Notes de cours de réseaux de neurones
L1 Maths-Info

Philippe Preux
Université de Lille

12 janvier 2025

Résumé

Ce cours est une introduction aux réseaux de neurones. On s'intéresse à leur fonctionnement et à la compréhension de ceux-ci. Il existe des tas de types de réseaux de neurones différents ; on ne traite que des plus populaires, les perceptrons multi-couches. Destiné à des étudiants de 1re année universitaire, les pré-requis en mathématiques et en informatique sont réduits : en mathématiques, il faut connaître la notion de dérivée d'une fonction à une variable réelle et celle de droite dans le plan ; en informatique, il faut connaître les rudiments de la programmation (variables, tests, boucles, fonctions).

1 Introduction

Cette première section donne quelques repères généraux sur les réseaux de neurones. Par la suite, on traite des réseaux de neurones de manière technique.

1.1 Repères historiques

- En 1791, L. Galvani reconnait la nature électrique des signaux nerveux.
- Fin XIXe siècle, considéré comme le premier neuroscientifique, R. y Cajal étudie les cellules nerveuses, propose une théorie neuronale, et affirme que la capacité des neurones à croître et à créer de nouvelles connexions peut expliquer l'apprentissage.
- Dans les années 1930, des mathématiciens éclaircissent la notion de calcul et de fonction calculable. A. Turing propose sa machine (abstraite) capable de calculer mécaniquement toute fonction calculable. Ces fonctions calculables traitent des nombres entiers.
- En 1943, W. Mc Culloch et W. Pitts proposent une description mathématique du fonctionnement d'un neurone biologique.
- En 1956, le mot « intelligence artificielle » est créé.
- En 1958, F. Rosenblatt propose le modèle du perceptron. Le perceptron est capable de réaliser des tâches très simples, autrement dit, de calculer des fonctions mathématiques très simples. On mettra ensuite 30 ans à trouver comment calculer à peu près n'importe quelle fonction (1986).

1.2 Neurone naturel et neurone artificiel

Très schématiquement, le cerveau reçoit des signaux en entrées provenant de nos sens et émet des signaux en sortie qui contrôlent nos muscles et nous fait donc agir et réagir dans notre environnement.

Le cerveau est constitué de cellules nerveuses, également nommées neurones.

Le neurone artificiel est une simplification extrême du neurone réel. Très grossièrement, un neurone réel est constitué d'un noyau, il reçoit des signaux électriques sur ses entrées (dendrites), et possède une sortie ramifiée (axone). La sortie d'un neurone est connectée à des dendrites via des synapses qui constituent l'interface entre un axone et une dendrite. Lorsqu'un neurone émet une décharge électrique sur son axone (cette décharge se nomme un potentiel d'action), elle se propage dans ses ramifications jusqu'à atteindre des synapses. Selon les conditions, la décharge est transmise dans la synapse puis parcourt une dendrite jusqu'à atteindre un neurone. Typiquement, un neurone reçoit de nombreuses décharges au fil du temps qui le chargent électriquement. Si sa charge électrique dépasse un certain seuil, il décharge en émettant un potentiel d'action sur son axone, *etc.* Les rayons lumineux qui atteignent nos yeux, les sons dans nos oreilles, les sensations de toucher, goût, *etc.* entraînent des décharges électriques vers des neurones. Cela entraîne une cascade d'activités neuronales pour, *in fine*, entraîner la décharge électrique de neurones qui agissent sur un ou des muscles. En conditions normales, dans un animal en bonne santé, ce processus met une fraction de seconde à se réaliser. La connexion entre deux neurones est plus ou moins forte, selon la valeur du poids synaptique. Dans notre cerveau, au fil du temps des neurones se créent et disparaissent, des connexions également, et les poids synaptiques changent en fonction de nos expériences. Le cerveau humain comprend environ 10^{11} neurones et environ 10^{15} connexions entre neurones.

1.3 Applications

Depuis les progrès réalisés en 1986, les neurones artificiels ont vu leur champ d'applications croître considérablement. On a commencé à réfléchir à la combinaison de plusieurs neurones, créant des réseaux de neurones. Vers 1990, ils sont utilisés pour reconnaître des chiffres écrits à la main et ainsi reconnaître les codes postaux sur les courriers et faire du tri automatique du courrier. Ils sont également utilisés pour réaliser un programme qui apprend à jouer au jeu de Backgammon à partir des règles du jeu et jouant au niveau des experts humains, jouant des coups qui étonnent ceux-ci. Vers 2000, ils sont utilisés pour reconnaître les chiffres sur les chèques et ainsi lire automatiquement le montant des chèques aux guichets automatiques des banques. Mis à part cela, le domaine stagne, même si les réseaux de neurones sont utilisés dans un très grand nombre d'applications. Par stagnation, j'entends que la complexité des tâches que l'on arrive à résoudre ne progresse pas. En effet, pour réaliser des tâches complexes, il faut combiner plusieurs neurones en un réseau et pendant 20 ans, on ne sait pas comment entraîner un tel réseau de neurones à calculer une certaine fonction dès que le nombre de neurones devient un peu grand. À cette époque, la puissance des ordinateurs est encore faible [1], ce qui limite la quantité de calculs. En étudiant avec précision les freins à l'utilisation de plus gros réseaux de neurones, la notion de réseau de neurones profond apparaît à la fin de la décennie 2000. Avec ces plus gros réseaux, un premier résultat expérimental est obtenu en 2013 avec le réseau ImageNet qui est capable de détecter des objets dans des images avec une précision jamais vue précédemment, réalisant cette tâche en commettant moins d'erreurs que des êtres humaines confrontés à la même tâche. À la suite de ce premier résultat, les avancées vont être très rapides que ce soit dans le domaine de la détection d'objets dans des images (par exemple la génération automatique de la légende d'une image, 2015), dans le domaine des langues naturelles, ou dans le domaine des jeux (jeu de go en 2017), puis des modèles génératifs. Outre la

1. Vers 2000, un bon ordinateur personnel de bureau est cadencé à 500 MHz et il est mono-cœur.

disponibilité de capacités de calcul de plus en plus énormes, ces progrès s'appuient sur la disponibilité massive (voire le pillage massif) de données sur Internet. Devenue une technologie aussi puissante, et furtive, ces progrès sont accompagnés de nombreux dangers liés par exemple à son usage en vidéo-surveillance, dans les armes, la réalisation de faux et la manipulation de l'opinion. Nécessitant de très grandes quantités d'énergie, elle participe également, et très activement, au réchauffement planétaire.

2 Perceptron binaire à sortie binaire

Il existe de très nombreux types de neurones artificiels et de réseaux de neurones artificiels. Dans cette introduction, on ne traite que des plus connus : le neurone artificiel dénommé **perceptron** et le réseau de neurones le plus classique constitué d'un ensemble de perceptrons, le perceptron multi-couches.

Nous présentons plusieurs types de perceptrons :

— perceptron binaire à sortie binaire : entrées et sortie sont à valeur binaire.

— perceptron à sortie binaire : sortie à valeur binaire, entrées à valeur réelle (*cf.* sec. 3).

— perceptron réel : entrées et sortie à valeur réelles (*cf.* sec. 4).

Ces différents types de perceptron se ressemblent très fortement.

2.1 Définition

On définit un ensemble de termes et de notations :

— Un **perceptron binaire** est une unité de calcul constituée de P **entrées** à valeur binaire et d'une **sortie** à valeur binaire également.

— On note e_j l'entrée numéro j.

— Chaque entrée e_j est pondérée par un **poids** à valeur réelle noté p_j.

— Il existe par ailleurs un poids particulier associé à une entrée qui vaut toujours 1. Ce poids est dénommé le **biais** et on le numérote 0, donc p_0 est le biais.

— Le **potentiel** du perceptron est noté v et il vaut : $v = p_0 + \sum_{j=1}^{j=P} p_j e_j$.

