République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saâd Dahlab Blida

Faculté des sciences

Département Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en informatique

Option : Ingénierie de logiciel

Thème :
Deep Learning pour l'extraction des itemsets Fréquents

Réalisé par :

Bouzourine Abdelhak

Aidi Faiçal Taha

Encadré par :

Mme. F.Z. Zahra

Mr. A.H. Kameche

Devant le jury :

Président : FERFERA

Examinateur : L OUAHRANI

Soutenu le : 27 Juin 2018

Promotion: 2017/2018

Remerciements

Avant toute chose, Nous remercierons ALLAH le tout puissant, de nous avoir donnée la force et la patience pour mener à terminer ce travail.

Nous présentons nos sincères remerciements à nos encadrants Mme ZAHRA et Mr KAMECHE pour avoir assuré le suivi de ce projet mais aussi pour leur soutien, leurs conseils et leur disponibilité.

Nous tenons à remercier aussi tous nos enseignants qui ont participé à notre formation tout au long de notre cursus.

Nous tenons à remercier nos parents, nos familles et amis pour leur soutien tout au long de ce mémoire.

Nous remercions également toutes ces personnes sur internet qui ont partagé leurs connaissances et expériences et répondu à nos questions, ils nous ont été d'une grande aide.

Enfin, nous remercions tous ceux qui ont aidé de près ou de loin à l'élaboration de ce mémoire et à la réussite de ce projet.

Merci.

Résumé

L'extraction des motifs fréquents est une tâche essentielle et importante dans le domaine de la fouille de données. De nombreux algorithmes ont été implémentés pour trouver ces motifs dans le but de rechercher les combinaisons fréquentes d'items. Toutefois, la plupart des algorithmes d'extraction des items fréquents voient leurs performances et leurs efficacités diminuées lorsque la taille des données augmente.

Pour maintenir les performances de ces algorithmes, des méthodes et outils parallèles et distribués semblent être la démarche la plus communément utilisé. Notre travail consiste d'introduire une nouvelle approche pour l'extraction des items fréquents en se basant sur une nouvelle technologie appelé Deep Learning ou apprentissage profond. Cette technique a connu un grand succès dernièrement dans plusieurs domaines.

Mots clés : Extraction des Motifs Fréquents, Apprentissage Profond, Itemsets Fréquents, réseaux de neurones profonds.

Abstract

Frequent pattern mining is an essential and important task in the field of data mining. Many algorithms have been implemented to find these patterns in order to search for frequent combinations of items. However, the performance and the efficiency of frequent pattern mining algorithms decrease sinfully as the data size increases.

To maintain the performance of these algorithms, parallel and distributed methods seem to be the most commonly used approach. Our work deals with the introduction of a new approach for extracting frequent patterns based on new trend technology called Deep Learning. This technology has a lot of success in many applications.

Keywords: frequent pattern mining, deep learning, frequent itemsets extraction, deep neuron networks.

SOMMAIRE

Abréviations et sigles

ECD	Extraction de Connaissances à partir de Données
KDD	Knowledge Discovery from Databases
FPM	Frequent Pattern Mining
ARM	Association Rule Mining
MinSup	Minimum Support
FP-growth	Frequent Pattern growth
FP-Tree	Frequent Pattern tree
ML	Machine Learning
AI	Artificial Intelligence
RL	Reinforcement Learning
NLP	Natural language Processing
NN	Neural network
ANN	Artificial Neural network
DNN	Deep Neural network
GPU	Graphic Processing Unit
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory Networks
GRU	Gated Recurrent Unit
GAN	Generative Adversarial Networks
VAE	Variational Autoencoder
FFN	Feed Forward Neural Network
ONN	Optical Neural Network
SOM	Self Organizing Map

Liste de figures

INTRODUCTION GÉNÉRALE

1. Contexte

Les données prennent le pas sur la fonctionnalité. Elles sont au cœur de la réflexion, et souvent l'obsession des plus grosses entreprises en ce moment pour sa grande valeur.

L'extraction des motifs fréquents à partir de données est l'un des champs de recherche qui sont en plein d'expansion puisqu'elle constitue l'étape cœur de plusieurs méthodes de fouille de données à l'instar de l'extraction des règles d'associations, la construction de certains classifieurs, l'extraction de séries temporelles et d'autres techniques de fouille de motifs.

Le Deep Learning quant à lui est un domaine de l'intelligence artificiel qui a littéralement exploser c'est dernière année. C'est un domaine d'expertise assez vaste dont l'application est exploitée dans plusieurs industries.

Plus précisément, le Deep Learning ou l'apprentissage profond est une des techniques d'apprentissage automatique les plus remarquables, qui a connu un grand succès dans de nombreuses applications telles que l'analyse d'images, la reconnaissance de la parole et la compréhension de textes. Ce type d'apprentissage se base sur l'utilisation des stratégies supervisées et non supervisées pour apprendre des représentations multi-niveaux dans des architectures hiérarchiques appliqué généralement aux tâches de classification et de reconnaissance de formes.

2. Problématique et motivation

Au cours de ces dernières années, le Deep Learning a joué un rôle important dans le Data Mining. En effet, plusieurs techniques de Data Mining sont issues du domaine de l'apprentissage automatique. Le Deep Learning, étant un type d'apprentissage, ses modèles ont été utilisés dans des tâches de Data Mining.

L'extraction des motifs fréquents est une technique de Data Mining qui se repose sur un principe relativement simple. Il consiste à extraire les structures de données qui se répètent fréquemment dans un ensemble de données. Plusieurs méthodes et algorithmes ont été proposés dans la littérature dans ce contexte. La majorité de ces méthodes souffrent de

problème de passage à l'échèle surtout en passant à l'ère de Big Data, avec des quantités de données gigantesque.

Motivés par le succès de Deep Learning dans le traitement de données de masse et l'efficacité de l'application de ses modèles dans plusieurs domaines, nous nous somme poser la question suivante :" Peut on crée un modèle de Deep Learning pour l'extraction de motifs (itemsets) fréquents, qui peut être à la fois efficace tout en ayant un temps d'exécution capable de rivaliser avec les méthodes qui existe ?".

3. Objectifs

Le but de notre travail est en premier lieu, l'exploration de cet axe de recherche et la détermination est ce que le Deep Learning est un une méthode appropriée pour l'extraction des motifs fréquents. Cette dernière étant une méthode descriptive de Data Mining; or, le Deep Learning est appliqué généralement dans les méthodes prédictives.

En deuxième lieu l'adaptation d'un des modèles de Deep Learning ou la proposition d'un nouveau modèle qui permet d'extraire des itemsets fréquents. Autrement dit, développer une nouvelle méthode d'extraction des itemsets fréquents basée sur le Deep Learning.

4. Organisation du mémoire

Le mémoire se reparti en cinq chapitres

Chapitre 1 : « L'extraction des motifs fréquents »

Ce chapitre est consacré à la présentation des différents types d'algorithme pour l'extraction des items fréquents les plus connue.

Chapitre 2 : « Deep Learning »

Dans ce chapitre nous allons présenter c'est quoi le Deep Learning ainsi que les différentes architectures utilisées le domaine utilisé.

Chapitre 3 : « État de l'art »

Dans ce chapitre nous présenterons quelques revues et techniques passé qui se rapproches le plus de notre objectif.

Chapitre 4 : « Approche proposé »

Ce chapitre, sera réservé pour exposer notre solution proposée et ces résultats.

Chapitre 5 : « Tests et Expérimentation »

Le dernier chapitre concrétise et valide la méthode adoptée.

Chapitre I

L'extraction des itemset fréquents

1. Introduction

La découverte des connaissances utiles à partir des données est un processus de la fouille de données. L'une des techniques les plus utilisées pour extraire ces connaissances est la méthode d'extraction des règles d'association, elle permet de rechercher des corrélations entres les objets appelés motifs dans un ensemble de données.

La question qui se pose donc est, c'est quoi l'extraction des motifs fréquents ? Quels sont les algorithmes les plus utilisés pour l'extraction des motifs fréquents ?

2. Définitions

L'extraction des règles d'associations, la tâche de trouver des corrélations entre les items a reçu une attention considérable, particulièrement depuis la publication de l'algorithme Apriori par Agrawal 1993 pour l'analyse de panier d'achat sous forme de règles associatives. Ce qui est utile pour la découverte des relations intéressantes cachées dans des ensembles de données.

Depuis la première proposition de cette nouvelle méthode de fouille de données et ses algorithmes efficaces, il y a eu un flux de centaines de publications dans ce contexte. Par conséquent, L'analyse et l'extraction des règles d'association sont devenues un champ de recherche mature, les principes fondamentaux de cette méthode sont maintenant bien établis.

Le processus d'extraction de règles d'association passe par deux étapes :

1. L'identification motifs fréquents de l'ensemble de données.
2. La dérivation consécutive des règles d'inférence à partir des motifs fréquents.

Par conséquent, L'extraction des motifs fréquents constitue une étape intermédiaire pour la découverte des règles d'association. Néanmoins, le résultat de cette technique peut être aussi directement exploités et interprété, ou même utilisé comme une entré pour d'autre méthodes de découverte de connaissance [2].

Les motifs fréquents peuvent être définis comme étant des itemsets (ensembles d'attributs) qui apparaissent dans un jeu de données avec une fréquence non inférieure à un seuil défini par l'utilisateur.

- **Base de données :**

Soit O un ensemble fini d'objets, P un ensemble fini d'éléments ou items, et R une relation binaire entre ces deux ensembles. On appelle base de données ou contexte formel le triplet D = (O, P, R). La base de données D représente l'espace de travail [1].

- **Transaction :**

Soit T = {T1, T2, T3…, Tn}, Ti □ D. On appelle Ti un ensemble de lignes contenant les occurrences de la base de données D. Tous les Ti sont appelés des transactions. Dans l'exemple connu du panier de la ménagère, les transactions sont les tickets de caisse (c'est-à-dire les achats effectués par les clients) [1].

- **Item :** On appelle item toute variable Xi représentant une occurrence de D [1].

- **Itemset :** On appelle itemset, l'ensemble formé d'items. Exemple le singleton {X1} et la paire {X1, X2} sont des itemsets. Un itemset de taille k est noté k-itemset [1].

- **Support :** On appelle support le pourcentage de Ti où apparait la règle d'association, C'est-à-dire :

$$\textbf{(1)} \quad support(Xi \rightarrow Xj) = \frac{Freq(Xi \cup Xj)}{card(T)}$$

Ou $support(xi \rightarrow xj)$ Nbre de fois où Xi et Xj apparaissent ensemble dans les transactions T[1].

- **Confiance**

On appelle confiance le pourcentage de fois où la règle est vérifiée, c'est-à-dire [1].

$$\textbf{(2)} \quad confiance(Xi \rightarrow Xj) = \frac{Freq(Xi \cup Xj)}{Freq\ (Xi)}$$

- **Superset**

Un superset est un itemset défini par rapport à un autre itemset. Exemple {a, b, c} est un superset de {a, b} [1].

- **Itemset fréquent**

  Un Itemset fréquent est un itemset dont le support est ≥ à minsup (support minimal en dessous duquel l'itemset est considéré non fréquent). Si un itemset n'est pas fréquent, tous ses supersets ne le seront pas non plus. Si un superset est fréquent alors tous ses sous itemsets sont aussi fréquents (propriété anti-monotone) [1].

- **Itemset fermé**

  Un itemset fréquent est dit fermé si aucun de ses supersets n'a de support identique Autrement dit, tous ses supersets ont un support strictement plus faible [1].

- **Itemset libre**

  Un itemset est libre s'il n'est pas inclus dans la fermeture d'un de ses ensembles stricts [1].

- **Itemset maximal**

  Un itemset est dit maximal si aucun de ses supersets n'est fréquent [1].

- **Itemset générateur**

  Un itemset est dit générateur si tous ses sous-itemset sont un support strictement supérieur [1].

3. Les algorithmes d'extraction des itemsets fréquents

Un survol des algorithmes d'extraction des motifs fréquents parus dans la littérature a montré que la majorité de ces algorithmes sont des extensions de trois méthodologies principales à savoir, *Apriori*, *FP-growth* et *Eclat* qui appartiennent à deux catégories "générer et tester" et diviser pour régner"

On explicitera par conséquent dans cette section ces trois algorithmes de base, le reste des algorithmes ne sont que des variantes de ces principaux algorithmes.

3.1 L'approche de génération et de test des candidats

L'algorithme de base, représentant de cette catégorie est Apriori [3]. Le premier algorithme proposé dans la littérature introduisant ainsi ce domaine.

3.1.1 L'algorithme Apriori

Agrawal et Srikant (1994) ont initialement proposé l'algorithme Apriori. Cet algorithme est basé sur la propriété Apriori qui stipule que "chaque sous- (k-1)-Itemset de k-Itemset fréquents doit être fréquent"[3].

Deux processus principaux sont exécutés dans l'algorithme apriori :

- Le premier est le processus de génération d'un candidat, dans lequel le compte de support des items correspondants du capteur est calculé en scannant la base de données transactionnelle.
- le second est la génération d'un gros item set, qui est généré en éliminant les items candidats qui ont un compte de support inférieur au seuil minimum.

Ces processus sont répétés itérativement jusqu' à ce que les items candidats ou les gros items deviennent vides comme dans l'exemple illustré à la figure 1.1. La base de données originale est scannée pour la première fois pour l'ensemble de candidats, qui consiste d'un seul élément qui sont support a été compté, puis ces 1-Elément candidats ensemble sont élagués en supprimant simplement les items qui ont un nombre d'éléments inférieur au seuil spécifié par l'utilisateur (dans le cas supérieur seuil =20%). Dans la deuxième passe, la base de données est scannée à nouveau pour générer 2-Itemset candidats se composent de deux éléments, puis à nouveau taillé pour produire de grands 2-Itemset en utilisant la propriété apriori. Selon la propriété apriori, chaque sous-ensemble de 2 items fréquents doit être fréquent. Dans l'exemple ci-dessous le processus se terminera aux quatrièmes balayages de la base de données pour donner les 4-Itemset candidat et le jeu d'items de grande taille sera vide [5].

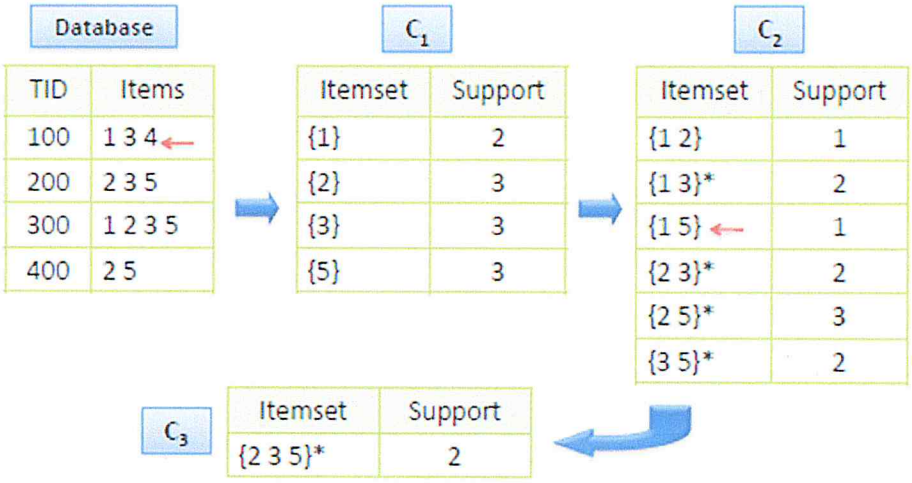


Figure 1. 1 : Exemple explicatif de l'algorithme Apriori [4].

3.1.1.1. Récapitulatif de l'algorithme Apriori.

Algorithme : Apriori [6]

Calculer L1
K = 2
TANTQUE L_{k-1} <> φ FAIRE
  C_k = apriori – gen(L_{k-1})
    TANTQUE t ∈ D FAIRE
      C_t = sousensemble (C_k , t)
        TANTQUE c ∈ C_t FAIRE
          c.count ++
        FIN TANTQUE
    FIN TANTQUE
  L_k + {c ∈ C_k | c.count ≥ minSup}
  K = k+1
FIN TANTQUE
RETOURNER $\cup_k L_k$

3.1.1.2. Variantes de l'algorithme Apriori

Il existe d'autres variantes de l'algorithme apriori comme :

- Apriori TID [7]
- DHP (Direct hashing and Pruning) [8]
- Apriori-Hybrid [13]

3.2 L'approche diviser pour régner

3.2.1 L'algorithme FP-Growth

Dans le domaine du data mining, l'algorithme le plus utilisé pour la détection de motifs est l'algorithme FP-Growth. Pour traiter les deux principaux inconvénients de l'algorithme Apriori [9]. Une nouvelle structure de données compressée nommée FP-tree est construite, qui est une structure d'arborescence de préfixes qui stocke des informations quantifiables sur les motifs fréquents. Un algorithme de croissance fréquente a été développé à partir de FP-tree [10].

L'algorithme FP-growth (Fréquent Pattern growth) consiste d'abord à compresser la base de données en une structure compacte appelée FP-Tree (Fréquent Pattern tree), puis à la diviser ainsi compressée en sous projections pour représenter la base de données. Chacune de ces projections est associée à un item fréquent. L'extraction des itemsets fréquents se fera sur chacune des projections séparément.

L'algorithme FP-growth apporte ainsi une solution au problème de la fouille de motifs fréquents dans une grande base de données transactionnelle. En stockant l'ensemble des éléments fréquents de la base de transactions dans une structure compacte, on supprime la nécessité de devoir scanner de façon répétée la base de transactions. De plus, en triant les éléments dans la structure compacte, on accélère la recherche des motifs. [11].

3.2.1.1. Structure d'un FP-tree

Deux éléments essentiels constituent la structure d'un FP-tree qui sont :

1. Une structure sous forme d'un arbre avec une racine étiquetée nulle.

2. Un index (une table des pointeurs des items fréquents).

L'arbre quant à lui est composé, d'une racine nulle et d'un ensemble de nœuds préfixé par l'élément représenté.

Un nœud de l'arbre est composé par :

- Le nom de l'item (nom-item). Il s'agit de l'item que représente le nœud.
- Le nombre d'occurrence (count) de transaction où figure la portion de chemin jusqu'au nœud.
- Un lien vers le nœud suivant dans l'arbre. Il s'agit d'un lien inter-nœud vers les autres occurrences du même élément (ayant le même nom-item) figurant dans d'autres séquences de transactions. Cette valeur prend la valeur nulle s'il n'y a pas un tel nœud.

L'Index est une table d'en-tête qui contient la liste des items fréquents et qui pointe sur la première occurrence de chaque élément. [11]

Chaque entrée dans cette table contient :

- Le nom de l'élément (nom-item).
- Le pointeur tête de la séquence des nœuds ayant ce même nom-item.

3.2.1.2. Construction d'un FP-Tree

La construction du FP-tree nécessite deux parcours de la base des transactions (T) et se fait de la manière suivante :

On effectue un premier parcours de la base T pour déterminer les items fréquents en fonction du support minimum fourni. Ces items seront triés par la suite par ordre décroissant de support dans une liste (L). Les items ainsi triés seront traités dans cet ordre.

Un second parcours de T est alors effectué. Chaque transaction est alors triée selon l'ordre des items dans L. Le nœud racine de l'arbre nulle est d'abord crée. Durant ce même parcours, une branche sera créé pour chaque transaction, mais des transactions ayant un même préfixe partageront le même début d'une branche de l'arbre, ainsi deux transactions identiques seront représentées par une seule et même branche.

La raison pour laquelle les items sont traités du plus fréquent au moins fréquent est que les items fréquents seront proches de la racine et seront mieux partagés par les transactions. Ceci fait du FP-tree une bonne structure compacte pour représenter les bases transactionnelles [11].

3.2.1.3. Récapitulatif de l'algorithme FP-growth

Algorithme : FP-growth [11]

1 Balayer la base de transaction T une première fois

- Créer L, la liste des items fréquents avec leur support.
- Trier L en ordre décroissant du support.

2 Créer l'arbre N contenant une racine étiquetée «Null ».

3 procedure FP-growth(Fp-tree, null)

A. Si FP-tree contient un seul chemin P alors
   - Pour chaque combinaison β de P faire
   - Générer l'itemset β U α de support = minimum des supports des nœuds de β.

B. Sinon pour chaque a_i dans l'index de FP-tree faire
   - Générer l'itemset $\beta = a_i$ U α de support = a_i. Support.
   - Construire l'itemset condition de base de β.
   - Construire FP-tree$_\beta$
   - Si FP-tree$_\beta \neq 0$
     - FP-growth (FP-tree$_\beta$, β)

3.2.1.4. Étapes de la construction du FP-tree

La construction d'une structure FP-tree passe par 6 étapes principales. Les étapes 1 à 5 préparent la structure et insèrent les éléments qui doivent s'y trouver. La sixième étape consiste à valider les informations insérées dans les étapes précédentes [11] :

1. Calculer le support minimum
2. Parcours de la base des transactions pour trouver la somme totale des différentes occurrences.
3. Définir la priorité des éléments, puis trier les items en fonction de leur priorité.

4. Création du nœud racine.
5. Insertion des nœuds enfants.
6. Validation.

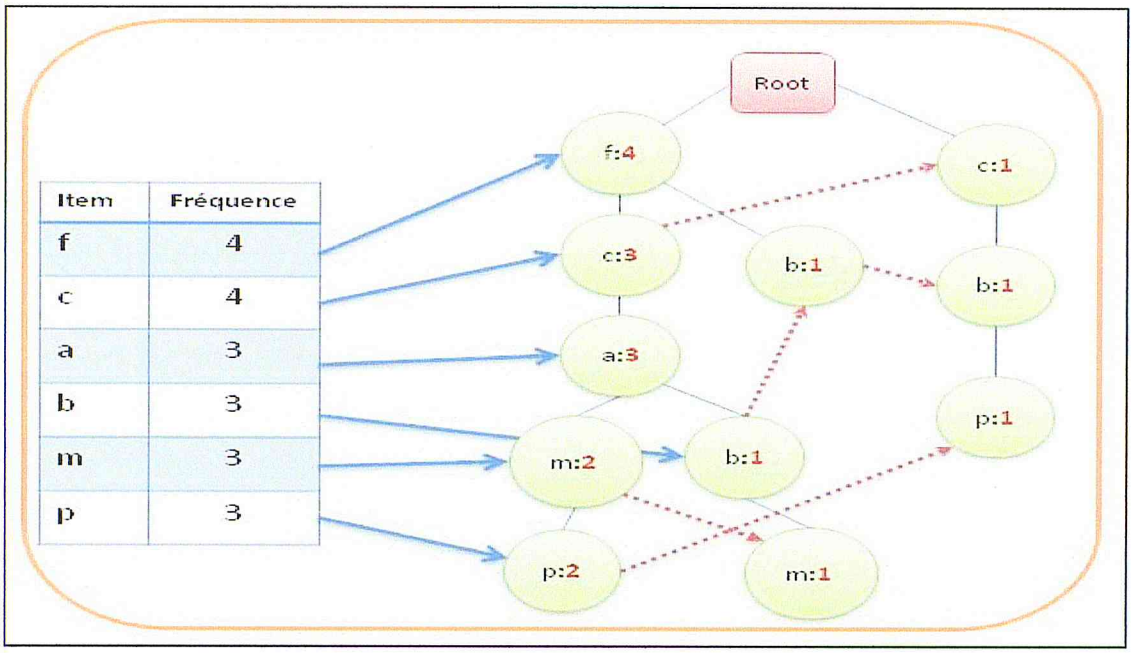


Figure 1. 2 : Exemple d'état final de la structure FP-tree [12].

3.2.2. L'algorithme H-Mine

H-mine, est proposé pour extraire des motifs fréquents pour les ensembles de données qui peuvent tenir dans la mémoire (principale). Une hyper-structure simple, basée sur la mémoire, H-struct, est conçue pour une extraction rapide. Théoriquement, H-mine a une complexité spatiale polynomiale et est donc plus efficace en termes d'espace que les méthodes de croissance de motifs telles que FP-growth et TreeProjection lors de l'exploitation d'ensembles de données éparses, et aussi plus efficace que les méthodes basées sur Apriori qui génèrent un grand nombre de candidats. Les résultats expérimentaux montrent que, dans de nombreux cas, H-mine a un espace plafond très limité et exactement prévisible et est plus rapide que les méthodes Apriori et FP basées sur la mémoire [17].

3.3.2.1. Récapitulatif de l'algorithme H-mine.

Algorithme : H-mine [17]
Entrée : une base de données de transactions TDB et un seuil de support. min sup. **Sortie :** L'ensemble complet des motifs fréquents. **Méthode :** **Étape 1.** Scan TDB une seule fois, trouver et sortir L, l'ensemble d'items fréquente. Laisser F-List : "x1- . . . xn" (n = |L|) être une liste des items fréquents. **Étape 2.** Scannez à nouveau TDB, construisez H-struct, avec la table d'en-tête H, et avec chaque xi-queue liée à la file d'attente correspondante. Entrée en H. **Étape 3.** Pour i = 1 à n faire **(a)** Appeler H-mine ({xi}, H, F-liste). **(b)** parcourir la xi-queue dans le tableau d'en-tête H, pour chaque projection X d'item fréquents, lier X à la xj-queue dans le tableau d'en-tête H, où xj est l'item dans X après xi immédiatement.

3.3 Les versions parallèles et distribuées des algorithmes d'extraction des itemsets fréquents

- **Les algorithmes parallèles**

  L'un des facteurs clés dans le choix d'utiliser des algorithmes parallèles au lieu de leurs homologues en série sont des données trop volumineuses pour pouvoir être stockées en mémoire sur un seul poste de travail. Même si l'ensemble de données d'entrée peut tenir dans la mémoire, les données intermédiaires telles que le candidat et leurs comptages, ou les structures de données utilisées lors de la fouille fréquente, ne peut pas.

  Un deuxième défi important dans la conception d'un algorithme parallèle efficace est de décomposer le problème en un ensemble de tâches, où chaque tâche représente une unité de travail, les tâches sont indépendantes et peuvent être exécutées simultanément, en parallèle [105].

- **Les algorithmes distribués**

Les algorithmes distribués sont un sous-type des algorithmes parallèles, généralement des parties séparées de l'algorithme sont exécutées simultanément sur des processeurs indépendants et ne disposent que de peu d'informations sur ce que font les autres parties de l'algorithme [106].

Un système de mémoire distribuée (ne partage rien), les processus n'ont accès qu'à un fichier l'espace adresse mémoire privée locale. Le partage des données d'entrée et des informations sur les tâches doit être fait explicitement au moyen de la communication entre les processus [107].

3.3.1 Parallel FP-Growth

L'algorithme FP-Growth génère récursivement d'énormes quantités de motifs conditionnels et des FP-tree conditionnels lorsque l'ensemble de données est énorme. Dans un tel cas, à la fois l'utilisation de la mémoire et le coût de calcul sont coûteux, de sorte que le FP-tree ne peut pas répondre aux exigences de mémoire. Parallèle FP-Growth (GFP-Growth), est conçu pour fonctionner sur le cluster des ordinateurs. Pour éviter le débordement de mémoire, cet algorithme recherche toutes les bases de motifs conditionnels des items fréquents par la méthode de projection sans construire un FP-tree. Par la suite, il divise la tâche d'exploitation en nombre de sous-tâches indépendantes, exécute ces sous-tâches en parallèle sur les nœuds et ensuite regroupe les résultats pour le résultat final. L'algorithme fonctionne indépendamment à chaque nœud. Il peut ainsi réduire efficacement le coût de communication entre nœuds. Les essais montrent que cet algorithme parallèle permet non seulement d'éviter le débordement de mémoire mais aussi d'accélérer la vitesse de calcul. De plus, il offre une meilleure scalabilité que l'algorithme FP-Growth [16].

3.2.2.1 Récapitulative de l'algorithme Parallel FP-Growth (GFP-Growth)

Algorithme : Parallel FP-Growth [16]

```
{
  Trouver la liste de base de motif conditionnel (PrefixDB) ;
  Le nœud principal déploie toutes les bases de motifs conditionnelles
  aux nœuds de cluster;
  Pour chaque Cond_phase (b) dans PrefixDB, faire
        {
```

```
        Cond_FPtree (b) = construire un FP-tree conditionnel selon Cond_phase (b);
        Si Cond_FPtree (b) != Ø alors
            Appeler FP-Growth (Cond_FPtree (b), b);
Sous-résulte est soumis au nœud principal ;
        }
Le nœud principal regroupe tous les sous-résultats en un résultat total;
Retourne le résultat ;
}
```

3.2.2.2 Caractéristiques de l'algorithme GFP-Growth

- D'abord, diviser la tâche entière en sous-tâches indépendantes : les algorithmes FP-Growth parallèles les plus anciens divisent la base de données originale en plusieurs sous-bases de données.

  Comme les bases de sous-bases de données peuvent être interdépendantes, une communication fréquente est nécessaire entre les threads d'exécution parallèles. dans GFP-growth la solution consiste à diviser l'ensemble de la tâche en un groupe de sous-tâches indépendantes les unes des autres, puis à déployer les sous-tâches indépendantes sur des nœuds de calcul pour traiter en parallèle. Une telle partition évitera le besoin de communication d'informations et de données entre ces exécutions de sous-tâches. et les coûts de communication sont réduits [16].

- Deuxièmement, l'équilibrage dynamique de charge : l'équilibrage dynamique de charge est une caractéristique très importante pour un cluster d'ordinateurs pour atteindre une utilisation optimale des ressources computationnelles d'un système. En supervisant l'état de charge des nœuds de computation, le plan de répartition des charges change au cours du temps de traitement. Selon l'algorithme GFP-Growth, chaque nœud communiquera périodiquement les sous-résultats et l'état de la charge au nœud principal afin que ce dernier puisse ajuster la décision de répartition de la charge dans le temps. Si un nœud n'a pas répondu au nœud principal pendant plus d'une durée supérieure à une valeur définie auparavant, le nœud principal l'indiquera comme étant mort et déplacera ses sous-tâches vers d'autres nœuds [16].

- Troisièmement, la communication asynchrone : dans l'algorithme GFP-Growth, la communication est nécessaire entre le nœud principal et les nœuds non principaux pour déployer des sous-tâches et obtenir des sous-résultats. En général, il existe deux méthodes de communication, la méthode synchrone et la méthode asynchrone. La méthode synchrone signifie que tous les nœuds commencent à communiquer ensemble après la fin de la tâche ; alors qu'avec la méthode asynchrone, la communication entre les nœuds ne commence pas ensemble. L'algorithme GFP-Growth adopte la méthode asynchrone : chaque nœud non principal communique avec le nœud principal dès qu'il termine une sous-tâche, de sorte que le nœud principal peut obtenir des informations d'autres nœuds dans les délais pour prendre une meilleure décision de répartition de la charge [16].

3.3.2 Parallel Eclat

Dans Parallel Eclat (Par-Eclat), les données sont en disposition verticale, ce qui se fait en transformant les données horizontales en liste verticale d'items. Les deux phases de l'Eclat Parallèle sont données ci-dessous [14] :

Phase d'initialisation : Il y a 3 sous-étapes dans l'étape d'initialisation

1. Le compte de support pour les 2-itemsets est lu après l'étape de prétraitement et tous les items fréquents ayant le compte >= support minimum sont introduits dans Freq2. (Freqk est l'ensemble des k-itemsets fréquents)
2. L'un ou l'autre des schémas de clustering entre ces deux - la classe d'équivalence ou le clustering maximum de clique hypergraphe - est appliqué à Freq2 pour produire l'ensemble des items maximums possibles qui sont ensuite divisés entre tous les processeurs pour atteindre l'équilibrage de charge.
3. La base de données est repartitionnée de façon à ce que chaque processeur obtienne la liste des TID de tous les 1- itemsets du cluster qui lui sont affectés.

Phase asynchrone :

Après la phase d'initialisation, les tid-lists sont accessibles sur le stockage local sur chaque processeur, de sorte que chaque processeur peut utiliser son cluster maximal assigné pour générer des items fréquents sans synchronisation avec un autre processeur. Un cluster est traité complètement avant que le processeur ne passe au cluster suivant.

3.3.2.1 Récapitulative de l'algorithme Parallel Eclat

Algorithme : Parallel Eclat [14]
1. Créer Freq2 à partir des comptes de support 2-itemset 2. Produire des clusters à partir de Freq2 en utilisant les classes d'équivalence 3. Affecter les clusters aux processeurs 4. Scanner une partie du jeu de données local 5. Partager les tid-lists appropriées avec les autres processeurs. 6. Obtenu les tid-lists des autres processors 7. Pour chaque cluster C trouve les objets fréquemment en utilisant la méthode Bottom-Up ou la méthode hybride.

