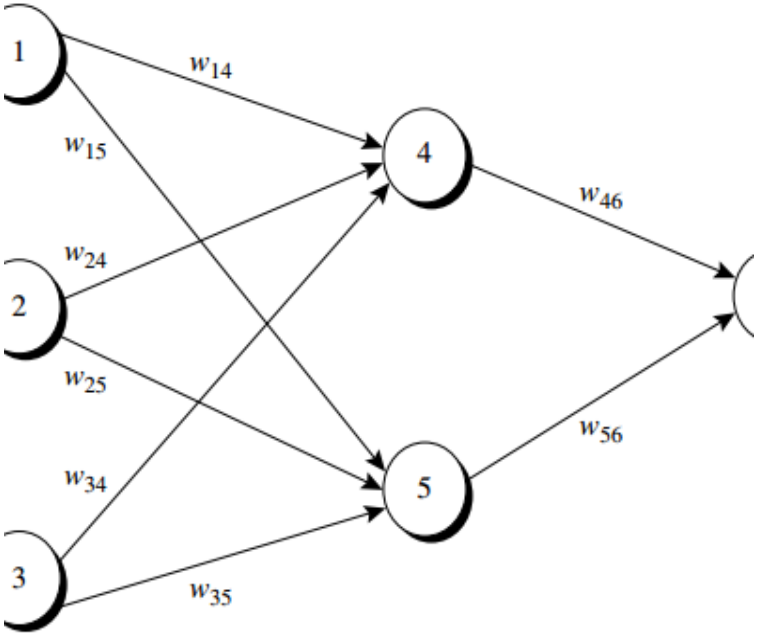
Net Input and Output Calculations

Unit, j	*Net Input, I_j*	*Output, O_j*
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Calculation of the Error at Each Node

