2_ Intelligence Artificielle – Réseaux de neurones

Compétences attendues :

- Analyser les principes d'intelligence artificielle. ⇆ I
- Choisir une démarche de résolution d'un problème d'ingénierie numérique ou d'intelligence artificielle. ⇆ $\boldsymbol{I}$
- Résoudre un problème en utilisant une solution d'intelligence artificielle. ⇆ $\boldsymbol{I}$

1. Introduction

1.1. Bref historique

Dans les années 40, 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 neurones. L'apprentissage automatisé va alors connaître des hauts et des bas, au gré des avancées scientifiques et technologiques. Dans les années 60, 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 80, la rétropropagation du gradient fut proposée. Mais devant le manque de capacité des ordinateurs, la recherche marquât un second coup d'arrêt. Dans les années 90/2000, l'apparition des réseaux convolutifs et leur capacité à analyser les données des images relançât alors les recherches dans ce domaine.

1.2. Exemples d'applications des réseaux de neurones

- 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.
- Archéologie : identification et datation de fossiles et d'ossements.
- Défense : identification de cibles.
- 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.
- Recherche scientifique : identification de spécimens, séquençages de protéines.
- Télécommunication : détection des pannes de réseaux.
- Transport : maintenance des voies.

2. Modèle de neurone

Définition d'un neurone (ou perceptron) :

Prenons la représentation suivante pour un neurone.

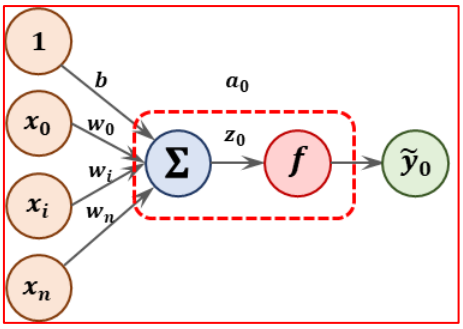
On note :

- **X** le vecteur d'entrée et x_i les données de la couche d'entrée.
- ω_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 :

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

Après la fonction d'activation, on a donc en sortie du neurone : $\tilde{y}_0 = f(z_0) = f(b + \sum_{i=0}^{n} \omega_i x_i)$



Remarque : La notation tilde ($\tilde{y}_0$) vient du fait que la valeur de sortie d'un neurone est une valeur estimée qu'il faudra comparer à y_0 valeur de l'étiquette utilisée pour l'apprentissage supervisé.

Remarque : Par la suite, dans la représentation graphique on ne fera pas apparaître la somme pondérée et la fonction d'activation, mais seulement la valeur de sortie du neurone (a_0).

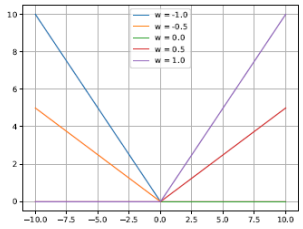
Remarque : Généralement, la fonction d'activation est choisie par le concepteur, puis les paramètres ω et b sont ajustés par une règle d'apprentissage de sorte que la relation entrée/sortie du neurone réponde à un objectif spécifique.

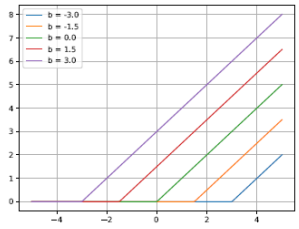
Définition d'une fonction d'activation :

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 neurone en utilisant une fonction d'activation *ReLU*.

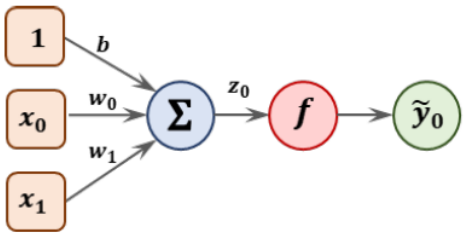




On peut ainsi voir qu'avec la fonction d'activation *ReLU*, plus le poids sera grand en valeur absolu, plus le neurone amplifiera le signal d'entrée. Le biais permettra de prendre en compte le « niveau » du signal d'entrée à partir duquel, le signal doit être amplifié, ou non.

Exemple : Prenons un neurone à deux entrées binaires.

