1 Introduction à l'intelligence artificielle

Le domaine appelé « apprentissage machine » *(machine learning)* étudie des systèmes capables de **modifier leur propre comportement** jusqu'à respecter à des **contraintes données**.

1.1 Apprentissage machine : supervisé ou non ?

Définition 1.

- On a affaire à un problème d'apprentissage **supervisé** lorsque les contraintes s'expriment avec **des entrées et des sorties** toutes fournies : pour telles valeurs en entrée, on aimerait que le système produise telle valeur en sortie.
- On a affaire à un problème d'apprentissage **non supervisé** lorsque les contraintes s'expriment en fournissant **les entrées mais pas les sorties** : pour des valeurs très proches en entrée, on aimerait que le système produise une même valeur en sortie ; et pour des données très différentes en entrée, on aimerait que le système produise des valeurs différentes en sortie.

Exemple 2.

▶ *On a déjà rencontré des problèmes de régression linéaire : trouver une droite qui approche au mieux un nuage de points fournis par leurs coordonnées planes (x_i, y_i). Dans ce contexte on a considéré qu'il fallait chercher une fonction f sous la forme $f(x) = ax + b$, avec a et b à déterminer, telle que les $f(x_i)$ ressemblent le plus possible aux y_i fournis.*

*Il s'agit donc d'un problème d'apprentissage **supervisé** : le système est constitué du couple (a, b) ajustable, et prend en entrée x pour sortir $f(x)$; on lui fournit des entrées x_i et les sorties voulues correspondantes y_i.*

Ici, les entrées et les sorties sont de nature continue.

▶ *Le célèbre jeu de données sur les iris (1935) d'Edgar Anderson (1897–1969) et Ronald Fisher (1890–1962) contient 4 mesures de longueurs (variables continues), ainsi qu'un identifiant parmi 3 espèces possibles (variable qualitative) pour 50 fleurs (individus).*

*Il est accessible dans la distribution basique de **R** par la commande* `iris`*.*

*Un problème d'apprentissage **supervisé** serait le suivant : construire une procédure qui permette, à partir des 4 longueurs mesurées, de retrouver l'espèce connue (cette procédure pourra ensuite être appliquée à de nouvelles mesures pour « deviner » l'espèce même si on ne la connaît pas).*
Par exemple : règle des k plus proches voisins.

*Un problème d'apprentissage **non supervisé** serait le suivant : sans tenir compte de l'espèce, définir des groupes de fleurs tels que des fleurs d'un même groupe se ressemblent (en termes de longueur) et que des fleurs de groupes différents ne se ressemblent pas (ici, on pourrait à posteriori comparer ces groupes avec les vraies espèces).*
Par exemple : algorithme des k moyennes.

▶ *On a sondé 1000 personnes à propos de 10 activités (sport, lecture, voyage…) : pour chaque activité, chaque personne a répondu parmi 4 possibilités : j'aime beaucoup, j'aime un peu, je n'aime pas trop, je déteste (on a donc 10 variables qualitatives ordinales).*

*Trouver des groupes de personnes qui ont des goûts similaires est un problème d'apprentissage **non supervisé**.*

Peut-on définir un problème d'apprentissage non supervisé avec sortie continue ?

Définition 3.
Dans ces deux types d'apprentissage, on appelle **cas** ou **exemple** chaque situation où on fournit les entrées (cela correspond à des individus statistiques, des enregistrements d'une base de données, etc.).

Remarque 4.
- Pour les problèmes supervisés : on parle de **régression** si la variable en sortie est quantitative continue, et de **classification** ou **discrimination** si la variable en sortie est qualitative (étiquettes). Une classification peut être « dure » (on veut une unique étiquette en sortie) ou « floue » (on veut une « confiance » entre 0 et 1 pour chaque étiquette possible).
- Pour les problèmes non supervisés (étiquette en sortie à construire) : on parle aussi de classification (attention à la confusion !) mais aussi de **clustering** (il n'y a pas d'ambiguïté sur ce terme).

1.2 Subdivisions des cas : apprentissage, validation, test

Définition 5.
En apprentissage **supervisé**, il est courant de répartir les exemples fournis dans une **partition** en 3 sous-ensembles :

- Le jeu d'**apprentissage** = jeu d'**entraînement** *(training set)* :
ensemble des cas qui vont servir à ajuster les **paramètres** "à l'intérieur" du système (par exemple les coefficients a et b de la droite en régression linéaire).
- Le jeu de **validation** *(validation set)* :
ensemble des cas qui vont servir à comparer les performances du système après apprentissages dans des variantes qui diffèrent par les **hyperparamètres**, c'est-à-dire le nombre et/ou la nature des paramètres (par exemple droite des moindres carrés $y = ax + b$ vs. parabole des moindres carrés $y = ax^2 + bx + c$: voir chapitre 2).
- Le jeu de **test** *(test set)* :
ensemble des cas qui vont servir à évaluer la performance du système sélectionné après les apprentissages et la validation.

Le but est d'évaluer si le système est capable de « généraliser » ses conclusions à de nouvelles données (non rencontrées dans le jeu d'apprentissage), ou s'il s'est « trop attaché à certains détails » du jeu d'apprentissage qui ne sont que des cas particuliers (absents du jeu de test) ; dans ce dernier cas on dit qu'il y a eu **surapprentissage** *(overfitting)*.

