Apprentissage supervisé
Cours 2 bis : Introduction aux réseaux de neurones

Agathe Guilloux, Geneviève Robin

Le problème de classification binaire

On a des données d'apprentissage (learning data) pour des individus $i = 1, \dots, n$. Pour chaque individu i :

- on a un vecteur de covariables (features) $x_i \in \mathcal{X} \subset \mathbb{R}^d$
- la valeur de son label $y_i \in \{-1, 1\}$.
- on suppose que les couples (X_i, Y_i) sont des copies i.i.d. de (X, Y) de loi inconnue et que l'on observe leurs réalisations (x_i, y_i) $(i = 1, \dots, n)$.

But

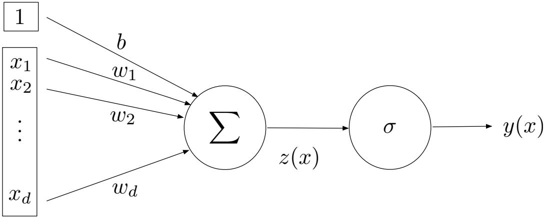
- On veut, pour un nouveau vecteur X_+ de features, prédire la valeur du label Y_+ par $\hat{Y}_+ \in \{-1, 1\}$
- Pour cela, on utilise les données d'apprentissage $\mathcal{D}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ pour construire un **classifieur** $\hat{c}$ de telle sorte que
$$\hat{Y}_+ = \hat{c}(X_+).$$
et $\hat{Y}$ est proche de Y_+ (dans un sens à préciser).

Régression logistique : rappel

- A partir des données $(x_1, y_1), \ldots, (x_n, y_n)$ avec $y_i \in \{-1, 1\}$
- Pour $x \in \mathbb{R}^d$, on modélise

$$\mathbb{P}(Y = 1 | X = x) = \sigma(\langle w, x \rangle + b) = \frac{1}{1 + e^{-\langle w, x \rangle - b}}$$

Régression logistique : représentation graphique



Vocabulaire du deep learning:

- x est l'input
- $z(x) = \langle w, x \rangle + b$ est la **pré-activation**
- $y(x) = \sigma(z(x))$ est l'**output** (dans $[-1, 1]$ dans le cas de la classification binaire)
- $w \in \mathbb{R}^d$ sont les **poids/weights** et $b \in \mathbb{R}$ est l'**intercept/bias**)
- σ est la fonction d'**activation**

On l'appelle une **unité/unit** ou un **neurone artificiel/artificial neuron**

Regression logistique multiclasses / régression multinomiale

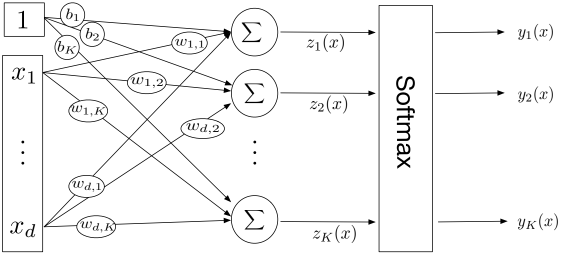
- A partir des données $(x_1, y_1), \ldots, (x_n, y_n)$ avec $y_i \in \{1, \ldots, K\}$
- Pour $x \in \mathbb{R}^d$, on modélise

$$\mathbb{P}(Y = k | X = x) = \frac{e^{\langle w_k, x \rangle + b_k}}{\sum_{k'=1}^{K} e^{\langle w_{k'}, x \rangle + b_{k'}}}$$

  for $k \in \{1, \ldots, K\}$
- Cela s'appelle la **régression softmax**, c'est une généralisation de la régression logistique en classification multiclasses.
- Il y a un vecteur de poids $w_k \in \mathbb{R}^d$ pour chaque classe k. On déifnit la matrice de taille $d \times K$ des poids $\mathbf{W}$ avec $\mathbf{W}_{\bullet, k} = w_k$
- La fonction objectif dans ce cas est

$$- \log \mathcal{L}(\mathbf{W}) = - \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}_{y_i = k} \log \left(\frac{e^{\langle w_k, x \rangle + b_k}}{\sum_{k'=1}^{K} e^{\langle w_{k'}, x \rangle + b_{k'}}} \right)$$