Unit, j	*Err_j*
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk};$$



1	0	1	1

L'algorithme Backpropagation: Exemple

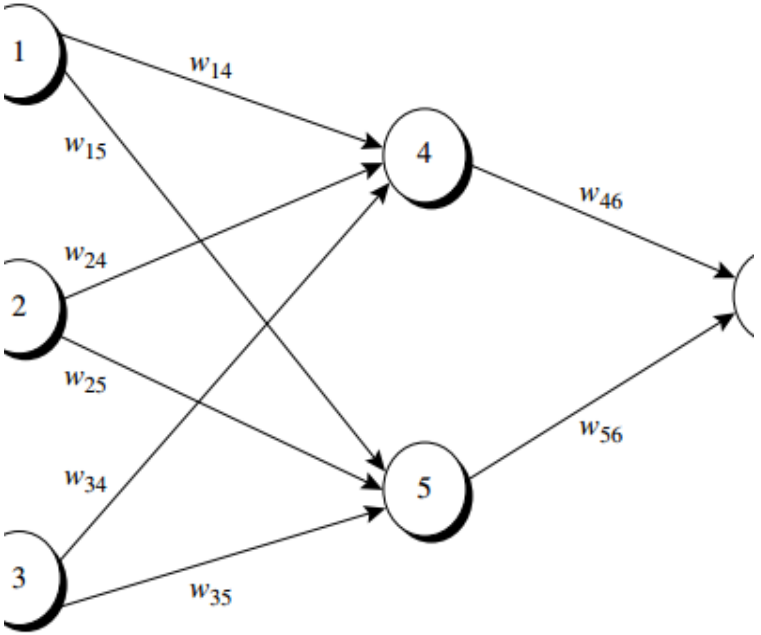
Learning Rate : l = 0.9

Calculations for Weight and Bias Updating

Weight or Bias	New Value
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

$$\Delta w_{ij} = (l) Err_j O_i$$
$$w_{ij} = w_{ij} + \Delta w_{ij};$$

$$\Delta \theta_j = (l) Err_j;$$
$$\theta_j = \theta_j + \Delta \theta_j;$$

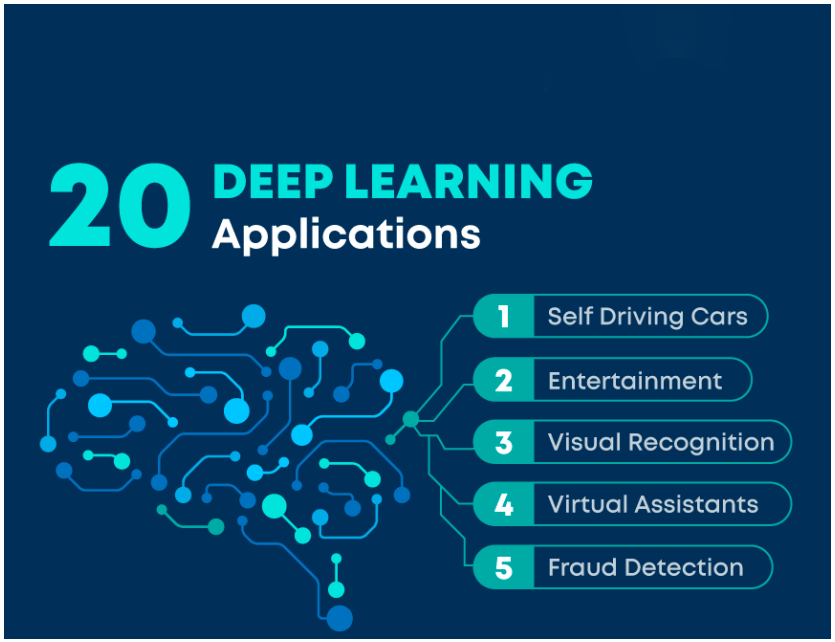


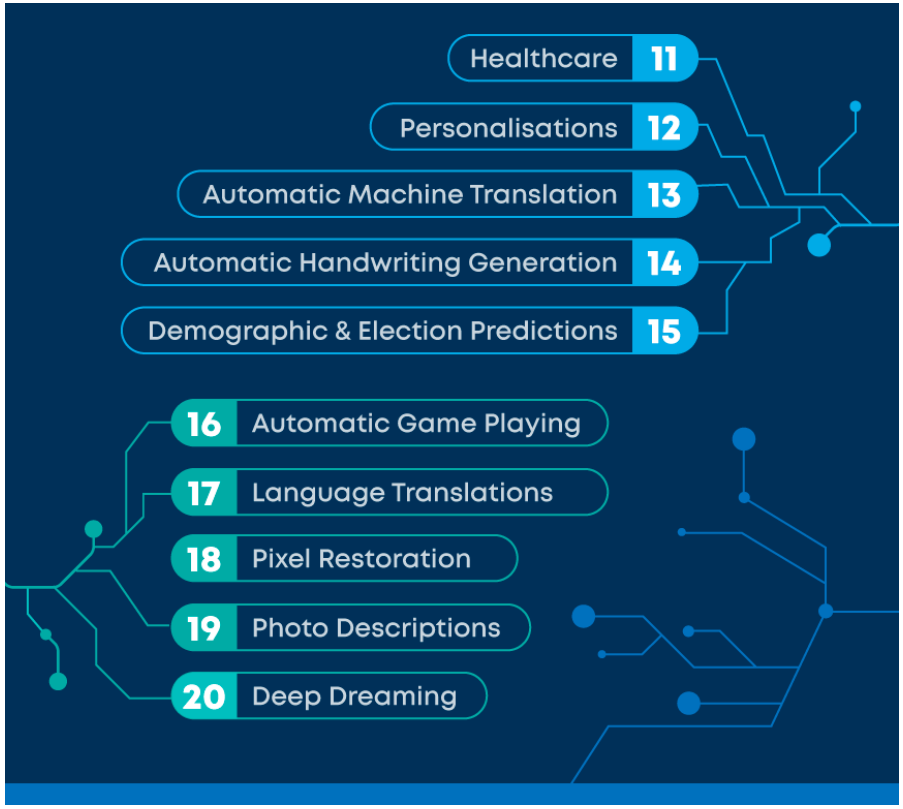
Observations

- ANN : Boite noire. Difficile d'analyser et comprendre ce qu'il a appris.
- L'ordre de présentation des exemples d'entrainement au réseau influe directement sur les résultats obtenus.
- Répéter l'apprentissage avec un ordre différent des exemples.
- Représentation de connaissance difficile à interpréter pour l'humain.

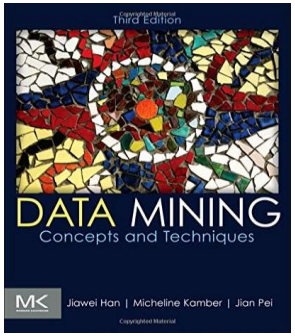
Réseaux de neurones artificiels

Domaines d'application



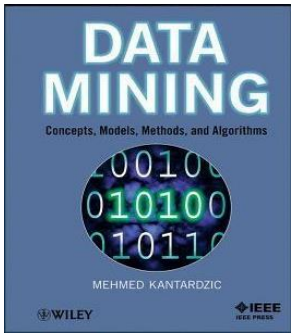


Ressources



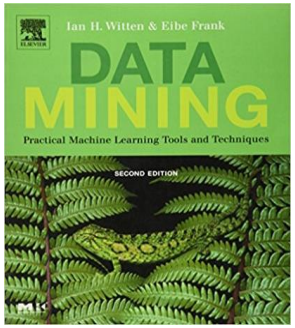
Data Mining : concepts and techniques, 3rd Edition

- ✓ Auteur : Jiawei Han, Micheline Kamber, Jian Pei
- ✓ Éditeur : Morgan Kaufmann Publishers
- ✓ Edition : Juin 2011 - 744 pages - ISBN 9780123814807



Data Mining : concepts, models, methods, and algorithms

- ✓ Auteur : Mehmed Kantardzi
- ✓ Éditeur : John Wiley & Sons
- ✓ Edition : Aout 2011 – 552 pages - ISBN : 9781118029121



Data Mining: Practical Machine Learning Tools and Techniques

- ✓ Auteur : Ian H. Witten & Eibe Frank
- ✓ Éditeur : Morgan Kaufmann Publishers
- ✓ Edition : Juin 2005 - 664 pages - ISBN : 0-12-088407-0

Références

Cours – Abdelhamid DJEFFAL – Fouille de données avancée

- ✓ www.abdelhamid-djeffal.net

WekaMOOC – Ian Witten – Data Mining with Weka

- ✓ https://www.youtube.com/user/WekaMOOC/featured

Cours - PJE : Analyse de comportements avec Twitter Classification supervisée - Arnaud Liefooghe

- ✓ http://www.fil.univ-lille1.fr/~liefooghe/PJE/bayes-cours.pdf

- ✓ https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/