— La sortie s du perceptron est une certaine fonction notée φ de v, soit $s = \varphi(v)$. Cette fonction se nomme la **fonction d'activation** du perceptron. Dans le cas du perceptron à sortie binaire, on prendra :

$$\varphi(v) = \begin{cases} 1 \text{ si } v \geq 0, \\ 0 \text{ si } v < 0. \end{cases}$$

Cette fonction se nomme la **fonction de Heaviside** [2].

2.2 Exercice

On considère un perceptron à deux entrées et sortie binaires. Calculer la sortie s pour des poids et des entrées ci-desous :

2. *Stricto senso*, ce n'est pas exactement la fonction de Heaviside telle qu'on la définit en mathématiques. En maths, Heaviside(0) = 1/2.

p_0	p_1	p_2	e_1	e_2	s
0,5	$-1,5$	2	1	0	
-1	0,25	-3	1	0	
-2	1	1	1	0	
-1	2	2	1	0	
-3	2	2	1	0	
-3	2	2	0	0	
-3	2	2	1	1	
-3	2	2	0	1	

2.3 Programmation

Programmer le calcul de la sortie d'un perceptron binaire (= TP 1 [3]).

2.4 Trouver les poids qui permettent de calculer une certaine fonction. Mise en bouche

La question essentielle que l'on se pose est : quelle valeur donner aux poids d'un perceptron pour que la sortie de celui-ci corresponde à ce que l'on veut ?

2.4.1 Exercice

On considère les fonctions logiques à 2 entrées. Trouver, si c'est possible, des poids pour qu'un perceptron calcule : le ET logique, le OU logique, le NON-ET logique, le NON-OU logique, le OU-EXCLUSIF logique.

Par exemple pour le ET, on trouve en tatonnant : $p_0 = -5, p_1 = 2, p_2 = 4$, ou $p_0 = -1, p_1 = 0{,}5, p_2 = 0{,}5$.

On peut aussi réfléchir sur les contraintes à satisfaire :

— 0 et 0 $\rightarrow$ 0 : donc, $p_0 < 0$

— 0 et 1 $\rightarrow$ 0 : donc, $p_0 + p_1 < 0$

— 1 et 0 $\rightarrow$ 0 : donc, $p_0 + p_2 < 0$

— 1 et 1 $\rightarrow$ 1 : donc, $p_0 + p_1 + p_2 \geq 0$

et en déduire des poids qui vérifient ces 4 contraintes.

2.4.2 Interprétation géométrique

Le potentiel d'un perceptron à deux entrées est $v = p_0 + p_1 e_1 + p_2 e_2$. La sortie est 1 si le potentiel est positif ou nul, 0 s'il est négatif.

Intéressons-nous à l'équation : $0 = p_0 + p_1 e_1 + p_2 e_2$.

Que l'on peut écrire : $e_2 = -\frac{p_0}{p_2} - \frac{p_1}{p_2} e_1$ qui a la forme $y = ax + b$. C'est donc l'équation d'une droite dans le plan e_1 en absicsses, e_2 en ordonnées.

Une droite $y = ax + b$ sépare le plan en deux sous-espaces : l'un correspond à $y \geq ax + b$ et l'autre à $y < ax + b$.

On peut ré-écrire : $y \geq ax + b \rightarrow 0 \geq ax + b - y$ et l'autre à $y < ax + b \rightarrow 0 < ax + b - y$.

3. Les sujets des TP sont disponibles sur la page du cours à l'url `https://philippe-preux.github.io/ensg/l1-mi/`.

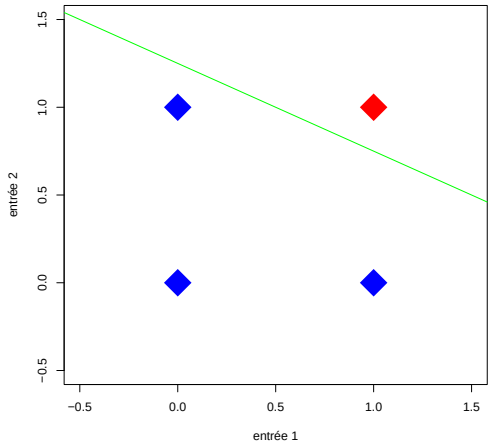


FIGURE 1 – Comme indiqué dans le texte, on a représenté les 4 exemples de la fonction ET à deux entrées, la couleur indiquant la valeur associée (bleu pour 0, rouge pour 1), c'est-à-dire la sortie attendue du perceptron pour cette donnée. Et on a indiqué la droite $e_2 = \frac{5}{4} - \frac{2}{4}e_1$ qui sépare le rouge des bleus.

Reprenons l'exemple du ET logique et le jeu de poids $p_0 = -5, p_1 = 2, p_2 = 4$.
Dessinons le plan (e_1, e_2).
Plaçons-y les 4 points et colorons la valeur en ce point en bleu si c'est 0, rouge si c'est 1.
Traçons la droite correspondant aux 3 poids, soit : $e_2 = \frac{5}{4} - \frac{2}{4}e_1$.
Les points bleus sont d'un côté de la droite, le(s) rouge(s) de l'autre. Voir la figure 1.

2.4.3 Exercice

Appliquer la même démarche aux fonctions OU, NON-ET, NON-OU. Que se passe-t-il pour le OU-EXCLUSIF ?

OU-EXCLUSIF : géométriquement (*cf.* fig. 2), on voit que quelle que soit la droite tracée, on n'arrivera jamais à séparer les 0 des 1.

Algébriquement, on peut écrire le système d'inégalités :

1. 0 et 0 $\rightarrow$ 0 : donc, $p_0 < 0$
2. 0 et 1 $\rightarrow$ 0 : donc, $p_0 + p_1 \geq 0$
3. 1 et 0 $\rightarrow$ 0 : donc, $p_0 + p_2 \geq 0$
4. 1 et 1 $\rightarrow$ 1 : donc, $p_0 + p_1 + p_2 < 0$

En ajoutant 1. et 4., on obtient $2p_0 + p_1 + p_2 < 0$.
En ajoutant 2. et 3., on obtient $2p_0 + p_1 + p_2 \geq 0$.
Ces deux inégalités ne peuvent pas être satisfaites en même temps, il n'y a donc pas de solution.

2.5 Exercice

Afficheur 7 LEDs : celui-ci est utilisé pour afficher un chiffre :

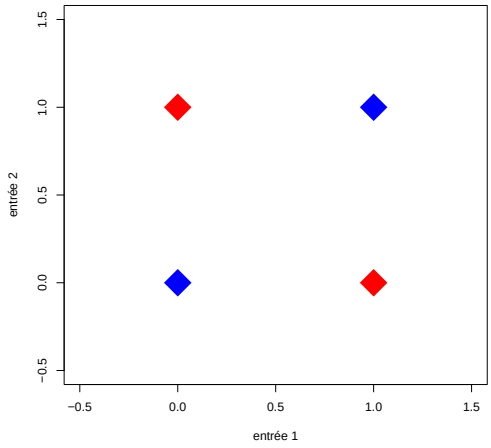
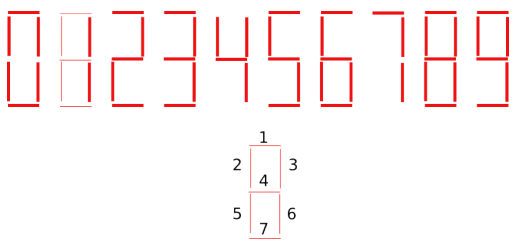


FIGURE 2 – On a représenté les 4 exemples de la fonction OU-EXCLUSIF à deux entrées, la couleur indiquant la sortie attendue du perceptron (bleu pour 0, rouge pour 1) pour cette donnée. Impossible de tracer une droite qui sépare les points bleus des points rouges.



On suppose que chaque entrée d'un perceptron est associée à un segment de l'afficheur et qu'elle reçoit 0 si le segment est éteint, 1 s'il est allumé.

