Rapport de stage
Algorithmes de minimisation du regret et intégrales 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 s'est déroulé au laboratoire de recherche en informatique (LRI), à Orsay. Son sujet était le suivant : *Algorithmes de minimisation du regret et intégrales de Choquet*. Je n'avais jamais fait d'informatique fondamentale (exepté IA en autodidacte) et n'ayant jamais vu la théorie de la mesure, il m'était d'appréhender 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 détail.

Ce papier se décompose en cinq parties : Premièrement, une introduction présentera le LRI ainsi que mon environement de travail. Le sujet du stage sera expliqué avec un point sur le vocabulaire technique. Ensuite, la partie materiel et méthodes présentera les expériences menées. Enfin les résultats seront présentées et discutés. Ce rapport se terminera par une conclusion de la recherche et une discussion des apprentissages de ce stage.

2 Cadre du stage

2.1 Présentation du laboratoire

Le Laboratoire de Recherche en Informatique 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 tiers 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, bio-informatique, interaction homme-machine...). 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 développe 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].

2.2 Mon équipe d'acceuil

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 algorithme.

3 Contexte et motivations

Le but du stage était de manipuler les intégrales de Choquet, un modèle mathématique d'aide a la décision et les réseaux de neurones. La prédiction du prix des maisons en fonction de leurs caracteristiques a été étudiée grâce à une base de données disponible sur internet [2].

4 Définition des différents cvoncetps

Afin de mieux comprendre les interactions entre les différentes notions, un petit point est nécessaire :

— Comprendre un réseau de neurones
— Comprendre les intégrales de Choquet
— Associer les intégrales de Choquet aux réseaux de neurones efficacement pour résoudre certains problèmes

4.1 Réseaux de neurones

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

L'intelligence artificielle est un concept informatique, un paradigme de programmation, cette notion n'a pas été abordée durant mon stage, il n'en sera pas question ici. Cependant si vous voulez en savoir plus, je vous redirige vers la chaine youtube de Lê Nguyên Hoang. Anciennement chercheur en mathématiques 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 permettant de faire des régressions de fonctions, de la généralisation et de l'optimisation. Il est composé, comme son nom l'indique, de neurones mis en réseau grâce à des connexions.
Il va donc être d'abord abordé la notion de neurone, puis la mise en réseau.

4.1.1 Un neurone

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

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

Elles s'agencent de la manière suivante :

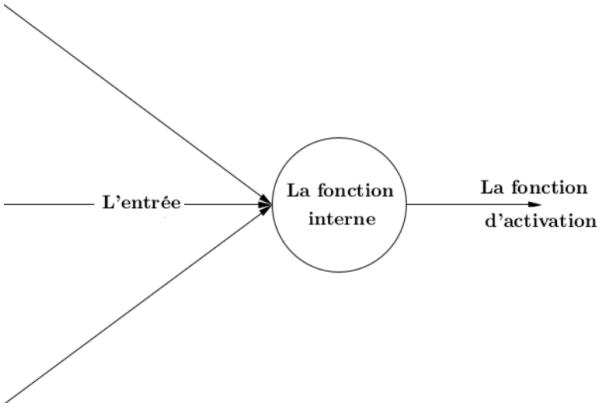

FIGURE 1 – Un neurone

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 maximum (il peut être conçu un neurone ne prenant pas d'entrée, ou au contraire en prenant une infinité), ni de contrainte sur les différents objets.

Exemple(s) :

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

Le vecteur entrée est noté X et chacun de ses element $x_1 \dots x_i \dots x_n$.

La fonction interne : Le réseau va donc faire un calcul à partir des entrées. Soit $f(X)$ la fonction interne du réseau (avec X le vecteur d'entrée). La fonction peut contenir des paramètres notés $w_1 \dots w_i \dots w_n$ et contenus dans le vecteur W. Ces valeurs définies pour chaque neurones peuvent varier durant l'apprentissage.

Exemple(s) :

$f(X) = W \cdot X = \sum_{i=1}^{n} w_i \times x_i$
$f(X) = \max(X)$
$f(X) =$ "nombre d'adénine sur les w_1 premières bases de x_1"
(avec x_1 une sequence ADN)

La fonction d'activation : Rappelons que ces neurones sont mis en réseau. Il est donc intéressant d'avoir une norme pour l'entrée et la sortie afin de pouvoir lier des neurones entre eux sans distinctions. La norme qui a été choisie est, comme précédement introduit, un réel entre 0 et 1.

