

# DESSINE-MOI UN MOUTON

## COMPTE-RENDU DE PROJET LONG

Jamesisvbhjsbjdshbdshjvsl BARRIOS, Antoine CORDELLE, Benjamin  
MOUSCADET, Emmanuel TRAN

encadrés par Arpad RIMMEL et Joanna TOMASIK

2017 – 2018



CentraleSupélec

# Table des matières

<b>1</b>	<b>Réseaux de neurones</b>	<b>3</b>
1.1	Problème et de classification . . . . .	3
1.2	Apprentissage supervisé . . . . .	3
1.3	Structure d'un neurone . . . . .	3
1.4	Réseaux de neurones . . . . .	4
1.5	Propagation dans un perceptron . . . . .	4
<b>2</b>	<b>Implémentation des premiers réseaux</b>	<b>6</b>
2.1	Implémentation du perceptron . . . . .	6
2.1.1	Présentation et diagramme UML . . . . .	6
2.1.2	NeuralNetwork . . . . .	7
2.1.3	DataCollector . . . . .	7
2.2	Etude du XOR . . . . .	7
2.2.1	Définition du problème . . . . .	7
2.2.2	Résolution . . . . .	8
<b>3</b>	<b>Reconnaissance des chiffres manuscrits</b>	<b>9</b>
3.1	La base de données MNIST . . . . .	9
3.1.1	Présentation de MNIST . . . . .	9
3.1.2	Extraction de MNIST . . . . .	9
3.2	Apprentissage de la base de données . . . . .	10
3.2.1	Paramétrage du réseau de neurones . . . . .	10
3.2.2	Résultat . . . . .	10

# Introduction

Ce compte-rendu présente le Projet Long Supélec *Dessine-moi un mouton*, mené par 8 élèves de Supélec - séparés en deux équipes de 4 - et encadré par deux enseignants-chercheurs, Monsieur Arpad Rimmel et Madame Joanna Tomasik. Plus particulièrement, c'est le travail de l'équipe **Couleuvre** qui est rapporté ici.

L'objectif de ce projet était d'étudier les réseaux de neurones et de les utiliser pour générer des images de synthèses. L'étude des réseaux de neurones comportait une dimension de recherche afin d'appréhender la théorie de ces objets, et une dimension de développement et de programmation dans l'implémentation de ces réseaux et leur utilisation. Le projet s'est donc naturellement décomposé en plusieurs étapes.

Dans un premier temps, il a fallu étudier le fonctionnement des réseaux de neurones, et établir les équations régissant le comportement des réseaux de type perceptron. Puis nous avons implémenté un perceptron afin de résoudre un problème de décision simple au début (**XOR**), puis plus complexe (application à la base de données de chiffres manuscrits MNIST).

Dans un second temps, nous avons commencé à utiliser la structure qui représentait le coeur du projet : les réseaux semi-supervisés en compétition mutuelle, selon la méthode du **Generative Adversarial Network**. Nous l'avons d'abord appliqué à la génération d'images synthétiques de chiffres, puis nous l'avons utilisé pour faire de la synthèses d'images plus complexes, comme des images de moutons.

Afin de mener à bien ce projet long, il a été nécessaire de mettre en place plusieurs outils de gestion de projet. Nous avons créé une organisation **Github** (<https://github.com/supelec-GAN>) afin de faciliter le partage du code; nous avons utilisé **Trello** pour s'organiser en tâches et optimiser le travail; une bibliographie commune a été mise en place grâce à **Zotero**; la documentation du projet a été faite avec **Doxygen**; enfin, une réunion hebdomadaire réunissant les deux équipes et les enseignants a été imposée, permettant de rendre compte des avancées, de recevoir les directives sur le travail à effectuer, et de manger du chocolat.

Enfin, nous avons choisi le langage C++ pour ce projet. En effet, la structure de réseaux de neurones et de computational graphs (qui seront détaillées plus loin) s'accordent naturellement avec la représentation objet; de plus les réseaux de neurones nécessitent par définition de mener des calculs en grandes quantités. La rapidité de calcul du C++ (héritée du C) ainsi que son approche objet robuste et particulièrement puissante en faisaient le candidat idéal. De plus, un seul membre du groupe n'était pas familier avec le C++ et a dû apprendre en cours de route. **Nous déconseillons vivement aux futurs membres de ce projet long d'utiliser un langage qui ne soit pas maîtrisé par au moins la moitié de l'équipe.**

# Chapitre 1

## Réseaux de neurones

### 1.1 Problème et de classification

Dans la théorie de l'apprentissage statistique, la classification a pour objectif de déduire d'un nombre fini d'observations indépendantes une partition de l'espace en un ensemble a priori inconnu de domaines de l'espace appelés classes.