Peut-on trouver un perceptron à 7 entrées qui :

— détermine que le chiffre affiché est 6 ?

— détermine la parité du chiffre affiché ?

— détermine que le chiffre affiché est < 5 ?

— détermine que le chiffre affiché $\in \{3, ..., 7\}$?

2.6 Méthode pour calculer les poids d'un perceptron binaire

On peut écrire un système d'inégalités : dans le cas des fonctions logiques plus haut, tous les cas sont énumérables, il y en a 2^n si on considère les fonctions logiques ayant n entrées. Cependant, résoudre ce genre de système d'inégalités n'est pas si simple (pas au programme de la L1 !). Et 2^n augmente très vite et cette méthode n'est pas applicable pour n pas très petit.

Dans le cas de l'afficheur, on ne s'intéresse pas à toutes les combinaisons mais à quelques-unes (10 par rapport aux $2^7 = 128$ combinaisons possibles d'entrées). Cela entraîne le fait que le nombre d'inégalités est très petit par rapport au nombre de combinaisons possibles, ce qui complique les choses [4].

4. En L1, on résout des systèmes de n équations linéaires à n inconnues, le nombre d'équations est égal au nombre d'inconnues. Quand ce n'est pas le cas, la résolution du système est moins simple.

3 Perceptron à entrées réelles et sortie binaire

On considère maintenant un perceptron dont les valeurs en entrée peuvent être n'importe quel nombre réel. La sortie est binaire.

Il y a plusieurs manières de définir la fonction d'activation. Pour l'instant, la sortie est calculée comme pour le perceptron binaire plus haut.

On suppose que l'on dispose d'un ensemble d'exemples, c'est-à-dire de couples (entrées du perceptron, sortie du perceptron). Par exemple pour le ET-logique rencontré plus haut, il y a 4 exemples :

— exemple 1 : (entrée = $(0, 0)$, sortie = 0)
— exemple 2 : (entrée = $(0, 1)$, sortie = 0)
— exemple 3 : (entrée = $(1, 0)$, sortie = 0)
— exemple 4 : (entrée = $(1, 1)$, sortie = 1)

3.1 Recherche des poids d'un perceptron

3.1.1 Intuition

Comment trouver les poids ?

On cherche des poids tels que la sortie du perceptron soit celle attendue en fonction des entrées.

On procède itérativement :

— **initialisation** : on commence avec des poids pris aléatoirement (en 2D si le perceptron a deux entrées, cela revient à prendre une droite au hasard dans le plan) ;
— **tant qu**'il y a des différences entre la sortie attendue et la sortie calculée par le perceptron :
  — **pour chaque exemple** :
    1. on le place en entrée et on calcule la sortie du perceptron ;
    2. **si** la sortie du perceptron n'est pas celle attendue, **alors** on modifie légérement les poids (cela revient à modifier légérement la position de la droite).

On peut dire qu'il existe une fonction des poids f ($f(p_0, p_1, p_2)$ en 2D) et que l'image de f est le nombre d'erreurs du perceptron, une erreur étant le fait que la sortie du perceptron pour une entrée donnée n'est pas la sortie attendue.

L'objectif est alors de minimiser le nombre d'erreurs, c'est-à-dire trouver un jeu de poids (p_0^*, p_1^*, p_2^*) (en 2D) pour lequel f est minimale. C'est ce que l'on appelle un « problème de minimisation de fonction ». Il existe plusieurs méthodes pour résoudre un tel problème. On va voir la méthode qui est utilisée dans le cadre des perceptrons, méthode qui est très générale et peut s'appliquer à la minimisation de tas de fonctions, pas seulement pour trouver les poids d'un perceptron.

3.1.2 Optimisation de fonction par descente de gradient

Intuition : on veut trouver le point où une fonction est minimale. On prend un x_0 initial quelconque (*cf.* Fig. 3.(a)). On calcule $f(x_0)$ et selon la pente de f en x_0, c'est-à-dire, selon le signe de $f'(x_0)$, on modifie légérément la valeur de x_0 ce qui fournit la valeur de x_1 (*cf.* Fig. 3.(b)). On itère jusqu'à ce que la pente soit nulle (*cf.* Fig. 3.(c)).

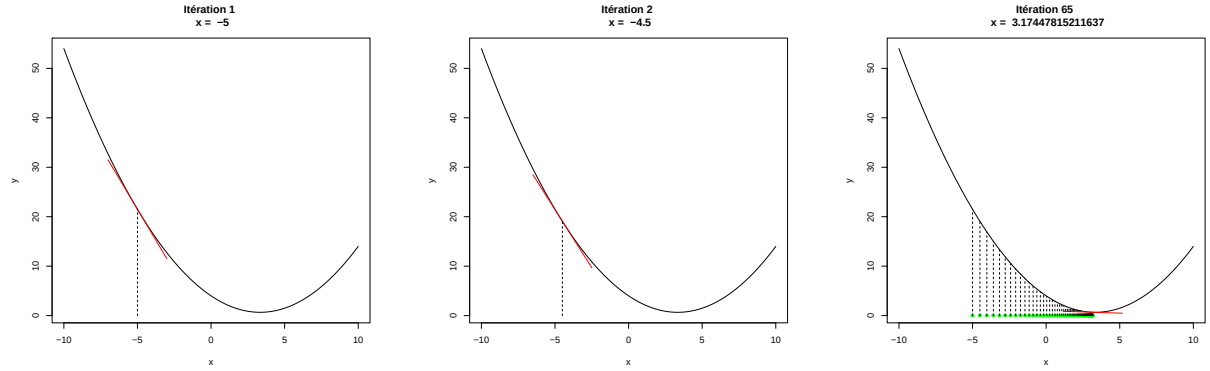


FIGURE 3 – Quelques étapes d'une descente de gradient stochastique pour trouver le minimum de cette fonction. (a) : x_0 choisi au hasard à partir duquel on commence la descente. (b) : x_1. (c) : la succession des x_k au fil des itérations de la descente de gradient stochastique. Sur chaque figure, le petit trait rouge indique la tangente à la courbe au point courant.

On peut appliquer l'idée à n'importe quelle fonction, quelle que soit la dimension de son domaine de définition. Il faut pouvoir calculer sa dérivée f' en tout point du domaine donc que f soit continue et dérivable sur son domaine de définition. Cela donne l'algorithme 1.

Algorithme 1 Minimisation d'une fonction réelle à une variable par descente de gradient stochastique. À la sortie de la boucle **tant-que**, x_k contient la valeur recherchée, $f(x_k)$ est un minimum (presque, voir le texte).

Nécessite: un seuil d'arrêt ϵ
Nécessite: une valeur pour η
Nécessite: fonction à minimiser : f
Nécessite: dérivée de la fonction à mnimiser : f'
1: $k \leftarrow 0$
2: initialiser x_k (aléatoirement ou autrement)
3: **tant-que** $|f'(x_k)| > \epsilon$ **faire**
4: $\quad k \leftarrow k + 1$
5: $\quad x_k \leftarrow x_{k-1} - \eta f'(x_{k-1})$
6: **fin tant-que**

Quelques remarques concernant la mise en pratique de cet algorithme :

— η est un nombre petit (de l'ordre d'un centième ou d'un millième).

— on arrête les itérations quand la pente est petite (voir ligne 3 de l'algorithme 2), car en pratique, elle ne sera jamais exactement nulle.

— La valeur de η doit diminuer au fil des itérations. D'une manière générale, savoir comment faire décroître η est loin d'être un sujet simple et pour notre brève présentation, nous nous contenterons de garder fixe la valeur de η.

— En gardant fixe la valeur de η, on ne peut pas atteindre le minimum, on ne fait que s'en approcher.
— Cette méthode de minimisation de fonctions n'est pas à utiliser pour des fonctions « habituelles » en mathématiques, en particulier pour des polynômes. On dispose de méthodes bien plus performantes pour celles-ci. Par contre, dans le cas du perceptron, c'est ce type de méthodes qui est utilisé, certes avec quelques raffinements dont la description dépasserait l'esprit de ce cours d'introduction.

