Réseaux de neurones (Notes de cours)

Romain Tavenard

Chapitre 1

Préambule

Ce document est un ensemble de notes associées au module de classification supervisée pour la deuxième année de master de Statistiques de l'Université de Rennes 2, et plus particulièrement la partie sur les réseaux de neurones. Il est distribué librement (sous licence CC BY-NC-SA plus précisément) et se veut évolutif, n'hésitez donc pas à faire vos remarques à son auteur dont vous trouverez le contact sur sa page web.

1.1 Contenu du cours

Chapitre 2

Régression logistique et perceptron

2.1 Retour sur la régression logistique

Dans la suite, nous nous focaliserons sur le cas de la classification binaire, mais tout ce qui est discuté dans ce document est bien entendu généralisable au cas multi-classes. Dans le cas de la régression logistique, le modèle est le suivant :

$$\log \frac{P(1|X)}{1 - P(1|X)} = \beta_0 + \sum_j \beta_j x_j \tag{2.1}$$

ce qui nous donne :

$$P(1|X) = \frac{1}{1 + e^{\beta_0 + \sum_j \beta_j x_j}} \tag{2.2}$$

Cela nous amène à chercher à maximiser la log-vraisemblance qui s'écrit :

$$\ell(\beta) = \sum_i y_i P(1|X_i) + (1 - y_i) \log(1 - P(1|X_i)) \tag{2.3}$$

En réintroduisant les formules (2.1) et (2.2), on obtient la quantité suivante à maximiser (ou son opposé à minimiser, c'est égal) :

$$\ell(\beta) = \sum_i - \log(1 + e^{\beta_0 + \sum_j \beta_j x_{ij}}) + y_i(\beta_0 + \sum_j \beta_j x_{ij}) \tag{2.4}$$

Pour minimiser cette quantité, on va effectuer une descente de gradient, et on devra donc calculer $\frac{\partial \ell}{\partial \beta}$:

$$\frac{\partial \ell}{\partial \beta_0} = \sum_i -\frac{e^{\beta_0 + \sum_j \beta_j x_{ij}}}{1 + e^{\beta_0 + \sum_j \beta_j x_{ij}}} + y_i \tag{2.5}$$

$$\forall j \geq 1, \frac{\partial \ell}{\partial \beta_j} = \sum_i -\frac{e^{\beta_0 + \sum_j \beta_j x_{ij}}}{1 + e^{\beta_0 + \sum_j \beta_j x_{ij}}} x_{ij} + y_i x_{ij} \tag{2.6}$$

2.2 Le perceptron : notations et représentation

2.2.1 Poids

2.2.2 Fonction d'activation

2.2.3 Calcul de gradient

Blabla je parle de l'équation (2.7) qui est super.

$$y = mx + b \tag{2.7}$$

Chapitre 3

Perceptron multi-couches

http://cs231n.github.io/neural-networks-1/

3.1 Cas d'un réseau de neurones à une couche cachée

3.1.1 Calcul de gradient

3.1.2 Visualisation de frontières apprises

Exemple intéressant : http://cs231n.github.io/neural-networks-case-study/

3.2 Cas multi-couches

3.2.1 Calcul de gradient

http://cs231n.github.io/optimization-2/

3.2.2 Visualisation

Comprendre qu'on apprend une représentation (eg PCA améliorée) suivie d'un classifieur linéaire (logistic regression)

— https://rajarsheem.wordpress.com/2017/05/04/neural-networks-dynamics/
— http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

3.2.3 Exemples adversaires

3.3 Quelques considérations pratiques

3.3.1 Initialisation des poids du réseau

3.3.2 Régularisation

http://cs231n.github.io/neural-networks-2/

3.3.2.1 Régularisations L1/L2

3.3.2.2 Utilisation du *dropout*

Chapitre 4

Réseaux de neurones convolutionnels

4.1 Motivation

4.2 Mise en oeuvre

http://cs231n.github.io/convolutional-networks/

4.3 Apprentissage par transfert et *fine-tuning*

http://cs231n.github.io/transfer-learning/

Chapitre 5

Mise en oeuvre avec `keras`

https://keras.io.