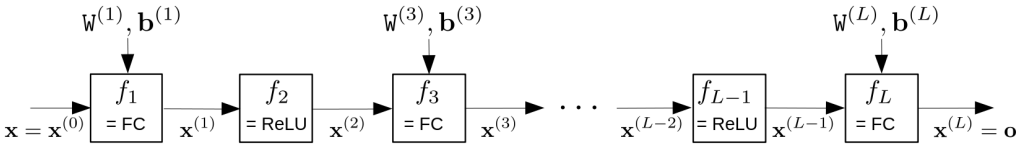
TD : Implémentation d'un réseau de neurones de type Perceptron multicouche pour un problème de classification

Guillaume Bourmaud

09/02/23

1 Introduction

Un réseau de neurones de type Perceptron multicouche (MLP) est composé d'une succession de couches, alternant une transformation affine (FC) et une fonction non-linéaire (historiquement la fonction sigmoïde mais ici nous utiliserons la fonction ReLU : $z = \max(0, x)$). Un tel réseau est illustré graphiquement ci-après :



Remarquons que selon les notations utilisées dans le schéma précédent, le nombre de couches L est nécessairement impair et supérieur ou égal à 3 (car les couches d'entrée et de sortie sont des transformations affines). Ainsi un MLP s'écrit comme une composition de fonctions :

$$\mathbf{o} = \text{MLP}\left(\mathbf{x}; \left\{\mathtt{W}^{(2j-1)}, \mathbf{b}^{(2j-1)}\right\}_{j=1,\dots,\lceil L/2 \rceil}\right) \tag{1}$$

$$= \text{FC}\left(\text{ReLU}\left(...\text{FC}\left(\text{ReLU}\left(\text{FC}\left(\mathbf{x}; \mathtt{W}^{(1)}, \mathbf{b}^{(1)}\right)\right); \mathtt{W}^{(3)}, \mathbf{b}^{(3)}\right)...\right); \mathtt{W}^{(L)}, \mathbf{b}^{(L)}\right). \tag{2}$$

Nous considérons le cas où les vecteurs $(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(L-1)})$ sont tous de taille H et sont des vecteurs ligne, donc de taille $1 \times H$. Le MLP que nous considérons a donc deux hyper-paramètres : L (le nombre de couches) et H. Nous nous intéressons ici à un problème de classification à C classes, donc $\mathbf{x}^{(L)} = \mathbf{o}$ est de taille $1 \times C$. L'espace de départ est de dimension D donc $\mathbf{x} = \mathbf{x}^{(0)}$ est de taille $1 \times D$. La fonction de coût utilisée est l'entropie croisée Multinoulli :

$$l(y, \mathbf{o}) = -\ln(\mathbf{p}_y) \tag{3}$$

où $\mathbf{p} = \text{softmax}(\mathbf{o})$ est un vecteur de probabilités (scores positifs qui se somment à 1) de taille $1 \times C$ et $y \in \{0, ..., C-1\}$ est l'étiquette (donc un scalaire 1×1). Par exemple, si $y = 2$, alors la notation $\mathbf{p}_y$ correspond à récupérer le 3ème élément du vecteur $\mathbf{p}$. Minimiser la fonction de coût revient donc à maximiser le score de probabilité $\mathbf{p}_y$. La fonction softmax est définie de la manière suivante :

$$\mathbf{p}_i = \frac{\exp(\mathbf{o}_i)}{\sum_{j=0}^{C-1} \exp(\mathbf{o}_j)}. \tag{4}$$

Les paramètres du réseau seront optimisés en minimisant la fonction de coût suivante sur la base d'apprentissage $\left\{\mathbf{x}_{[i]}, y_{[i]}\right\}_{i=0...N-1}$:

$$\arg \min_{\left\{\mathtt{W}^{(2j-1)}, \mathbf{b}^{(2j-1)}\right\}_{j=1,\dots,\lceil L/2 \rceil}} \frac{1}{N} \sum_{i=0}^{N-1} l\left(y_{[i]}, \text{MLP}\left(\mathbf{x}_{[i]}; \left\{\mathtt{W}^{(2j-1)}, \mathbf{b}^{(2j-1)}\right\}_{j=1,\dots,\lceil L/2 \rceil}\right)\right) \tag{5}$$

Nous utiliserons pour cela une méthode de descente de gradient. Comme vu en cours, le gradient, c'est-à-dire la dérivée de la fonction de coût par rapport aux paramètres du réseau, est obtenue en utilisant le théorème de dérivation des fonctions composées.

