# Réseau de neurones

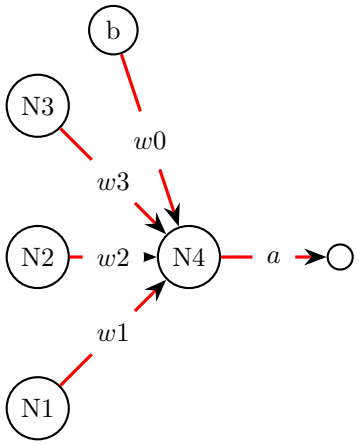
Corentin Behuet

Octobre 2019

## 1 Le réseau de neurones

### 1.1 Le neurone

Un neurone est une entité qui s'activera en fonction de certaine valeur et comme une neurone biologique. Il est relié au autres, via différentes connexion, il y'a tout d'abord celle qui sont pondéré c'est à dire avec un poids affecté à ça connexion avec le neurone précédent, et de plus il y'a une connexion supplémentaire le biais. Le biais est une valeur qui se rajoute pour corriger la somme pondéré, il y'a un seul biais par neurone. La somme pondéré est la pré activation du neurone, c'est à dire le moment ou il calcul la somme pondéré. Une fois cette valeur calculer, on applique la fonction d'activation, on appelle cette phase l'activation. Nous détaillerons plus loin les fonctions d'activations, car il en existe de plusieurs type en effet elles doivent remplir un cahier des charges précis. On considére un neurone étaint, quand il renvoie un zero.




*Neurone avec ces différentes connexion*

Ici N4,N3,N2,N1 désigne des neurones. On voit les différentes connexions qui relie le neurone, le résultat $a$ est issus des formules suivantes on commence par la somme pondéré :
$b * w0 + a1 * w1 + a2 * w2 + a3 * w3 = k$
que l'on peu généraliser à :
$k = \sum_{i=1}^{n} b * w0 + ai * wi$
un fois cette somme pondéré calculer, on appelle la fonction d'activation $\sigma$. Ici on fera donc $\sigma(k) = a$, et une fois ce $a$ calculer on l'envoie au autres neurones.

### 1.2 La fonction d'activation

La fonction d'activation est la fonction qu'on applique à la somme pondéré du réseaux de neurones. Cette fonction d'activation assez singulière doit renvoyer un résultat qui peut être borné ou non, la fonction d'activation $\sigma$ va donc de $\mathbb{R}$ l'ensemble des réels vers l'intervalle $\mathbb{R}$ ou seulement une partie de $\mathbb{R}$ comme par exemple la sigmoide qui va de $\mathbb{R}$ vers [0,1].
On peut donc écrire $\sigma : \mathbb{R} \to \mathbb{R}$
Cependant prendre une fonction avec plage de sortie bornée, sera plus table lors de l'utilisation du gradient mais aura un potentiel plus intéressant mais un plus grand nombre de poids.
Elle doit être différentiable pour pouvoir utiliser des techniques d'apprentissage basé sur le gradient. On accepte aussi le fait qu'elle soit non-linéaire.
Une fonction est dites seuillé si un neurone peut être éteint, c'est à dire que le biais fait office de seuil, car si la somme pondéré à une valeur insuffisante le neurone renverra zéro donc sera éteint.
Voici une petite liste de quelque fonction d'activation classique, pour le reste de ce document nous dirons que $\sigma$ est la fonction sigmoïde d'une part c'est valeur savent resté pondéré et surtout elle à une dérivé très simple ce qui nous simplifiera la tache pour les calculs. Pour des raisons de lisibilité je mettrais le graphe représentatif des fonctions en annexes.

— **Fonction sigmoïde :** C'est sans doute la plus connue de toute, cette fonction de :
$\sigma : \mathbb{R} \to \mathbb{R}$
$\sigma(x) = \frac{1}{1+e^{-x}}$

Elle permet de faire les techniques d'apprentissage par gradient, donc différentiable et donc dérivable. Ça dérivé est de la forme :
$\sigma'(x) = \sigma(x)(1 - \sigma(x))$
Cependant elle perd en efficacité avec trop de couches de neurones, et on lui préféra donc des fonctions plus efficaces et stable (problème de fuite du gradient).

— **Fonction Marche/Heaviside :** C'est un fonction dites seuillé et non linéaire, c'est l'un des premières utilisé mais elle présente pas mal de défaut dont celui de répondre seulement 1 quand le seuil est franchis sinon 0.
Cela limite les effets de l'apprentissage par gradient, cette fonction à été abandonné au profit d'autre fonction plus efficace. Elle se présente sous cette forme la :
$\sigma(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases}$
On voit bien son appellation de marche, avec cette non linéarité.

