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

Quentin Lieumont

18 Juin 2019

Table des matières

Sous la direction de : Johanne Cohen
Martine Thomas

Chapitre 1

Introduction

1.1 Objectif du stage

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

1.2 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 35 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 recherche 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 a jour [1].

1.3 Unitée GALaC

Durant mon stage, j'ai travaillé sous la direction de Johanne Cohen qui est la responsable qe l'équipe GALaC.
L'équipe GALaC est une équipe du LRI qui travaillent sur des thématiques théoriques comme l'algorithmique, la théorie des graphes ou les systèmes en réseaux.

Chapitre 2

Contexte et motivations

2.1 Objectif

Le but du stage était de prédire le juste prix d'une maison à partir de quelques unes de ces caracteristiques. 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. Un réseau de neurones et les intégrales de choquets ont donc été utilisées.

2.2 Un petit point théorique...

Afin de mieux comprende le déroulé du stage et les interactions des differentes notions, un petit point théorique est nescessaire :

2.2.1 Réseaux de neurones

Qu'est ce que c'est ?

On entend souvent dans les médias parler 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, ce n'est pas ce que j'ai fait durant mon stage, on n'en parlera pas ici mais si vous voullez en savoir plus, je vous redirige vers la chaine youtube de Lê Nguyên Hoang, chercheur en mathématique qui saura vous l'expliquer bien mieux que moi (à regarder sans moderation) [2].

Un réseau de neurones est une architecture informatique 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.

Un neurone

Un neurone (représenté par un cercle dans les schemas) est une unité de base du réseau, il se decompose en trois phases

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.

Exemples :

- un neurone prenant les trois valleur de coulleur d'un pixel (entier entre 0 et 255).
- un neurone prenant les trois valleur de coulleur d'un pixel (réel $[0, 1]$).
- un neurone prenant 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 poid se note W et chacun de ses element $w_1 \dots w_i \dots w_n$.

Exemples :

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

La fonction d'activation : Comme vus precedement, ces réseaux 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.

Exemples :

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

Pour resumer : On prend *des entrées*, on les passe dans la *fonction principale* du neurone, puis le résultat dans la *fonction d'activation*.
Le résultat resemble a cela :

$$f_{act}(f(X)) \tag{2.1}$$

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

Exemples :

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.

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.

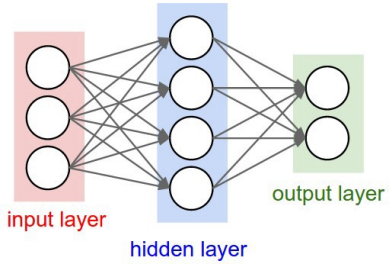

FIGURE 2.1 – Réseau dense simple

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.

L'apprentissage

Jusqu'ici on a vus comment creer une architecture modulable permetant d'aproximer toute les fonctions. 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.

Exemples :

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

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 : Pour minimiser loss, on aimerai bien descendre sa pente, cad dériver suivant les diférentes variables [1]. Étant donné que de manière générale, il y a plusieurs variables, la dérivée se transforme en gradient. Ainsi, afin de pouvoir calculer facilement les gradients, des applications linéaires sont privilégiées dans les fonctions des neurones.

On a donc expliqué d'ou vient la descente de gradient, mais que veut dire stochastique ? Le mot stochastique est synonyme de hasard. En effet le temps de calcul du gradient est exponentiel alors lorsque le réseau est grand (un réseau peut facillement atteindre le million de neurones voir même des milliards [3]) il est inimaginable de calculer la totalité du gradient, on utilise alors le gradient stochastique.

Cette descente de gradient est extrememement rapide, ainsi, il est possible de l'iterer de nombreuses fois pour essayer 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. Ainsi, avec plusieurs apprentissages on peut approximer très efficacement quasiment toutes les fonctions.

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

2.2.2 Intégrales de choquet

L'integrale de choquet est une intégrale découlant de la théorie de la mesure que je n'ai pas vus, de plus, 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'un oridnateur. Ces personnes de conaissent absolument rien en informatique. La seul chose qu'ils veullent c'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 chacuns des composant. Enfin on fait la somme de utilité fois poid pour tout les composant.
Ainsi, 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 deux a deux des utilitées par le biais des fonctions min et max.

