

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	3
2	Présentation générale du LRI	3
3	Équipe GALAC	3
4	Contexte et motivations	4
5	Un petit point théorique...	4
5.1	Réseaux de neurones	4
5.1.1	Un neurone	4
5.1.2	Un réseau	6
5.1.3	L'apprentissage	6
5.2	Intégrales de choquet	8
6	Matériel et méthodes	9
7	Keras	9
8	Implementation d'un réseau de choquet	10
9	Résultats	12
10	Discussion et conclusion	13
	Références	14

1 Introduction

Mon stage c'est déroulé au laboratoire de recherche en informatique, son sujet était le suivant : *Algorithmes de minimisation du regret et intégrales de choquet*. Je n'avais jamais fait d'informatique fondamentale (excepté IA) et n'ayant jamais vus 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.

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 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, bioinformatique, interaction homme-machine, etc). 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].

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 Supélec 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.

4 Contexte et motivations

Le but du stage était de manipuler les intégrales de choquet et les réseaux de neurones. La prédiction du prix des maisons a été étudiée grâce à une base de donnée trouvée sur internet [2]. On pourrait tout simplement multiplier le prix au mètre 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 comprendre les interactions entre les différentes notions, un petit point théorique est nécessaire :

- Compréhension du fonctionnement d'un réseau de neurones
- Fonctionnement des intégrales de choquet
- Pourquoi les intégrales de choquet associées aux réseaux de neurones sont particulièrement efficace pour résoudre certains problèmes

5.1 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 chaîne youtube de LÊ NGUYỄN HOANG. Anciennement chercheur en mathématique et aujourd'hui vulgarisateur sur internet et à l'EPFL ses vidéos sont à regarder sans modération [3].

Un réseau de neurones est une architecture informatique inventée en 1950 et remise 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.

5.1.1 Un neurone

Un neurone est une unité de base du réseau, il se décompose en trois phases.
image 1 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 pourrait imaginer un neurone ne prenant pas d'entrée, ou au contraire en prenant une infinité), ni de contrainte sur les différents objets.

Exemples :

Un neurone prenant :

- les trois valeurs de couleur d'un pixel (entier entre 0 et 255).
- les trois valeurs de couleur d'un pixel (réel $[0, 1]$).
- une séquence ADN en entrée

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

La fonction interne : Le réseau va alors faire un calcul à partir des entrées et de poids, des valeurs définies pour chaque neurone qui peuvent varier durant l'apprentissage.

Le vecteur poids se note W et chacun de ses éléments $w_1 \dots w_i \dots w_n$.

Exemples :

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

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

La fonction d'activation : Comme vu précédemment, 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 précédemment cité, un réel entre 0 et 1.

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

Exemples :

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

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

Pour résumer : Un neurone prend *des entrées*, les passe dans sa *fonction principale*, puis le résultat dans la *fonction d'activation*.

Formellement, on obtient la formule suivante :

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

Le tout dépendant bien évidemment du vecteur 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.

5.1.2 Un réseau

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

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

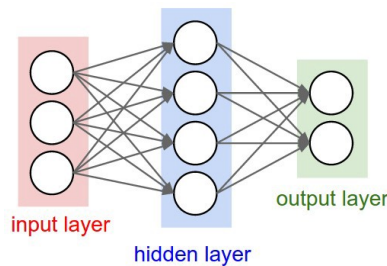


FIGURE 1 – Réseau dense simple

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

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

5.1.3 L'apprentissage

Jusqu'ici on a vu comment créer une architecture modulable permettant d'approximer toute les fonctions. Mais une question se pose toujours :

Quel est l'intérêt ? 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 connaît que certaines valeurs discrètes. On va commencer par définir une fonction nommée "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 :

$$\begin{aligned} loss(exp, obt) &= |exp - obt| \\ loss(exp, obt) &= (exp - obt)^2 \end{aligned}$$

Le problème de régression se résume alors à minimiser la fonction de perte, c'est ici que la descente de gradient stochastique fait son entrée : Pour minimiser $loss$, on aimerait bien descendre sa pente, c'est-à-dire dériver suivant les différentes variables¹. 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'où 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 facilement atteindre le million de neurones voir même des milliards [4]) il est inimaginable de calculer la totalité du gradient, on utilise alors le gradient stochastique.

Cette descente de gradient est extrêmement rapide, ainsi, il est possible de l'itérer de nombreuses fois pour essayer 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 un minimum local sera atteint. 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).

