# Rapport de stage
# Algorithmes de minimisation du regret et integrales de choquet
## LABORATOIRE DE RECHERCHE EN INFORMATIQUE

Quentin Lieumont

18 Juin 2019

Sous la direction de : Johanne Cohen
Martine Thomas

# Table des matières

# 1 Introduction

Mon stage c'est déroulé au laboratoire de recherche en informatique, son sujet était le suivant : *Algorithmes de minimisation du regret et integrales de choquet*. Je n'avais jamais fait d'informatique fondamentale (exepté IA) et n'ayant jamais vus la théorie de la mesure, il m'était d'aprhender en totalité la notion d'intégrale de choquet (découlant des intégrale de Lebesgue). C'est pourquoi, durant ce rapport, la partie mathématique théorique ne sera pas vue en detail. Organisation du document

# 2 Présentation générale du LRI

Le Laboratoire de Recherche en Informatique (LRI) est une unité de recherche de l'Université Paris-Sud et du CNRS. Créé il y a plus de 40 ans, le laboratoire est localisé sur le plateau du Moulon depuis début 2013. Il accueille plus de 250 personnes dont environ un tier de doctorants.

Organisés en neuf équipes, les recherches du laboratoire incluent à la fois des aspects théoriques et appliqués (ex : algorithmique, réseaux et bases de données, graphes, bioinformatique, interaction homme-machine, ect). De part cette diversité, le laboratoire favorise les recherches aux frontières de différents domaines, là où le potentiel d'innovation est le plus grand. En plus de son activité de publication (2000 publications entre début 2008 et mi 2013), le LRI dévelope de nombreux logiciels.

Pour plus d'informations, je vous invite à aller regarder sur leur site ou une présentation détaillée est continuellement tenue à jour [1].

# 3 Équipe GALaC

Durant mon stage, j'ai travaillé sous la direction de Johanne Cohen qui est la responsable de l'équipe GALaC.
L'équipe GALaC est une équipe du LRI composée d'environ 20 chercheurs issus du cnrs, de l'Université Paris-Sud et de Centrale Supelec qui travaillent sur des thématiques théoriques comme l'algorithmique, la théorie des graphes ou les systèmes en réseaux. Mon travail s'intègre dans cette équipe puisqu'il réside dans l'étude du fonctionnement d'un aglorithme.

# 4 Contexte et motivations

Le but du stage était de manipuler les intégrales de choquet et les résaux de neurones. La prédiction du prix des maisons à été étudiée grâce à une base de donnée trouvée sur internet [2]. On pourait tout simplement multiplier le prix au metre carré par la superficie de la maison mais on verra par la suite que cette technique est loin d'être optimale.

# 5 Un petit point théorique...

Afin de mieux comprende les interactions entre les differentes notions, un petit point théorique est necessaire :

— Compréhention du fonctionnement d'un réseau de neurones
— Fonctionnement des intégrales de choquet
— Pourquoi les integrales de choquet associées aux résaux de neurones sont particulierement efficace pour résoudre certains problèmes

## 5.1 Réseaux de neurones

Les médias parlent souvent d'intéligence artificielle et de réseaux de neurones, ces deux notions sont radicalements différentes mais elles sont souvent mélangées et confondues sur la place publique...

L'intéligence artificelle est un concept informatique, un paradigme de programation, cette notion n'a pas été abordée durant mon stage, il n'en serra pas question ici. Cependant si vous voullez en savoir plus, je vous redirige vers la chaine youtube de Lê Nguyên Hoang. Anciennement chercheur en mathématique et aujourd'hui vulgarisateur sur internet et à l'EPFL ses vidéos sont à regarder sans moderation [3].

Un réseau de neurones est une architecture informatique inventée en 1950 et remis à la mode grâce aux travaux de Yan Le Cun durant les années 1980 permetant de faire des regressions de fonctions, de la généralisation et de l'optimisation. Il est composé, comme son nom l'indique, de neurones mis en réseau grace à des connexions.