3.3.3 Algorithme Dist-Eclat

Dist-Eclat est une méthode en 3 étapes (figure 2.1). Chaque étape peut être répartie entre les cartographes pour maximiser l'efficacité [15].

1. **Trouver les items fréquents :** Dans cette étape, la base de données verticale est divisée en sous-bases de données de taille égale appelées shards et distribuée entre plusieurs mappeurs. Les cartographes extraient les singletons fréquents de leur shard et les envoient au réducteur. Tous les items fréquents sont rassemblés dans la phase de réduction.
2. **Génération de k-itemsets fréquents :** L'ensemble des k-itemsets fréquents, Freqk, est généré. Tout d'abord, chaque mappeur se voit attribuer la forme combinée des ensembles d'items fréquents locaux. Mappers trouver les supersets fréquents de taille k des items en utilisant l'algorithme d'Eclat et en l'exécutant jusqu'au niveau k. Puis un réducteur assigne les items fréquents à un nouvel ensemble de Mappers. La distribution aux cartographes est effectuée en utilisant l'algorithme Round-Robin.
3. **Extraction de sous arbres :** Cette étape utilise l'algorithme Eclat pour extraire l'arbre de préfixes des sous-ensembles assignés. Les sous arbres peuvent être exploités indépendamment par les cartographes puisque les sous arbres n'ont pas besoin d'informations provenant d'autres sous arbres.

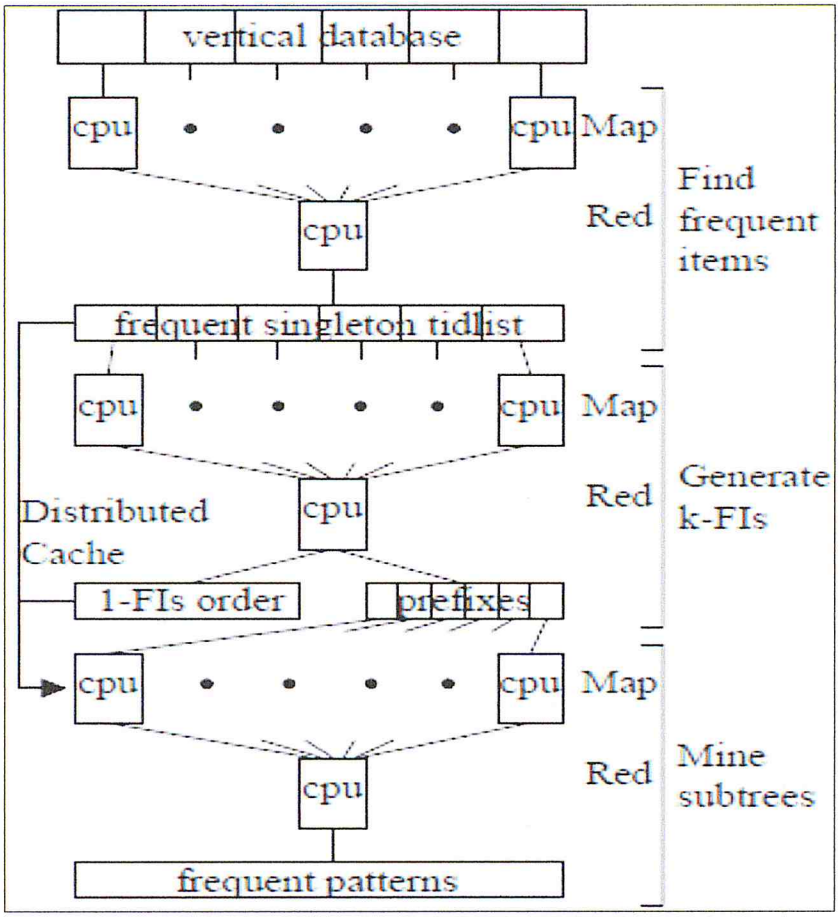


Figure 1. 3 : Eclat sur MapReduce Framework [15].

4. Conclusion

Dans ce premier chapitre, nous avons vu les algorithmes d'extractions d'itemsets fréquents. En effet, on a donné une réponse à la question posée dans l'introduction de ce chapitre, nous avons aussi expliqué leur fonctionnement. Dans le prochain chapitre nous allons étudier l'apprentissage profond (Deep Learning).

Chapitre II

Apprentissage Approfondi

1. Introduction

Comment faire apprendre un ordinateur ? Vous pouvez soit écrire un programme fixe ou vous pouvez permettre à l'ordinateur d'apprendre par lui-même. Un être doté d'une conscience n'a pas de programmeur qui lui permet d'apprendre ou de développé des compétences, Ils apprennent par eux-mêmes, tout ca sans des connaissances extérieures au préalable.et peut donc résoudre les problèmes mieux que n'importe quel ordinateur aujourd'hui. Quelles qualités sont nécessaires pour qu'un ordinateur est les mêmes comportements ? [18] Dans ce chapitre nous présentons une étude sur les réseaux de neurones, ainsi qu'une introduction au Deep Learning, suivie des architectures actuellement utilisés.

2. Apprentissage automatique

L'apprentissage automatique fait habituellement référence aux changements dans les systèmes qui exécutent des tâches associées à l'intelligence artificielle (IA). De telles tâches comprennent la reconnaissance, le diagnostic, la planification, la commande de robot, la prédiction, etc. Les " changements " pourraient être soit des améliorations à des systèmes déjà performants, soit une synthèse de nouveaux systèmes [19].

Nous pourrions dire, très largement, qu'une machine apprend chaque fois qu'elle modifie sa structure, son programme ou ses données (sur la base de ses entrées ou en réponse à des informations externes) d'une manière telle que ses performances futures s'améliorent. Certains de ces changements, comme l'ajout d'un document à une base de données, se situent confortablement dans la province d'autres disciplines et ne sont pas nécessairement mieux compris pour ce qu'on appelle apprentissage. Mais, par exemple, lorsque la performance d'une machine de reconnaissance vocale s'améliore après avoir entendu plusieurs échantillons de la parole d'une personne, nous nous sentons tout à fait justifiés dans ce cas de dire que la machine a appris [19].

Les méthodes et techniques d'apprentissage automatique sont divisées en plusieurs catégories, adoptant chacune d'elle un type d'apprentissage. Ces types d'apprentissage sont explicités dans ce qui suit :

2.1. L'apprentissage supervisé

L'apprentissage supervisé ajuste les paramètres du réseau par une comparaison directe entre la sortie réseau réelle et la sortie souhaitée. L'apprentissage supervisé est un système de rétroaction en boucle fermée, où l'erreur est le signal de rétroaction. La mesure d'erreur, qui montre la différence entre la sortie réseau et la sortie des échantillons de l'entrainement La mesure de l'erreur est habituellement définie par l'erreur quadratique moyenne (EQM).

$$E = \frac{1}{N} \sum_{p=1}^{N} \left\| y_p - \hat{y}_p \right\|^2 , \quad (3)$$

Où **N** est le nombre de paires de motifs dans l'ensemble d'échantillons, **yp** est la partie de sortie de la **p**'eme paire de motifs, et **ˆyp** est la sortie réseau correspondant à la paire de motifs **p**. L'erreur **E** est calculée à nouveau après chaque époque. Le processus d'apprentissage se termine lorsque **E** est suffisamment petit ou lorsqu'un critère d'échec est satisfait [20].

2.2. L'apprentissage Non supervisé

L'apprentissage non supervisé n'implique aucune valeur cible. Il essaie d'associer automatiquement l'information des entrées avec une réduction intrinsèque de la dimensionnalité des données ou de la quantité totale des données d'entrée. L'apprentissage non supervisé est uniquement basé sur les corrélations entre les données d'entrée, et est utilisé pour trouver les motifs ou les caractéristiques significatives dans les données d'entrée sans l'aide d'un enseignant. L'apprentissage non supervisé est particulièrement adapté à l'apprentissage biologique en ce sens qu'il ne dépend pas d'un enseignant et qu'il utilise des primitives intuitives comme la compétition et la coopération neuronale [20].

2.3. L'apprentissage semi-supervisé

Dans de nombreuses applications d'apprentissage automatique, telles que la bio-informatique, le web et le texte mining, la catégorisation de texte, le marketing de base de données, la détection du spam, la reconnaissance des visages et l'indexation vidéo, des quantités abondantes de données non étiquetées peuvent être collectées automatiquement.

Cependant, l'étiquetage manuel est souvent lent, coûteux et sujet aux erreurs. Lorsque seul un petit nombre d'échantillons étiquetés est disponible, des échantillons non étiquetés pourraient être utilisés pour prévenir la dégradation de la performance due à un sur apprentissage.

L'objectif de l'apprentissage semi-supervisé est d'employer une grande collection de données non étiquetées. Avec quelques exemples étiquetés pour améliorer la performance de généralisation. Certaines méthodes d'apprentissage semi-supervisé sont basées sur des hypothèses selon lesquelles relier la probabilité P(x) à la distribution conditionnelle P (Y = 1|X = x). L'apprentissage semi supervisé est lié au problème de l'apprentissage transductif [20].

2.4. L'apprentissage par renforcement

L'apprentissage par renforcement est un cas particulier d'apprentissage supervisé, dans lequel la valeur exacte de la sortie désirée n'est pas connue. L'enseignante ou l'enseignant ne fournit qu'une évaluation de la réussite ou l'échec d'une réponse. Cela est plus plausible sur le plan cognitif que l'apprentissage supervisé. Étant donné qu'une réponse correcte et entièrement spécifiée n'est pas toujours à la disposition de l'apprenant ou même l'enseignant [20].

Il est fondé uniquement sur l'information permettant de savoir si la sortie réelle est proche ou non de l'estimation. L'apprentissage par renforcement est une procédure d'apprentissage qui récompense le réseau neuronal pour son bon résultat de sortie et le punit pour le mauvais résultat de sortie [20].

Le calcul explicite des dérivés n'est pas nécessaire. Ceci, cependant, présente un processus d'apprentissage plus lent. Pour un système de contrôle, si le contrôleur fonctionne toujours correctement après une entrée, la sortie est jugée bonne ; sinon, elle est considérée comme mauvaise. L'évaluation de la sortie binaire, appelée renforcement externe, est utilisée comme signal d'erreur [20].

3. Les réseaux de neurones artificiels

Les réseaux de neurones artificiels sont communément appelés systèmes connexionnistes , Ces systèmes sont capables d'apprendre tout en améliorant progressivement les performances afin de réaliser différentes tâches en considérant quelques exemples mais sans programmation spécifique à la tâche [22], par exemple, dans la reconnaissance d'images, les réseaux de neurones artificiels pourrait facilement apprendre à identifier les images, qui contiennent des chats simplement en analysant différentes images d'exemple qui ont été étiquetées manuellement comme des images de chat. Les RNAs utilise en outre ces résultats d'analyse afin d'identifier les chats dans d'autres images obtenues.

Les RNAs sont couramment utilisés dans différentes applications en particulier lorsque c'est difficile d'utiliser la programmation basée sur des règles avec un algorithme traditionnel [22]. Généralement, les neurones sont organisés en couches. Il est noté que les différentes couches on la capacité d'effectuer différents types de transformations quand il s'agit de leurs entrées. Les signaux partent de l'entrée initiale à la dernière couche appelée sortie, en générale après avoir traversé ces couches plusieurs fois.

Le but ultime des réseaux de neurones artificiels est de résoudre différents problèmes exactement de la même manière qu'un cerveau humain [21]. Le modèle le plus simple qui illustre le fonctionnement d'un réseau de neurones est le perceptron, qui est un réseau composé d'une seule couche, Notons l'existence de réseaux de neurones à multiple-couches dits "acycliques" feedforward neural network. Dans ce réseau particulier, les différentes données ne peuvent se déplacer que dans une seule direction qui est l'avant. L'information est transférée de l'entrée à travers des nœuds cachés et finalement aux nœuds de sortie. Il n'y a aucun cycle formé dans cette structure.

3.1. Définition d'un neurone

Un neurone, est une fonction algébrique non linéaire et bornée, dont la valeur dépend de paramètres appelés coefficients ou poids. Les variables de cette fonction sont habituellement appelées « entrées » du neurone, et la valeur de la fonction est appelée « sortie ». Un neurone est donc avant tout un opérateur mathématique, dont on peut calculer la valeur numérique par quelques lignes de programme informatique. Il est très rarement réalisé physiquement sous la forme d'un objet (circuit électronique par exemple). Il est cependant pratique de le représenter graphiquement [23].

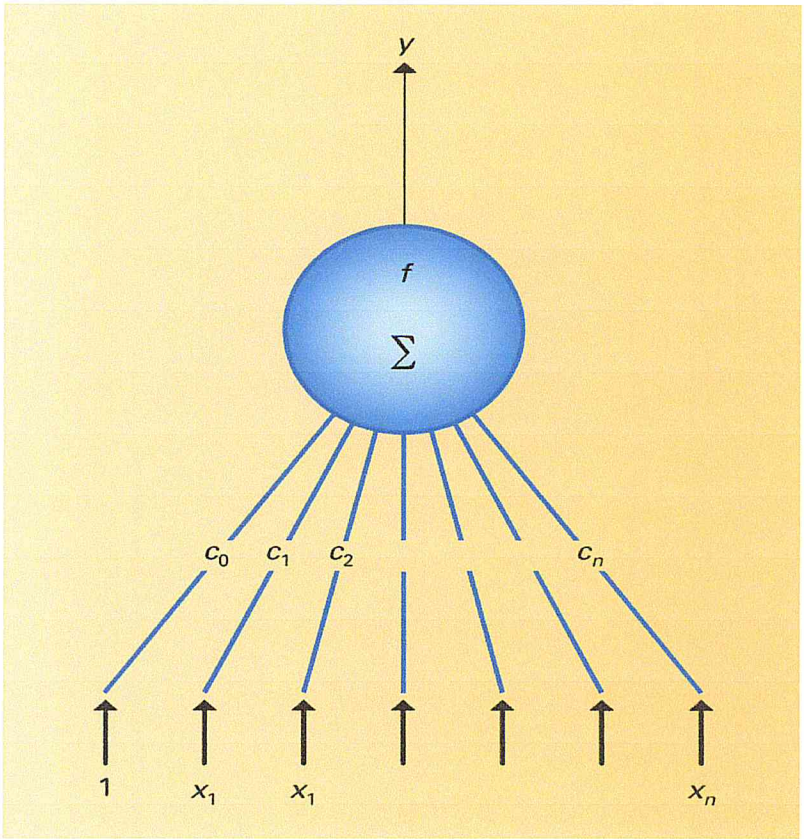


Figure 2. 1 : Représentation graphique d'un neurone [23].

3.2. Définition des réseaux de neurones

Un neurone réalise une fonction non linéaire de ses entrées. Ces dernières peuvent être les sorties d'autres neurones : on réalise ainsi un réseau de neurones, qui n'est donc rien d'autre que la composition de fonctions non linéaires élémentaires constituées par les neurones

individuels. La manière dont on « relie » entre elles les sorties des neurones aux entrées d'autres neurones définit l'architecture du réseau [23].

3.3 Fonction d'activation

Les fonctions d'activation sont une composante extrêmement importante des réseaux de neurones artificiels. Ils décident si un neurone doit être activé ou non. Si l'information que le neurone reçoit est pertinente pour l'information donnée ou si elle doit être ignorée [48].

$$Y = \text{Activation}\left(\sum (w \times x) + b\right)$$

W : poids

X : entrée

B : biais

Les fonctions d'activation peuvent être divisées en 2 types :

1) **Fonction d'activation linéaire**

La fonction est une ligne ou linéaire Par conséquent, la sortie des fonctions ne sera pas confinée entre n'importe quelle range.

Equation: $f(x) = x$

Range: $(-\infty, +\infty)$

Cela n'aide pas avec la complexité ou les divers paramètres des données habituelles qui sont introduites dans les réseaux de neurones [49].

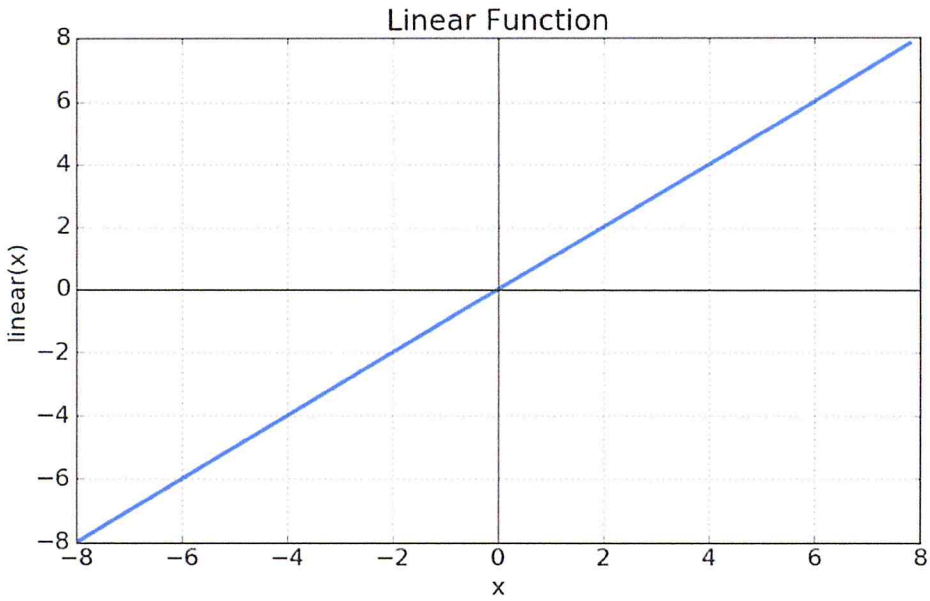


Figure 2. 2 : Fonction d'activation linéaire [49].

1) Fonction d'activation non linéaire

Les fonctions d'activation non linéaire sont les fonctions d'activation les plus utilisées. Il permet au modèle de généraliser ou de s'adapter à une variété de données et de différencier les résultats [10].

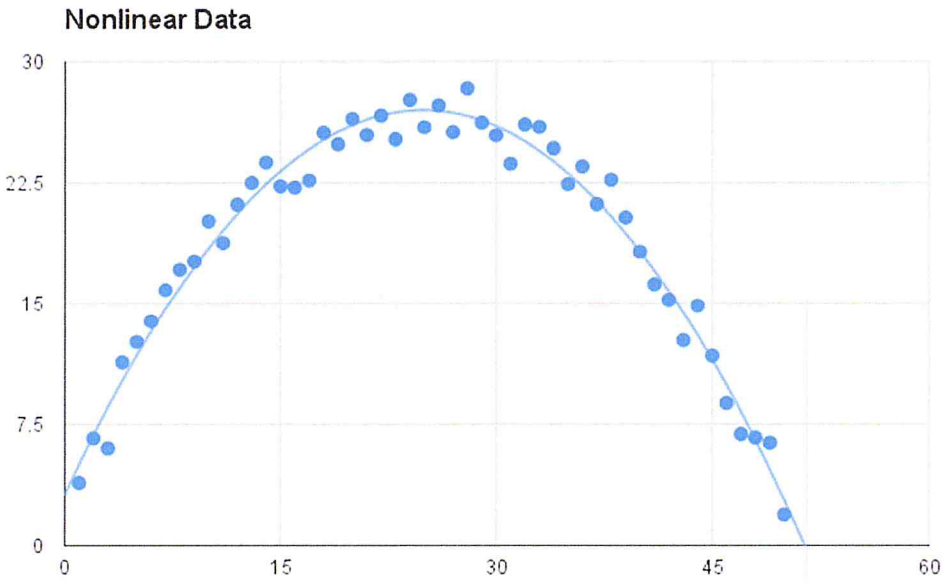


Figure 2. 3 : Fonction d'activation non linéaire [49].

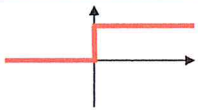
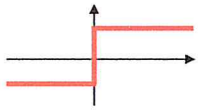
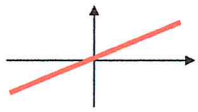
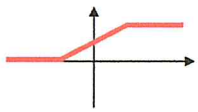
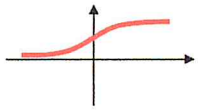
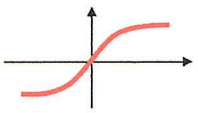
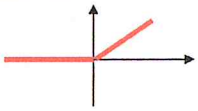
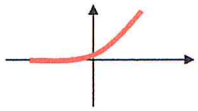
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Figure 2. 4 Fonctions d'activation [50].

3.4. Les limitations des réseaux de neurones classiques

La convergence est l'un des plus grands problèmes du perceptron. Frank Rosenblatt a prouvé mathématiquement que la règle d'apprentissage perceptron converge si les deux classes peuvent être séparées par un hyperplan linéaire, mais des problèmes surgissent si les classes ne peuvent pas être séparées parfaitement par un classificateur linéaire [24].

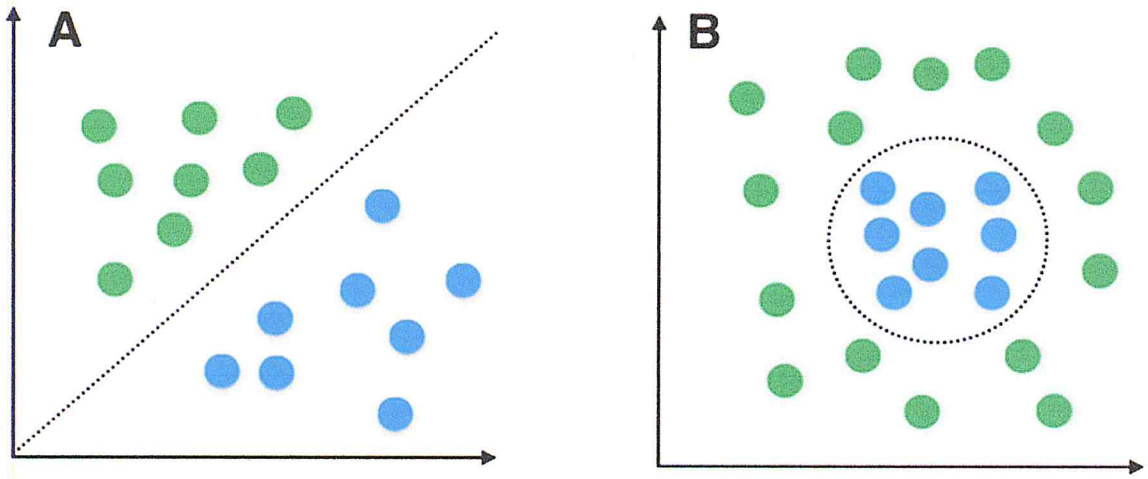


Figure 2. 5 : Données linéaire vs données non linéaire [38].

4. Apprentissage profond

Le Deep Learning ou L'apprentissage profond est un domaine relativement nouveau de méthodes plus larges d'apprentissage automatique et de recherche, qui a été introduit principalement dans le but de rapprocher la recherche en apprentissage automatique de l'intelligence artificielle. Les méthodes d'apprentissage profond consistent essentiellement à apprendre de grandes représentations de données ou, en d'autres termes, à apprendre des algorithmes de réseau neuronal [22].

Il convient de noter que l'apprentissage profond pourrait être effectué dans trois variétés : supervisé, non supervisé ainsi que l'apprentissage profond semi-supervisé [4]. De nombreuses interprétations d'apprentissage profond sont basées sur divers traitements de l'information et différents modèles de communication.

Les chercheurs en apprentissage profond se consacrent également à la création de systèmes capables d'apprendre diverses représentations d'apprentissage profond à partir de grandes collections de données ou de données volumineuses. Diverses structures d'apprentissage profond, y compris les réseaux de neurones profonds comme le convolutional neural networks sont largement utilisées dans différents domaines, y compris la reconnaissance de la parole, reconnaissance audio, traduction automatique de traitement de langage neuronal, filtrage de réseaux sociaux et autres.

L'apprentissage profond fait partie d'une plus grande famille d'apprentissage automatique, et les techniques d'apprentissage profond comprennent une cascade contenant de nombreuses couches d'unités cachées et en arrière-plan qui sont représentées de manière non linéaire.

Pour résumer l'apprentissage profond nous pouvons simplement dire que les méthodes d'apprentissage profond utilisent différentes unités de couche et que l'apprentissage supervisé et non supervisé de diverses représentations d'entités de données est représenté dans chacune de ces couches. Leur composition dépend principalement du problème qui va être résolu [18] [22] [25].

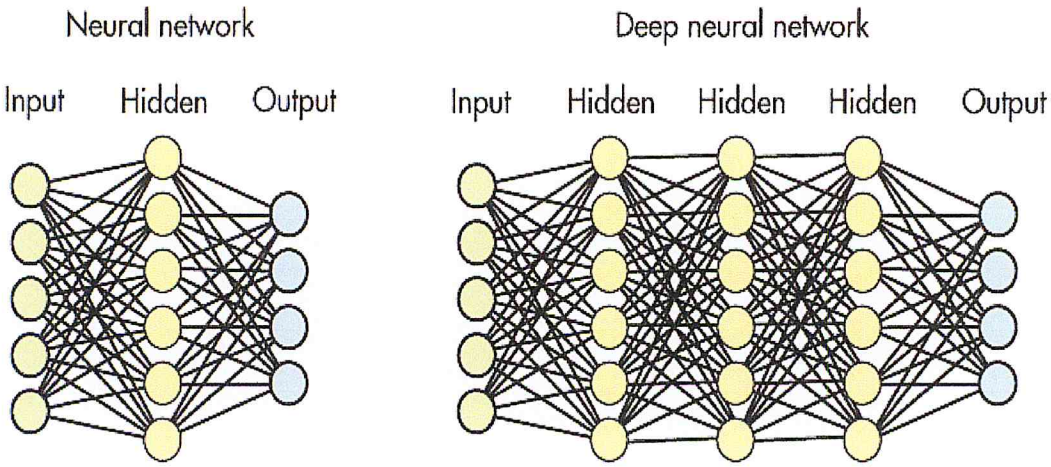


Figure 2. 6 : Différence entre "Neural network" et "Deep neural network" [42].

4.1. Différence entre intelligence artificiel Deep Learning et Machine Learning

1. Intelligence Artificiel

Créée en 1956 par John McCarthy, l'intelligence artificielle fait appel à des machines capables d'accomplir des tâches caractéristiques de l'intelligence humaine. Bien que cela soit plutôt général, cela inclut des choses comme la planification, la compréhension du langage, la reconnaissance des objets et des sons, l'apprentissage et la résolution de problèmes [43].

On peut classer l'IA en deux catégories, générale et étroite. L'IA générale aurait toutes les caractéristiques de l'intelligence humaine, y compris les capacités mentionnées ci-dessus. L'intelligence artificielle étroite présente certaines facettes de l'intelligence humaine, et peut très bien faire cette facette, mais elle fait défaut dans d'autres domaines. Une machine qui reconnaît très bien les images, mais rien d'autre, serait un exemple d'IA étroite [43].

2. Machine Learning

Au fond, l'apprentissage automatique n'est qu'un moyen d'atteindre l'IA. Arthur Samuel a inventé la phrase peu de temps après AI, en 1959, la définissant comme " la capacité d'apprendre sans être explicitement programmé ". Vous pouvez obtenir l'IA sans utiliser l'apprentissage automatique, mais cela nécessiterait la construction de millions de lignes de codes avec des règles et des arbres de décision complexes [43].

Au lieu de coder des logiciels avec des instructions spécifiques pour accomplir une tâche particulière, l'apprentissage automatique est une façon de " entraîner " un algorithme afin qu'il puisse apprendre comment. " L'entraînement " consiste à fournir d'énormes quantités de données à l'algorithme et à permettre à l'algorithme de s'ajuster et de s'améliorer [43].

3. Deep Learning

L'apprentissage profond est l'une des nombreuses approches de l'apprentissage automatique. D'autres approches comprennent l'apprentissage par arbre de décision, la programmation logique inductive, le clustering, l'apprentissage par renforcement et les réseaux bayésiens, entre autres [43].

L'apprentissage profond a été inspiré par la structure et la fonction du cerveau, à savoir l'interconnexion de nombreux neurones. Les réseaux de neurones artificiels (RNA) sont des algorithmes qui imitent la structure biologique du cerveau [43].

Dans les RNA, il y a des "neurones" qui ont des couches discrètes et des connexions à d'autres "neurones". Chaque couche choisit une caractéristique spécifique à apprendre, comme les courbes/bords dans la reconnaissance d'image. C'est cette superposition qui

donne à l'apprentissage profond son nom, la profondeur est créée en utilisant plusieurs couches plutôt qu'une seule couche [43].

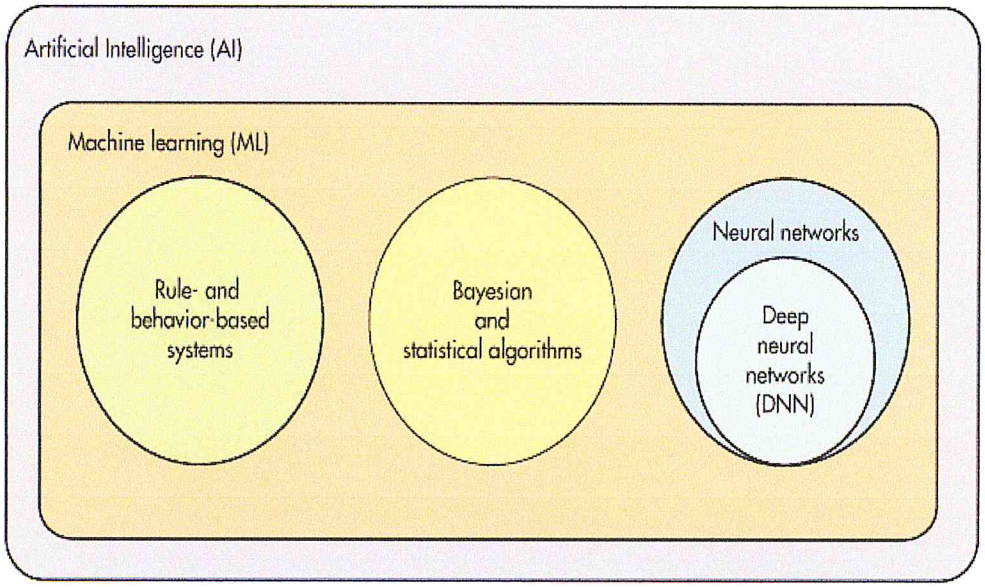


Figure 2. 7 : Différence entre intelligence artificiel deep learning et machine learning [42].

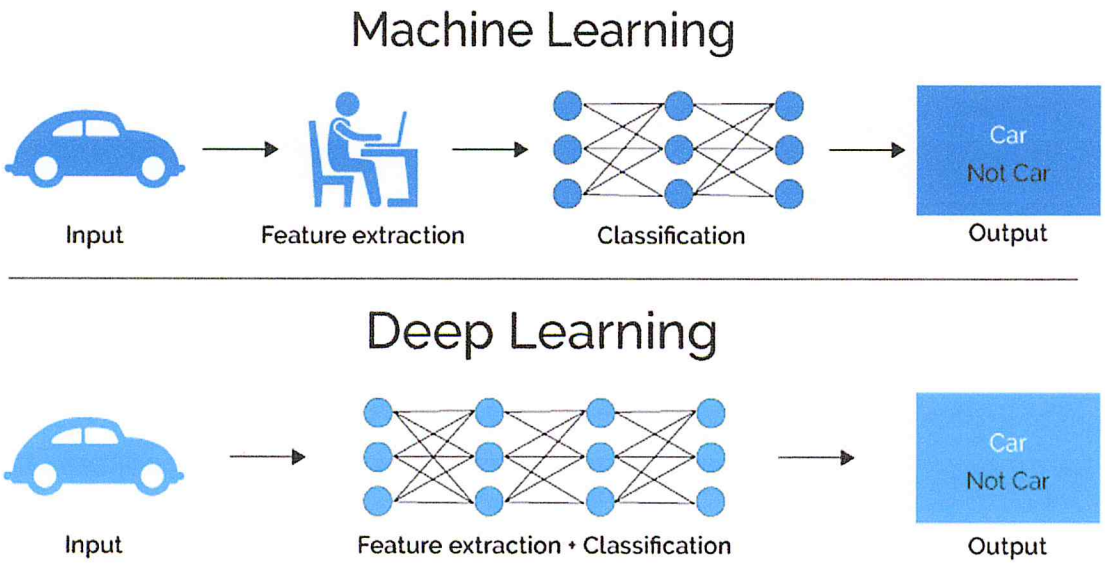


Figure 2. 8 : Différence entre le Deep Learning et le Machine Learning [44].