### 1.2 Apprentissage supervisé

### 1.3 Structure d'un neurone

L'extraordinaire capacité d'apprentissage et d'adaptation des réseaux de neurones biologiques a poussé les scientifiques à tenter de modéliser informatiquement leur fonctionnement afin d'exploiter ces capacités.

Les observations biologiques ont mené au premier modèle du neurone, divisé en 3 parties. Le neurone reçoit des signaux chimio-électriques par ses dendrites; ces signaux sont traités dans le corps cellulaire, où un effet de seuil est appliqué; enfin l'axone permet la transmission du signal chimio-électrique de sortie. De nouvelles découvertes sur les neurones ont permis de complexifier grandement ce modèle, mais c'est celui qui sert de base aux neurones artificiels.

**Définition 1** (Neurone). Un neurone à  $n$  entrées permet de modéliser une fonction de  $\mathbb{R}^n$  dans  $\mathbb{R}$ . Un neurone est défini par la données de trois paramètres :

- un vecteur de poids  $\omega \in \mathbb{R}^n$
- un biais  $b \in \mathbb{R}$
- une fonction d'activation  $g \in \mathbb{R}^{\mathbb{R}}$

La fonction  $f$  modélisée par le neurone s'écrit alors :

$$\forall x \in \mathbb{R}^n, f(x) = g(\omega^T x - b) = g\left(\sum_{i=0}^{n-1} \omega_i x_i - b\right) \quad (1.1)$$

*Remarque 1* (Biais). Il est possible de rajouter une composante  $x_n = -1$  à tous les vecteurs d'entrées afin de pouvoir considérer le biais  $b$  comme la composante  $\omega_n$  du vecteur de poids. L'équation (1.1) devient alors

$$\forall x \in \mathbb{R}^n, f(x) = g(\omega^T x) = g\left(\sum_{i=0}^n \omega_i x_i\right) \quad (1.2)$$

Cependant, considérer le biais comme un poids pose des problèmes lorsqu'on utilise l'approche des graphes de calculs, qui sera explicités dans la suite de ce rapport. Nous n'avons donc pas appliqué cette simplification pour nos propres réseaux.

## 1.4 Réseaux de neurones

Les possibilités limitées d'un neurone seul ont donc poussé l'informatique à agencer les neurones en réseaux afin de créer des modèles plus complexes.

**Définition 2** (Réseau de neurone - Perceptrons). Un réseau de neurones est défini par un graphe orienté  $\mathcal{G}(V, A)$  où les nœuds sont des neurones et les arêtes des liens entre les neurones et par un ensemble de neurones d'entrée  $V_{in} \subset V$  et de neurones de sorties  $V_{out} \subset V$ . Une arête partant d'un neurone  $i$  vers un neurone  $j$  signifie que la sortie du neurone  $i$  est une entrée pour le neurone  $j$ . Notons  $f_j$  la fonction représentant le neurone  $j$ . Finalement, nous noterons un réseau  $\mathcal{N}(V, A, V_{in}, V_{out}, f)$  où  $f$  est l'ensemble des fonctions des neurones.

**Définition 3** (Perceptron). Un réseau de neurones est de type perceptron lorsque le graphe qui le représente est acyclique (réseau feedforward) et qu'il peut être représenté comme une suite finie de couches de neurones telle que les sorties d'une couche sont exactement les entrées de la couche suivante.

## 1.5 Propagation dans un perceptron

On cherche à obtenir les équations qui régissent la propagation des entrées dans un réseau perceptron. Comme les neurones sont agencées par couches, on peut indiquer chaque neurone par son numéro de couche et sa position dans la couche. De plus, on parlera dans la suite indistinctement du neurone ou de la fonction qu'il modélise.