### 5.1.1 Un neurone

Un neurone est une unité de base du réseau, il se decompose en trois phases :

— L'entrée
— La fonction interne
— La fonction d'activation

Elles s'agencent de la manière suivante :

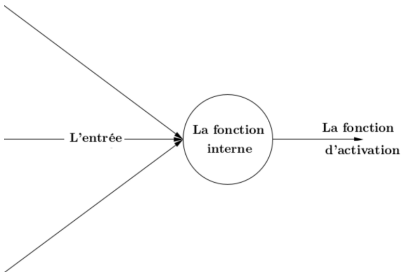


FIGURE 1 – Un neurone

**L'entrée :** Un neurone prend des informations en entrée, de manière générale un nombre réel entre 0 et 1 mais d'autres objets sont envisageables (image, pixel, son...). Il n'y a pas de nombre minimum ou maximums (on pourait imaginer un neurone ne prenant pas d'entrée, ou au contraire en prenant une infinité), ni de contrainte sur les differents objets.

**Exemple(s) :**

Un neurone prenant :
- les trois valleur de coulleur d'un pixel (entier entre 0 et 255).
- les trois valleur de coulleur d'un pixel (réel $[0, 1]$).
- une sequence ADN en entrée

On note le vecteur entrée $X$ et chacun de ses element $x_1 \dots x_i \dots x_n$.

**La fonction interne :** Le réseau vas alors faire un calcul à partir des entrées et de poids, des valleurs définies pour chaque neurones qui peuvent varier durant l'apprentissage.
Le vecteur poids se note $W$ et chacun de ses element $w_1 \dots w_i \dots w_n$.

**Exemple(s) :**

$f(X) = W.X = \sum_{i=1}^{n} w_i \times x_i$
$f(X) = e^{w_1+x_1} \times w_2$
$f(X) = \max(X)$
$f(X) =$ "nombre de A sur les $w_1$ premières bases de $x_1$"
(avec $x_1$ une sequence ADN)

On note la fonction interne $f(X)$ (avec $X$ le vecteur d'entrée).

**La fonction d'activation :** Comme vus precedement, ces neurones sont mis en résau, il est donc interessant d'avoir une norme pour l'entré et la sortie afin de pouvoir lier des neurones entre eux sans distinctions.
La norme qui a été choisie est, comme précedement cité, un réel entre 0 et 1.

Or, il peut être remarqué que les fonctions ci dessus ne renvoient pas forcement des nombre entre 0 et 1... On utilise donc la fonction d'activation afin de normaliser les sorties.

**Exemple(s) :**

Pour le cas precedent sur l'ADN : $f(X)/w_1$
Pour une fonction dans $\mathbb{R}$ : fonction sigmoide
Pour une fonction dans $[0, 1]$ : fonction identité

On note la fonction d'activation $f_{act}$.

**Pour resumer :** Un neurone prend *des entrées*, les passe dans sa *fonction principale*, puis le résultat dans la *fonction d'activation.*
Formellement, on obtient l'équation suivante :

$$f_{act}(f(X)) \tag{1}$$

Le tout dépendant bien évidement du vecteur $W$ que l'on peut faire varier afin de modifier la fonction du neurone.

**Exemple(s) :**

Prenons un neurone avec deux entrées : $x_1$ et $x_2$,
de fonction principale $f(X) = X.W$
et de fonction d'activation identité.
Si on a $W = (1, 1)$, ce neurone fait une somme.
Mais si $W = (0.5, 0.5)$ ce neurone fait une moyenne.

### 5.1.2 Un réseau

Une fois que nous avons de nombreux neurones faisant chacuns des actions bien spécifiques, on voudrais les relier afin de gérer des comportements plus complexes comme faire une somme de moyenne (ou controller un drone, prédire les mouvements boursiers...).

Il sufit alors de relier les neurones entre eux, de maniere générale de manière linéaire de gauche à droite. De nombreuse manière de relier les neurones existent (recurent, convolution...), ici on ne parlera que de la connexion dite "Dense" ou "fully connected" : le réseau se découpe en $n$ couches de neurones, chacunes composée de neurones dont les entrées sont les sorties des neurones le la couche précédente.

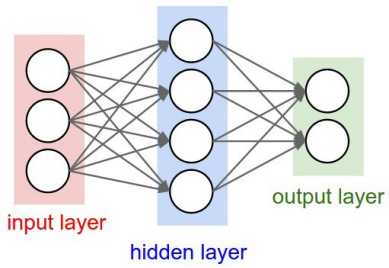


FIGURE 2 – Réseau dense simple

techburst.io/experiment-finding-objects-with-a-neural-network-caa4cec7d2c4

Comme on peut le voir sur la figure, les neurones les plus à gauches sont les neurones d'entrée (capteurs) et ceux les plus à droite ceux de sortie (controle des moteurs, affichage d'une note...). Tout les autres neurones sont "cachés", il n'intéragissent pas directement avec l'environement.

