République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: *Modèle Intelligent et Décision(M.I.D)*

Thème

Deep Learning pour la classification des images

Réalisé par :

- **Moualek Djaloul Youcef**

Présenté le 3 Juillet 2017 devant le jury composé de MM.

- **Hadjila Fethallah** *(Président)*
- **Benazzouz Mourtada** *(Encadreur)*
- **El Habib Daho Mostafa** *(Examinateur)*
- **Nessma Settouti** *(Examinatrice)*

Année universitaire : 2016-2017

DEDICACES ET REMERCIEMENTS

LA FAMILLE, LES AMIS, LES ENSEIGNANTS, LES COLLEGUES

Table des matières

Table des figures

Liste des tableaux

Acronymes

ADAM Adaptive Moments.

BBTT Backpropagation Through Time.

CNN Convolutional Neural Network.

ELU Exponential Linear Units.

GAP Global Average Pooling.
GRU Gated Recurrent Unit.

IA Intelligence Artificielle.

KNN K-Nearest Neighbors.

L-BFGS Limited-Broyden–Fletcher–Goldfarb–Shanno.
LSTM Long Short Term Memory.

MCP McCulloch–Pitts.
ML Machine Learning.
MLP Multi Layer Perceptron.

NAG Nesterov's Accelerated Gradient.

ReLu Rectified Linear Unit.
RNN Recurrent Neural Network.

SGD Stochastic Gradient Descent.
SVM Support Vector Machine.

tanh Hyperbolic tangent.

Introduction générale

1 Introduction

Intelligence Artificielle (IA) est l'un des domaines les plus récents de la science et de l'ingénierie. Les travaux ont sérieusement débute après la seconde guerre mondiale, et le nom lui même a été inventé en 1956. Régulièrement cité comme "domaine ou j'aimerai bien y être" par les scientifiques dans d'autres disciplines.

Un étudiant en physique peut raisonnablement se dire que toutes les bonnes idées ont été trouvées par Galilée, Newton, Einstein et le reste. De l'autre coté, tout reste ouvert en IA.

Historiquement, quatre approches de l'IA ont été suivies, chacune par des gens différents avec des méthodes différentes. Une approche axée sur l'homme doit être en partie une science empirique, impliquant observation et hypothèse sur le comportement humain. Une approche rationnelle implique une combinaison de mathématique et d'ingénierie . Les différents groupes se sont décriés et se sont aides mutuellement, voici les quatre approches :

— **Acting humanly** : Le test de Turing "L'art de créer des machines qui exécutent des fonctions requérant une intelligence lorsqu'elles sont exécutées par des êtres humains" [2]
— **Thinking humanly** : La modélisation cognitive "L'excitent nouveau défi de construire des ordinateurs qui pensent, des machines avec des consciences, au sens propre comme au sens figuré "[3]
— **Thinking rationally** : Les lois de la pensée "L'étude des calculs qui rendent possibles la perception, le raisonnement et les actes"[4]
— **Acting rationally** : les agents rationnels "L'IA est l'étude de la conception des agents intelligents"[5]

L'IA englobe plusieurs sous domaines allant du plus général (apprentissage et perception) au plus spécifique, comme jouer aux échecs, démontrer des théorèmes mathématiques, écrire des poèmes, conduire une voiture ou diagnostiquer des maladies. L'IA se révèle être utile dans toutes les taches intellectuelles. C'est vraiment un domaine universel et pluri-disciplinaire.

2 problématique

Le but de la recherche en IA est de créer une technologie qui permette aux ordinateurs et aux machines de fonctionner d'une manière intelligente. Le problème général de la création d'une intelligence a été divisé en plusieurs sous problèmes. Celles-ci consistent en des capacités que les chercheurs espèrent qu'un système intelligent pourra exécuter

Raisonnement et résolution de problème

Pour les problèmes difficiles, un algorithme requiert de grandes ressources de calculs et la quantité de mémoire et le temps d'exécution demandés deviennent astronomiques pour certains problèmes. La recherche d'algorithme de résolution de problème plus efficace est une priorité absolue

Représentation de connaissance

La représentation de connaissance est importante en IA. Plusieurs problèmes que les machines vont devoir résoudre demande une grande connaissance sur le monde. Parmi les choses que l'IA doit représenter : les objets, les propriétés, les catégories, les relations entre les objets, les situations, les événements, les causes et les effets, la connaissance sur la connaissance et beaucoup d'autres

planning

Les agents intelligents doivent être capables de choisir des buts et de les achever. Ils ont besoin d'un moyen pour représenter l'état du monde et être capables de faire des prédictions sur comment vont impacter leurs actions

L'apprentissage

Le ML ou l'apprentissage automatique est l'étude d'algorithme qui se perfectionne à travers l'expérience

Créativité

La créativité artificielle est le croisement des domaines de l'IA, la psychologie cognitive, la philosophie et les arts. L'objectif est de simuler ou de répliquer la créativité en utilisant des ordinateurs afin de construire des machines dotées de créativité humaine, de mieux comprendre la créativité humaine et de hausser son niveau sans pour autant être créative soi même

Perception et manipulation

Les systèmes biologiques sont capables de s'adapter à de nouveaux environnements, pas parfaitement, parfois ils meurent mais parfois ils réussissent. Actuellement, les programmes humains sont fragiles, si ils sont compilés pour une certaine architecture, ils ne peuvent pas tourner sur une autre architecture. Est ce qu'on peut construire un programme qui s'installe automatiquement sur une architecture inconnue ?

3 Contribution

Dans ce travail, nous allons reproduire les résultats en l'état de l'art actuel en utilisant un certain type d'algorithmes de ML appliqués au problème de classification d'images en comparant les résultats de plusieurs modèles différents allant du plus simple au plus sophistiqué possible que nous permettent les ressources disponibles.

Nous essayerons par la suit de faire plusieurs optimisations en faisant varier les différents paramètres qui constituent le modèle afin de comprendre l'impact de chacun d'eux sur le résultat final.

État de l'art

1 Qu'est ce que le machine learning ?

Le ML est une discipline de l'IA qui offre aux ordinateurs la possibilité d'apprendre à partir d'un ensemble d'observations que l'on appelle ensemble d'apprentissage.

Chaque observation, comme par exemple « j'ai mangé tels et tels aliments à tel moment de la journée pendant telle période ce qui a causé telle maladie » est décrite au moyen de deux types de variables :

- Les premières sont appelées les variables prédictives (ou attributs ou caractéristiques), dans notre exemple mon age, mon dossier médical, mes antécédents médicaux. Ce sont les variables à partir desquelles on espère pouvoir faire des prédictions. les n variables prédictives associées à une observation seront notées comme un vecteur $\mathbf{x}=(x_1, ..., x_n)$ à n composantes. Un ensemble de M observations sera constitué de M tels vecteurs $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(M)}$.
- Une variable cible dont on souhaite prédire la valeur pour des événements non encore observés. Dans notre exemple, il s'agirait de la maladie contractée. on notera y cette variable cible.

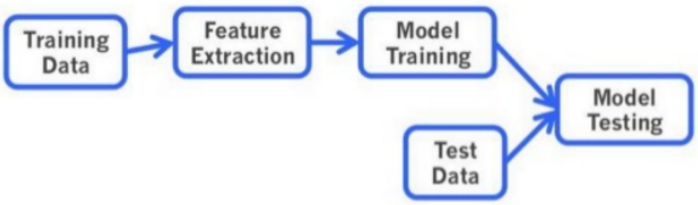

FIGURE 1 – le processus typique du ML

En résume, la valeur de la variable y dépend de :

- Une fonction $F(x)$ déterminée par les variables prédictives.
- Un bruit $\varepsilon(x)$ qui est le résultat d'un nombre de paramètres dont on ne peut pas tenir compte.

Aussi bien F que ε ne seront jamais connues mais l'objectif d'un modèle de ML est d'obtenir la meilleure approximation possible de F à partir des observations disponibles. Cette approximation sera notée f, on l'appelle la fonction de prédiction.

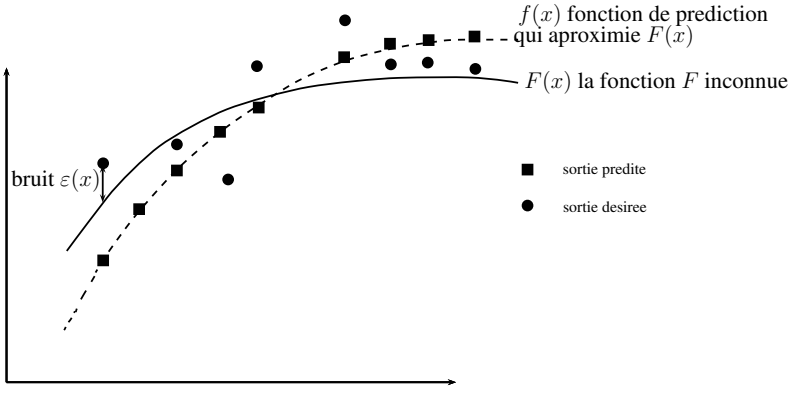

FIGURE 2 – un modèle de ML qui essaye d'obtenir la meilleure approximation possible de F

Voici quelques exemples d'utilisation du ML :

- Vision par ordinateur [6]
- Détection de fraude [7]
- Classification (image, texte, video, son, ...)
- Les publicités ciblées [8]
- Diagnostic médical [9]

2 performance et surapprentissage

On peut penser que la performance d'un modèle sera en fonction des prédictions correctes faites sur l'ensemble d'observation utilisées pour l'apprentissage, plus elle est élevé, mieux c'est.

Pourtant c'est complètement faux, ce qu'on cherche à obtenir du ML n'est pas de prédire avec exactitude les valeurs des variables cibles connues puisque elles ont été utilisées pour l'apprentissage mais bien de prédire celles qui n'ont pas encore été observées. Par conséquent, la qualité d'un algorithme de ML se juge sur sa capacité à faire les bonnes prédictions sur les nouvelles observations grâce aux caractéristiques apprises lors de la phase d'entraînement. Il faut donc éviter le cas ou on a un modèle de MLtellement trop entraîné qu'il arrive à prédire à la perfection les données d'apprentissage mais qui n'arrive pas à généraliser sur les données de test. On l'appelle le sur-apprentissage.

La cause du sur-apprentissage est que le modèle est trop complexe par rapport à la fonction F que l'on souhaite apprendre.

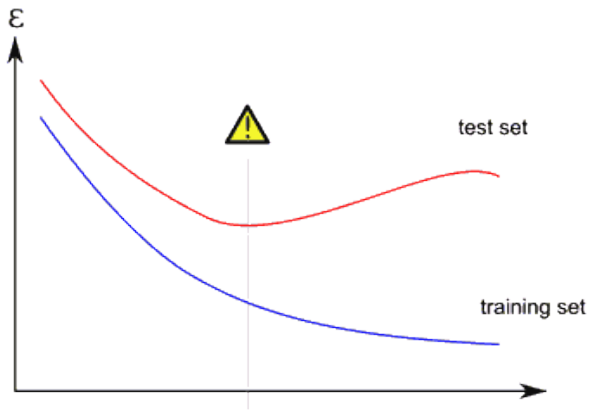

FIGURE 3 – Le sur-apprentissage : le graphe montre l'évolution de l'erreur commise sur l'ensemble de test par rapport à celle commise sur l'ensemble d'apprentissage, les deux erreurs diminuent mais dés que l'on rentre dans une phase de sur-apprentissage, l'erreur d'apprentissage continue de diminuer alors que celle du test augmente.

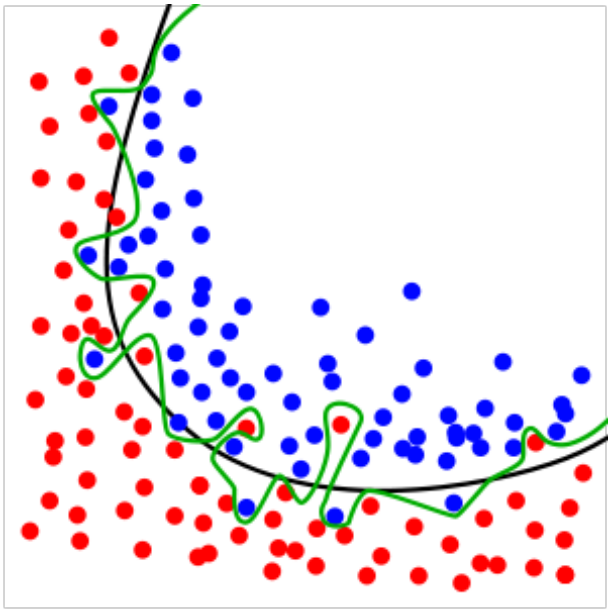

FIGURE 4 – La ligne verte représente un modèle surentraîné et la ligne noire représente un modèle régularisé. Ce dernier aura une erreur de test moins importante.

Pour résoudre ce problème, on divise les données disponibles en deux groupes distincts. Le premier sera l'ensemble d'apprentissage, et le deuxième sera l'ensemble de test.

Pour avoir une bonne séparation des données en données d'apprentissage et données de test, on utilise la validation croisée. L'idée c'est de séparer aléatoirement les données dont on dispose en k parties séparées de même taille. Parmi ces k parties, une fera office d'ensemble de test et les autres constitueront l'ensemble d'apprentissage. Après que chaque échantillon ait été utilisé une fois comme ensemble de test. On calcule la moyenne des k erreurs moyennes pour estimer l'erreur de prédiction.

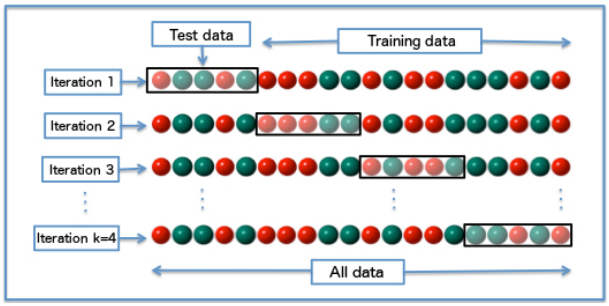

FIGURE 5 – La validation croisée.

3 Les différents types de machine learning

3.1 apprentissage supervisé et non supervisé

- La tache de l'apprentissage supervisé c'est :

  soit l'ensemble d'apprentissage composé de N exemples de pair entrée-sortie :

  $$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(M)}, y^{(M)})$$

  chaque $y^{(j)}$ a été généré par une fonction $F(x) = y$ inconnue,

  découvrir la fonction f qui se rapproche de F

- L'apprentissage non supervisé ou clustering ne demande aucun étiquetage préalable des données. Le but est que le modèle réussisse à regrouper les observations disponibles en catégories par lui même

  l'apprentissage semi supervisé est à mi chemin entre ces deux méthodes. On fournit au modèle quelques exemples étiquetés mais la grande partie des données ne le sont pas. On trouve des cas d'application partout où l'obtention des données est facile mais leur étiquetage demande des efforts, du temps ou de l'argent comme par exemple :
  — En reconnaissance de parole, il ne coûte rien d'enregistrer une grande quantité de parole, mais leur étiquetage nécessite des personnes qui les écoutent.
  — Des milliards de pages web sont disponibles, mais pour les classer il faut les lire.