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

$$C(X) = \sum_{i=1}^{n} w_i \times x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Big(w_{M\,ij} \times \max(x_i, x_j) + w_{m\,ij} \times \min(x_i, x_j) \Big) \quad (2.2)$$

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

Chapitre 3

Matériel et méthodes

Ce stage 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, la librairie Keras à été utilisée [4]. La partie sur les intégrales de choquet à été entierement recodée en python. L'ensembe du code est bien évidement disponible gratuitement et sous licence libre sur github [5].

3.1 Keras

La librairie Keras est une interface Python/TensorFlow permetant de travailler avec des réseaux de neurones. Cette librairie est très complète et permet de nombreuses choses. Ici, on ne s'attardera que sur les fonctionnalitées principales, a savoir la création d'un réseau simple, la regression par SGD et l'évaluation des performances.

Voici un exemple de réseau de neurone assez simple qui vas essayer de deviner l'application linéaire suivante : $f(X) = 0.2x_1 + 0.8x_2$

```
1  from keras import Sequential, optimizers, layers
2  import numpy as np
3  from random import random
4
5
6  def f(X):                    # la fonction que l'on cherche
7      W = np.array([0.2, 0.8])
8      return W @ X             # Produit scalaire
9
10 act = "linear"    # fonction d'activation
11 n_input = 2       # nombre de neurones
12 questions = np.array([np.array([random(), random()])
13                       for i in range(10000)])
14 reponses  = np.array([f(q) for q in questions])
15
16 sgd = optimizers.SGD(lr=0.01, decay=1e-6,
17         momentum=0.9, nesterov=True)
18 neurones = layers.Dense(1, activation=act,
19         input_dim=n_input, use_bias=False)
20 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, on obtient :

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

On peut donc bien voir que le réseau de neurones fonctionne et réussit à apprendre des fonctions avec plusieurs paramètres. Il est cependant assez embetant de toujours devoir faire appel a toutes ces fonctions. Une librairie à alors été codée afin de simplifier son utilisation. Elle pourra être appelée avec de nombreux paramtres qui seront abordés dans les parties suivantes.

On peut maintenant faire apprendre au réseau nimporte quelle application linéaire. Par exemple la moyenne :

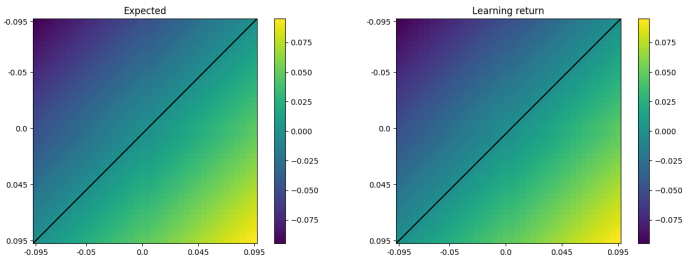

FIGURE 3.1 – Apprentissage de la moyenne

A gauche, le résutltat atendu, a droite, celui renvoyé.
x_1 en absice, x_2 en ordonées.

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 :

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

3.2 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 pre-

cedement (2.2.2), une fonction de choquet a une architecture complexe. Voici l'architecture d'un réseau de choquet entierement générée par un réseau de neurones :

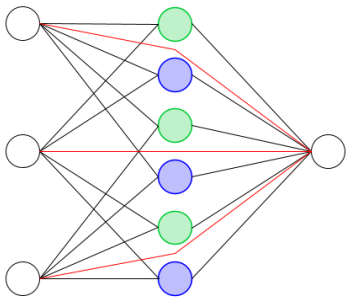

FIGURE 3.2 – Architecture d'un réseau de choquet

En bleu, des neurone appliquant $\min(X)$*, en vert* $\max(X)$.

On peut voir que trois problemes non triviaux se posent :
— Les neurones collorées n'appliquent pas une fonction simple.
— Les neurones ne sont pas reliés

Chapitre 4

Résultats

Chapitre 5

Discution et conclusion

Bibliographie

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

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

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

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

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