2 Théorème de dérivation d'une fonction composée

Prenons le cas d'une fonction composée de deux fonctions ($f : \mathbb{R}^D \to \mathbb{R}^H$ et $g : \mathbb{R}^H \to \mathbb{R}$):

$$a = g\left(f\left(\mathbf{x}\right)\right). \tag{6}$$

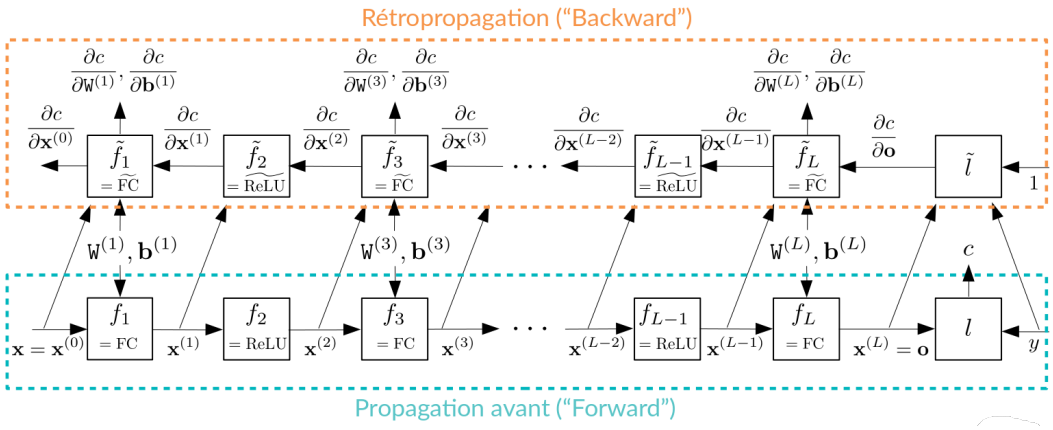
En définissant $\mathbf{z} = f(\mathbf{x})$, le théorème de dérivation d'une fonction composée nous indique que la dérivée de $a \in \mathbb{R}$ par rapport au i-ème élément du vecteur $\mathbf{x}$ s'exprime de la manière suivante :

$$\frac{\partial a}{\partial \mathbf{x}_i} = \sum_{j=0}^{H-1} \frac{\partial a}{\partial \mathbf{z}_j}\Big|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{z}_j}{\partial \mathbf{x}_i}. \tag{7}$$

Ainsi nous pouvons introduire la notation suivante :

$$\frac{\partial a}{\partial \mathbf{x}} = \tilde{f}\left(\frac{\partial a}{\partial \mathbf{z}}\Big|_{\mathbf{z}=f(\mathbf{x})}, \mathbf{x}\right), \tag{8}$$

où $\tilde{f}$ est la fonction qui implémente l'équation (7), $\frac{\partial a}{\partial \mathbf{x}} = \left[\frac{\partial a}{\partial \mathbf{x}_0}, \frac{\partial a}{\partial \mathbf{x}_1}, ..., \frac{\partial a}{\partial \mathbf{x}_{D-1}}\right]$ et $\frac{\partial a}{\partial \mathbf{z}}\Big|_{\mathbf{z}=f(\mathbf{x})} = \left[\frac{\partial a}{\partial \mathbf{z}_0}, \frac{\partial a}{\partial \mathbf{z}_1}, ..., \frac{\partial a}{\partial \mathbf{z}_{H-1}}\right]$.
En appliquant ce théorème à un MLP dans le but de calculer la dérivée de la fonction de coût par rapport aux paramètres du MLP, nous obtenons l'étape de rétropropagation du gradient, sous la forme d'un **graphe de calcul**, illustrée dans la figure ci-après.



