Réseaux de neurones

Romain Tavenard

M2 Stats – Université de Rennes 2

Contenu du cours

- Modèles
  - Perceptron + Perceptron multi-couches
  - Réseaux de neurones convolutionnels (CNN)
  - Réseaux de neurones récurrents (RNN)
- Librairie `keras`

Retour sur la régression linéaire

- Régression linéaire aux moindres carrés ordinaires

$$y_i = \underbrace{\sum_j \beta_j x_{i,j}}_{\beta X_i} + \epsilon_i$$

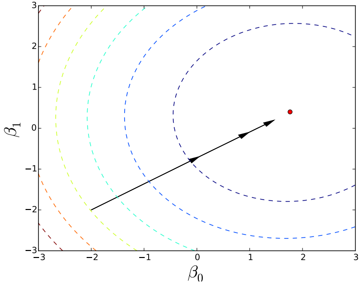
- Optimum connu : que ferait-on sinon ?

Retour sur la régression linéaire

- Régression linéaire aux moindres carrés ordinaires

$$y_i = \underbrace{\sum_j \beta_j x_{i,j}}_{\beta X_i} + \epsilon_i$$

- Optimum connu : que ferait-on sinon ?



Retour sur la régression linéaire – 2

- Comment mettre en œuvre cette descente de gradient ?
  1. Se fixer une fonction de coût à minimiser

$$L(\beta) = \sum_i (y_i - \beta X_i)^2$$

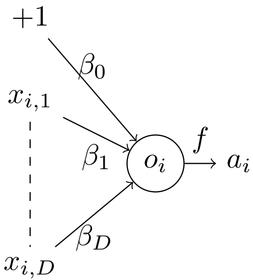
  2. Exprimer le gradient de cette fonction de coût

$$\forall j, (\nabla L(\beta))_j = \frac{\partial L}{\partial \beta_j} = -\sum_i (y_i - \beta X_i) X_{i,j}$$

  3. Choisir un pas η et effectuer les mises à jour

$$\beta^{(t+1)} = \beta^{(t)} - \eta \nabla L(\beta)$$

Le modèle perceptron (1 neurone)



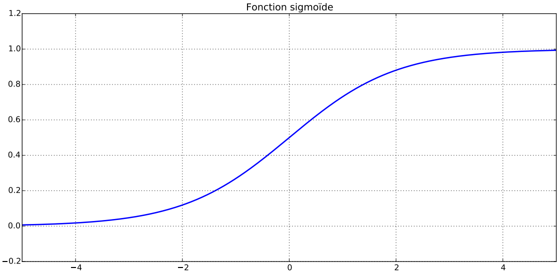
- Soit le modèle suivant :

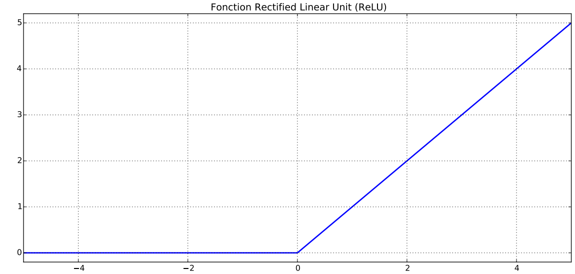
$$y_i = \underbrace{f\left(\underbrace{\beta_0 + \sum_j \beta_j x_{i,j}}_{o_i}\right)}_{a_i} + \epsilon_i$$

- Paramètres : $\{\beta_j\}_{j \geq 0}$
- f : fonction d'activation

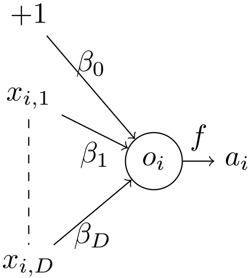
Perceptron & fonction d'activation

- Exemples typiques de fonctions d'activation f





Perceptron & Optimisation



$$a_i = f\left(\underbrace{\beta_0 + \sum_j \beta_j x_{i,j}}_{o_i}\right)$$

Descente de gradient

1. Se fixer une fonction de coût à minimiser

$$L(\beta) = \sum_i (y_i - a_i)^2 = \sum_i L_i(\beta)$$

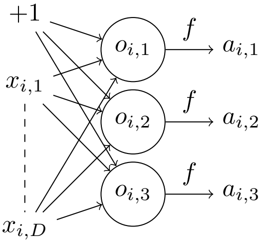
2. Exprimer le gradient de cette fonction de coût

$$\forall j, (\nabla L_i(\beta))_j = \frac{\partial L_i}{\partial \beta_j} = \frac{\partial L_i}{\partial a_i}\frac{\partial a_i}{\partial o_i}\frac{\partial o_i}{\partial \beta_j}$$

3. Choisir un pas η et effectuer les mises à jour

$$\begin{aligned} \nabla L(\beta) &= \sum_i \nabla L_i(\beta) \\ \beta^{(t+1)} &= \beta^{(t)} - \eta \nabla L(\beta) \end{aligned}$$