Remarque 6.
- Il faut que ces 3 ensembles soient « similaires » : par exemple si les cas sont des instants de mesure sur plusieurs mois, il faut éviter d'apprendre sur l'hiver et de tester sur l'été…
- En général on répartit les cas au hasard dans ces 3 sous-ensembles, avec une quantité nettement plus grande dans le jeu d'apprentissage.
- Parfois on n'a pas besoin d'un jeu de validation : seulement apprentissage et test ; dans ce cas le mot validation peut être utilisé pour désigner le jeu de test.
- En apprentissage non supervisé on ne dispose pas de "vraies" réponses à comparer avec les sorties du système, donc on ne peut pas procéder ainsi.

2 Apprentissage supervisé avec perceptron multi-couches (introduction aux réseaux de neurones artificiels)

Source pour ce cours : I.U.T. de Bourg-en-Bresse.

2.1 Modèle général de neurone artificiel

Remarque 7.
Dans un cerveau humain, il y a environ cent milliards (10^{11}) de neurones. Chaque neurone se compose d'un *corps cellulaire* et de deux types de prolongements :

- les *dendrites* : jusqu'à plusieurs milliers par neurone ;
- un *axone* : unique pour chaque neurone.

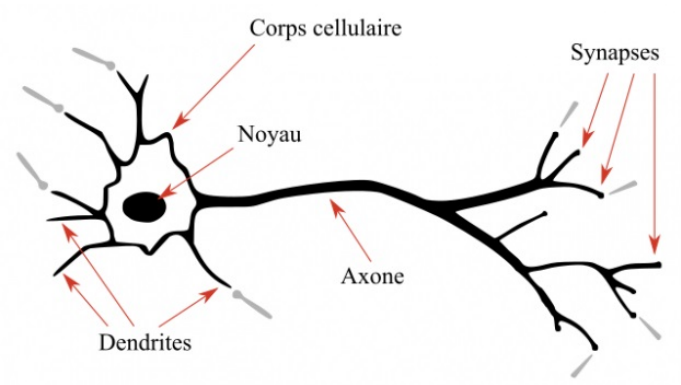


FIGURE 1 – Schéma simplifié d'un neurone biologique. ©Julien Chevallier
Image issue de http://images.math.cnrs.fr/Nos-neurones-se-synchronisent-ils.html

Un neurone reçoit des signaux (électriques, chimiques) en entrée par ses dendrites, et émet un signal de sortie par son axone. Dans ce chapitre, on s'intéresse à modéliser la **transformation** des signaux d'entrées en signal de sortie par un neurone (considéré de façon isolée).

Les connexions entre axones et dendrites se font par l'intermédiaire de *synapses*. Dans les années 1960, on a commencé à modéliser les neurones. Voici une manière de le faire :

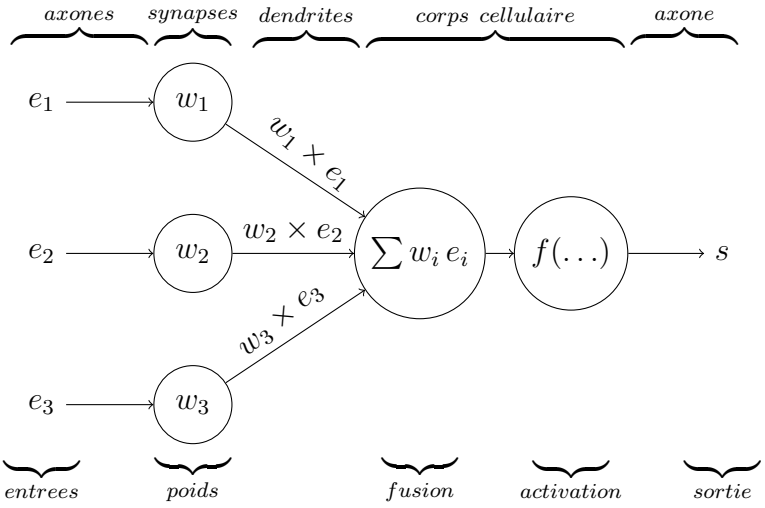


FIGURE 2 – Schéma d'un neurone artificiel.

Remarque 8 (notations)**.**
- e_i, w_i, s désignent des nombres (à priori réels), et f une **fonction d'activation**.
- Les e_i et s codent les informations transmises entre neurone(s) et environnement.
- Les w_i et f décrivent la transformation effectuée « en interne » par le neurone.
- Ces notations ne sont pratiques que pour un neurone « seul » ; dans le cas des réseaux à plusieurs neurones elles seront un peu modifiées.

Dans la suite on simplifie le schéma de la figure 2 ainsi :

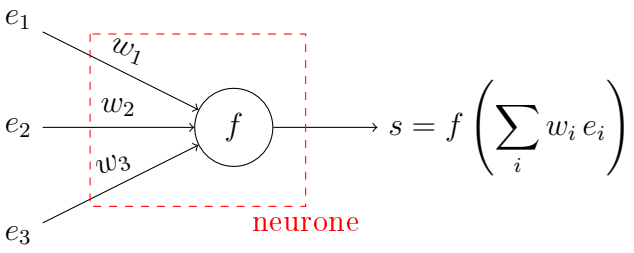


FIGURE 3 – Schéma simplifié d'un neurone artificiel.

Définition 9. Définir un neurone, c'est donc donner :
- son nombre d'entrées n ;
- ses poids synaptiques $w_1, \ldots, w_n$;
- sa fonction d'activation f.