3.1.3 Programmation

Implanter la DGS pour une fonction quelconque (= TP 3).

3.1.4 Optimisation de fonction générale par descente de gradient pour le perceptron

On définit quelques notations : on suppose disposer de N exemples. Un **exemple** i est un couple (x_i, y_i) où x_i est une donnée et y_i son **étiquette**. Une **donnée** est un vecteur de P nombres que l'on appelle les **attributs** de la donnée. On note $x_{i,j}$ la valeur de l'attribut j de la donnée i. C'est un nombre réel. Chacun des attributs d'une donnée est placé sur l'une des entrées du perceptron qui compte donc $P+1$ entrées : l'entrée e_j reçoit l'attribut numéro j. Quand une donnée x_i est placée en entrée du perceptron, on veut alors que la sortie du perceptron soit y_i. Dans cette partie du cours, on considère que la sortie du perceptron, donc les étiquettes y_i, peuvent prendre deux valeurs distinctes (perceptron à sortie binaire). Pour l'instant, on a considéré que ces deux valeurs sont 0 et 1, donc $y_i \in \{0, 1\}$.

La question est : comment trouver les $P+1$ poids tels que si l'on place la donnée x_i sur les entrées du perceptron, sa sortie soit y_i ?

Quand on place x_i en entrée du perceptron et que sa sortie n'est pas y_i, on dit que le perceptron commet une **erreur de prédiction**.

Pour trouver les poids, on procède par descente de gradient stochastique.

Comme on l'a dit plus haut, on cherche à minimiser le nombre d'erreurs commises par le perceptron. L'ensemble des exemples étant fixé, ce nombre d'erreurs est une fonction des poids. C'est cette fonction que l'on veut minimiser. À chaque ensemble de N exemples correspond un nombre d'erreurs de prédiction. L'idée est donc de modifier légérement les poids pour faire diminuer ce nombre d'erreurs.

Pour cela, la règle est la suivante : on suppose que l'on a placé x_i en entrée du perceptron. La sortie du perceptron est s. Si s est différent de y_i, alors :

1. on calcule $d = s - y_i$,
2. on modifie le biais $p_0 \leftarrow p_0 - \eta d$,
3. on modifie les P poids numérotés 1 à P comme suit : $p_j \leftarrow p_j - \eta d x_{i,j}, \forall j \in \{1, ..., P\}$.

Cette modification des poids doit être réalisée pour tous les exemples dont l'étiquette ne correspond pas à la sortie du perceptron.

Et on doit continuer d'effectuer ces corrections tant que la somme des (valeurs absolues des) corrections apportées à l'ensemble des N exemples demeure supérieure à un certain seuil. Ces corrections correspondent à la dérivée que nous avons rencontrée dans la descente de gradient stochastique (Sec. 3.1.2).

η porte différents noms. Dans ce cours, on l'appelle le **taux d'apprentissage**.

Rédigé sous la forme d'un algorithme, cela nous donne l'algorithme de descente de gradient pour calculer les poids d'un perceptron binaire. Cet algorithme a été découvert plusieurs fois et a été nommé avec des noms différents par ses différents découvreurs. Ainsi, il se nomme également la règle Delta, ou Adaline, ou règle d'apprentissage du perceptron. Il est décrit par l'algorithme 2.

Algorithme 2 La règle d'apprentissage du perceptron.

Nécessite: N exemples (x_i, y_i)
Nécessite: une valeur pour η
Nécessite: une valeur pour ϵ
initialiser les poids $p_j, j \in \{0, ...P\}$ (aléatoirement ou autrement)
corrections $\leftarrow 1$
tant-que $|\text{corrections}| > \epsilon$ **faire**
  corrections $\leftarrow 0$
  pour chaque exemple i **faire**
    calculer la sortie s du perceptron quand x_i est placé sur ses entrées.
    $d \leftarrow s - y_i$
    si $d \neq 0$ **alors**
      $p_0 \leftarrow p_0 - \eta d$
      corrections $\leftarrow$ corrections $+ \eta|d|$
      pour chaque poids $p_j, j \in \{1, ..., P\}$ **faire**
        $p_j \leftarrow p_j - \eta d x_{i,j}$
        corrections $\leftarrow$ corrections $+\eta|dx_{i,j}|$
      fin pour
    fin si
  fin pour
fin tant-que

Comme on l'a vu plus haut, les poids d'un perceptron ont une interprétation géométrique. Dans le cas d'un perceptron à deux entrées, on peut visualiser un jeu de poids sous la forme d'une droite ; modifier légérement la valeur des poids entraîne une modification légère de la droite correspondante. Ainsi, géométriquement, le jeu de poids initialement aléatoire correspond à une droite prise au hasard dans le plan. Au fil de l'algorithme de descente de gradient stochastique, la droite se positionne peu à peu dans une configuration dans laquelle les données étiquetées 0 sont séparées des données étiquetées 1. Cela est illustré par la figure 4.

3.1.5 Programmation

Programmer la descente de gradient pour le calcul des poids d'un perceptron à partir d'un ensemble d'exemples (= TP 4).

3.1.6 Remarque

En section 3.1.2, on a écrit que l'on corrige la variable en fonction de la dérivée de la fonction que l'on minimise. Or, dans l'algorithme 2, on ne voit pas apparaître de dérivée.

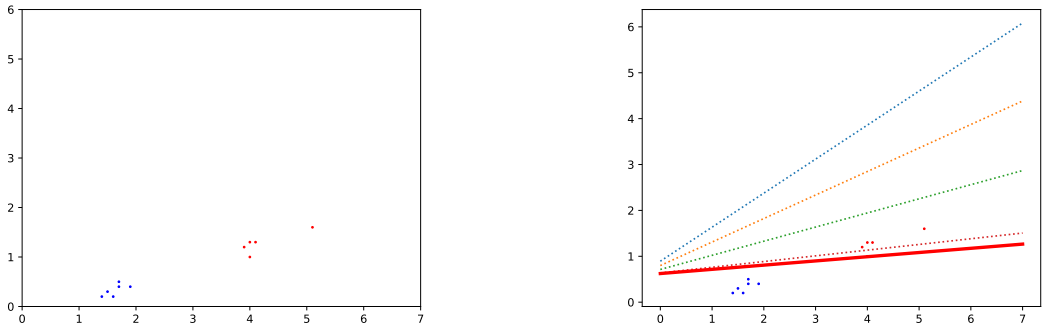


FIGURE 4 – Représentation graphique du déroulement du calcul des poids d'un perceptron à deux entrées durant une descente de gradient stochastique. La figure de gauche représente 11 données, la couleur indiquant la sortie attendue pour chacune d'entre-elles. À droite, l'initialisation aléatoire des poids du perceptron correspond à la droite la plus haute. On voit ensuite la succession de droites obtenues à la sortie de la boucle **pour**, ligne 16 dans l'algorithme 2. À la dernière itération, on obtient les poids correspondant à la droite rouge en trait gras : cette droite sépare les points bleus des points rouges et en utilisant les poids correspondant, la sortie du perceptron est celle attendue pour les 11 données.

Pourquoi ?

C'est une excellente question.

La lecture de ce qui suit n'est pas forcément facile. Je vous encourage fortement à la lire et à essayer de comprendre.

Revenons à la fonction qui est minimisée par la DGS quand celle-ci est appliquée au calcul des poids d'un perceptron.