Or, il peut être remarqué que les fonctions ci-dessus ne renvoient pas forcement des nombres strictements compris entre 0 et 1. La fonction d'activation est donc utilisée pour normaliser les sorties.

Exemple(s) :

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

La fonction d'activation est noté f_{act}.

Pour resumer : Un neurone prend des valeurs en *entrée*, les transforme via sa *fonction principale*, puis retourne le résultat grâce à la *fonction d'activation*. Formellement, l'équation suivante est obtenue :

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

Le tout dépendant bien évidemment du vecteur W qui varie 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 $W = (1, 1)$, ce neurone fait une somme.
Mais si $W = (0.5, 0.5)$ ce neurone fait une moyenne.

4.1.2 Un réseau

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

Il suffit donc de relier les neurones entre eux, de manière générale de manière linéaire de gauche à droite. De nombreuse manière de relier les neurones existent (récurent, convolution...), ici il sera abordé en détail 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.

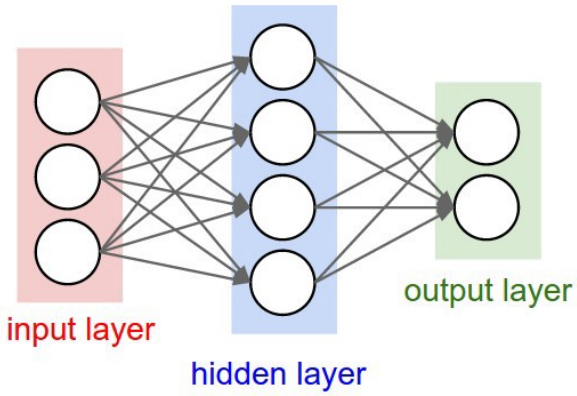

FIGURE 2 – Réseau dense simple
techburst.io/experiment-finding-objects-with-a-neural-network-caa4cec7d2c4

La FIGUE 2 représente un réseau fully connected : 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...). Tous les autres neurones sont "cachés", ils n'intéragissent pas directement avec l'environnement.

4.1.3 L'apprentissage

Jusqu'ici il a été abordé la création d'une architecture modulable permettant de générer une fonction à paramètres. Mais une question se pose toujours :
Quel est l'intérêt ? Pourquoi ne pas directement écrire la fonction "en dur" ?
La réponse tient en quatre 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, une fonction générale est cherché à partir de certaines valeurs discretes. Posons une fonction nommé "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) :

Avec exp le résultat attendu et obt le résultat obtenu :

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

Le problème de régression se résume donc à minimiser la fonction de perte, c'est ici que la descente de gradient stochastique fait son entrée :

Descente de gradient : Pour minimiser loss, il serait bien de descendre sa pente, c'est-à-dire dériver suivant les différentes variables [1], en déduire l'orientation de la pente et la descendre. Étant donnés 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 : Il a été expliqué d'où 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 vu que chaque neurones est relié a n neurones, eux même reliés à n autres neurones...
Donc lorsque le réseau est grand (un réseau peut facilement 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...), le gradient stochastique est donc utilisé pour rendre le calcul réalisable.

Cette descente de gradient est bien plus rapide, il est donc possible de l'itérer de nombreuses fois pour tenter de minimiser la fonction de perte. L'espace 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 atteint. Avec plusieurs apprentissages il peut donc être approximé très efficacement quasiment toutes les fonctions.

1. NB : Ici les variables suivant lesquelles la dérivée se fait sont les w_i (pas les x_i).

4.2 Intégrales de Choquet

L'intégrale de Choquet est une intégrale découlant de la théorie de la mesure [5]. Ici modèle discret est utilisé.

Pour l'expliquer simplement, prenons un exemple : Supposons qu'une entreprise veuille conseiller des personnes sur l'achat d'ordinateurs. Ces personnes ne connaissent absolument rien en informatique. La seule chose qu'ils veulent est une note, plus elle est élevée, plus l'ordinateur est performant. Essayons de faire un algorithme assez simple pour résoudre ce problème : Chaque composant se voit attribué une note (en fonction de la puissance, la qualité de fabrication...), cette note est nommée *l'utilité* du composants. Une fois toutes les utilités étudiées, un poids est associé à chaque famille de composants (RAM, processeur, carte mère...). Enfin la somme des utilités fois poids pour tous les composant permet d'obtenir cette note.
De ce fait, si un ordinateur a de meilleurs composant ou plus de puissance, sa note sera supérieure.