Travail : écrire la taille de chacune des variables présentes dans le graphe de calcul précédent

Ainsi, pour mettre en œuvre une méthode de descente de gradient dans le but d'optimiser les paramètres d'un MLP, nous aurons uniquement besoin de connaître les expressions des fonctions $\tilde{l}(1, \mathbf{o})$, $\widetilde{\text{FC}}\left(\frac{\partial l}{\partial \mathbf{x}^{(i)}}\Big|_{\mathbf{x}^{(i)}=\text{FC}(\mathbf{x}^{(i-1)})}, \mathbf{x}^{(i-1)}, \mathtt{W}^{(i)}, \mathbf{b}^{(i)}\right)$ et $\widetilde{\text{ReLU}}\left(\frac{\partial l}{\partial \mathbf{x}^{(i)}}\Big|_{\mathbf{x}^{(i)}=\text{ReLU}(\mathbf{x}^{(i-1)})}, \mathbf{x}^{(i-1)}\right)$. C'est le travail qui sera réalisé dans la suite de ce TD. Les expressions obtenues seront codées ultérieurement lors d'un TP en Python utilisant la bibliothèque Numpy.

3 Notations

Dans le but de paralléliser les calculs, **et ainsi d'obtenir une implémentation en Python efficace**, nous considérons le cas où les N vecteurs de taille D $\left\{\mathbf{x}_{[i]}\right\}_{i=0...N-1}$ de la base d'apprentissage sont organisés en une matrice :

$$\underbrace{\mathtt{X}}_{N\times D} = \begin{bmatrix} \mathtt{X}_{00} & \mathtt{X}_{01} & \dots & \mathtt{X}_{0(D-1)} \\ \mathtt{X}_{10} & \mathtt{X}_{11} & \dots & \mathtt{X}_{1(D-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathtt{X}_{(N-1)0} & \mathtt{X}_{(N-1)1} & \dots & \mathtt{X}_{(N-1)(D-1)} \end{bmatrix} \tag{9}$$

Les étiquettes $\left\{y_{[i]}\right\}_{i=0...N-1}$ sont organisées sous la forme d'un vecteur :

$$\mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{(N-1)} \end{bmatrix}^\top \tag{10}$$

Ainsi le MLP s'appliquera directement sur cette matrice $\mathtt{X}$ et produira une matrice de scores $\mathtt{O}$:

$$\mathtt{O} = \text{MLP}\left(\mathtt{X}; \boldsymbol{\theta}\right) = \text{FC}\left(\text{ReLU}\left(...\text{FC}\left(\text{ReLU}\left(\text{FC}\left(\mathtt{X}; \mathtt{W}^{(1)}, \mathbf{b}^{(1)}\right)\right); \mathtt{W}^{(3)}, \mathbf{b}^{(3)}\right)...\right); \mathtt{W}^{(L)}, \mathbf{b}^{(L)}\right) \tag{11}$$

où

$$\underbrace{\mathtt{O}}_{N\times C} = \begin{bmatrix} \mathtt{O}_{00} & \mathtt{O}_{01} & \dots & \mathtt{O}_{0(C-1)} \\ \mathtt{O}_{10} & \mathtt{O}_{11} & \dots & \mathtt{O}_{1(C-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathtt{O}_{(N-1)0} & \mathtt{O}_{(N-1)1} & \dots & \mathtt{O}_{(N-1)(C-1)} \end{bmatrix} \tag{12}$$

Après softmax, les vecteurs de scores normalisés peuvent également être rangés dans une matrice :

$$\underbrace{\mathtt{P}}_{N\times C} = \begin{bmatrix} \mathtt{P}_{00} & \mathtt{P}_{01} & \dots & \mathtt{P}_{0(C-1)} \\ \mathtt{P}_{10} & \mathtt{P}_{11} & \dots & \mathtt{P}_{1(C-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathtt{P}_{(N-1)0} & \mathtt{P}_{(N-1)1} & \dots & \mathtt{P}_{(N-1)(C-1)} \end{bmatrix}. \tag{13}$$

Par définition, les matrices ($\mathtt{X}^{(1)}$,..., $\mathtt{X}^{(L-1)}$) sont donc de taille $N \times H$.