4.2. Historique

Les réseaux de neurones artificiels existent depuis longtemps. Leur application a été historiquement appelée cybernétique (années 1940-1960), connexionnisme (années 1980-1990), puis en tant qu'apprentissage profond vers 2006, lorsque les réseaux de neurones ont commencé à s'approfondir. Mais ce n'est que récemment que nous avons vraiment commencé à gratter la surface de leur plein potentiel [46].

Comme le décrit Andrej Karpathy, il y a généralement "quatre facteurs distincts qui freinent le deep learning [45] :

1. Puissance de calcul (la plus évidente : loi de Moore, GPU).
2. Données (sous une bonne forme, c'est-à-dire pas seulement quelque part sur Internet - par exemple ImageNet).
3. Algorithmes (recherche et idées, par exemple. Backpropagation, Gradient descent).
4. Infrastructure (software, Linux, Git, AWS, TensorFlow, café, theano etc.).

Au cours de la dernière décennie environ, le plein potentiel de l'apprentissage profond est finalement débloqué par les progrès réalisés dans les facteurs (1) et (2), ce qui a conduit à d'autres percées pour les facteurs (3) et (4) [45].

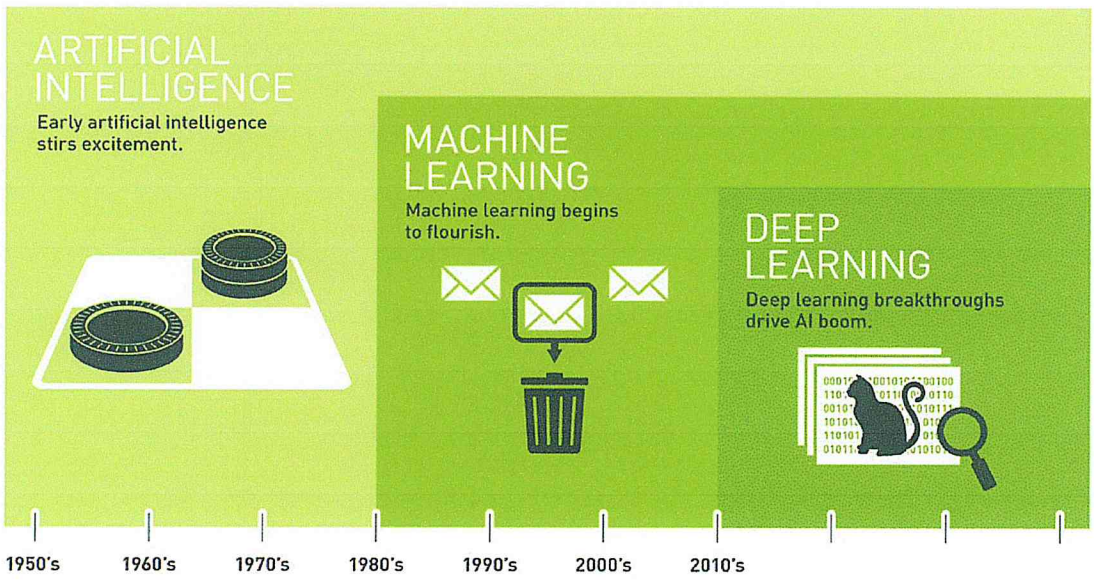


Figure 2. 9 : Historique de deep learning [47].

4.3. Domaines d'application

L'apprentissage profond est en train de transformer le monde dans pratiquement tous les domaines. Voici quelques exemples des choses incroyables que l'apprentissage en profondeur peut faire

- **Voitures auto-conduit**

  Les voitures qui s'auto-conduisent dépendent de l'apprentissage profond pour des tâches visuelles comme la compréhension des panneaux routiers, la détection des voies et la reconnaissance des obstacles [46].

  Drive.ai offrira un service d'auto-conduite pour utilisation publique à Frisco, Texas, à partir de juillet 2018 [51].

- **Traduction automatique**

  La traduction automatique existe depuis longtemps, mais l'apprentissage profond permet d'obtenir les meilleurs résultats dans deux domaines spécifiques [52] [53] [54] :

  - Traduction automatique de texte.
  - Traduction automatique des images.

- **Classification et détection d'objets dans les photos**

  Des résultats de haute qualité ont été obtenus sur des exemples de benchmarking de ce problème en utilisant de très grands réseaux de neurones convolutionnaires. Une percée dans ce problème par Alex Krizhevsky [55] des résultats sur le problème de classification d'ImageNet appelé AlexNet [56] [57].

4.4. Réseaux de neurones profonds

Les réseaux de neurones profonds sont principalement des paradigmes remarquables d'inspiration biologique qui permettent à n'importe quel ordinateur d'apprendre une large

gamme d'informations utiles et pertinentes simplement en observant différentes données. L'apprentissage profond et les réseaux de neurones fournissent actuellement les meilleurs exemples de la qualité des données. Ils fournissent également les meilleures solutions possibles à divers problèmes dans différents domaines, y compris le traitement du langage naturel, la reconnaissance de la parole et de l'image, ect [22].

Les réseaux de neurones sont conçus pour nous aider à classer et à regrouper une grande quantité de données. En d'autres termes, pensez aux réseaux de neurones comme couche de classification et de clustering qui sont placés au-dessus des données qui seront gérées et stockées. Les réseaux de neurones sont également conçus pour aider à regrouper diverses données non étiquetées en fonction de certaines similitudes existant dans différentes collections de données [26].

On peut également dire que les réseaux de neurones extraient diverses caractéristiques de données qui sont transmises à différents algorithmes pour une classification et un regroupement plus poussés. Il faut penser au réseau de neurones en tant que divers composants de la plus grande famille de modèles et d'applications d'apprentissage automatique. Les réseaux de neurones impliquent différents algorithmes conçus pour la régression des données, la classification des données et le renforcement de l'apprentissage profond [26].

Les techniques d'apprentissage en profondeur sont également capables de nous fournir une approximation de la fonction entre n'importe quelle entrée possible et n'importe quelle sortie potentielle. Nous devons juste supposer que les entrées et les sorties sont liées via une certaine corrélation [26].

Il y'a plusieurs type des réseaux de neurones profonds.

4.4.1. Réseaux de neurones convolutionnels

Les réseaux de neurones à convolution sont utilisés dans l'apprentissage en profondeur non supervisé. Ces techniques d'apprentissage étaient, en fait, le premier modèle artificiel utilisé pour la reconnaissance afin d'atteindre une grande performance dans diverses tâches [27]. Les réseaux de neurone à convolution sont créés afin de traiter diverses données qui sont sous la forme de tableaux numériques, par exemple, une image colorée qui est composée de

trois tableaux bidimensionnels qui contiennent également différentes intensités de pixel dans ces trois couches de couleur [28]. Derrière le réseau de neurone a convolution, il y a quatre idées fondamentales :

A. Étape de convolution

Le but principal de Convolution est d'extraire des caractéristiques de la donnée d'entrée. Par exemple une image en couleur qui contient 3 couches devra passer pas un traitement à convolution qui consiste à faire glisser un filtre ou bien le détecteur de caractéristiques (Feature detector) sur l'image d'origine pixel par pixel. On calcule la multiplication par éléments (entre les deux matrices) et on ajoute les sorties de multiplication pour obtenir l'entier final qui forme un seul élément de la matrice de sortie qui est appeler 'Convolved Feature' ou 'Feature Map' [28].

B. Non linéarité (Relu)

Une opération supplémentaire appelée ReLU est utilisée après chaque opération de convolution, ReLU signifie unité linéaire rectifiée et est une opération non linéaire.sa formule de sortie est : Sortie =Max (Zéro, Entré).

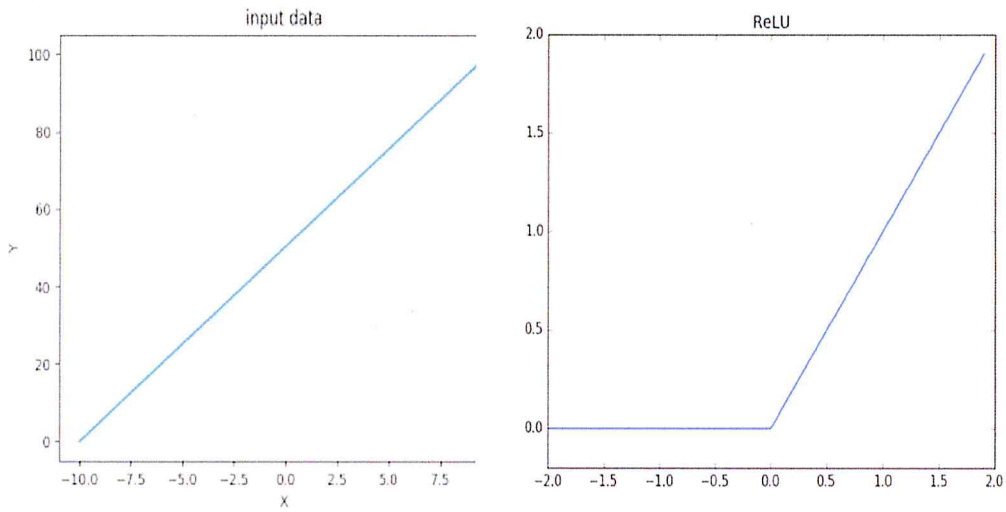


Figure 2. 10 : Relu Non-Linéarité [28].

ReLU est une opération appliquer par pixel et remplace toutes les valeurs de pixels négatifs dans la Feature map par zéro ().le but de ReLu est d'introduire la notion de non-linéarité,

puisque la plupart des données du monde réel que nous voudrions que notre convolutional network apprenne serait non-linéaire [28].

C. Couche Pooling

L'étape Pooling réduit la dimensionnalité de chaque Feature Map mais conserve les informations les plus importantes. Les deux forme les plus utiliser sont le 'Average Pooling' qui consiste à faire la moyen des valeurs de la zones, ou bien 'Max Pooling' qui prend l'élément le plus grand de la Feature Map. Le Pooling rend les représentations d'entrée plus petites et plus faciles à gérer. Il réduit aussi le nombre de paramètres et de calculs dans le réseau, contrôlant ainsi le sur-apprentissage [28].

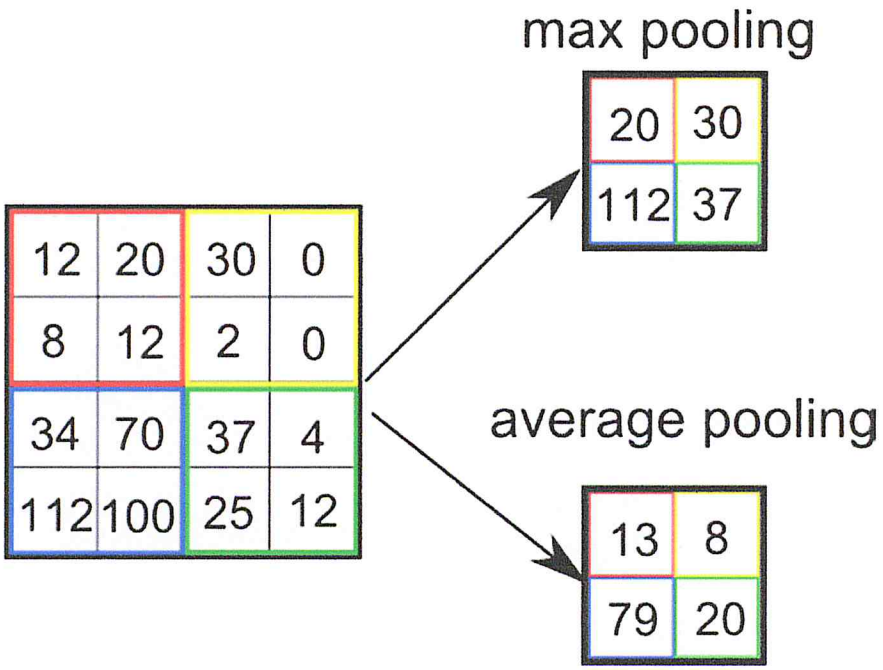


Figure 2. 11 : Étape Max/average Pooling [27].

D. Couche Fully Connected

La couche Fully Connected est configurée exactement comme son nom l'indique : elle est entièrement connectée à la sortie de la couche précédente. Une couche entièrement connectée prend tous les neurones de la couche précédente (qu'elle soit entièrement connectée, en pool ou convolutionnelle) et la connecte à chaque neurone qu'elle possède [28].

4.4.2. Réseaux de neurones récurrents (RNN)

L'idée général derrière les réseaux de neurones récurent est d'utiliser des informations séquentielles. Dans un réseau neuronal traditionnel, nous supposons que toutes les entrées (et sorties) sont indépendantes les unes des autres. Mais si on veut prédire le mot suivant dans une phrase, on doit savoir quels mots sont antérieurs. Les RNN sont appelés récurrents car ils effectuent la même tâche pour chaque élément d'une séquence, avec la sortie étant dépendante des calculs précédents [29]. Voici à quoi ressemble un RNN typique :

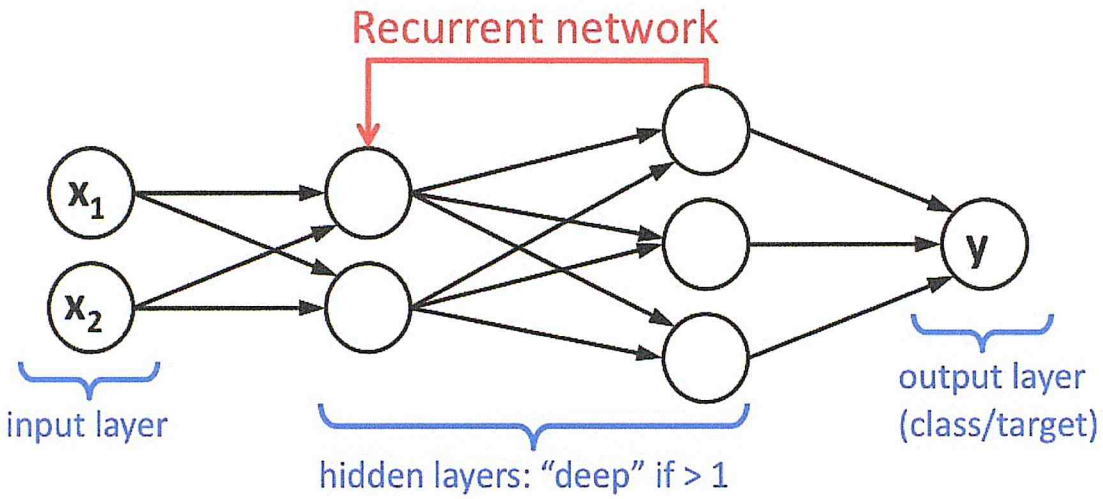


Figure 2. 12 : Architecture d'un réseau de neurones récurrents [41].

4.4.2.1. Réseaux récurrents à longue mémoire à court terme (LSTM)

Les LSTM sont inspirés des RNN, contrairement à eux, les LSTM implémentent spécifiquement Des "cellules de mémoire" qui ont la capacité de stocker de la mémoire pour de plus longues périodes de temps [29].

Les LSTM contiennent des informations en dehors du flux normal du réseau récurrent dans une cellule 'gated'. Les informations peuvent être stockées, écrites ou lues dans une cellule, comme les données de la mémoire d'un ordinateur. La cellule prend des décisions sur ce qu'il faut stocker, et quand autoriser les lectures, les écritures et les effacements, via des portes qui s'ouvrent et se ferment. Contrairement au stockage numérique sur ordinateur, ces portes sont analogiques, implémentés avec une multiplication par sigmoids, qui sont toutes dans la plage de 0-1 [30].

Ces portes agissent sur les signaux qu'elles reçoivent et, comme les nœuds du réseau de neurones, elles bloquent ou transmettent des informations basées sur leur force et leur importation, qu'ils filtrent avec leurs propres ensembles de poids. Ces poids, comme les poids qui modulent les états d'entrée et les états cachés, sont ajustés via le processus d'apprentissage des réseaux récurrents. C'est-à-dire que les cellules apprennent à autoriser les données à entrer, à quitter ou à être supprimées au cours du processus itératif consistant à faire des suppositions, des erreurs de rétropropagation et à ajuster les pondérations [30].

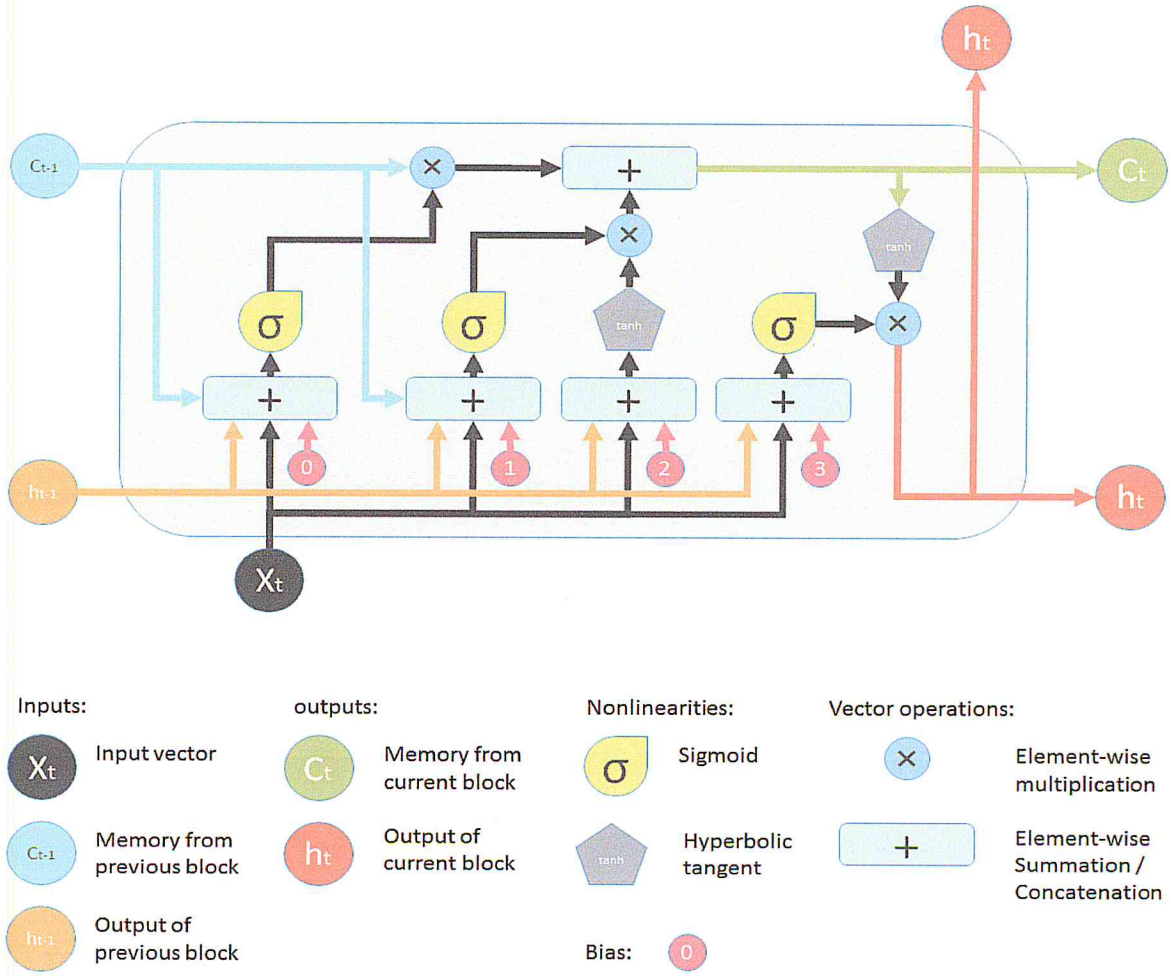


Figure 2. 13 : Architecture d'un LSTM [39].

4.4.2.2. Réseau récurrent à portes (Gated Recurrent Unit -GRU-)

Un réseau récurent a portes est une variation d'un réseau de neurones récurent, il utilise également des unités de contrôle pour moduler le flux d'informations à l'intérieur de l'unité, mais sans cellules de mémoire, les portes d'entrée et d'oubli fusionne pour devenir une

(update gate). Et enfin la porte de sortie est remplacée par une porte de réinitialisation (reset gate) [31].

LSTM, c'est que GRU a l'avantage d'être moins lourde en calculs car possédant moins de paramètres et d'équations, mais elle a des performances équivalentes au modèle LSTM de base. [31].

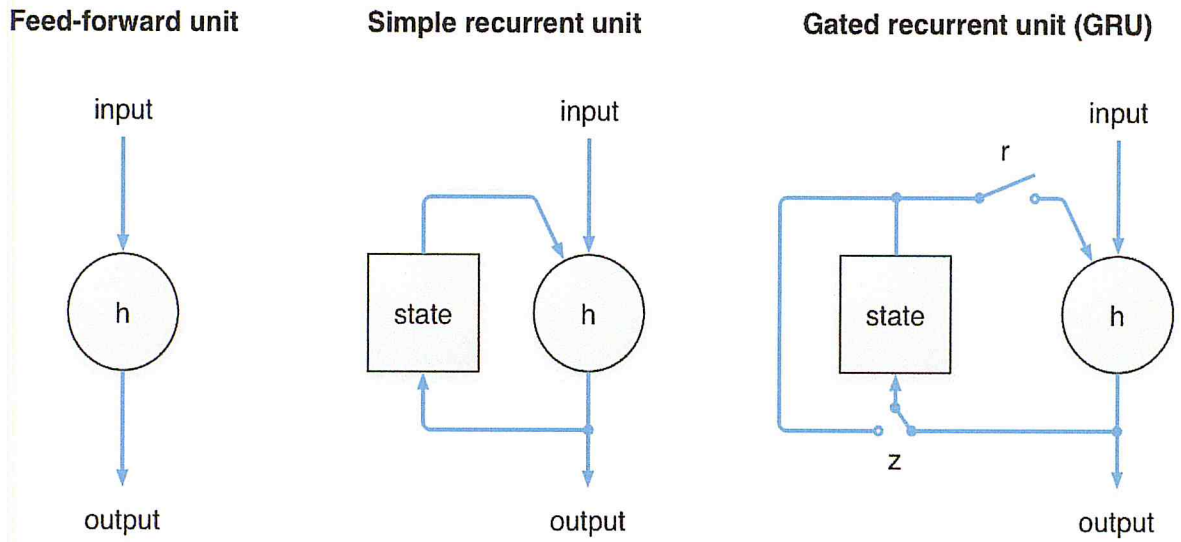


Figure 2. 14 : Réseau récurrent à portes [40].

4.4.3. Generative Adversarial Networks (GAN)

L'idée derrière les GAN est que nous avons deux réseaux, un générateur et un discriminateur, en concurrence entre eux. Le générateur génère de fausses données pour les transmettre au discriminateur. Le discriminateur voit également des données réelles et prédit si les données qu'il reçoit sont réelles ou fausses. Le générateur est entrainé pour tromper le discriminateur, il veut produire des données qui se rapprochent le plus possible des données réelles. Et le discriminateur est entraîné pour savoir quelles données sont vraies et lesquelles sont fausses. Ce qui finit par se produire, c'est que le générateur apprend à générer des données qui ne peuvent pas être distinguées des données réelles par rapport au discriminateur [32].

- **Le générateur**

  L'entrée dans le générateur est une série de nombres générés aléatoirement appelés échantillons latents. Une fois entraîné, le générateur peut produire des images à partir d'échantillons latents [33].

- **Le discriminateur**

  Le discriminateur est un classificateur entraîné en utilisant l'apprentissage supervisé. Il classe si une image est réelle (1) ou non (0) [33].

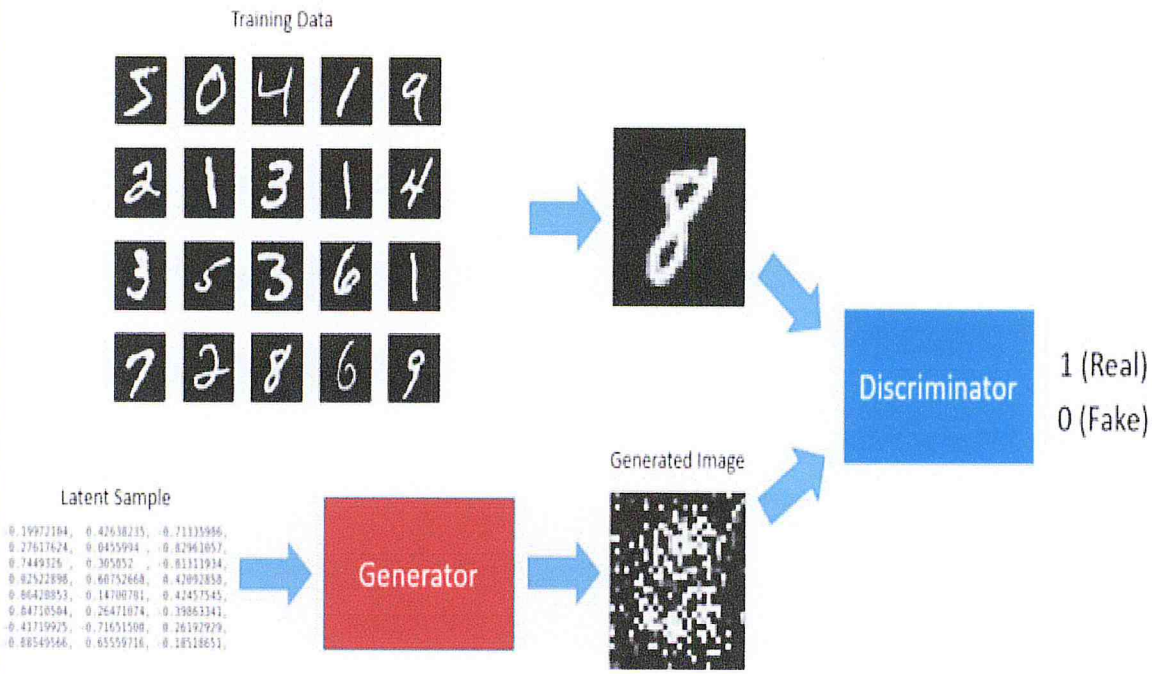


Figure 2. 15 : Generative Adversarial Networks [33].

4.4.4. Les Autoencoders

Un autoencoder est un algorithme de compression de données. C'est un réseau de neurones entraîné pour copier son entrée vers sa sortie. Une couche cachée décrit le code utilisé pour représenter l'entrée, il mappe l'entrée vers la sortie à l'aide d'un code de représentation compressé [70].

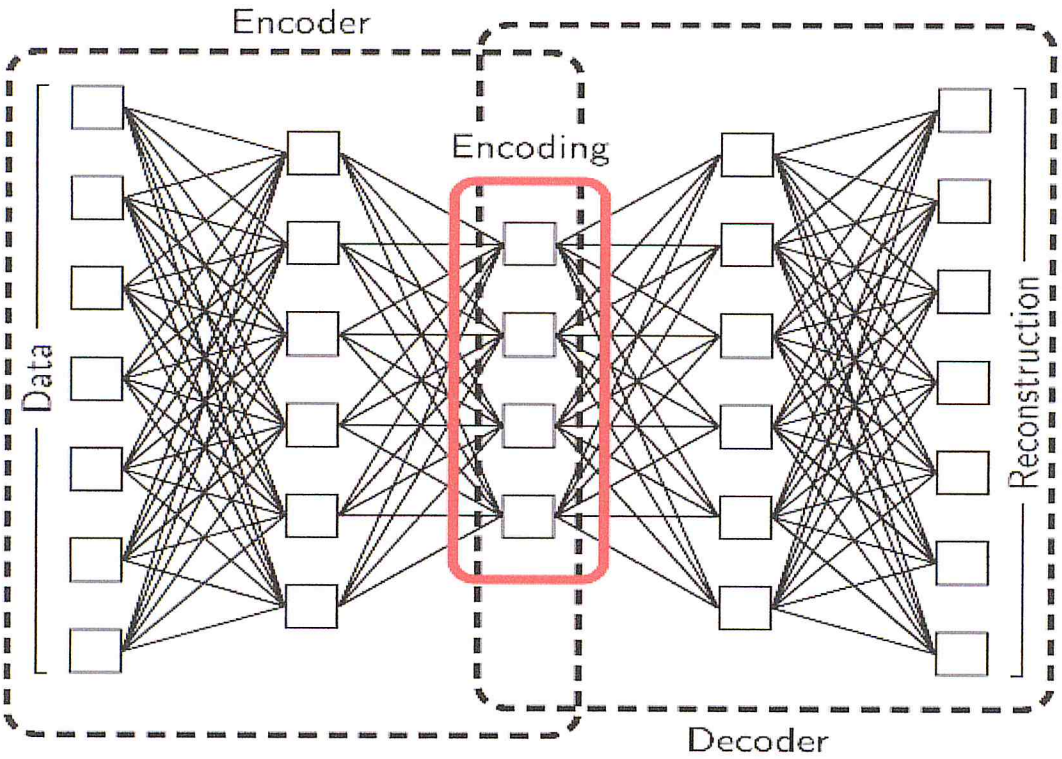


Figure 2.16 : Architecture d'un autoencoder.

Les couches successives détectent des motifs dans les données d'entrée et les utilisent pour générer une représentation encodée des données. Le décodeur correspondant prend des encodages et tente de reconstruire les entrées originales [69].

4.4.4.1 Différence entre les autoencoders et la compression des données

- **Les autoencoders sont spécifiques aux données :**
    - Ils ne peuvent comprimer que des données similaires à celles sur lesquelles ils ont été entraînés [70].
    - Font des suppositions générales sur les données, mais pas sur les types spécifiques exemple : .mp3 .png .mp4 [70].
    - Par exemple un autoencoder pour des images de chats aurait du mal à compresser des images d'arbres parce que les caractéristiques qu'il aurai apprises seraient spécifiques aux chats [70].
    -
- **Les autoencoders subissent des pertes :**

- Ce qui signifie que les sorties décompressées seront dégradées par rapport aux entrées originales [70].

- Les autoencoders apprennent des caractéristiques significatives sur les données.

4.4.4.2 Structure générale d'un autoencoder

Un autoencoder est un réseau de neurones qui est entraîné à essayer de copier son entrée vers sa sortie il a une couche cachée $\boldsymbol{h}$ qui décrit un code utilisé pour représenter l'entrée. Le réseau peut être considéré comme étant composé de deux parties [70] :

- Une fonction codeur $\boldsymbol{h} = f(\boldsymbol{x})$
- Décodeur qui produit une reconstruction $\boldsymbol{r} = g(\boldsymbol{h})$

Cette architecture est présentée dans la figure **4.4**. Si un autoencoder réussit à apprendre simplement à établir $g(f(\boldsymbol{x})) = \boldsymbol{x}$ partout, alors ce n'est pas particulièrement utile. Au lieu de cela, les autoencoders sont conçus pour être incapables d'apprendre à copier parfaitement. Habituellement, ils sont restreints d'une manière qui leur permet de ne copier qu'approximativement et de ne copier que les données d'entrée qui ressemblent aux données de leur entraînement. Comme le modèle est forcé de prioriser les aspects de l'entrée qui doivent être copiés, il apprend souvent les propriétés utiles sur les données [70].

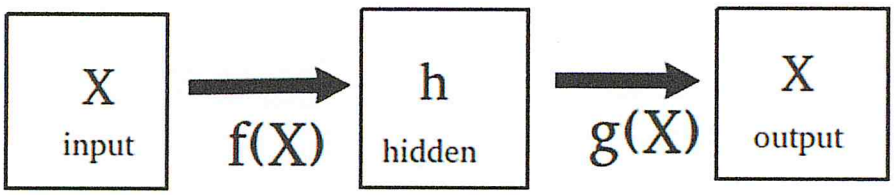


Figure 2. 17 : La structure générale d'un autoencoder. Le mappage d'une entrée x à une sortie. (Appelée reconstruction) r par une représentation interne ou un code h. L'autoencoder a deux composantes : le codeur f (mappage de x à h) et le décodeur g (mappage de h à r)

4.4.4.3 variantes de l'autoencoder

4.4.4.3.1. Vanilla Autoencoders

Les autoencoders sont un type spécifique de réseaux de neurone, où l'entrée est identique à la sortie. Ils compressent l'entrée en un code en dimensions inférieures et reconstruisent ensuite la sortie à partir de cette représentation. Le code est un "résumé" ou "compression" compact de l'entrée, également appelé représentation d'espace latent.