5.2 Intégrales de choquet

L'intégrale 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 modèle discret.

Pour l'expliquer simplement, prenons un exemple : Supposons que l'on veuille conseiller des personnes sur l'achat d'un ordinateur. Ces personnes ne connaissent absolument rien en informatique. La seule chose qu'ils veulent c'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...), on appelle cette note *l'utilité* du composant. Une fois toutes les utilités étudiées, on donne un poids à chacun des composants. Enfin on fait la somme de utilité fois poids pour tous les composants.

Ainsi, si un ordinateur a de meilleurs composants, plus de puissance, etc., sa note sera supérieure.

Ce modèle paraît raisonnable dans la plupart des cas mais il est extrêmement 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. 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 poids aux utilités, un poids est donné aux interactions deux à deux des utilités par le biais des fonctions min et max.

Voici donc la fonction que l'on va vouloir approximer :

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

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

6 Matériel et méthodes

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

Afin d'implémenter informatiquement les concepts théoriques vus précédemment, la librairie Keras a été utilisée [5]. La partie sur les intégrales de Choquet a été entièrement recodée en python. L'ensemble du code est bien évidemment disponible gratuitement et sous licence libre sur github [6].

7 Keras

La librairie Keras est une interface Python/TensorFlow permettant 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 fonctionnalités principales, à savoir la création d'un réseau simple, la régression par SGD et l'évaluation des performances.

Voici un exemple de réseau de neurone assez simple qui va 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):
7     W = np.array([0.2, 0.8])
8     return W @ X
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
21 model.add(neurones) # On lui ajoute des neurones
22 model.compile(optimizer=sgd, # On compile le tout
23              loss="mean_squared_error")
24 model.fit(questions, reponses) # On essaye de coller aux donn es
25
26 for i, weight in enumerate(model.get_weights()[0]):
27     # on affiche les poids
28     print(f"w{i} : {weight[0]}")
```

code/reseau1.py

En executant ce code, on obtient :

```
1 w0 : 0.19988934695720673
2 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 embêtant de toujours devoir faire appel à toutes ces fonctions. Une librairie a alors été codée afin de simplifier son utilisation. Elle pourra être appelée avec de nombreux paramètres qui seront abordés dans les parties suivantes.

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

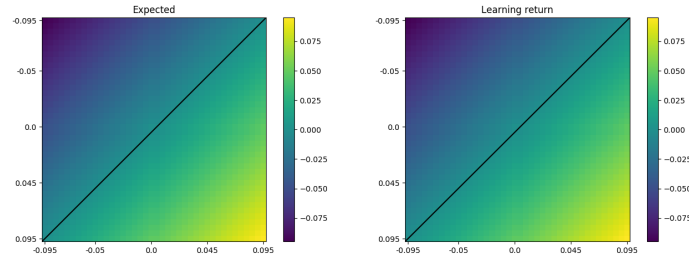


FIGURE 2 – Apprentissage de la moyenne

*A gauche, le résultat attendu, à droite, celui renvoyé.
 x_1 en abscisse, x_2 en ordonnées.*

On peut voir que la descente de gradients se fait à merveille. Aucune différence ne sont visibles : en effet les poids associés aux deux nœuds sont les suivants :

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

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

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

8 Implementation d'un réseau de choquet

Nous appellerons ici "réseau de choquet" un réseau de neurones ayant une architecture adaptée à la régression d'une intégrale de choquet. Comme vu précédemment (5.2), une fonction de choquet a une architecture complexe. Voici l'architecture d'un réseau de choquet entièrement générée par un réseau de neurones :

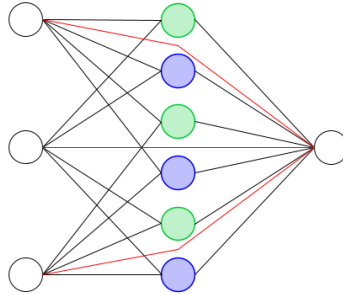


FIGURE 3 – Architecture d'un réseau de choquet

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

On peut voir que trois problèmes non triviaux se posent :

- Les neurones collorées n'appliquent pas une fonction simple.
- Les neurones ne sont pas reliés

9 Résultats

10 Discussion et conclusion

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] “KERAS : The python deep learning library.” <https://keras.io/>.
- [6] “GITHUB.” github.com/QuentinN42/Stage_LRI.