Cette fonction n'a pas été explicitée plus haut ; j'ai juste écrit que l'on veut minimiser le nombre d'erreurs. La fonction qui est minimisée est $\zeta = \sum_i (s_i - y_i)^2$ où s_i est la valeur en sortie du perceptron quand la donnée x_i est placée sur ses entrées. C'est ce que l'on appelle l'erreur quadratique : on fait la somme du carré de la différence entre la sortie du perceptron (s_i) et la sortie voulue (y_i) sur tous les exemples (d'où $\sum_i$). Pourquoi prendre le carré et pas seulement la différence ? On somme cette différence sur l'ensemble des exemples, aussi il faut que ses termes soient positifs sinon ils peuvent s'annuler les uns les autres. Pourquoi ne pas prendre la valeur absolue ? Le carré est une fonction dérivable sur l'ensemble de son domaine contrairement à la valeur absolue qui n'est pas dérivable en 0 (et bientôt, on va vouloir dériver la fonction à optimiser, donc c'est mieux si elle est dérivable).

Rappelons que le potentiel v du perceptron est $v = p_0 + p_1 e_1 + p_2 e_2$ et sa sortie s vaut 1 si $v \geq 0$, 0 sinon. Aussi, s_i vaut 1 si le potentiel $v_i \geq 0$, 0 sinon. On peut exprimer cela avec la fonction de Heaviside : $s = \text{Heaviside}(v)$.

Donc $\zeta = \sum_{i=1}^{i=N} (Heaviside(p_0 + p_1 x_{i,1} + p_2 x_{i,2}) - y_i)^2$.

ζ est une fonction des poids : $\zeta(p_0, p_1, p_2)$: on veut trouver la valeur de ces poids qui minimisent ζ.

Aussi, on doit s'intéresser à la dérivée de ζ par rapport à chacun des 3 poids.

En L1, on connait la dérivée d'une fonction à une variable. Un jour, vous allez apprendre comment dériver une fonction de plusieurs variables par rapport à l'une de ses variables. C'est tout à fait possible, et en fait, ce n'est pas plus difficile que de calculer une dérivée par rapport à une variable comme vous

avez appris à le faire au lycée : quand on dérive une fonction par rapport à une variable, on fait comme si les autres variables étaient des constantes. Et, en termes de notation, au lieu d'écrire $\frac{df}{dx}$, on écrit $\frac{\partial f}{\partial x}$ [5].

Ici, on doit dériver ζ par rapport à chacun des poids, donc calculer $\frac{\partial \zeta}{\partial p_0}$, $\frac{\partial \zeta}{\partial p_1}$ et $\frac{\partial \zeta}{\partial p_2}$.

Donc, pour p_0 : $\frac{\partial \zeta}{\partial p_0} = \frac{\partial \sum_i (Heaviside(p_0+p_1x_{i,1}+p_2x_{i,2})-y_i)^2}{\partial p_0} = \sum_i \frac{\partial (Heaviside(p_0+p_1x_{i,1}+p_2x_{i,2})-y_i)^2}{\partial p_0}$.

Le problème est qu'en L1, on sait qu'une fonction qui n'est pas continue n'est pas dérivable. Or, la fonction de Heaviside n'est pas continue en 0, donc n'est pas dérivable. Si vous poursuivez vos études, vous apprendrez peut-être un jour qu'il existe une notion de dérivée plus générale que celle que vous connaissez et que toute fonction, même non continue, est dérivable. Cela pourrait être la solution à notre problème mais non, ça ne l'est pas car la dérivée de la fonction de Heaviside n'est pas utilisable ici.

Comme on est bloqué, on triche et on se dit : après tout, pour calculer ces dérivées de ζ par rapport aux poids, je vais faire comme si je n'appliquais pas la fonction de Heaviside à la somme, donc je vais juste calculer $\sum_{i=1}^{i=N} \frac{\partial (p_0+p_1x_{i,1}+p_2x_{i,2}-y_i)^2}{\partial p_0}$.

Et ça, c'est très facile [6] :

— pour le biais, on obtient : $\sum_{i=1}^{i=N} \frac{\partial (p_0+p_1x_{i,1}+p_2x_{i,2}-y_i)^2}{\partial p_0} = 2\sum_{i=1}^{i=N}(p_0 + p_1x_{i,1} + p_2x_{i,2} - y_i) = 2\sum_{i=1}^{i=N}(s_i - y_i)$.

— pour le poids associé à l'entrée j, on obtient : $\sum_{i=1}^{i=N} \frac{\partial (p_0+p_1x_{i,1}+p_2x_{i,2}-y_i)^2}{\partial p_j} = 2\sum_{i=1}^{i=N} x_{i,j}(p_0 + p_1x_{i,1} + p_2x_{i,2} - y_i) = 2\sum_{i=1}^{i=N} x_{i,j}(s_i - y_i)$.

Et on reconnait les corrections qui sont appliquées par l'algorithme 2 aux lignes 9 pour le biais, et 12 pour les autres poids.

3.2 Quelle fonction peut calculer un perceptron binaire ?

On parle ici de fonctions à valeur dans $\{0, 1\}$.

En deux dimensions (perceptron ayant deux entrées), un perceptron peut calculer une fonction séparable par une droite ; en 3 dimensions, par un plan ; en plus grande dimension ($\mathbb{R}^D$), par un hyper-plan (en dimension ($\mathbb{R}^{D-1}$)).

Ces fonctions portent le nom de « fonctions linéairement séparables ».

4 Perceptron non linéaire à entrées et sortie réelles

4.1 Définition

Si l'utilisation d'un perceptron à sortie binaire est possible, on lui préfère l'utilisation d'un perceptron à sortie réelle. Dans ce cas, la fonction d'activation qui passe de manière brutale (non dérivable) de 0 à 1 est remplacée par une fonction dérivable qui varie progressivement entre 0 et 1. Cette fonction est en forme de 'S' et se nomme une fonction sigmoïde. Deux sont bien connues :

— la fonction logistique $\varphi(v) = \frac{1}{1+e^{-v}} \in]0, 1[$.

— la fonction tangente hyperbolique $\varphi(v) = \tanh(v) \in]-1, 1[$ [7].

5. Et le curieux petit caractère ∂ se nomme tout simplement « d rond ».

6. Si cela n'est pas facile, il faut regarder votre cours de maths.

7. Rappelons que la tangente hyperbolique est définie par : $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

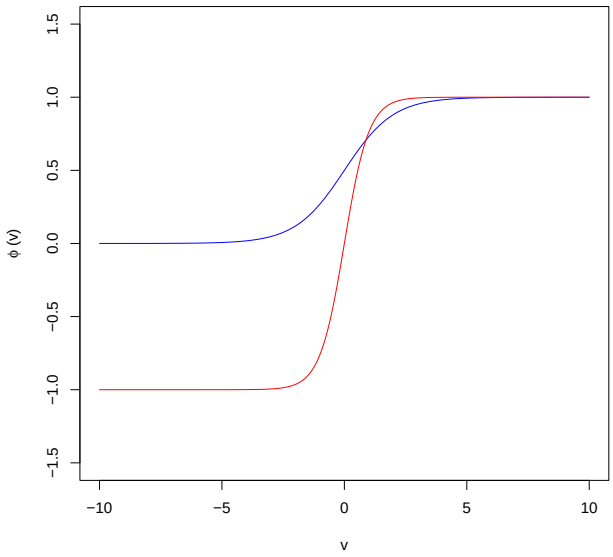


FIGURE 5 – En bleu, la fonction logistique ; en rouge, la fonction tangente hyperbolique.

La figure 5 représente graphiquement ces deux fonctions. On voit bien une forme de 'S'. Les deux fonctions tendent asymptotiquement vers $+1$ quand l'abscisse est positif et croît. Elles tendent vers 0 pour la fonction logistique, -1 pour la tangente hyperbolique, quand l'abscisse est négatif et décroît. On voit sur la figure que le passage de -1/0 vers 1 est rapide [8] et que ces deux fonctions sont symétriques en 0.

Pour utiliser la tangente hyperbolique, il faut utiliser les étiquettes -1 et 1 et non pas 0 et 1.

D'une manière générale, la tangente hyperbolique produit de meilleurs résultats que la fonction logistique, notamment parce que son image est symétrique autour de 0.

