

Cahier de laboratoire

Quentin LIEUMONT

20 mai 2019

Table des matières

1	Le 13/05/2019 :	2
1.1	Quentin LIEUMONT :	2
2	Le 14/05/2019 :	3
2.1	Quentin LIEUMONT :	3
3	Le 15/05/2019 :	4
3.1	Quentin LIEUMONT :	4
4	Le 16/05/2019 :	5
4.1	Quentin LIEUMONT :	5
5	Le 17/05/2019 :	7
5.1	Quentin LIEUMONT :	7
6	Le 20/05/2019 :	8
6.1	Quentin LIEUMONT :	8

1 Le 13/05/2019 :

1.1 Quentin LIEUMONT :

Le matin :

Ce matin, Roman BRESSON, un thesard m'a présenté sa thèse :

Données utilisées :

- Data set public
- Entre 6 et 15 critères.
- Entre 200 et 1600 donnés

Des informations m'ont été communiquées :

- Choquet de base en python
- Ctypes pour importer des fonctions C dans python
- Un article pour comprendre les integrales de choquet [1]

Le réseau de neurones utilisé est configuré de la maniere suivante :

- Vecteur X en entrée
- Réseau de neurone -> dit si une le choix est "bon" ou pas.
- Extraction du réseau : les 'ui' et les 'wjj' de par son architecture.
- Regression : testé avec des gaussienne et des sigmoïdes
- Modele : trois sommes :

$$C_n(X) = \sum_{i=1}^n w_i \times x_i + \sum_{i=1}^n \sum_{j=i+1}^n \left(w_{M\ ij} \times \max(x_i, x_j) + w_{m\ ij} \times \min(x_i, x_j) \right) \quad (1)$$

- Probleme :

On ne sait pas combien en mettre de regression pour etre au plus proche des donné sans faire de l'overfeeting.

L'après midi :

Un premier réseau de neurones à été créé afin que je me familiarise avec ces nouvelles notions. J'utilise la librairie keras [2] (Python).

Le premier reseau est assez simple : Il fait une somme pondérée. C'est la première partie de la formule1, j'ai aussi trouvé un lien pour faire le max/min de plusieurs noeuds mais je ne sait pas l'implementer.

2 Le 14/05/2019 :

2.1 Quentin LIEUMONT :

Le matin :

La matinée a été passée à finaliser le travail d'hier. J'ai aussi développé un package afin d'encapsuler la librairie Keras et donc de simplifier son utilisation. Voici la forme du réseau :

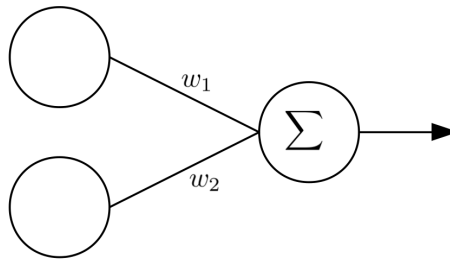


FIGURE 1 – Réseau de neurone très simple

Le réseau va essayer de faire varier les poids w_1 et w_2 afin de coller avec la fonction qui doit être apprise.

L'après midi :

On peut maintenant simplement faire apprendre au réseau une fonction simple. Par exemple la moyenne :

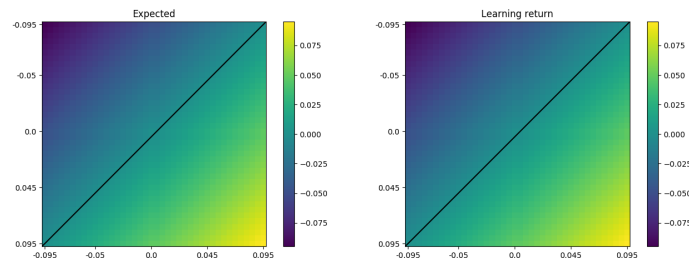


FIGURE 2 – Apprentissage de la moyenne

Le graphique de droite est celui attendu et celui de gauche celui obtenu. On peut voir que la descente de gradients se fait à merveille. Aucune différence n'est visible : en effet les poids associés aux deux nœuds sont les suivants :

$$w_1 = 0.50000010 \quad w_2 = 0.50000024$$

On peut voir qu'aux approximations processeur, les poids sont les bons pour faire la moyenne de deux nombres :

$$m = \frac{n_1 + n_2}{2} = 0.5 \times n_1 + 0.5 \times n_2$$

3 Le 15/05/2019 :

3.1 Quentin LIEUMONT :

Le matin :

L'ensemble des commandes ont été recodées et commentées afin de simplifier leur utilisation.

