



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 marquât un second coup d'arrêt. Dans les années 1990/2000, l'apparition des réseaux convolutifs et leur capacité à analyser les données des images relançât 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.

- Phases d'apprentissage et d'inférence.
- Réseaux de neurones (couches d'entrée, cachées et de sortie, neurones, biais, poids et fonction d'activation).

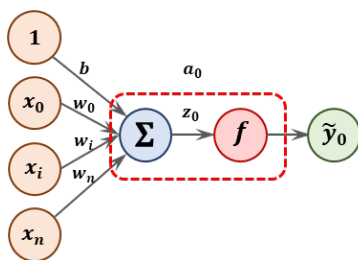
► Résoudre un problème en utilisant une solution d'intelligence artificielle :

- 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 :

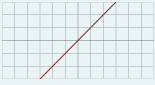
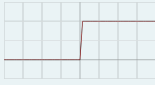


$$\tilde{y}_0 = a_0 = f(z_0) = f\left(b + \sum_{i=0}^n w_i x_i\right).$$

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).

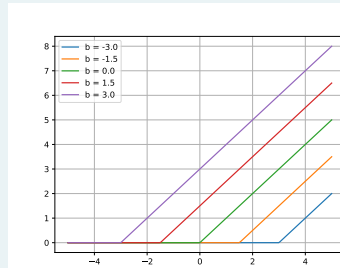
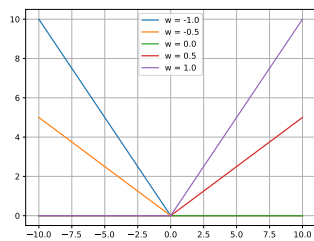
Définition – Fonction d'activation

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.

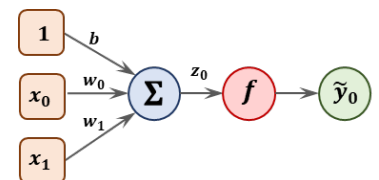
Le biais permettra de prendre en compte le « niveau » du signal d'entrée à partir duquel, le signal doit être amplifié, ou non.

Exemple –

Prenons un neurone à deux entrées binaires. Initialisation les poids et le biais avec des valeurs aléatoires : $w_0 = -0,3$, $w_1 = 0,8$ 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



16.3 Réseaux de neurones

16.3.1 Modélisation d'un réseau de neurones

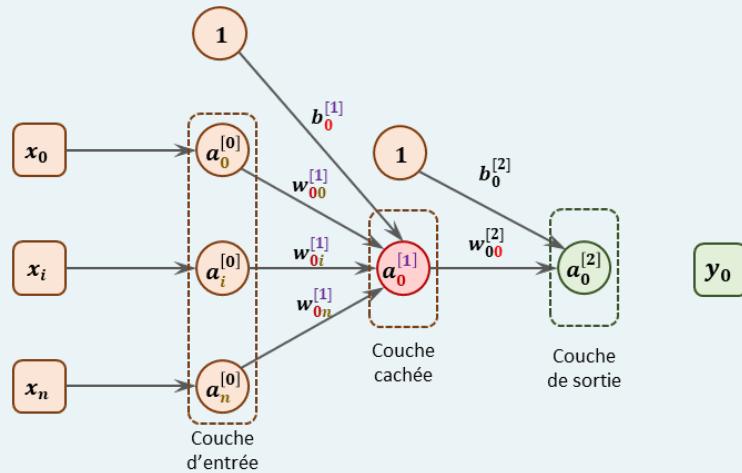
Définition – Couches

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 aux nombre de données d'entrées. On note $\mathbf{X} = (x_0, \dots, x_n)$ le vecteur d'entrées.

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

- 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.

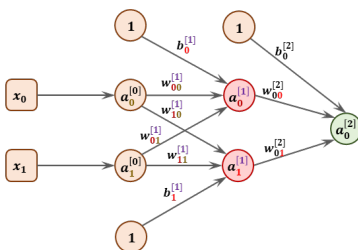
Notations :

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

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

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

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 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 ($a_\ell = p$).