Alors, soit un réseau perceptron à  $N$  couches. Pour  $i \in \llbracket 1, N \rrbracket$  soit  $n_i$  le nombre de neurones de la couche  $i$ . Pour tout  $i, j$  tel que  $i \in \llbracket 1, N \rrbracket, j \in \llbracket 1, n_i \rrbracket$ , le neurone  $(i, j)$  est défini par

- sa fonction d'activation  $f_{i,j}$
- son vecteur de poids  $\omega_{i,j}$
- son biais  $b_{i,j}$

Comme on est dans un réseau perceptron, tous les neurones de la couche  $i - 1$  sont connectés au neurone  $(i, j)$ . Par conséquent le vecteur d'entrée de ce neurone est un vecteur  $X_n$  de dimension  $n_{i-1} \times 1$ . Alors la sortie du neurone  $(i, j)$  est égale à  $f_{i,j}(\omega_{i,j} X_{i-1}^T - b_{i,j})$ . En écrivant  $X_n = \begin{pmatrix} X_0^n & \dots & X_{n_i-1}^n \end{pmatrix}^T$ , on a alors la relation de récurrence

$$X_i^{j+1} = f_{j,i}(\omega_{j,i} X_j^T - b_{j,i}) \quad (1.3)$$

Dans un réseau perceptron, on suppose de plus que toutes les fonctions d'activations des neurones d'une même couche sont identiques. On a alors  $\forall i \in \llbracket 1, N \rrbracket, \forall j \in \llbracket 1, n_i \rrbracket, f_{i,j} = f_i$ . On peut alors réécrire l'équation (1.3) sous forme matricielle. On pose

$\forall i \in \llbracket 1, N \rrbracket, F_i = \begin{pmatrix} f_i & \dots & f_i \end{pmatrix}$  (Fonction d'activation de la couche)

$W_i = \begin{pmatrix} \omega_{i,0} & | & \dots & | & \omega_{i,n_i-1} \end{pmatrix}^T$  (Matrice des poids de la couche)

$B_i = \begin{pmatrix} b_{i,n_i-1} & \dots & b_{i,n_i-1} \end{pmatrix}^T$

En nommant  $X_0$  le vecteur en entrée du perceptron, on a la nouvelle équation matricielle :

$$\forall i \in \llbracket 1, N \rrbracket, X_i = F_i(W_i X_{i-1} - B_i) \quad (1.4)$$

## Chapitre 2

# Implémentation des premiers réseaux

Ce chapitre décrit l'implémentation de nos réseaux de neurones sous forme de perceptrons, que nous utiliserons par la suite dans les différentes applications. Nous verrons ici cette implémentation et le premier exemple avec le XOR.

### 2.1 Implémentation du perceptron

#### 2.1.1 Présentation et diagramme UML

Comme décrit dans la partie théorique sur les perceptrons (1.5), nous avons choisi de représenter nos réseaux de neurones sous forme matricielle. Les neurones ne seront donc pas représentés individuellement, mais par couche de neurones (`neuronLayer`). Cela permet d'effectuer les calculs sous forme matricielle par rapport à des propagations et rétro-propagations neurone par neurone, ce qui fait gagner beaucoup de temps de calcul. Cela est possible en gardant les mêmes fonctions d'activation pour chaque neurone d'une même couche, ainsi que les mêmes entrées et sorties. Dans le cadre du perceptron, cela convient.

Le diagramme UML comporte une classe principale, `Application`, qui va gérer les expériences et réseaux de neurones. Cette classe contient un réseau de neurones (`NeuralNetwork`) composé de plusieurs couches de neurones (`NeuronLayer`). `Application` possède aussi un collecteur qui est utilisé pour récupérer les données des expériences, les traiter et les exporter dans un fichier `.csv`. Enfin, la classe `Teacher` a pour rôle de gérer la rétropropagation et l'apprentissage du réseau de neurones.