Une librairie Useful a été créée, elle contient les fichiers suivants :

- `functions.py` : Un module contenant les fonctions utiles de nombreuses fois.
- `data.py` : Un module permettant de générer des données.
- `network.py` : Un module permettant de générer un réseau de neurone adaptatif.
- `simpleNetwork.py` : Un module permettant de générer en une ligne un réseau simple de régression.

L'après midi :

Le module `choquet.py` a été codé : Il permet :

- De configurer une fonction qui applique la formule (1) (class Choquet).
- De générer des données qui collent avec cette fonction (class ChoquetData).
- De générer un réseau de neurones qui régresse cette fonction grâce à ces données (class ChoquetNetwork).

La génération des données est cependant très longue (environ 2m40). Il serait bien de générer une base de données ce soir afin d'accélérer le calcul. Les fonctions `write_json` et `get_json` ont été importées afin de simplifier la création de la base de données.

Le soir :

Après réflexion, la class ChoquetData ne présente pas de valeur ajoutée, elle a donc été fusionnée avec la class Data. On peut sauvegarder des données via la méthode `Data.save(lien)`.

4 Le 16/05/2019 :

4.1 Quentin LIEUMONT :

Le matin :

Réunion :

- Environ n^2 vecteurs avec n le nombre de dimensions pour l'apprentissage.
- Très important que les poids soit positifs (sinon overfitting).
- La somme des poids doit valoir 1.
- Les entrées sont comprises entre 0 et 1.

Pour implementer les conditions precedement citées, la matinée a été passée a coder. La fonction de perte peut être redéfinie. Différentes ont été testé, la fonction suivante marche relativement bien :

$$loss(expected, returned) = |expected - returned| + |1 - ||W||| \quad (2)$$

L'après midi :

Un probleme se pose :

Prenons la formule (1) avec un $n = 2$ (pour simplifier, mais c'est généralisable) :

$$C_n(X) = \sum_{i=1}^n w_i \times x_i + \sum_{i=1}^n \sum_{j=i+1}^n \left(w_{Mij} \times \max(x_i, x_j) + w_{mij} \times \min(x_i, x_j) \right)$$

$$C_2(X) = w_1.x_1 + w_2.x_2 + w_M.\max(x_1, x_2) + w_m.\min(x_1, x_2)$$

Etant donné que les données d'apprentissage sont des réels randoms indépendents entre 0 et 1 :

$$P(x_1 > x_2) = P(x_1 < x_2) = \frac{1}{2} \quad (3)$$

On obtient donc :

$$\begin{aligned} C_2(X) &= w_1.x_1 + w_2.x_2 + w_M.\max(x_1, x_2) + w_m.\min(x_1, x_2) \\ < C_2(X) > &= w_1.x_1 + w_2.x_2 + w_M.\frac{x_1 + x_2}{2} + w_m.\frac{x_1 + x_2}{2} \end{aligned}$$

On obtient donc :

$$< C_2(X) > = x_1 \times \left(w_1 + \frac{w_m + w_M}{2} \right) + x_2 \times \left(w_2 + \frac{w_m + w_M}{2} \right) \quad (4)$$

Le réseau vas donc essayer d'atteindre les valeurs (4) sans garantir la véracité des coeficients. Cela est particulièrement visible sur le graphique suivant :