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 $\hat{\mathbf{Y}}$ 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 $\hat{\mathbf{Y}}$ 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

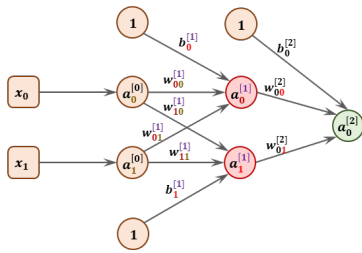
Notons nb le nombre de données dans la base d'entraînement. Dans le cadre d'un problème de régression, on peut définir la fonction coût comme la moyenne des erreurs quadratique entre la valeur donnée par l'équation de propagation et la

valeur de l'étiquette :

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

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 :

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

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 :

$$\begin{aligned} \text{► } \frac{\partial C}{\partial a_0^{[2]}} &= \text{► } \frac{\partial a_0^{[2]}}{\partial z_0^{[2]}} = \text{► } \frac{\partial z_0^{[2]}}{\partial w_{00}^{[2]}} = a_0^{[1]}. & \text{► } \frac{\partial z_0^{[2]}}{\partial b_0^{[2]}} = 1. \end{aligned}$$

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

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

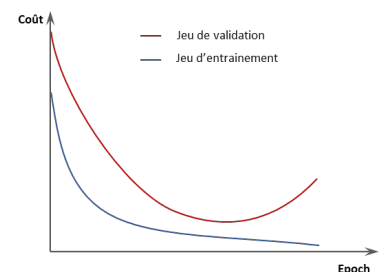
On appelle *epoch* un cycle d'apprentissage où tous les poids et tous les biais ont été mis à jour en faisant passer toutes les données du jeu d'entraînement dans les algorithmes de propagation et de rétropropagation.

Les méthodes pour stopper l'apprentissage sont essentiellement empiriques. On pourrait en effet fixer un nombre d'époch 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'époch 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.)

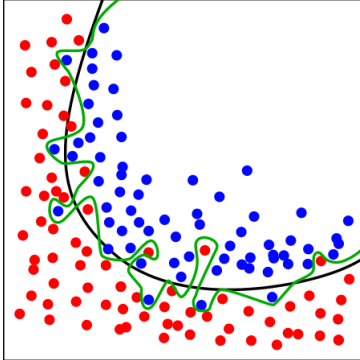
On réalise de même à la fin de l'époch suivante etc...

« 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'époch 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.

TD 1

Contrôle du correcteur de facteur de puissance par Intelligence Artificielle ★ – Sujet

CCINP – PSI – 2024 – Modélisation.

Le correcteur de facteur de puissance est utilisé pour minimiser les pertes en lignes engendrées par l'installation électrique d'un petit studio.

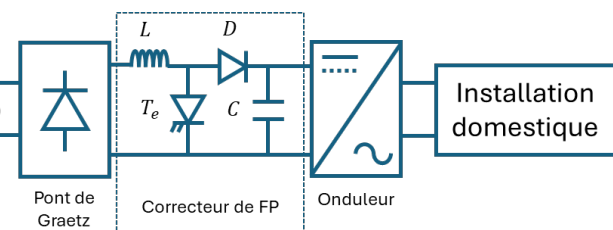


FIGURE 16.1 – Correcteur de facteur de puissance utilisé pour une installation domestique

A3-08

C3-03

La figure 16.1 présente le schéma de principe du dispositif complet : la tension aux bornes du condensateur C est placée en entrée de l'association d'un onduleur de tension et de l'installation électrique du studio. Une boucle de régulation permet de réguler cette tension à une valeur constante.

L'installation étudiée comporte un ensemble box internet-télévision, un système d'éclairage, un radiateur électrique, un chauffe-eau.

Contrôle du paramètre Δ avec un réseau de neurones

Il faut mettre en place un contrôleur capable de fournir la consigne Δ_{opt}^1 en fonction des différentes combinaisons (x_1, x_2, x_3, x_4) . Le contrôleur associé au correcteur de facteur de puissance doit permettre en temps réel de :

- ▶ soit de désactiver le correcteur de facteur de puissance en connectant l'installation électrique du studio au réseau électrique (cas $\Delta = -1^2$);
- ▶ soit de fixer de la valeur du paramètre Δ lorsque le correcteur de facteur de puissance est activé.