Liste de fonctions d'activation usuelles :

Fonction de Heaviside (ou créneau) : à valeurs dans $\{0;1\}$ définie par

$$\boxed{\mathcal{H}(x) = \begin{cases} 0 & \text{si } x < 0, \\ 1 & \text{si } x \geqslant 0. \end{cases}}$$

Fonction logistique (ou sigmoïde) : à valeurs dans $]0;1[$ définie par

$$\boxed{\mathcal{S}(x) = \frac{1}{1+e^{-x}}} \quad \text{pour tout } x \in \mathbb{R}.$$

Propriété : $\mathcal{S}(x) + \mathcal{S}(-x) = 1$.

Fonction tangente hyperbolique : à valeurs dans $]-1;1[$ définie par

$$\boxed{tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}} \quad \text{pour tout } x \in \mathbb{R}.$$

Propriété : $tanh(x) = \mathcal{S}(2x) - \mathcal{S}(-2x)$ et donc $\mathcal{S}(x) = \frac{1}{2}\left[1 + tanh\left(\frac{x}{2}\right)\right]$.
Les fonctions $\mathcal{S}$ et $tanh$ sont donc très liées entre elles...

Fonction de rectification linéaire (ReLU) : à valeurs dans $\mathbb{R}_+$ définie par

$$\boxed{x^+ = max(0;x) = \begin{cases} 0 & \text{si } x < 0, \\ x & \text{si } x \geqslant 0. \end{cases}}$$

Fonction de rectification linéaire lisse : à valeurs dans $\mathbb{R}_+^*$ définie par

$$\boxed{softplus(x) = ln(1+e^x)} \quad \text{pour tout } x \in \mathbb{R}.$$

Remarque 10.
Pour simplifier, on considèrera dans toute la suite que l'architecture (nombre d'entrées et f) est fixée à l'avance et ne peut pas être modifiée.
Les « réglages internes » du neurone sont alors juste les poids w_i.

2.2 Liens avec les chapitres précédents

Remarque 11.
Les réseaux de neurones que nous étudions dans ce chapitre servent dans le cadre d'un **apprentissage supervisé** (cf chapitre 1) :

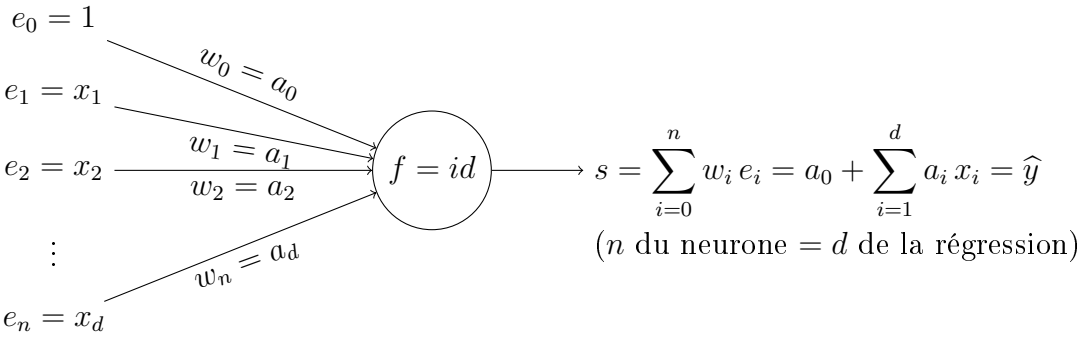
- la machine apprenante est l'ensemble des poids synaptiques w_i ;
- les entrées sont $e_1, \dots, e_n$ et la sortie est s ;
- on dispose d'un ensemble de cas sous la forme $(e_1, \dots, e_n ; r)$ où r =réponse voulue ;
- la machine doit modifier ses w_i jusqu'à ce que les sorties calculées s correspondent « le mieux possible » aux réponses voulues r.

Définition 12.
Comme déjà rencontré en régression, on utilise le critère du MSE pour mesurer les performances de la machine sur un ensemble de cas :

$$MSE = \frac{1}{K}\sum_{k=1}^{K}(r_k - s_k)^2$$

- K est le nombre d'exemples (cas),
- k est le numéro d'un exemple particulier,
- s_k est la sortie calculée avec les entrées de l'exemple k,
- r_k est la réponse voulue pour l'exemple k.

Remarque 13.
En fait, on peut voir le problème de régression linéaire (multiple) comme un réseau à un seul neurone et fonction d'activation $f(x) = x$, en renommant les différents constituants du modèle :



Les K exemples d'apprentissage ont bien la forme $(e_1, \dots, e_n ; r) = (\, x_1[k]\,, \dots\,, x_d[k]\,;\, y[k]\,)$ et le critère à minimiser est bien $MSE = \frac{1}{K}\sum_{k=1}^{K}\Big(\, y[k] - \widehat{y}\big(x_1[k], \dots, x_n[k]\big)\,\Big)^2 = moy\big([y-\widehat{y}]^2\big)$.

Enfin, le terme constant a_0 du modèle de régression correspond à ajouter au neurone une entrée e_0 constamment égale à 1 (dans tous les cas : apprentissage, validation, test, nouveaux...).

Par contre dans ce chapitre les y (réels : variable continue) sont remplacés par des sorties s, r entre 0 et 1 (soit binaires, soit pourcentages, pouvant traduire des variables qualitatives comme vu au semestre 4).

2.3 Perceptron simple : réseau à un seul neurone

Définition 14.
En suivant la définition 9, un *perceptron* simple est un réseau avec $n = 1$ neurone, un poids w_i pour chaque entrée e_i, et activé par la fonction de Heaviside $\mathcal{H}$.

