Statistiques multidimensionnelles

Réseaux de neurones (NN)

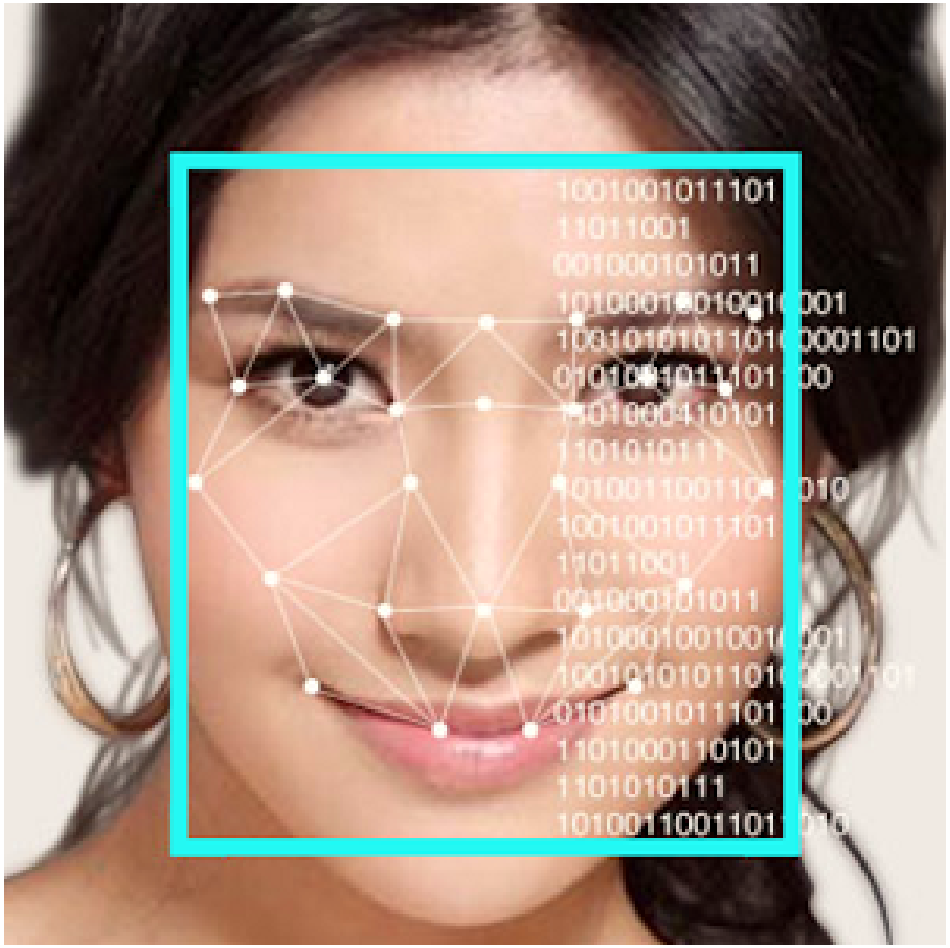
Benjamin André, Alexis Lecocq

UMONS
Faculté des Sciences
BA 3 Sciences Informatiques

UMONS Faculté des Sciences

Juin 2017

Machine Learning



Certains problèmes comme la reconnaissance faciale comportent trop de variables et des relations trop compliquées entre celles-ci pour être décrite efficacement de façon algorithmique.
On préférera le machine learning qui proposera une solution approchée grâce à l'apprentissage automatique.