### 5.1.3 L'apprentissage

Jusqu'ici on a vus comment creer une architecture modulable permetant de générer une fonction à paramètres. Mais une question se pose toujours :
Quel est l'intéret ? Pourquoi ne pas directement écrire la fonction "en dur" ?
La réponse tient en trois mots : *Descente de gradient stochastique* (ou SGD en anglais).
Ce concept est ce qui fait que les réseaux de neurones sont les architectures favorites des data scientists et des chercheurs en IA.

L'idée est assez simple, on recherche une fonction générale dont on ne connais que certaines valeurs discretes. On vas commencer par définir une fonction nomé "loss function" qui décrit la précision du réseau actuel : elle prend en entrée deux valeurs : la valeur théorique et la valeur obtenue. elle retourne un réel positif, plus il est faible, plus le réseau est proche de la fonction théorique.

**Exemple(s) :**

$$loss(exp, obt) = |exp - obt|$$
$$loss(exp, obt) = (exp - obt)^2$$

Le probleme de regression se résume alors à minimiser la fonction de prete, c'est ici que la descente de gradient stochastique fait son entrée :

**Descente de gradient :** Pour minimiser loss, on aimerai bien descendre sa pente, c'est à dire dériver suivant les diférentes variables [1], en déduire l'orientation de la pente, et la descendre. Étant donné que de manière générale, il y a plusieurs variables, la dérivée se transforme en gradient. Des applications linéaires sont privilégiées dans les fonctions des neurones, afin de pouvoir calculer facilement les gradients.

**Stochastique :** On a donc expliqué d'ou vient la descente de gradient, mais que veut dire stochastique ? Ce mot est synonyme de hasard. En effet le temps de calcul du gradient est exponentiel vus que chaque neurones est relié a $n$ neurones, eux même reliés à $n$ autres neurones ect...
Alors lorsque le réseau est grand (un réseau peut facillement atteindre le million de neurones voir même des milliards [4]) il est inimaginable de calculer la totalité du gradient (les calculs peuvent parfois depasser plusieurs fois la durée de l'univers...), on utilise alors le gradient stochastique.

1. NB : Ici les variables sur lesquels on intervient sont les $w_i$ (pas les $x_i$).

Cette descente de gradient est bien plus rapide, il est donc possible de l'iterer de nombreuses fois pour tenter de minimiser la fonction de perte. L'espaces des solutions étant rarement idéal (rarement une "cuvette" mais constitué de "bosses" et de "trous" chaotiques) il n'est pas obligatoire d'arriver à atteindre le minimum global mais au moins minimum local sera atteind. Avec plusieurs apprentissages on peut alors approximer très efficacement quasiment toutes les fonctions.

## 5.2 Intégrales de choquet

L'integrale de choquet est une intégrale découlant de la théorie de la mesure [5]. Ici on ne parlera que du modele discret.

Pour l'expliquer simplement, prenons un exemple : Suposons que l'on veuille conseiller des personnes sur l'achat d'ordinateurs. Ces personnes de conaissent absolument rien en informatique. La seul chose qu'ils veullent est une note, plus elle est élevée, plus l'ordinateur est performant. Essayons de faire un algorithme assez simple pour résoudre ce problemme : Chaque composant se voit atribué une note (en fonction de la puissance, la qualité de fabrication...), on appelle cette note *l'utilité* du composant. Une fois toutes les utilitées étudiées, on donne un poid à chaque famille de composants (RAM, processeur, carte mère...). Enfin on fait la somme de utilité fois poid pour tout les composant.
De ce fait, si un ordinateur a de meilleurs composant, plus de puissance, ect, sa note sera superieure.

Ce modèle parait raisonable dans la plupart des cas mais il est extraimement mauvais dans les cas extremes : Suposons qu'un constructeur peu scrupuleux propose un ordinateur assez étrange : Le processeur le moins cher du marché (assez mauvais) mais énormément de RAM, par exemple 128Go. Votre classement le placera forcément en haut de la liste, même si vous en conviendrez, cet ordinateur est assez inutile...

