Emmanuel Franck

Année 2022-2023
Mathématiques Appliquées L3

Informatique S6 Projet "Machine learning"

1 Introduction

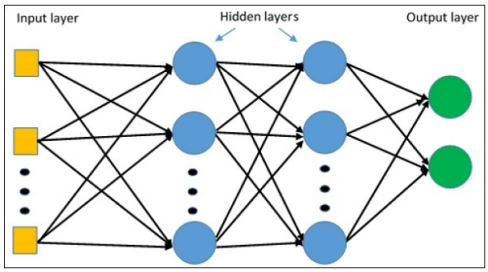
Ce projet se place dans le contexte de l'apprentissage statistique profond très utilisé en intelligence artificielle. L'idée consiste à construire des modèles à partir d'un grand nombre de données. On souhaite étudier le problème suivant

$$\mathbf{F}(\mathbf{X}) = \mathbf{Y}$$

avec $\mathbf{X} = (x_1, ...x_n)$ des données d'entrée, $\mathbf{Y} = (y_1, ...y_m)$ des sorties et $\mathbf{F}$ une fonction inconnue. Dans le cadre de la reconnaissance d'image (par exemple un visage est-il celui d'une femme ou non), les x_i représentent les pixels de l'image (de l'ordre du milieu) et y représente la probabilité que le visage soit celui d'une femme ou non. On suppose qu'il existe une fonction $\mathbf{F}$ qui une transformation, potentiellement très compliquée et non linéaire qui à partir des pixels permet d'obtenir la réponse souhaitée.
Le but de l'apprentissage consiste à déterminer $\mathbf{F}$ (souvent impossible) ou une approximation de $\mathbf{F}$ noté $\mathbf{F}_\omega$ à partir des données d'exemple.
La fonction $\mathbf{F}_\omega$ est une fonction composée de beaucoup de petites transformations simples paramétrer par un ensemble de paramètres ω (il peut en avoir un nombre important). Une fois cette fonction construire on va calculer les coefficients ω en **minimisant** une erreur :

$$\| \mathbf{F}_\omega(\mathbf{X}_e) - \mathbf{Y}_e \|$$

avec $\mathbf{X}_e$, $\mathbf{Y}_e$ des données d'exemples.
Il faut d'abord construire $\mathbf{F}_\omega$ puis calculer les paramètres ω.
Comme fonction $\mathbf{F}_\omega$ on choisit ici les **réseaux de neurones dit MPC**.



Un réseau peut se voir comme une succession de produit matrice-vecteur et de non-linéarité. Cependant on va utiliser une écriture qui utilise un objet intermédiaire appelé un neurone.

Un **neurone** peut s'écrire sous la forme suivante :

$$z = \mathcal{N}(\mathbf{X}) = \sigma((\mathbf{W}, \mathbf{X}) + b)$$

avec z une sortie, $\mathbf{X}$ les données d'entrée et σ une fonction non linéaire. Un réseau de **neurones** est composé de plusieurs **couches** composée de plusieurs **neurones**. Partant d'une entrée $\mathbf{X}$ une couche donne un nouveau vecteur $\mathbf{X}_1$ de (taille potentiellement différente, ou chaque composante de $\mathbf{X}_1$ est donnée par un **neurone**). La couche suivante va reprendre $\mathbf{X}_1$ et donner $\mathbf{X}_2$ et ainsi de suite.

Le 1er travail va porter sur la construction de ces **Réseaux de neurones**.

1.1 Neurones

A partir du cours numéro 3.
Objectif (6 points) : Construire la classe **neurone**.
On souhaite donc construire et gérer des objets du type :

$$z = \sigma((\mathbf{W}, \mathbf{X}) + b)$$

La classe **neurone** contiendra donc comme attributs :

- un entier n_n désignant la taille du vecteur d'entrée,
- deux pointeurs de fonction pour la fonction dite d'activation ($\sigma(x)$ au-dessus) et sa dérivée,
- un tableau (mémoire dynamique) qui contiendra à la fois les poids (W de taille n_n) et le biais.
- un "double" pour stocker la valeur pre-activation pe_a définis par

$$pe_a = (\mathbf{W}, \mathbf{X}) + b$$

un "double" pour stocker la valeur post-activation po_a définie par

$$po_a = \sigma(pe_a)$$

La classe **neurone** contiendra donc comme méthodes/constructeurs :

- Trois constructeurs : nul, par copie et un prenant en paramètre le nombre d'entrées. Ils devront allouer la mémoire,
- un destructeur, un opérateur d'affectation,
- des fonctions "accésseurs", pour lire et écrire la valeur du ième poids ou dérivée de poids, de pe_a, de po_a, lire la taille du vecteur d'entrée, donner la fonction activation et sa dérivée à l'objet,
- des fonctions permettant l'initialisation les poids de façon aléatoire ou avec des 1 et une permettant d'initialiser les dérivées de poids à zéro,
- des tests unitaires pour valider le tout,
- une fonction "*evaluation*", qui utilise les poids et les données d'entrée pour mettre à jour pe_a et po_a.