Régression softmax : représentation graphique



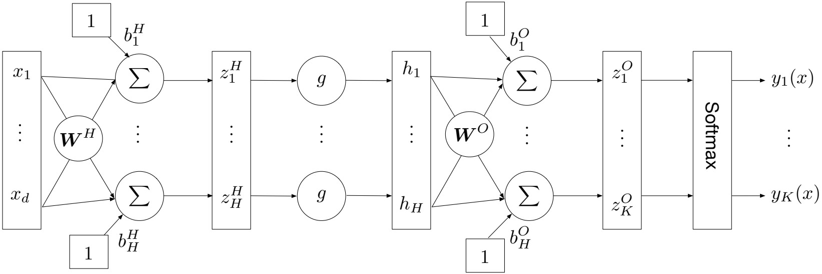
- x est l'input
- $z_k(x) = \langle \mathbf{W}_{\bullet,k}, x \rangle + b_k$ sont des pré-activations ou "logits"
- les outputs sont définis par $y_k(x) = \frac{e^{z_k(x)}}{\sum_{k'=1}^{K} e^{z_{k'}(x)}}$ après l'activation softmax

On peut aussi écrire

$$y(x) = \text{softmax}(z(x)) = \text{softmax}(\mathbf{W}^\top x + b)$$

Un couche cachée

Un réseau à une couche cachée (hidden layer) de largeur H se représente de la façon suivante



On peut écrire

$$
\begin{aligned}
y(x) &= \mathrm{softmax}(z^{(O)}) = \mathrm{softmax}(\mathbf{W}^{(O)\top} h + b^{(O)}) \\
&= \mathrm{softmax}(\mathbf{W}^{(O)\top} g(z^{(H)}) + b^{(O)}) \\
&= \mathrm{softmax}(\mathbf{W}^{(O)\top} g(\mathbf{W}^{(H)\top} x + b^{(H)}) + b^{(O)})
\end{aligned}
$$

g est une **fonction d'activation / activation function** appliquée entrée par entrée à $z_1^H, \ldots, z_H^H$.

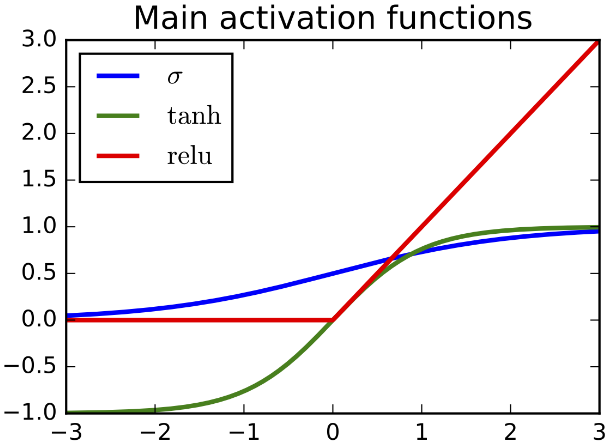
Fonctions d'activation

- Elles sont appliquées entrée par entrée aux inputs
- Les plus courantes sont les fonctions **sigmoid**, **tanh** et **relu** (rectified linear unit)

$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad \tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}, \quad \mathrm{relu}(z) = \max(0, z)$$

- de dérivées

$$\begin{aligned} \sigma'(z) &= \sigma(z)(1-\sigma(z)) \\ \tanh(z) &= 1 - \tanh(z)^2, \\ \mathrm{relu}'(z) &= \mathbb{1}_{z>0} \end{aligned}$$


Main activation functions
σ
tanh
relu
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3

Feed-forward neural network (FFNN)

- On cherche une approximation $\widehat{f}$ de la fonction f^* qui associe l'input x à l'output y: $y = f^*(x)$.
- Chaque couche à une fonction, puis elles sont composées $\widehat{f}(x) = \widehat{f_3}(\widehat{f_2}(\widehat{f_1}(x)))$ dans le cas de 3 couches.
- Le réseau décrit la façon dont les fonctions sont composées.