Il faudrait donc trouver un autre système modélisant les interactions entre les composants. Ce modèle est exactement celui sur lequel j'ai travaillé durant mon stage : *les intégrales de choquet*. En plus de donner un poid aux utilité, un poid est donné aux interactions des utilitées par le biais des fonctions min et max. Ces intéractions peuvent être faites deux à deux, trois à trois voir à plus. Mais pour des raison de temps de calcul, ces interactions se sont limités à l'interactions simple (deux à deux) durant ce travail. En effet, la taille du calcul évolue exponentiellement avec les interactions.

Voici donc la fonction que l'on vas vouloir aproximer :

$$C(X) = \sum_{i=1}^{n} w_i \times x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Big( w_{M\,ij} \times \max(x_i, x_j) + w_{m\,ij} \times \min(x_i, x_j) \Big) \quad (2)$$

Avec $X$ le vecteur des utilitées et $W$, $W_m$ et $W_M$ les vecteurs des poids.

# 6 Matériel et méthodes

Ce travail s'est déroulé en deux principales phases : Une étape de developement durant laquelle des données et la fonction à apprendre ont été générées. Et une étape plus appliquée durant laquelle une base de donnée réelle à été étudiée.

Afin d'implémenter informatiquement les concepts théoriques vus précédement, il a été choistit d'utiliser la librairie Keras [6], une reference en python pour faire du machine learning. La partie sur les intégrales de choquet à été entierement recodée en python avec la librairie numpy [7] pour optimiser le temps de calcul. L'ensembe du code est disponible gratuitement et sous licence libre sur github [8].

# 7 Keras

La librairie Keras est une interface Python/TensorFlow [9] permetant de manipuler des réseaux de neurones. Ici, seul les fonctionnalitées principales seront étudiées, a savoir création d'un réseau simple, regression par SGD et évaluation de performances.

Voici un exemple de réseau de neurone simple (FIGURE 3) qui va tenter d'apprendre l'application linéaire suivante : $f(X) = W.X$. Avec :
$X$ **:** le vecteur d'entrée tel que : $\dim(X) = 2$.
$W$ **:** le vecteur de poids tel que : $w_1 = 0.2$ et $w_2 = 0.8$.

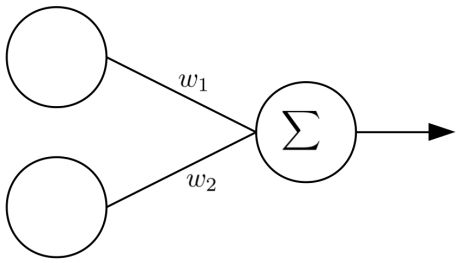


FIGURE 3 – Réseau simple
*Avec* $w_1 = 0.2$ *et* $w_2 = 0.8$.

Pour créer ce réseau et lui faire apprendre la fonction precedement citée, le code suivant est nescessaire :

```
1  # imports
2  from keras import Sequential, optimizers, layers
3  import numpy as np
4  from random import random
5
6  def f(X):                    # La fonction que l'on cherche
7      W = np.array([0.2, 0.8])# Le vecteur poids
8      return W @ X             # Produit scalaire
9
10 act = "linear"   # Fonction d'activation
11 n_input = 2      # Nombre de neurones
12 # generation des questions/reponses attendues
13 questions = np.array([np.array([random(), random()])
```

```
                    for i in range(10000)])
reponses  = np.array([f(q) for q in questions])

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=
    True)
neurones = layers.Dense(1, activation=act, input_dim=n_input,
    use_bias=False)
model = Sequential()              # On cree un reseau
model.add(neurones)               # On lui ajoute des neurones
model.compile(optimizer=sgd,      # On compile le tout
              loss="mean_squared_error")
model.fit(questions, reponses)  # On essaye de coller aux donn es

for i, weight in enumerate(model.get_weights()[0]):
    # on affiche les poids
    print(f"w{i} : {weight[0]}")
```

code/reseau1.py

En executant ce code une fois, on obtient :

```
w0 : 0.19988934695720673
w1 : 0.8001111745834351
```

Comme on peut le voir ci-dessus, les résultats obtenus sont proches de ceux attendus (0.2 et 0.8). La syntaxe de la librairie keras est cependant assez lourde. Une librairie à donc été codée pour simplifier son utilisation. Elle pourra être appelée avec de nombreux paramtres qui seront abordés dans les parties suivantes.