Perceptron & Classification supervisée

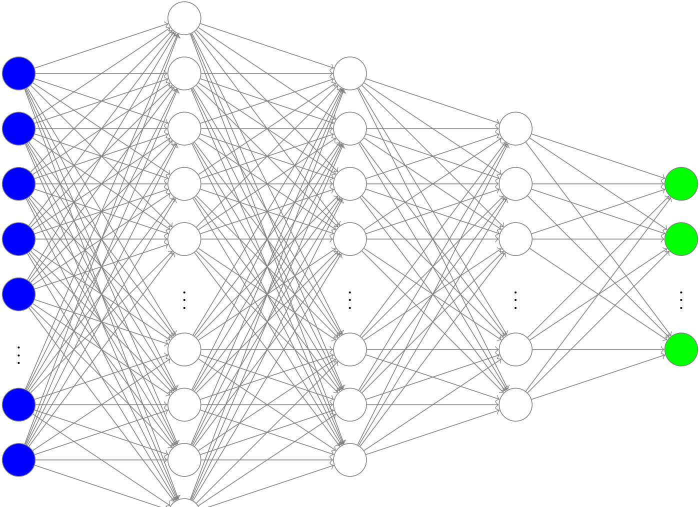


- Classification binaire
  - a_i : estimation par le modèle de $P(y_i = 1)$
  - il suffit de bien choisir f
- Classification multi-classes
  - nécessité d'empiler les perceptrons
  - $a_{i,k}$: estimation par le modèle de $P(y_i = k)$
  - f : fonction softmax

$$softmax(o_{i,k}) = \frac{e^{o_{i,k}}}{\sum_{k'=1}^{K} e^{o_{i,k'}}}$$

Perceptron multi-couches

- Principe : ajouter des couches *cachées*
- But : augmenter l'expressivité du modèle
- Impact : explosion du nombre de paramètres



Universal approximation theorem

- Hypothèse : f non constante, croissante, continue
- Énoncé :

$$\forall \epsilon > 0, \forall g \text{ continue sur } [0,1]^m,$$

$$\exists N, \{(v_j, \beta_j)\}_j, \text{ tels que :}$$

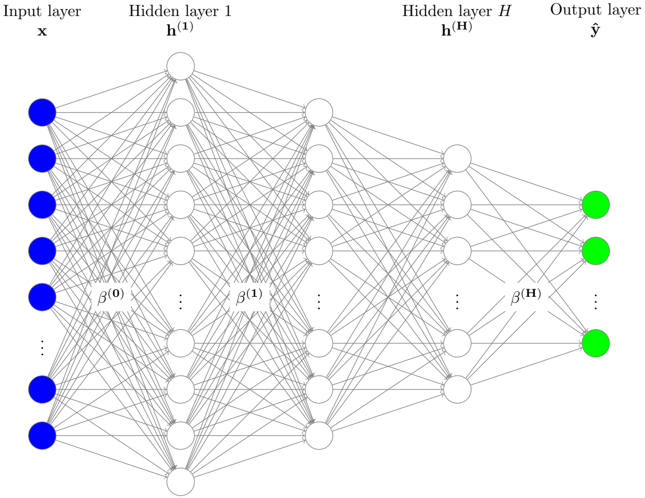
$$\forall x \in [0,1]^m, \left| \sum_{j=1}^{N} v_j f\left(\beta_j x + \beta_{j,0}\right) - g(x) \right| < \epsilon$$

Universal approximation theorem : en pratique

- Objectif d'expressivité rempli
- N potentiellement très grand
- Pas d'assurance de réussir à apprendre les paramètres qui vérifient la condition
- En pratique, on observe que les réseaux *profonds*[1]
  - atteignent la même qualité de représentation à des tailles (nb de paramètres) réduites
  - généralisent mieux à des données inconnues

[1] *vs* réseaux superficiels, a une seule couche cachée

Perceptron multi-couches : apprentissage en pratique – 1



Réseaux de neurones : apprentissage en pratique – 2

Descente de gradient en pratique : 3 options

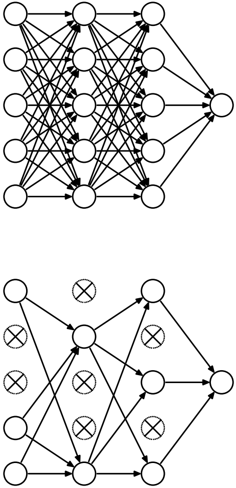
- *Batch Gradient Descent*
- *Stochastic Gradient Descent*
- *Mini-Batch Gradient Descent*

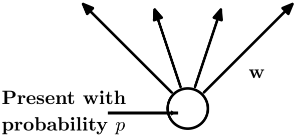
Réseaux de neurones : apprentissage en pratique – 3

Pour régulariser lors de l'apprentissage, 3 options :

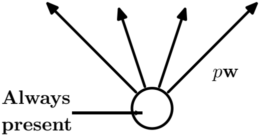
- Pénalisation à la ElasticNet (L1/L2)
- *Drop-Out*
- *Early Stopping*

Réseaux de neurones : le principe du *drop-out*





(a) At training time



(b) At test time

Figures issues de "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Srivastava *et al.*, JMLR, 2014.

Perceptron multi-couches : principe général

- Couches cachées : transformations non linéaires des données
- Dernière couche : régression logistique
- On parle de *end-to-end learning*

Exemple avec un réseau à 3 couches cachées