# Statistiques multidimensionnelles

## Réseaux de neurones (NN)

Benjamin André, Alexis Lecocq

UMONS
Faculté des Sciences
BA 3 Sciences Informatiques


Juin 2017

# Machine Learning

Lorsque nous voulons résoudre un problème algorithmiquement, nous devons donner la marche à suivre exacte du programme qui mène à une solution.
Dans certains cas, celle-ci est trop complexe que pour être écrite de cette façon, et les paramètres d'entrée (position, température, couleur,..) peuvent influencer le résultat de façon très complexe.
L'approche du machine learning propose de trouver une solution approchée en minimisant l'erreur.

# Machine Learning

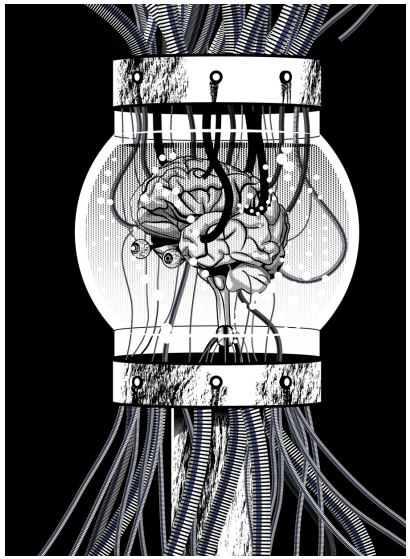
## Utilisation

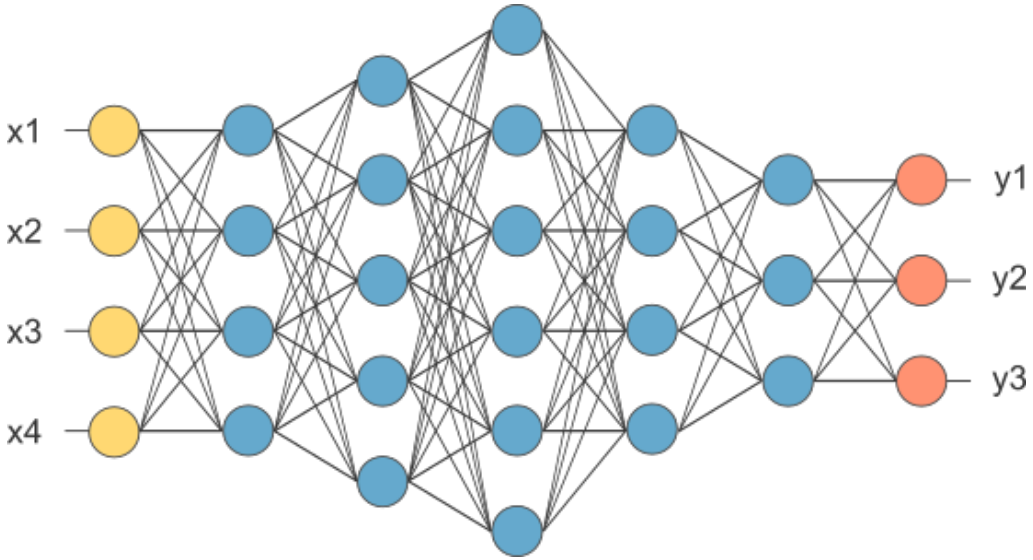
Comme expliqué précédemment, il s'agit d'une solution approchée. Dans le cas ou il existe un algorithme efficace il sera préférable de l'utiliser.

## Exemples

- AlphaGo
- DeepBlue
- Voitures autonomes
- Filtre à spam
- Reconnaissance d'image
- Traduction
- Prévision de la bourse

# Neural Networks





# Neural Networks

## Types de réseaux de neurones

- Deep Neural Network (DNN) : Réseau avec un grand nombre de couches.
- Convolutional Neural Network (CNN) : Inspiré du fonctionnement de la vue, les couches de neurones ici sont à plus d'une dimension.
- Recurrent Neural Network (RNN) : Réseau où la sortie est réintroduite dans le réseau pour les tests suivants : effet mémoire.

=> Nous nous concentrerons sur les DNN, avec peu de couche dans l'exemple (3).

# Fonctionnement global

Dans le cas où on a $n$ paramètres, comment trouver $f(x_1, x_2, ..., x_n)$ ?
Si la fonction est trop compliquée pour être écrite directement, on prend des *exemples*, où l'on connaît déjà l'image par notre fonction pour certaines entrées. Typiquement, des milliers (millions). Ces exemples, nous les donnons à un réseau de neurones, constitué des fonctions mathématiques simples reliées par des arcs pondérés. Ils jouent le rôle *d'axiome*.
Nous donnons les exemples au réseau qui ajuste les poids pour approximer au mieux la fonction, tout en la généralisant.
Maintenant, en lui présentant des variables, le réseau peut *prédire* $f(x_1, x_2, ..., x_n)$.

