Cahier de laboratoire

Quentin Lieumont

23 mai 2019

Table des matières

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

Le réseau de neurones utilisé est configuré de la maniere suivante :
- Vecteur X en entré
- Réseau de neurone -> dit si une le choix est "bon" ou pas.
- Extraction du réseau : les 'ui' et les 'wij' de par son architecture.
- Regression : testé avec des gaussiene et des sigmoides
- Modele : trois sommes :

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

- 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 familliarise avec ces nouvelles notions. J'utilise la librairie keras [?] (Python).

Le premier reseau est assez simple : Il fait une somme pondérée. C'est la première partie de la formule??, 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 matiné a été passée a finaliser le travail d'hier. J'ai aussi developé un package afin d'encapsuler la librairie Keras et donc de simplifier son utilisation. Voici la forme du réseau :

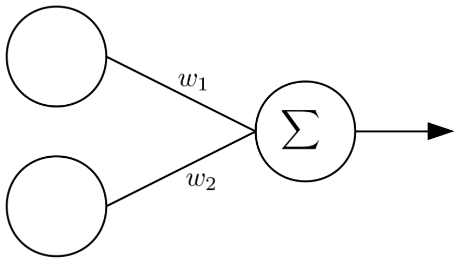

Figure 1 – Réseau de neurone très simple

Le reseau vas essayer de faire varier les poids w_1 et w_2 afin de coller avec le fonction qui doit etre aprise.

L'après midi :

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

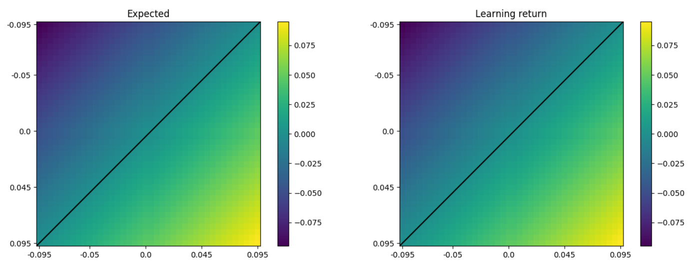

Figure 2 – Apprentissage de la moyenne

Le graphique de droite est celui attendu et celui de gauche celui obtenu. On peut voir que la descente de gradients se fait a merveille. Aucunes difference ne sont visibles : en effet les poids associés aux deux noeuds sont les suivants :

$$w_1 = 0.50000010 \qquad w_2 = 0.50000024$$

On peut voir qu'aux aproximations 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 commantées afin de simplifier leur utilisation.
Une libraire Useful à été créée, elle contient les fichiers suivants :

functions.py : Un module contenant les fonctions utiles de nombreuses fois.

data.py : Un module permetant de generer des données.

network.py : Un module permetant de generer un reseau de neurone adaptatif.

simpleNetwork.py : Un module permetant de generer en une ligne un reseau simple de regression.

L'après midi :

Le module choquet.py à été codé : Il permet :

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

La generation des données est cependant tres longue (environ 2m40). Il serait bien de generer une base de données ce soir afin d'accelerer le calcul. Les fonctions write_json et get_json ont été importées afin de simplifier la creation de la base de données.

Le soir :

Apres reflexion, la class ChoquetData ne presente pas de valeur ajouté, elle a donc été fusionné 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 :

Réunuion :

- Environ n^2 vecteurs avec n le nombre de dimensions pour l'apprentissage.
- Très important que les poids soit positifs (sinon overfiting).
- 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 :

$$loss(expected, returned) = |expected - returned| + |1 - ||W||\,| \tag{2}$$

L'après midi :

Un probleme se pose :
Prennons la formule (**??**) avec un $n = 2$ (pour simplifier, mais c'est généralisable) :

$$C_n(X) = \sum_{i=1}^{n} w_i \times x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Big(w_{M\,ij} \times \max(x_i, x_j) + w_{m\,ij} \times \min(x_i, x_j) \Big)$$
$$C_2(X) = w_1.x_1 + w_2.x_2 + w_M.\max(x_1, x_2) + w_m.\min(x_1, x_2)$$

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

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

On obtient donc :

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

On obtient donc :

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

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

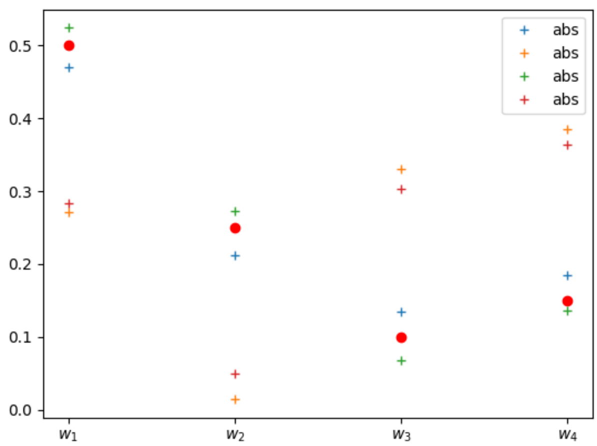

FIGURE 3 – Entrainement de 4 réseaux
*Les points rouge representent les valeurs réelles, les croix representent un entrainement par couleur. La fonction d'entrainement est la fonction (**??**)*