Propriété 15.
Dans cette situation, les sorties calculées s et les réponses voulues r sont binaires (0 ou 1), donc les écarts $(s-r)$ sont toujours dans l'ensemble $\{-1;0;+1\}$. Par conséquent, les $(s-r)^2$ sont binaires et testent simplement si s diffère de r (cela vaut 1 si $s \neq r$, et 0 si $s = r$).
Ainsi, le MSE est ici le pourcentage de cas où on obtient $s \neq r$.

Définition 16.
Vu que les données d'entrée e_i sont des réels, on peut interpréter les entrées d'un cas d'apprentissage $(e_1[k], e_2[k], \ldots, e_n[k])$ comme les coordonnées d'un point en dimension n.

- On appelle « classe 0 » l'ensemble de ces points en dimension n pour lesquels la réponse voulue $r[k]$ vaut 0.
- De même, la « classe 1 » est l'ensemble des points pour qui la réponse voulue vaut 1.

Théorème 17.
Le problème d'apprentissage avec un *perceptron* simple possède des solutions (rappel : les inconnues sont les poids w_i) sans aucune erreur ($MSE = 0$) si et seulement s'il existe un hyperplan (sous-espace de dimension $n-1$) qui partage l'espace (de dimension n) en un demi-espace qui contient toute la classe 0, et un demi-espace qui contient toute la classe 1. On dit alors que les classes sont **linéairement séparables**.

En général : soit il n'existe aucune solution, soit il en existe une infinité.

(Théorème admis mais voir illustration en exercice pour $n=2$: hyperplan = droite.)

Remarque 18.
Avec les notations ci-dessus, on est limité aux hyperplans linéaires (passant par l'origine). Par contre en introduisant un terme constant comme à la remarque 13, on a accès aux hyperplans affines (pouvant passer par, ou éviter, n'importe quel point). Il y a deux manières de noter cela :

- ▶ soit on modifie juste la définition de la fusion en $x = b + \sum w_i e_i$ où b est un coefficient ajustable supplémentaire, parfois appelé biais (attention : différent du terme de statistique inférentielle) ;
- ▶ soit comme à la remarque 13, on fait comme si le neurone avait une entrée supplémentaire e_0 qui vaut toujours 1 ; son poids w_0 correspond au biais b ci-dessus, ce qui donne des formules un peu plus compactes ($\sum w_i e_i$) mais attention : on continue à raisonner en dimension n (alors qu'il y a maintenant $n+1$ entrées).

Dans le cadre de ce cours, on utilisera plutôt la seconde manière.

Algorithme 19.
Il n'y a pas de formule qui résoudrait le problème d'apprentissage du *perceptron* simple (un neurone, activation Heaviside) mais l'algorithme suivant converge vers une solution quand il en existe. Si les classes ne sont pas linéairement séparables, l'algorithme ne se "stabilise" jamais...).

```
in :  Tmax   // nombre maximum d'itérations
in :  pas    // constante strictement positive (assez petite)
in :  W   // tableau initial des poids synaptiques (éventuellement aléatoire...)
out :  W   // tableau des poids synaptiques mis à jour
pour( t = 1, ..., Tmax )
      k <- ...  // numéro aléatoire (sélectionner un cas dans l'échantillon)
      s <- ...  // sortie du neurone, calculée avec les  W[...]  et les  E[k,...]
      delta <- ( R[k] - s )     // écart entre réponses voulue et calculée
      pour( i = 0, ..., n )   // toutes les entrées y compris le biais
            W[i] <- W[i] + pas*delta*E[k,i]    // mise à jour du poids correspondant
      fin.pour
fin.pour
```

Remarque 20.
L'algorithme 19 ne fait que modifier petit à petit une proposition initiale des poids, en espérant l'améliorer à chaque itération. En pratique, on initialise avec des valeurs w_i aléatoires proches de zéro : en effet on espère ainsi avoir une moitié de bonnes réponses dès le début, et de la facilité à faire changer le signe des poids quand on en a besoin.

Définition 21.
Si on considère un seul neurone avec fonction d'activation sigmoïde (cf définition 9), c'est le problème de la **régression logistique** : la machine doit « apprendre » des w_i tels que

$$\mathbb{P}(Y=1) \approx \mathcal{S}\left(\sum_{i=0}^{n} w_i\, e_i\right)$$

ce qui équivaut à une sorte de régression linéaire

$$logit\Big(\mathbb{P}(Y=1)\Big) \approx \sum_{i=0}^{n} w_i\, e_i$$

mais le membre de gauche n'est pas directement observé...

Rappel du semestre 4 : la fonction *logit* est la réciproque de la sigmoïde

$$logit(p) = \ln\left(\frac{1}{1-p}\right)$$

Dans ce cas le MSE est une fonction continue et dérivable en les w_i :

$$MSE = \frac{1}{K}\sum_{k=1}^{K}\left(r[k] - \frac{1}{1+exp(-\sum w_i e_i[k])}\right)^2.$$

Il y a une unique solution (c'est-à-dire des w_i qui minimisent MSE) mais on ne possède pas de formule explicite pour l'exprimer...

Algorithme 22.
Pour les fonctions d'activation f **dérivables** sur $\mathbb{R}$, l'algorithme d'apprentissage s'adapte facilement : toutes les étapes restent identiques sauf le calcul de δ.