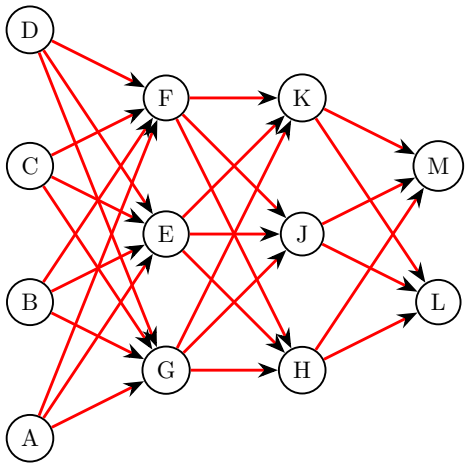
— **Fonction Tangente Hyperbolique :** Fonction usuelle elle présente l'avantage de pouvoir etre boorné et être dérivable ce qui en fait une candidate idéale, ça dérivé reste assez simple un peu à l'image de la sigmoide
Elle est définis comme telle :
$\sigma(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
Et de dérivé : $\sigma'(x) = 1 - f(x)^2$ $(-1, 1)$

— **Fonction Unité de Rectification Linéaire (ReLU) :** Cette fonction est seuillé et non linéaire
Elle est définis de cette manière, $\sigma(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ x & \text{si} \quad x \geq 0 \end{cases}$
Elle possède une dérivé elle aussi assez simple et utilisable avec le gradient ce qui est intéressant
$\sigma'(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases}$
Elle renvoie des valeurs sur l'intervalle : $[0, \infty[$
Elle existe en plusieurs variante comme la paramétrique (PReLU).

— autre possibilité de fonction gaussienne, sinusoïdale, douce, courbé etc...

## 1.3 Le réseau de neurones

Un réseau de neurones est un amas de neurone relié entre eux, dans les faits le réseau de neurone est organisé, il est constitué de 3 couches bien différentes. Une représentation graphique souvent admises serait celle si :



*Réseau de neurone*

Dans cette représentation A, B, C, D, E, F, G, H, J, K, L, M représente chacun un neurone du réseau, pour des raisons de clarté ils sont nommé par des lettres, et les biais ne sont pas représenté, car sur cette représentation ils représentent quand même 10 connexions supplémentaire. Chaque arc entre les neurones à bien sur un poids attribué, et on commence à voir que même ici pour un réseaux de neurones assez limité on a déjà 36 poids, ce qui est quand même non négligeable

Une couches correspond à un nombre $n$ de neurones sur une même ligne. Nous allons maintenant voir comment son organisé les couches du neurones :

— **Couche d'entrée :** La couche d'entrée qui porte bien son nom, en effet elle fait le lien entre les donnés qui passeront par le réseau de neurone et le réseau de neurones. Les données arrivent par ces neurones qui vont ensuite les envoyer aux autres. Les donnés sont transformé en une seule valeur et ensuite est transmises à la couche suivante.En suite elles sont envoyé dans le réseau, et passe par les différentes couches du réseau. Cette couche particulière possède des biais mis à 0, car il n'y a pas besoin de biais pour elle.

— **Couche caché :** Les couches caché car il peut y en avoir plusieurs, chaque couches de neurones caché composé de $n > 0$, les neurones d'une couches $k-1$ sont tous relié au neurones d'une couches

$k$, on dénombre donc $N_{k-1}$ le cardinal de la couche $k-1$ , et de la même manière on définis $N_k$ le cardinal de la couche $k$. On aura donc $N_{k-1}$*$N_k$ connexion entre chaque couche, plus les $N_{k-1}$+$N_k$ biais. C'est ici qu'un majeur partis des calculs se fera. C'est la dernière couches de neurones caché qui communiquera ces résultats à la couche de sorties.

— **Couche de sortie :** La couche de sortie est la dernière couche du réseau de neurones, c'est cette couche qui nous transmettra les résultats du réseaux de neurones. C'est avec celle ci qui pourra nous aider à déterminer le résultat en fonction de l'entrée.

Les réseaux de neurones sont linéaires, les neurones ne communiquent que dans un seul sens. on peut les considéré comme une seule fonction pour simplifié ça représentation. En effet on peut considérer un réseau comme étant une fonction de $\mathbb{R}^n$ dans $\mathbb{R}^k$, on aura tendance à considérer comme une boite noire. On a donc un neurone qui est une fonction qu'on peut améliorer avec l'apprentissage.

# 2 Apprentissage

L'apprentissage va être la façon qu'un réseau de neurones se paramètre, en effet l'apprentissage permet de réattribuer les poids sur les différentes connexions. Le but sera d'avoir une série de test auquel on connais les résultats et de le confronté au réseau de neurones, une fois terminé le réseau de neurone sera (moyennant une certaine erreur quand même) de prédire les résultats pour un jeu de donné ,sur lequel il à été formé (détection de lettre sur une photos, de phrase etc...).