Feed-forward neural networks

(FFNN) De manière générale, on a

$$
\begin{aligned}
h^{(1)} &= g^{(1)}(\mathbf{W}^{(1)\top} x + b^{(1)}) \\
h^{(2)} &= g^{(2)}(\mathbf{W}^{(2)\top} h^{(1)} + b^{(2)}) \\
&\vdots \\
h^{(L)} &= g^{(L)}(\mathbf{W}^{(L)\top} h^{(L-1)} + b^{(L)}) \\
y &= \mathrm{softmax}(\mathbf{W}^{(O)\top} h^{(L)} + b^{(O)})
\end{aligned}
$$

pour L couches. Il faut choisir

- la largeur de chaque couche
- la profondeur du réseau

Il est admis que c'est mieux de prendre plusieurs couches de largeur assez petite.

Entraînement d'un FFNN

Pour entraîner un FFNN, on procède comme suit

- pour une valeur de poids, on calcule la loss et les prédictions par **forward-propagation**
- on calcule les gradients puis on utilise la **back-propagation**
- on utilise des méthodes de descente de gradient stochastique.

Un exemple : le playground tensorflow.

Entraînement du réseau (1)

Pour le FFNN à une couche cachée, on a

$$\widehat{y} = \text{softmax}(\mathbf{W}^{(O)\top} g(\mathbf{W}^{(H)\top} x + b^{(H)}) + b^{(O)})$$

On mesure le goodness-of-fit pour la valeur $\boldsymbol{\theta} = (\mathbf{W}^H, b^H, \mathbf{W}^O, b^O)$ avec l'opposé de la log-vraisemblance (on l'appelle aussi cross-entropy)

$$- \text{LogLik}(\boldsymbol{\theta}) =$$

$$- \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}_{y_i = k} \log \left(\frac{\exp\left(\left\langle \mathbf{W}^{(O)}_{\bullet,k}, g(\mathbf{W}^{(H)\top} x_i + b^{(H)}) \right\rangle + b^{(O)}_k \right)}{\sum_{k'=1}^{K} \exp\left(\left\langle \mathbf{W}^{(O)}_{\bullet,k'}, g(\mathbf{W}^{(H)\top} x_i + b^{(H)}) \right\rangle + b^{(O)}_{k'} \right)} \right)$$

On peut également ajouter des pénalisations par exemple la ridge

$$\text{pen}(\boldsymbol{\theta}) = \text{pen}(\mathbf{W}^{(O)}, \mathbf{W}^{(H)}) = \lambda(\|\mathbf{W}^{(O)}\|_F^2 + \|\mathbf{W}^H\|_F^2)$$

Remarque. On ne pénalise jamais les biais.

Entraînement du réseau (2)

On doit minimiser

$$F(\boldsymbol{\theta}) = -\mathsf{LogLik}(\boldsymbol{\theta}) + \mathsf{pen}(\boldsymbol{\theta})$$

avec

$$-\mathsf{LogLik}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(x_i; \boldsymbol{\theta}))$$

où ℓ est une fonction de perte (par exemple le softmax en classificaiton).

Back-propagation (1)

- C'est un algorithme qui permet de calculer efficacement les gradients
- en utilisant la structure du réseau de neurone
- et la **chain rule** (gradient de fonctions composées)

Back-propagation (2)

- L'idée est très simple $f(g(x))' = f'(g(x)) \times g'(x)$
- Pour calculer $f'(y) \times g'(x)$, on remarque que

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

- Ou, en dimension supérieure, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ et $g : \mathbb{R}^m \to \mathbb{R}^n$ alors si $y = g(x)$ and $z = f(y)$ on a

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

Back-propagation (3)

- En notation vectorielle, on

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^\top \nabla_y z,$$

  où $\frac{\partial y}{\partial x}$ est le Jacobien (de taille $n \times m$) de g
- Le gradient de z par rapport à x est obtenu en multipliant $\frac{\partial y}{\partial x}$ par $\nabla_y z$
- La **back-propagation** effectue des produits de Jacobien sur toute la profondeur du réseau jusqu'à l'update sur $\boldsymbol{\theta} = (\mathbf{W}^H, b^H, \mathbf{W}^O, b^O)$

Exercise. Expliciter les étapes back-propagation sur le FFNN à une couche cachée décrit plus haut.