Algorithme du Deep learning

Mathieu Tourrette

2021–05–17

Contents

1 Introduction

Le Deep learning est un domaine du machine learning. Le principe reste le même que dans la machine learning. Le but de l'utilisation du deep learning est de pouvoir coller à des modèles de donner non lineaire.

Nous pouvons retrouver different modèle de deep learning dans la reconnaissance de nombre manuscrit ou le modèle va attravers les fonctions reconnaitre des formes et sortir une probabilité du nombre qu'il pense juste. Nous avons des données, un modèle, un algorithme d'optimisation et enfin une fonction de minimisation de l'erreur.

Definition 1 (Deep learning)**.** *Développer un modèle, en se servant d'un algorithme d'optimisation pour minimiser les erreurs entre le modèle et nos données.*

2 Construction

Un réseau de neurones est une multitude de fonctions reliées entre elles par leur I/O. Ce principe nous permet de conceptualiser des tâches complexes. Il s'agit d'un perceptron multicouche. Voyons ce qu'est un simple perceptron.

2.1 Perceptron

Le perceptron est n'a qu'un seul neurone. Il n'a qu'une seule couche. C'est un classifieur linéaire. Il sort exclusivement 1 si le seuil d'activation est supérieur au biais θ sinon 0.

$$o = f(z) = \begin{cases} 1 & \text{si} \quad \sum_{i=1}^{n} w_i x_i > \theta \\ 0 & \text{sinon} \end{cases}$$

n : entrées
w : coefficients synaptiques
θ : biais

Lemma 2.1 (Théorie de Hebb)**.** *Lorsque 2 neurones biologiques sont excités conjointement alors ils renforcent leur lien synaptique.*

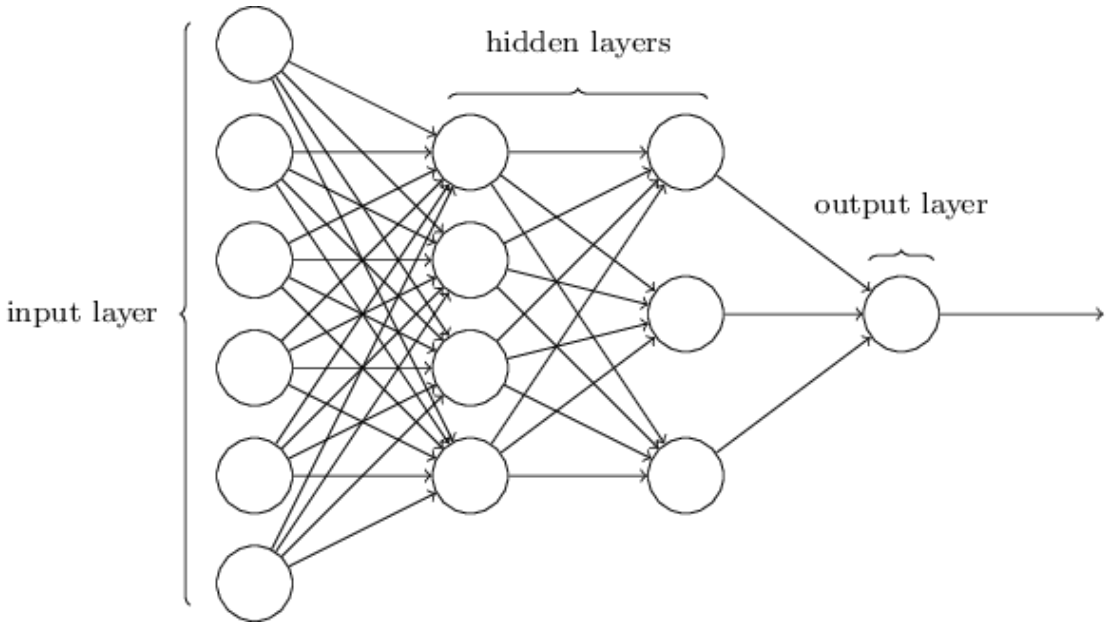


Figure 1: Schéma neuronal Network

Nous pouvons voir la formule permettant de renforcer un lien synaptique entre deux neurones ci-dessous :

$$W = W + \alpha(y_{true} - y)X \tag{1}$$

α : vitesse d'apprentissage
y_{true} : sortie de référence
y : sortie produite
X : entrée du neurone

2.2 Constitution d'un neurone

Notre *neurone* est en réalité une fonction qui prend x paramètres en entrée et qui retourne une sortie y.