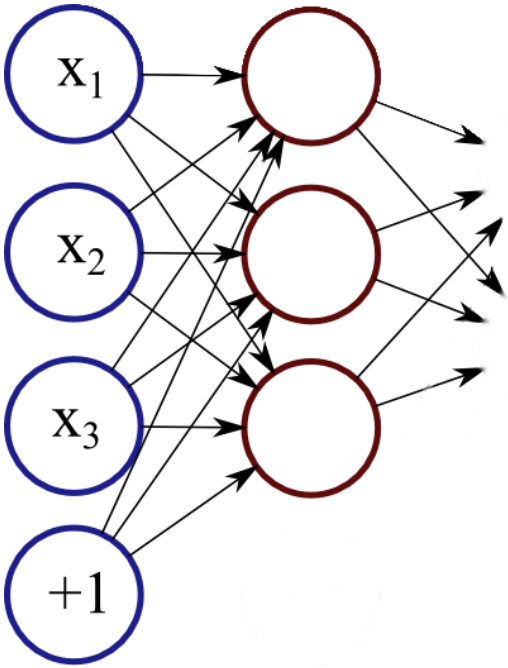
# Structure des couches

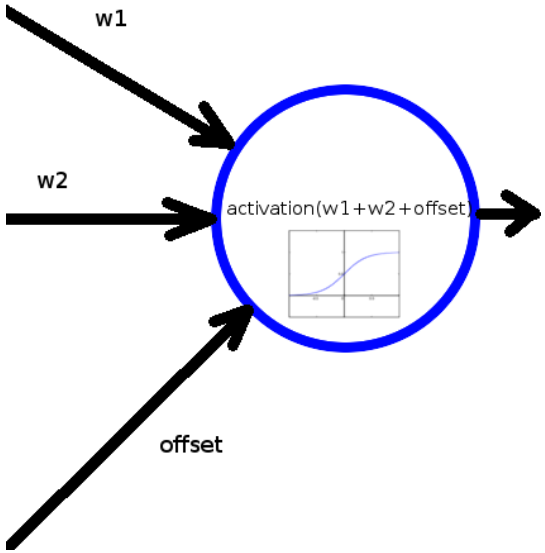
3 types de couches :

- L'entrée : Couche correspondant aux variables à évaluer
- Les couches "cachée" : Couches correspondant au comportement interne du réseau, le nombre de couches dépend du problème, et il n'y a pas de règles prédisant le nombre de couches nécessaires. De nombreuses opérations mathématiques se font ici.
- La sortie : Couche retournant un nombre correspondant soit à une classe, soit à la valeur d'évaluation des paramètres d'entrées

Tous les neurones (décrits plus tard) sont dans une couche et une seule. Ceux-ci sont interconnectés avec tous les neurones de la couche précédente et tous ceux de la suivante, et rien d'autre (parfois avec un poids 0 cependant).

# Couche cachée



# Fonction d'activation

Chaque neurone reçoit les valeurs de la couche précédente, ainsi qu'un offset, qui est propre à toute la couche actuelle. Il en effectue une transformation linéaire (ici, avec les poids 1, 1, 1), mais on pourrait très bien imaginer $activation(0.2 * w1 + 1.4 * w2 - 2.1)$. Le soucis, c'est qu'on pourrait vite finir avec des résultats très grand, si notre fonction d'activation ne présente pas de caractéristique spécifique : être bornée sur le domaine des images (et continue).
La fonction d'activation appliquée sur la transformation linéaire la plus répandue est la sigmoïde, qui retourne un résultat entre 0 et 1 :

$$\frac{1}{1 + e^{-x}}$$

# Backpropagation

## C'est fini ?

On a un réseau de neurones, c'est bien beau, mais comment définir les poids ?

Il existe plusieurs stratégies pour les initialiser, mais il est tout à fait valable de commencer avec des poids aléatoires. Vous vous souvenez des exemples pour l'entraîner ? C'est ici qu'ils interviennent ! On nourrit le réseau avec les exemples un à un. On compare sa réponse avec le résultat attendu. Le taux d'erreur est très élevé au début.
Heureusement ! Il existe un algorithme pour ajuster les poids *en détectant lequel est le plus responsable de l'erreur*. Il s'agit de la backpropagation !

# Sources

https://www.r-bloggers.com/fitting-a-neural-network-in-r-neuralnet-package/

http://www.parallelr.com/r-deep-neural-network-from-scratch/

https://www.youtube.com/watch?v=BR9h47Jtqyw

http://doctor-morbius.deviantart.com/

https://datascienceplus.com/fitting-neural-network-in-r/