Ce modèle parait raisonnable dans la plupart des cas, mais il est permettra mauvais dans les cas extrêmes : Supposons 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. Ce classement le placera forcément en haut de la liste, même si cet ordinateur est assez mauvais...

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 poids aux utilités, un poids est donné aux interactions des utilités par le biais des fonctions *min* et *max*. Ces interactions peuvent être faites deux à deux, trois à trois voir à plus. Mais pour des raisons de temps de calcul, ces interactions se sont limitées à l'interaction simple (deux à deux) durant ce travail. En effet, la taille du calcul évolue exponentiellement avec les interactions.

Voici donc la fonction qui va être approximée durant cette étude :

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

Avec X le vecteur des utilités et W le vecteur des poids sans intéraction, W_m le vecteur des poids des intéraction *min* et W_M le vecteur des poids des intéraction *max*.

5 Matériel et méthodes

Ce travail s'est déroulé en deux principales étapes : La première de développement durant laquelle des données et la fonction à apprendre ont été générées. Et la seconde plus appliquée durant laquelle une base de données réelle a été étudiée.

Afin d'implémenter informatiquement les concepts théoriques vu précédemment, il a été choisi d'utiliser la librairie Keras [6], une réference en python pour faire du machine learning. La partie sur les intégrales de Choquet a été entièrement 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].

5.1 Keras

La librairie KERAS est une interface Python/TensorFlow [9] permettant de manipuler des réseaux de neurones. Ici, seul les fonctionnalités principales seront étudiées, à savoir création d'un réseau simple, régression 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 : $f(X) = W \cdot X$. Avec :
X **:** le vecteur d'entrée tel que : $\dim(X) = 2$.
W **:** le vecteur de poids tel que : $w_1 = 0.2$ et $w_2 = 0.8$.

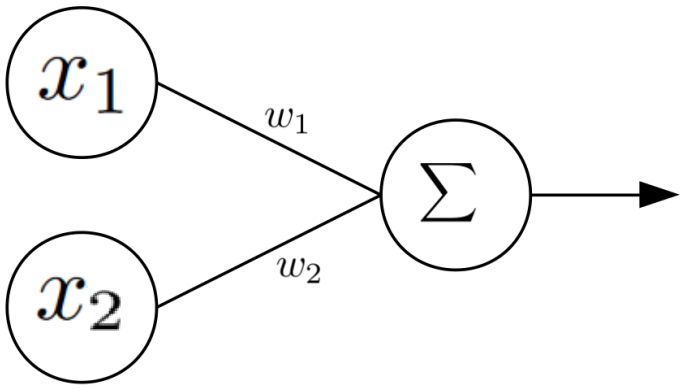

FIGURE 3 – Réseau simple
Avec $w_1 = 0.2$ *et* $w_2 = 0.8$.