Un autoencoder se compose de 3 composants : encoder, code et décoder. L'encoder compresse l'entrée et produit le code, le décoder reconstruit ensuite l'entrée en utilisant uniquement ce code [34].

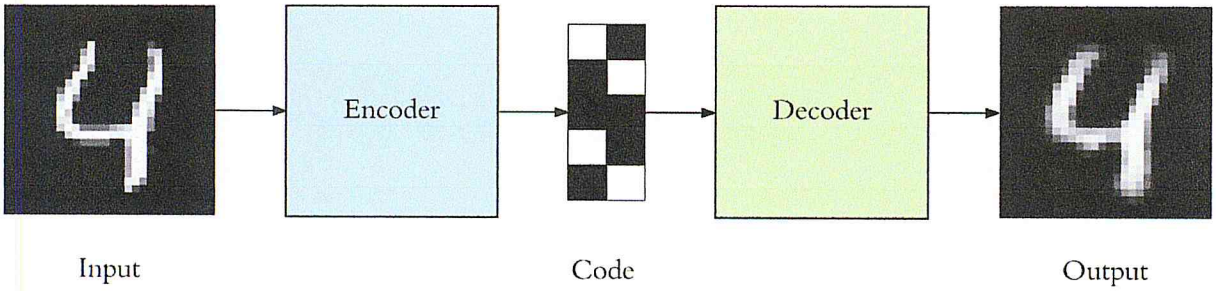


Figure 2. 18 : Architecture d'un vanilla Autoencoder [34].

4.4.4.3.2. Variational Autoencoders (VAE)

Les Variational Autoencoders (VAE) ont une propriété fondamentalement unique qui les sépare des vanilla Autoencoders, et c'est cette propriété qui les rend si utiles pour la modélisation générative : leurs espaces latents sont, de par leur conception, continus, permettant un échantillonnage et une interpolation aléatoires faciles [35].

Il atteint cet objectif en faisant quelque chose qui semble d'abord assez surprenant : rendre son codeur non sorti d'un vecteur de codage de taille n, mais plutôt, sortie de deux vecteurs de taille n : un vecteur de moyens, μ et un autre vecteur d'écarts-types, σ [35].

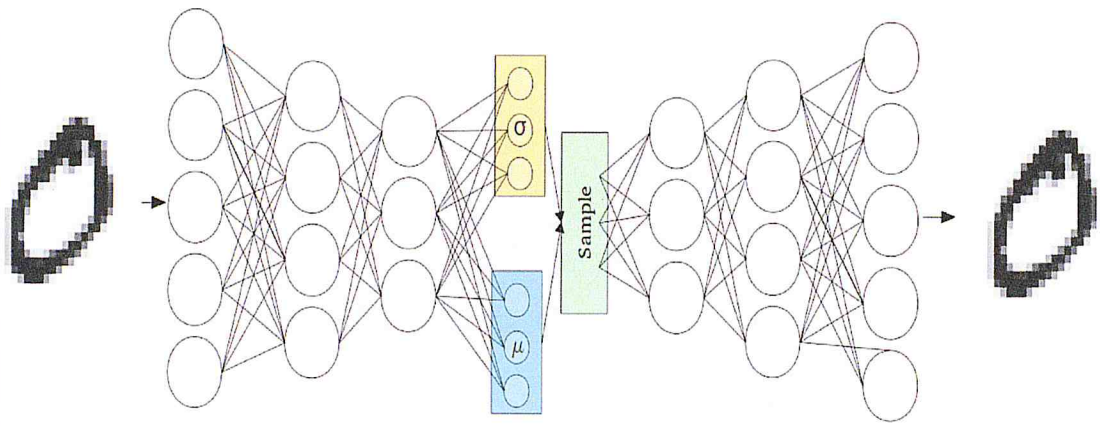


Figure 2. 19 : Exemple d'un autoencoder [35].

4.4.4.3.3 Denoising Autoencoders

Les autoencoders automatiques sont des réseaux de neurones qui sont couramment utilisés pour la sélection et l'extraction de caractéristiques. Cependant, lorsqu'il y a plus de nœuds dans la couche cachée qu'il n'y a d'entrées, le réseau risque d'apprendre la fonction dite "Identity Function", également appelée "Null Function", ce qui signifie que la sortie est égale à l'entrée, marquant l'Autoencodeur inutile [36].

Denoising Autoencoders résoudre ce problème en corrompant les données à dessein en tournant au hasard certaines des valeurs d'entrée à zéro. En général, le pourcentage de nœuds d'entrée qui sont mis à zéro est d'environ 50 %. D'autres sources suggèrent un compte plus bas, comme 30%. Cela dépend de la quantité de données et des nœuds d'entrée dont vous disposez [36].

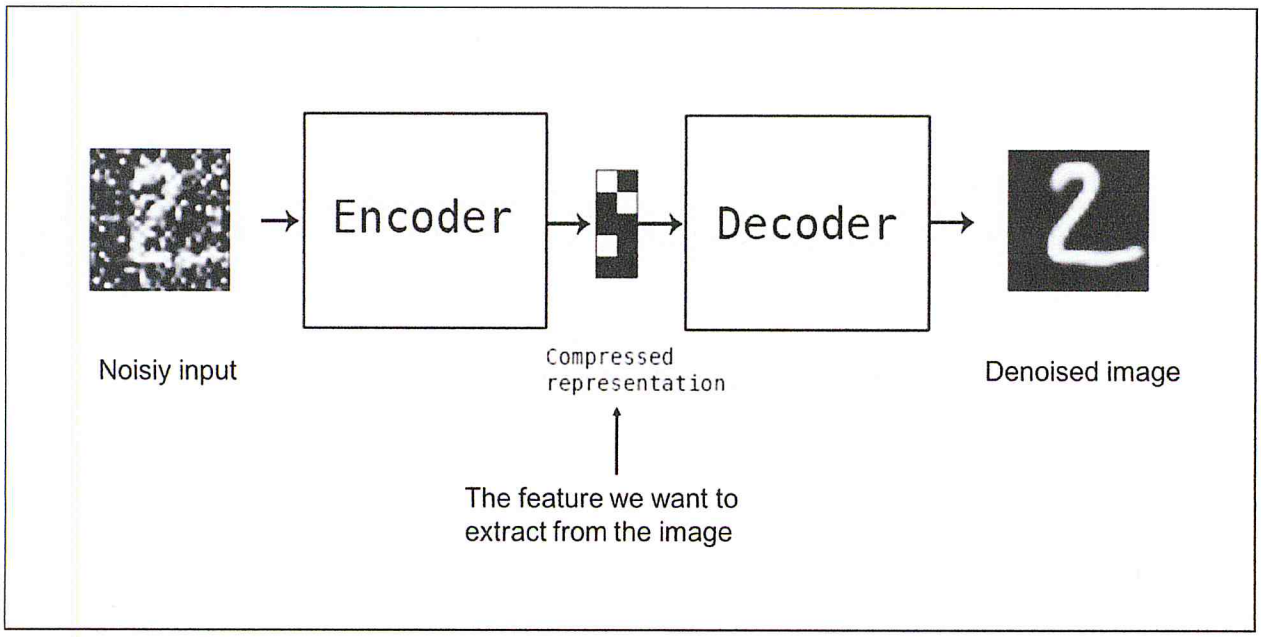


Figure 2. 20 : Denoising Autoencoder [37].

5. Concepts de deep learning

- **Le gradient**

  Le gradient est la dérivée partielle d'une fonction qui prend plusieurs vecteurs et fournit une valeur unique (les fonctions de coût dans les réseaux de neurones). Si nous augmentons les variables d'entrée (l'entrée et les poids de réseau) le gradient nous

indiquera dans quelle direction aller sur le graphe pour augmenter notre sortie. Puisque nous souhaitons diminuer notre fonction de coût nous utilisons le gradient et allons dans la direction opposée [81].

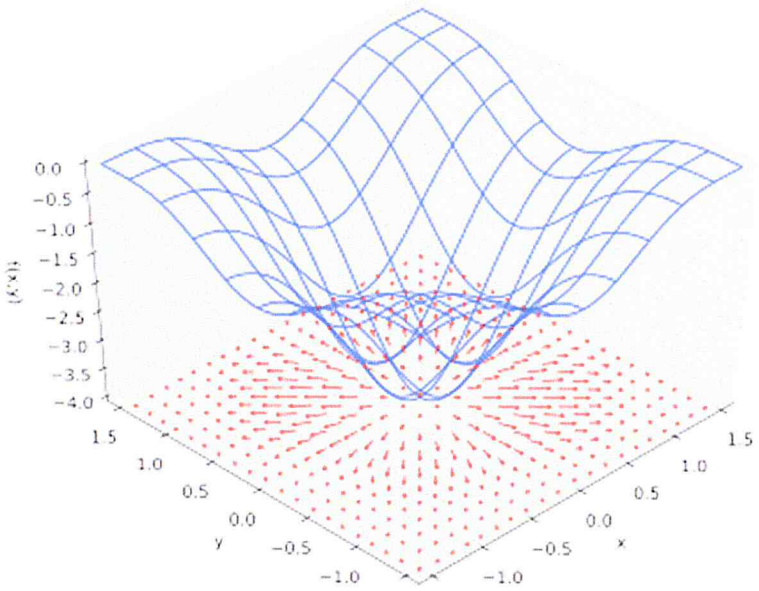


Figure 2. 21 : Visualisation du gradient. Les flèches rouges représentent le gradient de la fonction tracée en bleu [81].

- **Backpropagation (Rétropropagation)**

  La rétropropagation est une méthode efficace pour calculer les gradients dans les graphes de computations, comme les réseaux de neurones. Ce n'est pas une méthode d'apprentissage, mais plutôt une forme de calcul qui est souvent utilisée dans les méthodes d'apprentissage. Il s'agit en fait d'une simple implémentation de la règle de la chaîne des dérivés, celle-ci donne simplement la possibilité de calculer toutes les dérivées partielles requises dans un temps linéaire en termes de taille du Graphe, alors que les approches naïves le calculer de gradient se développeraient exponentiellement par rapport à la profondeur du réseau [83].

- **Vanishing Gradient (Disparition des gradients)**

  Lorsque nous faisons la rétropropagation, c'est-à-dire en reculant dans le réseau et en calculant les gradients de la fonction de coût (erreur) par rapport aux poids, les

gradients tendent à devenir de plus en plus petits à mesure que nous continuons à reculer dans le réseau. Cela signifie que les neurones des couches antérieures apprennent très lentement par rapport aux neurones des couches ultérieures de la Hiérarchie. Les couches antérieures du réseau sont les plus lentes à s'entraîner.

C'est donc ce que le problème de la disparition de gradient fait à notre modèle. Le processus d'entrainement prend trop de temps et la précision de prédiction du modèle diminuera [84].

- **Gradient Descent**

  C'est un algorithme d'optimisation qui vise à minimiser certaines fonctions de coût en fonction du gradient de cette fonction. Des itérations successives sont employées pour s'approcher progressivement d'un minimum local ou global de la fonction de coût. La figure ci-dessous montre un exemple du « **Gradient descent**» fonctionnant dans une seule dimension [82].

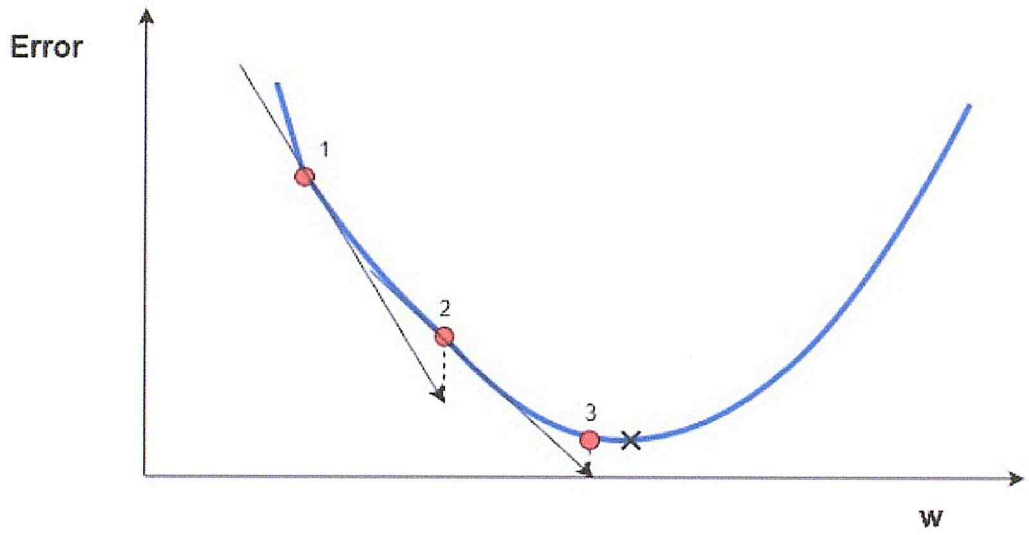


Figure 2. 22 : Visualisation du 'gradient descent' en fonctionnement [82].

- **Overfitting (Sur apprentissage)**

  Le mot overfitting fait référence à un modèle qui apprend par cœur les données d'entraînement. Au lieu d'apprendre la distribution générale des données, le modèle apprend la sortie attendue pour chaque point de données et c'est pour cette raison que le modèle ne pourra pas prédire par rapport à des données généralisées

  La difficulté réside dans le fait qu'à première vue, il peut sembler que le modèle fonctionne correctement parce qu'il comporte une très petite erreur sur les données

d'entraînement. Cependant, dès qu'on lui demande de prédire de nouveaux points de données, il va échouer [85].

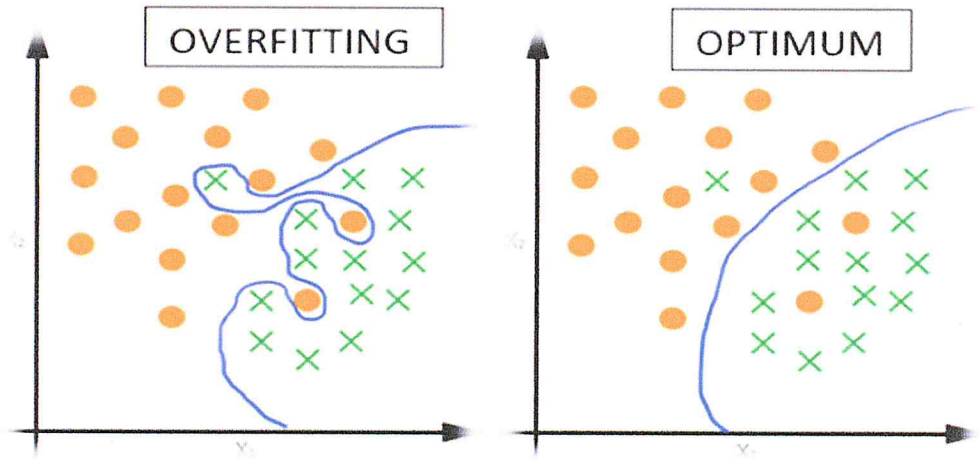


Figure 2. 23 : Exemple d'overfitting [86].

6. Conclusion

Dans ce chapitre nous avons introduit l'apprentissage et exposé la différence entre les réseaux de neurones classiques et l'apprentissage profond, nous avons décrit les différentes architectures des réseaux de neurones profonds et leur principe d'utilisation, pour pouvoir expliquer dans le chapitre suivant le choix de la structure utilisé.

Chapitre III

Extraction des motifs fréquents basée sur les réseaux de neurones artificiels

1. Introduction

Contrairement à d'autres sujets de recherche du Deep Learning et de Data Mining, particulièrement l'extraction des itemsets fréquents, dans lesquels la littérature est si vaste, le manque de travaux de recherche est une propriété du sujet qu'on va l'aborder qui est l'extraction des itemsets fréquents en utilisant le Deep Learning. Néanmoins, un résumé de la petite littérature disponible qui parle de quelques travaux un peu proches de ce sens est présenté dans ce chapitre.

2. Extraction des motifs fréquents basées sur les réseaux de neurones artificiels

À notre connaissance, Sallans est le premier à explorer l'utilisation des réseaux de neurones pour l'extraction des règles d'association [58]. Dans ce travail, il a utilisé les techniques non supervisées d'Analyse Factorielle et un Mélange de modèles gaussiens pour son étude. Ce travail a lié les réseaux de neurones artificiel et règles d'association en raison du fait que les règles de classification peuvent être générées à partir des ANNs (Artificial neural network) pour la classification. Les séries de transactions ont été simulées et basées sur certains modèles sous-jacents, ce qui pourrait ou ne pourrait pas être corrélé, et avec l'ajout d'un peu de bruit.

La conclusion de cette étude a rapporté que si le modèle AF ne pouvait jamais converger avec les données de RA, et qu'aucune raison claire n'a été trouvée pour un tel comportement, le réseau MGM a montré qu'il est capable d'apprendre les modèles de graines utilisés pour générer les transactions avec bruit.

La littérature fait plus référence à l'utilisation d'un réseau de neurone pour la classification des règles d'association que l'extraction des itemsets.

Par exemple Gupta [60] dans son article présente une nouvelle métrique de distance pour classer les règles d'association. Basé sur cette métrique de distance, il propose une nouvelle technique de clustering. Il intègre les distances en utilisant la mise à l'échelle multidimensionnelle et regroupe les points résultants dans un espace euclidien en utilisant un self-organizing map (qui est un type de réseau de neurone artificiel) [60].

Au final même si son travail consistait à proposer une nouvelle métrique de distance pour regrouper les règles d'association et non l'extraction des règles, l'utilisation d'un réseau de neurone (SOM) fait partie de la méthodologie de regroupement proposée.

2.1 Feed Forward Neural Network Algorithm pour l'extraction des motifs fréquents

En 2010 *Amit et Bhagat* [61] ont proposé un algorithme appelé Apriori-Feed Forward (AFF). Cet algorithme a pour but de combiner les avantages de l'algorithme Apriori et de la structure du réseau neuronal.

L'approche qu'ils ont proposée commence par la préparation du réseau de neurone en fonction du nombre maximal d'éléments présents dans l'ensemble de données.et ensuite suivie d'une étape d'analyse ; la première transaction de l'ensemble des données est analysée pour trouver les élément fréquents F1, puis le réseau de neurone est mis à jour pour trouver les F2, F3 et ainsi de suite [61]. Enfin, le réseau neuronal construit est exploité par l'algorithme Apriori-FeedForward.

Dans cet algorithme, un réseau neuronal Feed Forward complet est préparé en fonction du nombre maximum d'éléments présents dans les ensembles de données. La première couche du réseau constitue l'item fréquent 1, la deuxième couche représente les combinaisons de 2 items sets fréquent jusqu'à l'obtention d'une couche finale qui est un nœud unique comprenant tous les éléments présents dans les ensembles de données. Chaque couche n + 1 est une combinaison de l'élément n par rapport à tous les autres éléments présents dans cette couche.la figure ci-dessous représente la structure de données des nœuds dans ce réseau [61].

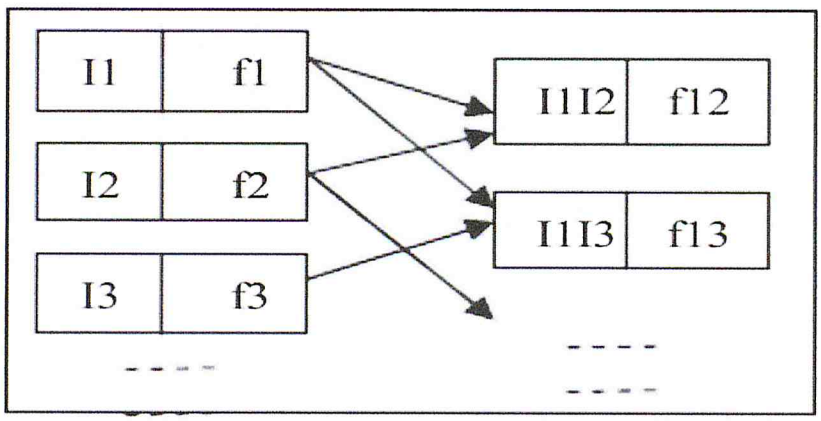


Figure 3. 1 : Structure de données des nœuds dans le réseau neuronal FeedForward [61].

2.2 Mining Frequent Itemsets without Candidate Generation using Optical Neural Network

Quant à Divya Bhatnagar [62], ils proposent une technique d'extraction d'ensemble d'items fréquents dans de grandes bases de données en utilisant un modèle de réseau de neurones physique. Ils se focus sur la façon dont le modèle peut être utile pour générer des modèles fréquents pour diverses applications. L'objectif de cet article est de trouver un moyen rapide, polyvalent et adaptatif en utilisant un réseau neuronal optique (ONN) [62].

Dans les réseaux de neurones optiques, les neurones sont interconnectés avec des faisceaux lumineux. Aucune isolation n'est requise entre les chemins de signaux. Les rayons lumineux passent entre eux sans entrelacement. La densité du trajet de transmission n'est limitée que par l'espacement des sources de lumière, l'effet de divergence et l'espacement des détecteurs. Par conséquence, tous les chemins de signaux fonctionnent simultanément, ce qui se traduit par un vrai débit de données [63]. Les forces des poids sont stockées dans des hologrammes à haute densité et peuvent être modifiés pendant l'opération pour produire un système entièrement adaptatif [63].

Comme de nombreux k-itemsets doivent être extraits en parallèle, plusieurs neurones sont nécessaires pour former une couche, donc il établit un réseau de neurones à propagation en avant à S neurones, ou chacune des entrées R est connectée à chacun des neurones et la matrice de poids à S lignes [62].

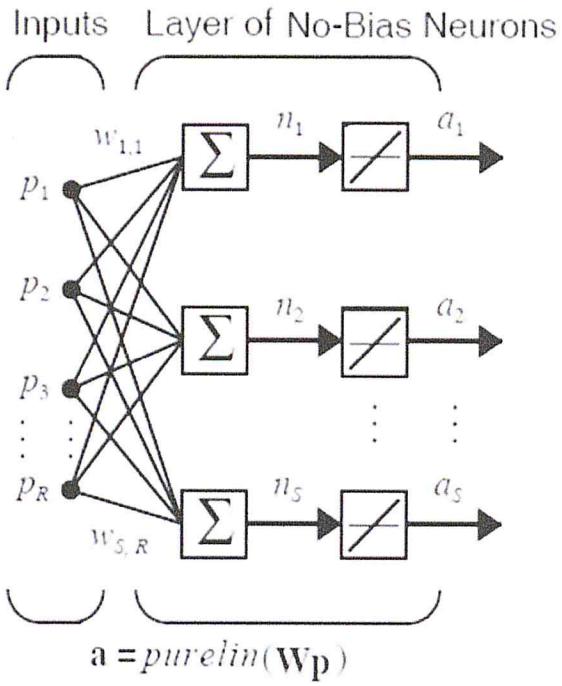


Figure 3. 2 : Une architecture monocouche d'un réseau homogène a propagation à l'avant [62].

Chaque élément du vecteur d'entrée p est connecté à chaque neurone à travers la matrice de poids W. Chaque neurone a une fonction de transfert et une sortie a_i.La différence par rapport à d'autre réseau a propagation à l'avant est que tous les neurones ont la même fonction de transfert. De tels réseaux neuronaux sont communément appelé homogènes et ils conviennent le mieux aux réalisations optoélectroniques [64]. Afin de trouver le support des items, la présence d'un élément dans toutes les transactions doit être résumée. La présence ou l'absence d'un élément dans une transaction est représentée par 1 ou 0 qui est stocké en tant que poids dans la matrice de pondération. En optique les valeurs de lumière peuvent être numérisé en binaires [65]. Dans ce modèle, le vecteur d'entrée P est constitué de tous les 1, c.-à-d. P= [1 1 1 1 .....1] comme 1 multiplié par 1 ou 0 donne le même résultat indiquant la présence ou l'absence d'un item.

Notre compréhension du sujet n'est pas précise, du fait que ce soit du domaine électro-optique et qui contient beaucoup d'information relative à des interactions avec des composants électroniques. Nous essayons de résumé l'article sans rentrer dans les détaille concernant ces composant.

Les trois composants matériels les plus élémentaires d'un système de traitement d'informations optiques sont une source, un modulateur et un détecteur [62]. Les lasers sont une source d'illumination les plus communément utilisé car ils sont mathématiquement plus simples à comprendre [66]. Les fonctions logiques exécutées sont la multiplication et l'addition qui peuvent être implémentées en utilisant des modulateurs de lumière spatiale (Spatial Light Modulators -**SLMs**-) [62]. Les SLM comprennent les films photographiques et des dispositifs électro-optiques, ce sont des masques optiques contrôlés à l'aide d'électrodes [67].

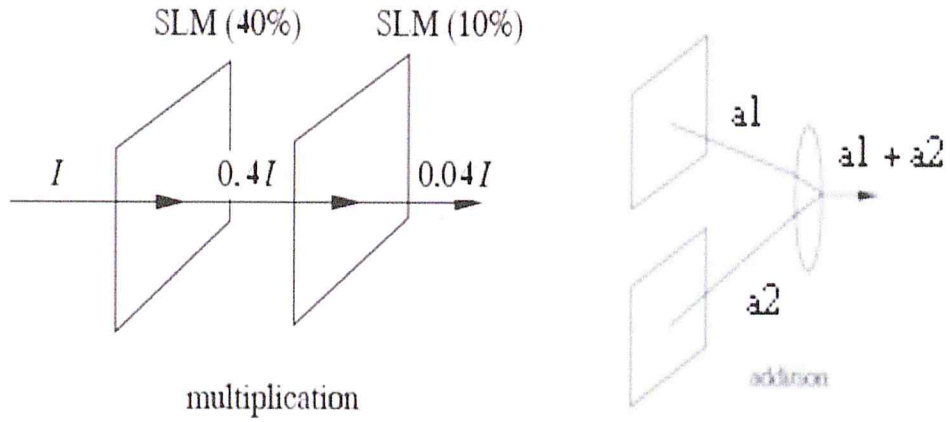


Figure 3. 3 : Implémentations optique des multiplications et addiction [62].

La multiplication a lieu grâce un équilibrage du voltage appliqué qui rend le SLM plus sombre ou réduit la quantité de lumière projetée sur un SLM. L'addition de signaux optiques est réalisée en réduisant deux signaux lumineux à un en utilisant des lentilles, des prismes ou d'autres dispositifs capables de traiter les rayons lumineux. Pour effectuer une multiplication de vecteur matriciel, ils utilisent un SLM qui est divisé en champs R * S [62]. L'obscurité dans chaque champ est maintenue constante à sa valeur numérique, c'est-à-dire, 1 ou 0 selon les poids appliqués. Le vecteur d'entrée est projeté sur le SLM.

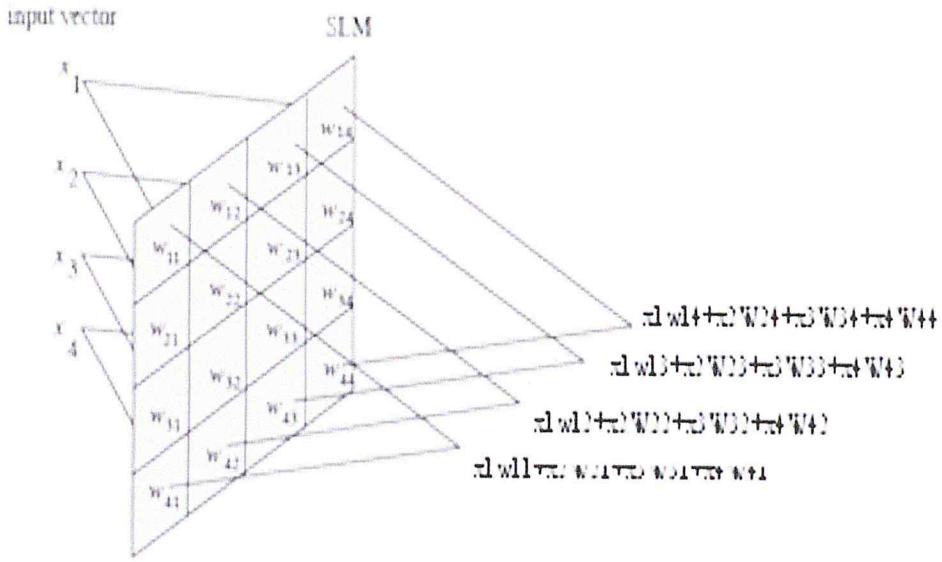


Figure 3. 4 : Multiplication matrice-vecteur avec masque SLM [62].

Si la lumière passe à travers de nombreux SLM disposés en pile, une chaîne de multiplications peut être calculée instantanément et l'intensité de la lumière est déterminée proportionnellement au produit de la ration d'obscurité des SLMs. Afin d'obtenir le nombre de support de deux ou plusieurs itemsets, plusieurs SLM sont disposés et la sortie est obtenue sous la forme de lumière sortante [67]. Le seuil peut être calculé et implémenté électroniquement ou optiquement [65].

La procédure qu'ils proposent se fait en trois étapes. Tout d'abord l'installation et la configuration du réseau de neurone optique qui consiste à utiliser une base de données de transaction à extraire avec **m** transactions et **n** éléments, puis la création d'une matrice de poids **Mm * n,** qui a **m** lignes et **n** colonnes. Ils scannent la base de données. Si l'item **Ij** est dans la transaction **Ti** le poids **Wij** est mis à "1", sinon "0". Les cellules de pondération contenant un 0 ont un masque haute densité à travers lequel la lumière ne passe pas, et les cellules contenant un 1 ont un masque transparent à travers lequel la lumière peut passer facilement [68].

La deuxième étape consiste de trouver le support de tous les items. Lorsque la lumière tombe sur les lignes de la matrice et que cette lumière traverse les cellules contenant le poids 1 elle donne forcément un 1 (1*1) et devient un 1 correspondent comme la sortie à ces cellules. Toutes ces sorties individuelles sont ensuite accumulées par des photo-détecteur comme les supports de chaque itemset. Le support ainsi obtenu est ensuite stocké

électroniquement et comparé au support minimum. Si le support de l'item> = le min-sup, le tem est fréquent.

Exemple, si on prend une base de données **D**, le masque de poids qui lui correspond serai **W** qui est générée à partir des transactions **D** comme montrer dans la figure 5 ci-dessous [62].

D

Tid	Items
1	I1 I3 I4 I6
2	I2 I3 I5 I6
3	I1 I2 I3 I5 I6
4	I2 I5 I6
5	I1 I2 I3 I5 I6

W

I1	I2	I3	I4	I5	I6
1	0	1	1	0	1
0	1	1	0	1	1
1	1	1	0	1	1
0	1	0	0	1	1
1	1	1	0	1	1

Figure 3. 5 : Passage de base de données à une matrice de poids [62]

Le vecteur d'entrée [11111] est multiplié par la matrice de poids et la sortie correspond à l'existence ou non de l'item dans les transactions. Si le support minimum dans cet exemple est fixé à 3 les itemsets fréquent dans F1 serai {I1}, {I2}, {I3}, {I5}, {I6} (Figure 6). La procédure est répétée jusqu'à l'extraction de tous les itemset fréquents [62].