Initialisation les poids et le biais avec des valeurs aléatoires : $\omega_0 = -0{,}3 \quad \omega_1 = 0{,}8 \;\; et \; b = 0{,}2$.



On peut donc évaluer l'ensemble des sorties calculable par le neurone.

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

3. Réseaux de neurones

3.1. Modélisation d'un réseau de neurones

Généralement, un seul neurone, même avec de nombreuses entrées, n'est pas suffisant. Il peut en falloir cinq ou dix, fonctionnant en parallèle, dans ce que l'on appelle une couche.

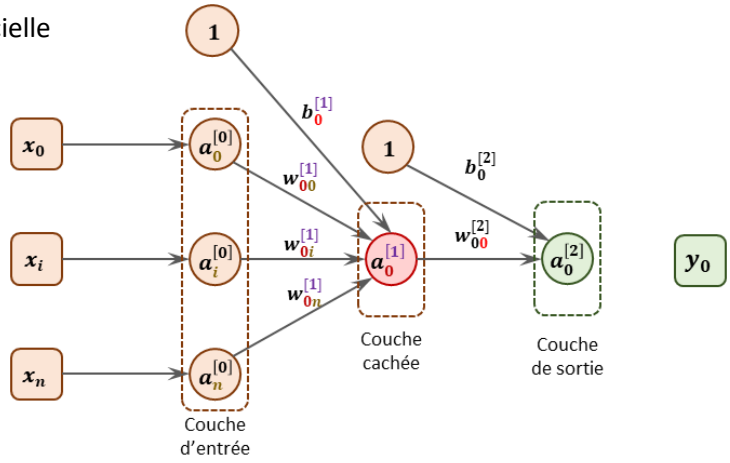
Définitions des couches :

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 seront 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 aux nombres de données d'entrées. On note **X** = $(x_0, x_1, \dots, x_n)$ 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 **Y** = $(y_0, y_1, \dots, y_m)$ le vecteur des sorties.



En utilisant la loi de comportement du modèle de neurone, on peut donc exprimer **Y** = F (**X**) où F est une fonction dépendant des entrées, des poids et des biais.

Notations :

- $\omega_{jk}^{[l]}$: poids permettant d'aller vers la couche l depuis le neurone *k* vers le neurone *j*
- $b_j^{[l]}$: biais permettant d'aller sur le neurone *j* de la couche l
- $f^{[l]}$: fonction d'activation de la couche l
- $n^{[l]}$: nombre de neurones de la couche l

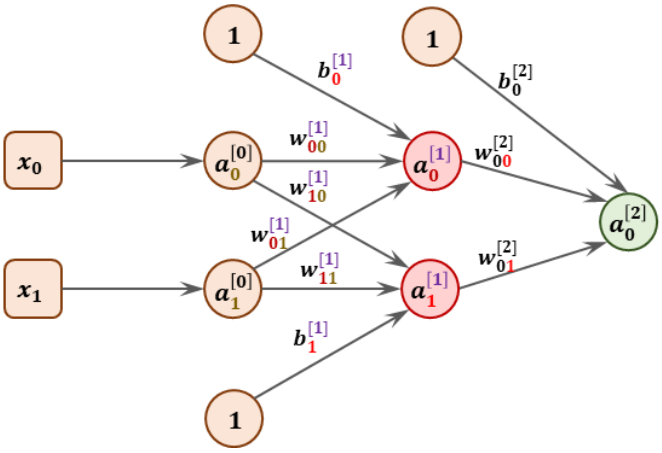
Remarque : Chaque couche a sa propre matrice de poids, son propre vecteur de biais, un vecteur d'entrée et un vecteur de sortie.

Définition de l'équation de propagation :

Pour chacun des neurones $a_j^{[l]}$ on peut donc écrire l'équation de propagation qui lui est associé :

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

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 : $\omega_0 = -0{,}3 \quad \omega_1 = 0{,}8 \;\; et \; b = 0{,}2.$

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

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

Au final, on a donc :

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

Définitions des 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 :

Soit un jeu de données étiquetées avec *n* entrées et *p* sorties.
On construit un réseau possédant l couches et a_l le nombre de neurones de la couche l.
Dans ce cas, la première couche est la couche d'entrée ($a_1 = n$) et la dernière couche et la couche de sortie ($a_l = p$).

