



## 16 Introduction aux réseaux de neurones

### 16.1 Introduction

#### 16.1.1 Bref historique

Dans les années 1940, les chercheurs tentent de fabriquer une machine capable d'apprendre à partir de données fournies, de mémoriser des informations et de traiter des informations incomplètes. Pour cela, ils essayent de réaliser des modèles mathématiques de neurones biologiques. S'en suit alors la naissance des premiers modèles de perceptrons.

L'apprentissage automatisé va alors connaître des hauts et des bas, au gré des avancées scientifiques et technologiques. Dans les années 1960, un des premiers coups d'arrêt fût provoqué par la non-capacité des réseaux de neurones à traiter des problèmes non linéaires. Dans les années 1980, la rétropropagation du gradient fut proposée. Mais devant le manque de capacité des ordinateurs, la recherche marqua un second coup d'arrêt. Dans les années 1990/2000, l'apparition des réseaux convolutifs et leur capacité à analyser les données des images relança alors les recherches dans ce domaine.

#### 16.1.2 Exemples d'applications des réseaux de neurones

<https://fr.wikiversity.org/>

- ▶ Banque : prêts et scoring.
- ▶ Cartes de crédit : détection des fraudes.
- ▶ Finance : analyse d'investissements et de fluctuations des taux de change.
- ▶ Assurance : couverture assurantielle et estimation des réserves.
- ▶ Marketing : ciblage des prospections, mesures et comparaisons des campagnes et des méthodes.
- ▶ Archéologie : identification et datation de fossiles et d'ossements.
- ▶ Défense : identification de cibles.
- ▶ Environnement : prévisions de la qualité de l'air et de l'eau.
- ▶ Production : contrôles qualité.
- ▶ Médecine : diagnostics médicaux.
- ▶ Energies : estimations des réserves, prévisions de prix.
- ▶ Pharmacie : efficacité de nouveaux médicaments.
- ▶ Psychologie : prévisions comportementales.
- ▶ Immobilier : études de marchés.

- ▶ Analyser les principes d'intelligence artificielle.

16.1	Introduction	1
16.2	Le réseau des neurones (couches d'entrée, cachées et de sortie, neurones, biais, poids et fonction d'activation)	2
16.3	Réseaux de neurones	3
16.4	Pour aller plus loin...	9

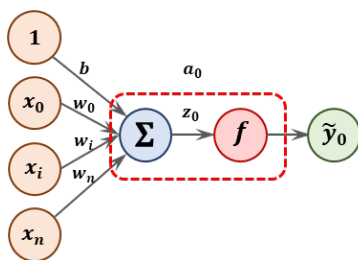
- ▶ Résoudre un problème en utilisant une solution d'intelligence artificielle :

- Apprentissage supervisé.
- Choix des données d'apprentissage.
- Mise en œuvre des algorithmes (réseaux de neurones).
- Phases d'apprentissage et d'inférence.

- Recherche scientifique : identification de spécimens, séquençages de protéines.
- Télécommunication : détection des pannes de réseaux.
- Transport : maintenance des voies.

## 16.2 Le neurone, les réseaux de neurones

### 16.2.1 Modèle de neurone



#### Définition – Neurone (ou perceptron)

Prenons la représentation suivante pour un neurone. On note :

- $\mathbf{X}$  le vecteur d'entrée constitué des données  $x_i$ . Ce vecteur constitue la couche d'entrée ;
- $w_i$  les poids (poids synaptiques) ;
- $b$  le biais ;
- $z_0$  la somme pondérée des entrées ;
- $f$  une fonction d'activation ;
- $\tilde{y}_0$  : la valeur de sortie du neurone.