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

```python
# imports
from keras import Sequential, optimizers, layers
import numpy as np
from random import random

def f(X):                       # La fonction que l'on cherche
    W = np.array([0.2, 0.8])# Le vecteur poids
    return W @ X                # Produit scalaire

act = "linear"   # Fonction d'activation
n_input = 2      # Nombre de neurones
# generation des questions/reponses attendues
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)  # Creation des neurones
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 exécutant ce code une fois, le résultat suivant est obtenu :

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

Comme il peut être remarqué 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 a donc été codée pour simplifier son utilisation. Elle pourra être appelée avec de nombreux paramètres qui seront abordés dans les parties suivantes.

Pour tester la robustesse de cet apprentissage, une fonction avec perturbations a été étudiée : Une fonction simple a été générée comme précédemment, il y a été ajouté une perturbation aléatoire équiprobable. L'erreur d'apprentissage en fonction de cette valeur a donc été étudiée.

Un problème se pose : Reprenons l'équation (2) :

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

Si on prend un vecteur de taille $n = 2$, le résultat suivant est obtenu :

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

Étant donné que les données d'apprentissage sont des réels aléatoires indépendants entre 0 et 1 :

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

Donc en combinant (2) et (3) :

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

Et donc :

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

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

Exemple(s) :

Les deux quadruplés de valeurs suivantes vont genrer ce problème :
Les bonne valeurs : $(0.5, 0.25, 0.1, 0.15)$
D'autres valeurs : $(0.28, 0.2, 0.33, 0.37)$
Si l'équation (4) est appliquée sur cet exemple, le résultat suivant est obtenu :

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 résultats du tableau précédent sont similaires : en moyenne, le réseau ne les différenciera 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 problème, il faut ne pas satisfaire l'équation (3) pour supprimer 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 sera instantanément pénalisé par le testing set.

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

5.2 Implementation d'un réseau de Choquet

Nous appellerons ici *réseau de Choquet* un réseau de neurones ayant une architecture adaptée a la régression d'une intégrale de Choquet. Comme vu précédemment (4.2), une fonction de Choquet a une architecture complexe. Voici l'intégrale de Choquet entièrement développée 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 entièrement générée par un réseau de neurones :

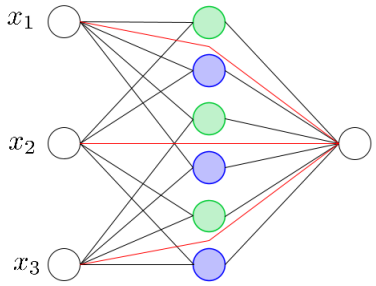

FIGURE 4 – Architecture d'un réseau de Choquet
En bleu, des neurone de fonction principale $\min(X)$.
En vert, des neurone de fonction principale $\max(X)$.