3.2 régression et classification

- Un modèle de classification est un modèle de MLdont les sorties y appartiennent à un ensemble fini de valeurs (exemple : bon, moyen, mauvais)
- Un de modèle de régression est un modèle de MLdont les sorties y sont des nombres (exemple : la température de demain)

4 Les différents type d'algorithme

4.1 La régression linéaire

Une régression linéaire est un modèle de MLsupervisé, avec x en entrée et y en sortie elle est de la forme $y = w_1x + w_0$ ou w_0 et w_1 sont des valeurs réelles à apprendre. On définit $\mathbf{w}$ comme le vecteur $[w_0, w_1]$, et définir :

$$f(x) = w_1x + w_0 \tag{1}$$

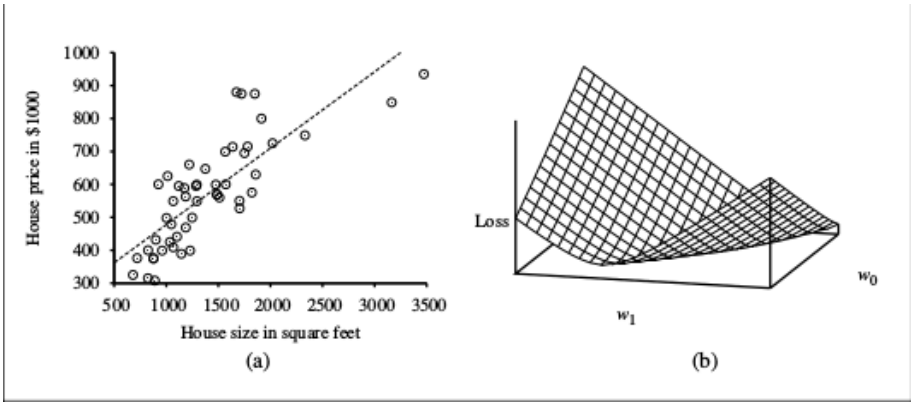

FIGURE 6 – (a) représente le prix des maisons en fonction de leur surface qui sont en vente à Berkeley, CA, Juillet 2009, représente par la fonction linéaire qui minimise l'erreur carrée : $y = 0.232x + 246$. (b) représente la fonction coût $\Sigma(w_1x_i + w_0 - y_i)^2$ pour différentes valeurs de w_1 et w_0. à noter que le fonction coût est convexe (c'est une somme de carrés) et elle possède un seul minimum global [10]

Figure 6(a) montre un exemple d'ensemble d'apprentissage de n points sur le plan x, y, chaque point représente la surface et le prix d'une maison en vente. La tache de trouver f qui convient le mieux à ces données s'appelle la régression linéaire. Pour trouver f il suffit de trouver les valeurs $[w_1, w_0]$ qui minimise l'erreur empirique.

$$Loss(f) = \sum_{i=1}^{N} (y_i - (w_1x_i + w_0))^2 \tag{2}$$

- Avantages :
  - L'avantage avec ce modèle est que l'apprentissage se résume à trouver la solution de l'équation $w^* = argmin_w Loss(f)$ qui est exacte :

$$w^* = (X^T X)^{-1} X^T y \tag{3}$$

  - le modèle est facile à interpréter

- Inconvénients
  - Sensible aux bruits
  - Négligence des interactions entre les variables prédictives

4.2 Les k plus proches voisins

L'algorithme des K-Nearest Neighbors (KNN) (K plus proches voisins) est un algorithme de classification supervisé. Chaque observation de l'ensemble d'apprentissage est représentée par un point dans un espace à n dimensions ou n est le nombre de variables prédictives. Pour prédire la classe d'une observation, on cherche les k points les plus proches de cet exemple. La classe de la variable cible, est celle qui est la plus représentée parmi les k plus proches voisins. Il existe des variantes de l'algorithme ou on pondère les k observations en fonction de leur distance à l'exemple dont on veut classer [11], les observations les plus éloignées de notre exemple seront considérées comme moins importantes.

Une variante de l'algorithme est utilisée par NetFlix [12]pour prédire les scores qu'un utilisateur attribuera à un film en fonction des scores qu'il a attribués à des films similaires

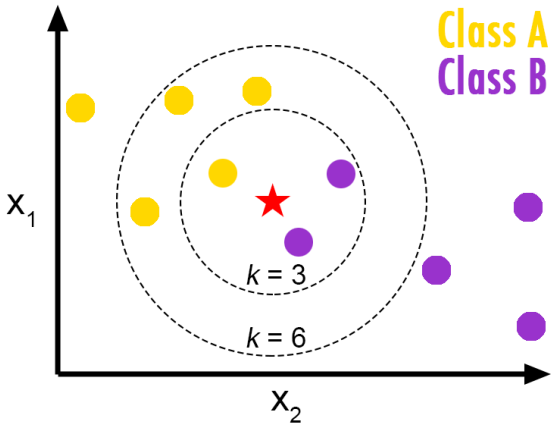

FIGURE 7 – Pour $k = 3$ la classe majoritaire du point central est la classe B, mais si on change la valeur du voisinage $k = 6$ la classe majoritaire devient la classe A

- Avantages :
  - simple à concevoir

- Inconvénients
  - Sensible aux bruits
  - Pour un nombre de variable prédictives très grands, le calcul de la distance devient très coûteux.

4.3 Le classifieur naïf de Bayes

Le classifieur naïf de Bayes est un algorithme supervisé probabiliste qui suppose que l'existence d'une caractéristique pour une classe, est indépendante de l'existence d'autres caractéristiques, raison pour laquelle on utilise l'adjectif «naif» . Une personne peut être considérée comme un homme si il pèse un certain poids et mesure une certaine taille. Même si ces caractéristiques sont liées dans la réalité, un classifieur bayésien naïf déterminera que la personne est un homme en considérant indépendamment ces caractéristiques de taille et de poids

Malgré des hypothèses de base extrêmement simplistes, ce classifieur conduit à de très bons résultats dans beaucoup de situations réelles complexes. En 2004, un article a montré qu'il existe des raisons théoriques derrière cette efficacité inattendue [13]. Toutefois, une autre étude de 2006 montre que des approches plus récentes (arbres renforcés, forêts aléatoires) permettent d'obtenir de meilleurs résultats[14].

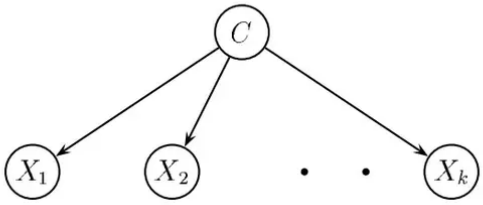

FIGURE 8 – Le classifieur naïf de Bayes est basé sur le théorème de Bayes avec une indépendance (dite naïve) des variables prédictives.

- Avantages :
  - L'algorithme offre de bonne performance
- Inconvénients
  - La prédiction devient erronée si L'Hypothèse indépendance conditionnelle est invalide

4.4 k-means

L'algorithme des k-moyennes (k-means) est un algorithme non supervisé. Chaque observation est représentée par un point dans un espace à n dimensions ou n est le nombre de variables descriptives. À partir d'un ensemble d'apprentissage de M observations $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(M)}$, cet algorithme va repartir ces observations en k clusters de manière à ce que la distance euclidienne qui sépare les points au centre de gravité du groupe auquel ils sont affectés soit minimale. Les étapes de l'algorithme sont :

- Choisir k points qui représentent la position moyenne des cluster
- répéter jusqu'à stabilisation des points centraux :
  - affecter chacun des M points au plus proche des k points centraux
  - mettre à jour les points centraux en calculant les centres de gravité des k cluster
- Avantages :
  - implementable pour des grands volumes de données
- Inconvénients
  - Le choix du paramètre k n'est pas découvert mais choisi par l'utilisateur
  - La solution dépend des k centre de gravité choisi lors de l'initialisation

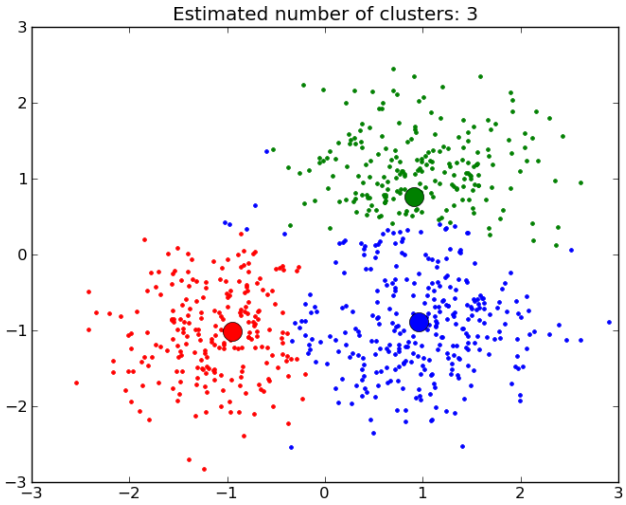

FIGURE 9 – L'algorithme k-means regroupe les données en k cluster, ici $k = 3$. Les centres de gravité sont représentés par de petit cercle

4.5 Les arbres de décision

Les arbres de décision sont des modèles de MLsupervisés, pouvant être utilisés pour la classification que pour la régression.

Un arbre de décision représente une fonction qui prend comme entrée un vecteur d'attributs et retourne une décision qui est une valeur unique. Les entrées et les sorties peuvent être discrètes ou continues.

Un arbre de décision prend ses décisions en exécutant une sequence de test, chaque nœud interne de l'arbre correspond à un test de la valeur d'un attribut et les branches qui sortent du nœud sont les valeurs possibles de l'attribut. La classe de la variable cible est alors déterminée par la feuille dans laquelle parvient l'observation à l'issue de la sequence de test.

La phase d'apprentissage consiste à trouver la bonne sequence de test. Pour cela, on doit décider des bons attributs à garder. Un bon attribut divise les exemples en ensembles homogènes c.à .d qu'ils ne contiennent que des observations appartenant à la même classe, alors qu'un attribut inutile laissera les exemples avec presque la même proportion de valeur pour la variable cible .

Ce dont on a besoin c'est d'une mesure formelle de "bon" et "inutile". Pour cela, il existe des métriques standards homogénéisées avec lesquels on peut mesurer l'homogénéité d'un ensemble. Les plus connus sont l'indice de diversité de Gini et l'entropie [15]

En général l'entropie d'une variable aléatoire V avec des valeurs v_k chacune avec une probabilité $P(v_k)$ est définie comme :

$$Entropie : H(V) = -\sum_{K} P(v_k) log_2 P(v_k) \tag{4}$$

- Avantages :

  — C'est un modèle boîte blanche, simple à comprendre et à interpréter
  — Peu de préparation des données.
  — Les variables prédictives en entrée peuvent être aussi bien qualitatives que quantitatives.
  — Performant sur de grands jeux de données

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

FIGURE 10 – L'ensemble d'apprentissage contient 12 observations décrites par 10 variables prédictives et une variable cible. [10]

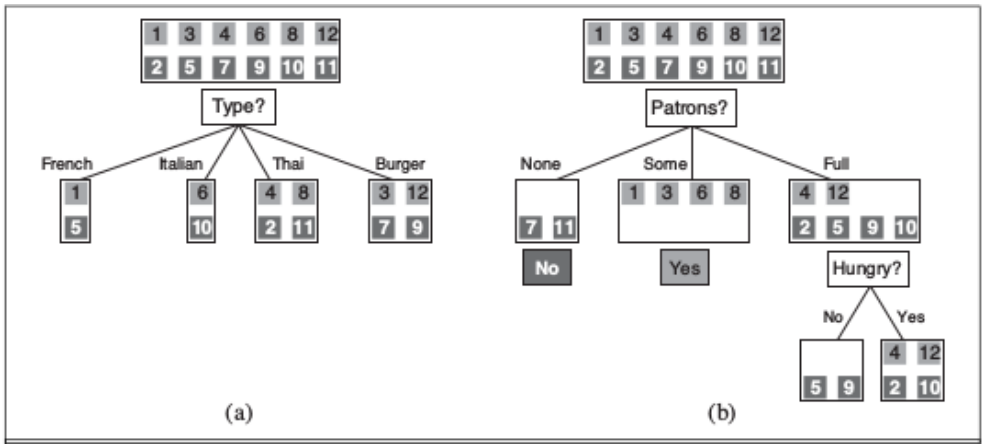

FIGURE 11 – Les exemples positifs sont représentés par des cases claires alors que les exemples négatifs sont représentés par des cases sombres. (a) montre que la division par l'attribut *Type* n'aide pas à avoir une distinction entre les positifs et les négatifs exemples. (b) montre qu' avec la division par l'attribut *Patrons* on obtient une bonne séparation entre les deux classes. Après division par *Patrons*, *Hungry* est un bon second choix. [10]

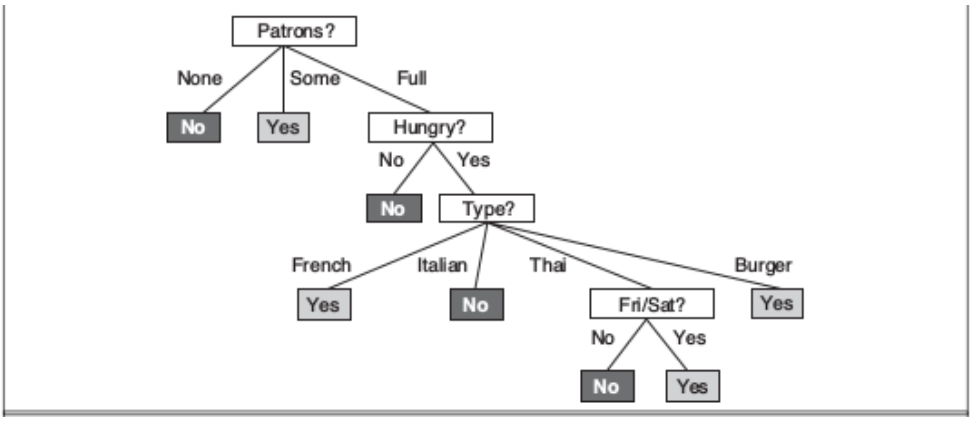

FIGURE 12 – L'arbre de décision déduit à partir des 12 exemples d'apprentissage.[10]

- Inconvénients

  — L'existence d'un risque de sur-apprentissage si l'arbre devient très complexe. On utilise des procédures d'élagage pour contourner ce problème.

4.6 Les forets aléatoires

Combiner un ensemble de classifieurs individuels faibles pour former un unique système de classification — appelé Ensemble de Classifieurs a suscité un intérêt grandissant de la communauté scientifique. L'efficacité des combinaisons de classifieurs repose principalement sur leur capacité à tirer parti des complémentarités des classifieurs individuels, dans le but d'améliorer autant que possible les performances en généralisation de l'ensemble.

Parmi les différentes approches de construction d'ensembles de classifieurs, l'algorithme des forets aléatoires [16], composé d'un ensemble de classifieurs élémentaires de type arbres de décision, le but de l'algorithme est de conserver les avantages des arbres de décision tout en éliminant leurs inconvénients et particulièrement leur vulnérabilité au sur-apprentissage.C'est un algorithme qui peut être utilisé aussi bien pour la classification que pour la régression.

L'algorithme repose sur trois idées principales :

- Pour M observations de l'ensemble d'apprentissage, chacune décrite par n variables prédictives, on crée B nouveau échantillon de même taille M par tirage avec remise. Cette technique s'appelle le **boostrap**. Chacun des B échantillon servira à l'apprentissage d'un arbre de décision.
- Pour n caractéristiques, un nombre $k < n$ (généralement $\sqrt{n}$)est tiré aléatoirement de sorte qu'à chaque nœud de l'arbre, un sous-ensemble de k caractéristiques soit tiré aléatoirement, parmi lesquelles la meilleure est ensuite sélectionnée pour le partitionnement.
- Pour classer une nouvelle observation, on procède par vote majoritaire. On fait passer cette observation par les B arbres et sa classe c'est la classe majoritaire parmi les B prédictions

Quelques travaux de recherche se sont intéressés au nombre d'arbres de décision à construire au sein d'une forêt. Quand Breiman a introduit les forets aléatoires, il démontra également qu'au delà d'un certain nombre d'arbres, en ajouter d'autres ne permettait pas forcement d'améliorer les performances de l'ensemble. Ce résultat indique que le nombre d'arbres d'une foret aléatoire ne doit pas nécessairement être le plus grand possible pour produire un classifieur performant. Figure 13

- Avantages :

  — C'est un des meilleur algorithme pour ce qui est de la precision.
  — Incorporation de la validation croisée.
  — Il conserve une bonne puissance de prédiction même si il y' a des données manquantes
  — Il ne souffre pas du sur-apprentissage

- Inconvénients

  — Une implémentation difficile

4.7 Les machines à vecteur de support

Les Support Vector Machine (SVM) (machines à vecteur de support) [17] sont des algorithmes de classification binaire non linaire très puissant.