On a donc, dans un premier temps :

$$z_0 = b + \sum_{i=0}^n w_i x_i.$$

Après la fonction d'activation, on a donc en sortie du neurone :


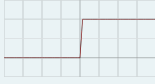


$$\tilde{y}_0 = a_0 = f(z_0) = f\left(b + \sum_{i=0}^n w_i x_i\right).$$

#### Remarque

1. La notation tilde ( $\tilde{y}_0$ ) vient du fait que la valeur de sortie d'une neurone est une valeur estimée qu'il faudra comparer à  $y_0$  valeur de l'étiquette utilisée pour l'apprentissage supervisé.
2. Par la suite, dans la représentation graphique on ne fera apparaître ni la somme pondérée ni la fonction d'activation, mais seulement la valeur de sortie du neurone (notée par exemple  $a_0$ ).

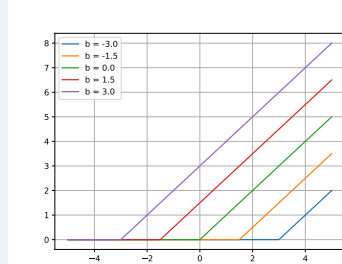
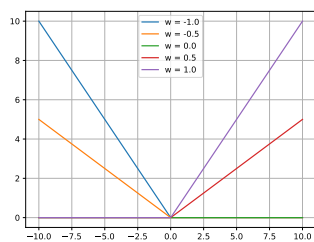
#### Définition – Fonction d'activation

Les fonctions d'activation sont des fonctions mathématiques appliquées au signal de sortie ( $z$ ). Il est alors possible d'ajouter des non linéarités à la somme pondérée. On donne ci-dessous quelques fonctions usuelles :

Identité	Heaviside	Logistique (sigmoïde)	Unité de rectification linéaire (ReLU)
			
$f(x) = x$	$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$	$f(x) = \frac{1}{1 + e^{-x}}$	$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$

### Remarque

Influence des poids et des biais sur la sortie du perceptron en utilisant une fonction d'activation ReLU.

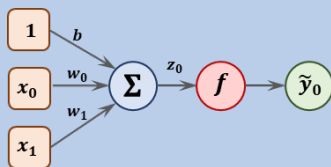


On peut ainsi voir qu'avec la fonction d'activation ReLU, plus le poids sera grand en valeur absolue, plus le neurone amplifiera le signal d'entrée.

Le biais permettra de prendre en compte le « niveau » du signal d'entrée à partir duquel, le signal doit être amplifié, ou non.

### Exemple –

Prenons un neurone à deux entrées binaires. Initialisation les On peut donc évaluer l'ensemble des sorties poids et le biais avec des valeurs calculable par le neurone.  
aléatoires :  $w_0 = -0,3$ ,  $w_1 = 0,8$   
et  $b = 0,2$ .



$x_0$	$x_1$	$z$	Id.	H.	Sig.	ReLu
0	0	0,2	0,2	1	0.549	0,2
0	1	1	1	1	0.731	1
1	0	-0.1	-0.1	0	0.475	0
1	1	0.7	0.7	1	0.668	0.7

## 16.3 Réseaux de neurones

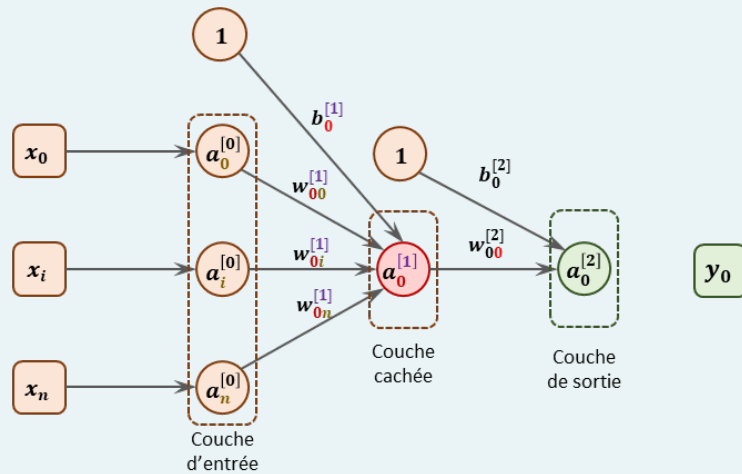
<https://playground.tensorflow.org/>

### 16.3.1 Modélisation d'un réseau de neurones

#### Définition – Couches

Un réseau de neurones est un ensemble de neurones reliés, par couches, entre eux. Dans un réseau de neurones **dense** tous les neurones de la couche  $i$  sont reliés à tous les neurones de la couche  $i + 1$ .