Ici, trois problèmes non triviaux se posent (*cf.* 4.1) :
— 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.

Comme vus dans la partie 4.1, cette architecture est assez complexe. Il serait donc pratique de trouver une architecture plus simple afin de gagner du temps en implémentation et en exécution.

Une autre piste à donc 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.

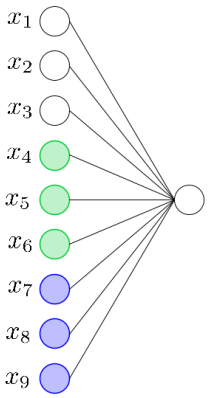

FIGURE 5 – Réseau alternatif

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 en entrée.

Observation :

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

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

Il faut donc créer un vecteur d'entrée à partir une base de données d'apprentissage. 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.

5.3 Données réelles

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

Pour rechercher les fonctions d'utilités, une liste de fonctions possible a 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 régressée grâce à celles-ci. L'efficacité de la regression a été mesurée grâce aux moindres carrés (mean squared error ou R^2). La fonction obtenant le meilleur R^2 a été conservé pour chaque pages de données. Afin de retirer les données n'agissant pas sur le prix, les fonctions d'utilités présentant un R^2 inférieur à 0.3 ont été retirées.

En utilisant les fonctions d'utilités trouvées précédemment, un réseau de Choquet a été créé puis entrainé sur cette base de données. L'entrainement est cependant long, environ 8h sur cluster de calcul du LRI. Ce temps a pu être réduit grâce à de l'optimisation et de la programmation parallèle.

6 Résultats et discution

Dans cette partie vont être présentés les résultats en deux sections : premièrement les tests sur des données générées artificiellement puis une seconde section sera dédiée aux résultats obtenus sur la base de données en ligne.

6.1 Données artificielles :

Dans cette partie, un exemple simple d'apprentissage sera étudié. Ensuite, la résistance aux perturbations sera observée. Enfin une étude des performances de cette méthode d'apprentissage sera discutée.

6.1.1 Moyenne :

Il a été demandé au réseau de régresser une fonction moyenne à deux dimensions : Les poids qui sont censés être obtenus sont les suivants :

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

Le graphique à trois dimensions des valeurs attendues (gauche) et obtenues (droite) est le suivant :

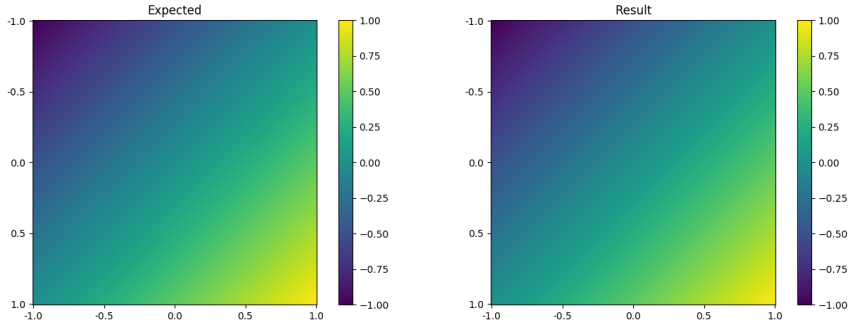

FIGURE 6 – Apprentissage de la moyenne
Les axes x et y correspondent respectivement à x_1 et x_2. La valeur en tout point de la fonction est représentée par un gradient de couleur dont l'échelle est sur la droite du graphique.

Ici aucune variation n'est observable. Les deux gradients sont équivalents, en effet les poids associés aux deux noeuds sont les suivants :

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

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

6.1.2 Étude de la resistance aux perturbations

Le réseau essaye de régresser 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'apprentissage en fonction de cette erreur (allant de 0 à $\pm$ le maximum de la fonction) :