Le tableau ??³ nous donne la correspondance entre les états des appareils x_i^4 et le réglage Δ du correcteur. Les données d'entraînement sont celles qui vont servir à l'entraînement du réseau de neurones. Les données de test sont celles qui vont permettre de valider si l'entraînement s'est bien effectué. D'un point de vue système, cela revient à chercher la relation entre les données d'entrées x_i et la sortie Δ (figure ??).

L'objectif est de trouver un modèle de comportement qui permet, à partir du tableau ?? de retrouver les valeurs d'entraînement et de prédire les valeurs de tests.

1: A VERIFIER

2: a vérifier

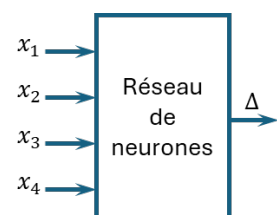


FIGURE 16.2 – Entrées-sortie du réseau de neurones

3: à récup

4: verif

Pour y arriver, l'idée est d'entraîner un réseau de neurones de manière supervisée à partir des données du tableau ??.

Question 1 Quels sont les avantages et les inconvénients du choix d'un réseau de neurones pour modéliser la commande ?

Question 2 L'apprentissage supervisé est-il pertinent ? Justifier votre réponse.

Définition d'un perceptron

Le réseau de neurones choisi est un perceptron multicouche à une première couche cachée (couche d'entrée), une deuxième couche cachée, et une couche de sortie. Le système prend en entrée un vecteur à quatre composantes (x_1, x_2, x_3, x_4) . La couche d'entrée est composée de quatre neurones numérotés de 1 à 4. La deuxième couche cachée contient également quatre neurones numérotés de 1' à 4'. La couche de sortie contient un seul neurone numéroté 1''. La sortie de la couche de sortie est Δ . Chaque neurone prend en entrée les sorties de la couche précédente et renvoie une unique sortie.

La figure ?? présente cette structure.

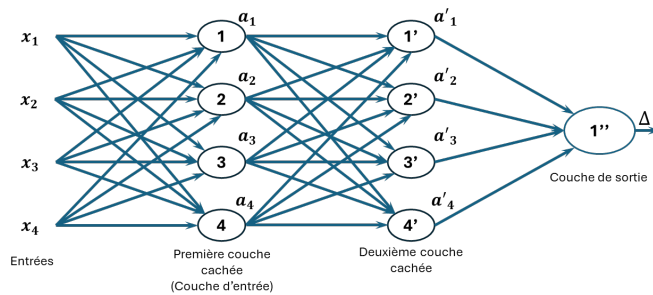


FIGURE 16.3 – Structure du réseau de neurones choisi

Le détail d'un neurone est défini par la figure ??.

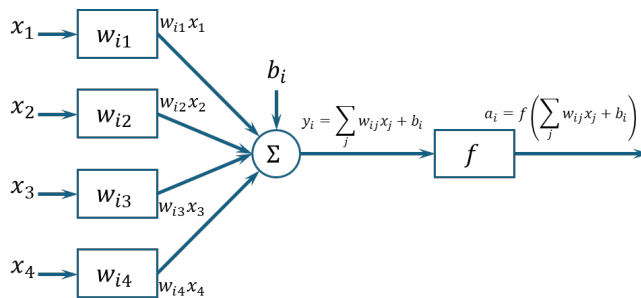


FIGURE 16.4 – Structure d'un neurone i de la première couche

La figure ?? d'un neurone numéroté « i » de la première couche permet, à partir des valeurs d'entrées (x_1, x_2, x_3, x_4) de calculer la valeur de sortie a_i . Les sorties de la première couche deviennent les entrées de la couche suivante et ainsi de suite. Tous les neurones vont se comporter comme ceux de la première couche. Il est à noter que la sortie du dernier neurone correspond à Δ .

Nous noterons :

- w_{ij} : le poids entre l'entrée j et le neurone i , c'est un nombre réel ;
- b_i : le biais du neurone i , c'est un nombre réel ;
- f : la fonction d'activation du neurone i qui s'applique à $\sum_j w_{ij} x_j + b_i$.

Le processus permettant de trouver les valeurs de b_i et de w_{ij} va être itératif. Les valeurs initiales seront choisies aléatoirement.