- Couche d'entrée : cette couche est une copie de l'ensemble des données d'entrées. Le nombre de neurones de cette couche correspond donc au nombre de données d'entrées. On note  $\mathbf{X} = (x_0, \dots, x_n)$  le vecteur d'entrées.
- Couche cachée (ou couche intermédiaire) : il s'agit d'une couche qui a une utilité intrinsèque au réseau de neurones. Ajouter des neurones dans cette couche (ou ces couches) permet donc d'ajouter de nouveaux paramètres. Pour une couche, la même fonction d'activation est utilisée pour tous les neurones. En revanche la fonction d'activation utilisée peut être différente pour deux couches différentes. Les fonctions d'activations des couches intermédiaires sont souvent non linéaires.
- Couche de sortie : le nombre de neurones de cette couche correspond au nombre de sorties attendues. La fonction d'activation de la couche de sortie est souvent linéaire. On note  $\mathbf{Y} = (y_0, \dots, y_p)$  le vecteur des sorties.



En utilisant la loi de comportement du modèle de perceptron, on peut donc exprimer  $\mathbf{Y} = \mathcal{F}(\mathbf{X})$  où  $\mathcal{F}$  est une fonction dépendant des entrées, des poids et des biais.

Notations :

- on note  $w_{jk}^{[\ell]}$  les poids permettant d'aller vers la couche  $\ell$  depuis le neurone  $k$  vers le neurone  $j$  ;
- $b_j^{[\ell]}$  le biais permettant d'aller sur le neurone  $j$  de la couche  $\ell$  ;
- $f^{[\ell]}$  la fonction d'activation de la couche  $\ell$  ;
- $n^{[\ell]}$  le nombre de neurones de la couche  $\ell$ .

#### Définition – Équation de propagation

Pour chacun des neurones  $a_j^{[\ell]}$  on peut donc écrire l'équation de propagation qui

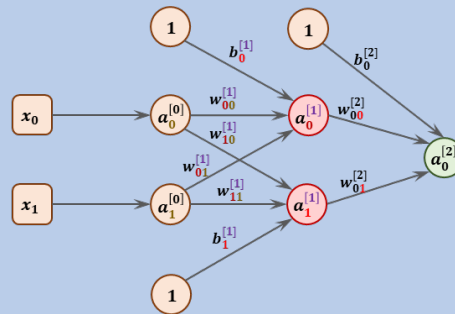
lui est associé :

$$a_j^{[\ell]} = f^{[\ell]} \left( \sum_{k=0}^{n^{[\ell-1]}} (w_{jk}^{[\ell]} a_k^{[\ell-1]}) + b_j^{[\ell]} \right) = f^{[\ell]} (z_j^{[\ell]}).$$

### Exemple –

Prenons un réseau de neurones à 3 couches :

- 1 couche d'entrée à 2 neurones ;
- 1 couche cachée à 2 neurones, de fonction d'activation  $f_1$  ;
- 1 couche de sortie à 1 neurone, de fonction d'activation  $f_2$  ;



Initialisation les poids et le biais avec des valeurs aléatoires :  $w_0 = -0,3$ ,  $w_1 = 0,8$  et  $b = 0,2$ .

Il est possible d'écrire que  $y_0 = a_0^{[2]} = f_2 \left( b_0^{[2]} + w_{00}^{[2]} a_0^{[1]} + w_{01}^{[2]} a_1^{[1]} \right)$ .