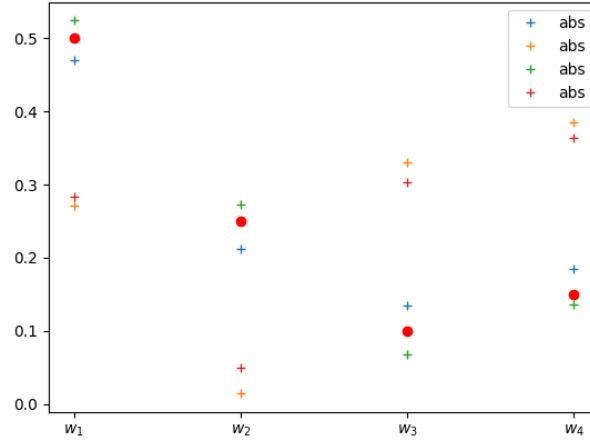


FIGURE 3 – Entrainement de 4 réseaux

Les points rouge representent les valeurs réelles, les croix representent un entraînement par couleur. La fonction d'entraînement est la fonction (2)

Voici le graphique de 4 entraînements indépendents du même réseau de neurones. Il est bien visible que deux quadruplets de valeurs adherent :

Les bonne valeurs : $(0.5, 0.25, 0.1, 0.15)$

Les valeurs : $(0.28, 0.2, 0.33, 0.37)$

Si on applique la formule (4) on obtient bien :

Vecteur	$w_1 + \frac{w_m + w_M}{2}$	$w_2 + \frac{w_m + w_M}{2}$
$(0.5, 0.25, 0.1, 0.15)$	62.5	37.5
$(0.28, 0.2, 0.33, 0.37)$	63	37

TABLE 1 – Valeurs retournées par les réseaux

On peut voir que les resultats sont sensiblement similaires : le réseau adhère a de mauvaises valeurs. Pour résoudre ce probleme, il faut ne pas satisfaire (3) et donc tirer un learning set statistiquement différent du testing set.

5 Le 17/05/2019 :

5.1 Quentin LIEUMONT :

Le matin :

Pour résoudre le problème d'hier, une piste est envisagée : Avant l'entraînement, les données sont triées. Cela permet de créer deux sets différents, le premier aura une moyenne plus faible que le second. J'ai aussi ajouté mis le paramètre shuffle à False dans l'entraînement du modèle.

L'après midi :

Le tri avant de passer à l'apprentissage a l'air d'améliorer les résultats. Cet après midi a été essentiellement passé à créer des plots pour visualiser les données : Voici les résultats (moyenne \pm écart type) de 100 entraînements avec des données triées et non triées.

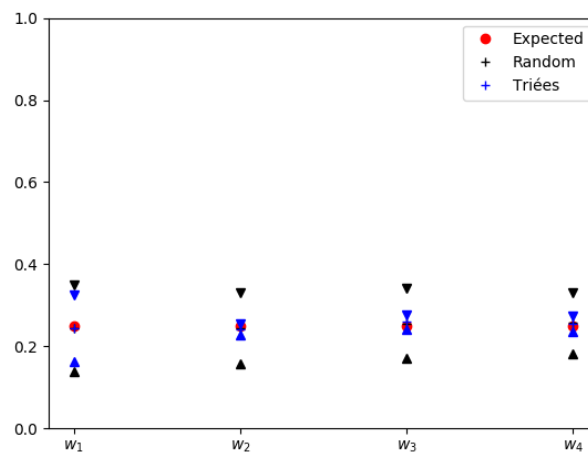


FIGURE 4 – Apprentissage d'une fonction de Choquet

On peut voir que le tri affecte l'apprentissage, pour mieux savoir à quel point, de nouvelles mesures sont nécessaires.

6 Le 20/05/2019 :

6.1 Quentin LIEUMONT :

Le matin :

La matinée à été passée a comenter le code afin de simplifier son utilisation.
Ee nouvelles loss functions ont été codées.

L'après midi :

Une réunion a eut lieu avec Johanne : les résultats sont bons mais pour pouvoir dire plus de choses, il serait bien de tracer un graphique ecart type de l'erreur de l'apprentissage en fonction de : la taille de la base de donnée, le trie des données et la normalisation dans la loss function.
De nouvelles fonctions ont donc été codées dans ce but.

Références

- [1] M. Grabisch and C. Labreuche, “Fuzzy measures and integrals in mcda,” in *Multiple Criteria Decision Analysis*, pp. 553–603, Springer, 2016.
- [2] “KERAS : The python deep learning library.” <https://keras.io/>.