Le principe des SVM consiste à construire une bande séparatrice non linéaire de largeur maximale qui sépare deux ensembles d'observations et à l'utiliser pour faire des prédictions. l'astuce des SVM pour y parvenir consiste à utiliser une transformation φ non linéaire qui envoie les points $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(M)}$ de l'espace original à n dimensions (n est le nombre de variable prédicatives) vers des nouveaux points $\varphi(\mathbf{x}^{(1)}), ..., \varphi(\mathbf{x}^{(M)})$ dans un espace de dimension plus grand que n ou ils seront plus facile à séparer.

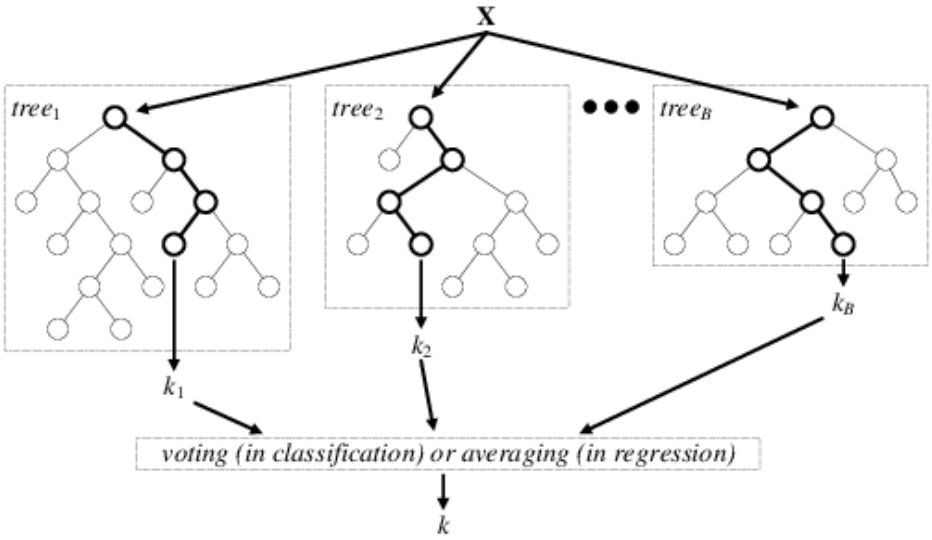

FIGURE 13 – Création de B boostrap à partir des exemples d'apprentissage, chacun servira à entraîner un arbre de décision. Si c'est un cas de classification, la classe finale est attribuer par vote majoritaire.Si c'est un cas de régression, on fait la moyenne des prédictions

Les SVM sont des classificateurs qui reposent sur deux idées clés :

La première idée consiste à trouver un séparateur linéaire de largeur maximale, c'est la notion de marge maximale. La marge est la distance entre la frontière de séparation et les échantillons les plus proches. Ces derniers sont appelés vecteurs supports. Le problème est de trouver cette frontière séparatrice optimale.

Dans le cas où le problème est linéairement séparable, le choix de l'hyperplan séparateur n'est pas évident. Il existe en effet une infinité d'hyperplans séparateurs, dont les performances en phase d'apprentissage sont identiques, mais dont les performances en phase de test peuvent être très différentes. Pour résoudre ce problème, il a été montré [18], qu'il existe un unique hyperplan optimal, défini comme l'hyperplan qui maximise la marge entre les échantillons et l'hyperplan séparateur.

Il existe des raisons théoriques à ce choix. Vapnik a montré [18] que la capacité des classes d'hyperplans séparateurs diminue lorsque leur marge augmente.

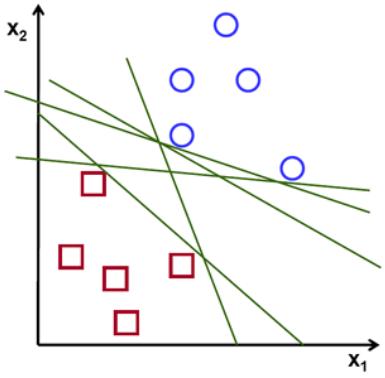

FIGURE 14 – On cherche un hyperplan qui divise les observations en deux catégories. Considérons un exemple x qu'on veut classer, si $f(x) > 0$, il appartient à la classe des cercles, sinon il appartient à la classe des carrés. Dans cette figure on peut voir qu'il existe une infinité d'hyperplans séparateurs possibles.

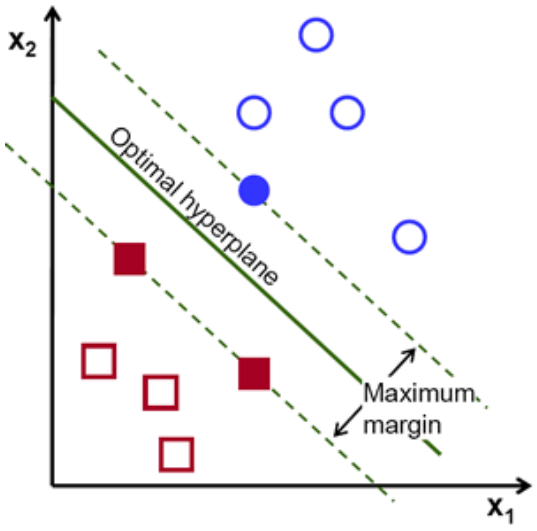

FIGURE 15 – L'hyperplan optimal (en vert) avec la marge maximale. Les échantillons remplis sont des vecteurs supports..

Afin de pouvoir traiter des cas où les données ne sont pas linéairement séparables, la deuxième idée clé des SVM est de transformer l'espace de représentation des données d'entrées en un espace de plus grande dimension, dans lequel il est probable qu'il existe une séparation linéaire. Ceci est réalisé grâce à une fonction noyau, qui doit respecter les conditions du théorème de Mercer [19], et qui a l'avantage de ne pas nécessiter la connaissance explicite de la transformation à appliquer pour le changement d'espace. Les fonctions noyaux permettent de transformer un produit scalaire dans un espace de grande dimension, ce qui est coûteux, en une simple évaluation ponctuelle d'une fonction. Cette technique est connue sous le nom de kernel trick.

Les deux fonctions noyaux les plus utilisées sont le noyau polynomial et le noyau gaussien

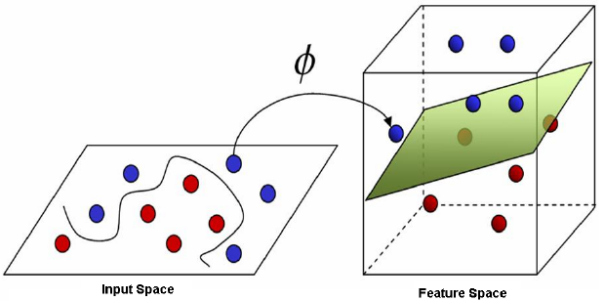

FIGURE 16 – Exemple d'un problème non linéairement séparable. La courbe devient une bande linéaire après avoir appliqué la transformation φ non linéaire

- Avantages :
  - Il permet de traiter des problèmes de classification non linéaire complexe.
  - Les SVM constituent une alternative aux réseaux de neurones car plus faciles à entraîner.
- Inconvénients
  - Les SVM sont souvent moins performants que les forets aléatoires.

4.8 Le perceptron multicouches

Multi Layer Perceptron (MLP) (perceptron multicouches)[20] est un classifieur de type réseaux de neurones qui est organisé en plusieurs couches, chaque couche étant formée d' un ou plusieurs neurones formels. Utilisé dans un cas d'apprentissage supervisé, il utilise l'algorithme de retro propagation de gradient [21]

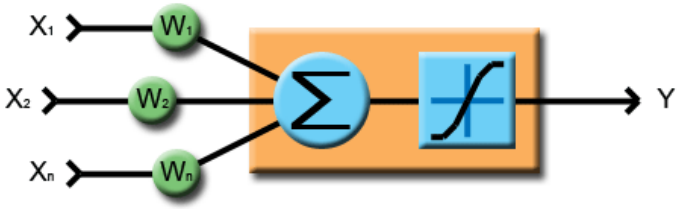

FIGURE 17 – La figure représente le fonctionnement d'un neurone formel. C'est un composant calculatoire qui fait la somme pondérée des signaux reçus en entrée, puis on leur applique une fonction d'activation afin d'obtenir Y

Voici les caractéristiques du modèle :

- Il comporte une seule couche d'entrée et une seule couche de sortie.
- il peut comporter une ou plusieurs couches cachées.
- Chaque neurone est relié uniquement à tous les neurones de la couche suivante.
- Chaque lien de la couche i vers la couche suivante j sert à propager l'activation a_i de i jusqu'à j et qui possède un poids w_{ij} qui détermine l'intensité du signal de la connexion. Chaque unité de la couche j calcul la somme pondéré de ses entrées :

$$in_j = \sum_{i=0}^{n} w_{ij} a_i \tag{5}$$

puis leur applique une fonction d'activation :

$$a_j = g(in_j) = g(\sum_{i=0}^{n} w_{ij} a_i) \tag{6}$$

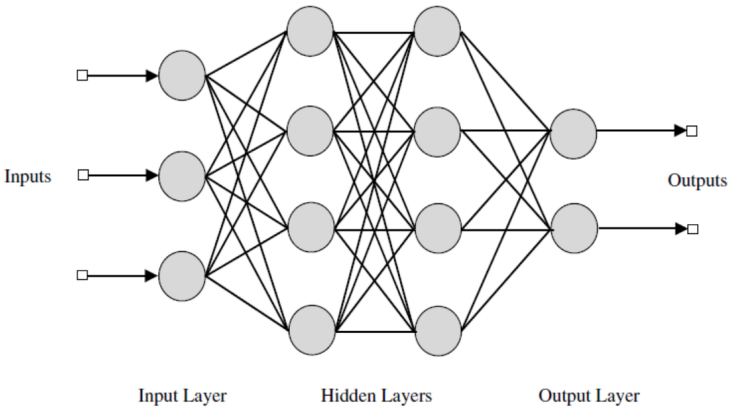

FIGURE 18 – Structure d'un PMC composé d'une couche d'entrée, deux couches cachées et une couche de sortie

L'algorithme d'apprentissage consiste à présenter au réseau des entrées et on lui demande de modifier sa pondération de façon à retrouver la sortie correspondante. D'abord on propage vers l'avant les entrées jusqu'à obtenir une sortie calculée par le réseau puis on compare cette sortie avec la sortie désirée, enfin on modifie les poids de telle sorte qu'à la prochaine itération l'erreur commise entre les sortie réelles et les sortie désirées soit minimisée. On répète ce processus jusqu'à ce qu'on obtienne une erreur de sortie négligeable.

- Avantages :
  - Capacité à découvrir les dépendances par lui même.
  - Résistance aux bruits
- Inconvénients
  - C'est un modèle boite noire qui n'explique pas ses décisions.

5 Conclusion

Le ML est un sujet vaste en évolution permanente. Les algorithmes qu'il met en œuvre ont des sources d'inspiration variées qui vont de la théorie des probabilités aux intuitions géométriques en passant par des approches heuristiques.

Deep Learning

1 Introduction

Le Deep Learning est un nouveau domaine de recherche du ML, qui a été introduit dans le but de rapprocher le MLde son objectif principal : l'intelligence artificielle. Il concerne les algorithmes inspirés par la structure et le fonctionnement du cerveau. Ils peuvent apprendre plusieurs niveaux de représentation dans le but de modéliser des relations complexes entre les données.

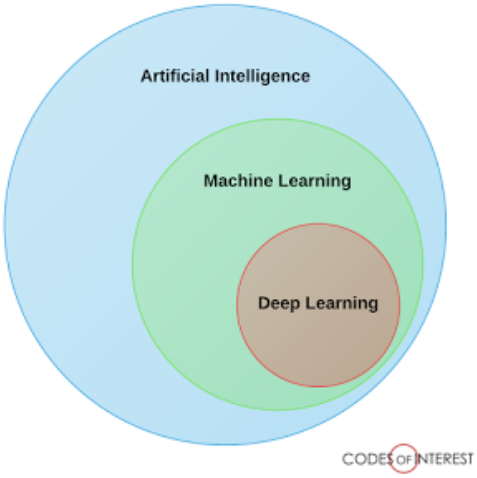

FIGURE 19 – La relation entre l'intelligence artificielle, le ML et le deep learning

Le Deep Learning est basé sur l'idée des réseaux de neurones artificielles et il est taillé pour gérer de larges quantités de données en ajoutant des couches au réseau. Un modèle de deep learning a la capacité d'extraire des caractéristiques à partir des données brutes grâce aux multiples couches de traitement composé de multiples transformations linéaires et non linéaires et apprendre sur ces caractéristiques petit à petit à travers chaque couche avec une intervention humaine minime [22][23][24][25][26][][27][28].

Sur les cinq dernières années, le deep learning est passé d'un marché de niche ou seulement une poignée de chercheurs s'y intéressait au domaine le plus prisé par les chercheurs. Les recherches en relation avec le deep learning apparaissent maintenant dans les top journaux comme Science [29], Nature [30] et Nature Methods [31] pour ne citer que quelques-uns. Le deep learning a coquerie le GO [32], appris à conduire une voiture [33], diagnostiquer le cancer [34] et l'autisme [35] et même devenu un artiste [36].

Le terme "Deep Learning" a été introduit pour la première fois au ML par Dechter (1986) [37], et aux réseaux neuronaux artificiels par Aizenberg et al (2000) [38].

2 Histoire du deep learning

Année	Contributeur	Contribution
300 AC	Aristotle	introduction de l'associationnisme, début de l'histoire des humains qui essayent de comprendre le cerveau
1873	Alexander Bain	introduction du Neural Groupings comme les premiers modèles de réseaux de neurones
1943	McCulloch and Pitts	introduction du McCulloch–Pitts (MCP) modèle considérer comme L'ancêtre des réseaux de neurones artificielles
1949	Donald Hebb	considérer comme le père des réseaux de neurones, il introduit la règle d'apprentissage de Hebb qui servira de fondation pour les réseaux de neurones modernes
1958	Frank Rosenblatt	introduction du premier perceptron
1974	Paul Werbos	introduction de la retro propagation
1980	Teuvo Kohonen	introduction des cartes auto organisatrices
1980	Kunihiko Fukushima	introduction du Neocognitron, qui a inspiré les réseaux de neurones convolutif
1982	John Hopfield	introduction des réseaux de Hopfield
1985	Hilton and Sejnowski	introduction des machines de Boltzmann
1986	Paul Smolensky	introduction de Harmonium, qui sera connu plus tard comme machines de Boltzmann restreintes
1986	Michael I. Jordan	définition et introduction des réseaux de neurones récurrent
1990	Yann LeCun	introduction de LeNet et montra la capacités des réseaux de neurones profond
1997	Schuster and Paliwal	introduction des réseaux de neurones récurrent bidirectionnelles
1997	Hochreiter and Schmidhuber	introduction de LSTM, qui ont résolu le problème du vanishing gradient dans les réseaux de neurones récurrent
2006	Geoffrey Hinton	introduction des Deep belief Network
2009	Salakhutdinov and Hinton	introduction des Deep Boltzmann Machines
2012	Alex Krizhevsky	introduction de AlexNet qui remporta le challenge ImageNet

Table 1 – Les étapes majeurs du Deep Learning [1]

3 Pourquoi le deep learning ?

Les algorithme de ML décrits dans la première partie fonctionnent bien pour une grande variété de problèmes. Cependant ils ont échoues à résoudre quelques problèmes majeurs de l'IA telle que la reconnaissance vocale et la reconnaissance d'objets.

Le développement du deep learning fut motivé en partie par l'échec des algorithmes traditionnels dans de telle tache de l'IA.

Mais ce n'est qu'après que de plus grandes quantités de données ne soit disponibles grâce notamment au Big Data et aux objets connectés et que les machines de calcul soient devenues plus puissantes qu'on a pu comprendre le potentiel réel du Deep Learning.

Une des grandes différences entre le Deep Learning et les algorithmes de MLtraditionnelles c'est qu'il s'adapte bien, plus la quantité de données fournie est grande plus les performances d'un algorithme de Deep Learning sont meilleurs. Contrairement à plusieurs algorithmes de MLclassiques qui possèdent une borne supérieure a la quantité de données qu' ils peuvent recevoir des fois appelée "plateau de performance", les modèles de Deep Learning n'ont pas de telles limitations (théoriquement) et ils sont même allés jusqu'à dépasser la performance humaine dans des domaines comme l'image processing.