1.2 Couche

A partir du cours numéro 3.
Objectif (6 points) : Construire la classe **couche**.
On souhaite donc construire et gérer des objets du **couche**. Une **couche** est un ensemble de **neurones**. Ici tous les **neurones** vont avoir les même données d'entrée qui seront les

données obtenues à partir de la **couche** précédente.

La classe **couche** contiendra donc comme attributs :
- un entier désignant le nombre de **neurones**,
- un entier désignant le nombre de données d'entrée,
- un tableau de **neurones**.

La classe **couche** contiendra donc comme méthodes/constructeurs :
- Trois constructeurs : nul, par copie et un prenant en paramètre le nombre de données d'entrée et le nombre de **neurones**. Ils devront allouer la mémoire,
- un destructeur, un opérateur d'affectation,
- des fonctions "accésseurs", pour lire et écrire la valeur du jème poids ou dérivée de poids du ième **neurone**, de pe_a, po_a du ième neurone, obtenir la taille du vecteur d'entrée, obtenir le nombre de **neurones**, donner la fonction activation et sa dérivée à l'objet au ième **neurone**,
- une fonction permettant de sommer une valeur à la valeur courante de la jème dérivée de poids du ième **neurone**,
- des fonctions permettant l'initialisation les poids en appelant les fonctions d'initialisation des **neurones**,
- une fonction pour appliquer la fonction d'activation du ième **neurone** à un double x. Idem pour la dérivée de la fonction d'activation,
- des tests unitaires pour valider le tout,
- une fonction "*evaluation*", prend un vecteur **X** et renvois un vecteur **Y** après avoir evaluer la **couche** (ce qui consiste à évaluer les **neurones**),
- surcharger l'opérateur "()" qui prend un numéro de **neurone** et renvoit le **neurone**.

1.3 Classe Réseaux "feed-forward"

A partir du cours numéro 5 et 9.
Objectif : construire la classe feed-forward (6 points). Dans un premier temps on souhaite écrire la classe "feed-foward" ou "réseaux acyclique". Fondalement un réseau est enchainement de couche. Il s'agira d'un template de classe avec trois paramètres du template :
- la taille du vecteur d'entrée n_i,
- la taille du vecteur de sortie n_o,
- le nombre de couches internes n_{hi}.

La classe contiendra comme attribut :
- un pointeur vers un tableau qui contiendra le vecteur d'entrée,
- un pointeur vers un tableau de couche (taille $n_{hi} + 1$),
- un entier contenant le nombre total de poids.

Si il existe zéro couche interne on a donc une seule couche de neurones avec comme nombres de neurones le nombre de sortie. Dans cette classe il faudra coder :
- un constructeur par défaut et par copie, un constructeur prenant une "loss" comme paramètre, un destructeur,
- une fonction pour initialiser les poids (elle peut utiliser celle de "couche"),

— une fonction pour mettre à zéro les dérivées des poids (elle peut utiliser celle de "couche"),
— la surcharge de l'opérateur "()" pour accéder à une couche,
— une fonction qui calcule le nombre total de poids et une qui le renvoit,
— une fonction qui permet de mettre le contenu d'un vecteur dans les poids et une qui permet de mettre les poids dans un vecteur,
— la fonction "evaluation" qui, pour une donnée $\mathbf{X}^j$ renvoie $\mathbf{Y}^j = F_\omega(\mathbf{X}^j)$. Il s'agit de faire l'évaluation des neurones (calcul de la sortie à l'aide des poids et de la fonction d'activation) couche par couche.
— une fonction virtuelle "construire" qui construira le réseau.

1.4 Classe Réseaux multi-perceptron

A partir du cours numéro 8 et 9.
Objectif : construire la classe mpc (3.0 points). On écrira un template de classe "MPC" avec les même paramètres que la classe au-dessus. Cette classe sera héritée de "feed-forward". Elle ne contiendra pas de données supplémentaires. Dans cette classe il faudra coder :

— Les constructeurs et destructeurs qui appellent ceux de la classe mère,
— des fonctions permettant de donner à chaque couche une fonction d'activation et la dérivée,
— la fonction "constuire" qui crée un MPC. Elle prend comme paramètres un tableau du nombre de neurones pour chaque couche interne. Le principe est simple il s'agit d'un réseau totalement connecté. Cela veut dire que : chaque couche à comme nombre d'entrées le nombre de neurones de la précédente (taille de l'entrée $\mathbf{X}$ dans le cas de la première couche). La dernière couche à comme nombre de neurones la taille de la sortie.