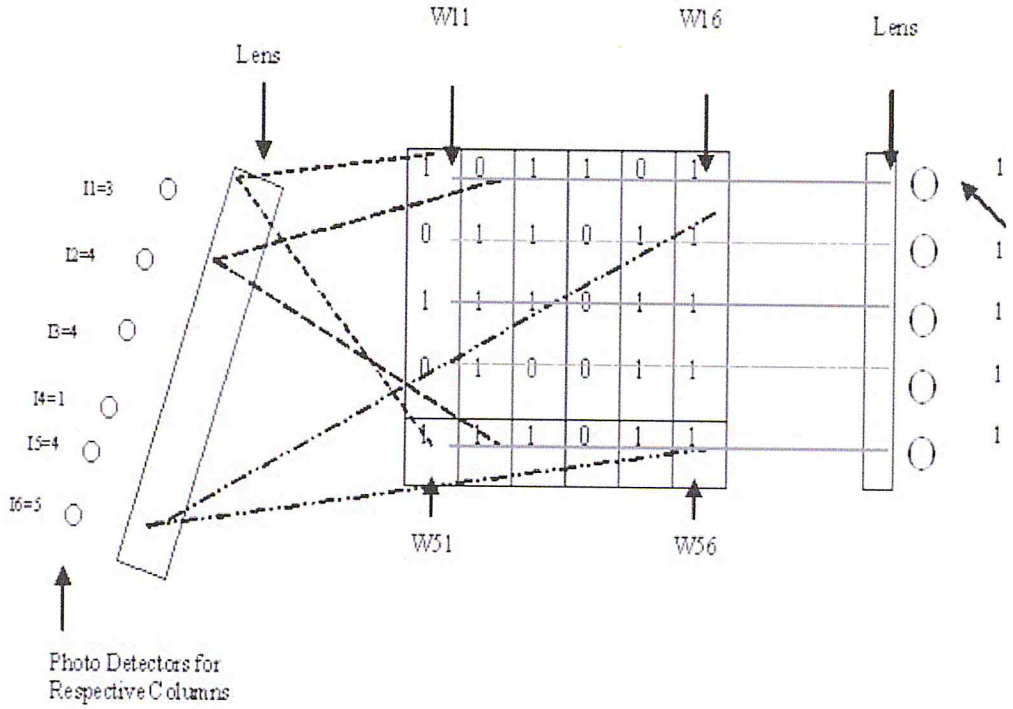


Figure 3. 6 : Model ONN pour l'extraction des itemsets fréquents F1 [62].

Au final Divya Bhatnagar et al proposent une technique d'exploration de modèle basée sur un modèle de réseau de neurones optiques, ça ne produit pas d'ensembles d'items candidats, mais utilise un réseau de neurone optique pour découvrir des ensembles d'items fréquents [62].

3. Discussion

La littérature fait plus référence à l'utilisation d'un réseau de neurone pour la classification des règles d'association que l'extraction des itemsets.

- Pour le réseau MGM de Sallan il a montré qu'il est capable d'apprendre les modèles de graines utilisés pour générer les transactions avec bruit, et que la pire performance du MGM a été montrer quand les modèles étaient fortement corrélés et les données contenait du bruit, tandis que la meilleure performance résultait de modèles non corrélés et de données propres.
- Quant a Gupta, au final même si son travail consistait à proposer une nouvelle métrique de distance pour regrouper les règles d'association et non l'extraction des règle, l'utilisation d'un réseau de neurone (SOM) fait partie de la méthodologie de regroupement proposée.

- Pour la methode d'Apriori-Feedforward proposé par Amit Bhagat, l'absence d'expériences est d'indication plus détaillé sur la façon d'interpréter le réseau de neurones proposer fait qu'il est difficile de considérer ce travail comme une solution formelle au problème posé.

- Pour le réseau de neurones optique de Divya Bhatnagar. Le défaut de cette technique c'est que l'utilisation de multiple masque peut causer une surcharge, et que le résultat est lié aux composants électroniques disponibles en plus d'être difficile à implémenté [62]. Il a quand même l'avantage qu'il soit très rapide.

Enfin, en tenant compte de nos objectifs et des recherches générales en cours sur le sujet. Nous croyons fermement qu'il est pratique de définir que cette recherche diffère des approches résumées ci-dessus car notre travail ne se concentrera pas sur l'utilisation d'un réseau de neurone classique comme par exemple le FFN de Amit Bhagat Notre travail se concentre davantage sur la conception d'une architecture de réseau de neurone profond capable d'extraire les itemsets fréquents.

4. Conclusion

Dans ce chapitre, la littérature impliquée dans nos principaux objectifs est quasi inexistante. Nous avons essayé tant bien que mal de trouvé des articles qui sont en relation avec notre objectif. Nous avons énoncé certains travaux qui utilise des réseaux de neurone classique pour tenter d'analysé et d'extraire les itemsets fréquents, mais il n'y a aucun article ou thèse qui mentionne la création d'un modèle de Deep Learning pour l'extraction des items fréquents.

Chapitre IV

Approche Proposé

1. Introduction

Dans ce chapitre, nous proposerons une nouvelle méthode pour extraire les itemsets fréquents au lieu d'utiliser des algorithmes classiques tels qu'apriori. Nous utilisons un réseau de neurones profond appelé Autoencoder. En effet, dans ce chapitre on va présenter en détails l'approche proposée.

2. Présentation de l'approche proposée

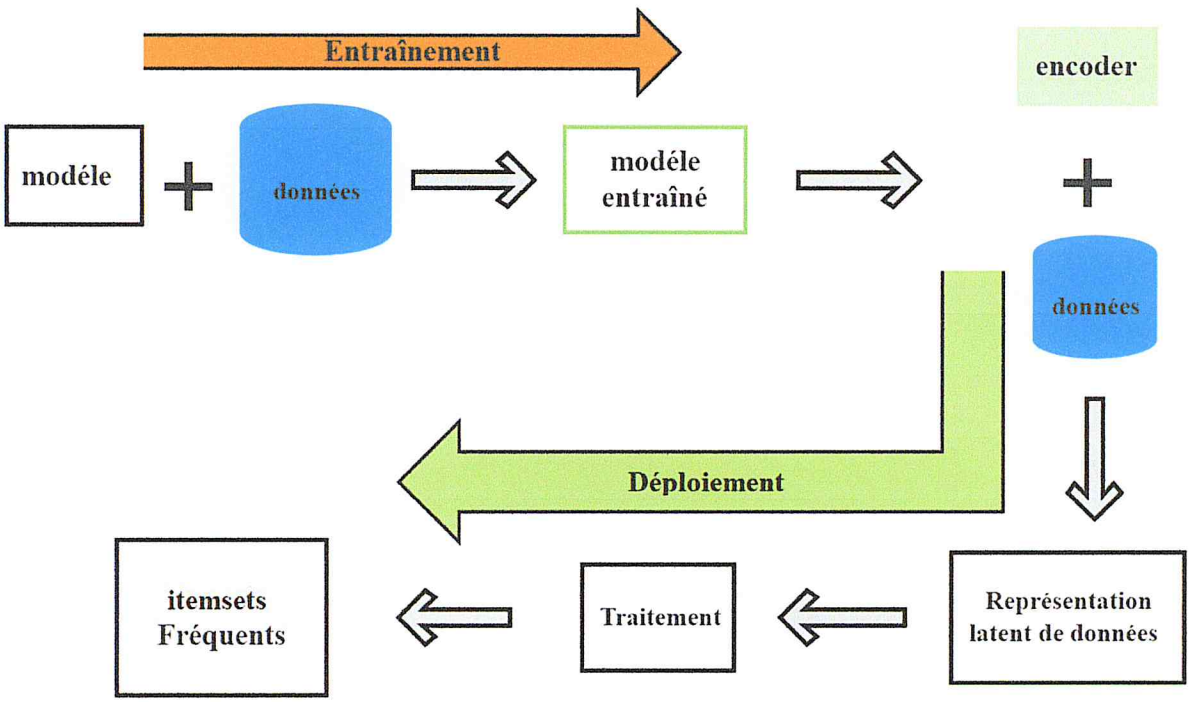


Figure 4. 1 : Schéma global de l'approche proposée.

Dans la construction de notre système nous passons par deux phases, la première phase est l`entraînement du modèle et la deuxième phase est le déploiement et l'utilisation du modèle pour extraire les itemsets fréquents.

2.1 Modèle de réseau profond adopté

Pour notre approche nous allons travailler avec l'autoencoder ou nous utiliser juste le premier partie de notre réseau (l'encoder) pour obtenir la représentation latent des transactions on va utiliser le deuxième partie parce que on n'a pas besoin de reconstruire les transactions. Pour plus de détails sur l'autoencoder consulter le chapitre 2 (architectures de deep learning).

2.1 Les motivations pour utiliser l'autoencoder

- **Les étiquettes des transactions**

  L'étape la plus essentielle dans l'entrainement des réseaux de neurones est le calcul de la fonction de coût, qui est calculé de la façon suivante (étiquette de transaction – sortie produite par le réseau), donc on ne peut pas entrainer un réseau si on n'a pas ses étiquette. L'autoencoder c'est un cas particulier des réseaux de neurones où nous n'avons pas besoin des étiquettes pour l'entrainer.

- **Les propriétés de l'espace latent**

  L'espace latent des autoencoders possède la propriété de regrouper les transactions identiques donc on peut utiliser cette propriété pour grouper les transactions et calculer les supports de ces transactions on divisant le nombre de transactions dans le groupe par le nombre total des transactions dans la base de données.

2.2 Raisons pour lesquelles nous ne pouvons pas utiliser une autre architecture

- **CNN**

  On ne peut pas utiliser les réseaux de neurones à convolution, par ce que l'entrainement de ce réseau se fait de manière non supervisée, si on trouve une méthode pour l'entrainer, l'exploration des filtres (couches de convolutions) pour trouver les itemsets fréquents est un travail presque infaisable.

- **LSTM**

  Les LSTM sont les variantes les plus puissante des réseaux de neurones récurrents, si les RNN classiques sont capable de tenir 10 entrées on mémoire les LSTM sont capable de tenir presque 1000 entrées, mais cet avantage n'est pas suffisant si on travaille avec des bases de transaction qui dépasse les 10000 transactions par exemple.

- **VAE**

  Les VAE ou Variational autoencoder, c'est un variant de l'autoencoder ou il est utilisée généralement comme un modèle générateur ou il génère des données à partir

de son espace latent, mais pour faire ça au lieu d'utiliser un seul vecteur pour la représentation latent il faut utiliser 2 vecteurs, et pour cette raison l'effet que les même transactions sont regroupée ne sera plus là.

2.3 Dimensions de l'autoencoder utilisé

Après des tests empiriques, nous avons choisi les nombre de neurones et de couches suivants :

- **Nombre de neurones**
  - Couche d'Entrée : nombre de neurones = nombre des items.
  - Couche caché(L) : nombre de neurones couche (L-1) / 2.
  - Couche du milieu : 20% de couche d'Entrée
  - Après la couche du milieu le nombre de neurones dans les couches est symétrique jusqu'à la couche de sortie qui est égale à la couche d'entrée.
    Si on a **N** nombre de couches, le nombre de neurones dans la couche **N** = nombre de neurones dans la couche **1**, et le nombre de neurones dans la couche **N-1** = nombre de neurones dans la couche **2** et ainsi de suite.
- **Nombre de couches**

  Pour ne pas tomber dans le problème d'overfitting nous avons opté pour 5 couches :
  - Couche d'Entrée
  - Couche de sortie
  - 3 Couches intermédiaires

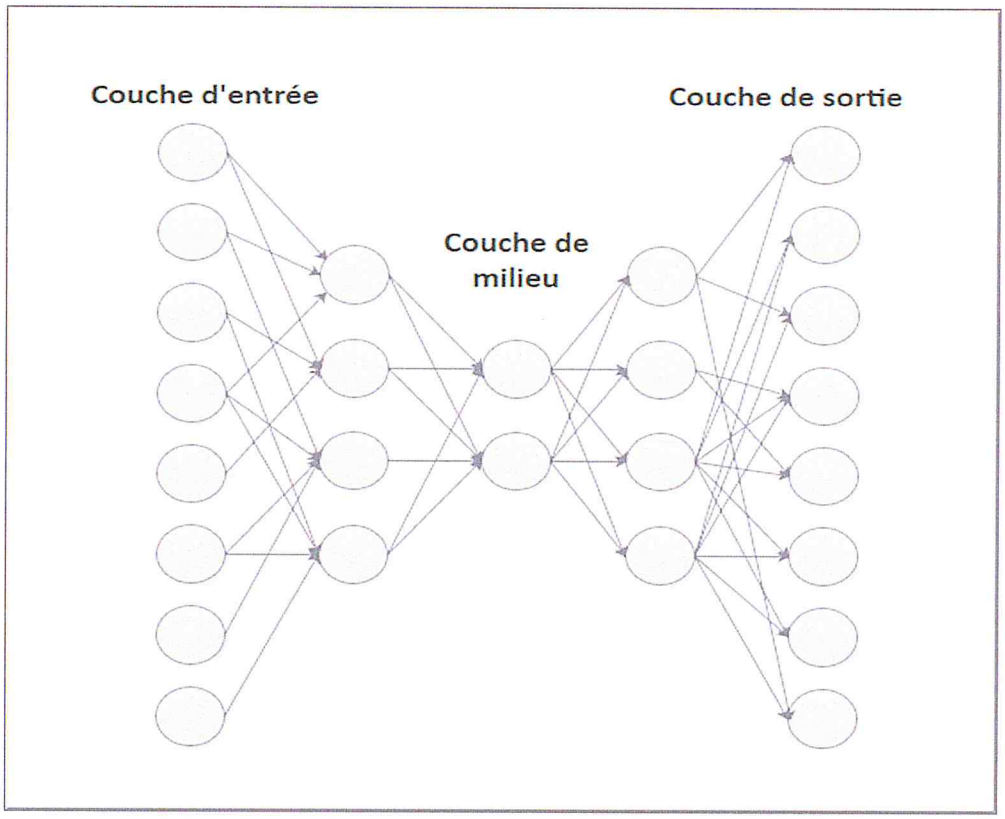


Figure 4. 2 : Illustration du nombre de couche et de neurones dans le modèle.

3. phases de construction du system

3.1 Phase d'entrainement du modèle

Dans cette phase, nous créons un modèle et nous l'entraînons sur un dataset de listes d'itemsets, jusqu'à ce que nous obtenions un modèle entrainé qui est prêt à être utilisé pour extraire les itemsets fréquents. Cette phase est représentée dans la figure ci-dessous.

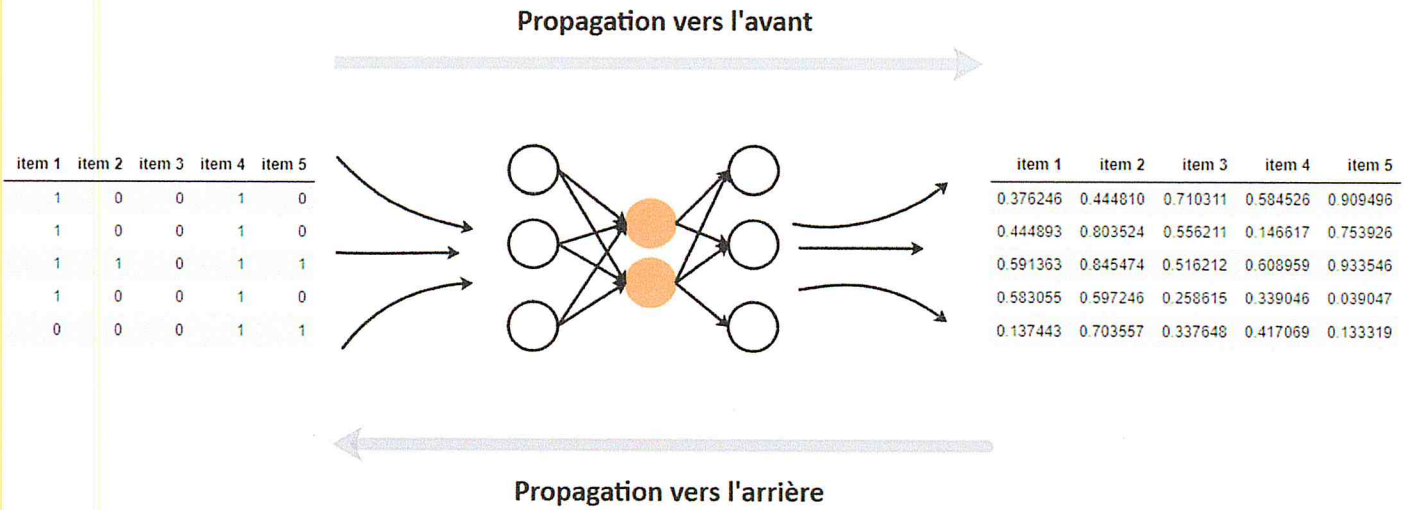


Figure 4. 3 : La phase d'entrainement du modèle.

3.2 La phase de déploiement du modèle

Dans cette phase, le modèle est déjà entraîné. Il suffit d'utiliser une partie ou la globalité du model pour trouver les itemsets fréquents.

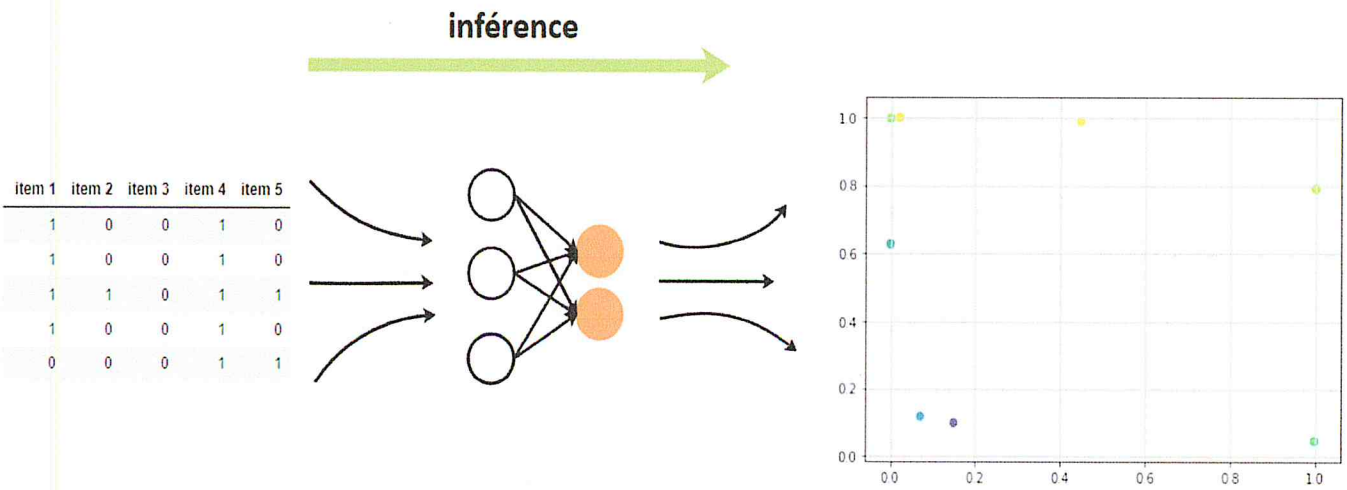


Figure 4. 4 : La phase de déploiement du modèle.

4. Pseudo algorithme

L'algorithme proposé est nommé Autoencoder-FIM (A-FIM) comme son nom l'indique, il est basé sur le principe d'autoencoder. L'algorithme se présente comme suit :

4.1 Récapitulatif de l'algorithme

Algorithme : A-FIM

- **Phase d'entrainement :**

Entrainer le model pour n itérations

- **Phase de déploiement :**

Entrées : *données*

Pour chaque (transactions) *Tj* **faire**

Obtenir sa représentation latente

Si la représentation latent existe déjà **alors** incrémenter le support.

Sinon

Ajouter la représentation latente

Fin Si.

Fin Pour.

Sortie : itemsets globaux avec leurs supports.

- **Phase d'extraction des itemsets fréquents :**

Entrées : itemsets globaux, minSup.

Pour chaque **item** calculer support.

Si (Supp-item < minSup*)* **alors** éliminer l'item.

Fin Si

Fin Pour.

Sommer les supports des sous ensemble des itemsets pour obtenir les itemsets fréquents.

4.2 Complexité de l'algorithme

- **Complexité d'entrainement**

Si nous supposons que l'entrainement prend N itérations [96] [97]:

$$O\,(\text{time gradient descent}) = n\,.\,n^5 = n^6 \qquad (4)$$

n^5 : Pour trouver tous les poids entre les couches

- **Complexité de déploiement**

Si nous supposons qu'il y a le même nombre de neurones dans chaque couche, et que le nombre de couches est égal au nombre de neurones dans chaque couche, nous trouvons [97] [98]:

$$Nmul = O\,(n\,.\,n^3) = n^4 \qquad (5)$$

n : Nombre de couches et de neurones

n^3 : Complexité de multiplication des matrices (n*n)

5. Entraînement du modèle

5.1 Données d'entrainement

L'une des premières choses à considérer avant l'entraînement d'un modèle sont les données d'entrainement. Pour notre modèle nous avons travaillé sur des donnés d'item de valeur numérique comprise entre 1 et 0 comme l'illustre la figure suivante.

	item 1	item 2	item 3	item 4	item 5	item 6	item 7	item 8	item 9	item 10
1	0	0	0	1	0	0	1	1	1	0
2	0	1	1	0	1	1	1	1	1	0
3	0	0	0	0	1	1	1	0	0	1
4	0	1	1	0	1	0	0	1	1	1
5	1	1	0	1	1	0	0	1	0	1
6	0	1	0	0	0	0	1	0	1	0
7	1	0	0	0	1	1	1	0	0	0
8	0	0	0	1	0	1	0	0	0	1

Figure 4. 5 : Exemple de données utilisées dans l'entrainement du modèle.

5.2 Fonction d'activation

Les fonctions d'activation sont une composante extrêmement importante des réseaux de neurones profonds. Ils décident si un neurone doit être activé ou non. Si l'information que le neurone reçoit est pertinente pour l'information donnée ou si elle doit être ignorée [77].

$$Y = \text{Activation}\left(\sum (w \times x) + b\right)$$

W: poids

X : entrée

B : biais

Pour notre travaille nous avons opté pour la fonction d'activation Sigmoid dans tout le model sauf pour la couche de milieu, où on a utilisé la fonction d'activation Leaky ReLU. La raison principale pour laquelle nous utilisons la fonction Sigmoid est qu'elle existe entre (0 à 1). Cette propriété de la fonction sigmoïde est utile lorsque nous travaillons avec des listes d'itemsets. La raison pour utiliser Leaky ReLU est d'avoir plus de flexibilité dans la représentation latente.

Équation de la fonction Sigmoid : $f(x) = \frac{1}{1+e^{-x}}$

Équation de la fonction Leaky ReLU : $f(x) = \begin{cases} x, & si \quad x > 0 \\ 0.01x, & sinon \end{cases}$

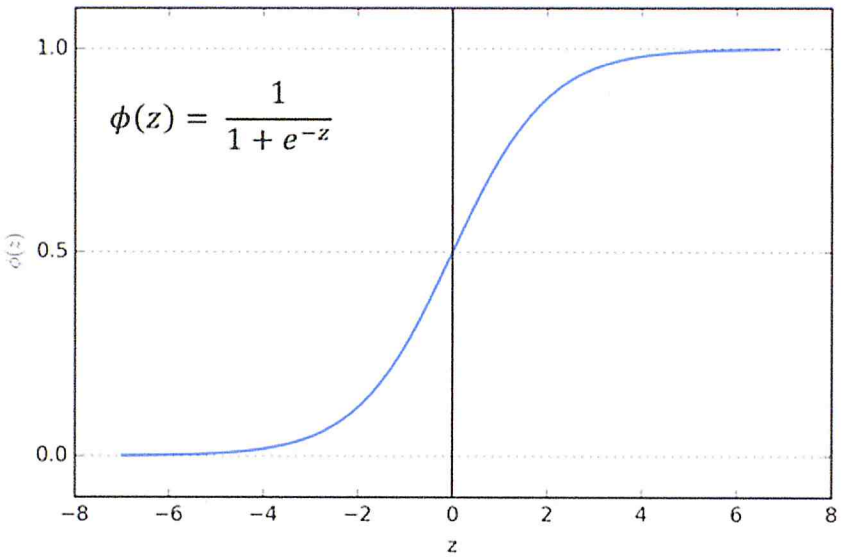


Figure 4. 6 : Représentation graphique de la fonction d'activation Sigmoid [78].

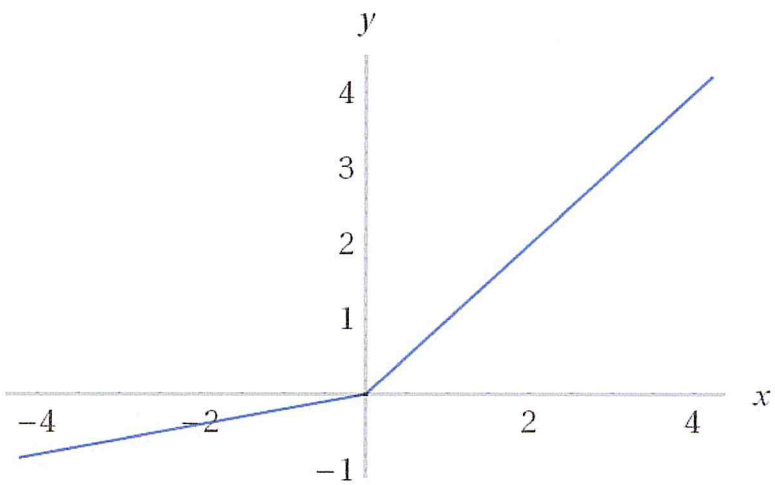


Figure 4.7 : Représentation graphique de la fonction d'activation Leaky ReLU [78].

5.3 Fonction de coût (Loss Function)

Souvent, lorsque nous construisons un modèle d'apprentissage automatique, nous développons une fonction de coût capable de mesurer l'efficacité de notre modèle. Cette fonction pénalisera toute erreur de notre modèle (en assignant un coût) par rapport aux valeurs actuelles des paramètres. Ainsi, en minimisant la fonction de coût, nous pouvons trouver les paramètres optimaux qui donnent les meilleures performances du modèle [75].

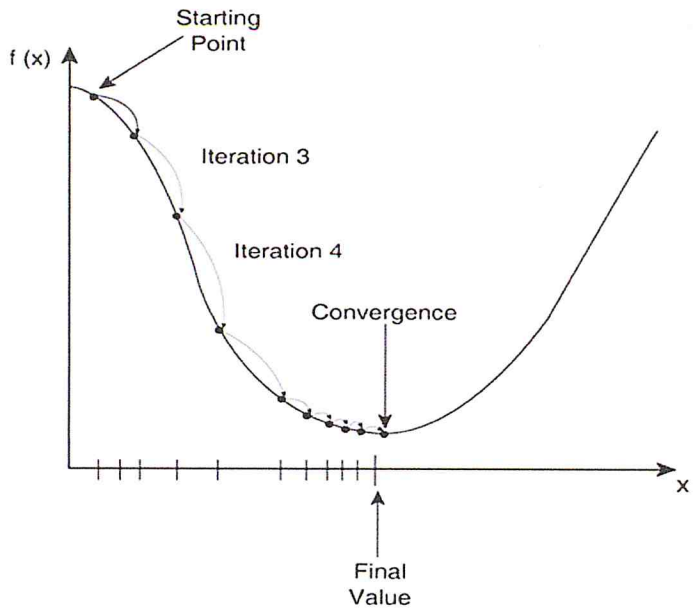


Figure 4.8 : Représentation graphique d'une fonction de coût [76].

Pour notre model on a utilisé **Mean squared error** (**Erreur quadratique moyenne**) comme fonction de coût. Supposons que $\hat{X}i$ soit le vecteur dénotant les valeurs de $\boldsymbol{n}$ nombre de prédictions. Aussi, Xi est un vecteur représentant $\boldsymbol{n}$ nombre de valeurs vraies, La formule de l'erreur quadratique moyenne est donnée ci-dessous :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\widehat{Xi} - Xi)^2$$

Lorsque la valeur de MSE est nulle, cela signifie qu'il y a une précision parfaite trouvée par l'estimateur. Cette condition est idéale et généralement impossible à réaliser dans la pratique [89].

5.4 Optimisateur

Les algorithmes d'optimisation nous aident à minimiser (ou maximiser) une "Fonction Objective" (autre nom pour la fonction Error) E(x) qui est simplement une fonction mathématique qui dépend des paramètres d'apprentissage internes du modèle. Il existe plusieurs algorithmes d'optimisation à savoir [79] :

- **Gradient Descent**

  C'est l'algorithme d'optimisation le plus populaire utilisé dans l'optimisation d'un réseau de neurones [79].

- **Variantes de Gradient Descent**

  Gradient Descent calculera le gradient de l'ensemble des données mais n'effectuera qu'une seule mise à jour ; donc il peut être très lent et difficile à contrôler pour les datasets qui sont très volumineux.

  - **Stochastic gradient descent**

    Effectue une mise à jour des paramètres pour chaque exemple d'entraînement, il s'agit généralement d'une technique beaucoup plus rapide [79].

- **Optimisation de gradient descent**

  Il existe plusieurs algorithmes qui sont utilisés pour optimiser encore plus le Gradient descent.

  - **Momentum**

    La grande variance des oscillations de la SGD rend difficile l'atteinte de la convergence. C'est pourquoi on a inventé une technique appelée Momentum qui accélère la SGD en naviguant dans la direction pertinente et attenue les oscillations dans des Directions non pertinentes [79].

  - **Nesterov accelerated gradient**

    Un chercheur nommé Yurii Nesterov a relevé un problème avec Momentum. Ce qui se passe en réalité, c'est que lorsque nous atteignons les minima, c'est-à-dire le point le plus bas de la courbe, l'élan (Momentum) est assez élevé et il ne sait pas ralentir ce qui lui faire rater complètement les minima [79].

  - **Adagrad**

    Il permet simplement au taux d'apprentissage de s'adapter en fonction des paramètres. Il fait donc de grosses mises à jour pour les paramètres peu fréquents et de petites mises à jour pour les paramètres fréquents [79].

  - **AdaDelta**

    Il s'agit d'une extension d'AdaGrad qui tend à supprimer le problème du taux d'apprentissage en décomposition [79].

  - **Adam**

    Adam signifie Adaptive Moment Estimation. L'estimation du moment adaptatif est une autre méthode qui calcule les taux d'apprentissage adaptatif pour chaque paramètre [79].

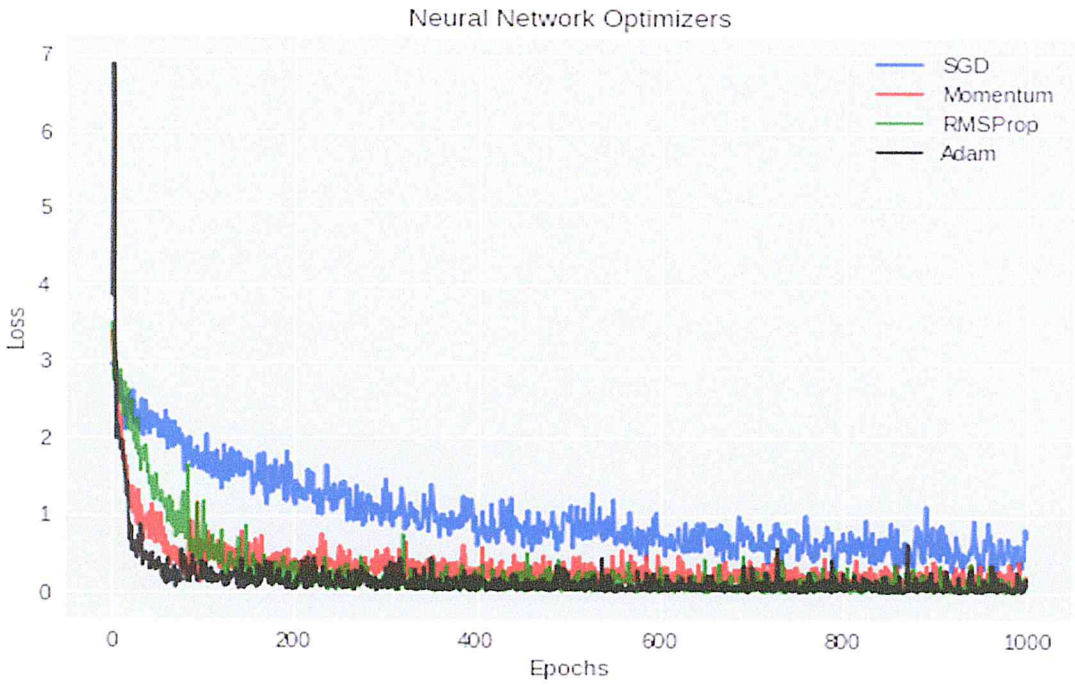


Figure 4. 9 : Illustration des différentes valeurs de fonction de coût obtenues par les différents algorithmes d'optimisation. [80]

Nous avons travaillé avec l'optimisateur Adam parce qu'il donne la valeur minimale de la fonction de coût et est le premier à converger -voir figure ci-dessus-.

5.5 Taux d'apprentissage

Le taux d'apprentissage est un hyper-paramètre qui nous permet de contrôler dans quelle mesure nous ajustons les poids de notre réseau en fonction du gradient de perte. Plus la valeur est basse, plus la pente descendante est lente. Bien que ce soit une bonne idée (en utilisant un faible taux d'apprentissage) pour s'assurer que nous ne manquons aucun minimum local. Cela pourrait aussi signifier que nous prendrons beaucoup de temps pour converger, surtout si nous sommes coincés sur un plateau [74].