Machine Learning

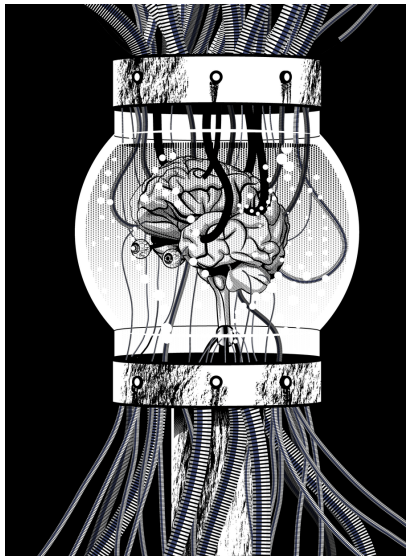
Utilisation

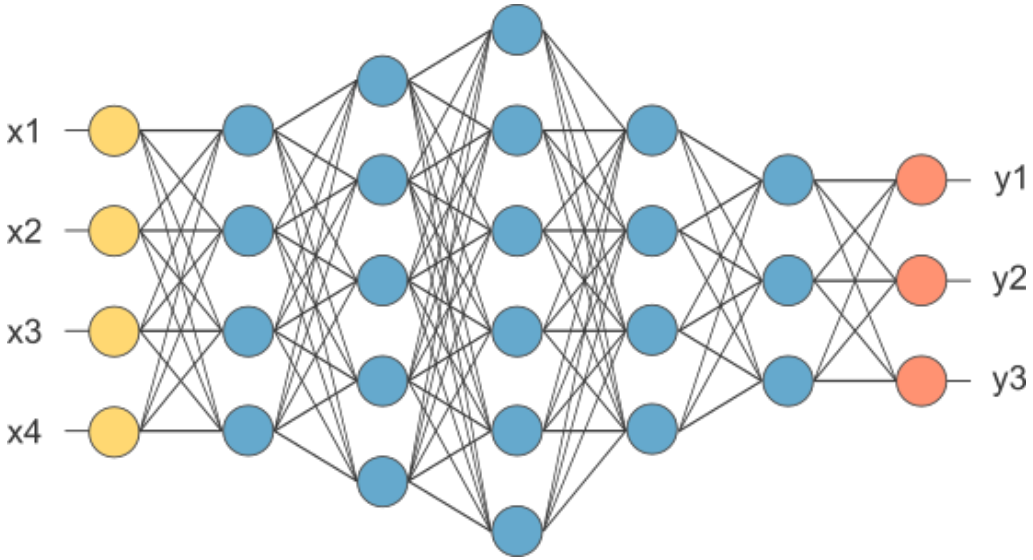
Comme expliqué précédemment, il s'agit d'une solution approchée. Dans le cas ou il existe un algorithme efficace il sera préférable de l'utiliser.

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 utiliserons une version généraliste de la technique, fonctionnant comme un DNN avec peu de couches.

Fonctionnement global

Comment trouver $f(x_1, x_2, ..., x_n)$?
On l'écrit sous la forme $f(x_1, x_2, ..., x_n) = a(w_1 * z_1, w_2 * z_2, ..., w_n * z_n)$
Où z_i pourrait lui-même être le résultat d'une fonction mathématique.
En composant les fonctions (avec un certain poids $\vec{w}$) jusqu'à des fonctions simples sur $\vec{x}$, on crée un réseau de neurones (fonctions).
A force d'exemples, on ajuste les *poids* (axones) w_i afin d'approximer f.

