UNIVERSITÉ PARIS DAUPHINE

MASTER 1 MIAGE

Projet d'Intelligence Artificielle

Réalisé par :
Lyes MEGHARA
Nadia MSELLEK

Encadré par :
Mr Stephane AIRIAU

18 Décembre 2017

I Introduction

La notion de réseaux de neurones artificiels a été évoquée depuis plus de cinquante ans. Cela a permis l'évolution concrète de l'intelligence artificielle au plus grand bonheur des informaticiens et étudiants. Le projet que nous allons exposer traite de la capacité d'un algorithme d'apprentissage supervisé suivant le modèle du perceptron.

Nous disposons pour cela d'une base de données créée par Yann LECUN nommée « *Mnist handwritten database* » composée de plusieurs milliers d'exemples d'images de 28*28 pixels chacune représentant un chiffre manuscrit.
Notre objectif est de récupérer parmi cette base de données un fichier input contenant 60 000 exemples sur lesquels notre algorithme va « apprendre » et ainsi pouvoir prédire les chiffres contenus sur chacun des 10 000 exemples donnés pour le test.

II Outils de développement

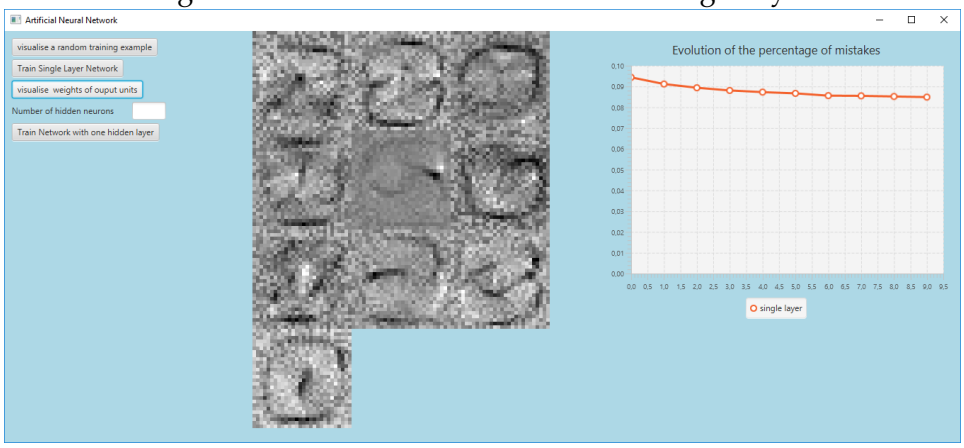
Dans le cadre de ce projet, nous avons utilisé GitHub pour une meilleure collaboration, eclipse sous la JDK 1.8, Doxygène nous a permis de générer une documentation visible depuis le fichier index.html.
Pour finir ce rapport a été rédigé en utilisant LateX.

III Résultats des tests

Avec un SingleLayer, l'exécution pour 28*28 neurones d'entrées ainsi que 10 neurones de sorties dure 5 minutes, cette dernière corrige lentement le taux d'erreurs et avoisine les 8% à la fin de l'exécution.

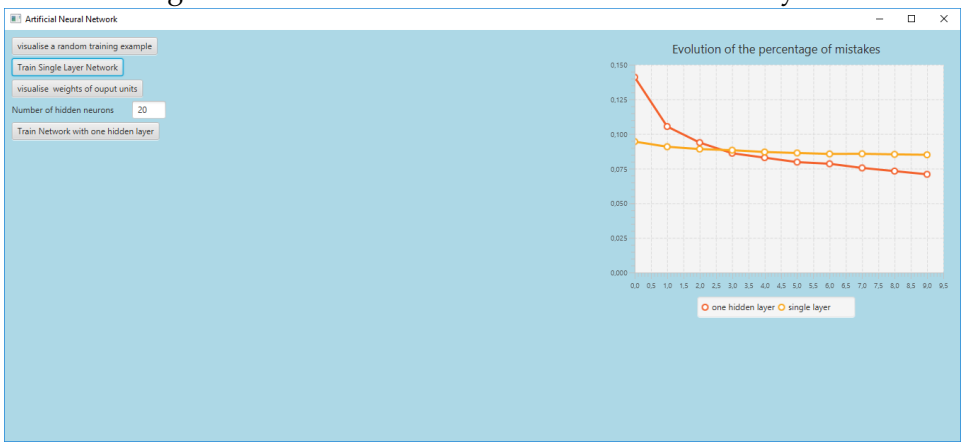
Un HiddenLayer, cette fois-ci, avec 10 neurones cachés donne à priori les mêmes résultats pour le même temps d'exécutions, en revanche, en passant à 20 neurones cachés, les courbes se croisent, celle du réseau caché passe en dessous comme visible à la figure 2, augmenter le nombre des neurones cachés améliore l'apprentissage, mais ralentit fortement l'exécution.