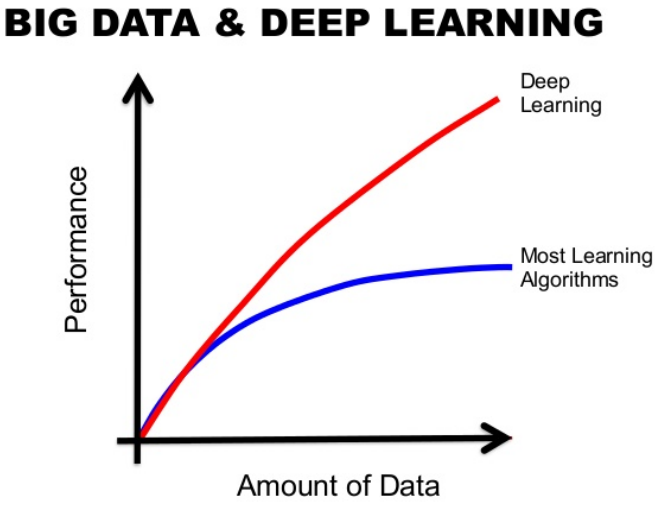

FIGURE 20 – La différence de performance entre le Deep Learning et la plupart des algorithmes de MLen fonction de la quantité de données

Autre différence entre les algorithmes de ML traditionnelles et les algorithmes de Deep Learning c'est l'étape de l'extraction de caractéristiques. Dans les algorithmes de ML traditionnelles l'extraction de caractéristiques est faite manuellement, c'est une étape difficile et coûteuse en temps et requiert un spécialiste en la matière alors qu'en Deep Learning cette étape est exécutée automatiquement par l'algorithme.

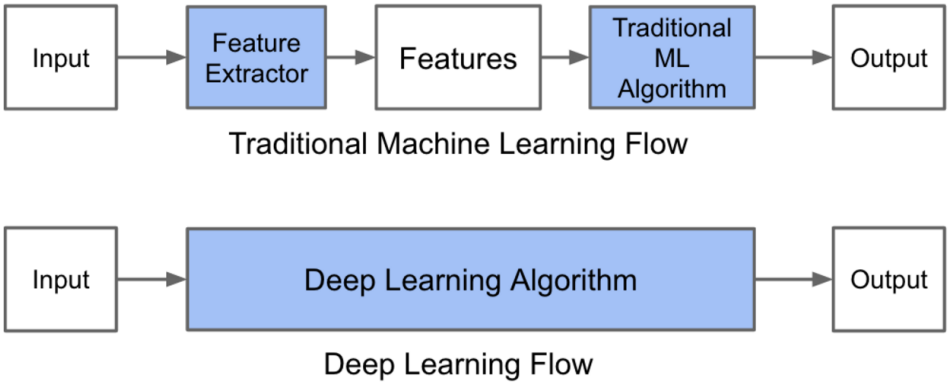

FIGURE 21 – Le procède du MLclassique comparé à celui du Deep Learning

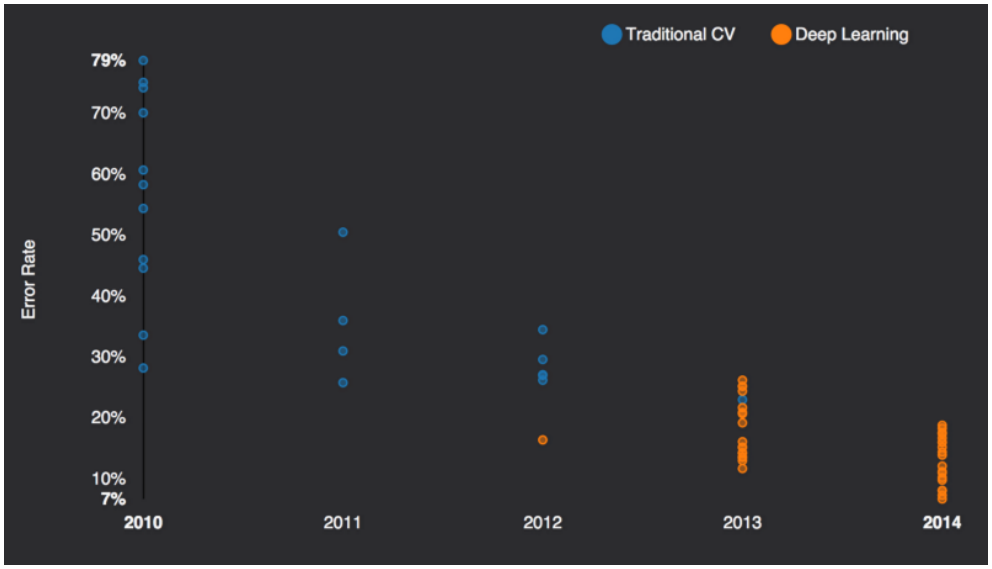

FIGURE 22 – Deep Learning vs Machine Learning dans ImageNet challenge

4 Les différents types de modèles

Il existe un grand nombre de variantes d'architectures profondes. La plupart d'entre elles sont dérivées de certaines architectures parentales originales. Il n'est pas toujours possible de comparer les performances de toutes les architectures, car elles ne sont pas toutes évaluées sur les mêmes ensembles de données. Le Deep Learning est un domaine à croissance rapide, et de nouvelles architectures, variantes ou algorithmes apparaissent toutes les semaines.

4.1 Les réseaux de neurones convolutifs

Convolutional Neural Network (CNN) (réseaux de neurones convolutifs) sont un type de réseau de neurones spécialisés pour le traitement de données ayant une topologie semblable à une grille. Les exemples comprennent des données de type série temporelle, qui peuvent être considérées comme une grille 1D en prenant des échantillons à des intervalles de temps réguliers et des données de type image, qui peuvent être considérées comme une grille 2D de pixels. Les réseaux convolutifs ont connu un succès considérable dans les applications pratiques. Le nom « réseau de neurones convolutif » indique que le réseau emploie une opération mathématique appelée convolution. La convolution est une opération linéaire spéciale. Les réseaux convolutifs sont simplement des réseaux de neurones qui utilisent la convolution à la place de la multiplication matricielle dans au moins une de leurs couches.

Ils ont de larges applications dans la reconnaissance de l'image et de la vidéo, les systèmes de recommandations [39] et le traitement du langage naturel [40].

4.1.1 Inspiration

Dans le ML, un réseau convolutif est un type de réseau de neurones feed-forward, il a été inspiré par des processus biologiques [41]. Le modèle de connectivité entre les neurones d'un CNN s'inspire de l'organisation du cortex visuel animal.

Les neurones corticaux individuels répondent aux stimuli dans une région restreinte de l'espace connu sous le nom de champ réceptif. Les champs réceptifs des différents neurones se chevauchent partiellement de sorte qu'ils couvrent le champ visuel.

La réponse d'un neurone individuel aux stimuli dans son champ réceptif peut être approchée mathématiquement par une opération de convolution.

4.1.2 L'opération de convolution

Dans sa forme la plus générale, la convolution est une opération sur deux fonctions d' argument réel. Pour comprendre la motivation derrière la convolution, nous commençons par des exemples de deux fonctions qu'on pourrait utiliser.

Supposons que nous suivons l'emplacement d'un vaisseau spatial avec un capteur laser. Notre capteur laser fournit une sortie $x(t)$ qui est la position du vaisseau spatial au moment t. x et t sont des réelles, c'est-à-dire que nous pouvons obtenir une lecture différente du capteur laser à tout moment.

Supposons maintenant que notre capteur laser soit quelque peu bruité. Pour obtenir une estimation moins bruitée de la position du vaisseau spatial, nous aimerions combiner plusieurs mesures. Bien sûr, les mesures plus récentes sont plus pertinentes, donc nous voulons que ces mesures soient une moyenne pondérée et donner plus de poids aux mesures récentes. Nous pouvons le faire avec une fonction de pondération $w(a)$, où a est l'âge de la mesure.

Si nous appliquons une telle opération de moyenne pondérée à chaque instant, nous obtenons une nouvelle fonction qui fournit une estimation lisse de la position du vaisseau spatial :

$$s(t) = \int x(a)w(t-a)da \tag{7}$$

Cette opération s'appelle convolution. L'opération de convolution est généralement désignée par un astérisque :

$$s(t) = (x * w)t \tag{8}$$

Dans notre exemple, l'idée d'un capteur laser qui peut fournir des mesures à chaque instant dans le temps n'est pas réaliste. Habituellement, lorsque nous travaillons avec des données sur l'ordinateur, le temps sera discrétisé (numérisé), et notre capteur fournira des données à des intervalles réguliers. Dans notre exemple, il est plus réaliste de supposer que notre laser fournit une mesure une fois par seconde. L'index de temps t ne peut alors prendre que des valeurs entières. Si on suppose maintenant que x et w sont des entiers, on peut définir la convolution discrète :

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{9}$$

Enfin, nous utilisons souvent des convolutions sur plus d'un axe à la fois. Figure 23. Par exemple, si nous utilisons une image bidimensionnelle x comme entrée :

$$s(i,j) = (x * w)(i,j) = \sum_m \sum_n x(m,n)w(i-m, j-n) \tag{10}$$

Dans la terminologie du réseau convolutif, le premier argument (dans cet exemple, la fonction x) de la convolution est souvent appelé **l'entrée** (input)et le second argument (dans cet exemple, la fonction w) comme **noyau** (kernel). La sortie est parfois appelée **feature map**.

4.1.3 Couche convolutif

La convolution s'appuie sur trois idées importantes qui peuvent aider à améliorer un système de ML : **sparse interactions, parameter sharing et equivariant representations.**

- **Sparse interactions** : Les réseaux de neurones traditionnelles utilisent la multiplication matricielle par une matrice de paramètres avec des paramètres séparés décrivant l'interaction entre chaque unité d'entrée et chaque unité de sortie. Cela signifie que chaque unité de sortie interagit avec chaque unité d'entrée ce qui n'est pas le cas des réseaux de neurones convolutifs. Ceci est accompli en rendant le noyau plus petit que l'entrée. Par exemple, lors du traitement d'une image, l'image d'entrée peut avoir des milliers ou des millions de pixels, mais nous pouvons détecter de petites caractéristiques significatives telles que les bords avec des noyaux qui n'occupent que des

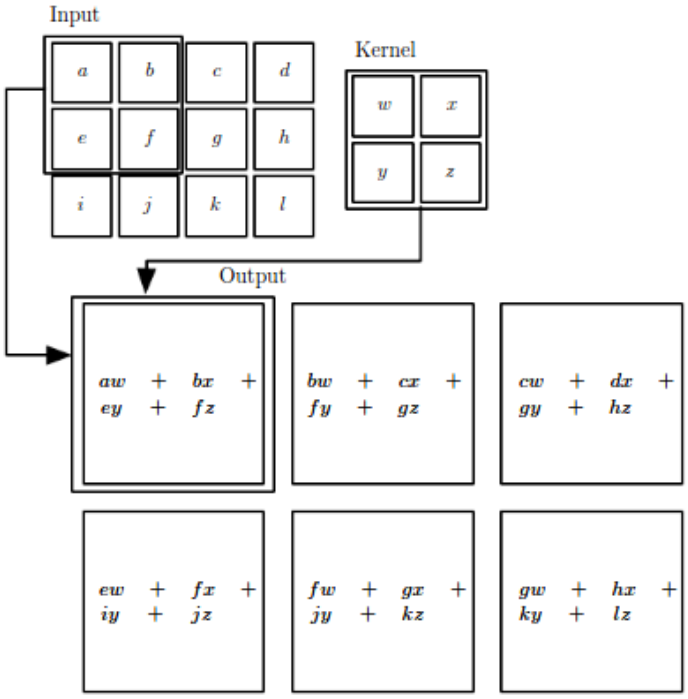

FIGURE 23 – Exemple d'une convolution 2D. [42]

dizaines ou des centaines de pixels. Cela signifie que nous pouvons stocker moins de paramètres, ce qui réduit les besoins en matière de mémoire du modèle et améliore son efficacité. Cela signifie également que le calcul des sortie nécessite moins d'opérations. Ces améliorations en matière d'efficacité sont généralement assez importantes. Figure 24.

- **parameter sharing** : (le partage des paramètres) Se réfère à l'utilisation du même paramètre pour plus d'une fonction dans un modèle. Dans un réseau de neurones convolutif, chaque élément du noyau est utilisé à chaque position de l'entrée (sauf peut-être quelques-uns des pixels des bords, selon le choix de conception concernant la frontière). Le **parameter sharing** utilisé par l'opération de convolution signifie que, plutôt que d'apprendre un ensemble de paramètres distincts pour chaque emplacement, nous n'apprenons qu'un ensemble ce qui réduit encore davantage les exigences de stockage du modèle. Figure 25.

- **equivariant representations** : Dans le cas de la convolution, la particularité du **parameter sharing** fait que la couche possède une propriété appelée **equivariant representations**. Pour dire qu'une fonction est équivariante signifie que si l'entrée change, la sortie change de la même manière. Spécifiquement, une fonction $f(x)$ est équivariante à la fonction g si $f(g(x)) = g(f(x))$. Dans le cas du traitement d'image, la convolution crée une carte 2-D de certaines caractéristiques apparaissant dans l'entrée. Si nous déplaçons l'objet dans l'entrée, sa représentation va se déplacer de la même façon dans la sortie. Par exemple, dans le cas d'une image, Il est utile de détecter les arêtes dans la première couche d'un réseau convolutif. Les mêmes arêtes apparaissent plus ou moins partout dans l'image, donc il est pratique de partager des paramètres sur l'image entière.

4.1.4 Couche de pooling

Une architecture atypique d'un réseau convolutif se compose de trois types de couches différentes. D'abord une couche convolutive pour générer un ensemble d' activations linéaires ensuite, on les fait passer à travers une couche d'activation non linéaire telle que Rectified Linear Unit (ReLu), enfin on utilise la fonction pooling. 26.

- Il permet de réduire progressivement la taille des représentations afin de réduire la quantité de paramètres et de calcul dans le réseau et, par conséquent, de contrôler également le sur-apprentissage.
- Il permet l'invariance aux petites translations
- Utile lorsque on préfère savoir si une caractéristique est présente plutôt que la région de sa présence.
- Plusieurs type de pooling diffèrent (MAX pooling (très populaire), AVG pooling, ...)