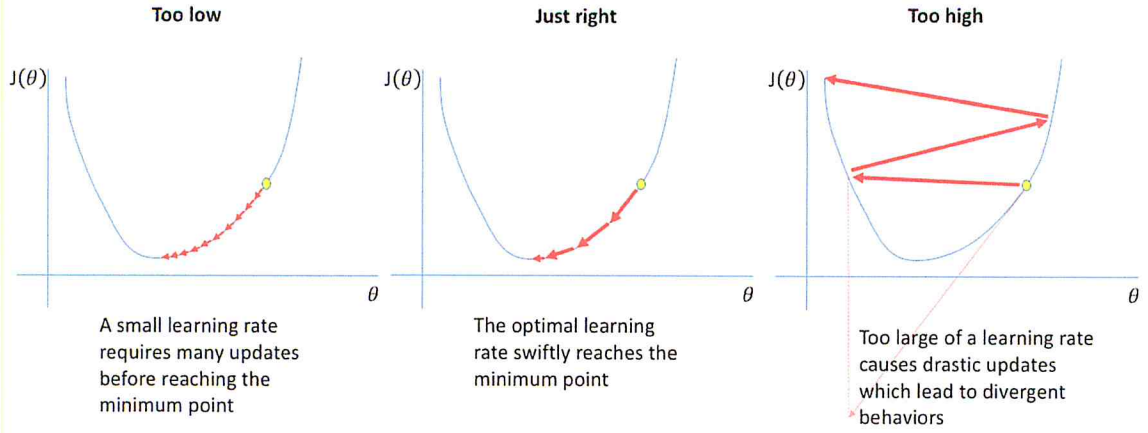


Figure 4. 10 : Figure illustrant les différentes valeurs de taux d'apprentissage et son impact [74].

Après les tests empiriques que nous avons faits nous avons opté pour un taux d'apprentissage de 0.01.

5.6 Étapes de l'entraînement

L'entraînement d'un réseau de neurones se compose de 5 étapes :

1. Initialisation des poids et des biais

L'un des points de départ à prendre en charge lors de la construction de notre réseau est d'initialiser correctement notre matrice de poids en considérant deux scénarios qui peuvent causer des problèmes pendant l'entraînement du modèle [71] :

1) **Initialisation de tous les poids à 0 :**

   Cela rend notre modèle équivalent à un modèle linéaire. Lorsque nous réglons tous les poids à 0, la dérivée par rapport à la fonction de coût est la même pour chaque w dans W^l donc, tous les poids ont les mêmes valeurs dans l'itération suivante.

   Il est important de noter que le réglage des biais à 0 ne créera pas de problèmes car les poids non nuls prennent soin de briser la symétrie et même si le biais est à 0, les valeurs dans chaque neurone sont toujours différentes [71].

2) **Initialisation aléatoire des poids :**

   L'initialisation aléatoire des poids, suivant la distribution normale standard tout en travaillant avec un réseau (profond) peut potentiellement conduire à 2 problèmes à savoir la disparition des gradients ou l'explosion des gradients [71].

a) Disparition des gradients

Dans le cas des réseaux profonds, pour toute fonction d'activation, Abs (dW) deviendra de plus en plus petit au fur et à mesure que l'on recule avec chaque couche pendant la propagation arrière. Les couches antérieures sont les plus lentes à entraîner dans un tel cas.

La mise à jour du poids est mineure et se traduit par une convergence plus lente. Cela ralentit l'optimisation de la fonction de coût. Dans le pire des cas, cela peut empêcher complètement le réseau de neurones de s'entraîner encore plus [71].

Abs (dW) : valeur absolue de la dérivée partielle du poids W.

b) Explosion des gradients

C'est exactement le contraire de disparition des gradients. Lorsque ces poids sont multipliés le long des couches, ils provoquent un changement important dans le coût. Ainsi, le gradient serra également importants et cela signifie que les changements des valeurs des paramètres serra énorme, il peut en résulter une oscillation autour des minimums ou même un dépassement de l'optimum de ce fait le modèle n'apprendra jamais.

Un autre impact de l'explosion des gradients est que des valeurs énormes des gradients peuvent causer le débordement des nombres, ce qui entraîne des calculs incorrects ou l'introduction de NaN. Cela pourrait également conduire à ce que la fonction de coût prenne aussi la valeur NaN [71].

NaN : pas un numéro (not a number)

1.1 Initialisation des poids

Pour les réseaux profonds, nous pouvons utiliser une heuristique pour initialiser les poids en se basant sur la fonction d'activation non linéaire. Bien que ces heuristiques ne résolvent pas complètement le problème de disparition/explosion du gradient, elles aident à réduire ce problème en grande ampleur [71].

Des tests d'initialisation sur les trois techniques et les mêmes données (dataset de Mnist) en été effectué, et on peut observer le résultat suivant [72] :

- **Initialisation à zéro :**

  Coût après 15000 itérations : **0.7**

  Précision : **0,5**

- **Initialisation aléatoire :**
  Coût après 15000 itérations : **0.38**
  Précision : **0,83**
- **Initialisation avec heuristique :**
  Coût après 15000 itérations : **0.07**
  Précision : **0,96**

Dans notre modèle nous avons initialisé les poids avec l'initialisation de Xavier [88] :

$$var(W) = \frac{2}{Nin+Nout}$$

N*in*, **N*out* :** nombre des entrées et de sortie dans la couche.

1.2 Initialisation des biais

Nous initialisons les biais à zéro, puisque la rupture de l'asymétrie est fournie par les petits nombres dans les poids [87].

2. Propagation vers l'avant

En utilisant l'entrée X, les poids W et les biais b, pour chaque couche nous calculons Z et A. La couche finale, nous calculons f (A^ (L-1)) qui pourrait être une fonction Sigmoid, softmax ou linéaire de A^ (L-1) et cela donne la prédiction Y_hat [71].

- $W^{[l]}$: matrice des poids de dimension (taille de couche L, taille de couche L-1).
- $b^{[l]}$: vecteur des biais de dimension (taille de couche L, 1).
- $Z^{[l]}$: linéaire activation dans la couche L.
- $g^{[l]}$: function non linéaire.
- $A^{[l]}$: activation non linéaire ; sortie de $g^{[l]}(Z^{[l]})$, où $A^{[0]}$ est l'entrée X.

$$Z^{[l]} = W^{[l]} * A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

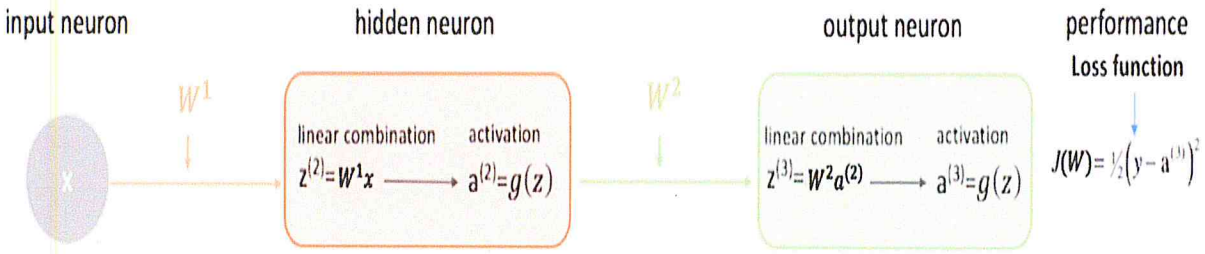


Figure 4. 11 : Propagation vers l'avant [73].

3. Calculer la fonction de coût

Il s'agit d'une fonction de l'étiquette actuelle Y et de l'étiquette prévue Y_hat. Il montre à quel point nos prédictions sont éloignées de la cible réelle. Notre objectif est de minimiser cette fonction de coût [71].

4. Propagation vers l'arrière

Dans cette étape, nous calculons les gradients de la fonction de coût f(Y, Y_hat) par rapport à A, W et b appelés *d*A, *d*W et *d*b. En utilisant ces gradients, nous mettons à jour les valeurs des paramètres de la dernière couche à la première [71].

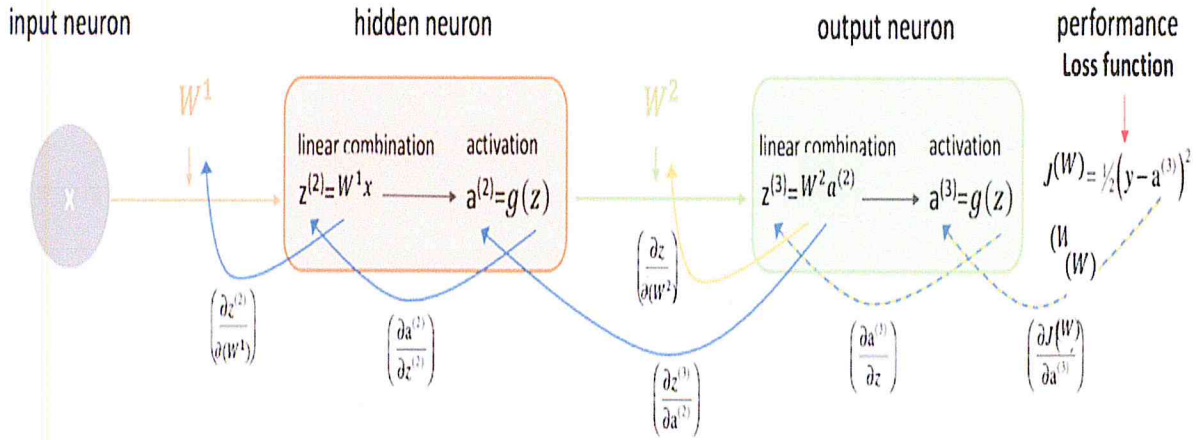


Figure 4. 12 : Propagation vers l'arrière [73].

5. Répéter

Répéter les étapes **2** à **4** pour n itérations jusqu'à ce que nous ayons l'impression d'avoir minimisé la fonction de coût sans Overfitting sur les données de l'entraînement [71].

5.7 Condition d'arrêt d'entraînement

Dans l'entraînement des réseaux de neurones, l'entraînement est arrêté lorsque la fonction de coût converge, c'est-à-dire lorsque nous ne pouvons pas avoir une petite valeur par rapport à la valeur que nous avions dans les itérations précédentes.

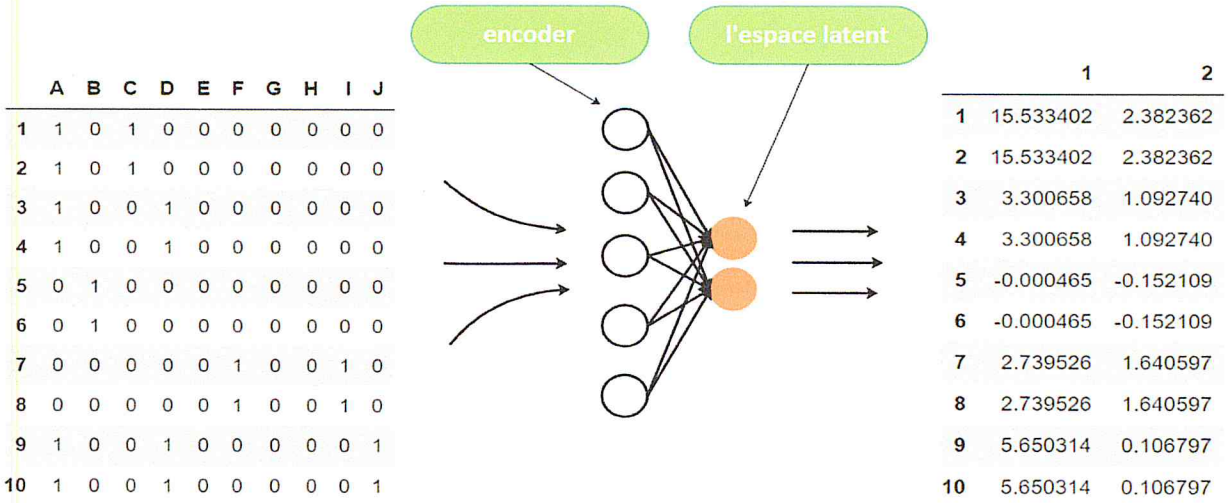
Le tableau suivant montre que la fonction de coût a diminué de 0,004 dans les dernier 4000 itérations, mais dans les première 2000 itérations la fonction de coût a diminué de 0,14.

Itération	Fonction de cout
1	0.2635
2000	0.1211
4000	0.114
6000	0.1108
8000	0.1092
10000	0.1082
12000	0.1076
14000	0.1072
16000	0.1069
18000	0.1066
20000	0.1065

Tableau 4. 1 : Changement de fonction de coût par rapport au nombre d'itération.

6. Déploiement de modèle

Le déploiement d'un modèle est l'étape qui vient après l'entraînement où nous sauvegardons les matrices de poids et nous utilisons la partie encodeur du réseau de neurones pour mapper les itemsets à l'espace latent.



	A	B	C	D	E	F	G	H	I	J
1	1	0	1	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	0	0	1	0	0	0	0	0	0
4	1	0	0	1	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	1	0
8	0	0	0	0	0	1	0	0	1	0
9	1	0	0	1	0	0	0	0	0	1
10	1	0	0	1	0	0	0	0	0	1

	1	2
1	15.533402	2.382362
2	15.533402	2.382362
3	3.300658	1.092740
4	3.300658	1.092740
5	-0.000465	-0.152109
6	-0.000465	-0.152109
7	2.739526	1.640597
8	2.739526	1.640597
9	5.650314	0.106797
10	5.650314	0.106797

Figure 4. 2 : Déploiement du modèle et l'utilisation de l'encoder.

La figure ci-dessus montre que les mêmes itemsets ont la même valeur dans l'espace latent prennent les itemsets N° 1 et 2, ils contiennent les deux, les items A et C et ils ont la même valeur dans l'espace latent [15.533402, 2.382362]. Nous pouvant utiliser cette caractéristique pour explorer l'espace latent et extraire les itemsets fréquents.

6.1 L'exploration de l'espace lattent

En utilisant les itemsets de l'exemple de la figure 4.18 nous pouvons voir après le passage à travers le réseau de neurones que les mêmes itemsets sont groupés. La figure suivante illustre cet effet.

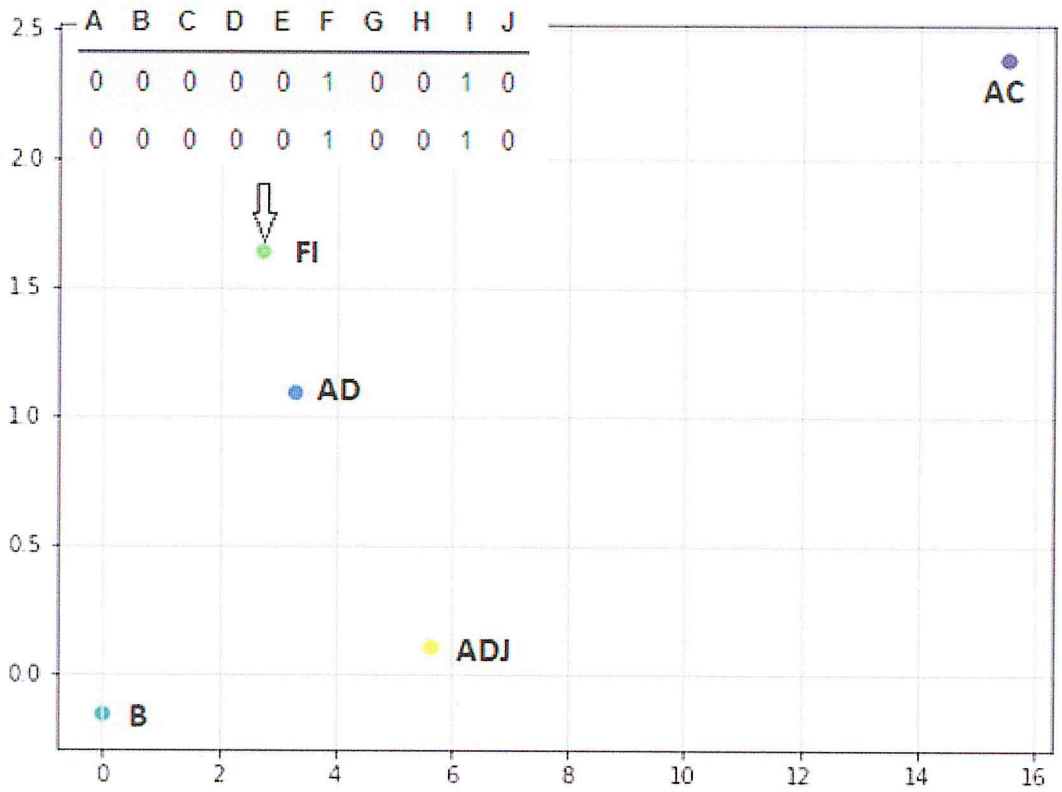


Figure 4. 3 : Représentation graphique de l'espace latent.

Nous calculons ensuite le support de ces itemsets en divisant le nombre d'itemsets dans le groupe par tous les itemsets.

FI: 0.2 → 2/10

AC: 0.2 → 2/10

AD: 0.2 → 2/10

B: 0.2 → 2/10

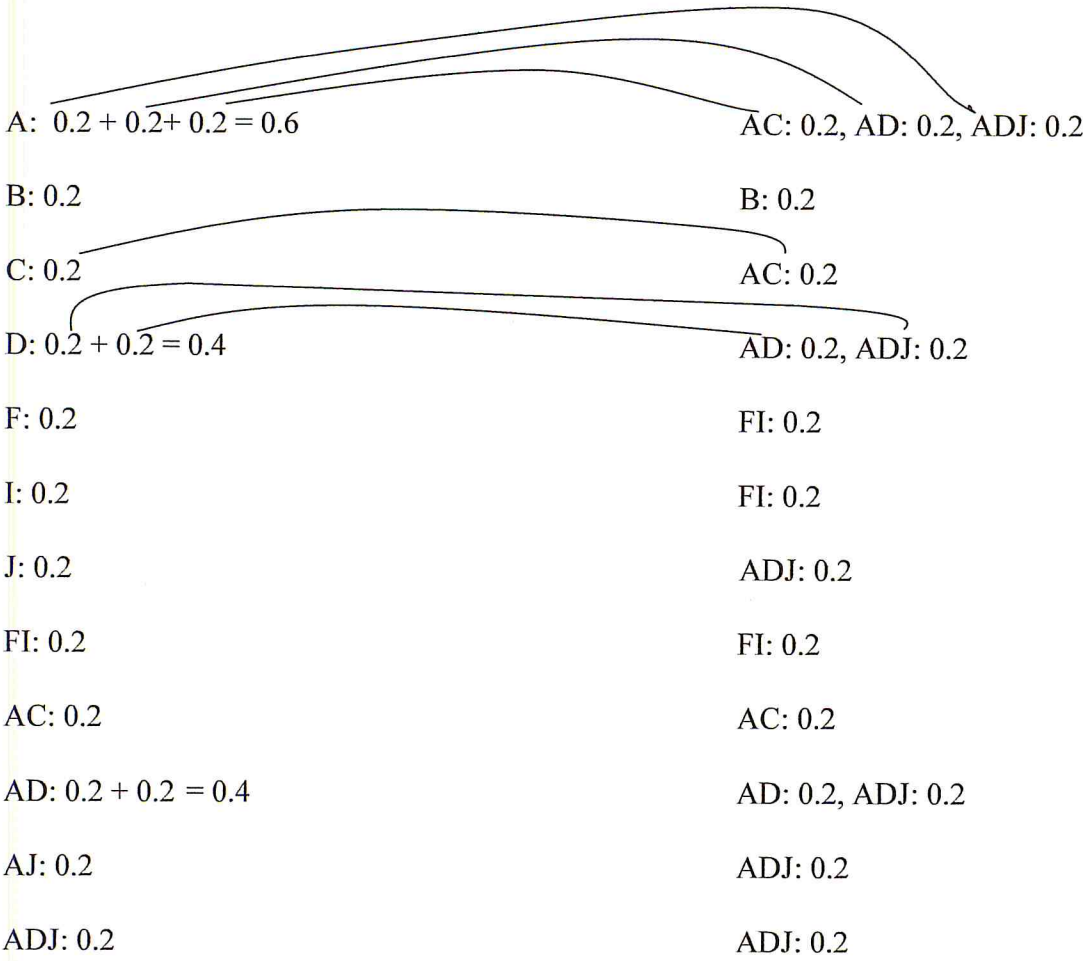
ADJ: 0.2 → 2/10

On peut aller plus loin et obtenir le support de tous les itemsets en sommant juste le support des sous itemsets, en utilisant la propriété de fermeture vers le bas des itemsets fréquents.

Exemple :

(ADJ) itemset fréquent →(A), (D), (J), (AD), (AJ), (DJ) sont tous fréquent

6.1.1 Exemple de calcule de supports des itemsets



6.1.2 Résultat

Après la sommation du support des sous itemsets on obtient le résultat suivant

[A]: 0.6
[B]: 0.2
[C]: 0.2
[D]: 0.4
[F]: 0.2
[I]: 0.2
[J]: 0.2
[FI]: 0.2
[AC]: 0.2
[AD] : 0.4

[AJ] : 0.2
[ADJ]: 0.2

Remarque :

Nous obtenons le résultat ci-dessus avec un seul passage sur l'ensemble de données. Les données passent par le réseau neuronal où ils seront groupés pour permettre le calcul des supports. D'autre part si nous avions utilisé l'algorithme apriori par exemple il aurait fallu plus de 4 passages pour obtenir le même résultat.

7. Conclusion

Dans ce chapitre nous avons proposé un modèle de réseau neuronal profond (autoencoder) qui peut être utilisé pour trouver des itemsets fréquents. Nous avons montré les étapes à prendre et les paramètres à considérer pour entraîner un réseau de neurones en général ainsi que notre modèle.

Il n'y a pas de méthode exacte à utiliser ou un manuel pour obtenir les meilleurs de résultats. Le choix des paramètres est pris après des tests que nous avons faits. Après avoir entraîné notre modèle, nous avons profité des propriétés de l'espace latent créé dans la couche intermédiaire du réseau (les mêmes itemsets vont au même point dans le latent espace) pour calculer le support de ces itemsets.

Chapitre V

Tests & Expérimentation

1. Introduction

Après avoir décrit notre solution, nous aborderons dans ce chapitre la partie des expérimentations de notre approche. En première partie, nous présenterons l'environnement de travail et les outils de développement utilisés. Ensuite, nous présentons notre jeu d'essai. Suivie d'une discussion pour mieux comprendre notre approche.

Nous répondrons dans ce chapitre a la question de « Est-ce que notre approche donne des résultats correcte et efficace, que ce soit en temps d'exécution ou en qualité de résultat ? Et enfin quelles sont les apports de notre approche ? »

2. Environnement de développement

2.1. L'environnement matériel

On a utilisé :

- Un MacBook PRO 2013 avec les caractéristiques suivantes :
  - 16 GO RAM 1600 MHz DDR3
  - 2.3 GHz Intel Core i7
  - NVIDIA GeForce GT 750 M
  - Système d'exploitation : MacOs Hight Sierra 10.13.4
- Un ordinateur de bureau avec les caractéristiques suivantes :
  - 3 GO RAM
  - 360 GO DDR
  - Intel® Core(TM) de duo CPU 2.93 GHz
  - Système d'exploitation : Windows 10 de 64-bit
- Un ordinateur portable Lenovo avec les caractéristiques suivantes :
  - 4 GO RAM
  - 512 GO DDR
  - Intel® Core(TM) i3-4005U CPU 1.7 GHz
  - Système d'exploitation : Windows 10 de 64-bit

2.2. L'environnement logiciel

2.2.1 Python

Python est un langage de programmation dynamique de haut niveau, interprété et polyvalent qui met l'accent sur la lisibilité du code. La syntaxe en Python aide les programmeurs à coder en moins d'étapes que Java ou C+++. Le langage a été fondé en 1991 par le développeur Guido Van. Le Python est largement utilisé dans les grandes organisations en raison de ses multiples paradigmes de programmation. Ils impliquent généralement une programmation fonctionnelle impérative et orientée objet. Il dispose d'une bibliothèque standard complète et volumineuse qui dispose d'une gestion automatique de la mémoire et de fonctions dynamiques [90].

Python est accompagné d'un grand nombre de bibliothèques intégrées. Beaucoup de bibliothèques sont destinées à l'intelligence artificielle et à l'apprentissage automatique. Certaines des bibliothèques sont TensorFlow (qui est une bibliothèque de deep learning de haut niveau), scikit-learn (pour l'exploration de données, l'analyse de données et l'apprentissage automatique).

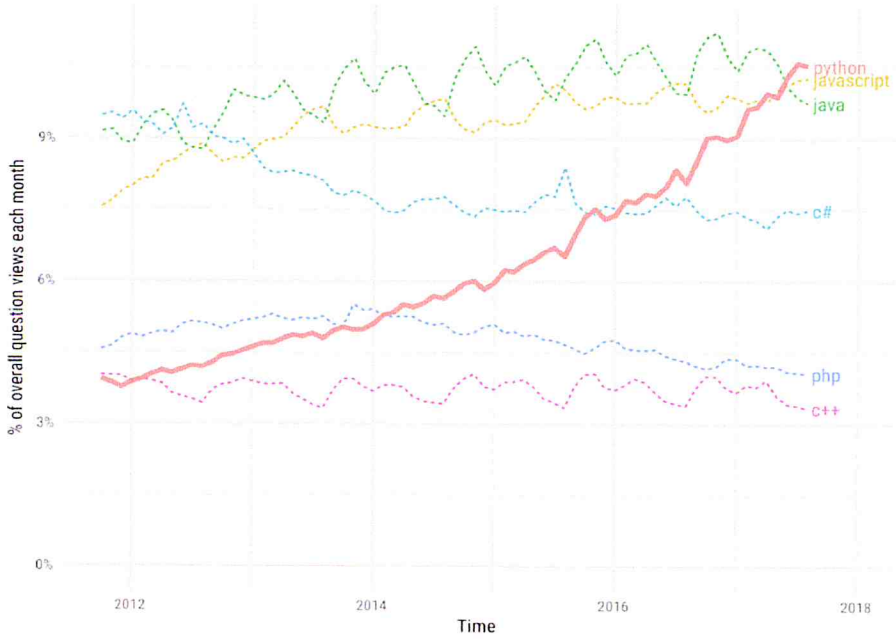


Figure 5. 1 : La croissance des langages de programmation [91].

2.2.2 Jupiter Notebook

Jupyter Notebook est une application web open-source qui vous permet de créer et de partager des documents contenant du code, des équations, des visualisations et du texte explicatif, Jupyter Notebook est utilisé pour [92]:

- Nettoyage et transformation de données.
- Simulation numérique.
- Modélisation statistique.
- Visualisation de données.
- Apprentissage automatique.

2.2.3 Bibliothèques utilisées

- **Pandas**

  Pandas est un package Python qui offre des structures de données rapides, flexibles et expressives conçues pour rendre le travail avec des données "relationnelles" ou "étiquetées" à la fois facile et intuitif [93].

- **TensorFlow**

  TensorFlow est une bibliothèque de logiciels open source pour le calcul numérique de haute performance. Son architecture flexible permet un déploiement facile de la computation sur une variété de plates-formes (CPU, GPU, TPU), et des ordinateurs de bureau aux clusters de serveurs. Développé à l'origine par des chercheurs et des ingénieurs de l'équipe Google Brain au sein de l'organisation d'intelligence artificielle de Google, il s'accompagne d'un grand support pour l'apprentissage automatique et le Deep Learning [94].

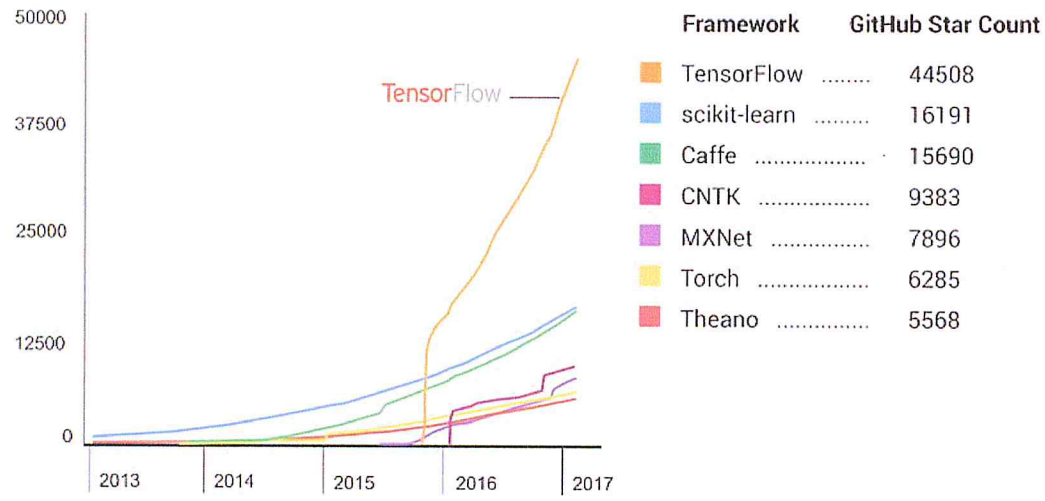


Figure 5. 2 : Popularité de TensorFlow par rapport aux d'autres bibliothèques de Deep Learning [95].

- **mlxtend**

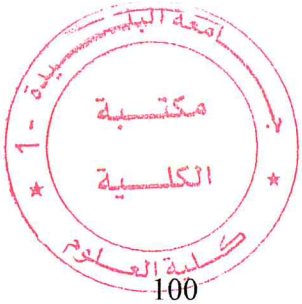
  Mlxtend est une bibliothèque d'apprentissage automatique pour python qui contient une implémentation de l'algorithme apriori [99].

3. Création du modèle

Pour la création de notre modèle d'extraction des itemsets, on passe par deux étapes, l'étape de génération des données pour l'entrainement (qui est une facultative, elle peut être omise dans présence des données réelles), et l'étape d'entrainement.

3.1 Génération des données

Pour générer les données, on utilise la bibliothèque pandas. La figure suivante montre comment se fait la génération, on génère 5000 itemsets de 20 items avec la probabilité de 0.7 que ces items ont la valeur 0 et 0.3 pour la valeur 1.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	0	1	0	0	1	1	0	0	1	0	1	0	0	1	0	0	1	1	1	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
3	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	1	1
4	0	1	0	1	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	0
5	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
6	0	0	0	1	1	0	1	1	0	0	1	0	0	0	1	0	0	0	1	1
7	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	1	0	1
8	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	0
9	0	0	1	1	0	0	0	1	0	0	0	1	0	1	1	0	0	1	1	0
10	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 5. 3 : Génération des données avec pandas.

3.2 L'entrainement

Dans cette phase, nous entraînerons le modèle pour 15000 itérations jusqu'à ce que la fonction de coût converge. La figure 5.5 montrer la minimisation de la fonction de coût par rapport au nombre d'itérations nous remarquons que la fonction de cout n'a pas changer pendant les 4000 dernières itérations donc si on entraine notre model pendant 30000 itérations la valeur de la fonction de cout sera toujours la même.