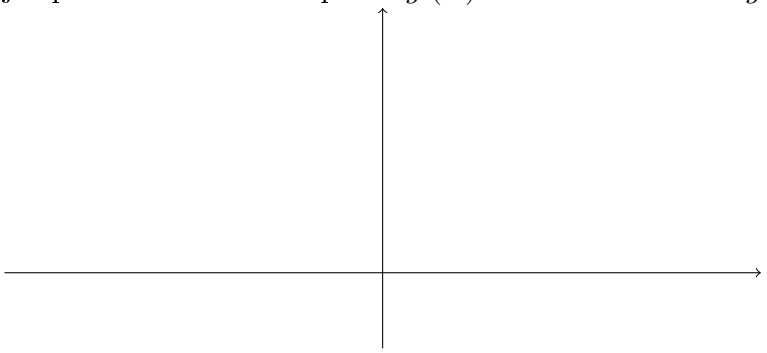
```
in :  Tmax, pas, W   // comme précédemment
out :  W   // tableau des poids synaptiques mis à jour
pour( t = 1, ..., Tmax )
      k <- ...  // numéro aléatoire (sélectionner un cas dans l'échantillon)
      s <- ...  // sortie du neurone, calculée avec les W[...] et les E[k,...]
      delta <- ( R[k] - s ) * f'(x)   avec x = ∑ wi ei l'entrée fusionnée
      pour( i = 0, ..., n )   // toutes entrées y compris le biais
           W[i] <- W[i] + pas*delta*E[k,i]   // mise à jour du poids correspondant
      fin.pour
fin.pour
```

Remarque 23.

- De manière générale `s` $= f($`x`$)$.
- Pour la fonction sigmoïde on a $\forall x \in \mathbb{R},\ \mathcal{S}'(x) = \mathcal{S}(x) \times [1-\mathcal{S}(x)]$, donc la formule devient `delta <- ( R[k] - s ) * s * (1-s)`
- Pour la fonction tangente hyperbolique on a $\forall x \in \mathbb{R},\ tanh'(x) = 1 - [tanh(x)]^2$, donc la formule devient `delta <- ( R[k] - s ) * (1-s^2)`
- Dans ces deux cas ($\mathcal{S}$ et $tanh$), on n'a donc pas besoin de stocker `x`, car il suffit de réutiliser la valeur de `s`.

Remarque 24.
La formule encadrée dans l'algorithme 22 est une **descente de gradient**.

- Cela généralise la célèbre méthode de Newton-Raphson connue pour les fonctions g d'une seule variable (notée w pour mieux se repérer). Cette méthode itère :
$$\underbrace{w}_{m.a.j.} \longleftarrow \underbrace{w}_{courant} - \underbrace{\frac{1}{g''(w)}}_{pas} \times g'(w)$$
jusqu'à "se stabiliser" quand $g'(w) \approx 0$: minimum de g lorsque $g''(w) > 0$.



- Dans ce chapitre la fonction g à minimiser est une fonction de $(n+1)$ variables qui sont les poids w_i (rappel de la définition 12) :
$$g(w_0, w_1, \dots, w_n) \;=\; MSE \;=\; \frac{1}{K}\sum_{k=1}^{K}\left[R[k] \;-\; f\left(\sum_{i=0}^{n} w_i\, e_i[k]\right)\right]^2 \qquad (*)$$
et chaque w_i va être mis à jour suivant une formule similaire à celle de Newton :
$w_i \longleftarrow w_i - pas \times \dfrac{\partial g}{\partial w_i}$.
En rassemblant toutes ces équations en une seule avec des matrices (et un pas unique) :
$$\underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}}_{\mathbf{W}} \longleftarrow \mathbf{W} - pas \cdot \underbrace{\begin{bmatrix} \partial g/\partial w_0 \\ \partial g/\partial w_1 \\ \vdots \\ \partial g/\partial w_n \end{bmatrix}}_{gradient\ de\ g}.$$
De plus, ici les dérivées partielles de g peuvent s'écrire à l'aide de la dérivée de f (qui est une fonction d'une seule variable). En effet d'après (*) on a pour tout $i = 0, \dots, n$ en appliquant les formules $(u^2)' = 2\,u'\,u$ et $(f \circ v)' = v' \times (f' \circ v)$:
$$\frac{\partial g}{\partial w_i} \;=\; \frac{1}{K}\sum_{k=1}^{K} 2 \times \underbrace{\Big[0 - e_i[k]\, f'(\dots)\Big]}_{u'=v'\times(f'\circ v)} \times \underbrace{\Big[R[k] \;-\; f(\dots)\Big]}_{u} \text{ où } \textstyle\sum w_i e_i[k] \text{ est abrégé } (\dots)$$
Avec les notations x et s de l'algorithme (qui dépendent de k) cela devient :
$$\frac{\partial g}{\partial w_i} \;=\; \frac{-2}{K}\sum_{k=1}^{K} e_i[k] \times \underbrace{f'\big(x[k]\big) \times \Big[R[k] \;-\; s[k]\Big]}_{\delta[k]}$$
et la formule de mise à jour devrait être $w_i \longleftarrow w_i + \underbrace{pas \times \frac{2}{K}}_{autre\ pas}\sum_{k=1}^{K} \delta[k]\, e_i[k]$.

- L'algorithme 22 utilise donc le $(\delta \times e)$ de chaque exemple k individuel pour faire des mises à jour au fur et à mesure, plutôt que d'attendre d'avoir calculé les $(\delta \times e)$ de tous les k, puis faire une mise à jour avec leur somme. Est-ce équivalent ? ...

2.4 Réseau avec plusieurs neurones

Définition 25.
Définir un réseau avec plusieurs neurones, c'est connecter certaines sorties de neurones aux entrées d'autres neurones. L'architecture du réseau est donc un **graphe orienté**.