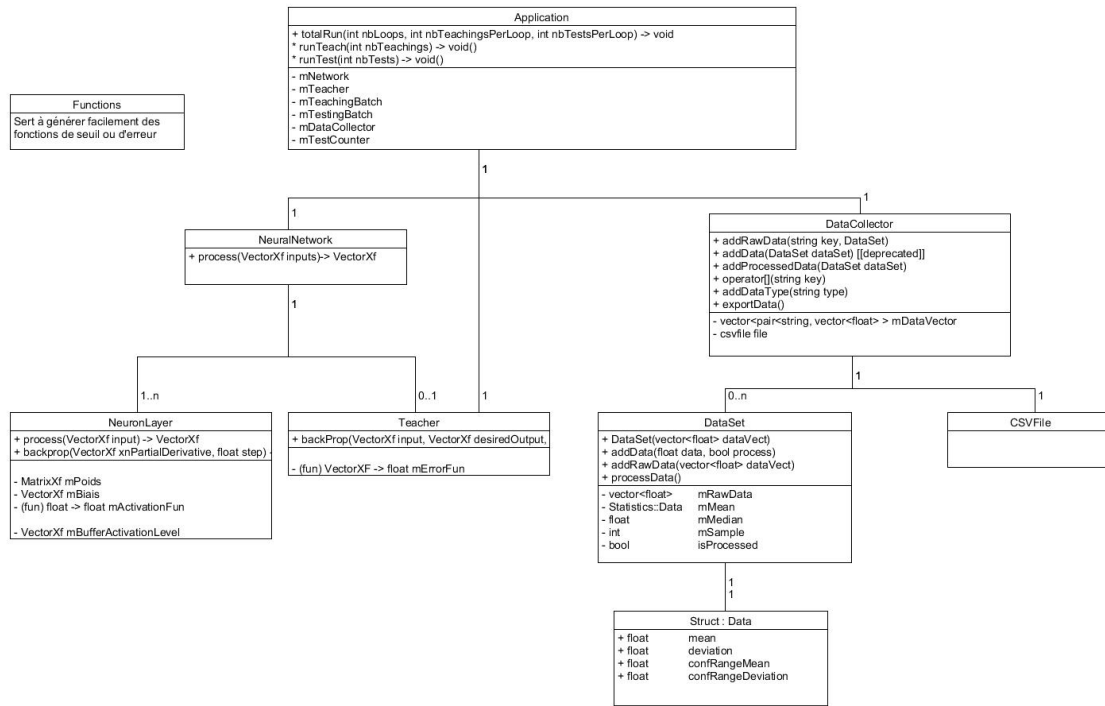


FIG. 2.1 – *Diagramme UML*

### 2.1.2 NeuralNetwork

### 2.1.3 DataCollector

## 2.2 Etude du XOR

Le problème du XOR est un problème de classification standard, auquel toutes les méthodes de classification sont confrontées. Nous avons donc naturellement commencé par le traitement de ce problème pour nous exercer à la manipulation des perceptrons.

### 2.2.1 Definition du problème

Le problème du XOR est un problème en deux dimension : les entrées sont des paires  $(x_1, x_2)$  et la sortie est un booléen. Deux approches différentes existent, selon le choix du domaine de définition :

- 1<sup>er</sup> choix : on veut obtenir un XOR purement booléen, dont les 4 entrées possibles sont les paires  $(0,0)$ ,  $(0,1)$ ,  $(1,0)$ ,  $(1,1)$ . Les résultats attendus sont respectivement 0, 1, 1, 0.
- 2<sup>nd</sup> choix : on veut obtenir un XOR défini sur le carré unité du plan :  $(x,y) \in [-1,1]^2$ . Dans ce cas, on attend du XOR qu'il renvoie 1 lorsque  $x \times y < 0$  et 0 sinon. On fait donc un XOR sur le signe des coordonnées.

Selon l'approche, le réseau requis sera différent. Dans le premier cas, il est possible de classifier les 4 points selon leur appartenance à un ensemble convexe (qui prend la



forme d'une bande). Par conséquent, le problème peut se résoudre avec deux couches de neurones.

Dans le second cas, il faut séparer le plan en 4 quadrants, et regrouper les quadrants par deux. On obtient des classes non convexes, et le problème requiert 3 couches de neurones pour être résolu.

Nous avons choisi de travailler sur la deuxième situation, le but est donc d'obtenir un réseau qui découpe le carré unité en quatre quadrants bien distincts, qui correspondent aux quatre quadrants délimités par les axes des abscisses et des ordonnées.

### 2.2.2 Résolution

Il s'agit d'un problème à deux dimensions, renvoyant une seule valeur, et nécessitant 3 couches cachées. Nous optons donc pour un réseau 2-2-1.

Qualitativement, on peut dire que

- les deux neurones de la première couche vont se charger de tracer les deux droites correspondant aux abscisses et aux ordonnées
- les deux neurones de la seconde couche vont se charger de réaliser les fonctions  $\sigma$  sur les quadrants ainsi tracés, afin de d'obtenir deux quadrants correspondants au résultat 1 et deux quadrants correspondants au résultat 0.
- la troisième couche se charge de réaliser la fonction  $\sigma$  sur les résultats renvoyés par la couche 2, afin que le résultat final soit 1 si le point se trouve dans l'un des deux quadrants correspondant au résultat 1.