Par ailleurs :  $a_0^{[1]} = f_1 \left( b_0^{[1]} + w_{00}^{[1]} a_0^{[0]} + w_{01}^{[1]} a_1^{[0]} \right)$  et  $a_1^{[1]} = f_1 \left( b_1^{[1]} + w_{10}^{[1]} a_0^{[0]} + w_{11}^{[1]} a_1^{[0]} \right)$ .

Au final, on a donc

$$y_0 = a_0^{[1]} = f_2 \left( b_0^{[2]} + w_{00}^{[2]} \left( f_1 \left( b_0^{[1]} + w_{00}^{[1]} a_0^{[0]} + w_{01}^{[1]} a_1^{[0]} \right) \right) + w_{01}^{[2]} \left( f_1 \left( b_1^{[1]} + w_{10}^{[1]} a_0^{[0]} + w_{11}^{[1]} a_1^{[0]} \right) \right) \right)$$

### Définition – Paramètres

Les paramètres du réseau de neurones sont les poids et les biais, autant de valeurs que l'entraînement devra déterminer.

### Méthode –

Calcul du nombre de paramètres – à vérifier

Soit un jeu de données étiquetées avec  $n$  entrées et  $p$  sorties.

On construit un réseau possédant  $\ell$  couches et  $a_\ell$  le nombre de neurones de la couche  $\ell$ . Dans ce cas, la première couche est la couche d'entrée ( $a_1 = n$ ) et la dernière couche est la couche de sortie ( $a_\ell = p$ ).

**Nombre de poids :**  $n_w = \sum_{i=1}^{\ell-1} (a_i \times a_{i+1})$ .

**Nombre de biais :**  $n_b = \sum_{i=2}^{\ell} (a_i)$ .

Au final, le nombre total de paramètre à calculer est donné par  $N = n_w + n_b$ .

**Objectif**

Soit un jeu de données étiquetées. On note  $\mathbf{X}$  le vecteur des données d'entrées. On note  $\mathbf{Y}$  le vecteur des données de sorties. On note  $\mathbf{\hat{Y}}$  le vecteur de sortie calculé par le réseau de neurones.

L'objectif de la phase d'apprentissage du réseau de neurones est de déterminer les valeurs de l'ensemble des poids et des biais de telle sorte que l'écart entre  $\mathbf{Y}$  et  $\mathbf{\hat{Y}}$  soit minimal.

**16.3.2 Fonction de coût**

Dans le but de minimiser l'écart entre la sortie du réseau de neurones et la valeur réelle de la sortie, on utilise une fonction de coût (ou fonction de perte). Il est possible de définir plusieurs types de fonctions, notamment en fonction du type de problème à traiter (classification ou régression par exemple).

**Définition – Fonction coût régression**

Notons  $nb$  le nombre de données dans la base d'entraînement. Dans le cadre d'un problème de régression, on peut définir la fonction coût comme la moyenne des erreurs quadratique entre la valeur donnée par l'équation de propagation et la valeur de l'étiquette :

$$C = \frac{1}{nb} \sum_{i=1}^{nb} (\mathbf{Y}_i - \mathbf{\hat{Y}}_i)^2$$

**Objectif**

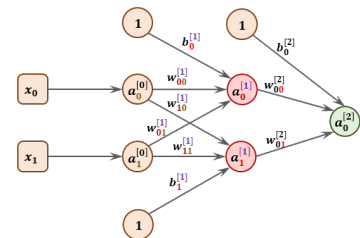
L'objectif est dès lors de déterminer les poids et les biais qui minimisent la fonction coût.

**16.3.3 Notion de rétropropagation – Descente de gradient**

En réutilisant l'exemple ci-contre, nous allons présenter succinctement comment est minimisée la fonction coût. Pour cela, il va falloir dériver la fonction coût par rapport à chacune des variables.