Il y aura toujours des neurones particuliers, en lien « direct » avec l'environnement extérieur :
- l'environnement « stimule » le réseau par ces entrées particulières ;
- le réseau « répond » par ces sorties particulières.

Chaque connexion possèdera un poids ajustable : cf figure 4.

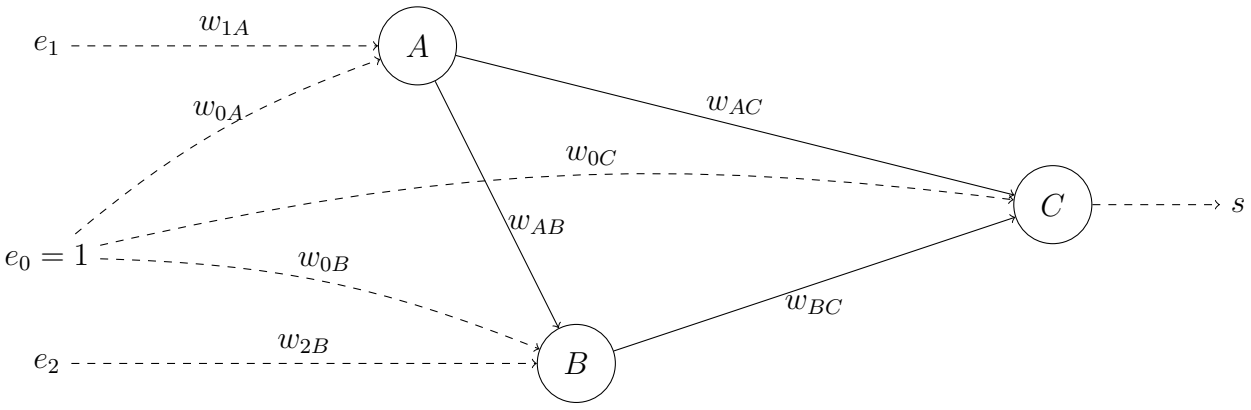


FIGURE 4 – Exemple de réseau avec les notations de ce cours.

Remarque 26.
Nous n'étudions ici que des réseaux
- ▶ **sans circuit** (donc graphe décomposable en niveaux),
- ▶ avec une seule sortie (donc un seul neurone au dernier niveau : on l'appellera Z),
- ▶ où tous les neurones (sauf éventuellement Z) ont la même fonction d'activation f dérivable.

Chaque neurone se comporte comme au paragraphe 2.1 : cf figure 5.

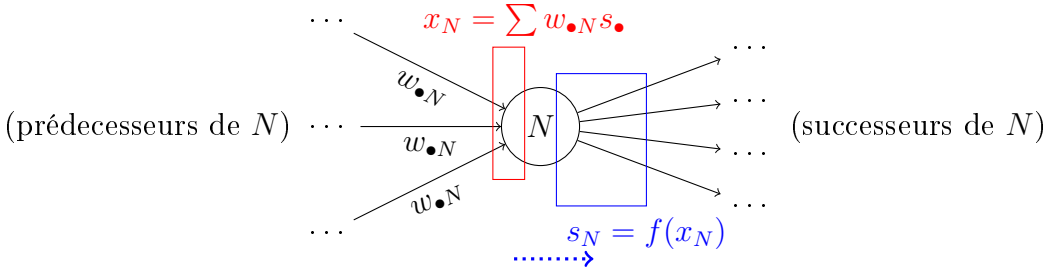


FIGURE 5 – Entrées et sorties d'un neurone dans le réseau.

Propriété 27.
Dans un graphe sans circuit, on peut ranger les sommets par « niveaux » :

- Ici au niveau 0, on trouvera les entrées du réseau : e_1, e_2, ... mais aussi e_0.
  Par convention on notera $s_i = e_i$ en faisant comme si les entrées étaient des neurones.
- Si on connaît les sorties s_N de tous les neurones N jusqu'au niveau ℓ, alors on peut calculer les sorties de tous les neurones au niveau $(\ell + 1)$.

- En **propageant** ainsi les sorties de niveau en niveau, on finit par calculer la réponse s du réseau ($s = s_Z$ pour le neurone Z seul au dernier niveau).

  Cela permet de calculer δ comme précédemment : écart entre la réponse voulue R et cette sortie s, éventuellement multiplié par $f'(x_Z)$ pour le neurone Z au niveau maximum.

- Comment ajuster les poids à partir de l'écart constaté δ ?
  L'idée est de dire qu'en fait, on a calculé δ_Z pour le neurone Z au niveau maximum
  (on est donc capable de mettre à jour ses poids entrants $w_{\bullet Z}$)
  mais que chaque autre neurone intermédiaire N doit avoir son propre δ_N
  (qui servira à mettre à jour ses poids entrants $w_{\bullet N}$).

Attention : les δ_N sont **tous calculés avant** la mise à jour des poids.

On les obtient en **rétro-propageant** les « erreurs » niveau par niveau, depuis le niveau maximum (neurone Z) jusqu'au niveau 1 : cf figure 6.
(Les neurones e_i du niveau 0 n'ont pas de connexion entrante, donc pas besoin de δ_{e_i}.)