2.2.1 Fonction d'agrégation

Dans cette fonction, nous retrouvons une fonction d'agrégation :

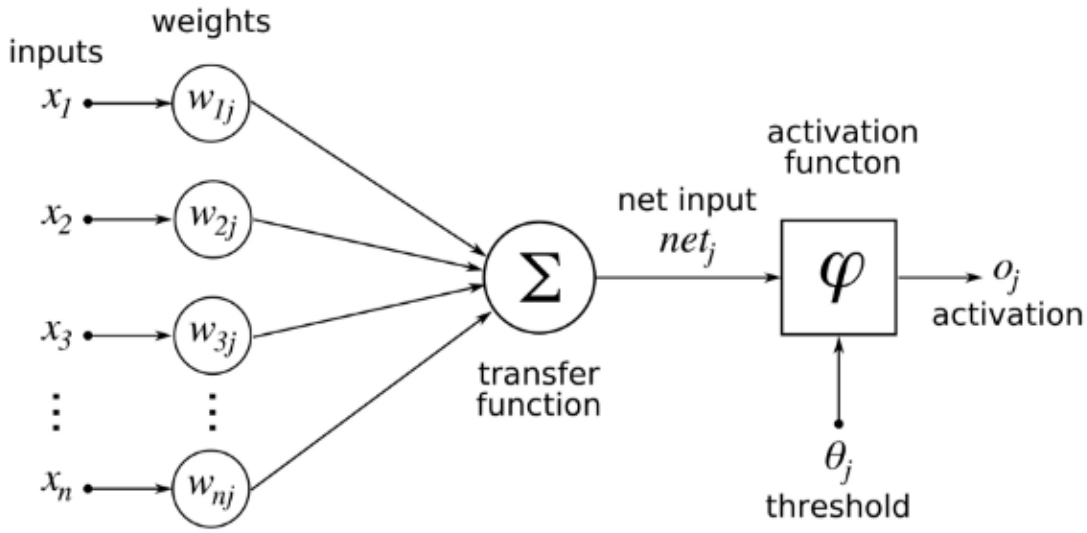


Figure 2: Schéma fonction agrégation

$$\sum_{j=1}^{n} x_j w_{ij} \tag{2}$$

Nous pouvons utiliser d'autres types de fonction d'activation. Nous avons également des fonctions linéaires ou encore des sommes, comme montrées dans notre exemple.

Or, nous voulons que notre réseau de neurones puisse résoudre des systèmes complexes. Par conséquent, celui-ci ne doit pas reposer sur un modèle linéaire. Nous allons donc introduire des fonctions d'activation non linéaires.

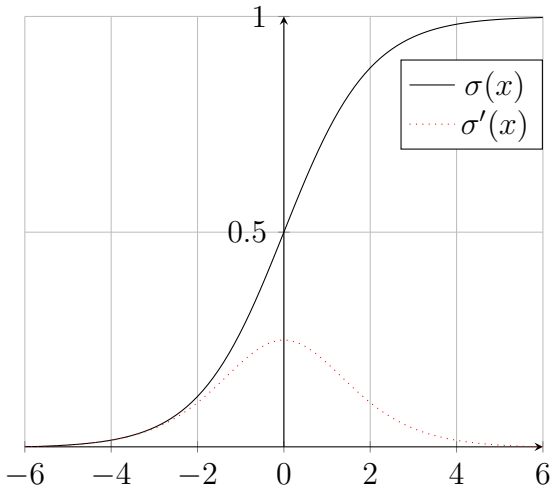
2.2.2 Fonction d'activation

Les fonctions d'activation sont présentes pour dire si le neurone doit être activé ou pas. Pour cela, cette fonction doit nous retourner 1 ou 0.

2.2.2.1 Fonction logistique (sigmoïde)

Pour la fonction sigmoïde, il s'agit de l'utiliser pour calculer la probabilité que x appartienne à la classe positive dans une classification binaire.

Cela fonctionne très bien pour une classe. En revanche, s'il y a des multi-classes, la fonction ne nous retournera pas forcement 1. Un autre problème se pose sur les entrées extrêmement négatives. La sortie de la fonction logistique sera alors très proche de 0. Ceci entrainera, par conséquent, un apprentissage lent pour notre algorithme, mais pourrait également se retrouver sur un minimum local.



2.2.2.2 Fonction softmax

Nous avons vu que dans la fonction logistique, nous rencontrons un problème pour une classification multi-couches plus que binaire.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{3}$$

Si nous calculons l'entrée nette et l'utilisons comme fonction d'activation, la valeur que nous obtenons peut-être interprétée comme une probabilité de classe. Les probabilités de classe prédites totaliseront 1.

2.2.2.3 Fonction de la tangente hyperbolique (Tanh)

La fonction tanh est le plus souvent utilisée dans les couches cachées parce qu'elle procure un plus grand spectre de sortie et un intervalle sur [-1, 1].

Toutes ces fonctions d'activation appartiennent à la même famille que la fonction sigmoïde mais elles ont toutes le problème de disparition des gradients. A cause de cela, l'apprentissage devient de plus en plus lent.

$$\text{Tanh}(x) = 1 - \frac{2}{\exp(2x) + 1} \tag{4}$$

Pour y remédier, la fonction d'activation d'unité linéaire rectifiée (ReLU) vient résoudre ce problème.