4 Obtention des expressions de $\widetilde{\text{FC}}$, $\widetilde{\text{ReLU}}$ et $\tilde{l}$

4.1 Dérivée transformation affine : $\widetilde{\text{FC}}$

De manière générale, en utilisant les notations précédemment introduites, une transformation affine peut s'écrire :

$$\mathtt{Z}_{kl} = \sum_{j=0}^{D-1} \mathtt{X}_{kj}\mathtt{W}_{jl} + \mathbf{b}_l \tag{14}$$

où $\underbrace{\mathtt{Z}}_{N\times H} = \begin{bmatrix} \mathtt{Z}_{00} & \mathtt{Z}_{01} & \dots & \mathtt{Z}_{0(H-1)} \\ \mathtt{Z}_{10} & \mathtt{Z}_{11} & \dots & \mathtt{Z}_{1(H-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathtt{Z}_{(N-1)0} & \mathtt{Z}_{(N-1)1} & \dots & \mathtt{Z}_{(N-1)(H-1)} \end{bmatrix}$, $\underbrace{\mathtt{W}}_{D\times H} = \begin{bmatrix} \mathtt{W}_{00} & \mathtt{W}_{01} & \dots & \mathtt{W}_{0(H-1)} \\ \mathtt{W}_{10} & \mathtt{W}_{11} & \dots & \mathtt{W}_{1(H-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathtt{W}_{(D-1)0} & \mathtt{W}_{(D-1)1} & \dots & \mathtt{W}_{(D-1)(H-1)} \end{bmatrix}$, $\underbrace{\mathbf{b}}_{1\times H} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{(H-1)} \end{bmatrix}^\top$. En utilisant le produit matrice/vecteur, l'équation devient :

$$\mathtt{Z} = \text{FC}\left(\mathtt{X}, \mathtt{W}, \mathbf{b}\right) = \mathtt{X}\mathtt{W} + \mathtt{1}_N\mathbf{b} \tag{15}$$

où $\underbrace{\mathtt{1}_N}_{N\times 1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$. La sortie $\mathtt{Z}$ de cette transformation affine est passée dans une fonction g dont la sortie s est scalaire :

$$s = g\left(\mathtt{Z}\right) \tag{16}$$

En supposant le gradient $\underbrace{\partial s/\partial \mathtt{Z}}_{N\times H} = \begin{bmatrix} \partial s/\partial \mathtt{Z}_{00} & \partial s/\partial \mathtt{Z}_{01} & \dots & \partial s/\partial \mathtt{Z}_{0(H-1)} \\ \partial s/\partial \mathtt{Z}_{10} & \partial s/\partial \mathtt{Z}_{11} & \dots & \partial s/\partial \mathtt{Z}_{1(H-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \partial s/\partial \mathtt{Z}_{(N-1)0} & \partial s/\partial \mathtt{Z}_{(N-1)1} & \dots & \partial s/\partial \mathtt{Z}_{(N-1)(H-1)} \end{bmatrix}$ connu, l'objectif est de calculer la dérivée de s par rapport aux entrées $\mathtt{X}$ et aux paramètres $\mathtt{W}$ et $\mathbf{b}$.

Travail : dessiner le graphe de calcul correspondant aux deux équations précédentes, en faisant apparaître s, g, $\mathtt{Z}$, $\mathbf{FC}$, $\mathtt{X}$, $\mathtt{W}$, $\mathbf{b}$, $\tilde{g}$, $\partial s/\partial \mathtt{Z}$, $\widetilde{\mathbf{FC}}$, $\partial s/\partial \mathtt{X}$, $\partial s/\partial \mathtt{W}$ et $\partial s/\partial \mathbf{b}$.