De plus, nous avons choisi d'utiliser des fonctions d'activations sigmoïdes en  $\frac{1}{1+\exp(-\lambda x)}$ . Ce choix empêche d'avoir un résultat booléen à la fin, puisque la valeur de sortie varie continuellement entre 0 et 1. Le résultat est donc une densité de probabilité.

Enfin, nous utilisons un pas d'apprentissage de  $\eta \leq 0.01$ . Bien que la littérature suggère souvent le pas empirique  $\eta = 0.2$ , la valeur de 0.01 est la valeur maximale que nous ayons trouvée qui permette au XOR de converger. Au delà, le réseau se bloque presque systématiquement dans un état non désiré.

Avec une dizaine de milliers d'apprentissages, on obtient le résultat suivant :

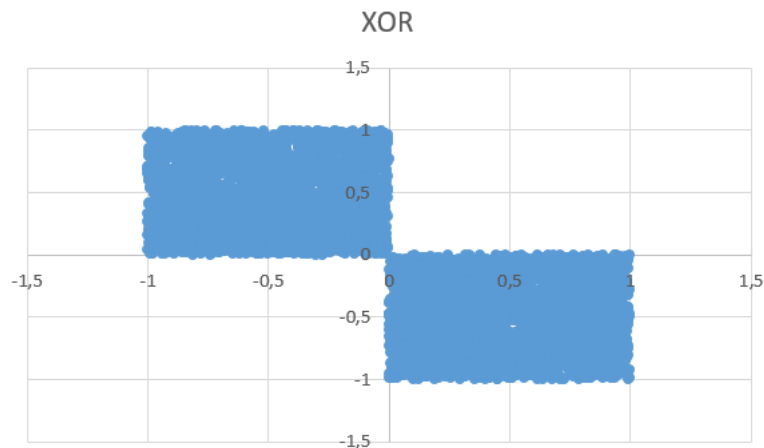


FIG. 2.2 – Résultat d'un XOR sur 1000 tests après 10000 apprentissages pour un réseau 2-2-1

## Chapitre 3

# Reconnaissance des chiffres manuscrits

### 3.1 La base de données MNIST

#### 3.1.1 Présentation de MNIST

La base de données MNIST (ou Mixed National Institute of Standards and Technology) est un regroupement de chiffres 70 000 écrits à la main. 60 000 d'entre eux servent à l'apprentissage et les 10 000 autres permettent de tester le réseau de neurones après l'apprentissage. Ce sont des images normalisées en noir et blanc, de 28 pixels de côtés chacun codé sur un octet. Nous nous baserons sur cette base de données pour faire apprendre à notre réseau de neurones la reconnaissance de chiffres.

#### 3.1.2 Extraction de MNIST

Les données sont dans le format idx1 et idx3 tel que :

offset	type	valeur	description
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of items
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
...	...	...	...
xxxx	unsigned byte	??	pixel

On peut distinguer :

- Le *magic number* qui permet d'identifier le format de la base de données
- Les *number of items*, *number of rows*, *number of columns* donne des informations sur les données, ce qui nous permettra d'extraire les images
- Les *pixels* qui sont les octets en niveau de gris des images

L'extraction se fait donc en lisant successivement les octets et en les stockant dans des *vector<Eigen:Matrix>* grâce aux données récupérées. En faisant de même avec les labels dont le format est très sensiblement le même, on obtient une liste de vecteur colonne

d'*Eigen::Matrix* de taille 784 (28\*28) ainsi que leur label sur un autre *Eigen::Matrix*. On fait de même avec les échantillons de test et on est fin prêt pour l'apprentissage.

## **3.2 Apprentissage de la base de données**

### **3.2.1 Paramétrage du réseau de neurones**

Bien que le format de l'entrée est le même (vecteur colonne si ce n'est la taille qui change), un réseau à 3 ou 4 neurones ne suffit pas. Il a fallu adapter de façon empirique les différents paramètres. On peut jouer sur :

- Le nombre de couches de neurones
- Le nombre de couches
- Le pas d'apprentissage
- Les fonctions d'activations par couches
- Le nombre d'apprentissages

**Le nombre de couches et de neurones**

**Le pas d'apprentissage**

**Les fonctions d'activations par couches**

**Le nombre d'apprentissages**

### **3.2.2 Résultat**