Cherchons uniquement à déterminer le coût par rapport à un seul vecteur d'entrée du jeu d'entraînement. On a alors :

- ▶  $C = (\mathbf{Y}_i - \mathbf{\hat{Y}}_i)^2 = (a_0^{[2]} - y)^2$  ;
- ▶  $a_0^{[2]} = f^{[2]}(z_0^{[2]})$  ;
- ▶  $z_0^{[2]} = \sum_{k=0}^1 (w_{0k}^{[2]} a_k^{[1]}) + b_0^{[2]}$ .



Commençons par déterminer la dérivée partielle par rapport à un poids de la couche de sortie :

$$\frac{\partial C}{\partial w_{00}^{[2]}} = \frac{\partial C}{\partial a_0^{[2]}} \frac{\partial a_0^{[2]}}{\partial z_0^{[2]}} \frac{\partial z_0^{[2]}}{\partial w_{00}^{[2]}}.$$

De même, on peut calculer la dérivée partielle du coût par rapport au biais :  $\frac{\partial C}{\partial b_0^{[2]}} =$

$$\frac{\partial C}{\partial a_0^{[2]}} \frac{\partial a_0^{[2]}}{\partial z_0^{[2]}} \frac{\partial z_0^{[2]}}{\partial b_0^{[2]}}.$$

Calculons les dérivées nécessaires :

$$\begin{aligned} \blacktriangleright \frac{\partial C}{\partial a_0^{[2]}} &= \blacktriangleright \frac{\partial a_0^{[2]}}{\partial z_0^{[2]}} = \blacktriangleright \frac{\partial z_0^{[2]}}{\partial w_{00}^{[2]}} = a_0^{[1]}. & \blacktriangleright \frac{\partial z_0^{[2]}}{\partial b_0^{[2]}} &= 1. \\ 2(a_0^{[2]} - y); & f'^{[2]}(z_0^{[2]}); \end{aligned}$$

On a donc,  $\frac{\partial C}{\partial w_{00}^{[2]}} = 2(a_0^{[2]} - y) f'^{[2]}(z_0^{[2]}) a_0^{[1]}$  et  $\frac{\partial C}{\partial b_0^{[2]}} = 2(a_0^{[2]} - y) f'^{[2]}(z_0^{[2]})$ .

Prenons le cas où la fonction d'activation est la fonction identité. On a alors  $\frac{\partial C}{\partial w_{00}^{[2]}} =$

$$2(a_0^{[2]} - y) a_0^{[1]} \text{ et } \frac{\partial C}{\partial b_0^{[2]}} = 2(a_0^{[2]} - y) \dots$$

On va ainsi pouvoir exprimer  $\frac{\partial C}{\partial w_{00}^{[2]}}$ ,  $\frac{\partial C}{\partial w_{01}^{[2]}}$ ,  $\frac{\partial C}{\partial b_0^{[2]}}$ , ... On pourrait ici écrire 9 équations en fonction des différents poids, des biais et des entrées  $x_0$  et  $x_1$ .

À partir de cela, on va modifier les poids comme suit :

$$\begin{aligned} \blacktriangleright w_{00,i+1}^{[2]} &= w_{00,i}^{[2]} - \eta \left. \frac{\partial C}{\partial w_{00}^{[2]}} \right|_{w_{00,i}^{[2]}} ; \\ \blacktriangleright b_{0,i+1}^{[2]} &= b_{0,i}^{[2]} - \eta \left. \frac{\partial C}{\partial b_0^{[2]}} \right|_{b_{0,i}^{[2]}} . \end{aligned}$$

On réitère ensuite les opérations précédentes jusqu'à ce que la fonction coût ait été suffisamment réduite.

### Définition – Taux d'apprentissage

On définit l'hyperparamètre  $\eta \in [0, 1[$  comme étant le taux d'apprentissage. Si ce taux d'apprentissage est très grand, l'algorithme d'apprentissage mettra beaucoup de temps à trouver le minimum. S'il est trop grand, le minimum peut ne jamais être trouvé.