Pour tester la robustesse de cet apprentissage, une fonction avec perturbations à été étudiée : Une fonction simple à été générée comme precedement, il y a été ajouté une perturbation random equiprobable. L'erreur d'apprentissage en fonction de la valeur de cette perturbation à alors été étudiée.

**Un probleme se pose :** Prennons l'équation (2) avec, par exemple, un vecteur de taille 2 :

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

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

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 :

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

$$\mathbb{E}(C_2(X)) = w_1.x_1 + w_2.x_2 + w_M.\frac{x_1 + x_2}{2} + w_m.\frac{x_1 + x_2}{2}$$

On obtient donc :

$$\mathbb{E}(C_2(X)) = x_1 \times \left(w_1 + \frac{w_m + w_M}{2}\right) + x_2 \times \left(w_2 + \frac{w_m + w_M}{2}\right) \qquad (4)$$

Le réseau va donc tenter d'atteindre les valeurs solutions de l'équation (4) sans garantir l'exactitude des coeficients.

**Exemple(s) :**

Les deux quadruplets de valeurs suivantes vont genrer ce probleme :
Les bonne valeurs : $(0.5, 0.25, 0.1, 0.15)$
D'autres valeurs : $(0.28, 0.2, 0.33, 0.37)$
Si on applique l'équation (4) sur cet exemple, on obtient :

| 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

Les resultats du tableau precedent sont similaires : en moyenne, le réseau ne les differencieras pas. Il va donc apprendre a de mauvaises valeurs car en moyenne, le réseau à meilleurs résultats qu'avec des poids aléatoires. C'est un minimum local de la fonction de perte.

Pour résoudre ce probleme, il faut ne pas satisfaire l'equation (3) pour suprimer l'équation (4) et donc tirer un learning set statistiquement différent du testing set. De ce fait, si le réseau apprend ces mauvais poids, il serat instantanément pénalisé par le testing set.

Dans l'optique de minimiser le temps de calcul, differentes fonctions de pertes ont été testées et comparées (moindres carrés et erreur absolue). Il à été étudié la variation de la precision du réseau en fonction des differentes fonctions de perte, de si les données était triées et de la taille de la base de donnée.

# 8 Implementation d'un réseau de choquet

Nous appellerons ici *réseau de choquet* un réseau de neurones ayant une architecture adaptée a la regression d'une intégrale de choquet. Comme vus precedement (5.2), une fonction de choquet a une architecture complexe. Voici l'intégrale de choquet entierement dévelopée pour un vecteur d'entrée taille 3 :

$$\begin{aligned} C \ &= w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3 \\ &+ w_{m1} \times \min(x_1, x_2) + w_{m2} \times \min(x_1, x_3) + w_{m3} \times \min(x_2, x_3) \\ &+ w_{M1} \times \max(x_1, x_2) + w_{M2} \times \max(x_1, x_3) + w_{M3} \times \max(x_2, x_3) \end{aligned}$$

Voici l'architecture d'un réseau de choquet entierement générée par un réseau de neurones :

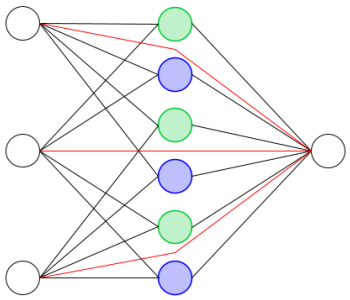

FIGURE 4 – Architecture d'un réseau de choquet
*En bleu, des neurone appliquant pour fonction principale* $\min(X)$.
*En vert, des neurone appliquant pour fonction principale* $\max(X)$.

On peut voir que trois problemes non triviaux se posent (*cf.* 5.1) :
— Les neurones collorées n'appliquent pas une simple application linéaire.
— Les neurones ne sont pas reliés en mode Dense mais en convolution.
— Certains neurones passent des informations en sautant une couche de neurones.

**Une autre piste à alors été envisagée :** Créer un réseau simple comme dans la figure 5. Dans ce réseau, aucuns neurone n'a de fonction complexe : ceux de gauche sont les entrées et celui de droite fait le produit scalaire avec un vecteur poids.

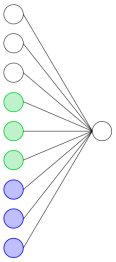

FIGURE 5 – Réseau alternatif

Ce réseau est bien plus simple à générer : c'est un réseau *Dense* de taille 9 en entrée. Et plus généralement $n^2$ pour un vecteur de taille $n$.