L'objectif de cette partie est d'obtenir des expressions permettant d'implémenter la fonction $\widetilde{\mathbf{FC}}$. Il faut donc calculer les expressions de $\partial s/\partial \mathtt{X}$, $\partial s/\partial \mathtt{W}$ et $\partial s/\partial \mathbf{b}$ en les simplifiant au maximum pour faire apparaître des opérations permettant une implémentation efficace en Python (par exemple des produits de matrices).

4.1.1 Calcul de $\partial s/\partial \mathtt{X}$

Le théorème de dérivation d'une fonction composée nous dit que :

$$\partial s/\partial \mathtt{X}_{ij} = \sum_{k=0}^{N-1}\sum_{l=0}^{H-1} \partial s/\partial \mathtt{Z}_{kl}.\partial \mathtt{Z}_{kl}/\partial \mathtt{X}_{ij} \tag{17}$$

Travail :

- **montrer que l'équation précédente se simplifie en $\partial s/\partial \mathtt{X}_{ij} = \sum_{l=0}^{H-1} \partial s/\partial \mathtt{Z}_{il}.\mathtt{W}_{jl}$**
- **sachant que $\underbrace{\partial s/\partial \mathtt{X}}_{N\times D} = \begin{bmatrix} \partial s/\partial \mathtt{X}_{00} & \partial s/\partial \mathtt{X}_{01} & \dots & \partial s/\partial \mathtt{X}_{0(D-1)} \\ \partial s/\partial \mathtt{X}_{10} & \partial s/\partial \mathtt{X}_{11} & \dots & \partial s/\partial \mathtt{X}_{1(D-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \partial s/\partial \mathtt{X}_{(N-1)0} & \partial s/\partial \mathtt{X}_{(N-1)1} & \dots & \partial s/\partial \mathtt{X}_{(N-1)(D-1)} \end{bmatrix}$, montrer que $\underbrace{\partial s/\partial \mathtt{X}}_{N\times D} = \underbrace{\partial s/\partial \mathtt{Z}}_{N\times H}\underbrace{\mathtt{W}^\top}_{H\times D}$**

Ainsi $\partial s/\partial \mathtt{X}$ pourra être calculé lors de l'implémentation en Python par un simple appel à une fonction réalisant un produit matriciel.

4.1.2 Calcul de $\partial s/\partial \mathtt{W}$

Le théorème de dérivation d'une fonction composée nous dit que :

$$\partial s/\partial \mathtt{W}_{ij} = \sum_{k=0}^{N-1}\sum_{l=0}^{H-1} \partial s/\partial \mathtt{Z}_{kl}.\partial \mathtt{Z}_{kl}/\partial \mathtt{W}_{ij} \tag{18}$$

Travail :

- **montrer que l'équation précédente se simplifie en $\partial s/\partial \mathtt{W}_{ij} = \sum_{k=0}^{N-1} \partial s/\partial \mathtt{Z}_{kj}.\mathtt{X}_{ki}$**
- **sachant que $\underbrace{\partial s/\partial \mathtt{W}}_{D\times H} = \begin{bmatrix} \partial s/\partial \mathtt{W}_{00} & \partial s/\partial \mathtt{W}_{01} & \dots & \partial s/\partial \mathtt{W}_{0(H-1)} \\ \partial s/\partial \mathtt{W}_{10} & \partial s/\partial \mathtt{W}_{11} & \dots & \partial s/\partial \mathtt{W}_{1(H-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \partial s/\partial \mathtt{W}_{(D-1)0} & \partial s/\partial \mathtt{W}_{(D-1)1} & \dots & \partial s/\partial \mathtt{W}_{(D-1)(H-1)} \end{bmatrix}$ montrer que $\underbrace{\partial s/\partial \mathtt{W}}_{D\times H} = \underbrace{\mathtt{X}^\top}_{D\times N}\underbrace{\partial s/\partial \mathtt{Z}}_{N\times H}$**

Ainsi, comme $\partial s/\partial \mathtt{X}$, $\partial s/\partial \mathtt{W}$ pourra être calculé lors de l'implémentation en Python par un simple appel à une fonction réalisant un produit matriciel.