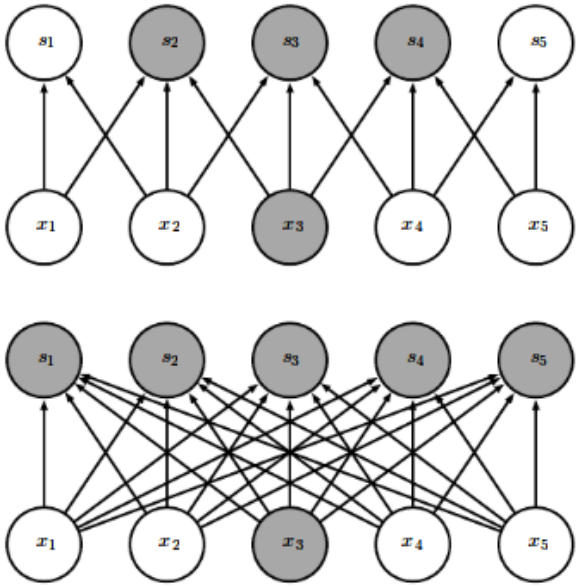

FIGURE 24 – Sparse interactions : on met en évidence l' unité d'entrée, x_3, et les unités de sortie qui sont affectées par cette unité. (en haut) Lorsque s est formé par convolution avec un noyau de largeur 3, seules trois sorties sont affectées par x. (en bas) Lorsque s est formé par une multiplication matricielle, toutes les sorties sont atteintes par x_3. [42]

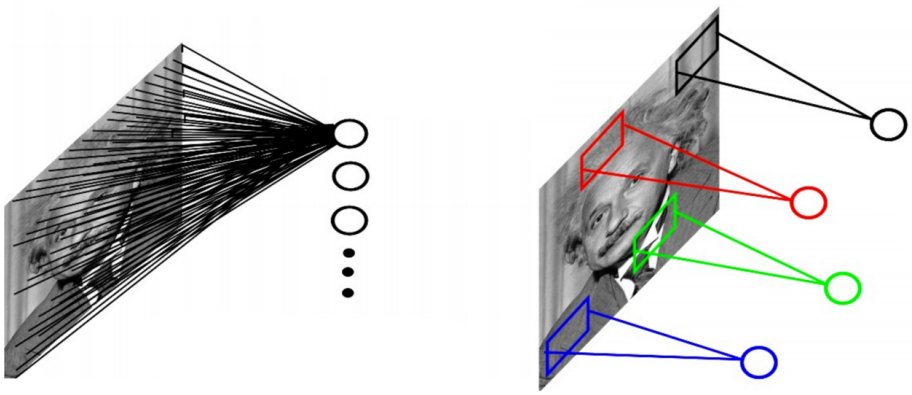

FIGURE 25 – Exemple image 1000x1000. (à gauche) Le non parameter sharing nous oblige à concevoir une couche cachée de 10^6 neurones, chaque neurone est connecté à 10^6 pixels, en tout ça fait 10^{12} paramètres. (à droite) Avec un noyau 10x10 et une couche cachée de 10^6 neurones, le nombre de paramètres est de 10^8

4.1.5 Perceptron

Après avoir extrait les caractéristiques des entrées, on attache à la fin du réseau un perceptron ou bien un MLP. Le perceptron prend comme entrée les caractéristiques extraites et produit un vecteur de N dimensions ou N est le nombre de classe ou chaque élément est la probabilité d'appartenance à une classe. chaque probabilité est calculée à l'aide de la fonction **softmax** dans le cas où les classes sont exclusivement mutuelles.

$$softmax(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{11}$$

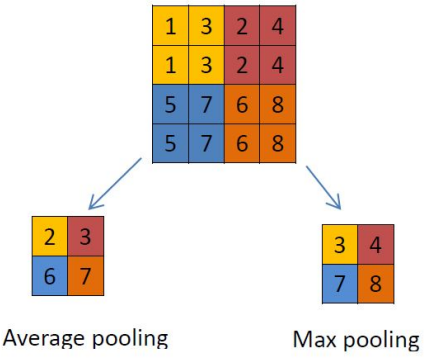

FIGURE 26 – (à gauche) Average pooling : chaque case correspond à la moyenne du carré d'entrée de la même couleur, ex de la case jaune : $(1+3+1+3)/4 = 2$. (à droite) Max pooling : chaque case correspond à la valeur maximum du carré d'entrée de la même couleur, ex de la case bleu : $max(5, 7, 5, 7) = 7$

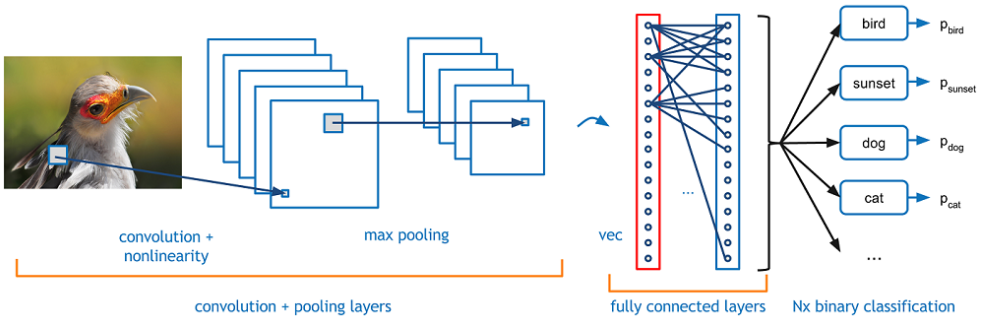

FIGURE 27 – Un reseau de neurones convolutif qui reçois une image 2D comme entrée et qui est composé d'une couche convolutive, une fonction d'activation non lineaire, une couche MAX pooling et enfin un perceptron multi couche

4.1.6 Quelques réseaux convolutifs célèbres

- **LeNet** [43] : Les premières applications réussies des réseaux convolutifs ont été développées par Yann LeCun dans les années 1990. Parmi ceux-ci, le plus connu est l'architecture LeNet utilisée pour lire les codes postaux, les chiffres, etc.
- **AlexNet**[44] : Le premier travail qui a popularisé les réseaux convolutifs dans la vision par ordinateur était AlexNet, développé par Alex Krizhevsky, Ilya Sutskever et Geoff Hinton. AlexNet a été soumis au défi ImageNet ILSVRC [45] en 2012 et a nettement surpassé ses concurrents. Le réseau avait une architecture très similaire à LeNet, mais était plus profond, plus grand et comportait des

couches convolutives empilées les unes sur les autres (auparavant, il était commun de ne disposer que d'une seule couche convolutifs toujours immédiatement suivie d'une couche de pooling).

- **ZFnet**[46] : Le vainqueur de ILSVRC challenge 2013 était un réseau convolutif de Matthew Zeiler et Rob Fergus. Il est devenu ZFNet (abréviation de Zeiler et Fergus Net). C'était une amélioration de AlexNet en ajustant les hyper-paramètres de l'architecture, en particulier en élargissant la taille des couches convolutifs et en réduisant la taille du noyau sur la première couche.
- **GoogLeNet**[47] : Le vainqueur de ILSVRC challenge 2014 était un réseau convolutif de Szegedy et al. De Google. Sa principale contribution a été le développement d'un *module inception* qui a considérablement réduit le nombre de paramètres dans le réseau (4M, par rapport à AlexNet avec 60M). En outre, ce module utilise le **global AVG pooling** au lieu du PMC à la fin du réseaux, ce qui élimine une grande quantité de paramètres. Il existe également plusieurs versions de GoogLeNet, parmi elles, Inception-v4 [48].
- **ResNet**[49] : Residual network développé par Kaiming He et al. A été le vainqueur de ILSVRC 2015. Il présente des sauts de connexion et une forte utilisation de la batch normalisation. Il utilise aussi le **global AVG pooling** au lieu du PMC à la fin.

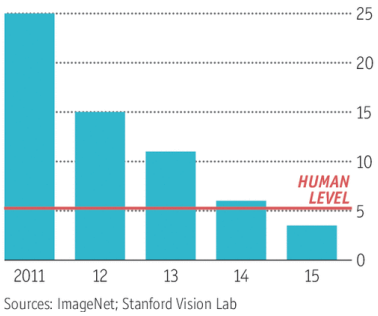

FIGURE 28 – Le taux d'erreur dans ImageNet Visual recognition Challenge. Le Deep Learning dépasse la performance humaine.

4.2 Réseau de neurones récurrents

Les humains ne commencent pas leur pensées à zéro à chaque seconde. Lorsqu'on lit un livre, on comprend chaque mot en fonction de la compréhension des mots précédents. On n'oublie pas tout et recommençons à réfléchir à nouveau. Nos pensées ont une persistance.

Les réseaux de neurones traditionnels ne peuvent pas le faire, et cela est un inconvénient majeur. Par exemple, imaginons qu'on souhaite classer quel genre d'événement se produit à chaque étape du film. On ne sait pas très bien comment un réseau de neurones traditionnels pourrait utiliser son raisonnement sur les événements précédents dans le film pour en informer les derniers.

RNN (les réseaux de neurones récurrents) traitent ce problème. Ce sont des réseaux avec des boucles, permettant aux informations de persister.

4.2.1 C'est quoi un RNN ?

L'idée derrière les RNN est d'utiliser des informations séquentielles. Dans un réseau neuronal traditionnel, nous supposons que toutes les entrées (et les sorties) sont indépendantes les unes des autres. Mais pour de nombreuses tâches, c'est une très mauvaise idée. Si on veut prédire le prochain mot dans une phrase, il faut connaître les mots qui sont venus avant. Les RNN sont appelés récurrents car ils exécutent la même tâche pour chaque élément d'une séquence, la sortie étant dépendante des calculs précédents. Une autre façon de penser les RNN est qu'ils ont une « mémoire » qui capture l'information sur ce qui a été calculé jusqu'ici. En théorie, les RNN peuvent utiliser des informations dans des séquences arbitrairement longues, mais dans la pratique, on les limite à regarder seulement quelques étapes en arrière. Voici à quoi ressemble un RNN typique :

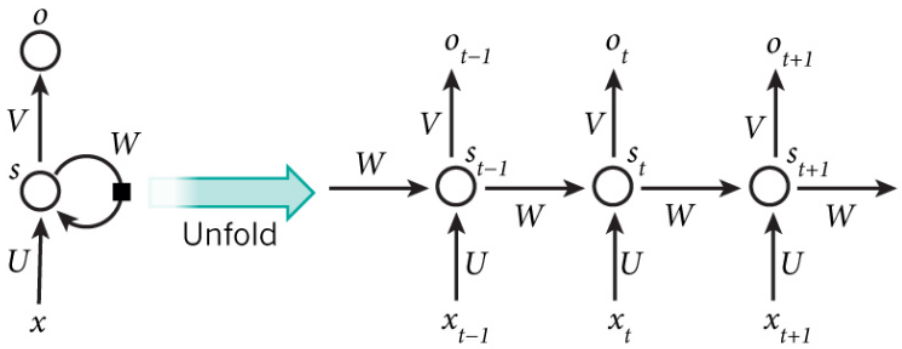

FIGURE 29 – (à gauche) Un RNN. (à droite) Sa version déroulé Source : Nature

Le schéma ci-dessus montre un RNN déroulé. En déroulant, nous signifions simplement qu'on montre le réseau pour la séquence complète. Par exemple, si la séquence qui nous intéresse est une phrase de 5 mots, le réseau serait déroulé en un réseau de neurones de 5 couches, une couche pour chaque mot. Les formules qui régissent les calculs dans un RNN sont les suivantes :

- x_t est l'entrée au moment t.
- U, V, W sont les paramètres que le réseau va apprendre des données de l'apprentissage.
- s_t est l'état caché au moment t. C'est la « mémoire » du réseau. s_t est calculé en fonction de l'état caché précédent et de l'entrée à l'étape actuelle :

$$s_t = f(Ux_t + Ws_{t-1}) \tag{12}$$

  Où f est une fonction non linéaire telle que : ReLu ou Hyperbolic tangent (tanh).
- o_t est la sortie au moment t. Par exemple, si on veut prédire le prochain mot dans une phrase, ce serait un vecteur de probabilités dans un vocabulaire.

$$o_t = softmax(Vs_t) \tag{13}$$

Quelques remarques importantes :

- On peut considérer l'état caché s_t comme la mémoire du réseau. s_t capture des informations sur ce qui s'est passé dans toutes les étapes précédentes. La sortie o_t est calculée uniquement en fonction de la mémoire au moment t. Comme mentionné brièvement ci-dessus, c'est un peu plus compliqué dans la pratique car s_t ne peut généralement pas capturer des informations depuis trop longtemps.
- Contrairement à un réseau de neurones traditionnel, qui utilise différents paramètres à chaque couche, un RNN partage les mêmes paramètres (**parameter sharing**) (ici : U, V, W) à travers toutes les étapes. Cela reflète le fait que nous effectuons la même tâche à chaque étape, juste avec des entrées différentes. Ce qui réduit le nombre total de paramètres que le réseau devra apprendre.
- Le schéma ci-dessus présente des sorties à chaque moment, mais selon la tâche, ceci peut ne pas être nécessaire. Par exemple, lors de la prédiction du sentiment véhiculé par phrase, nous ne pouvons nous soucier que de la sortie finale, pas du sentiment après chaque mot. De même, nous ne pouvons pas avoir besoin d'entrées à chaque étape.

4.2.2 Apprentissage

Pour l'apprentissage d'un RNN on utilise une version légèrement modifiée de la retro propagation appelé **Backpropagation Through Time (BBTT)** [50] [51]. Étant donné que les paramètres sont partagés à tous moment dans le réseau, le gradient à chaque sortie dépend non seulement des calculs du moment actuel, mais aussi les étapes précédentes. On applique alors la règle de la chaîne.

On considère o_t la prédiction du réseau au moment t. On traite la sequence complète (ex : une phrase complète) comme un seul exemple d'apprentissage, donc l'erreur totale est la somme des erreurs à chaque moment (chaque mot).

Le but est de calculer le gradient de l'erreur pour les paramètres U, V, W et d'apprendre de bon paramètre en utilisant Stochastic Gradient Descent (SGD). comme on a sommé les erreurs, on somme également les gradients à chaque étape pour un seul exemple d'apprentissage :

$$\frac{\delta J}{\delta W} = \sum_t \frac{\delta J_t}{\delta W} \tag{14}$$

Pour calculer ces gradients, nous utilisons la règle de la chaîne. Par exemple l'erreur au moment $t = 3$:

$$\begin{aligned} \frac{\delta J_3}{\delta V} &= \frac{\delta J_3}{\delta o_3}\frac{\delta o_3}{\delta V} \\ &= \frac{\delta J_3}{\delta o_3}\frac{\delta o_3}{\delta z_3}\frac{\delta z_3}{\delta V} \end{aligned} \tag{15}$$

Avec $z_3 = Vs_3$. La remarque importante ici c'est que $\frac{\delta J_3}{\delta V}$ dépend uniquement des valeurs du moment actuel o_3, s_3

Mais c'est différent pour $\frac{\delta J_3}{\delta W}$ (et pour U). Pour comprendre pourquoi, on écrit la règle en chaîne suivante :

$$\frac{\delta J_3}{\delta W} = \frac{\delta J_3}{\delta o_3}\frac{\delta o_3}{\delta s_3}\frac{\delta s_3}{\delta W} \tag{16}$$

On note que $s_3 = f(Ux_3 + Ws_2)$ dépend de s_2 qui dépend de W et de s_1 et ainsi de suite. Donc si on prend la dérivée par rapport à W on ne peut pas traiter s_2 comme une constante, on doit appliquer à nouveau la règle de la chaîne et ce qu' on obtient, c'est ceci :

$$\frac{\delta J_3}{\delta W} = \sum_{k=0}^{3} \frac{\delta J_3}{\delta o_3}\frac{\delta o_3}{\delta s_3}\frac{\delta s_3}{\delta s_k}\frac{\delta s_k}{\delta W} \tag{17}$$

On somme la contribution au gradient de chaque moment. En d'autres termes, puisque W est utilisé à chaque moment jusqu'à la sortie, on doit retro propager le gradient de $t = 3$ vers $t = 0$ à travers tout le réseau

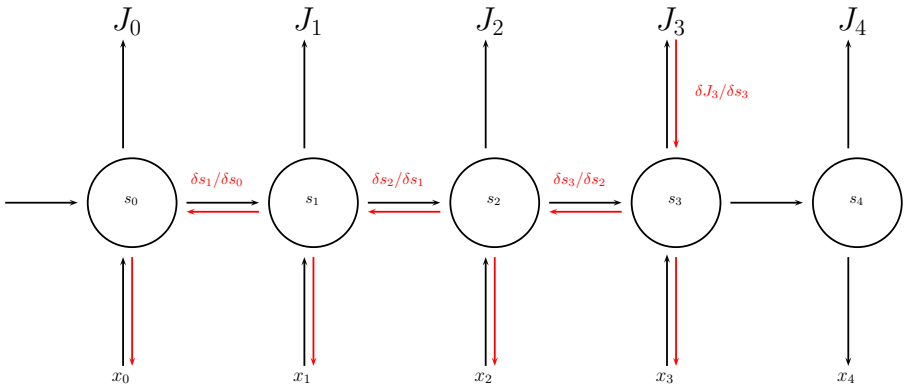

FIGURE 30 – Backpropagation Through Time

Les RNN ont des difficultés à apprendre des dépendances sur le long terme et les interactions entre des mots qui sont éloignes les uns des autres [52]. C'est problématique parce que la signification d'une

phrase est souvent déterminée par des mots qui ne sont pas très proches. Dans ce cas précis si la fonction d'activation est une *tanh* ou bien une sigmoïde alors nous aurons le problème du **vanishing gradient**

C'est pour cette raison qu'on se tourne vers la fonction d'activation ReLu qui elle ne souffre pas de ce problème, mais il existe une solution encore plus utilisé c'est l'utilisation des architectures **Long Short Term Memory (LSTM)** [53] ou **Gated Recurrent Unit (GRU)** [54]

4.2.3 Application

Les RNN ont connu un grand succès dans de nombreuses tâches de traitement du langage naturel. Les plus grands exploits des RNN ont été accomplie par l'architecture LSTM car ils sont bien meilleurs pour capturer des dépendances à long terme.