**Démonstation :**

Prenons un vecteur de taille $n$, Le but est d'énumerer le nombre de neurones utiles a l'équation (2). Le resultat est le suivant :

$$n + \binom{n}{2} + \binom{n}{2} = n + 2 \times \frac{n(n-1)}{2} = n^2 \tag{5}$$

Il faut alors créer un vecteur d'entrée à partir une base de donnée d'aprentissage. Cela se fait simplement en concaténant le vecteur $X$ avec les elements pris deux à deux passés dans les fonctions min et max.

Lors de la descente de gradient, le réseau traite les poids indiférement, pour les récupérer les poids de l'équation (2) il sufit de prendre les $n$ premiers pour $W$, les $\frac{n(n-1)}{2}$ suivant pour $W_m$ et les $\frac{n(n-1)}{2}$ derniers pour $W_M$.

# 9 Données réelles

Une base de donnée disponible en ligne sur KAGGLE à aussi été étudiée [10]. La base de donnée traite des prix de maisons vendues entre 2014 et 2015 dans le King Country, WA, USA.

Pour rechercher les fonctions d'utilitées, une liste de fonctions possible à été dressée :

- Polynomes de degrés allant de 1 à 3
- Exponentiel
- Logarithme
- Hyperbolique
- Racine
- Sigmoide
- Gaussienne

Une fois toutes ces fonctions codées, chaque plage de données à essayé d'être regressée grâce à celles ci. La fonction obtenant le meilleur $R^2$ à été conservé si celui ci était superieur à 0.3 afin de retirer les données n'agissant pas sur le prix.

En utilisant les fonctions d'utilitées trouvées precedement, une réseau de choquet à été créé puis entrainé sur cette base de données.

# 10 Résultats et discution

## 10.1 Moyenne :

Il a été demandé au réseau de regresser une fonction moyenne à deux dimentions : Les poids qui sont censé etre obtenus sont les suivants :

$$m(X) = \frac{x_1 + x_2}{2} = 0.5 \times x_1 + 0.5 \times x_2 \tag{6}$$

On peut voir ci dessous un graphique à 3 dimentions des valeurs atendues (gauche) et obtenues (droite) :

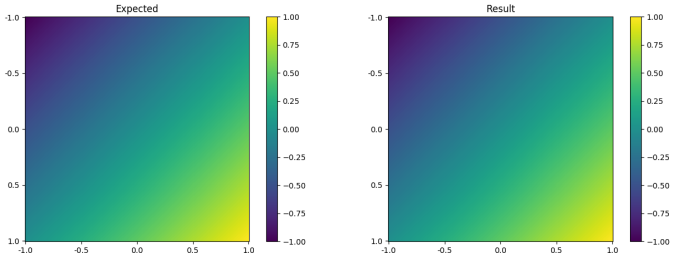


FIGURE 6 – Apprentissage de la moyenne
*Les axes $x$ et $y$ corespondent respectivement à $x_1$ et $x_2$. La valeur en tout point de la fonction est représentée par un gradient de couleur dont l'echelle est sur la droite du graphique.*

Aucune difference n'est visible, en effet les poids associés aux deux noeuds sont les suivants :

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

Aux aproximations processeur, les poids sont les mêmes que dans (6).

### 10.1.1 Étude de l'apprentissage :

Théoriquement, l'écart type de l'erreur d'apprentissage devrais diminuer quand la taille de la base de donnée augmente. Pour tester cette hypothèse, des graphiques ont étés générés avec differentes tailles de base de données :

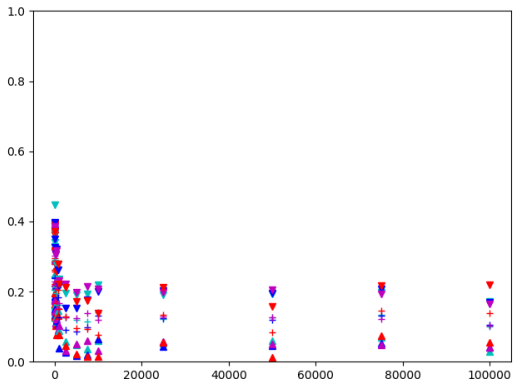


FIGURE 7 – Écart type de l'apprentissage en fonction de la taille de la base de données

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

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 ne pas stagner dans un minimum local.