Figure 1: Résultat d'entrainement d'un SingleLayer



The training of your single layer lasted for 310.314633146 seconds

Figure 2: Résultat d'entrainement d'un HiddenLayer



The training of your hidden layer lasted for 662.522861786 seconds

IV Conclusion sur les résultats

Comme nous aurions pu le prédire, les résultats de HiddenLayer nous fournissent beaucoup moins d'erreurs que SingleLayer. Nous pouvions en effet conjecturer que, puisque HiddenLayer pouvait avoir beaucoup plus de neurones que SingleLayer, il aurait plus de *synapses*; si l'on veut faire

l'analogie avec le cerveau humain; et donc plus de variables à modifier, donc un apprentissage plus efficace.
Néanmoins, il est important de souligner que le SingeLayer lors de ses débuts, apprend bien mieux que le HiddenLayer, sur la durée, HiddenLayer le rattrape et le dépasse .
Ici, une seule couche cachée permet déjà d'atteindre un degré d'erreur assez faible. On suppose, naturellement, que de meilleurs résultats peuvent être atteints avec plusieurs couches cachées.

D'autre part, augmenter le nombre d'itérations à 20, ne permet pas d'avoir une amélioration significative, en effet au bout de la 15eme itération, notre algorithme n'apprend plus, et reste à un nombre d'erreurs quasi constant. En revanche, augmenter le nombre de neurones cachés à 80, améliore significativement les résultats et aboutit à 5% d'erreurs au détriment d'un temps d'exécution très lent.

V Ressenti de l'implémentation

Il est important de souligner que mis à part le cours, avant de travailler sur ce projet nous n'avions pas d'expérience avec les réseaux de neurones artificiels.
En outre, la première difficulté a été de comprendre le code déjà existant, les différentes classes et le lien entre ces dernières, comme les méthodes s'appellent et sont liées entre elles, trouver la source de nos erreurs n'a pas toujours été une tâche aisée.
Une des principales difficultés relevées a été l'implémentation de la méthode train pour la classe SingleLayer, qui affichait une courbe constante avant correction de notre part. Le passage de SingleLayer à HiddenLayer a été plus fluide de sens et de logique.

VI Améliorations éventuelles / Hypothèses

Le programme étant assez lent à tourner, une des premières idées que nous avons eues, était d'accroitre sa vitesse, en passant par des Threads, solution qui n'a pas été implémentée par manque de temps.
D'autres points, sur le côté théorique cette fois-ci, l'utilisation de biais n'a pas été mentionnée de façon explicite dans ce cours.
Par ailleurs, l'algorithme d'apprentissage *Widrow-Hoff* a été préféré à *l'algorithme de gradient*, étant à la fois plus efficace, et plus simple à implémenter.

VII Webographie

- https://fr.wikipedia.org/wiki/Perceptron
- https://fr.wikipedia.org/wiki/Perceptron_multicouche
- alp.developpez.com/tutoriels/intelligence-artificielle/reseaux-de-neurones/
- https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-
- https://www.cs.cmu.edu/afs/cs/academic/class/15883-f15/slides/backprop.pdf
- http://www.dgp.toronto.edu/people/RMB/papers_old/p6.pdf
- https://www.labri.fr/perso/nrougier/downloads/Perceptron.pdf
- http://www.turingfinance.com/misconceptions-about-neural-networks/
- https://youtu.be/L7WNYvbvGBc

VIII Annexes