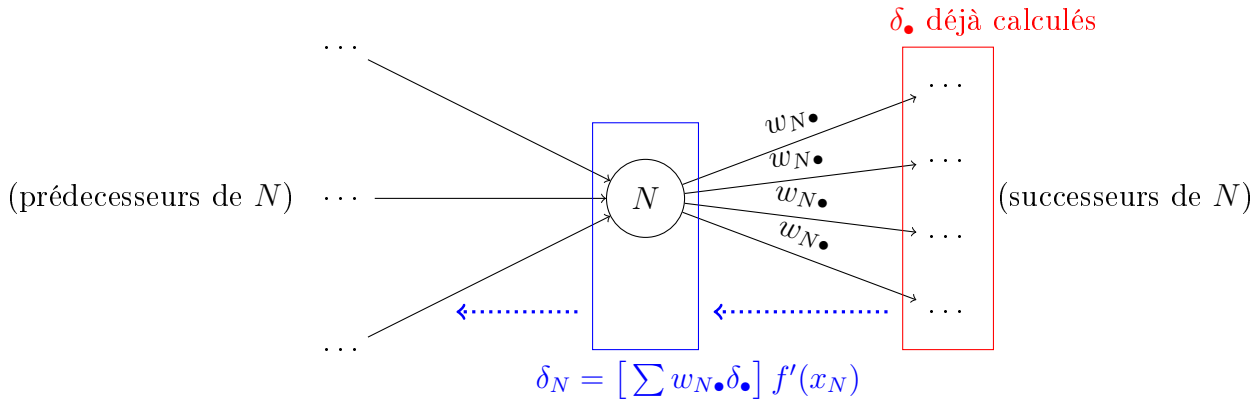


FIGURE 6 – Calcul des erreurs $\delta_\bullet$ par rétro-propagation.

Toutes ces formules peuvent être justifiées de manière similaire à la remarque 24.

Algorithme 28.
Voici une description de l'algorithme d'apprentissage par rétro-propagation.

Données : `Tmax`, `pas`, `W` *(cf algorithmes 19 et 22)*

Pour (itération `t` de `1` à `Tmax`)

Sélection d'un cas :
par exemple tirer `k` aléatoirement ▶ on extrait `E[k,...]` et `R[k]`

Comportement du réseau dans ce cas :
du niveau 1 jusqu'au niveau maximum, calculer tous les `x[N]` et `s[N]`

Calcul des erreurs :
on commence au niveau maximum : `delta[Z] <- (R[k] - s[Z]) * f'(x[Z])`
puis jusqu'au niveau 1 : `delta[N] <- sum( W[N,N']*delta[N'] ) * f'(x[N])`
(pour chaque neurone `N`, la somme porte sur tous les `N'` successeurs de `N`)

Mise à jour des poids :
pour chaque connexion `N`→`N'` faire
`W[N,N'] <- W[N,N'] + pas * delta[N'] * s[N]`

FinPour

2.5 Perceptron multi-couches

Définition 29.

- Il y a n entrées $(e_1, \dots, e_n)$ et un biais (e_0) au niveau 0.
- Il y a un seul neurone de sortie, au niveau L.
- À chaque niveau $\ell = 0, \dots, L$ le nombre de neurones est appelé $N(\ell)$.
  En particulier on a donc $N(0) = n + 1$ et $N(L) = 1$.
- Pour tout $\ell = 0, \dots, (L-1)$, chaque neurone du niveau ℓ possède une connexion sortante vers chaque neurone du niveau $(\ell + 1)$, et il n'y a pas d'autres connexions.
  Seule exception : e_0 a une connexion sortante vers chaque neurone de chaque niveau.

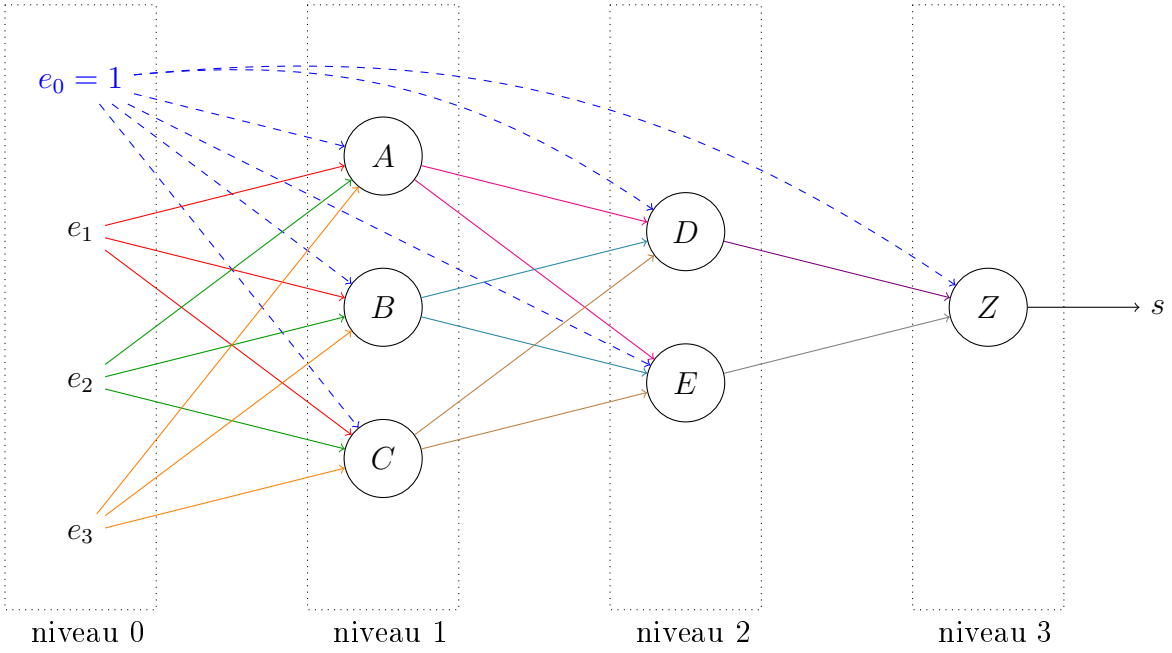


FIGURE 7 – Exemple de perceptron multi-couches.