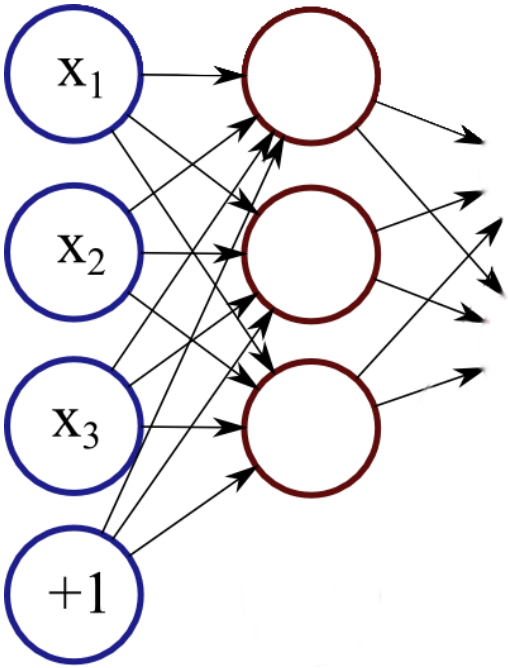
Structure des couches

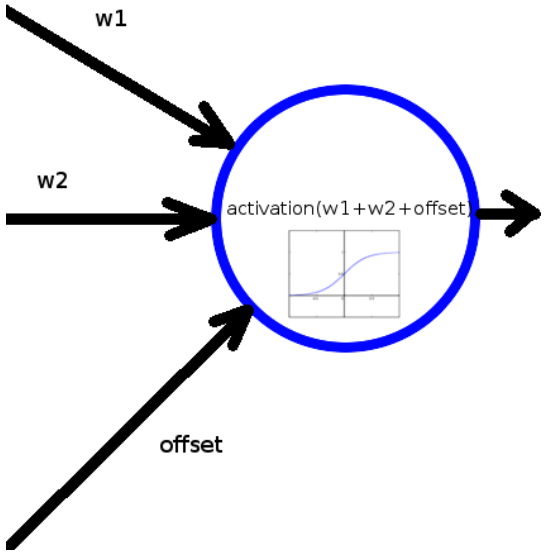
3 types de couches :

- 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

Aucun neurone (décrits plus tard) n'est dans plusieurs couches. 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 effectue la somme pondérée par w de tous les résultats de la couche précédente.
Par exemple : $(0.2 * w1 + 1.4 * w2 - 2.1)$.
Pour à cette somme, nous appliquons une fonction dite "d'activation".
La fonction d'activation appliquée sur la transformation linéaire la plus répandue est la sigmoïde :

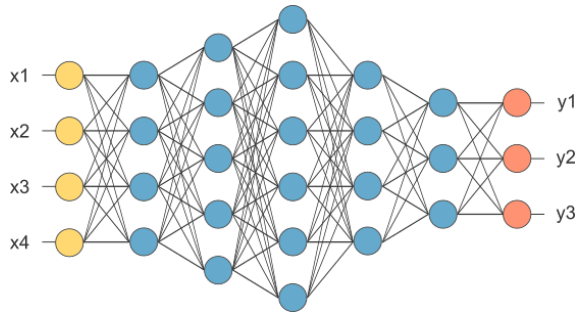
$$activation(x) = \frac{1}{1 + e^{-x}} \in [0, 1]$$

Fonction d'activation

Cette fonction a plusieurs propriété intéressantes :

- $activation(x) \in [0, 1]$, utile pour borner le résultat et éviter une explosion des valeurs dans le réseau
- Fonction continue sur $\Re$
- Elle est dérivable en $activation(x) * (1 - activation(x))$, ce qui est utilisé dans le calcul de l'erreur

Représentation



$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}, w_1 = \begin{pmatrix} w_{x_1 h_1} & w_{x_1 h_2} & w_{x_1 h_3} & w_{x_1 h_4} \\ w_{x_2 h_1} & w_{x_2 h_2} & w_{x_2 h_3} & w_{x_2 h_4} \\ w_{x_3 h_1} & w_{x_3 h_2} & w_{x_3 h_3} & w_{x_3 h_4} \\ w_{x_4 h_1} & w_{x_4 h_2} & w_{x_4 h_3} & w_{x_4 h_4} \end{pmatrix}, w_2 \in \Re^{4x5}, ..., y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Sans oublier les offset pour chaque couche, stockés séparément. On représente donc notre problème sous la forme matricielle.

Backpropagation

C'est fini ?

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

Plusieurs stratégies pour l'initialisation dont l'aléatoire. Ensuite on nourri le réseau d'exemples. On mesure le taux d'erreur entre l'approximation et les données de résultat. A chaque fois, un algorithme appelé *backpropagation* est appliqué pour ajuster les poids *en détectant lequel est le plus responsable de l'erreur*.