Voici le graphique de 4 entrainements 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 (**??**) 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 sensiblements similaires : le réseau adhère a de mauvaises valeurs. Pour résoudre ce probleme, il faut ne pas satisfaire (**??**) 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 probleme d'hier, une piste est envisagée :
Avant l'entrainement, les données sont triées. Cela premet de créer deux sets different, le premier auras une moyene plus faible que le second. J'ai aussi ajouté mis le parametre shuffle a False dans l'entrainement du model.

L'après midi :

Le trie avant de passer a l'apprentissage a l'air d'ameliorer les resultats. Cet après midi a été essentiellement passé a creer des plots pout visualiser les données : Voici les résultats (moyenne $\pm$ ecart type) de 100 entrainement avec des données trié et non trié.

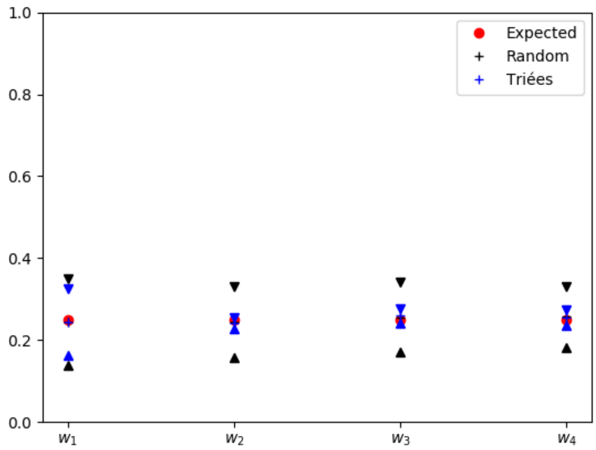

Figure 4 – Apprentissage d'une fonction de choquet

On peut voir que le trie affecte l'apprentissage, pour mieux savoir a quel point, de nouvelles mesures sont nescessaires.

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.

Théoriquement, l'écart type devrais diminuer quand la taille de la base de donnée augmente.

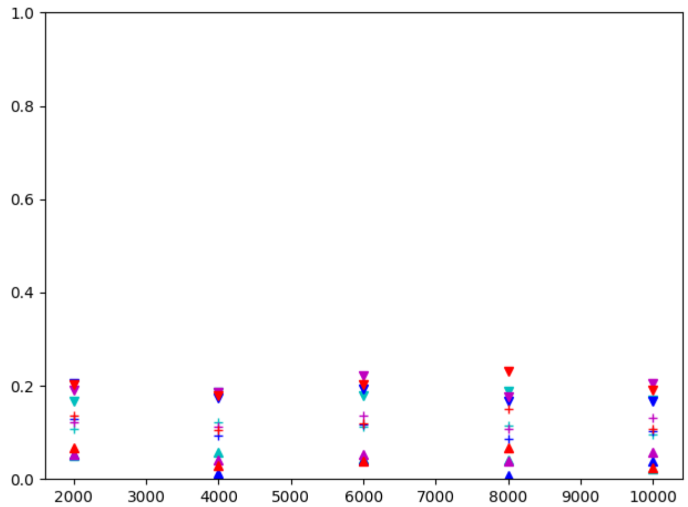

Figure 5 – Ecart type de l'apprentissage en fonction de la taille de la base de données

On peut bien voir qu'avec les données ci dessus, l'echantillon est trop petit pour voir une variation de l'ecart type : les diferences ne sont pas significatives. On refait l'experience avec les tailles suivantes en entrée : 10, 50, 100, 500, 1000, 5000, 10000. Le calcul prend du temps, j'aurais les resultats demain matin.

7 Le 21/05/2019 :

7.1 Quentin Lieumont :

Le matin :

Voici le graphique attendu :

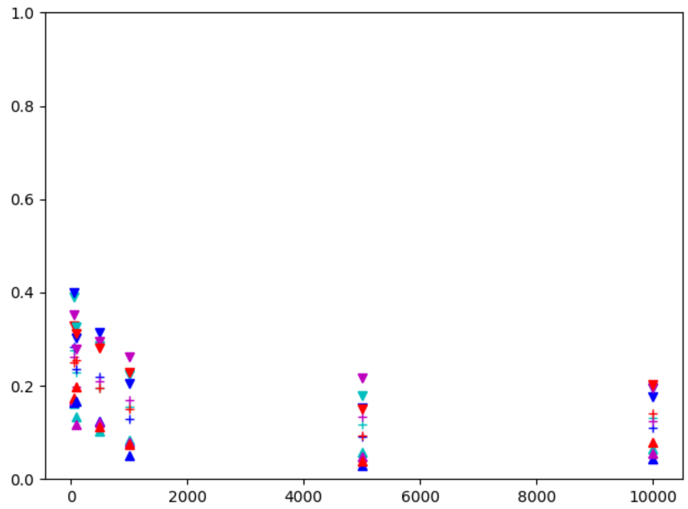

Figure 6 – Ecart type de l'apprentissage en fonction de la taille de la base de données

Ici, contrairement à la Fig.??, une décroissance est visible.