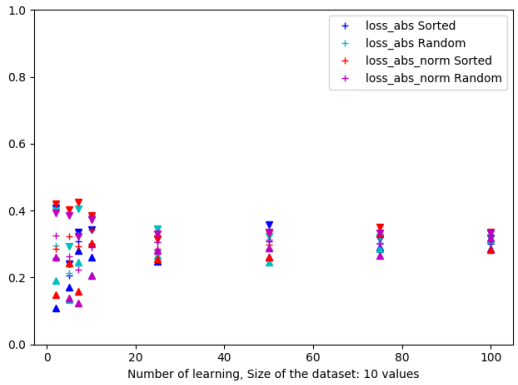


FIGURE 8 – Ecart type de l'apprentissage moyen en fonction du nombre d'aprensitssages

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 dans la partie 10.3.

## 10.2 Étude de la resistance aux perturbations

Le réseau essaye de regresser une fonction toute simple :

$$\frac{1}{3} \times x + \frac{2}{3} \times y \tag{7}$$

Ici, les poids 1/3 et 2/3 ont été choisit pour casser la simétrie. Une perturbation random équiprobable entre $-err$ et $err$ est ajouté aux données. Voici un graphique de l'erreur de l'aprentissage en fonction de cette erreur (allant de 0 à $\pm$ le maximum de la fonction) :

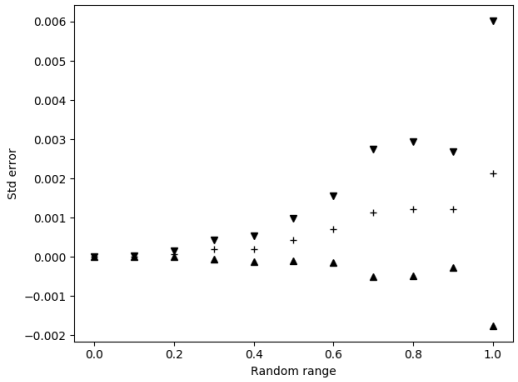


FIGURE 9 – Variation d'aprensitssage en fonction de la perturbation

Même si l'erreur moyenne augmente exponentiellement avec la perturbation, elle reste extremement faible $R^2 > 0.99$.

Cet exemple illustre très bien l'extrême robustesse des réseaux de neurones aux perturbations. Ce réseau n'auras donc pas trop de mal à apprendre sur des données réelles (souvent très ébruitées).

## 10.3 Base de données réelle

Les données ont été passées dans un réseau de neurones sans traitement specifique. Voici le resultat de l'apprentissage :

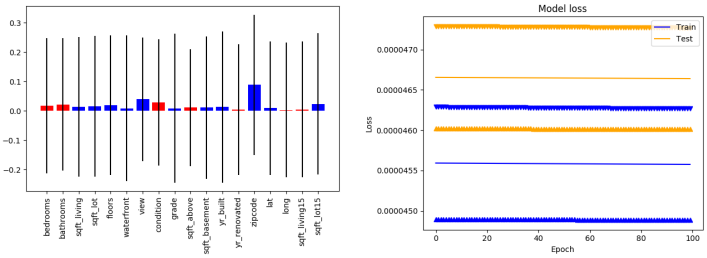


FIGURE 10 – Apprentissage sur les données réelles

L'apprentissage est ici totalement cahotique. Ce resultat était attendu étant donné que les fonction d'utilité n'ont pas été calculées et que le prix d'une maison n'est pas un calcul simple. Ici, il est impossible de calculer le prix avec un simple $Surface \times Prix/m^3$.

Les fonctions d'utilitées ont donc été calcullées. Voici le resultat :

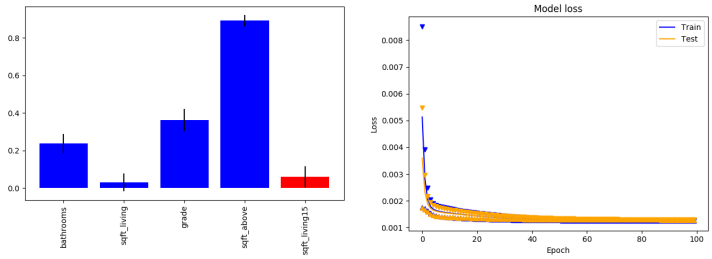


FIGURE 11 – Apprentissage sur les données réelles

Il peut être remarqué peut que l'aprentissage est bien plus précis. En effet, l'espace des poids etant bien plus restraint, le réseau explore cet espace bien plus rapidement.