Nombre de poids : $n_\omega = \sum_{i=1}^{l-1}(a_i.a_{i+1})$ **Nombre de bais :** $n_b = \sum_{i=2}^{l}(a_i)$

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

Objectif : Soit un jeu de données étiquetées. On note **X** le vecteur des données d'entrées. On note **Y** le vecteur des données de sorties. On note $\tilde{Y}$ 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 **Y** et $\tilde{Y}$ soit minimale.

3.2. Etapes préliminaires à l'entrainement

Un certain nombre d'étapes doivent être réalisées avant l'entrainement du réseau. Elles peuvent être regroupés en trois catégories :

- Sélection des données.
- Prétraitement des données.
- Choix du type de réseau (imposé ici par le type de problème) et de son architecture.

3.2.1. Sélection des données

Il est généralement difficile d'intégrer des connaissances préalables dans un réseau de neurones, par conséquent, la qualité du réseau sera fonction de la qualité des données utilisées pour l'entraîner. Les réseaux de neurones représentent une technologie qui est à la merci des données. Les données relatives à l'entrainement doivent couvrir l'ensemble de l'espace des entrées d'utilisation du réseau.

Une dernière question que nous devons nous poser sur la sélection des données est « *avons-nous suffisamment de données ?* »

Il est difficile de répondre à cette question, surtout avant d'entrainer le réseau. La quantité de données requises dépend de la complexité de la fonction que nous essayons d'approximer. C'est pourquoi l'ensemble du processus d'entrainement du réseau neuronal est itératif. À l'issue de l'entrainement, nous analyserons la performance du réseau. Les résultats de cette analyse peuvent nous aider à décider si nous avons suffisamment de données ou non.

3.2.2. Prétraitement des données

L'objectif principal de la phase de prétraitement des données est de faciliter l'entrainement au réseau. Le prétraitement des données comprend des étapes telles que la normalisation, les transformations non linéaires, l'extraction de caractéristiques, le traitement des données manquantes, etc… L'idée est d'effectuer des traitements des données pour faciliter l'entrainement des réseaux de neurones.

3.2.3. Choix du réseau

L'étape suivante du processus d'entrainement du réseau est le choix de l'architecture du réseau. Le type de base de l'architecture de réseau est déterminé par le type de problème que nous souhaitons résoudre. Une fois que l'architecture de base est choisie, nous devons décider de détails spécifiques tels que le nombre de neurones et de couches que l'on veut utiliser, combien de sorties le réseau devrait avoir et quel type de fonction de performance nous voulons utiliser pour l'entrainement.

3.3. Entrainement

Après la préparation des données et la mise en place de l'architecture du réseau sélectionnés, nous sommes prêts à entrainer le réseau.
Avant d'entrainer le réseau, nous devons initialiser les poids et les biais. La méthode que nous utiliserons dépendra du type de réseau. Pour les réseaux multicouches, les poids et les biais sont généralement fixés à de petites valeurs aléatoires (par exemple, répartis uniformément entre -0,5 et 0,5).
L'entrainement des réseaux de neurones est un processus itératif. Même après convergence de l'algorithme, l'analyse post-entrainement peut suggérer que le réseau soit modifié et/ou de nouveau entrainé. En outre, plusieurs cycles d'entrainement doivent être effectués pour chaque réseau potentiel afin de s'assurer qu'un minimum global ait été atteint.

3.4. Fonction de coût (*cost function* ou de perte, *loss function*)

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 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 de la fonction coût :

Notons n_b 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 :

$$C = \frac{1}{n_b} \sum_{i=1}^{n_b} (\tilde{Y}_i - Y_i)^2$$

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

3.5. Fin d'apprentissage

Définition de l'epoch :