Il suffit de fournir : $N(0)$ ou n, puis L, et enfin les $N(\ell)$ avec $\ell = 1, \dots, (L-1)$ pour générer automatiquement toute l'architecture.

Par exemple, on peut construire la figure 7 à partir des seules informations suivantes : $n = 3$; $L = 3$; $N(1) = 3$ et $N(2) = 2$.

Enfin, tous les neurones ont la même fonction d'activation f dérivable (souvent la sigmoïde dans ce cours).

Propriété 30.
L'intérêt de l'architecture en couches est que les neurones d'une même couche peuvent travailler « simultanément ». Mathématiquement, on exprime cela avec des formules matricielles (cf dernier exercice). Du point de vue de l'implémentation, on utilise des bibliothèques de calcul parallèle, notamment sur cartes graphiques.

Exercice 2.1

On considère un neurone artificiel avec 4 entrées, les poids synaptiques donnés par le tableau $\mathbf{W} = [\ 0.1\ ;\ 0.5\ ;\ -1.25\ ;\ 1\ ]$, et la fonction de Heaviside comme fonction d'activation.

1. Représentez ce neurone par un schéma dans le style de la figure 3.
2. Combien vaut la sortie lorsque les entrées sont données par le tableau $\mathbf{E} = [\ 1\ ;\ 0.5\ ;\ 0.75\ ;\ 0.9\ ]$?
3. Combien vaut la sortie quand toutes les entrées sont nulles ? Ceci dépend-il des w_i ?

Exercice 2.2

Soit l'échantillon suivant à discriminer :

Cas	e_1	e_2	rép.voulue
1	0.1	0.5	1
2	0.8	0.9	1
3	0.7	0.3	0

1. Représentez les « cas » par des points de coordonnées (e_1, e_2) dans un plan, de couleurs différentes suivant la réponse voulue.
2. Traduisez chaque « cas » par une inéquation.
3. Établissez un lien entre les poids w_i cherchés et les coefficients (a, b) pour une certaine droite qui « sépare » les points selon leur couleur.
4. Déterminez graphiquement un couple (a, b) cohérent avec les inéquations.
5. Déduisez-en deux triplets (w_0, w_1, w_2) différents, solutions du problème posé.

Exercice 2.3

Pour chaque échantillon ci-dessous, appliquez la méthode de l'exercice précédent pour répondre au problème d'apprentissage avec un *perceptron* à 2 entrées et 1 biais.

éch.A :

Cas	e_1	e_2	rép.voulue
1	0.1	0.2	1
2	0.8	0.9	1
3	0.7	0.3	0

éch.B :

Cas	e_1	e_2	rép.voulue
1	0.1	0.2	1
2	0.8	0.9	1
3	0.7	0.3	0
4	0.3	0.6	0

éch.C :

Cas	e_1	e_2	rép.voulue
1	0.1	0.2	1
2	0.8	0.9	1
3	0.7	0.3	0
4	0.8	0.1	0
5	0.7	0.6	1
6	0.2	0.5	1

éch.AND :

Cas	e_1	e_2	rép.voulue
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

éch.OR :

Cas	e_1	e_2	rép.voulue
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	1

éch.XOR :

Cas	e_1	e_2	rép.voulue
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Exercice 2.4

On souhaite ajuster un perceptron pour l'échantillon d'apprentissage suivant :

Cas	e_1	e_2	e_3	rép.voulue
1	0	1.5	−0.7	1
2	0.4	1.6	1.1	1
3	−0.1	2.5	0.3	0

Initialement, on fixe les poids par $\mathbf{W} = \left[\ -1\ ;\ 0.3\ ;\ 0.6\ ;\ 0\ \right]$.

1. Dessinez une représentation de ce perceptron.
2. Calculez les sorties de ce perceptron dans les 3 cas, avec les poids initiaux.
3. Détaillez le premier passage dans la boucle (`t=1`), quand `k` tombe sur *2*.
   Puis, même question quand `k` tombe sur *1*. On prendra un pas égal à 0,1.
4. Recalculez les sorties du perceptron dans les 3 cas, avec les poids ainsi mis à jour.
5. Interprétez la formule de mise à jour des poids.

Exercice 2.5

Refaites l'exercice précédent, avec la fonction d'activation sigmoïde.
Quel(s) problème(s) poserait la fonction de rectification (x^+) ?

Exercice 2.6

Dans la liste des fonctions d'activation (définition 9) :

1. lesquelles sont la dérivée d'une des autres ?
2. pourquoi dit-on que $\mathcal{S}$ est une version lissée de $\mathcal{H}$?
   et de même pourquoi le nom *softplus*, qui signifie version lissée de (x^+) ?

Exercice 2.7

Reformulez l'algorithme d'apprentissage 28 pour le perceptron multi-couches (définition 29) avec des opérations matricielles : elles devront exploiter le fait que les neurones à l'intérieur d'un même niveau peuvent travailler « en parallèle » avec les mêmes entrées.

On pourra noter :
$\times$ le produit matriciel,
$\cdot$ le produit d'un scalaire et d'une matrice,
$\odot$ le produit de deux matrices de mêmes dimensions, coefficient par coefficient.