La fonction d'activation choisie est une sigmoïde définie par $f : \mathbb{R} \rightarrow \mathbb{R}, x \rightarrow f(x) = \frac{1}{1 + e^{-x}}$. En Python, l'utilisation de la fonction `exp` de la librairie `numpy` permet de calculer, pour une entrée `x` de type tableau, la matrice $f(x)$, issue de l'application de la fonction f à tous les éléments de la matrice `x`. Cette fonction d'activation se traduit par la fonction suivante.

```

1 def f(x) :
2     return (1/(1+np.exp(-x)))

```

Question 3 Calculer la dérivée de la fonction f notée $f'(x)$ et écrire une fonction Python notée `f_prime` qui prend en argument x et qui renvoie la dérivée $f'(x)$. L'argument d'entrée x est de type tableau.

Phase d'inférence

La phrase d'inférence consiste à calculer la sortie à partir d'un vecteur d'entrée connu. La sortie du dernier neurone du réseau est la valeur Δ recherché et peut s'écrire sous la forme suivante : $\Delta = f(W_3(f(W_2(f(W_1X + B_1))))$.

Question 4 Donner les dimensions des matrices W_3 , W_2 , B_2 et b_3 .

On cherche à modéliser la couche d'entrée du réseau de neurones de la figure ??.

Question 5 Donner l'expression de la matrice W_1 qui vérifie :

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = W_1 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}.$$

Question 6 Ecrire en Python la fonction `inference_couche` qui va prendre en argument le vecteur d'entrées noté X , la matrice des points notée W , le vecteur biais noté B et qui renvoie le vecteur des sorties de la couche noté A ⁵.

5: xxx

La figure ?? montre le tracé de la fonction d'activation sigmoïde.

XXX

FIGURE 16.5 – Représentation graphique de la fonction sigmoïde f pour $x \in [-20, 20]$ (à gauche) et $x \in [-1, 1]$ (à droite)

Question 7 Calculer $A = f(Y)$ dans le cas où : $A = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$, $Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$, $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$,

$$W = \begin{pmatrix} 0,3 & 0,2 & 0,25 & -0,03 \\ 0,4 & -0,3 & 0,6 & -0,3 \\ -0,4 & 0,3 & -0,6 & -0,5 \\ -1 & -0,75 & -0,1 & -0,5 \end{pmatrix}, B = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Notons que, pour cette première phase d'apprentissage qu'on appelle souvent l'initialisation, les valeurs des poids ont été générées aléatoirement et les biais choisis nuls. En reproduisant le même calcul pour les couches suivantes, nous calculons que $\Delta = 0,59$ V. Cela signifie que pour l'éclairage allumé et tous les appareils éteints, il faudrait un réglage de $\Delta = 0,59$ V.

Cependant le tableau ?? indique que pour cette combinaison d'états, le réglage optimal est $\Delta_{\text{opt}} = 0,18$ V.

Il faut modifier les valeurs des poids et des biais pour que la sortie Δ du réseau de neurones se rapproche de Δ_{opt} et ce pour toutes les combinaisons d'entrées possibles. Autrement dit, pour toute combinaison d'entrée (x_1, x_2, x_3, x_4) , on souhaite minimiser l'erreur (choisie quadratique ici) entre la sortie Δ calculée par le réseau de neurones

Notations :

- les grandeurs scalaires seront notées en minuscule, par exemple w ;
- les grandeurs matricielles seront notées en majuscule, par exemple W ;
- l'indice k fait référence à la couche, par exemple W_2 est pour la deuxième couche.

et le réglage optimal Δ_{opt} , déterminé à l'avance. Pour ne pas alourdir les notations on notera désormais t le réglage optimal Δ_{opt} . Le calcul d'erreur $(t - \Delta)^2$ va être utilisé pour calculer des « meilleures » valeurs de poids et de biais. C'est la phase d'entraînement qui fait partie de la partie suivante.

6: REVOIR NOTATIONS

Rétropropagation ⁶

Nous disposons de $N = 9$ données d'entraînement c'est-à-dire toutes les paires $\{X_\ell, t_\ell\}$ où $X_\ell = (x_1, x_2, x_3, x_4)$ est le vecteur de la combinaison des entrées t_ℓ est simplement la valeur optimale Δ_{opt} , évaluée empiriquement et qui constitue la valeur cible souhaitée en sortie du réseau de neurones.