## 2.1 Fonctionnement de l'entraînement

Un réseau de neurones va s'entraîner en utilisant une jeu de donnée d'apprentissage dont on connais le résultats, le but du jeu va être de rééquilibrer les poids et les bais en fonction des résultats du jeu d'entraînement. Donc on va avoir des biais et des poids initialisé au hasard. Le jeu de donné d'entraînement est déterminé par des experts, ce jeu peut être de nombreux types , et il faudra souvent crée un lien entre donnée fournis et ce qu'accepte un réseau de neurone. De plus, plus un réseau de neurones apprend plus il deviens bon dans ces prédictions (évitons quand même le sur apprentissage) mais un réseau de neurones bien entraîné fera beaucoup moins de fautes lors ce qu'il faudra prédire des choses. Un réseau de neurones qui a une infinité de donné apprentissage théoriquement fera 0% d'erreur. C'est de la théorie car on aura jamais un tel jeu de donnée, mais on a des exemples que plus on a de donnée d'entraînement il devient performant. Mais malgré tout ça, on a pas de chiffre qui nous dit à partir de combien de donnée d'entraînement le réseau deviens performant.
On voit donc la place primordiale des donnés d'entraînement dans l'apprentissage du réseau de neurones. Mais on peut se demander comment augmenter ce nombre de donnée ? et bien il existe aussi des techniques pour entraîné leur nombre de manière "artificiels", en effet pour quelqu'un souhaitant paramétré un réseau de neurone pour une application il peut être intéressant d'avoir plus de donnée pour augmenter la sûreté des prédictions. Et par exemple pour augmenter ce nombres, sur une application déterminant si une image contient un chat, il est utile de retourner l'image comme dans un miroir car un chat reste un chat quelque soit l'orientation, la couleur, la dimension, la luminosité, etc...
Maintenant nous allons nous attardé sur le fonctionnement de cette apprentissage via une fonction de perte, car avant qu'un réseau de neurones fasse des prédictions il y'a la période d'apprentissage ou il faudra rectifié ces erreurs.

## 2.2 Fonction de perte

La fonction de pertes, sera la fonction issus de la différence entre le résultats d'entraînement et le résultat obtenus et ce résultats sera ensuite changer selon différent type de fonction d pertes, mais nous utiliserons dans ce document la plus simple la distance quadratique.
voici les résultats obtenus après un calcul du réseau en fonction d'une donnée d'entraînement, nous le généraliserons en le présentant comme le vecteur suivant :

$A = \begin{pmatrix} a1 \\ a2 \\ ... \\ a_{k-1} \\ a_k \end{pmatrix}$ On peut avoir par exemple $\begin{pmatrix} 0.1 \\ 0.02 \\ 0.89 \\ 0.1 \end{pmatrix}$ comme résultat d'un réseau de neurones après avoir utilisé une donné J sur ce réseau on récupère de chaque neurone de sortie ça valeur (après activation ou non).
On le compara ensuite au vecteur du résultat attendue, le vecteur Solution :

$S = \begin{pmatrix} s1 \\ s2 \\ ... \\ s_{k-1} \\ s_k \end{pmatrix}$ On peut avoir par exemple $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$ comme résultat d'un réseau de neurones.
La différence donnera le vecteur $E$ l'erreur (ou la distance) entre le résultat et ce qu'on attend réellement :

$$E = \begin{pmatrix} a1 \\ a2 \\ ... \\ a_{k-1} \\ a_k \end{pmatrix} - \begin{pmatrix} s1 \\ s2 \\ ... \\ s_{k-1} \\ s_k \end{pmatrix} = \begin{pmatrix} a1 - s1 \\ a2 - s2 \\ ... \\ a_{k-1} - s_{k-1} \\ a_k - s_k \end{pmatrix}$$

On peut réutiliser ce vecteur en utilisant les indices, et en faisant la somme quadratique :

$$\text{Peut s'écrire : } \frac{1}{m}\sum_{i=0}^{m} e_i^2 \text{ où autrement exprimer : } \frac{1}{m}\sum_{i=0}^{m}(a_i - s_i)^2$$

Cette somme va être appeler la distance quadratique (raccourcis par MSE)
Elle est élevé au carré pour permettre de jouer avec des écarts positifs et celle permet dans certain cas d'avoir une meilleur idée de la distance. On essaiera de minimiser cette fonction, pour avoir des indices les plus petit possible, en effet vus qu'elles correspond à la distance observé entre les données obtenue et celle à prédire il faut donc rétrécir au maximum cette distance donc la minimiser. On peut aussi l'appeler fonction de coût ou objectif
Dans les faits une fois l'apprentissage terminé et le réseau faisant une erreur tolérable on considère le max du vecteur de sortie pour coller au lieux aux objectif.