Backpropagation

Sans entrer trop dans les détails, ce que fait l'algorithme :

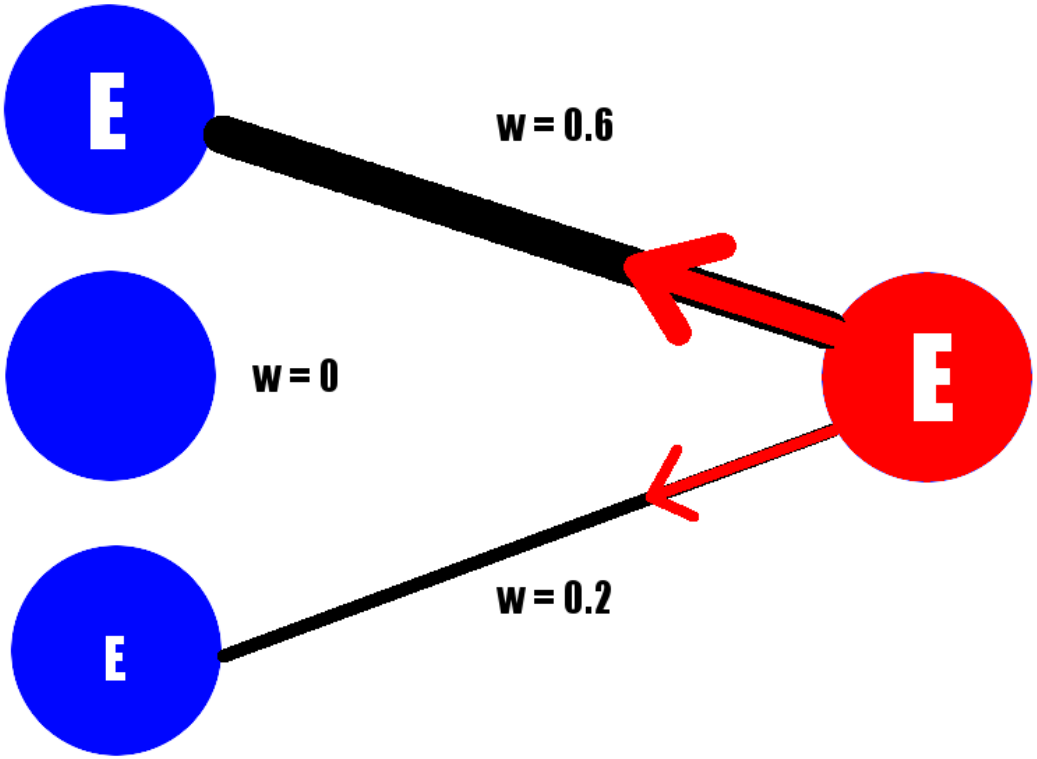
Entrées: Un jeu de données
Sorties: Les poids
$w \leftarrow$ valeurs aléatoires entre -1 et 1
répéter
  pour chaque *exemple du training set* **faire**
    pour chaque *couche dans le réseau* **faire**
      pour chaque *noeud dans la couche* **faire**
        $sum \leftarrow$ la somme des entrées
        $sum \leftarrow sum + offset$
        $noeud_{sortie} \leftarrow activation(sum)$
      fait
    fait
    pour chaque *noeud dans la couche de sortie* **faire**
      $node_{error} \leftarrow \Delta(attendu, calcul)$
    fait
    pour chaque *couche cachée dans le réseau* **faire**
      pour chaque *noeud dans la couche* **faire**
        Calculer $node_{error}$
        Mettre à jour le poids du nœud dans le réseau.
      fait
    fait
    Calculer l'erreur globale
  fait
jusqu'à *nombre d'itération maximum atteint ou l'erreur globale est passée sous un seuil fixé*;
retourner *arbre*

Backpropagation

Pour faire simple :

- On initialise les poids aléatoirement
- On présente un exemple et évalue l'erreur
- On fait remonter l'erreur en fonction des poids
- On ajuste les poids en fonction de l'erreur et continue à faire remonter l'erreur
- On donne un autre exemple, et s'arrête après un certain temps ou si le NN est efficace

Backpropagation



Choix de librairie

La librairie choisie est **neuralnet**. C'est une librairie de très haut niveau, mise à jour régulièrement, documentée, et avec une certaine communauté (qui a d'ailleurs écrit un tutoriel).

Données utilisées et objectif

On utilisera
les données "Boston" fournies
de base par RStudio. Celles-ci
mettent en relation 13 variables
sur une propriété et sa valeur.
Notre but sera
de prédire la valeur d'un bâtiment
en fonction des 13 variables.

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/https://play.google.com/store/
apps/details?id=com.seakleng.facedetection&hl=fr