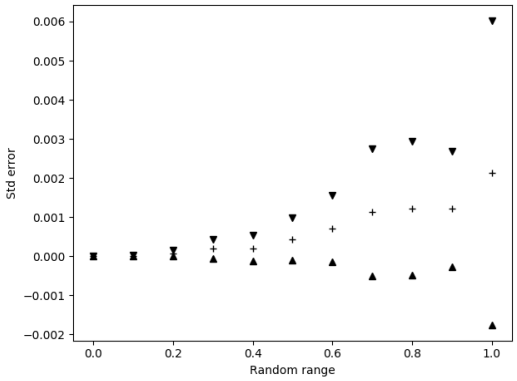

FIGURE 7 – Variation d'apprensitssage en fonction de la perturbation

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).

6.1.3 Étude de l'apprentissage :

Théoriquement, l'écart type de l'erreur d'apprentissage devrais diminuer quand la taille de la base de données augmente. Pour tester cette hypothèse, de nombreux apprentissages ont été réalisées avec différentes tailles de base de données. Les résultats obtenus sont présentés sur le graphique suivant.

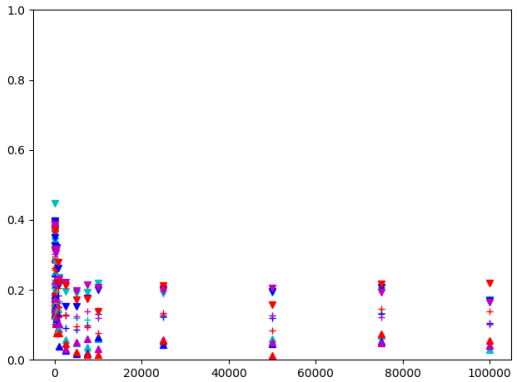

FIGURE 8 – Écart type de l'apprentissage en fonction de la taille de la base de données avec écart type

Ici, l'erreur d'apprentissage s'amoindrit avec l'augmentation de la taille de la base de données. Cette évolution s'arête vers les 1000 données, dépassé ce cap, l'écart type stagne.

Un graphique a taille de base de données fixée de l'écart type de l'apprentissage en fonction du nombre d'apprensitssages réalisés par le réseau. Cette technique permet de ne pas stagner dans un minimum local.

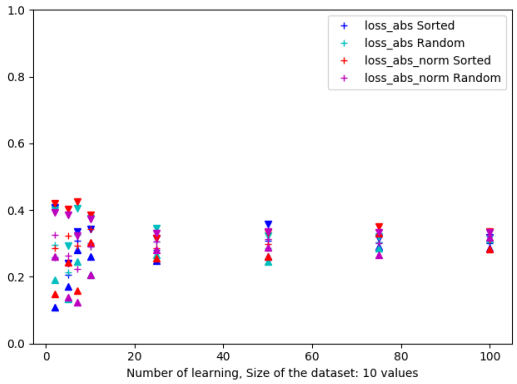

FIGURE 9 – écart type de l'apprentissage moyen en fonction du nombre d'apprensitssages

Ici, contrairement à la FIGURE 8, la moyenne ne varie pas significativement mais l'écart type diminue drastiquement. Ce paramètre combiné au précédent permettra d'augmenter la précision du réseau dans la partie 6.2.

6.2 Base de données réelle

Afin d'observer l'efficacité des différentes étapes de ce travail, à chaque étape, la courbe d'apprentissage ainsi que les résultats moyens (plus écart type) pour cent apprentissages ont été tracées.

6.2.1 Réseau simple

Les données ont tout d'abord été passées dans un réseau de neurones sans traitement spécifique. Voici le résultat de l'apprentissage :

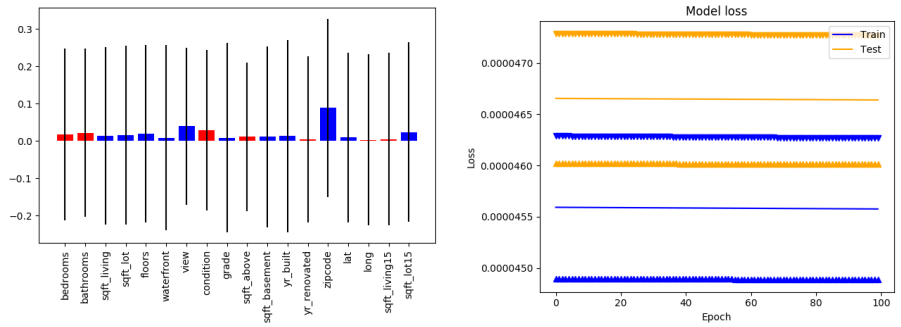

FIGURE 10 – Apprentissage sur les données réelles

L'apprentissage est ici totalement chaotique. Ce résultat était attendu étant donné que les fonctions 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 une formule de type $Surface \times Prix/m^3$.

6.2.2 Fonctions d'utilités

Les fonctions d'utilités ont donc été calcullées. Voici le résultat :

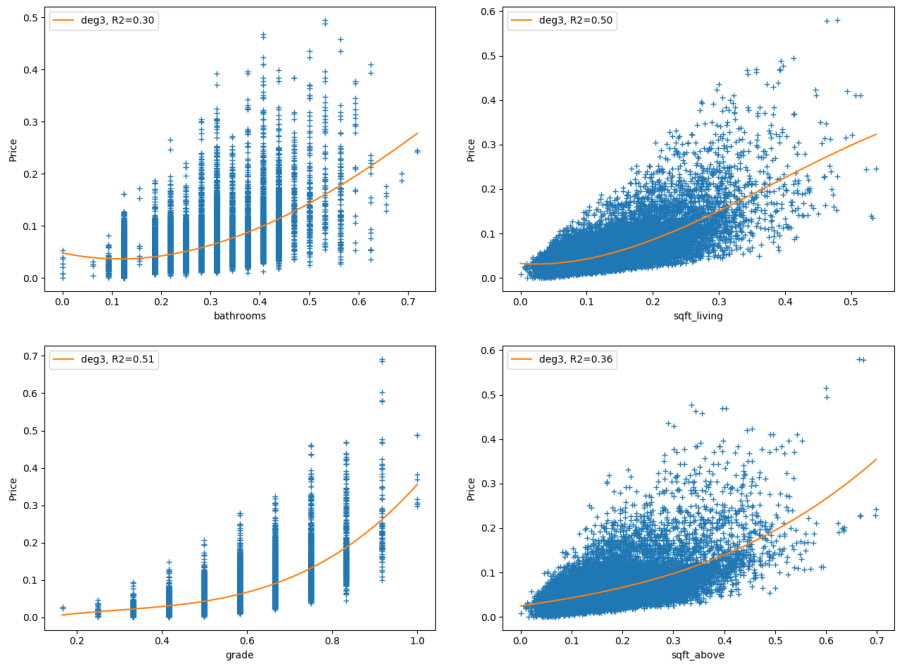

FIGURE 11 – Différentes fonctions apprises

Comme il peut être observé dans la FIGURE 11, seul quatre fonctions d'utilité ont pu être trouvées. Un réseau simple a donc été entrainé à partir de celles-ci. Le résultat est le suivant :

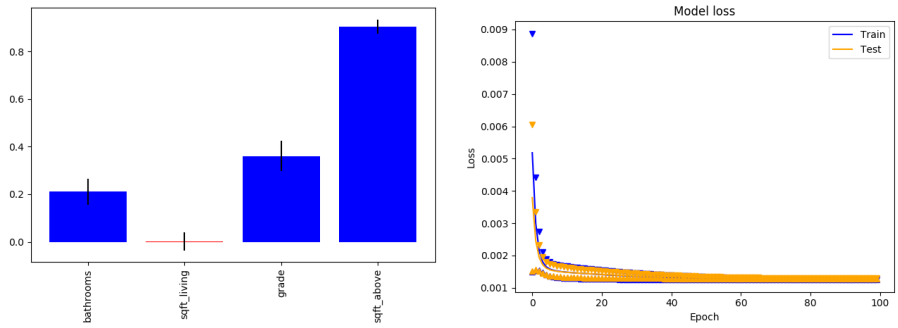

FIGURE 12 – Apprentissage sur les données réelles

Il peut être remarqué peut que l'apprentissage est bien plus précis. En effet, l'espace des poids étant bien plus restreint, le réseau explore cet espace bien plus rapidement.

6.2.3 Réseau de choquet

Les fonctions d'utilitées sont donc passés dans un réseau de Choquet afin d'en calculer tout ces coefficients :

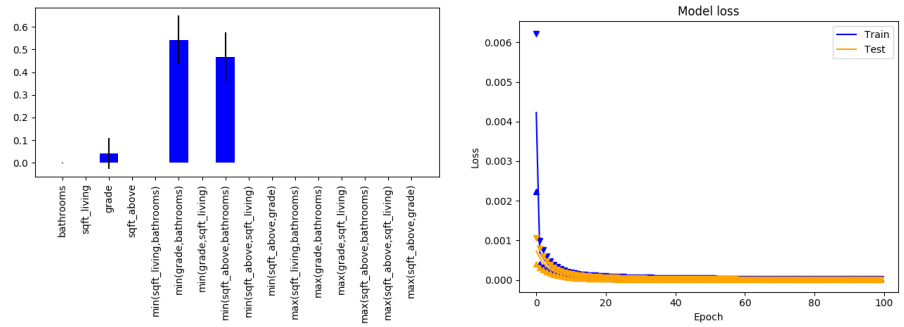

FIGURE 13 – Apprentissage avec le réseau de Choquet