### Définition – Gradient

Dés lors, dans le cas présenté, on peut définir le gradient comme le vecteur

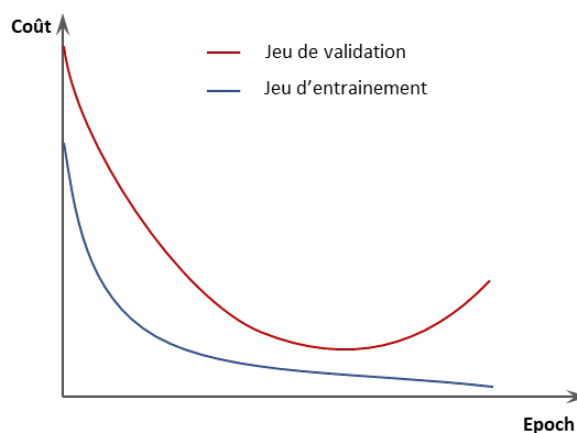
$$\text{grad}C = \begin{bmatrix} \frac{\partial C}{\partial w^{[1]}} \\ \frac{\partial C}{\partial b^{[1]}} \\ \vdots \\ \frac{\partial C}{\partial w^{[l]}} \\ \frac{\partial C}{\partial b^{[l]}} \end{bmatrix}.$$

### 16.3.4 Fin d'apprentissage

#### Définition – Epoch

On appelle *epoch* un cycle d'apprentissage où tous les poids et tous les biais ont été mis à jour en faisant passer toutes les données du jeu d'entraînement dans les algorithmes de propagation et de rétropropagation.

Les méthodes pour stopper l'apprentissage sont essentiellement empiriques. On pourrait en effet fixer un nombre d'epoch ou une valeur d'erreur admissible et s'arrêter à ce moment là. Dans la pratique, se focaliser sur l'erreur n'est généralement pas satisfaisant. En effet, il y a risque de « surapprentissage ».



Dans la figure suivante, on réalise un entraînement sur le jeu de données (une epoch) à la fin de l'epoch on dispose d'un premier modèle de réseau de neurones. On détermine alors l'erreur commise en utilisant le jeu d'entraînement puis l'erreur commise sur le jeu de validation. On calcule ensuite l'erreur commise par le modèle sur le jeu de validation. (On rappelle que le jeu de validation ne sert pas à modifier les poids et les biais.)

On réalise de même à la fin de l'epoch suivante etc...

« Logiquement » l'erreur décroît toujours avec le jeu d'entraînement car la descente du gradient a pour objectif de réduire cette erreur.

Sur le jeu de validation, l'erreur décroît pendant un certain nombre d'epoch puis augmente.

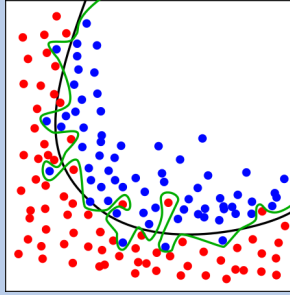
Il existe en fait un stade à partir duquel le réseau de neurones se spécialise sur le jeu d'entraînement et devient donc incapable de réaliser des prédictions fiables sur un nouveau jeu de données. On parle de surentraînement (ou d'overfitting).

#### Exemple –

Wikipedia



La ligne verte représente un modèle sur-appris et la ligne noire représente un modèle régulier. La ligne verte classe trop parfaitement les données d'entraînement, elle généralise mal et donnera de mauvaises prévisions futures avec de nouvelles données. Le modèle vert est donc au final moins bon que le noir.



## 16.4 Pour aller plus loin...

Les réseaux présentés ci-dessus sont les réseaux dits denses (ou fully-connected). On utilise d'autres types de réseaux pour une meilleure prédiction en fonction des données d'entrées :

- ▶ réseaux convolutifs (CNN) pour l'analyse d'images ;
- ▶ réseaux de neurones récurrents pour les données temporelles *etc.*

Le cours suivant présente les réseaux de type NARX et MRAC qui sont utilisés dans Matlab pour gérer les signaux temporels. Ils ont la particularité de réutiliser les sorties estimées par le réseau comme entrées décalées.