- **Modélisation du langage et génération de texte**[55][56][57] : Compte tenu d'une séquence de mots, nous voulons prédire la probabilité de chaque mot sachant les mots précédents.

- **Traduction automatique** [58] [59] [60] : La traduction automatique est semblable à la modélisation du langage en ce sens que l'entrée est une séquence de mots dans une langue source (par exemple, l'arabe). Nous voulons générer une séquence de mots dans une langue cible (par exemple, l'anglais). Une différence majeure est que notre sortie ne commence que lorsque nous avons vu l'entrée complète, car le premier mot de la phrases traduites nécessite des informations capturées à partir de la séquence d'entrée complète.

- **La reconnaissance vocale** [61] :La reconnaissance vocale est une technique informatique qui permet d'analyser la voix humaine pour la transcrire sous la forme d'un texte exploitable par une machine.

- **Description des images** [62][63][64] : Ensemble avec les réseaux de neurones convolutifs, les RNN ont été utilisés pour générer des descriptions pour des images non étiquetées. Il est tout à fait incroyable de voir à quel point cela semble fonctionner. Le modèle combiné aligne même des mots générés avec des caractéristiques trouvées dans les images.

4.3 deep generative model

Alors qu'un modèle discriminatif (ex : CNN, RNN, MLP) essaye de prédire $p(y|x)$ avec y étant le label et x l'entrée, un modèle génératif décrit comment les données sont générer, il apprend $p(x, y)$ et fait des prédictions en utilisant la loi de Bayes pour calculer $p(y|x)$ [65].

Si le but est juste la classification, alors il faut utiliser un modèle discriminatif, cependant les modèles génératifs sont capables de bien plus que la simple classification comme par exemple générer de nouvelles observations.

Voici quelques exemple de modèle génératif :

- **Boltzmann Machines**[66]
- **Restricted Boltzmann Machines** [67][68]
- **Deep Belief Networks**[69]
- **Deep Boltzmann Machines**[70][71]
- **Generative Adversarial Networks**[72][73][74]
- **Generative Stochastic Networks**[75]
- **Adversarial autoencoders**[76]

5 Optimisation pour l'apprentissage en Deep Learning

5.1 Les variantes de la descente de gradient

La descente de gradient est la méthode la plus célèbre pour optimiser un réseau de neurones où on doit minimiser une fonction objective $J(\theta)$ caractérisée par les paramètres θ en les mettant à jour dans

la direction opposée de celle de gradient de la fonction objective. Le taux d'apprentissage η détermine l'importance de l'étape qu'on va prendre pour atteindre le minimum local.

Il existe trois variantes de cette méthode. En fonction de la quantité de données, nous ferons un compromis entre la precision de la mise à jour des paramètres et le temps d'exécution de cette mise à jour.

5.1.1 Batch gradient descent

C'est la descente de gradient classique, on calcule le gradient de la fonction coût aux paramètres θ pour tout l'ensemble d'apprentissage

$$\theta = \theta - \eta.\nabla_\theta.J(\theta) \tag{18}$$

Comme on doit calculer le gradient pour tout l'ensemble de données pour exécuter juste une seule mise à jour, cette méthode peut être très lente et irréalisable si les données ne peuvent pas être stockées en mémoire.

Avec cette méthode on est sur de converger vers l'optimum global si la fonction cout est convexe et vers un optimum local si elle est non convexe.

5.1.2 Descente de gradient stochastique

SGD (la descente de gradient stochastique) met à jour les paramètres pour chaque exemple de l'ensemble de données $x^{(i)}$ et label $y^{(i)}$

$$\theta = \theta - \eta.\nabla_\theta.J(\theta; x^{(i)}; y^{(i)}) \tag{19}$$

Cette méthode est plus rapide mais les mises à jours des paramètres trop fréquentes causent à la fonction objectif des oscillations, ces oscillations d'une part permettent d'atterrir dans des minimums locaux potentiellement meilleurs mais d'autre part rendent la convergence plus difficile. Cependant il à été montré qu'en baissant le taux d'apprentissage η, SGD montre la même convergence que la batch gradient descent.

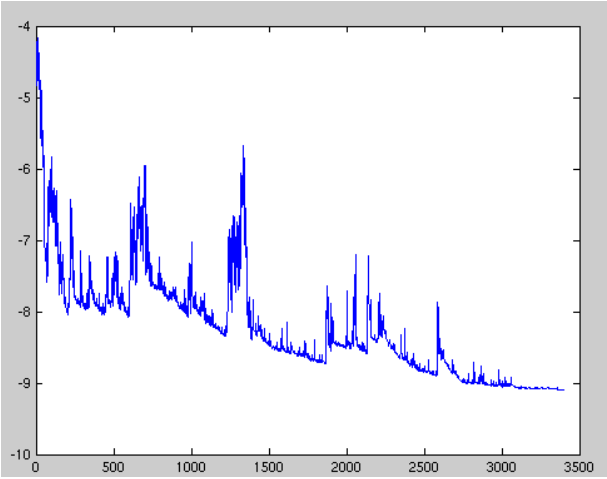

FIGURE 31 – Les oscillations causées par SGD

5.1.3 Mini-batch gradient descent

Cette méthode prends le meilleur des deux méthodes et met à jour les paramètres pour chaque mini groupes de n exemples :

$$\theta = \theta - \eta.\nabla_\theta.J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{20}$$

Cette méthode réduit la variance des mises à jour des paramètres ce qui conduit à une convergence plus stable. Généralement les mini groupes contiennent de 50 à 256 exemples. C'est une méthode de choix pour entraîner un réseau de neurones.

La descente de gradient classique n'offre pas toujours une bonne convergence et pose quelques challenges qui ont besoin d'être résolus :

- Le choix du taux d'apprentissage est difficile, si il est trop petit cela peut emmener à une convergence trop lente, si il est trop grand il peut causer des oscillations à la fonction coût voir même ne pas converger du tout.
- Le même taux d'apprentissage est appliqué à tous les paramètres. Si les caractéristiques des observations dont on dispose ont une fréquence différentes, on peut vouloir appliquer une plus grande mise à jour aux paramètres pour les caractéristiques qui reviennent plus rarement.
- Un autre challenge de la minimisation des fonctions coût non convexe très courantes dans les réseaux de neurones c'est éviter d'être piéger dans des optimums locaux. Dauphin et al. [77] soutiennent que la difficulté ne survient pas des minimum locaux mais des points col (saddle point).

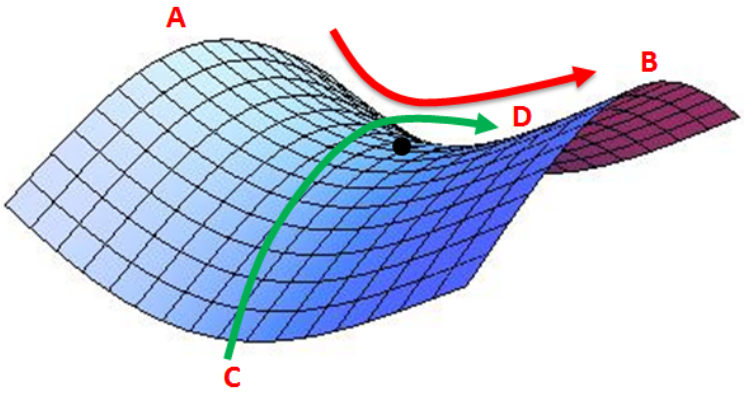

FIGURE 32 – Saddle point : Le point noir joue le rôle du minimum par rapport à l'axe A-B et du maximum par rapport à l'axe C-D

5.2 Algorithmes d'optimisation de la descente de gradient

Dans ce qui va suivre, on exposera quelques algorithmes largement utilisés par la communauté du Deep Learning pour résoudre les challenges cités précédemment.

5.2.1 Momentum

SGD a du mal dans les régions ou la surface est beaucoup plus courbée en une dimension plutôt que dans une autre [78], et ils sont communs autour des minimums locaux. Dans ces cas la, SGD oscille à travers les pentes de ces régions et n'achève que des progrès hésitant vers des optimums locaux. Une des méthodes qui peut aider le réseau à se sortir de ces pièges c'est d'utiliser le coefficient du momentum [79].

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{21}$$

ou $\gamma \in [0, 1]$ représente le coefficient de momentum

Lors de l'utilisation du momentum, on pousse une balle en bas d'une colline. La balle accumule du momentum alors qu'elle roule vers le bas devenant de plus en plus rapide. La même chose se passe lors

de la misa à jour des paramètres. Le momentum augmente pour les dimensions dont le gradient pointe vers la même direction et réduit les mises à jours dont le gradient change de direction. comme résultat, on a une convergence plus rapide et on réduit les oscillation.

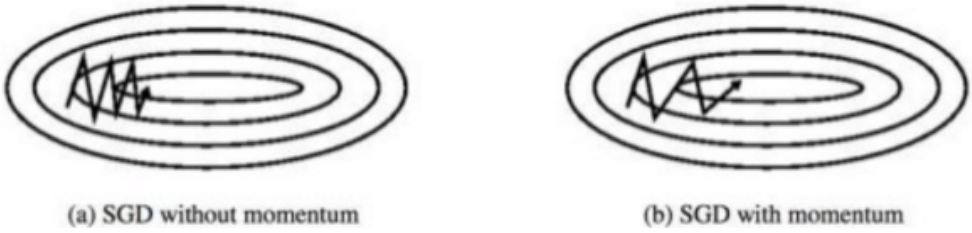

FIGURE 33 – Accélération de SGD par la méthode du momentum et réduction des oscillation

5.2.2 Nesterov accelerated gradient

Sutskever et al. (2013) ont introduit le momentum de Nesterov [80] qui est inspiré de la méthode de Nesterov's Accelerated Gradient (NAG) [81] et qui est une variante du momentum classique et qui consiste à mettre à jour les paramètres de la façon suivante :

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{22}$$

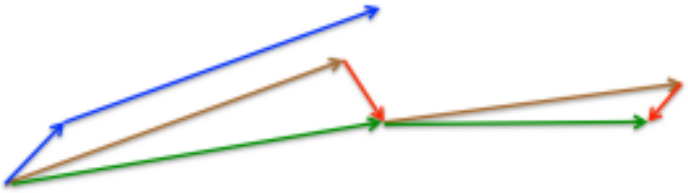

FIGURE 34 – Nesterov accelerated gradient (Source : G. Hinton's lecture 6c)

Alors que le momentem classique calcul le gradient courant (petit vecteur bleu) ensuite fait un grand pas vers la direction de gradient cumulé(grand vecteur bleu), NAG fait d'abord un grand pas dans la direction de gradient cumulé précédemment (vecteur marron) , mesure le gradient puis fais la correction (vecteur vert). Cette méthode a augmenté de façon significative les performances des réseaux de neurones récurrents dans de nombreuses taches [82].

5.2.3 Adagrad

Adagrad [83] est un algorithme d'optimisation basé sur la descente de gradient qui ne fait qu'adapter le taux d'apprentissage aux paramètres, effectuant de plus grandes mises à jour pour les caractéristiques peu fréquentes et de plus petites pour les caractéristiques plus fréquentes. Dean et al. [84] ont trouvé que Adagrad a grandement amélioré la robustesse de SGD et utilisé pour l'apprentissage de grands réseaux de neurones chez Google qui -entre autres- ont appris à reconnaître les chats dans les vidéos Youtube [85]. De plus Pennington et al. [86]ont utilisé Adagrad pour l'apprentissage de GloVe word embeddings, comme les mots les moins fréquents demandent de plus grandes mises à jour que ceux qui sont plus fréquents.

Précédemment, il a été effectué les mises à jour des paramètres θ en utilisant le même taux d'apprentissage η. Adagrad utilise un taux d'apprentissage diffèrent pour chaque paramètre θ_i et à chaque étape t.

Soit $g_{t,i}$ le gradient de la fonction objective sachant le paramètre θ_i à l'étape t :

$$g_{t,i} = \nabla_\theta J(\theta_i) \tag{23}$$

La mise à jour de chaque paramètre θ_i à l'étape t devient :

$$\theta_{t+1,i} = \theta_{t,i} - \eta.g_{t,i} \tag{24}$$

Dans sa règle de mise à jour, Adagrad modifie le taux d'apprentissage général η à chaque étape t pour chaque paramètre θ_i basé sur les gradients passés calculés pour θ_i :

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}}.g_{t,i} \tag{25}$$

$G_t \in \mathbb{R}^{d\times d}$ est une matrice diagonale où chaque élément i, i est la somme des carrés de gradient sachant θ_i à l'étape t et ϵ généralement fixé à $1e-8$ est utilisé pour éviter la division par zero. Fait intéressant, sans l'opération racine carrée, l'algorithme produit des résultats plus mauvais.

Un des avantages de Adagrad c'est qu'il élimine le besoin de modifier manuellement le taux d'apprentissage.

L'inconvénient majeur de Adagrad est l'accumulation des carrés des gradients dans le dénominateur. Comme chaque terme ajouté est positif, l'accumulation des sommes continue d'augmenter au cours de l'apprentissage ce qui va faire baisser le taux d'apprentissage et éventuellement devenir infinitésimalement petit au point que l'algorithme n'est plus capable d'acquérir de nouvelles connaissances.

5.2.4 RMSprop

RMSprop est une méthode de taux d'apprentissage adaptative non publiée proposée par Geoff Hinton dans Lecture 6e de sa Coursera Class [87] . Au lieu d'accumuler tous les carrés des gradients précédents, on restreint la fenêtre des gradients accumulés à une taille fixée w.

Au lieu de stocker les w carrés des gradients précédents, on applique une moyenne mobile exponentielle des carrés des gradients précédents. La moyenne courante $E[g^2]_t$ à l'étape t dépend uniquement de la moyenne précédente et de gradient courant.

$$\begin{aligned} E[g^2]_t &= \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}}.g_t \end{aligned} \tag{26}$$

Hinton suggère que γ soit fixé a 0.9, tandis qu'une bonne valeur par default pour le taux d'apprentissage η est de 0.001.

5.2.5 Adam optimizer

Adaptive Moments (ADAM) optimizer [88] est une autre méthode qui calcule un taux d'apprentissage adaptatif pour chaque paramètre. En plus de stocker une moyenne décroissante exponentielle des précédents carrés des gradients v_t comme RMSprop, ADAM conserve aussi moyenne décroissante exponentielle des précédents gradients m_t

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{27}$$

m_t et v_t sont respectivement des estimations d'ordre 1 (la moyenne) et d'ordre 2(la variance) de gradient. Comme m_t et v_t sont initialisés comme des vecteurs de 0, les auteurs ont observé qu' ils sont biaisés vers zero surtout durant les premières étapes, et surtout lorsque le coefficient de décroissance est

petit (β_1, β_2 proche de 1) alors ils ont calculé une correction des biais pour les estimations du premier et du deuxième ordre :

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{28}$$

Puis vient la mise à jour des paramètres θ :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{29}$$

Les auteurs proposent des valeur par default de 0.9 pour β_1 , 0.999 pour β_2 et 10^{-8} pour ϵ

5.3 Les méthodes du second ordre

Un autre groupe de méthodes d'optimisation dans le contexte du Deep Learning est basé sur la méthode de Newton :

$$\theta = \theta - [HJ(\theta)]^{-1}\nabla J(\theta) \tag{30}$$

Ici, $HJ(\theta)$ est la matrice Hessienne qui est la matrice carrée des deuxièmes dérivées partielles de la fonction. Le terme $\nabla J(\theta)$ est le gradient

La matrice Hessienne décrit la courbure locale de la fonction coût, ce qui permet des mises à jour plus efficaces. En particulier, la multiplication par l'inverse de la matrice Hessienne conduit l'optimisation à prendre des pas plus grands dans la direction des courbures peu profondes et inversement dans les courbures raides. Les adeptes citent que l'absence de tout taux d'apprentissage dans la formule de mise à jour est un grand avantage par rapport aux méthodes du premier ordre.