Ces données d'entraînement sont recensées dans le tableau ?? . Pour chaque donnée d'entraînement $\{X_\ell, t_\ell\}$, on notera Δ_ℓ le résultat réellement renvoyé par le réseau de neurones. Le but de la phase de rétropropagation est de calculer des valeurs de paramètres W et B qui « rapprochent » tous les Δ_ℓ calculés pour les X_ℓ des valeurs optimales t_ℓ . Une instance de rétropropagation s'effectue en cherchant les valeurs des paramètres W et B qui minimisent la fonction de coût quadratique entre la cible t_ℓ et la valeur calculée $\Delta_\ell = \mathcal{L}_\ell = (t_\ell - \Delta_\ell)^2$.

Pour des raisons de lisibilité, on notera $\mathcal{L} = \mathcal{L}_\ell$.

Pour ce faire, les poids W et les biais B sont mis à jour par la méthode de la descent du gradient, étudiée dans les questions ci-après. Afin de se fixer les idées, on s'intéresse à la première couche du réseau de neurones, ayant pour entrée le vecteur X et pour sortie le vecteur d'activation $A : A = f(W_A X + B_1)$. Le fonctionnement des autres couches sera identique. Pour des raisons de lisibilité, on posera désormais $W = W_1$, $B = B_1$, $Y = WX + B$. On a donc : $A = f(WX + B) = f(Y)$.

Les valeurs des paramètres W et B de cette couche sont mis à jour grâce aux formules

$$\text{suivantes : } \begin{cases} W = W - \alpha \frac{\partial \mathcal{L}}{\partial W} \\ B = B - \alpha \frac{\partial \mathcal{L}}{\partial B} \end{cases}$$

où α est le taux d'apprentissage, un paramètre à choisir manuellement. On adopte la notation suivante :

$$\frac{\partial \mathcal{L}}{\partial W} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial w_{11}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{14}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial w_{41}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{44}} \end{pmatrix}, \frac{\partial \mathcal{L}}{\partial X} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial x_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial x_4} \end{pmatrix}, \frac{\partial \mathcal{L}}{\partial A} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial a_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial a_4} \end{pmatrix}.$$

Nous avons $\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_1} \frac{\partial a_1}{\partial w_{ij}} + \cdots + \frac{\partial \mathcal{L}}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}} + \cdots + \frac{\partial \mathcal{L}}{\partial a_4} \frac{\partial a_4}{\partial w_{ij}}$ et $a_i = f\left(\sum_j w_{ij} x_j + b_i\right)$.

Question 8 Montrer que $\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_i} f'(y_i) x_j$.

Question 9 En utilisant la relation précédente, donner la relation matricielle entre $\frac{\partial \mathcal{L}}{\partial W}$, $\frac{\partial \mathcal{L}}{\partial A}$, $f'(Y)$ et X , puis entre $\frac{\partial \mathcal{L}}{\partial W}$, $\frac{\partial \mathcal{L}}{\partial A}$, W , X et B .

En poursuivant l'analyse il est possible de trouver la relation suivante qui va permettre d'exprimer l'erreur en entrée en fonction de l'erreur en sortie : $\frac{\partial \mathcal{L}}{\partial X} =$

$$\frac{\partial \mathcal{L}}{\partial A} (WX f'(WX + B))^T.$$

7: xxxx

Il est à remarquer que dans la formule précédente, la multiplication se fait terme à terme.

Question 10 Ecrire la fonction Python `retropropagation_couche` qui prend en argument :

- ▶ la valeur d'entrée d'une couche, notée X ;
- ▶ la matrice de poids d'une couche notée W ;
- ▶ le vecteur de biais de la couche noté B ;
- ▶ le vecteur d'erreur en sortie de la couche, noté Ea ;
- ▶ la valeur du coefficient d'apprentissage, notée α .

Cette fonction doit calculer $E_W = \frac{\partial \mathcal{L}}{\partial W}$, $E_B = \frac{\partial \mathcal{L}}{\partial B}$, $E_X = \frac{\partial \mathcal{L}}{\partial X}$. Cette fonction doit renvoyer :

- ▶ la matrice de poids W mise à jour ;
- ▶ la vecteur de biais B mis à jour ;
- ▶ le vecteur gradient du coût $E_x = \frac{\partial \mathcal{L}}{\partial X}$.

Question 11

Question 12