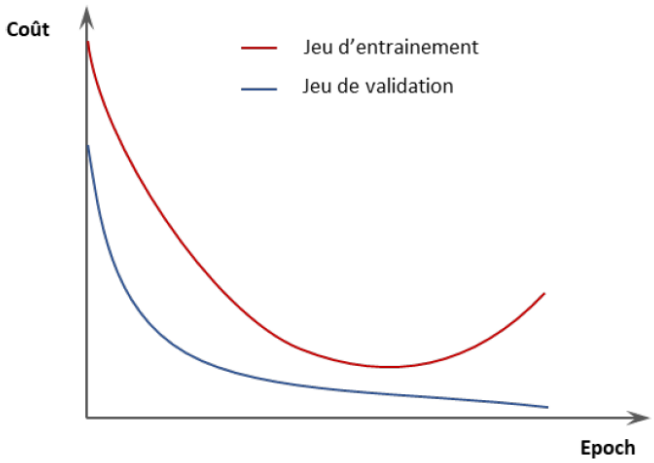
On appelle epoch, un cycle d'apprentissage où tous les poids et tous les biais ont été mis à jour en faisant passer toutes les données du jeu d'entraînement dans les algorithmes de propagation et de rétropropagation.

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 « suraprentissage ».

Dans la figure ci-contre, on réalise un entraînement sur le jeu de données. A 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.)*



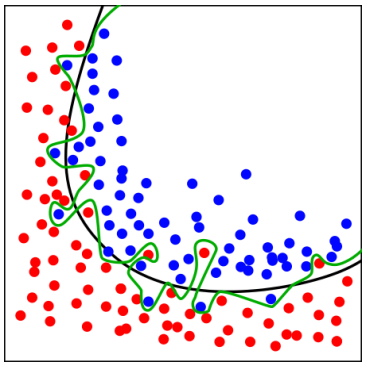
On réalise de même à la fin de l'epoch suivante etc...

« Logiquement » l'erreur décroît toujours avec le jeu d'entraînement.

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 *overfitting*).

Exemple : La ligne verte représente un modèle sur-appris et la ligne noire représente un modèle régulier. La ligne verte classifie 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.



4. Choix d'une solution pour un problème de prédiction

Notations : On suppose que l'on travaille avec des entrées $x \in \mathbb{R}^d$, des sorties $y \in \mathbb{R}^p$ et une base de données d'entraînement de n exemples.

	Modèle linéaire (régression ou classification)	k plus proches voisins	Réseaux de neurones (éventuellement profonds)
Temps de calcul à l'apprentissage	+ Inversion de matrice et calcul de $X^T X$ $\mathcal{O}(d^3 + nd^2)$ mais existence de méthodes numériques plus efficaces	++ Stockage des données de la base de donnée d'entraînement	- - Procédure longue et coûteuse notamment sur de grandes bases de données
Temps de calcul à la prédiction	++ 1 Produit matrice vecteur $\mathcal{O}(d \times p)$	- Tri d'un tableau de distances $\mathcal{O}(n \log(n))$	+ Séquence de produits matrice vecteur
Performances avec peu de données d'entrainement	++ Peu de paramètres à apprendre → bonnes performances avec peu de données	+ Performances correctes avec peu de voisins	+ Peu de paramètres par couche conduisent à un modèle proche du linéaire
Performances avec données abondantes	- Peu de paramètres apprenables	+ Réglage du nombre de voisins	++ Grande flexibilité de la fonction de prédiction
Performance avec données complexes (image / texte)	- Représentation des entrées inadaptable efficacement	- Difficile de choisir une distance correcte	++ Possibilité d'adapter les couches cachées aux propriétés des entrées

5. Pour aller plus loin…

Les réseaux présentés ci-dessus sont les réseaux dits denses (ou *fully-connected*). On utilise d'autre 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..

Références :

M.T. Hagan, H.B. Demuth, M.H. Beale et O. De Jesús : Neural Network Design. Martin Hagan, 2014. ISBN 9780971732117.

Martin T Hagan, Howard B Demuth et Orlando De Jesús : An introduction to the use of neural networks in control systems. International Journal of Robust and Nonlinear Control : IFAC-Affiliated Journal, 12 (11):959–985, 2002.

Kurt Hornik, Maxwell Stinchcombe et Halbert White : Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

Derrick Nguyen et Bernard Widrow : Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In 1990 IJCNN International Joint Conference on Neural Networks, pages 21–26. IEEE, 1990.

George EP Box, Gwilym M Jenkins, Gregory C Reinsel et Greta M Ljung : Time series analysis : forecasting and control. John Wiley & Sons, 2015.