Cependant, la formule de mise à jour ci-dessus n'est pas pratique pour la plupart des modèles de Deep Learning car le calcul (et l'inversion) de la matrice Hessienne dans sa forme explicite est un processus très coûteux en espace et en temps. Par exemple, un réseau de neurones avec un million de paramètres aurait une matrice Hessienne de taille $[10^6, 10^6]$ occupant approximativement 3725 gigabytes de RAM. Par conséquent, une grande variété de méthodes quasi-Newton ont été développées qui cherchent à rapprocher la matrice Hessienne inverse. Parmi ces méthodes, la plus populaire est Limited-Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [89], qui utilise l'information dans les gradients au fil du temps pour former l'approximation implicitement.

Un des inconvénients de l'application naïve de L-BFGS est qu'elle doit être calculée sur tout l'ensemble d'apprentissage qui peut contenir des millions d'exemples contrairement à mini-batch SGD. Faire fonctionner L-BFGS sur des mini-batch est plus difficile et relève plus de la recherche. Figure 35

5.4 Les limites théoriques de l'optimisation

Plusieurs résultats théoriques montrent qu'il existe une limite de performance de n'importe quel algorithme d'optimisation qu'on peut concevoir pour un réseau de neurones [90] [91].

Certains résultats théoriques s'appliquent uniquement au cas où les unités de réseaux ne produisent que des valeurs discrètes. Des résultats théoriques révèlent qu'il existe des classes de problèmes insolubles, mais il peut être difficile de dire si un problème particulier appartient à cette classe. D'autres résultats montrent que trouver une solution pour un réseau d'une taille donnée est insoluble, mais en pratique, nous pouvons trouver une solution facilement en utilisant un réseau plus large pour lequel de nombreux paramètres correspondent à une solution acceptable. En outre, dans le contexte de l'apprentissage d' un réseau de neurones, nous ne nous soucions généralement pas de trouver le minimum exact d'une fonction, mais nous cherchons seulement à réduire sa valeur de manière suffisante pour obtenir une bonne erreur de test. L'analyse théorique de savoir si un algorithme d'optimisation peut atteindre cet objectif est extrêmement difficile [42].

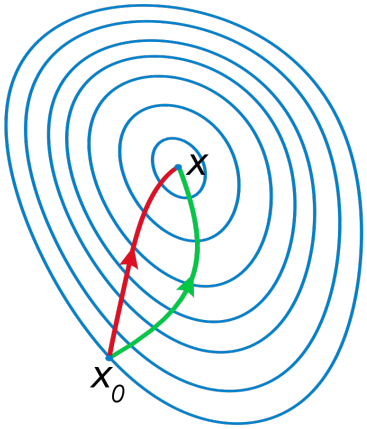

FIGURE 35 – Comparaison entre la descente de gradient (vert) et la méthode Newton (rouge) pour la minimisation d'une fonction. la méthode Newton utilise les informations sur la courbure pour prendre des chemins plus directes.

6 Les fonctions d'activation

la fonction d'activation est une fonction mathématique appliquée à un signal en sortie d'un neurone artificiel. Le terme de "fonction d'activation" vient de l'équivalent biologique "potentiel d'activation", seuil de stimulation qui, une fois atteint entraîne une réponse du neurone. La fonction d'activation est souvent une fonction non-linéaire. Leur but est de permettre aux réseaux de neurones d'apprendre des fonctions plus complexes qu'une simple régression linéaire car le fait de multiplier les poids d'une couche cachée est juste une transformation linéaire.

6.1 Caractéristiques

- **Non-linéarité** [92] : Quand une fonction est non linéaire, un réseau neuronal à deux couches peut être considéré comme un approximateur de fonction universel.
- **Partout différentiable** : Cette propriété permet de créer des optimisations basées sur les gradients.
- **Étendue** [93] : Quand la plage d'activation est finie, les méthodes d'apprentissage basées sur les gradients sont plus stables (impact sur un nombre de poids limités). Quand la plage est infinie, l'apprentissage est généralement plus efficace (impact sur davantage de poids).
- **Monotone** : Lorsque la fonction est monotone, la surface d'erreur associée avec un modèle monocouche est certifié convexe.
- **dérivée monotone** : Les fonctions à dérivée monotone ont été montrées comme ayant une meilleure capacité à généraliser dans certains cas. Ces fonctions permettent d'appliquer des principes comme le rasoir d'Ockham.
- **Identité en 0** : Ces fonctions permettent de faire un apprentissage rapide en initialisant les poids de manière aléatoire. Si la fonction ne converge pas vers l'identité en 0, alors un soin spécial doit être apporté au lors de l'initialisation des poids.

6.2 Quelque fonction d'activation

6.2.1 Rectified Linear Units et leur généralisations

Les ReLu [94] utilisent la fonction d'activation

$$g(z) = max\{0, z\} \tag{31}$$

ReLu sont faciles à optimiser car ils sont tellement similaires aux unités linéaires. La seule différence entre une unité linéaire et ReLu est que ReLu délivre zéro sur tous $\mathbb{R}^-$ ceci rend les dérivées via ReLu

importantes chaque fois que l'unité est active.

Des recherche montrent que ReLu sont un modèle encore meilleur de neurones biologiques [95] et produisent une performance égale ou meilleure que la tangente hyperbolique et un de leur avantage majeur est la réduction de la probabilité du **vanishing gradient**. Cela se produit lorsque $z > 0$. Dans ce cas, le gradient a une valeur constante puisque la dérivée est égale à 1 partout. En revanche, le gradient des sigmoïdes devient de plus en plus petit car la valeur absolue de z augmente. Le gradient constant de ReLu entraîne un apprentissage plus rapide.

Lors de l'initialisation des paramètres il peut être judicieux d'initialiser les valeurs des biais à une petite valeur positive, telle que 0,1. Il est très probable que ReLu seront initialement actives pour la plupart des entrées de l'ensemble d'apprentissage et permettent aux dérivées de passer à travers.

Il existe plusieurs généralisations de ReLu. La plupart de ces généralisations fonctionnent de manière comparable et de temps à autre, se comportent mieux.

Un inconvénient de ReLu est qu'ils ne peuvent pas apprendre par la descente de gradient sur des exemples pour lesquels leur activation est nulle *"dying ReLu" problem*. Une variété de générations de ReLu garantit qu'elles reçoivent le gradient partout.

Les généralisations de ReLu sont basées sur l'utilisation d'une pente non nulle α_i lorsque $z_i < 0$: $h_i = g(z, \alpha) = max(z_i, 0) + \alpha_i min(0, z_i)$. **Absolute value rectification** fixe $\alpha_i = -1$ pour obtenir $g(z) = |z|$. **Leaky ReLu (LReLu)** [96]fixe α_i à une petite valeur comme 0.01 alors que **parametric ReLu (PReLu)** [97] traite α_i comme un paramètre apprenable.

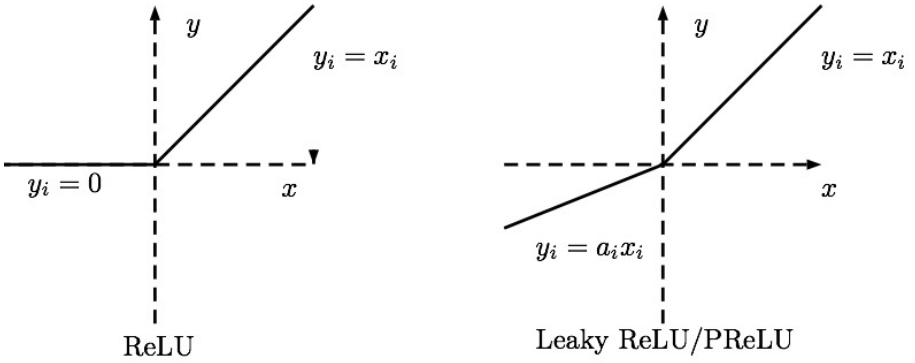

FIGURE 36 – (à gauche)La fonction d'activation ReLu. Cette fonction d'activation est la fonction d'activation par défaut recommandée pour l'utilisation des réseaux de neurones feedforward. L'application de cette fonction à la sortie d'une transformation linéaire donne une transformation non linéaire. Ils préservent plusieurs des propriétés qui rendent les modèles linéaires faciles à optimiser avec la descente de gradient. (à droite) Leaky ReLu et Parametric ReLu qui corrige l' inconvénient de ReLu et qui sont différentiables à 0

6.2.2 Sigmoïde et tangente hyperbolique

Avant l'introduction de ReLu, la plupart des réseaux neuronaux utilisent la Sigmoïde ou tanh :

la Sigmoïde :

$$g(z) = \frac{1}{1 + e^{-\lambda z}} \tag{32}$$

tanh :

$$g(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}} \tag{33}$$

Ce qu'on reproche aux unités sigmoïdales c'est qu' elles saturent dans la plupart de leur domaine : elles saturent à une valeur élevée lorsque z est très positif et à une valeur faible lorsque z est très négatif et ne sont que très sensibles à leur entrée lorsque z est proche de 0. La saturation généralisée des unités sigmoïdales peut rendre l'apprentissage par la méthode de la descente de gradient très difficile. Pour cette raison, leur utilisation dans les couches cachées est maintenant déconseillée.

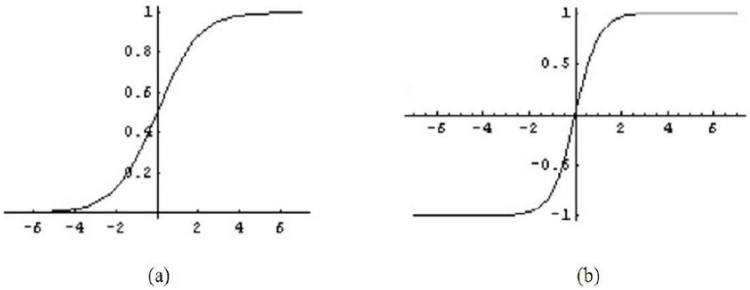

FIGURE 37 – (a) Sigmoïde. (b) tangente hyperbolique.

7 Conclusion

Dans cette partie, nous avons vu qu'est ce que le Deep Learning et comment il se différencie des algorithmes de ML traditionnelles. Nous avons vu quelques étapes majeures de son évolution et les exploits qui ont été accomplis avec. Nous avons introduit quelques méthodes utilisée par la communauté du Deep Learning et nous avons expliqué le principe de chacun. Nous avons également parlé des trois familles majeures de modèle à savoir les réseaux convolutifs, les réseaux récurrents ainsi que les modèles génératifs, ces derniers font encore l'objet de recherches intensives.

Contribution

1 Introduction

Après avoir étudié dans la deuxième partie l'état de l'art des différents outils du Deep Learning, nous allons présenter dans cette partie une étude comparative entre les différentes méthodes utilisées afin d'améliorer les performances d'un modèle que ce soit en temps ou en efficacité.

2 Problématique étudié

Nous allons nous intéresser à la problématique de la classification d'image qui est la tâche d'attribuer à une image d'entrée x un label y à partir d'un ensemble fixe de catégories. C'est l'un des problèmes fondamentaux de la vision par ordinateur qui, malgré sa simplicité, a une grande variété d'applications pratiques.

Plusieurs bases ont été présentées dans la littérature, nous avons choisis la base **CIFAR-10** [98]. Cette base se compose de 60 000 petites images couleur de 32 pixels de hauteur et largeur. Chaque image est étiquetée avec l'une des 10 classes (par exemple « avion, automobile, oiseau, etc.»). Ces 60 000 images sont divisées en un ensemble d'apprentissage de 50 000 images et un ensemble de tests de 10 000 images. Dans l'image ci-dessous.

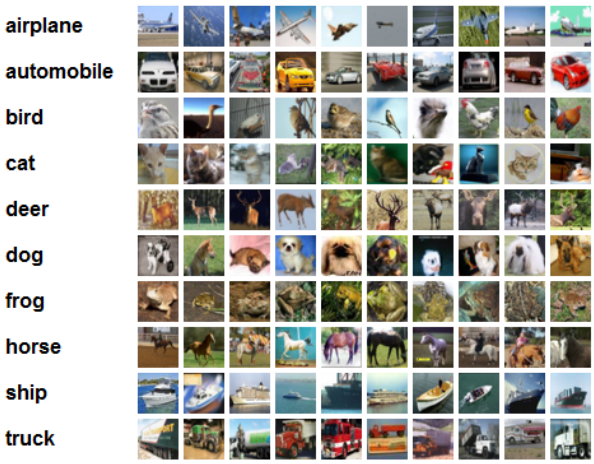

FIGURE 38 – Exemples de 10 images aléatoires de chacune des 10 classes

3 Présentation des outils

3.1 Le software

Plusieurs framework open sources sont disponibles dans la littérature, la grande majorité supporte le langage Python. Voici une liste non exhaustive de quelques framework.

3.1.1 Theano

Cré par Frédéric Bastien et l'équipe de recherche derrière le laboratoire de l'Université de Montréal, MILA.

- Avantages
  - — Python
  - — Performant si utilisé correctement
- Inconvénients
  - — Le supporte du Multi GPU nécessite une solution de contournement

— Les grands modèles peuvent nécessiter un long temps de compilation
— API bas niveau

3.1.2 TensorFlow

TensorFlow fut cré par l'équipe Google Brain pour mener des recherches sur le ML et le Deep Learning. Il est considéré comme une version moderne de Theano.

- Avantages
  — Python
  — supporté par Google
  — Une très grande communauté
  — Le support du multi-GPU
- Inconvénients
  — Plus lent que les autres framework dans de nombreux benchmarks, bien que Tensorflow se rattrape.
  — Le soutien des RNN est encore surclassé par Theano

3.1.3 Keras

Le framework le plus haut niveau, le plus convivial de la liste. Il permet aux utilisateurs de choisir si les modèles qu'ils construisent sont exécutés sur Theano ou TensorFlow. Il est écrit et entretenu par Francis Chollet, un autre membre de l'équipe Google Brain.

- Avantages
  — Python
  — Le backend par excellence pour Theano ou TensorFlow
  — Interface haut niveau, intuitive
- Inconvénients
  — Moins flexible que les autres API

3.1.4 PyTorch

Le dernier né de la liste. PyTorch est soutenu par l'équipe de recherche en intelligence artificielle de Facebook. Il permet le traitement des entrées et des sorties de longueur variable (**dynamic computation graphs**), ce qui est utile lorsque on travaille avec des RNN. Une caractéristique absente de TensorFlow et Theano.

- Avantages
  — Python
  — Soutenu par Facebook.
  — Mélange d'API de haut niveau et de bas niveau.
  — Support des graphes dynamiques.
  — Il semblerait que c'est le meilleur outil pour les chercheurs.
- Inconvénients
  — Beaucoup moins mûr que ses concurrents (Dans leurs propres termes : "We are in an early-release Beta. Expect some adventures.")
  — Références / ressources limitées en dehors de la documentation officielle

3.1.5 Autres frameworks

- **CNTK** : par Microsoft
- **Neon** : par Nervana Systems. Il a récemment été classé comme le frameworks le plus rapide dans plusieurs catégories
- **Deeplearning4j** : Il supporte le langage java
- **Caffe** : par Berkeley Vision and Learning Center

Après ce petit tour d'horizon des différents frameworks disponibles, notre choix s'est porté sur le framework TensorFlow de Google. La raison principale de ce choix c'est la très grande et aussi très active communauté qui est derrière cette librairie.