L'après midi :

Le reste de la journée à été passée a générer des données. Le programme à été legerement modifié : Au lieu de creer un graphique directement, les données sont stoquées en JSON. On peut ensuite creer un graphique à partir de ces données.

8 Le 22/05/2019 :

8.1 Quentin Lieumont :

Le matin :

Des graphiques ont étés générés a partir des données :

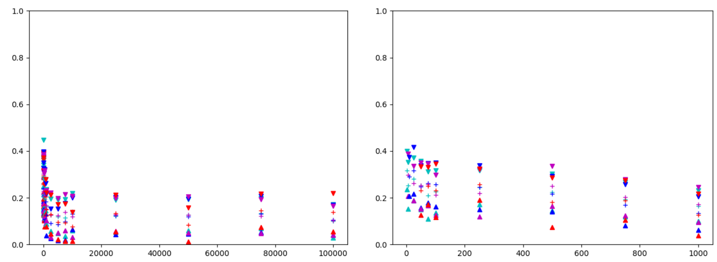

Figure 7 – Ecart type de l'apprentissage en fonction de la taille de la base de données

On peut voir ici que l'erreur d'aprentissage s'amoindrit avec l'augmentation de la taille du learning set. Cette évolution s'arette vers les 1000 données, dépassé ce cap, l'écart type stagne.

L'après midi :

Tout un module a été codé afin de visualiser les données. Une variante des premiers graphiques a été réalisé : Un graphique a taille de base de donnée fixée de l'écart type de l'apprentissage en fonction du nombre d'aprensitssages réalisés par le réseau. Cette technique permet de s'afranchir du problème posé le 16 mai en ne stagnant pas dans un minimum local.

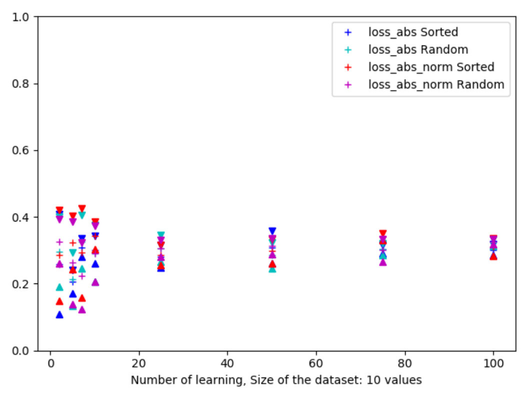

Figure 8 – Ecart type de l'apprentissage moyen en fonction du nombre d'aprensitssages

Ici on peut voir que la moyenne ne varie pas significativement mais l'ecart type diminue drastiquement. Ce parametre combiné au precedent permetra d'augmenter la précision du réseau.

9 Le 23/05/2019 :

9.1 Quentin Lieumont :

comment

Le matin :

Une réunion à été faite :
— Faire un réseau simple.
— Lui faire apprende la fonction $1/3\,x + 2/3\,y$.
— Ajouter du bruit.

L'après midi :

Le réseau essaye une fonction toute simple :

$$\frac{1}{3} \times x + \frac{2}{3} \times y \tag{5}$$

Les poids 1/3 et 2/3 ont été choisit pour casser la simétrie. On y rajoute une perturbation random equiprobable entre $-err$ et err. On essaye de visualiser l'erreur d'apprentissage en fonction de err.
Un petit script à donc été codé afin de visialiser la variation de l'erreur d'aprentissage :

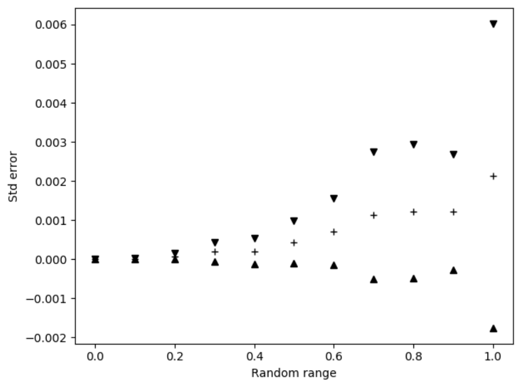

Figure 9 – Variation d'aprensitssage en fonction de la perturbation

On peut voir que même si l'erreur moyenne augmente exponentiellement avec la perturbation, elle reste très faible tant que la perturbation ne dépasse pas la moitié de la valeur théorique :

$$err < 1/2 \quad \Rightarrow \quad R^2 > 0.999 \tag{6}$$

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/`.