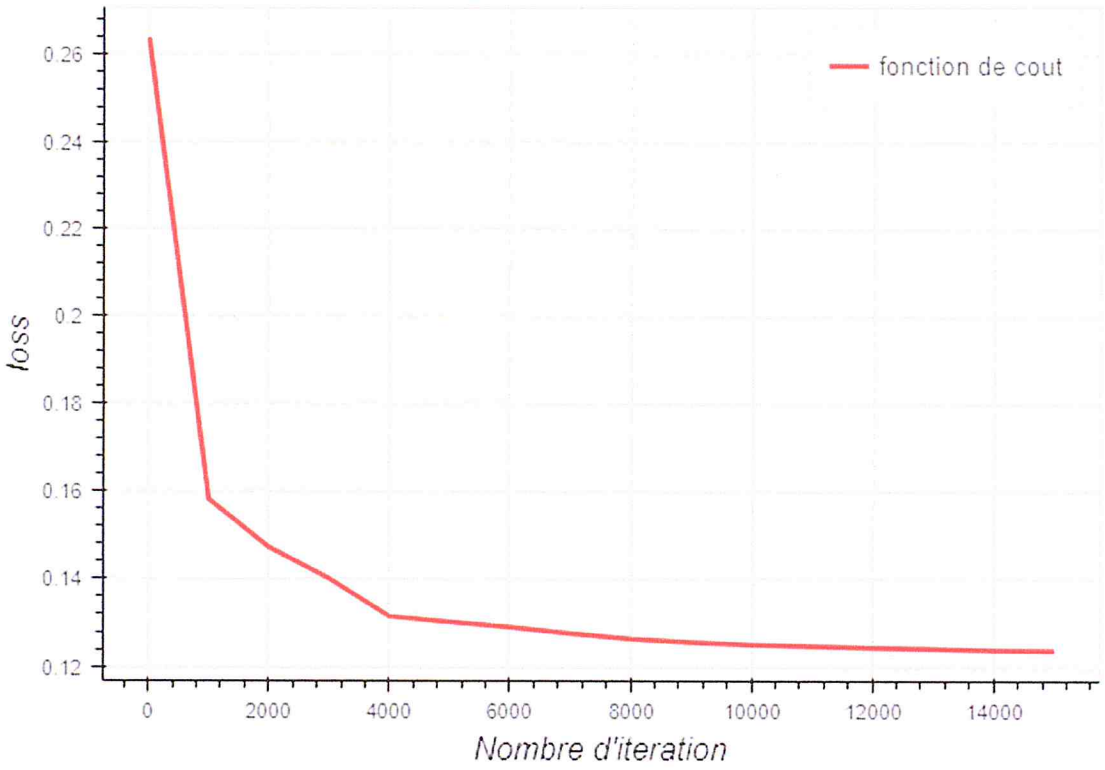


Figure 5. 4 : Changement de fonction de cout par rapport au nombre d'itérations

4. Tests et comparaison

Nous avons effectué des tests par rapport à deux critères : le temps d'exécution, et la qualité des résultats avec une comparaison avec l'algorithme classique apriori pour évaluer la performance et la qualité de notre approche.

4.1 Tests de qualité des résultats

4.1.1 Les résultats obtenus

Cet exemple de test est fait avec la dataset de mushrooms [100] cet dataset comprend des descriptions d'échantillons hypothétiques correspondant à 23 espèces de champignons ces enregistrements sont tirés à partir de The Audubon Society Field Guide pour les champignons nord-américains le donneur de cet dataset est Jeff Schlimmer en 27 avril 1987.

Nous allons travailler un minSup de 0.4, le résultat obtenu va être comparé avec le résultat de l'algorithme apriori pour voir si les résultats sont les mêmes.

	support	itemsets
16	1.000000	[85]
109	0.976914	[85, 86]
17	0.976914	[86]
6	0.975802	[34]
61	0.975802	[34, 85]
216	0.974815	[34, 85, 86]
62	0.974815	[34, 86]
18	0.921852	[90]
110	0.921852	[85, 90]
63	0.899753	[34, 90]

	itemset	supp
0	[i85]	1.000000
1	[i86]	0.976914
2	[i85, i86]	0.976914
3	[i34]	0.975802
4	[i85, i34]	0.975802
5	[i86, i34]	0.974815
6	[i85, i86, i34]	0.974815
7	[i90]	0.921852
8	[i85, i90]	0.921852
12	[i85, i34, i90]	0.899753

Figure 5. 5 : Comparaison des résultats d'apriori (tableau gauche) avec notre approche (tableau de droite)

Interprétation :

Nous remarquons que notre approche donne le même résultat qu'apriori -qui est un algorithme exact- en ce qui concerne le nombre d'itemsets fréquents et valeur des supports de ces itemsets. Donc nous pouvant dire que notre algorithme est un algorithme à résultat exacte.

4.1.2 Nombre d'itemsets fréquent

Nous avons effectué ce test avec des bases de transactions de 100 transactions, et de longueurs de transactions différentes, 10 items suivis de 20 items et ainsi de suite. Ce test est fait pour montrer le fait que la taille de l'espace de recherche augmente exponentiellement.

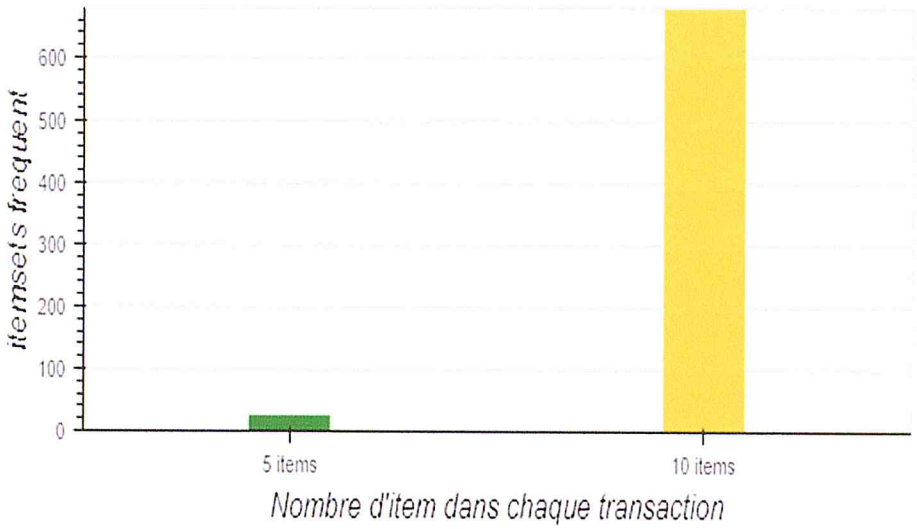


Figure 5. 6 : Nombre d'itemset fréquent dans des transactions de 5 item et 10.

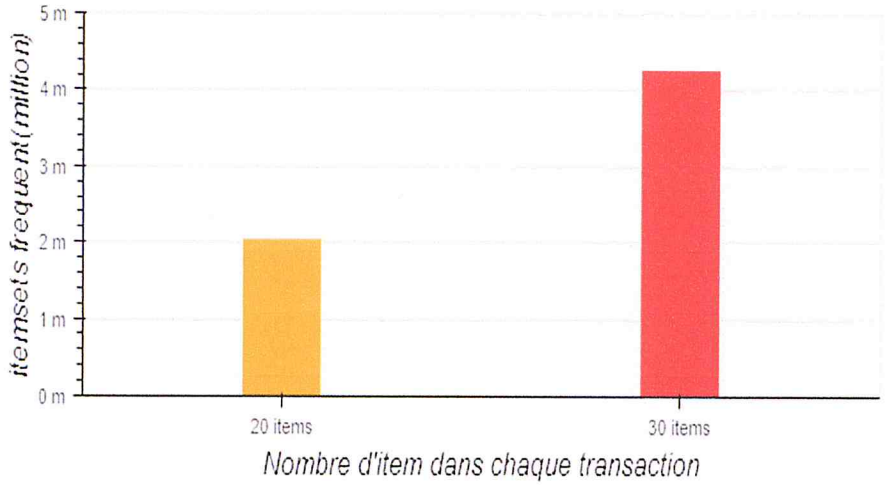


Figure 5. 7 : Nombre d'itemset fréquent dans des transactions de 20 item et 30.

Interprétation :

On remarque que le nombre d'itemsets obtenus en résultat dépend du nombre d'items entré en paramètre ou à chaque fois qu'on augmente le nombre des items, les itemsets fréquents obtenu en résultat augmente le contraire est vrai.

4.2 Tests de temps d'exécution

Les tests de temps d'exécution par rapport au nombre d'item et au nombre de transactions sont faits avec des données générer pour simuler les différents scénarios et un minsup = 0.00001 pour trouver tous les itemsets possibles.

4.2.1 Comparaison par rapport au nombre d'item

Dans ce test nous avons utilisé des bases de transactions de longueurs de différentes. On passe graduellement de 10 items par transaction a 100 items. Ce test est fait pour évaluer la performance de l'approche dans un grand espace de recherche.

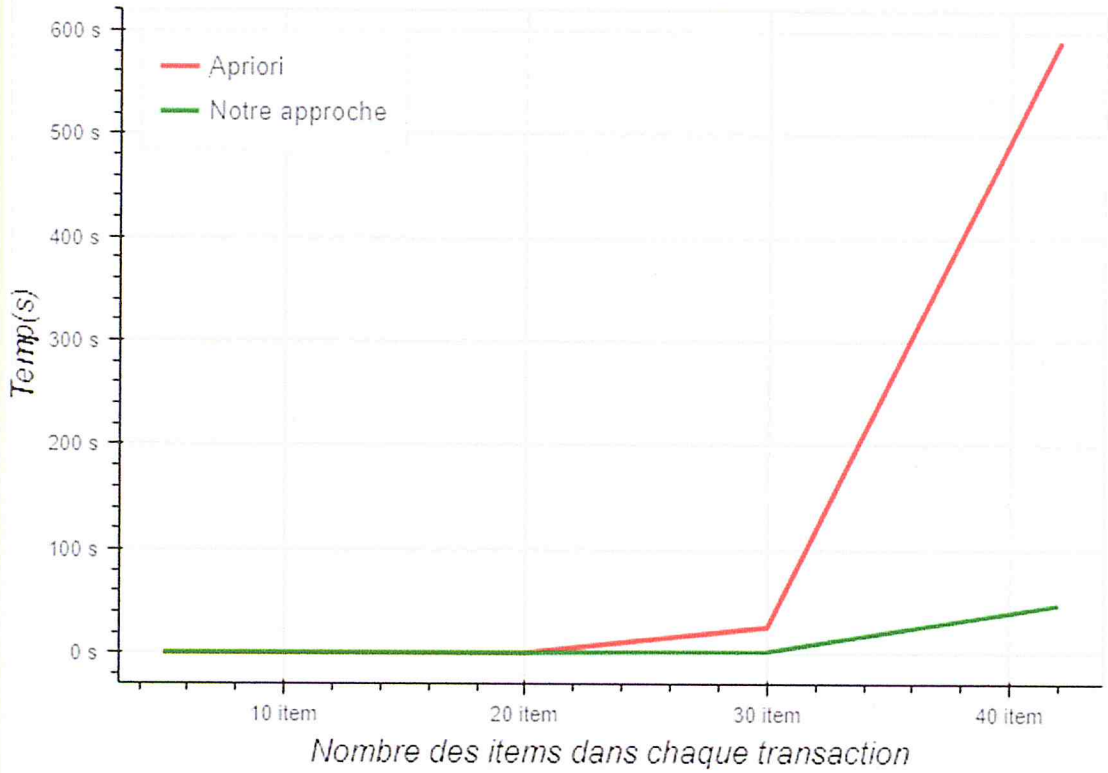


Figure 5. 8 : Comparaison de temps d'exécution par rapport au nombre des items.

Interprétation :

On remarque que le temps d'exécution augmente quand le nombre d'item augmente. Avec 42 item apriori prend presque 10min pour terminer le calcule alors que notre approche prend 50s la raison pour cela est que dans notre approche nous travaillons avec le paradigme top-down.

Remarque :

Dans la figure 5.6 le temps d'exécution commence à partir de la valeur 0.015s la figure suivant (un zoom de la figure 5.6) montrer le débute de temps d'exécution.

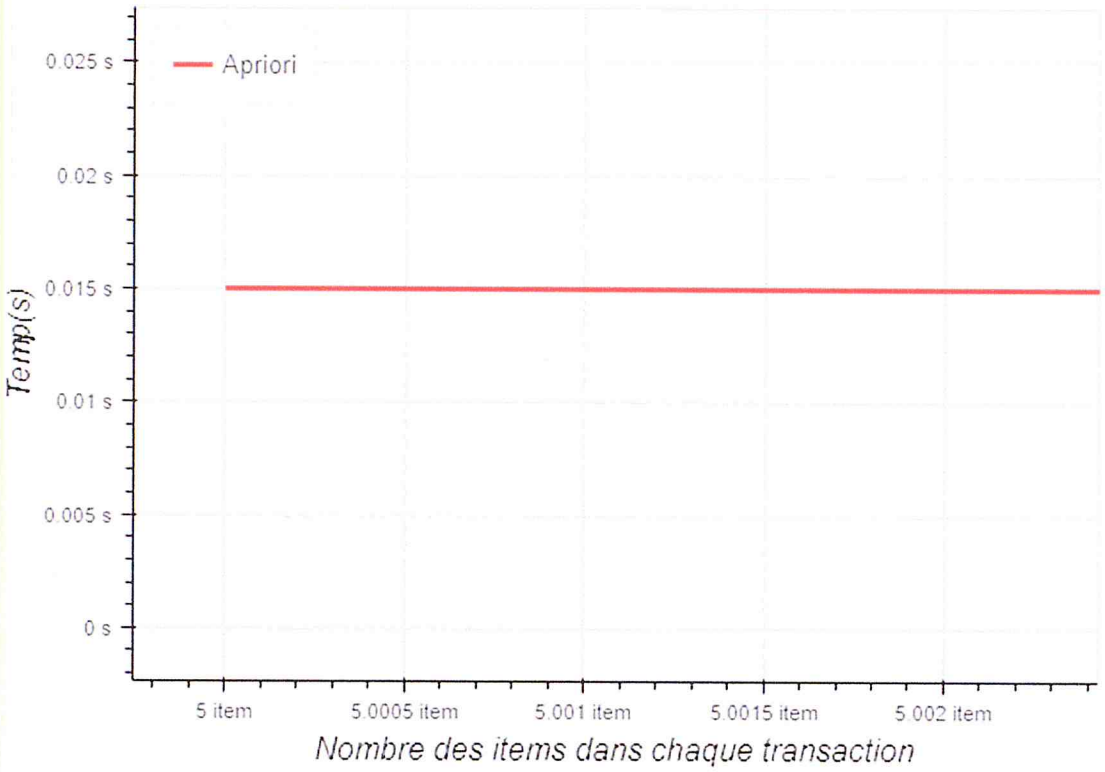


Figure 5. 9 : Début du temps d'exécution.

4.2.2 Comparaison par rapport au nombre de transactions

Dans ce test nous avons testé des bases de transactions de longueur différente avec 20 items. Ce test est fait pour évaluer la performance de l'approche quand on a une grande masse de données.

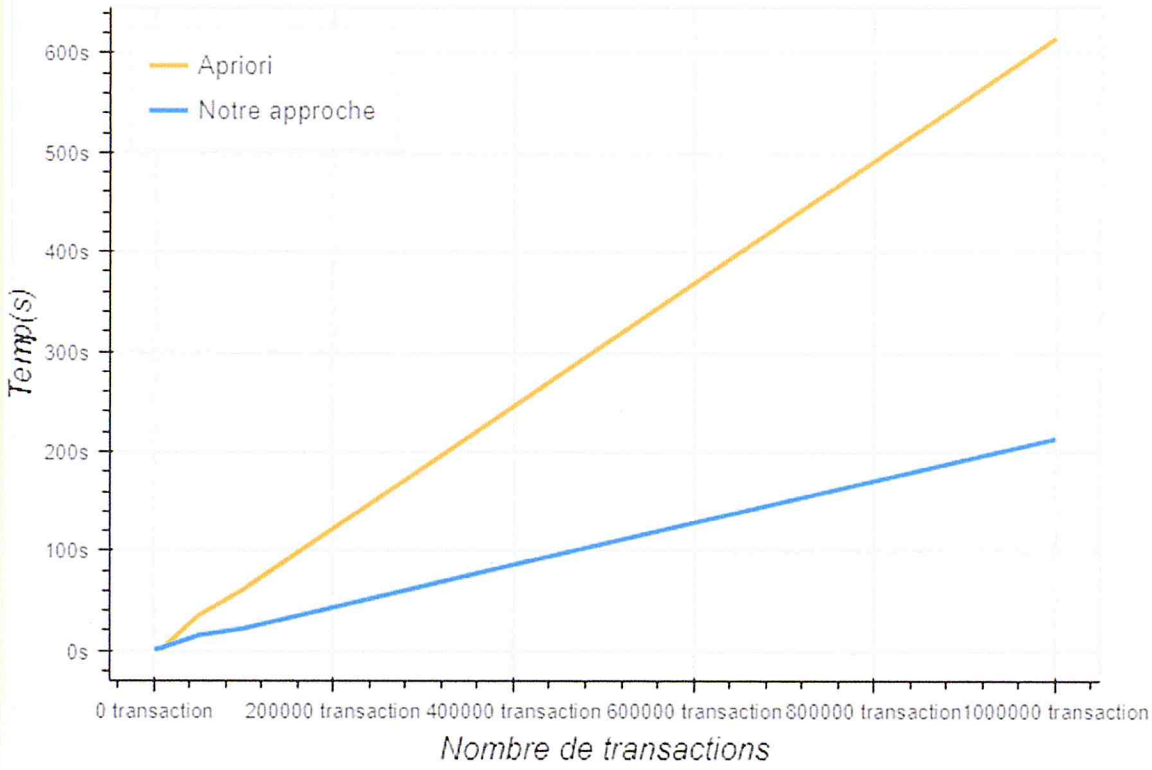


Figure 5. 10 : Comparaison de temps d'exécution par rapport au nombre de transactions.

Interprétation :

On remarque que le temps d'exécution augmente quand le nombre de transactions augmente, avec 1 million de transactions apriori prend presque 10min pour terminer le calcule alors que notre approche prend environ 3min. La raison c'est que dans notre approche nous atteignons ce résultat avec un seul passage sur la base des transactions alors qu'il faudra plusieurs passages à l'algorithme apriori pour atteindre le même résultat.

4.2.3 Comparaison par rapport au support

Ce test est fait avec la dataset de mushrooms qui contient 8123 transactions de longueur de 119 items [100], Ce test est fait pour évaluer la performance de l'approche avec des supportes différents, de 0.1 (10%) jusqu'à 0.9 (90%).

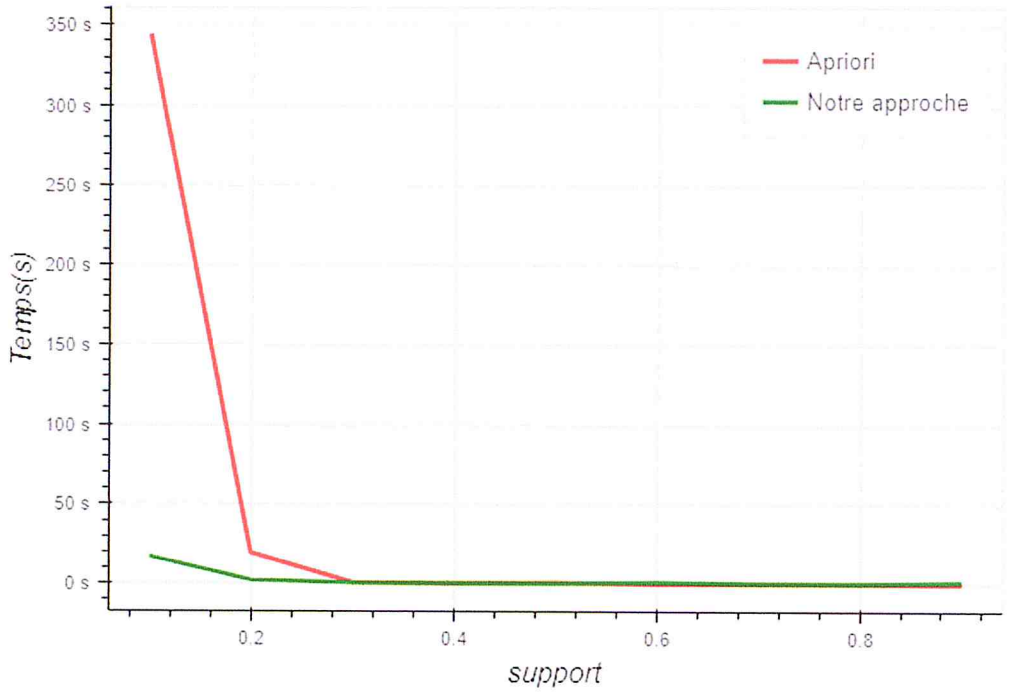


Figure 5. 11 : Comparaison de temps d'exécution par rapport aux différents supports.

Interprétation :

On remarque que le temps d'exécution augmente quand le support diminue. Avec un minsup de 0.1 apriori prend 6min pour terminer le calcule alors que dans notre approche l'exécution se fait en 20s. La raison pour cela est que la base de transactions contient 119 items si on diminuer le support l'espace de recherche augment c'est pour ça apriori prend 6min, alors que notre approche bénéficie d'une compression des données suivant le nombre d'item.

5. Discussion

5.1 Explication des performances supérieures de notre approche

Lorsque nous avons effectué les tests, notre approche a donné un meilleur temps d'exécution qu'apriori. La raison de ce résultat est que la base de données initial est compressée par notre modèle et c'est à partir de là que les itemsets fréquents sont extraits. Nous appelons cela l'étape intermédiaire où toutes les mêmes transactions sont regroupées pour que le support soit calculé.

5.1.1 Exemple illustratif

Nous avons utilisé une base de données de 50000 transaction de longueur de 10 items.

- **Base de données avant compression**

	A	B	C	D	E	F	G	H	I	J
0	0	1	0	0	1	1	0	0	0	1
1	1	1	0	1	0	1	0	1	0	0
2	0	0	0	1	0	0	0	1	1	1
3	1	1	1	0	1	1	1	1	1	0
4	0	0	1	0	0	1	1	1	0	1
5	0	0	0	0	0	1	0	0	0	1
6	1	0	0	0	0	1	0	1	0	1
7	1	0	0	0	1	1	0	0	1	0
8	1	1	1	0	0	1	1	0	0	0
9	0	1	0	0	1	0	1	1	0	0

Figure 5. 12 : Une partie de la base de données avant la compression.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 10 columns):
A    50000 non-null int32
B    50000 non-null int32
C    50000 non-null int32
D    50000 non-null int32
E    50000 non-null int32
F    50000 non-null int32
G    50000 non-null int32
H    50000 non-null int32
I    50000 non-null int32
J    50000 non-null int32
dtypes: int32(10)
memory usage: 1.9 MB
```

Figure 5. 13 : Information sur la base de donnes avant la compression.

- **Base de données après compression**

  Après la compression résultant de notre modèle, on obtient 904 transaction avec leurs supportes général, ces transactions vont être utilisée pour obtenir le support des itemsets fréquent final.

	intermediate_itemsets	support
0	ABEGIJ	0.00060
1	BFGJ	0.00100
2	BDEI	0.00218
3	BHJ	0.00192
4	ADEF	0.00108
5	ACDEG	0.00104
6	BCDGJ	0.00072
7	CGIJ	0.00114

Figure 5. 14 : Base de données après la compression

```
inter_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 904 entries, 0 to 903
Data columns (total 2 columns):
intermediate_itemsets    904 non-null object
support                  904 non-null float64
dtypes: float64(1), object(1)
memory usage: 14.2+ KB
```

Figure 5. 15 : Information sur la base de donnes après la compression.

- **Résultat final**

	itemset	support
0	G	0.41438
1	C	0.41014
2	H	0.40248
3	B	0.40224
4	I	0.40216
5	E	0.40200
6	A	0.40038
7	D	0.40024
8	J	0.39902
9	F	0.39690

Figure 5. 16 : Résultat final.

5.2 Possibilité d'obtenir les mêmes résultats avec une autre approche

Cette question se pose lorsque nous voyons la méthode de travail du modèle (compression de la base de données), pourquoi entraîner un modèle au lieu d'utiliser le clustering ou le hachage et pourquoi le modèle donne de meilleurs résultats.

- **Le Clustering**

  Si nous utilisons le clustering, deux problèmes se posent.

  1. Si l'on considère les itemsets intermédiaires comme des clusters et leur support sont calculé comme le nombre d'éléments dans le cluster sur nombre de transactions dans la base de données.

     Nous aurons le problème de ne pas connaître le nombre de clusters pour initialiser l'algorithme de clustering pour obtenir des résultats précis.

2. La complexité du calcul, nous devrons calculer la similitude entre toutes les transactions pour obtenir les clusters.

- **Le Hachage**

  Nous ne pouvons pas utiliser le hachage pour la simple raison du problème des collisions où le résultat obtenu ne sera pas précis.

5.3 Solution pour un nombre d'items qui diffère du nombre d'entré du modèle

Lorsqu'on travaille avec un nombre d'items dans les transactions qui diffère du nombre d'entrées du modèle, nous sommes confrontés à deux scénarios.

- **Nombre d'items inférieur au nombre d'entrées**

  La solution pour cette situation est de compléter les transactions avec des 0.

  Exemple : nombre d'item = 5, nombre d'entrées = 10

  1 0 1 0 0 → 1 0 1 0 0 0 0 0 0 0

  Noter que dans l'exemple ci-dessus les deux transactions ont les mêmes items qui ont la valeur **1** l'item **A** et **C**, malgré que les longueurs soient différentes.

- **Nombre d'items inférieur au nombre d'entrées**

  La solution pour cette situation est de prendre les matrices des poids du model ayant un nombre d'items petits et initialiser avec ces matrices le grand model, pour ne pas commencer l'entrainement du début. Cette opération est appelé Transfer Learning [101].

5.4 Les apportes de notre approche

Lorsque nous avons effectué les tests, nous avons parlé des avantages de notre approche comme le fait de réduire le nombre de passages sur la base de données a un seul passage, ainsi qu'un temps d'exécution optimale, l'étape intermédiaire peut être exploitée pour une meilleure compréhension des résultats.

5.4.1 Exemple illustratif

Prenons un site web de commerce électronique où les achats des clients sont représentés de la façon suivante :

	Article 1	A 2	A 3	A 4	A 5	A 6	A 7	A 8	A 9	A 10
Client 1	0	1	0	0	0	0	0	0	0	0
C 2	0	1	0	0	0	0	0	0	0	0
C 3	0	1	0	0	0	0	0	0	0	0
C 4	0	0	0	1	1	0	0	0	0	0
C 5	0	0	0	1	1	0	0	0	0	0
C 6	0	0	0	1	1	0	0	0	0	0
C 7	1	0	0	0	0	0	0	0	0	0
C 8	0	0	0	0	0	0	0	1	0	0
C 9	0	0	0	0	0	0	0	1	0	0
C 10	0	0	0	0	0	0	0	1	0	0

Figure 5. 17 : Les achats des clients.

Où le client 1 a acheté l'article 2, le client 5 a acheté l'article 4 et 5 et ainsi de suite si on applique un algorithme classique d'extraction des itemsets fréquents on obtient le résultat suivant :

- **Apriori**

Résultats obtenus :

(Article 2) support = 0.3

(Article 8) support = 0.3

(Article 1) support = 0.1

(Article 4, Article 5) support = 0.3

Nous pouvons utiliser le résultat pour recommander l'article 5 pour les clients qui ont acheté l'article 4.

- **Notre approche**

Si on utilise notre approche l'étape intermédiaire donne le résultat suivant (l'ordre des transactions n'est qu'un exemple représentatif)

	Article 1	A 2	A 3	A 4	A 5	A 6	A 7	A 8	A 9	A 10
Client 1	0	1	0	0	0	0	0	0	0	0
C 2	0	1	0	0	0	0	0	0	0	0
C 3	0	1	0	0	0	0	0	0	0	0
C 4	0	0	0	1	1	0	0	0	0	0
C 5	0	0	0	1	1	0	0	0	0	0
C 6	0	0	0	1	1	0	0	0	0	0
C 7	1	0	0	0	0	0	0	0	0	0
C 8	0	0	0	0	0	0	0	1	0	0
C 9	0	0	0	0	0	0	0	1	0	0
C 10	0	0	0	0	0	0	0	1	0	0

Figure 5. 18 : Résultat de l'étape intermédiaire.

L'étape intermédiaire :

Article 2 → {client 1, client 2, client 3,}

Article 8 → {client 8, client 9, client 10,}

Article 4, Article 5 → {client 4, client 5, client 6,}

Article 2 on le négliger il a un support de 0.1

On peut prendre l'avantage de cette segmentation des clients et appliquer l'approche sur les profiles des clients qui sont représentés comme suite :

	homme	femme	mariée	enfants	âge < 18	âge[18-40]	âge > 40
Client 1	1	0	0	0	0	1	0
Client 2	1	0	0	0	0	1	0
Client 3	1	0	0	0	0	1	0

	homme	femme	mariée	enfants	âge < 18	âge[18-40]	âge > 40
Client 8	0	1	0	0	1	0	0
Client 9	0	1	0	0	1	0	0
Client 10	0	1	0	0	1	0	0

	homme	femme	mariée	enfants	âge < 18	âge[18-40]	âge > 40
Client 4	1	0	1	1	0	1	0
Client 5	1	0	1	1	0	1	0
Client 6	1	0	1	1	0	1	0

Figure 5. 19 : Profiles des clients.

Résultats obtenus si on exploite l'étape intermédiaire :

- On peut à titre d'exemple recommander **l'article 2** pour les clients qui sont des **hommes** d'un **âge** entre **18** et **40** ans.
- On peut recommander **l'article 8** pour les clients qui sont des **femmes** d'un **âge** moins de **18** ans.
- On peut faire une **promotion** pour les clients qui **achète l'article 4 et 5 ensembles**, qui sont des **hommes mariés** avec **enfants**.

5.5 Scénarios où il est plus pratique d'utiliser notre approche

1. **Quand on a un nombre d'item important**
   Les tests de temps d'exécution montrent que notre approche a un meilleur temps d'exécution que les méthodes classiques lorsque le nombre d'items est important.
2. **Base de données qui contient beaucoup de répétitions**
   Les bases de données qui contiennent beaucoup de répétitions seront plus compressées, ce qui se traduira par un meilleur temps d'exécution.
3. **Si on veut une meilleure compréhension des résultats**

Cette avantage est expliqué dans les apportes de notre approche.

5.6 Les inconvénients de notre approche

- **Temps d'entrainement important**

  Lorsqu'on travaille avec des transactions de 1000 item le temps d'entrainement va être important et il faut utiliser les GPU pour accélérer l'entrainement.

  Solution :

  Utiliser des modèles qui sont déjà entrainé [102] ou utiliser les matrices de poids de ces modèles pour initialiser un nouveau model [101].

- **Représentation sparce de la base de données**

  Les modèles de deep learning suppose une représentation vectorielle de donnée. Dans le cas d'une séquence de longueur variable, cela nécessite que les données soient transformées de telle sorte que chaque séquence ait la même longueur [103] cela peut conduire à une utilisation de la mémoire qui est plus que nécessaire par rapport à la représentation normal.

- **Les réseaux de neurones profonds sont des boites noires**

  Nous sommes confrontés à ce problème lorsque nous essayons de donner un sens à l'espace latent dans le but de l'exploiter un peu plus, parce que le nombre de paramètres et de variables est important.

  Solution :

  Visualiser l'espace latent pour une meilleure compréhension des résultats [104].

6. Conclusion

Dans ce chapitre nous avons présenté l'environnement matériel et l'environnement logiciel, les bibliothèques utilisées pour implémenter notre approche. On a effectué des tests pour vérifier la qualité des résultats de notre approche et son l'efficacité par rapport à un algorithme classique.

Dans la discussion nous avons décrit pourquoi notre approche donne des performances supérieures. On a essayé de montrer les apportes de notre approche et répondu à la question posée dans l'introduction de ce chapitre.

Conclusion générale et Perspectives

Conclusion générale et Perspectives

1. Conclusions

L'extraction des motifs fréquents à partir des données reste toujours l'un des sujets les plus importants. La communauté scientifique voue un dévouement tout aussi important à la recherche et le développement de nouveaux outils et méthode de traitement. Tout au long de notre travail nous avons vu les différentes méthodes d'extraction d'itemsets fréquents. Nous avons ensuite exploré et étudié les différentes structure et architectures de Deep Learning.

Et enfin nous avons décrit notre solution qui se base sur la création et l'entrainement d'un auto-encodeur qui a pour premier fonction de compresser les données, et puis qui est capable d'extraire les motifs fréquents d'une base de données. Notre modèle nous permet de parcourir la base de données qu'une seule fois ce qui rend le processus plus rapide et plus efficace que d'autres techniques disponibles.

Les résultats de nos nombreux tests ont montré que le développement de nouvelle méthode basée sur le principe du Deep Learning est un chemin sur lequel on peut s'y attarder.

Ce travail nous a permis d'approfondir nos connaissances sur la notion d'extraction d'itemsets fréquents, et d'apprendre de nouvelles techniques concernent le Deep Learning.

L'étude effectuée représente une ébauche à des solutions qui peuvent être proposées, certes elle est efficace dans un certain contexte de l'extraction des motifs fréquents tout en apportant des nouveautés comme par exemple une meilleure compréhension des résultats obtenus par la phase de compression.

2. Perspectives

Les perspectives de ce travail préliminaire sont nombreuses. Nous terminons notre travail en dégagent quelques perspectives :

- Aborder et gérer le sujet de masse de données (Big Data), nous souhaitons poursuivre l'optimisation de notre approche en combinant les concepts de Big Data (Hadoop, HDFs, Spark) avec les concepts de deep learning. La combinaison de ces deux concepts pourrait encore réduire le coût de l'extraction des motifs fréquents.

- Adopter notre approche pour traiter les données incertaines. Le problème des valeurs imparfaites (manquantes, imprécises...) est un problème connu dans le domaine de la fouille de données et de l'apprentissage automatique, on peut utiliser une architecture appelée Denoising autoencoder pour traiter ces données, il sera intéressant de voir comment cette méthode pourrai traiter ces données.
- Aborder le sujet des itemsets pertinents, particulièrement les itemsets rares, alors que la littérature a été presque exclusivement centrée sur des itemsets fréquents, dans de nombreuses situations pratiques, les itemsets rares présentent un intérêt supérieur (par exemple, dans les bases de données médicales, des combinaisons de symptômes rares peuvent fournir des indications utiles aux médecins), on pourrai développer des algorithmes pour exploiter l'espace latent et essayer de trouver ces itemset, on peut filtrer par rapport au support par exemple.

Annexes

Annexe : Détails sur d'implémentation d'un prototype de la méthode proposée

1. Installation des dépendances

Les dépendances suivantes sont nécessaires :

- **Python**

  Pour installer python il faut installer anaconda depuis :

  https://www.anaconda.com/download/

- **TensorFlow**

  Bibliothèque de deep learning de google, pour installer TensorFlow il faut taper la commande suivante dans le terminale :