En observant la FIGURE 13, il est visible que seul deux paramètres ont des valeurs significativement différentes de 0. Ces paramètres 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 beaucoup de salles de bains sont plus cheres dans cette base de données.

min(superficie, nombre de salles de bains) : La taille de la maison est donc prise en compte, mais ce n'est pas le facteur déterminant pour donner un prix à une maison.

On peut cependant observer que l'écart type sur l'erreur est plus important que dans la FIGURE 12. De plus, la fonction de perte atteint la valeur 0. Ces deux notions combinées sont caractéristiques d'overfitting.

7 Conclusion

Données artificielles (6.1) :

Durant cette étude, l'efficacité des réseaux de neurones a pu être observée. Il n'est plus à démontrer la puissance de généralisation ainsi que la résistance aux perturbations de cette technique. De plus, l'influence du nombre d'apprentissages ainsi que du nombre de données disponibles sur la fiabilité des résultats ont pu être étudié. Ces résultats démontrent qu'un réseau peut apprendre efficacement même si peu de données lui sont fournies.

Données réelles (6.2) :

Les résultats sur les données réelles ne sont pourtant pas à la hauteur de ce que le modèle théorique (4.1) et que les tests réels (6.1) laissait présager. En effet, sans traitement particulier, l'apprentissage est totalement chaotique. Les fonctions d'utilités permettent d'améliorer l'apprentissage mais on observe grâce au réseau de Choquet que les plages de données sélectionnées sont des facteurs de confusions impliquant des problèmes d'overfitting.

Dévelopement 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 implémenter mon premier réseau de neurones. J'avais déjà des connaissances théoriques mais je n'avais jamais eu l'occasion de manipuler ce genre d'objets.

L'utilisation scientifique de python a aussi été une part très importante de mon stage. Des problématiques qui n'avaient jamais été abordées en cours tel que l'optimisation se sont posées. Les échanges avec les autres stagiaires/thésards ont été très instructifs (découverte de nouvelles librairies, bonnes méthodes à adopter...). J'ai aussi eut l'opportunité d'utiliser un cluster de calcul du LRI.

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 développement au sein d'une entreprise.

Perspectives :

Il serait tout d'abord intéressant de savoir si ce modèle overfit. Pour cela il faudrait tester celui-ci sur une autre base de données. Les résultats permettraient de conclure sur l'efficacité de ce réseau.

Pour pouvoir tirer de plus amples résultats il serait bien de tester la variation de l'apprentissage en fonction du nombre d'entrées du réseau utiles et inutiles. Par exemple un réseau faisant un produit scalaire entre son vecteur poids et son vecteur entrée, il permettra de définir une approximation de la taille de la base de données nécessaire à approximer une fonction. L'efficacité d'un réseau faisant la somme de ses deux premières entrées en fonction de la profondeur de l'apprentissage peut aussi être utile pour tester la résistance du réseau aux facteurs de confusion.

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.