La sortie du perceptron est cette fois-ci réelle. Dans le cas de la tangente hyperbolique, on utilise le signe de la sortie comme étiquette de la donnée placée en entrée. Par ailleurs, ces fonctions d'activation permettent de détecter si la sortie du perceptron est plus ou moins sûre. Si la sortie est proche de -1 ou de 1, c'est que le perceptron « est sûr de lui » ; au contraire, si la sortie est proche de 0, c'est que le perceptron « hésite ».

On adapte très facilement la règle d'apprentissage du perceptron vue plus haut. Il suffit de modifier la mise à jour des poids pour tenir compte de la dérivée (par rapport à chacun des poids) de la fonction d'activation qui existe pour ces deux fonctions d'activation sigmoïdes.

Pour la fonction logistique $\varphi(v) = \frac{1}{1+e^{-v}}$, la mise à jour des poids (ligne 12 de l'algorithme 2) est :

$p_j \leftarrow p_j - \eta dx_{i,j}\varphi(v)(1-\varphi(v))$, où v est le potentiel du perceptron : $v = p_0 + \sum_{j=1}^{j=P} p_j e_j$.

Pour la tangente hyperbolique $\varphi(v) = \tanh(v)$, cette mise à jour des poids est :

$p_j \leftarrow p_j - \eta dx_{i,j}(1-\varphi(v)^2)$.

8. Pour être plus rigoureux, le passage de -1/0 à 1 peut être plus ou moins rapide.

4.2 Programmation

Faire le TP 5 qui est identique au TP 4 si ce n'est que l'on utilise désormais une fonction d'activation sigmoïde.

Il faut :

— modifier la mise à jour des poids comme c'est indiqué ci-dessus.

— La sortie du perceptron étant un nombre réel, l'étiquette prédite n'est plus la sortie. Dans le cas de la fonction logistique, cette étiquette est 1 si la sortie du perceptron est $\geq 1/2$, 0 sinon. Pour la tangente hyperbolique, c'est 1 si la sortie est positive ou nulle, 0 sinon.

— Centrer et réduire les attributs des données. Un attribut est centré si sa moyenne est nulle. On le réduit en le divisant par son écart-type. Aussi, pour un attribut $x_{.,j}$, on le transforme en : $\frac{x_{.,j}-\bar{x_{.,j}}}{\sigma(x_{.,j})}$, où $\bar{v}$ est la moyenne de l'attribut v et $\sigma(v)$ est son écart-type. Un attrbut centré et réduit est de moyenne nulle et sa variance vaut 1.

En appliquant ces différents points, vous ne devriez rencontrer aucune difficulté et obtenir à peu près les mêmes résultats qu'avec un perceptron à sortie binaire.

5 Méthodologie pour résoudre un problème de classification supervisée binaire

5.1 Introduction

Pour l'instant, on a juste calculé des poids pour que la valeur en sortie d'un perceptron soit celle que l'on attend (l'étiquette de la donnée). Cela a peu d'intérêt et on n'a pas besoin d'un perceptron pour réaliser cette opération. Si on utilise un perceptron pour cela, c'est parce que notre objectif n'est pas celui-là. L'objectif que nous poursuivons est qu'après avoir calculé les poids à partir d'un ensemble d'exemples, le perceptron soit capable de produire la bonne sortie pour une donnée qu'il n'a jamais rencontré. Pour prendre un exemple (dont la mise en œuvre dépasse l'ambition de ce cours), imaginons que les données soient des images et que l'étiquette indique si un véhicule est visible dans l'image. Pour réaliser cela, on va utiliser un ensemble d'exemples, donc de couples (image, étiquette), pour calculer les poids du perceptron et bien sûr, notre espoir est qu'ensuite on puisse présenter n'importe quelle image au perceptron et que celui-ci indique bien si un véhicule est présent ou non. Réaliser cela porte le nom de tâche d'**apprentissage** ; une caractéristique clé d'une telle tâche est qu'une fois les poids calculés, on peut présenter n'importe quelle image et que la sortie du perceptron est correcte. Pour cela, le perceptron doit être capable de **généraliser** les situations qu'il a rencontrées dans les exemples. On parle également d'**induction**, ce qui consiste à formuler des règles générales à partir d'exemples.

Pour être tout à fait précis, on parle ici d'apprentissage qualifié de **supervisé** : d'une manière très générale, l'apprentissage supervisé consiste à associer une valeur à une donnée. La « valeur » peut être de type très divers. Si l'étiquette y ne peut prendre qu'un nombre fini de valeurs différentes, on parle de classification **supervisée**. Si l'étiquette a une valeur réelle, on parle de problème de **régression** (c'est encore un problème d'apprentissage supervisé). L'étiquette peut encore être un texte (utilisé pour générer automatiquement la légende d'une image par exemple), ou même une image (c'est ce que font par exemple les fameux modèles génératifs qui créent des images à partir d'un texte).

Le calcul des poids se nomme l'**entraînement** du perceptron. Lorsque l'on calcule la sortie du perceptron pour une certaine donnée placée en entrée, on parle d'**inférence**. Pendant l'inférence, les

poids ne sont pas modifiés.

Pour réaliser avec succès une telle tâche d'apprentissage supervisé, il y a une méthodologie à suivre que nous allons présenter. Auparavant, nous précisons en quoi consiste la notion d'« apprentissage ». De cette notion découle la méthodologie.

L'idée est que les exemples qui sont utilisés pour réaliser une tâche d'apprentissage sont collectées selon un processus aléatoire parmi l'ensemble des exemples possibles. Illustrons cela sur l'exemple des iris. Par un bel après-midi, vous vous promenez et vous ramassez des iris dans la nature. Il y en a des tas autour de vous et vous en ramassez certains, au hasard. Rentré chez vous, constatant qu'il semble y avoir différentes sortes d'iris dans votre bouquet, vous regardez une flore et déterminez qu'effectivement, les iris de votre bouquet se rattache à 3 espèces différentes. À chaque iris, vous associez son espèce, vous l'étiquetez. Vous voulez ensuite créer un système informatique qui détermine l'espèce d'un iris étant données certaines caractéristiques de sa fleur. Vous utilisez ces iris pour configurer ce système. Néanmoins, votre souhait est que ce système ne soit pas juste capable de déterminer l'espèce des fleurs de votre bouquet ; vous voulez qu'il puisse déterminer l'espèce de n'importe quel iris. Aussi, vous utilisez les iris dont vous disposez comme exemples de ce que sont les iris en général. C'est exactement ce que l'on veut faire quand on résout une tâche d'apprentissage supervisé : on a des exemples et à partir de ceux-ci, on veut configurer notre système (ici un perceptron, mais il existe des tas d'autres possibilités) qui fonctionne pour n'importe quelle donnée, même si elle ne fait pas partie de l'ensemble des exemples. Par « apprentissage » (supervisé), on entend cette capacité à **généraliser** au-delà des seuls exemples dont on dispose.

5.2 Méthodologie pour l'apprentissage supervisé

En se basant sur cette compréhension de ce que signifie « apprentissage » dans ce cours, disposant d'un ensemble d'exemples, une idée naturelle consiste à utiliser un sous-ensemble de ceux-ci pour calculer les poids d'un perceptron et conserver l'autre sous-ensemble d'exemples pour mesurer la performance de prédiction du perceptron. La descente de gradient stochastique va corriger les poids du perceptron tant que les erreurs de prédiction sur le second sous-ensemble d'exemples diminue.

L'ensemble de N exemples est découpé en deux parties, le sous-ensemble des exemples d'entraînement et le sous-ensemble des exemples de test. Typiquement, 80% des exemples constituent le jeu d'entraînement, mais cette proportion est un choix.

On peut alors mesurer deux choses : la proportion d'exemples d'entraînement dont l'étiquette est mal prédite, ce qui constitue une estimation de l'erreur de prédiction sur les exemples d'entraînement ou **erreur d'entraînement** ; la même quantité mesurée sur les exemples de test qui estime la probabilité d'erreur de prédiction en test, ou **erreur de test**. De ces deux quantités, la première n'est pas très importante, par contre la seconde est très significative et utile.