## 2.3 La descente par gradient

Une fois notre fonction de perte calculer nous allons utiliser différentes méthodes d'apprentissages, en effet il en existe plusieurs. On utilise classiquement les méthodes d'apprentissage par gradient qui sont efficace pour réattribuer les poids rapidement et de manière assez convergente car le gradient est un moyen de calcul efficace qui demande explique les choix de la fonction d'activation . Ce calcul de descente par gradient et la partis la plus coûteuse de l'apprentissage et on peut le voir sous forme de calcul vectoriel, c'est pour ça que nous utiliserons la gpu pour paralléliser le calcul.
Pour cela nous utilisons la rétropagation par gradient, en effet le gradient va donner des valeurs en fonction de la pente, en effet plus la pente est grande plus la valeur du gradient sera importante, donc on prend l'inverse du gradient est qu'on fait le calcul de changement sur les poids. Donc tant que l'erreur est grande, on a une grande pente donc on converge rapidement vers un minimum local, et dans un temps infinis on convergerai vers le minimum global. Et quand l'erreur est de plus en plus petite, on converge par petit coup vers le minimum ce qui garantit une certaine stabilité.
Bon maintenant nous allons voir la partie technique, introduisons quelque notations. Commençons par décrire la matrice des poids qui contient tout les poids de la couche $l$ du réseau de neurone $W^l$,Chaque $w_{i,j}^l$ est un des poids du réseau.
$l, i, j$ correspondant à :
$l$ étant la couche du neurone, $i$ le numéro de neurone et $j$ le poids vers le $j$ ème neurone.
Matrice des poids de la $m$-ème colonne :

$$W^m = \begin{pmatrix} w_{0,0}^m & w_{0,1}^m & ... & w_{0,k-1}^m & w_{0,k}^m \\ w_{1,0}^m & w_{1,1}^m & ... & w_{1,k-1}^m & w_{1,k}^m \\ ... & ... & ... & ... & ... \\ w_{n-1,0}^m & w_{n-1,1}^m & ... & w_{n-1,k-1}^m & w_{n-1,k}^m \\ w_{n,0}^m & w_{n,1}^m & ... & w_{n,k-1}^m & w_{n,k}^m \end{pmatrix}$$

le changement des poids ce fait en suivant cette formule, ou fonction de correction : $\alpha$ tant le taux d'apprentissage, plus le taux d'apprentissage est petit (tend vers 0) plus le réseau de neurone apprendra lentement. Les $b_i^l$ sont les différents biais d'un réseau de neurone, la notation des $l, i$ est toujours la même.

$$w_{i,j}^l = w_{i,j}^l - \alpha * \frac{\partial E}{\partial w_{i,j}^l} \tag{1}$$

$$b_i^l = b_i^l - \alpha * \frac{\partial E}{\partial b_i^l} \tag{2}$$

La notation obscur de $\frac{\partial f(x)}{\partial x}$ correspond à la dérivé partiel de $f(x)$ en fonction de $x$.
En effet le gradient correspond aux dérivé partiels successives en fonction des coordonnées (ici les différentes combinaison de $l, i, j$

## 2.4 Fonctionnement de la descente par gradient

Comment ça fonctionne :
Pour calculer la façon donc chaque poids sera corriger on fait, la formule (1) ci dessus :

$$w_{i,j}^{l} = w_{i,j}^{l} - \alpha * \frac{\partial E}{\partial w_{i,j}^{l}}$$

Hors il faut déjà comprendre à quoi correspond le $\frac{\partial E}{\partial w_{i,j}^{l}}$
pour cela on utilisera le théorème de dérivation des fonctions composées.

$$\frac{\partial E}{\partial w_{i,j}^{l}} = \frac{\partial k_{i}^{l}}{\partial w_{i,j}^{l}} \frac{\partial a_{i}^{l}}{\partial k_{i}^{l}} \frac{\partial E}{\partial a_{i}^{l}}$$

atre probleme lié au surappprentsisage des reseaux de neurones (overfeet), c'ets quand le reseau de neurone apprend trop avec le jeu de donnée fournis, on a pour ça differente valeur pour aider à eviter le surapprentssage, la désactivation de certain neurone à chaque epoch, les mini bachs, les corretctions de surapprentissage, en effet pour comprendre ça on peut essayer d'imaginer avec des fonctions en 2d, enfin des polnomes pour être precis : :image : :