4.1.3 Calcul de $\partial s/\partial \mathbf{b}$

Travail :

- **montrer que $\underbrace{\partial s/\partial \mathbf{b}}_{1\times H} = \underbrace{\mathtt{1}_N^\top}_{1\times N}\underbrace{\partial s/\partial \mathtt{Z}}_{N\times H}$**

Ainsi $\partial s/\partial \mathbf{b}$ pourra être calculé lors de l'implémentation en Python par un simple appel à une fonction réalisant un produit vecteur/matrice.

4.2 Dérivée ReLU : $\widetilde{\text{ReLU}}$

De manière générale, en utilisant les notations précédemment introduites, la fonction ReLU peut s'écrire :

$$\mathtt{Z}_{kl} = \max\left(0, \mathtt{X}_{kl}\right) \tag{19}$$

La sortie $\mathtt{Z}$ de cette transformation affine est passée dans une fonction g dont la sortie s est scalaire :

$$s = g\left(\mathtt{Z}\right) \tag{20}$$

En supposant le gradient $\underbrace{\partial s/\partial \mathtt{Z}}_{N\times D} = \begin{bmatrix} \partial s/\partial \mathtt{Z}_{00} & \partial s/\partial \mathtt{Z}_{01} & \dots & \partial s/\partial \mathtt{Z}_{0(D-1)} \\ \partial s/\partial \mathtt{Z}_{10} & \partial s/\partial \mathtt{Z}_{11} & \dots & \partial s/\partial \mathtt{Z}_{1(D-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \partial s/\partial \mathtt{Z}_{(N-1)0} & \partial s/\partial \mathtt{Z}_{(N-1)1} & \dots & \partial s/\partial \mathtt{Z}_{(N-1)(D-1)} \end{bmatrix}$ connu, l'objectif est de calculer la dérivée de s par rapport aux entrées $\mathtt{X}$.

Travail : dessiner le graphe de calcul correspondant aux deux équations précédentes, en faisant apparaître s, g, $\mathtt{Z}$, ReLU, $\mathtt{X}$ $\tilde{g}$, $\partial s/\partial \mathtt{Z}$, $\widetilde{\text{ReLU}}$ et $\partial s/\partial \mathtt{X}$.

- **montrer que** $\partial s/\partial \mathtt{X}_{kl} = \begin{cases} \partial s/\partial \mathtt{Z}_{kl} & \textbf{si } \mathtt{X}_{kl} > 0 \\ 0 & \textbf{si } \mathtt{X}_{kl} < 0 \end{cases}$
- **la dérivée est-elle définie en 0 ?**

4.3 Dérivée de la fonction de coût : $\tilde{l}$

De manière générale, en utilisant les notations précédemment introduites, la fonction de coût l peut s'écrire :

$$r = \frac{1}{N}\sum_{i=0}^{N-1} -\ln\left(\frac{\exp\left(\mathtt{O}_{i\mathbf{y}_i}\right)}{\sum_{j=0}^{C-1}\exp\left(\mathtt{O}_{ij}\right)}\right) \tag{21}$$

Travail : dessiner le graphe de calcul correspondant à l'équation précédente, en faisant apparaître r, l, $\mathtt{O}$, $\tilde{l}$, $\partial r/\partial \mathtt{O}$.

Travail :

- **montrer que** $\partial r/\partial \mathtt{O}_{ik} = \begin{cases} \frac{1}{N}\left(\mathtt{P}_{ik} - 1\right) & \textbf{si } k = \mathbf{y}_i \\ \frac{\mathtt{P}_{ik}}{N} & \textbf{sinon} \end{cases}$

Les différentes expressions obtenues au cours de ce TD seront utilisées au prochain TP pour implémenter en Python (à l'aide de la bibliothèque Numpy) un MLP.