Ensuite le principe est le suivant :

— on prend en considération les exemples d'entraînement un par un et on corrige les poids du perceptron comme on l'a vu plus haut ;

— on calcule l'erreur de test ;

— on recommence tant-que cette erreur diminue.

Cette idée se formalise dans l'algorithme 3.

Cet algorithme nécessite quelques précisions pour l'implanter :

Algorithme 3 Algorithme de calcul des poids d'un perceptron pour un problème de classification supervisée.

Nécessite: N exemples (x_i, y_i)

Nécessite: une valeur pour le taux d'apprentissage η

Nécessite: proportion d'exemples d'entraînement q

découper l'ensemble d'exemples en un sous-ensemble de qN exemples d'entraînement, les autres constituant le sous-ensemble d'exemples de test. On note N_{test} le nombre d'exemples composant le sous-ensemble de test.
initialiser les poids $p_j, j \in \{0, ...P\}$ (aléatoirement ou autrement)
répéter
    # on corrige les poids avec l'ensemble d'entraînement
    pour chaque exemple d'entraînement i **faire**
        calculer la classe prédite par le perceptron pour x_i et corriger les poids
    fin pour
    nb.erreurs $\leftarrow 0$
    pour chaque exemple de test i **faire**
        calculer la classe prédite c par le perceptron pour l'exemple de test i
        $d \leftarrow c - y_i$
        si $d \neq 0$ **alors**
            nb.erreurs $\leftarrow$ nb.erreurs +1
        fin si
    fin pour
    erreur.de.test $\leftarrow$ nb.erreurs $/N_{test}$
jusque critère d'arrêt

— habituellement, on prend $q = 0{,}8$.
— Aux lignes 5, 6, il faut que les exemples d'entraînement soient utilisés dans un ordre aléatoire qui change à chaque itération de la boucle **répéter**.
— Le critère d'arrêt n'est pas précisé à la ligne 17. On pourrait s'arrêter quand erreur.de.test ne diminue plus, ou quand les corrections tombent sous un certain seuil. Voir ce qui a été dit plus haut pour la règle d'apprentissage du perceptron.

  Mais ce ne sont pas de bonnes idées. La bonne manière de pratiquer consiste à se baser sur l'erreur de test. Sa mise au point est délicate. Tant que l'erreur de test diminue, il faut continuer à modifier les poids. Quand elle ne diminue pas, cela ne veut pas dire qu'il ne faut pas continuer à modifier les poids car ces modifications se basent sur d'autres exemples, les exemples d'entraînement. Si malgré de nombreuses corrections des poids l'erreur de test persiste à ne pas diminuer, il faut s'arrêter. En effet, si on n'arrête pas les corrections, l'erreur de test va finir par augmenter, ce qui est un très mauvais signe. C'est le signe du phénomène de **sur-apprentissage** : à force de corriger les poids, ceux-ci se sont trop adaptés aux exemples d'entraînement et ne sont plus adaptés à des exemples qui n'ont pas été utilisés pour le calcul des poids. Le sur-apprentissage est un phénomène extrêmement important que l'on rencontre presque systématiquement si l'on n'y prend pas garde. Un algorithme d'apprentissage supervisé doit absolument être conçu pour l'éviter.

5.2.1 Programmation

Programmer cet algorithme de classification supervisée à l'aide d'un perceptron (= TP 6).

5.3 Remarque concernant le mot « apprendre »

Vocabulaire : le mot apprendre signifie ici calculer les poids d'un perceptron. C'est effectivement le sens du mot « apprendre » en apprentissage artificiel/machine. Cela n'a strictement rien à voir avec le mot « apprendre » employé pour un animal [9]. En statistique, on appelle cela « ajuster les paramètres d'un modèle ». Il n'y a aucun apprentissage au sens de l'animal ou de l'humain dans ce genre d'algorithme. D'ailleurs, on sait que ce type d'apprentissage machine est extrêmement peu efficace par rapport à l'apprentissage chez les animaux qui est lui très efficace (condition de survie dans la nature).

6 Perceptron multi-couches

On a vu qu'un perceptron peut calculer la fonction logique NON-ET. Par ailleurs, on sait que toute fonction calculable peut être exprimée sous la forme d'une combinaison de fonctions NON-ET.

Exercice :

— exprimer la fonction ET binaire logique à l'aide de fonctions NON-ET.
— exprimer la fonction OU binaire logique à l'aide de fonctions NON-ET.
— exprimer la fonction OU-EXCLUSIF binaire logique à l'aide de fonctions NON-ET.

9. De même que le mot « intelligence » n'a strictement rien à voir si je parle d'un animal ou si ce mot est suivi du mot « artificiel ».

La conséquence immédiate est que par combinaison de perceptrons qui calculent chacun une fonction NON-ET binaire logique, on peut calculer n'importe quelle fonction.

C'est effectivement un résultat théorique important : toute fonction « calculable » peut être calculée par un ensemble de perceptrons. Dans cette phrase, le mot « calculable » a un sens mathématique très précis dont la définition précise nécessite des développements qui vont au-delà de ce cours. Disons de manière vague mais pragmatique qu'un ensemble de perceptrons peut calculer à peu près toutes les fonctions mathématiques que vous pouvez imaginer. Par ailleurs, j'écris « ensemble de perceptrons » : cette notion va être précisée dans la suite.

6.1 Définition du perceptron multi-couches

Une structure composée de plusieurs perceptrons est un **réseau de neurones**.

On peut imaginer de connecter des perceptrons via leurs entrées et leurs sorties de manière quelconque. Néanmoins, l'objectif est toujours de calculer les poids des entrées des perceptrons afin que l'ensemble de neurones associe les valeurs en sortie qui sont attendues pour certaines valeurs en entrée. Il faut donc pouvoir écrire un algorithme qui réalise ce calcul, un algorithme qui ne soit pas trop compliqué, pas trop long à exécuter et qui puisse s'appliquer à toutes les configurations d'interconnexions entre perceptrons.

Pour satisfaire ces attentes, on a défini le **perceptron multi-couches** (PMC). Dans un perceptron multi-couches, des perceptrons sont organisés en couches successives. Tout d'abord, comme pour le perceptron, on trouve la couche d'entrée qui reçoit la valeur des attributs d'une donnée. La couche d'entrée est composée de P entrées. Chacune de ces entrées est connectée à l'une des entrées des N_1 perceptrons de la couche cachée 1. La sortie de chacun de ces N_1 perceptrons est connectée à une entrée de chacun des N_2 perceptrons de la couche suivante (chacun de ces N_2 perceptrons possède $N_1 + 1$ entrées (+1 pour le biais)). Cela se poursuit de couche en couche, chacun des N_l perceptrons de la couche l ayant $N_{l-1} + 1$ entrées. Enfin, il y a une dernière couche N_L composée d'un perceptron connecté aux sorties des N_{L-1} perceptrons de la couche $L - 1$. Typiquement, tous ces perceptrons sont des perceptrons à entrées et sortie réelles. Pour une tâche de classification supervisée, tous les perceptrons ont une fonction d'activation sigmoïde (tangente hyperbolique de préférence).

La figure 6 présente graphiquement la structure d'un perceptron multi-couches.

On ne parle ici que de perceptrons multi-couches ayant 1 seul perceptron en couche de sortie. De manière générale, on peut en avoir plusieurs ce qui permet de traiter des problèmes de classification supervisée dans lesquels les étiquettes peuvent prendre plus de 2 valeurs.

6.2 Calcul des poids d'un perceptron multi-couches

On a buté sur le calcul des poids d'un perceptron multi-couches pendant des décenies. En 1986 a été proposé l'algorithme de rétropropagation du gradient de l'erreur. Celui-ci est une adaptation de l'algorithme de descente de gradient stochastique au cas du PMC. L'idée est toujours de modifier les poids synaptiques lorsque la sortie du PMC n'est pas la sortie attendue pour un exemple d'entraînement. Les modifications sont propagées dans les couches du PMC, depuis la couche de sortie vers la couche d'entrée (d'où le préfixe rétro- dans rétropropagation).