Les résultats sont donc passés dans un réseau de choquet afin d'en calculer tout ces coefficients :

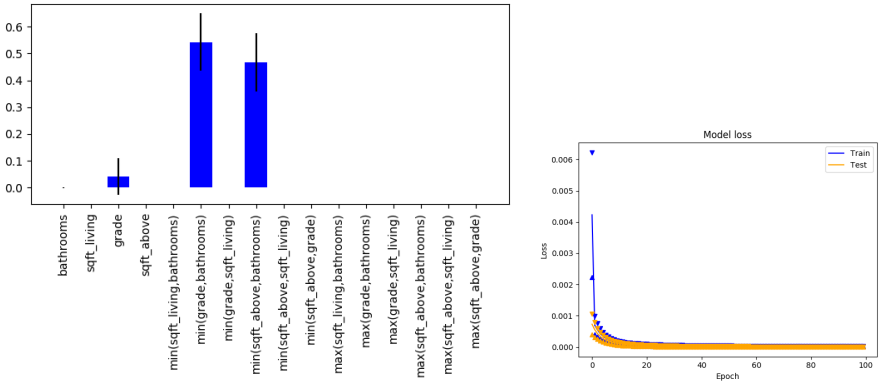


FIGURE 12 – Apprentissage avec le réseau de choquet

En observant la FIGURE 12, il est visible que seul deux paramètres ont des valeurs significativement differentes de 0. Ces parametres sont les suivants :

min(note, nombre de salles de bains) : La note est un indicateur donné par l'état sur le niveau de la maison.
Et les maisons ayant beaucoups de salles de bains sont plus cher dans cette base de données.

min(superficie, nombre de salles de bains) : On retrouve ici la taille de la maison, mais ce n'est pas le facteur déterminant pour donner un prix à une maison.

# 11 Conclusion

Il faut cependant nuancer les résultats obtenus dans la partie 10.3. En effet, il se peut que cette base de donnée soit biaisée, il n'est donc pas possible de tirer des conclusions générales à partir de ce cas particulier. De plus, toutes les fonctions d'utilitées n'ont pas été obtenues. Il est possible que certaines fonctions qui ne sont pas présentes dans 9 soient d'exelentes fonctions d'utilitées.

**Developement personel :**

Ce stage m'a appris de nombreuses choses sur le monde de la recherche tel que la manipulation de modèles mathématiques théoriques et leurs applications en informatique fondamentale.

J'ai aussi pu implementer mon premier réseau de neurones. J'avais déjàs des connaissances théoriques mais je n'avais jamais eut l'occasion de manipuler ce genre d'objets.

L'utilisation scientifique de python à aussi été une part très importante de mon stage. Des problématiques qui n'avaient jamais été abordées en cours tel que l'optimisation ce sont possée. Les échanges avec les autres stagiaires/thésards ont étés très instructifs (découverte de nouvelles librairies, bonnes méthodes à adopter...).

Enfin, durant ce stage, j'ai pu échanger avec des universitaires et des enseignants chercheurs sur ma poursuite d'études. J'ai par exemple pu découvrir les doctorats CIFRE [11] : Ce sont des doctorats en partenariat avec une entreprise c'est donc l'une des meilleures orientations pour faire de la recherche developement au sein d'une entreprise.

# Références


[1] "Laboratoire de recherche en informatique." www.lri.fr.

[2] "House sales in king county, USA." kaggle.com/harlfoxem/housesalesprediction.

[3] "Science4all." www.youtube.com/playlist?list=PLtzmb84AoqRTl0m1b82gVLcGU38miqdrC.

[4] "Spectrum ieee." spectrum.ieee.org/tech-talk/computing/software/biggest-neural-network-ever-pushes-ai-deep-learning.

[5] A. Fallah Tehrani, C. Labreuche, and E. Hüllermeier, "Choquistic utilitaristic regression," 11 2014.

[6] "Keras : The python deep learning library." https://keras.io/.

[7] "Numpy." www.numpy.org.

[8] "Github." github.com/QuentinN42/Stage_LRI.

[9] "Tensorflow." tensorflow.org.

[10] "Kaggle." kaggle.com.

[11] "CIFRE : faire son doctorat en entreprise." orientation-education.com/article/cifre-doctorat-entreprise.