  ```
  conda install -c conda-forge tensorflow
  ```

- **Pandas**

  Bibliothèque de manipulation des données pour installer pandas il faut taper :

  ```
  conda install -c anaconda pandas
  ```

- **Numpy**

  Bibliothèque de calcule matriciel, pour installer Numpy il faut taper :

  ```
  conda install -c anaconda numpy
  ```

- **Mlxtend**

  Bibliothèque d'apprentissage automatique :

  ```
  conda install -c conda-forge mlxtend
  ```

- **Bokeh**

  Bibliothèque de visualisation des données :

  ```
  conda install -c bokeh bokeh
  ```

2. Manipulation de données avec pandas

- **Lire CSV**

```
import pandas as pd
df = pd.read_csv('mushrooms.csv')
df.head()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a

- **Générer des données**

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.choice([0, 1], size=[1000, 50], p=[.6, .4]))
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43	44	45	46	47	48	49
0	0	0	0	1	0	1	0	0	1	0	...	1	1	1	1	0	0	1	1	1	0
1	0	0	1	0	0	0	0	0	0	0	...	0	1	0	1	0	0	0	1	1	0
2	0	1	0	0	0	1	0	0	0	0	...	1	0	1	0	0	0	0	0	0	1
3	1	1	0	1	1	0	1	0	1	1	...	0	0	0	0	0	1	1	0	0	0
4	0	0	0	0	1	1	0	0	1	0	...	1	1	0	0	1	1	0	0	0	1

3. Création du model avec TensorFlow

```
"les parametres de réseau"
num_inputs = nb_item "nombre de neurones dans l'entrées"
hidden_neu_1 = nb_item // 2 "nombre de neurones dans la 2eme couche"
hidden_neu_2 = 1 "nombre de neurones dans z"
hidden_neu_3 = hidden_neu_1 "nombre de neurones dans la 4eme couche"
num_outputs = num_inputs "nombre de neurones dans la sortie"
learning_rate = 0.01 "taux d'apprentissage"

X = tf.placeholder(tf.float32, shape=[None,num_inputs]) "place holder pour les données"
activ_func = tf.nn.sigmoid "fonction d'activation"
z_activ_func = tf.nn.relu "fonction d'activation de z"
"encoder"
def encoder(X):
    with tf.variable_scope("encoder", reuse=tf.AUTO_REUSE):
        hidden_layer_1 = tf.layers.dense(inputs=X, units=hidden_neu_1, activation=activ_func)
        z  = tf.layers.dense(inputs=hidden_layer_1, units=hidden_neu_2, activation=activ_func)
    return z
"decoder"
def decoder(Z):
    with tf.variable_scope("decoder", reuse=tf.AUTO_REUSE):
        hidden_layer_3 = tf.layers.dense(inputs=Z, units=hidden_neu_3, activation=activ_func)
        output_layer = tf.layers.dense(inputs=hidden_layer_3, units=num_outputs, activation=activ_func)
    return output_layer
"la connection entre l'encoder et le decoder"
sampled = encoder(X)
dec = decoder(sampled)
"la fonction de cout"
loss = tf.reduce_mean(tf.square(dec - X))
"optimisateure"
optimizer = tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(loss)
```

4. Entraînement et sauvegarde du model avec TensorFlow

```
num_steps = 5000

with tf.Session() as sess:
    sess.run(initializer)

    for iteration in range(num_steps+1):
        outputs = sess.run([train, loss],feed_dict={X:df})
        if iteration % 1000 == 0:
            print('Iter: {}'.format(iteration))
            print('Loss: {:.4}'. format(outputs[1]))
            print()
    saver.save(sess, './save/autoencoder_for_frequent_itemsets.ckpt')
```

5. déploiement et sauvegarde du model avec TensorFlow

```
dic_sous_itemset = {}
dic_itemset = {}

itemsets = test_df.values

with tf.Session() as sess:
    saver.restore(sess, './save/autoencoder_for_frequent_itemsets.ckpt')

    latent_rep = sess.run(sampled, feed_dict={X:test_df})


nb_trans = len(latent_rep)

i = 0
for itemset in itemsets:
    key = latent_rep[i][0]


    if key in dic_sous_itemset:
        dic_sous_itemset[key].support += 1/nb_trans
    else:
        dic_sous_itemset[key] = Sous_itemset((1/nb_trans),
                                             [j * chr(65 + i)  for i, j in enumerate(itemset) if j])

    i += 1

compressed_tran = []
compressed_tran_sup = []

for key, value in dic_sous_itemset.items():
    compressed_tran.append(value.itemset)
    compressed_tran_sup.append(value.support)
```

Bibliographie

[1] Claude Issa Nombré, Konan Brou, Kouadio Kimou. aloa2i: optimisation d'extraction des k- itemsets frequents (pour k <= 2) aloa2i: optimization of extraction k-itemsets frequent (for k <= 2). Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées, INRIA, 2016, 18. <Hal-01386948>

[2] Plantevit, M., Motifs, E. De, Dans, S., Données, D., & Plantevit, M. (2008). Marc Plantevit To cite This version : HAL Id : tel-00319242 Thèse Extraction De Motifs Séquentiels Dans Des Données Multidimensionnelles.

[3] Sourav S. Bhowmick Qiankun Zhao, "Association Rule Mining: A Survey," Nanyang Technological University, Singapore, 2003.

[4] Dr. Saed Sayad. Association Rules: Machine learning expert @ University of Toronto. **In** : saedsayad.com **[en ligne]** Mis en ligne 2010-2018. http://www.saedsayad.com/association_rules.htm (Consulté le 10/03/2018).

[5] Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: Current status and future directions. Data Mining and Knowledge Discovery, 15(1), 55-86.

[6] Khaled, tannir: CEO - Senior Big Data Architect & Trainer @ dataXper, Montreal, Canada. Algorithms, Apriori, Data Mining. **In** : khaledtannir.net **[en ligne]** Mis en ligne 3 May 2011. http://blog.khaledtannir.net/2011/05/apriori/ (Consulté le 10/03/2018).

[7] H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri R. Agrawal, "Fast Discovery of Association Rules," in Advances in Knowledge Discovery and Data Mining, 1996, pp. 307-328.

[8] M. Chen, and P.S. Yu J.S. Park, "An Effective Hash Based Algorithm for Mining Association Rules," in ACM SIGMOD Int'l Conf. Management of Data, May, 1995.

[9] Jian Pei, Jiawei Han, "Mining Frequent patterns without candidate generation," in SIGMOD '00 Proceedings of the 2000 ACM SIGMOD international conference on Management of data, New York, NY, USA, 2000, pp. 1-12.

[10] Nasreen, S., Azam, M. A., Shehzad, K., Naeem, U., & Ghazanfar, M. A. (2014). Frequent pattern mining algorithms for finding associated frequent patterns for data streams: A survey. Procedia Computer Science, 37, 109-116.

[11] Han, J., Pei, J., & Yin, Y. (2000). Frequent Pattern Tree: Design and Construction. Networks, 1-12. https://doi.org/10.1145/342009.335372.

[12] Khaled, tannir: CEO - Senior Big Data Architect & Trainer @ dataXper, Montreal, Canada. L'algorithme FP-Growth – Construction du FP-tree (2/3). **In** : khaledtannir.net **[en Ligne]** Mis en ligne 16 July 2012 http://blog.khaledtannir.net/2012/07/lalgorithme-fp-growth-construction-du-fp-tree-23/ (Consulté le 10/03/2018).

[13] kp, M., Singh, R. K., & Kumar, S. S. (2012). Apriori-hybrid algorithm as a tool for colon cancer microarray data classification. International Journal of Engineering Research and Development, 4(7), 53-57.

[14] Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W., 1997, Parallel Algorithms for Discovery of Association Rules, Data Mining and Knowledge Discovery, 1(4), pp. 343-373.

[15] Moens, S., Aksehirli, E., and Goethals, B., 2013, Frequent Itemset Mining for Big Data, Proc. International Conference on Big Data, Santa Clara, California, pp. 111-118.

[16] Chen, M., GAO, X. D., & Li, H. F. (2009). An efficient parallel FP-Growth algorithm. CyberC 2009 - International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 283-286. https://doi.org/10.1109/CYBERC.2009.5342148.

[17] Pei, J., Han, J., Lu†, H., Nishio, S., Tang, S., & Yang, D. (2007). H-Mine: Fast and space-preserving frequent pattern mining in large databases. IIE Transactions, 39(6), 593-605. https://doi.org/10.1080/07408170600897460.

[18] Kriesel, D. (2005). A Brief Introduction to Neural Networks. Retrieved August, 244.https://doi.org/10.1016/0893-6080 (94)90051-5.

[19] Nilsson, N. J. (2005). Introduction to Machine Learning. Machine Learning, 56(2), 387-399. https://doi.org/10.1016/j.neuroimage.2010.11.004.

[20] Du, K.-L., & Swamy, M. N. S. (2014). Neural Networks and Statistical Learning. https://doi.org/10.1007/978-1-4471-5571-3.

[21] Jason Brownlee: Founding Researcher of machine learning mastery / Senior Software Engineer @ Bureau of Meteorology, Melbourne, Australia. A Tour of Machine Learning Algorithms. **In:** machine Learning mastery **[en ligne]** Mis en ligne le 25 Novembre 2013.https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/ (Consulté le 06/03/2018).

[22] Python Programming By Anthony S.Williams 2017.

[23] Gérard DREYFUS: Professor of Machine Learning @ École Supérieure de Physique et de Chimie Industrielles, Paris. RÉSEAUX DE NEURONES. **In :** Encyclopædia Universalis **[en ligne]**.http://www.universalis.fr/encyclopedie/reseaux-de-neurones-formels/ (Consulté le 21/03/2018).

[24] Murphy, C., Gray, P., & Stewart, G. (2017). Verified perceptron convergence theorem. Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages - MAPL 2017, 43-50. https://doi.org/10.1145/3088525.3088673.

[25] Karol,G.,Ivo,D.,Alex,G.,Danilo,R.,and Daan,W.(2015).Draw: A recurrent neural network for image generation. pp. 1462-1471.

[26] Python Machine Learning Second Edition (2017), By Sebastian Raschka and Vahid Mirjalili.

[27] Harsh Singhal: Principal Data Scientist @ Palo Alto Networks, SANTA CLARA, California. Convolutional Neural Network with TensorFlow implementation **In:** Data Science Group, IITR (Medium) **[en ligne]** Mis en ligne le 17 Juin 2017 https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117 (Consulté le 21/03/2018).

[28] Ujjwal Karn: Machine Learning expert @ Facebook, Singapore. An Intuitive Explanation of Convolutional Neural Networks **In:** ujjwalkarn.me **[en ligne]** Mis en ligne le 11 Aout 2016 https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (Consulté le 21/03/2018).

[29] Yu Huang: President and Chief Scientist of Autonomous Driving @ Singulato, USA. Visual Detection, Recognition and Tracking with Deep Learning **In:** slideshare.net **[en ligne]** Mis en ligne le 28 December 2016 https://www.slideshare.net/yuhuang/visual-detection-recognition-and-tracking-with-deep-learning (Consulté le 21/03/2018).

[30] Réseaux de neurones récurrents et mémoire : application à la musique, Tristan Stérin, École Normale Supérieure de Lyon, Université Claude Bernard Lyon I.

[31] Dey, R., & Salemt, F. M. (2017). Gate-variants of Gated Recurrent Unit (GRU) neural networks. Midwest Symposium on Circuits and Systems, 2017-August, 1597-1600. https://doi.org/10.1109/MWSCAS.2017.8053243.

[32] Manish, Chablani: Machine Learning Researcher @ Curai / Software Engineer and Machine Learning expert @ Uber. GAN—Introduction and Implementation. **In** : Towards Data Science **[en ligne]** Mis en ligne le 27 Jun 2017. https://towardsdatascience.com/gan-introduction-and-implementation-part1-implement-a-simple-gan-in-tf-for-mnist-handwritten-de00a759ae5c (Consulté le 06/03/2018).

[33] Naoki, Shibuya: Research Engineer @ Ascent Robotics Inc, Tokyo japan. Understanding Generative Adversarial Networks **In**: Towards Data Science **[en ligne]** Mis en ligne le 03 Novembre 2017. https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef (Consulté le 06/03/2018).

[34] Arden, Dertat: Senior Research Engineer @ Netflix. Applied Deep Learning - Part 3: Autoencoders. **In** : Towards Data Science **[en ligne]** Mis en ligne le 03 Octobre 2017. https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798 (Consulté le 06/03/2018).

[35] Irhum, Shafkat: Rajuk Uttara Model College, Bangladesh. Intuitively Understanding Variational Autoencoders. **In** : Towards Data Science **[en ligne]** Mis en ligne le 04 février 2018. https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf (Consulté le 06/03/2018).

[36] Dominic Monn: Deep Learning Engineer @ Loom.ai, Switzerland. Denoising Autoencoders explained. **In** : Towards Data Science **[en ligne]** Mis en ligne le 17 juillet 2017. https://towardsdatascience.com/denoising-autoencoders-explained-dbb82467fc2 (Consulté le 26/05/2018).

[37] Adil Baaj: Data Scientist @ Sicara, Paris, France. Keras Tutorial: Content Based Image Retrieval Using a Convolutional Denoising Autoencoder. **In**: blog.sicara.com **[en ligne]** Mis en ligne le 14 septembre 2017. https://blog.sicara.com/keras-tutorial-content-based-image-retrieval-convolutional-denoising-autoencoder-dc91450cc511 (Consulté le 26/05/2018).

[38] Nahua Kang: Product Intern @ Twenty Billion Neurons GmbH, Berlin, Germany. Introducing Deep Learning and Neural Networks—Deep Learning for Rookies (1). **In**: Towards Data Science **[en ligne]** Mis en ligne le 18 Juin 2017. https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-for-rookies-1-bd68f9cf5883 (Consulté le 26/05/2018).

[39] Shi Yan: Senior Software Engineer @ NVIDIA, California, USA. Understanding LSTM and its diagrams. **In**: Towards Data Science **[en ligne]** Mis en ligne le 16 Mars 2016. https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714 (Consulté le 27/05/2018).

[40] Jean-Marc Valin: Senior Platform Developer @ Mozilla, Montreal, Canada. RNNoise Learning Noise Suppression. **In**: xiph.org **[en ligne]** Mis en ligne le 17 Septembre 2017. https://people.xiph.org/~jm/demo/rnnoise/ (Consulté le 27/05/2018).

[41] Leonardo Araujo dos Santos: Author @ gitbook.com. Artificial intelligence/Recurrent Neural Networks. **In**: leonardoaraujosantos.gitbooks.io **[en ligne]** Mis en ligne le 27 Janvier 2018 https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/recurrent_neural_networks.html (Consulté le 27/05/2018).

[42] William Wong: Senior Technology Editor @ Informa, Pennsylvania, USA. What's the Difference Between Machine Learning Techniques. **In**: electronicdesign.com **[en ligne]** Mis en ligne le 27 Avril 2017 http://www.electronicdesign.com/automotive/what-s-difference-between-machine-learning-techniques (Consulté le 28/05/2018).

[43] Calum McClelland: Director of Projects @ Leverege / Managing Editor @ IoT For All, Baltimore, USA. The Difference Between Artificial Intelligence, Machine Learning, and Deep Learning **In**: medium.com **[en ligne]** Mis en ligne le 4 December 2017 https://medium.com/iotforall/the-difference-between-artificial-intelligence-machine-learning-and-deep-learning-3aa67bff5991 (Consulté le 28/05/2018).

[44] George Seif: Machine Learning Engineer @ indus.ai / Researcher @ Ryerson University, Toronto, Canada. I'll tell you why Deep Learning is so popular and in demand **In:** medium.com **[en ligne]** Mis en ligne le 6 Janvier 2018 https://medium.com/swlh/ill-tell-you-why-deep-learning-is-so-popular-and-in-demand-5aca72628780 (Consulté le 28/05/2018).

[45] Andrej Karpathy: Director of AI @ Tesla / Research Scientist @ OpenAI San Francisco, California USA. Deep Reinforcement Learning: Pong from Pixels **In:** Andrej Karpathy blog **[en ligne]** Mis en ligne le 31 Mai 2016 http://karpathy.github.io/2016/05/31/rl/ (Consulté le 28/05/2018).

[46] Vishal Maini: Research Content Manager @ DeepMind, London, United Kingdom. Machine Learning for Humans, Part 4: Neural Networks & Deep Learning **In:** Machine Learning for Humans **[en ligne]** Mis en ligne le 19 Aout 2017 https://medium.com/machine-learning-for-humans/neural-networks-deep-learning-cdad8aeae49b (Consulté le 28/05/2018).

[47] Brad Nemire: Developer Communications Manager @ NVIDIA, San Francisco, USA. Deep Learning Demystified **In:** Nvidia Developer **[en ligne]** Mis en ligne le 6 Mars 2018 https://news.developer.nvidia.com/on-demand-webinar-deep-learning-demystified/ (Consulté le 28/05/2018).

[48] Dishashree Gupta: Machine Learning/ Deep Learning Researcher @ Analytics Vidhya. Fundamentals of Deep Learning – Activation Functions and When to Use Them. **In**: analyticsvidhya.com **[en Ligne]** Mis en ligne 23 Octobre 2017 https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/ (Consulté le 19/05/2018).

[49] Sagar Sharma: editor of Arduino Community @ Mediums. Activation Functions: Neural Networks. **In** : Towards Data Science **[en Ligne]** Mis en ligne 6 Septembre 2017 https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (Consulté le 19/05/2018).

[50] Dr Sebastian Raschka: Assistant Professor of Statistics @ University of Wisconsin-Madison. Activation Functions for Artificial Neural Networks. **In** : github.io **[en Ligne]** Mis en ligne Mai 2016 https://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/ (Consulté le 28/05/2018).

[51] Andrew Ng: Founder and CEO of Landing.AI / Managing Director of AI Fund / Founder of deeplearning.ai / Adjunct Professor @ Stanford. Self-driving cars are here. **In** : medium **[en Ligne]** Mis en ligne 7 Mai 2018 https://medium.com/@andrewng/self-driving-cars-are-here-aea1752b1ad0 (Consulté le 28/05/2018).

[52] Jiajun Zhang, C. Z. (2015). Neural Networks in Machine Translation: An Overview. IEEE Intelligent Systems, 1724-1734. https://doi.org/10.1109/MIS.2015.69.

[53] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. https://doi.org/10.3115/v1/D14-1179.

[54] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks, 1-9. https://doi.org/10.1007/s10107-014-0839-0.

[55] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems, 1-9. https://doi.org/http://dx.doi.org/10.1016/j.protcy.2014.09.007.

[56] Szegedy, C. (2013). Deep Neural Networks for Object Detection. Nips 2013, 1-9. https://doi.org/10.1109/CVPR.2014.276.

[57] Erhan, D., Szegedy, C., Toshev, A., & Anguelov, D. (2014). Scalable Object Detection Using Deep Neural Networks. Cvpr, 2155-2162. https://doi.org/10.1109/CVPR.2014.276.

[58] Vicente Oswaldo Baez Monroy , Neural Networks as Artificial Memories for Association Rule Mining ,The university of York , December 2006.

[59] Sallans, B. (1997). Data mining for association rules with unsupervised neural networks: Csc final project.

[60] Gupta, G., Strehl, A., and Ghosh, J. (1999). Distance based clustering of association rules

[61] Amit Bhagat,Feed Forward Neural Network Algorithm for Frequent Patterns Mining,Maulana Azad National Institute of Technology, Bhopal (M.P.)462051, India,2010.

[62] Mining Frequent Itemsets without Candidate Generation using Optical Neural Network, Divya Bhatnagar Jodhpur National University Jodhpur, Rajasthan, India, AIT 2011.

[63] Shivanandam, S. N., Sumathi, S., Deepa, S. N.: "Introduction to Neural Network using MATLAB 6.0. TATA Mc.Graw Hill."

[64] Nikolay N. Evtihiev, Rostislav S. Starikov, Boris N. Onyky, Vadim V. perepelitsa, Igor B. Scherbakov 1994: Experimetal investigation of the performance of the optical two-layer neural network, SPIE Vol. 2430 Optical Neural Networks, pages 189-197.

[65] E. Horward, Michel, Abdul A. S. Awwal: Analysis and Evaluation of Electro-Optic Artificial Neural Network Performance in the Presence of Non-Ideal Components.

[66] Damien Woods, Thomas J. Naughton 2009: Optical Computing.

[67] R. Rojas 1996: Neural Networks. Springer-Verlag, Berlin.

[68] D. Bhatnagar, A. K. Saxena 2011: An Optical Neural Network Model for mining Frequent Itemsets in Large Databases, Indian Journal of Computer Science and Engineering, Vol 2, No. 2, pages 212-217.

[69] Valentine, A. P., & Trampert, J. (2012). Data space reduction, quality assessment and searching of seismograms: Autoencoder networks for waveform data. Geophysical Journal International, 189(2), 1183-1202. https://doi.org/10.1111/j.1365-246X.2012.05429.x.

[70] Goodfellow, I. (s. d.). Deep Learning.

[71] Neerja Doshi: MS Data Science student @ University of San Francisco. Deep Learning Best Practices —Weight Initialization. **In**: Towards Data Science **[en Ligne]** Mis en ligne 26 Mars 2018 https://towardsdatascience.com/deep-learning-best-practices-1-weight-initialization-14e5c0295b94 (Consulté le 16/05/2018).

[72] Aditya Ananthram. Random Initialization for Neural Networks: A Thing Of The Past. **In** : Towards Data Science **[en Ligne]** Mis en ligne 25 février 2018 https://towardsdatascience.com/random-initialization-for-neural-networks-a-thing-of-the-past-bfcdd806bf9e (Consulté le 19/05/2018).

[73] Jeremy Jordan: Machine Learning Consultant @ Udacity. Neural networks: training with Backpropagation. **In** : jeremyjordan.me **[en Ligne]** Mis en ligne 18 Juliet 2017 https://www.jeremyjordan.me/neural-networks-training/ (Consulté le 16/05/2018).

[74] Jeremy Jordan: Machine Learning Consultant @ Udacity. Setting the learning rate of your neural network. **In** : jeremyjordan.me **[en Ligne]** Mis en ligne 1 Mars 2018 https://www.jeremyjordan.me/nn-learning-rate/ (Consulté le 16/05/2018).

[75] Jeremy Jordan: Machine Learning Consultant @ Udacity. Gradient descent. **In** : jeremyjordan.me **[en Ligne]** Mis en ligne 17 Juliet 2017 https://www.jeremyjordan.me/gradient-descent/ (Consulté le 17/05/2018).

[76] Conor McDonald: Data Scientist @ Amazon, PhD, Economics (University of Leeds). Machine learning Fundamentals (I): Cost functions and gradient descent. **In**: Towards Data Science **[en Ligne]** Mis en ligne 27 Novembre 2017 https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220 (Consulté le 17/05/2018).

[77] Dishashree Gupta: Machine Learning/ Deep Learning Researcher @ Analytics Vidhya. Fundamentals of Deep Learning – Activation Functions and When to Use Them. **In**: analyticsvidhya.com **[en Ligne]** Mis en ligne 23 Octobre 2017 https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/ (Consulté le 19/05/2018).

[78] Sagar Sharma: editor of Arduino Community @ Mediums. Activation Functions: Neural Networks. **In** : Towards Data Science **[en Ligne]** Mis en ligne 6 Septembre 2017 https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (Consulté le 19/05/2018).

[79] Anish Singh Walia: Author @ DataScience+ / Technical Member @ Microsoft Innovation Center — VIT Vellore. Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent. **In** : Towards Data Science **[en Ligne]** Mis en ligne 10 Juin 2017 https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f (Consulté le 20/05/2018).

[80] Vadim Smolyakov: Data Scientist @ Shopify / Research Assistant @ Massachusetts Institute of Technology. Neural Network Optimization Algorithms. **In** : Towards Data

Science **[en Ligne]** Mis en ligne 10 Janvier 2018 https://towardsdatascience.com/neural-network-optimization-algorithms-1a44c282f61d (Consulté le 20/05/2018).

[81] Camron Godbout: Researcher @ University of South Carolina / Co-founder & CTO @ Apteo. Deep Learning Cheat Sheet. **In** : Hacker Noon **[en Ligne]** Mis en ligne 6 Décembre 2016 https://hackernoon.com/deep-learning-cheat-sheet-25421411e460 (Consulté le 21/05/2018).

[82] Andy Thomas: engineer @ adventuresinmachinelearning.com. Stochastic Gradient Descent – Mini-batch and more. **In** : Adventures In Machine Learning **[en Ligne]** Mis en ligne 30 Mars 2017 http://adventuresinmachinelearning.com/stochastic-gradient-descent/ (Consulté le 21/05/2018).

[83] Wojciech Marian Czarnecki: Senior Research Scientist @ google DeepMind, London, UK. What is the difference between SGD and Backpropagation **In**: stackoverflow.com **[en Ligne]** Mis en ligne 21 Juin 2016 https://stackoverflow.com/questions/37953585/what-is-the-diffirence-between-sgd-and-back-propogation?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa (Consulté le 21/05/2018).

[84] Anish Singh Walia: Author @ DataScience+ / Technical Member @ Microsoft Innovation Center — VIT Vellore. The Vanishing Gradient Problem **In** Medium **[en Ligne]** Mis en ligne 1 Juin 2017 https://medium.com/@anishsingh20/the-vanishing-gradient-problem-48ae7f501257 (Consulté le 22/05/2018).

[85] Julien Despois: Deep Learning Engineer @ L'Oréal / Creator of AI-Odyssey.com. Memorizing is not learning!— 6 tricks to prevent overfitting in machine learning. **In** : Hacker Noon **[en Ligne]** Mis en ligne 22 Mars 2018 https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42 (Consulté le 22/05/2018).

[86] Sachin Joglekar: Engineer @ Google. Overfitting and Human Behavior **In:** Medium **[en Ligne]** Mis en ligne 24 Janvier 2018 https://medium.com/@srjoglekar246/overfitting-and-human-behavior-5186df1e7d19 (Consulté le 22/05/2018).

[87] Fei-Fei Li: Chief Scientist of Artificial intelligence/ML @ Google. CS231n Convolutional Neural Networks for Visual Recognition **In:** Stanford université CS231n lecture notes **[en Ligne]** Mis en ligne Mars 2018 http://cs231n.github.io/neural-networks-2/#init (Consulté le 22/05/2018).

[88] Glorot, Xavier., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Pmlr, 9, 249-256. https://doi.org/10.1.1.207.2059.

[89] Das, K., Jiang, J., & Rao, J. N. K. (2004). Mean squared error of empirical predictor. Annals of Statistics, 32(2), 818-840. https://doi.org/10.1214/009053604000000201.

[90] Mindfire Solutions: software service provider. Advantages and Disadvantages of Python Programming Language. **In** : Medium **[en Ligne]** Mis en ligne 24 Avril 2017

https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121 (Consulté le 2/06/2018).

[91] David Robinson: Chief Data Scientist @ Data Camp. The Incredible Growth of Python. **In** : stackoverflow.blog **[en Ligne]** Mis en ligne 6 Septembre 2017 https://stackoverflow.blog/2017/09/06/incredible-growth-python/ (Consulté le 2/06/2018).

[92] The Jupyter Notebook **In** : jupyter.org **[en Ligne]** http://jupyter.org/ (Consulté le 2/06/2018).

[93] pandas **In** : pandas.pydata.org **[en Ligne]** http://pandas.pydata.org/pandas-docs/version/0.23/ (Consulté le 2/06/2018).

[94] About TensorFlow **In** : tensorflow.org **[en Ligne]** https://www.tensorflow.org/ (Consulté le 2/06/2018).

[95] Fabrizio Milo: Deep learning Engineer and Architect @ H2O.ai. What happened at the TensorFlow Dev Summit 2017 - Part 1/3: Community & Applications **In**: Towards Data Science **[en Ligne]** Mis en ligne 19 février 2017 https://towardsdatascience.com/what-happened-at-the-tensorflow-dev-summit-2017-part-1-3-community-applications-77fb5ce03c52 (Consulté le 2/06/2018).

[96] Daniely, A. (2014). From average case complexity to improper learning complexity.

[97] Kasper fredenslund: Computational Complexity of Neural Networks. **In** : kasperfred.com **[en Ligne]** Mis en ligne 25 Mars 2018 https://kasperfred.com/posts/computational-complexity-of-neural-networks#fnref3 (Consulté le 13/06/2018).

[98] Kasper fredenslund: Big-O matrix multiplication: Just How Fast Is Your Algorithm? **In** : kasperfred.com **[en Ligne]** Mis en ligne 24 Mars 2018 https://kasperfred.com/posts/just-how-fast-is-your-algorithm#big-o-matrix-multiplication (Consulté le 13/06/2018).

[99] Dr Sebastian Raschka: Assistant Professor of Statistics @ University of Wisconsin-Madison / Data Scientist. Mlxtend Machine Learning Library Extension. **In** : Anaconda cloud **[en Ligne]** Mis en ligne Mai 2016 https://anaconda.org/conda-forge/mlxtend (Consulté le 16/06/2018).

[100] Jeff Schlimmer : Mushroom Data Set **In** : UCI Machine Learning Repository **[en Ligne]** Mis en ligne 27 avril 1987 https://archive.ics.uci.edu/ml/datasets/mushroom (Consulté le 17/06/2018).

[101] Jason Brownlee: Founding Researcher of machine learning mastery / Senior Software Engineer @ Bureau of Meteorology, Melbourne, Australia. A Gentle Introduction to Transfer Learning for Deep Learning **In:** machine Learning mastery **[en ligne]** Mis en ligne le 20 December 2017 https://machinelearningmastery.com/transfer-learning-for-deep-learning/ (Consulté le 17/06/2018).

[102] the Apache Software Foundation. MXNet Model Zoo **In:** mxnet.incubator.apache.org **[en ligne]** Mis en ligne 2015 https://mxnet.incubator.apache.org/model_zoo/index.html (Consulté le 18/06/2018).

[103] Jason Brownlee: Founding Researcher of machine learning mastery / Senior Software Engineer @ Bureau of Meteorology, Melbourne, Australia. Data Preparation for Variable Length Input Sequences **In:** machine Learning mastery **[en ligne]** Mis en ligne le 19 juin 2017 https://machinelearningmastery.com/data-preparation-variable-length-input-sequences-sequence-prediction/ (Consulté le 18/06/2018).

[104] Julien Despois: Deep Learning Engineer @ L'Oréal / Creator of AI-Odyssey.com. Latent space visualization—Deep Learning bits #2. **In** : Hacker Noon **[en Ligne]** Mis en ligne 24 février 2017 https://hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df (Consulté le 18/06/2018).

[105] Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta. Introduction to Parallel Computing (2nd Edition). Addison Wesley, second edition, 2003.

[106] Lynch, N. A., & Patt-Shamir, B. (1993). Distributed Algorithms, (January), 1-440.

[107] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large Clusters. Communications of the ACM, 51(1):107–113, January 2008.