Si l'idée est simple, l'implantation d'un programme qui réalise cette opération correctement et de manière efficace ne l'est pas. En effet, il y a des tas de petits détails à prendre en considération pour

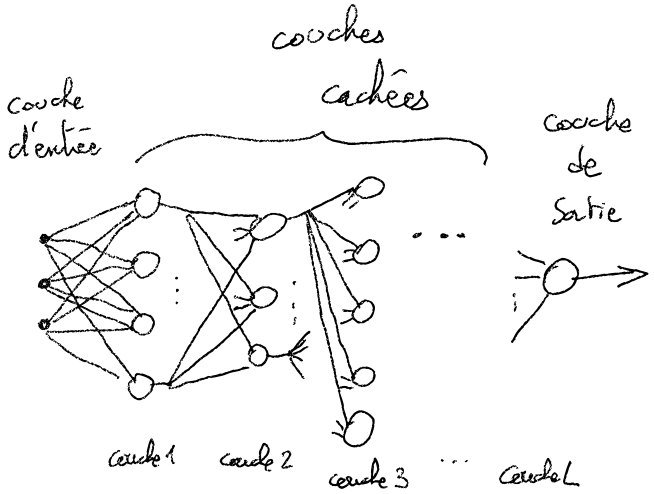


FIGURE 6 – Structure générale d'un perceptron multi-couches.

que l'entraînement se passe bien et soit réalisé efficacement. Depuis 1986, ces petits détails sont étudiés afin de pouvoir entraîner de plus en plus efficacement des PMC de plus en plus gros (voir l'historique en début de ces notes). Parmi les difficultés, on a buté pendant deux décennies sur l'entraînement de PMC ayant plus d'une dizaine de couches. Le franchissement de cette difficulté a ouvert la voie à ce que l'on appelle l'apprentissage profond.

6.3 Propriété des PMC

Un résultat théorique extrêmement important indique que toute fonction réelle continue et bornée définie sur un intervalle fermé de $\mathbb{R}$ peut être représentée avec une précision fixée arbitraire par un PMC ayant une couche cachée. Pour être plus précis, ce résultat indique que pour toute fonction de ce type et pour une précision fixée, il existe un PMC à une couche cachée qui peut réprésenter cette fonction avec cette précision. Cela signifie que l'écart entre la sortie du perceptron et la sortie attendue est inférieure à cette précision. En classification supervisée, cette précision signifierait que la sortie du PMC est correcte dans, par exemple, 95% des cas.

Ce résultat est en fait une conséquence directe du théorème qui exprime que n'importe quelle fonction de ce type peut être représentée comme une combinaison linéaire de fonctions sigmoïdes. Ce résultat indique qu'un PMC composé d'une seule couche cachée de perceptron dont la fonction d'activation est une sigmoïde et un perceptron linéaire en sortie peut représenter avec une précision voulue toute fonction continue et bornée sur un intervalle fermé de $\mathbb{R}$.

Si ce théorème est intéressant, il ne nous dit rien sur le nombre de perceptrons que doit compter la couche cachée pour obtenir cette représentation.

En pratique, plutôt qu'une seule couche cachée, on utilise généralement un PMC ayant plusieurs couches cachées. Cela permet de diminuer le nombre de perceptrons nécessaires pour réaliser cette représentation avec une précision fixée. Ce qui est important est le nombre le poids synpatiques, c'est-à-dire le nombre de paramètres du modèle.

D'un point de vue pratique, pour déterminer l'architecture d'un PMC pour résoudre une tâche donnée, on essaie : on teste des PMC de différentes tailles et on prend celui qui convient le mieux.

Notons enfin que ce théorème est souvent mal compris et on lit trop souvent des erreurs d'interprétation grossières. Ce théorème ne dit pas qu'un PMC peut représenter n'importe quelle fonction avec la précision que l'on veut. Ce théorème dit que pour une fonction donnée et une précision donnée, il existe un PMC qui peut représenter cette fonction avec cette précision. C'est un théorème d'existence de solution : dans l'ensemble de tous les PMC, il existe une solution pour représenter une fonction avec une certaine précision. En fait, il existe plusieurs solutions, pas une seule.

6.4 Remarque concernant le vocabulaire

Les PMC sont des modèles qualifiés de « paramétriques ». Cet adjectif ne signifie pas du tout qu'il y a des paramètres dans le modèle (les modèles non paramétriques ont eux-aussi des paramètres). Cet adjectif signifie que la structure (architecture) du réseau de neurones est fixée et que seuls les poids sont calculés. Il existe des réseaux de neurones dont la structure évolue au fil du temps : ce sont des modèles non paramétriques. Tous les réseaux de neurones dont on parle tant actuellement sont paramétriques, à structure fixée une fois pour toute.

6.5 Calcul efficace du potentiel d'un perceptron

On a défini plus le potentiel d'un perceptron ($v = p_0 + \sum_{j=1}^{j=P} p_j e_j$) qui est une combinaison linéaire des entrées par les poids auquel on ajoute le biais. Les P entrées d'un perceptron sont les P attributs d'une donnée x_i. On peut donc écrire v comme une fonction d'une donnée x_i : $v(x_i) = p_0 + \sum_{j=1}^{j=P} p_j x_{i,j}$. On peut obtenir une notation mathématique plus compacte en supposant que chaque donnée possède un attribut numéroté 0 qui vaut 1. Dès lors, on peut écrire :

$v(x_i) = p_0.x_{i,0} + \sum_{j=1}^{j=P} p_j x_{i,j} = \sum_{j=0}^{j=P} p_j x_{i,j}$.

Cette notation exhibe le fait que $v(x_i)$ est le produit scalaire entre le vecteur des poids et la donnée placée en entrée, vue comme un vecteur : $v(x_i) =< x_i, p >$.

On peut aller plus loin.

Considérons l'ensemble des N données que l'on numérote de 0 à $N-1$ (x_0, x_1, ... x_{N-1}) et empilons-les pour former une matrice :

$$\begin{pmatrix} x_{0,0} & x_{0,1} & ... & x_{0,P} \\ x_{1,0} & x_{1,1} & ... & x_{1,P} \\ & & ... & \\ x_{N,0} & x_{N,1} & ... & x_{N,P} \end{pmatrix}$$

Multiplions cette matrice par le vecteur des poids et nous obtenons le potentiel du perceptron pour chacun des données. Nommons X la matrice ci-dessous, Xp fournit un vecteur dont les éléments sont ces potentiels : $v = Mp$.

Tout cela pourrait n'être que de petites manipulations algébriques pour simplifier les notations mathématiques. Mais les conséquences de ces petites manipulations algébriques sont importantes en terme de calcul. En effet, il existe des microprocesseurs qui sont conçus et organisés pour réaliser ce type de calcul matriciel de manière très efficace. Aussi, exhiber le fait que calculer le potentiel d'un perceptron correspond au produit d'une matrice par un vecteur entraîne le fait que l'on peut tirer partie de ce type de processeurs pour accélérer les calculs nécessaires à l'utilisation de réseaux de neurones. Quand on doit effectuer ce type de calculs pour des milliers de données, des milliers voire des millions

de perceptrons, des milliers voire des millions d'épisodes, gagner du temps est crucial. C'est justement cela qui fait le succès actuel des réseaux de neurones dits « profonds » et le succès commercial des GPU. Les GPUs ont été initialement conçus pour effectuer ce type de calcul matriciel très rapidement pour générer des images dans des jeux vidéos (le G de GPU signifie *graphical*). À l'heure actuelle, si les GPUs demeurent utilisés pour générer des images, ils sont surtout utilisés pour calculer la sortie des réseaux de neurones.

Index

Table des matières