2.2.2.4 Fonction d'unité linaire rectifiée (ReLU)

Elle permet de régler le problème de la fuite de gradient. Elle est non linéaire. Elle est la fonction la plus adaptée pour l'apprentissage profond.

$$f(x) = max(0, x)$$

3 Forward propagation

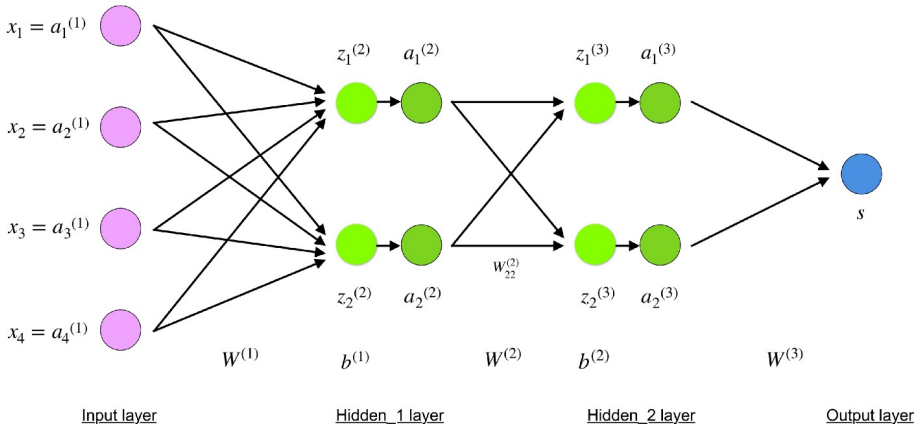


Figure 3: Exemple neural network

Sur le schéma ci-dessus, les neurones colorés en violet représentent nos entrées. Ainsi, nous pouvons retrouver des scalaires, des complexes, des vecteurs ou des matrices multidimensionnelles. Tout au long de la présentation, nous utiliserons la représentation matricielle.

La fonction de la couche 2 se note ainsi :

- $l = 2$

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$

- $l = 3$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$a^{(3)} = f(z^{(3)})$$

w : Poids des couches
b : Biais des couches

Les $a^{(2)}$ et $a^{(3)}$ sont les résultats d'une des fonctions d'activation que nous avons vue en amont. Nous répétons cela sur l'ensemble des neurones cachés.

Pour la couche de sortie que nous voyons en bleu, la solution se présente comme cela :

$$s = W^{(3)}a^{(3)}$$

Voici un résumé rapide de la forward propagation :

$$x = a^{(1)} \qquad \textit{Input layer}$$
$$z^{(2)} = W^{(1)}x + b^{(1)} \qquad \textit{neuron value at Hidden}_1 \textit{ layer}$$
$$a^{(2)} = f(z^{(2)}) \qquad \textit{activation value at Hidden}_1 \textit{ layer}$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \qquad \textit{neuron value at Hidden}_2 \textit{ layer}$$
$$a^{(3)} = f(z^{(3)}) \qquad \textit{activation value at Hidden}_2 \textit{ layer}$$
$$s = W^{(3)}a^{(3)} \qquad \textit{Output layer}$$

Figure 4: Résumer forward propagation

3.1 Fonction coup

Maintenant que notre modèle nous a fournis des résultats (x, y) peu précis, il va falloir ajuster ces erreurs. Afin de savoir qu'elle est ce degré d'erreur, nous allons devoir utiliser une fonction coup.

Il existe plusieurs types de fonction coup. Nous pouvons utiliser l'erreur quadratique moyenne, la cross entropy ou tout autre fonction de coup.

4 Back propagation

Le principe de la back propagation est de déterminer comment la sortie du réseau varie en fonction des paramètres (w, b) présents dans chaque couche et donc de pouvoir améliorer la précision. Après chaque propagation vers l'avant, il effectue une propagation vers l'arrière pour ajuster les paramètres du modèle.

4.1 Calcul de gradient

Le calcul de gradient va nous permettre de nous donner la direction pour minimiser notre fonction.

$$\frac{\partial C}{\partial x} = [\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_m}]$$

4.1.1 Chaine de gradients

La chaine de gradients nous permet de repropager vers l'arrière. Nous calculons alors les dérivés partiels à la chaine, comme présenté ci-dessous :

$$\ldots \leftarrow \frac{\partial f_3}{\partial f_2} \leftarrow \frac{\partial f_4}{\partial f_3} \leftarrow \frac{\partial f_4}{\partial f_3} \leftarrow \frac{\partial f_{final}}{\partial f_4}$$

Grace aux gradients, on peut alors mettre à jour les paramètres (w, b) de chaque couche de telle sorte à ce qu'ils minimisent l'erreur entre la sortie du modèle et la réponse attendue.

Afin d'améliorer la précision de notre modèle et de se rapprocher de la perte et d'un coût minimums, nous devrons recommencer un cycle de propagation.

5 Conclusion

Nous avons donc vue succintement une mise en contexte de l'utilisation d'un algorithme de deep learning. Pour finir que faut t'il retenir de cette article ?

La construction d'un algorithme de deep learning ce fait:

- Par succession de fonction appeler couche ou les entrées/sorties sont reliées entre elle.
- Une fonction d'activation non linaire
- Forward propagation avec une fonction coup
- Back propagation pour ajuster notre modèle avec la descente de gradient

6 Annexe

- Types of activation functions
- Activation function for multi-layer neural networks
- Backpropagation
- Backpropagation algorithme
- Backpropagation Main Ideas
- Latex :)