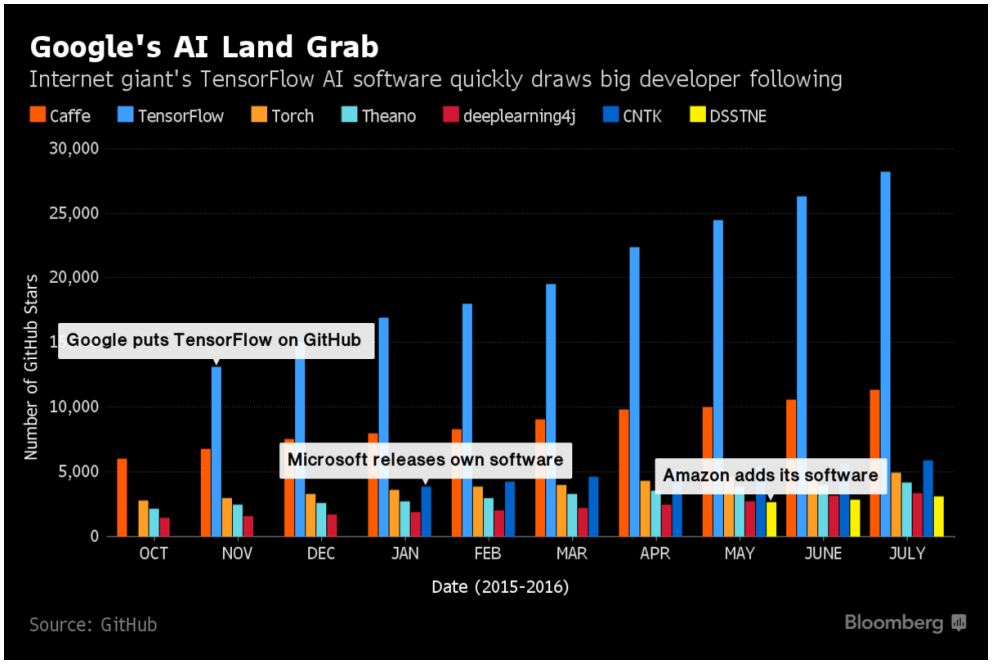

FIGURE 39 – La croissance de la popularité de TensorFlow

3.2 Le hardware

Le Deep Learning est un domaine avec des exigences en calculs intenses et la disponibilité des ressources (surtout en GPU) dédiés à cette tache vont fondamentalement influencer sur l'expérience de l'utilisateur car sans ses ressources, il faudra trop de temps pour apprendre de ses erreurs ce qui peut être décourageant.

Les expérimentations ont tous été effectuées sur une machine qui offre des performances acceptables dont voici les caractéristiques :

CPU	Intel Core i5-4440 (3.1 GHz)
GPU	MSI GEFORCE GTX 770 TWIN FROZR GAMING OC 2GB
RAM	8GB

4 Quelques notions

(Dans tout ce qui va suivre le terme noyau sera remplacé par le terme filtre).

Avant de décrire l'architecture du réseau, il faut d'abord expliquer les paramètres de la couche convolutive pour pouvoir déterminer les dimensions spatiales de sortie de chaque couche.

La sortie d'une couche convolutive est déterminé par la formule suivante :

$$output = \frac{N - F}{s} + 1 \tag{34}$$

ou :

- N : La dimension des entrées.
- F : La dimension du filtre.
- s : Le stride où le pas.

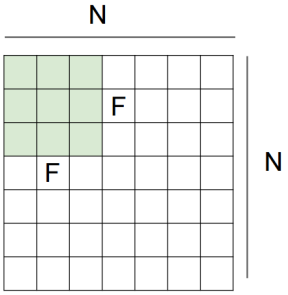

FIGURE 40 – Exemple : L'image d'entrée est de dimension $N = 7$. Supposons un noyau de taille $F = 3$. Si on utilise un pas $s = 1$ nous aurons une image de sortie $output = \frac{7-3}{1} + 1 = 5$. Donc l'image en sortie sera de dimension 5×5

Pour être tout à fait complet sur comment déterminer les dimensions spatiales des sorties, prenons l'exemple de la figure 40 mais avec $s = 3$. D'après la formule 34 :

$$output = \frac{7-3}{3} + 1 = 2.33 \tag{35}$$

Nous obtenons une image en sortie de dimension 2.33×2.33 qui est continue ce qui n'est pas possible.

En pratique on applique ce qu'on appelle un **Zero pad** p aux bordures des entrées. le **Zero pad** est déterminé par la formule suivante :

$$p = \frac{F-1}{2} \tag{36}$$

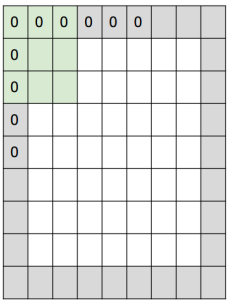

FIGURE 41 – Exemple : D'après la formule 36 : $p = \frac{3-1}{2} = 1$. c.à.d nous ajouterons des pixels d'intensité nulle sur tout le long de la bordure de l'image sur un seul niveau. La nouvelle dimension de l'image sera de 9×9

Maintenant nous allons pouvoir déterminer le volume des sorties ainsi que le nombre de paramètres par couche convolutive. Exemple :

Soit une image en entrée de dimension $32 \times 32 \times 3$ (3 pour le canal RGB). On décide d'appliquer une couche convolutive avec un nombre de filtres $k = 10$ et de dimension $F = 5$ et un stride $s = 1$.

On commence par appliquer le Zero padding :

$$p = \frac{5-1}{2} = 2 \tag{37}$$

La nouvelle dimension de l'image est de $32 + 2 * 2 = 36$. Maintenant avec la formule 34 on calcule la dimension des sorties : $\frac{36-5}{1} + 1 = 32$.

Puisque nous avons 10 filtres dans la couche alors nous obtenons $32 \times 32 \times 10$

Pour calculer le nombre de paramètre de la couche : $5*5*3+1 = 76$ (5*5 pour la dimension du filtre, 3 pour le nombre de canal de l'entrée et 1 pour le biais). Puisque la couche possède 10 filtres alors : le nombre de paramètres de la couche est : $76 * 10 = 760$.

5 Architecture

Nous allons commencer par entraîner un réseau de neurones convolutif des plus classiques. Le réseau possède 2389114 paramètres entraînables et il consiste en 6 couches convolutives, une couche de Max pooling est placé après chaque série de 2 couches convolutives et enfin un MLP à 3 couches. Pour un apprentissage plus rapide, ReLu sera la fonction d'activation.

L'architecture est résumée dans la figure 42

FIGURE 42 – Illustration du modèle de base. La notation **[Conv 3x3 s1, 48 / ReLu]** signifie qu'un filtre de 3×3 sera appliqué à la couche avec stride $s = 1$ et 48 pour le nombre de filtres de la couche, ReLu désigne la fonction d'activation. Le nombre de neurones par couche est : 49,152 - 49,152 - 24576 - 24576 - 12288 - 12288 - 3072 - 512 - 256 - 1000

6 initialisation des paramètres

Lorsque on travaille avec des réseaux de neurones profonds, une bonne initialisation des poids peut être la différence entre une convergence en un temps raisonnable ou non, même après des milliers d'itérations.

Dans le cas d'une sigmoïde ou bien tanh, si les poids sont trop petits, la variance du signal d'entrée commence à diminuer à mesure qu'il traverse chaque couche du réseau. L'entrée aboutit finalement à une valeur très faible et ne peut plus être utile.

Si les poids sont trop importants, la variance des données d'entrée tend à augmenter rapidement avec chaque couche de passage. Finalement, il devient si grand qu'il devient inutile parce que la fonction sigmoïde et tanh tendent à devenir plates pour des valeurs plus grandes. Cela signifie que nos activations deviendront saturées et les gradients commencent à s'approcher de zéro.

L'initialisation du réseau avec les bons poids est très importante si nous souhaitons que le réseau fonctionne correctement. Nous devons nous assurer que les poids soient dans un rang raisonnable avant de commencer à entraîner le réseau. Pour cela, nous utiliserons **Glorot initialization** [99].

6.1 Glorot initialization

Assigner les poids du réseau avant de commencer l'apprentissage semble être un processus aléatoire. Nous ne savons rien sur les données, nous ne sommes donc pas sûrs d'attribuer les bons poids qui fonctionneraient dans ce cas particulier. Une bonne façon est d'assigner les poids à partir d'une distribution gaussienne. Évidemment, cette distribution aurait une moyenne nulle et une variance finie. Considérons un neurone linéaire :

$$y = w_1x_1 + w_2x_2 + ... + w_nx_n + b \tag{38}$$

Avec x l'entrée constituée de n composantes. Avec chaque couche qui passe, nous voulons que la variance reste la même. Cela nous aide à empêcher le signal d'exploser à une valeur élevée ou à disparaître à zéro. En d'autres termes, nous devons initialiser les poids de telle sorte que la variance reste la même pour x et y. Ce processus d'initialisation est connu sous le nom de **Glorot initialization**

6.1.1 Comment ?

Nous voulons que la variance reste la même au fur et a mesure que nous passons dans chaque couche. La variance de y :

$$Var(y) = Var(w_1x_1 + w_2x_2 + ... + w_nx_n + b) \tag{39}$$

Calculons la variance des termes entre les parenthèses du côté droit de l'équation ci-dessus :

$$Var(w_ix_i) = \mathbb{E}(x_i)^2Var(w_i) + \mathbb{E}(w_i)^2var(x_i) + Var(w_i)Var(x_i) \tag{40}$$

Ici, $\mathbb{E}()$ signifie l'espérance d'une variable donnée, qui représente essentiellement la valeur moyenne. Nous avons supposé que les entrées et les poids proviennent d'une distribution gaussienne de moyenne zéro. Par conséquent, le terme $\mathbb{E}()$ disparaît et nous obtenons :

$$Var(w_ix_i) = Var(w_i)Var(x_i) \tag{41}$$

Notons que b est une constante et a une variance nulle, donc il va disparaître. Substituons dans l'équation originale :

$$Var(y) = Var(w_1)Var(x_1) + ... + Var(w_n)Var(x_n) \tag{42}$$

Ensuite, si nous supposons que les x_i et w_i sont tous indépendants et distribués de manière identique, nous pouvons déterminer que la variance de y est :

$$Var(y) = nVar(w_i)Var(x_i) \tag{43}$$

Donc, si nous voulons que la variance de y soit la même que x, alors le terme $nVar(w_i)$ devrait être égal à 1. Par conséquent :

$$\begin{aligned} nVar(w_i) &= 1 \\ Var(w_i) &= 1/n \end{aligned} \tag{44}$$

Nous sommes arrivés à la formule de **Glorot initialization**. Nous devons choisir les poids à partir d'une distribution gaussienne avec une moyenne nulle et une variance de $1/n$.

Dans l'article original, les auteurs prennent la moyenne du nombre de neurones d'entrée et des neurones de sortie. Donc, la formule devient :

$$\begin{aligned} Var(w_i) &= 1/n_{avg} \\ N_{avg} &= \frac{n_{in} + n_{out}}{2} \end{aligned} \tag{45}$$

La raison pour laquelle ils le font est pour préserver le signal lors de la retro propagation aussi.

Pour les fonctions d'activation de type ReLu, la formule suivante est utilisée [100] :

$$Var(w) = \frac{2}{n} \tag{46}$$

Ce qui est logique. une unité ReLu est nulle pour la moitié de ses entrées, donc nous devons doubler la taille de la variance du poids pour maintenir la variance du signal constante.

7 L'apprentissage

Afin de choisir un algorithme d'apprentissage parmi les 6 vus dans la section 5, nous allons expérimenter chacun d'eux avec le même modèle décrit par la figure 42. L'initialisation des poids se fait avec Glorot initialization et les biais sont initialisés à 0.1. Le nombre d'époque sera le même pour les 6 algorithmes. Le taux d'apprentissage est fixé à 10^{-4}. Le momentum est fixé à 0.9. Les hypers paramètres de ADAM et rms sont initialisé à leurs valeurs par default. La taille du batch est de 125 pour l'apprentissage et 100 pour le test. La fonction coût est l'entropie croisée :

$$H(p, q) = -\sum_i p \, log(q) \tag{47}$$

Avec p la sortie désirée et q la sortie du réseau. Voici le résultat :

7.0.1 Discussion

Le graphe n'offre aucune surprise, les 2 algorithmes rms et ADAM qui procèdent à un gradient adaptatif et une accumulation restreinte des gradients passés progresse rapidement et minimise l'erreur de l'apprentissage dés le début du processus. De la correction de nesterov résulte une convergence plus rapide que celle du momentum classique et les 2 convergent plus rapidement que SGD et adagrad. La lente progression de adagrad s'explique par le fait d'accumuler le carré des gradients dans le dénominateur et qui engendre une baisse du taux d'apprentissage mais qui reste malgré tout plus rapide que SGD.

	erreur	temps
SGD	2.174	1h11mn
ADAGRAD	1.953	1h11mn
MOMENTUM	1.293	1h11mn
NAG	1.197	1h11mn
RMS	$8e^{-3}$	1h11mn
ADAM	$5e^{-3}$	1h15mn

TABLE 2 – L'erreur et temps d'exécution de chaque algorithmes après 45 époques

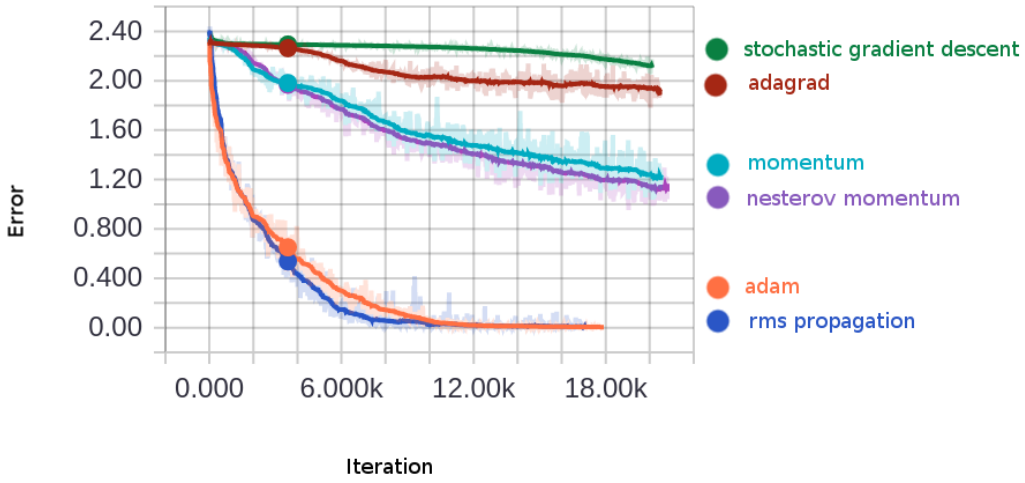

FIGURE 43 – L'évolution de l'erreur d'apprentissage des 6 algorithmes

A partir de maintenant, toutes les prochaines expérimentations se feront avec ADAM comme algorithme d'apprentissage.

Maintenant que nous avons vu la courbe de l'apprentissage, nous allons voir la courbe d'erreur de la phase de test.

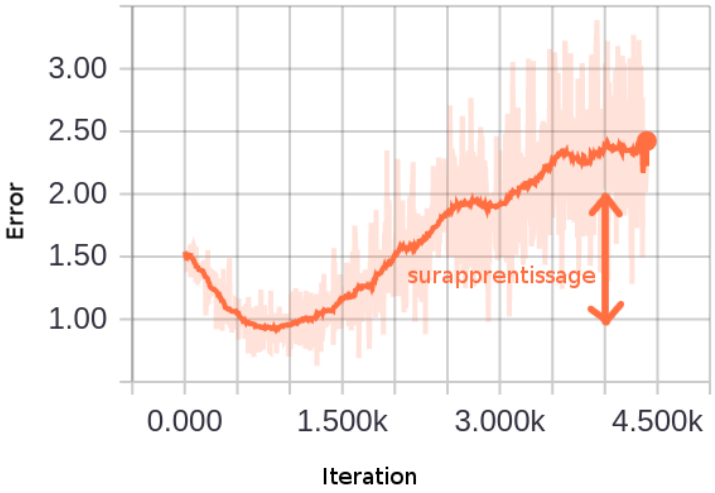

FIGURE 44 – L'évolution de l'erreur de test avec l'algorithme ADAM sur 45 époques

L'erreur est bien minimisée au tout début et atteint sa plus faible valeur après 9 époques, puis après la courbe diverge complètement et l'erreur commence à augmenter alors qu'au même moment l'erreur de l'apprentissage continue de diminuer. C'est le sur-apprentissage

erreur	precision %
2.5	70

TABLE 3 – L'erreur et la precision de l' algorithme lors de la phase de test après 45 époques. Par precision on entend : $precision = \frac{TP}{N} \times 100$. ou VP est le nombre d'exemple correctement classé et N la taille de l'ensemble

Pour corriger le problème du sur-apprentissage, notre modèle 42 à besoin de régularisation.

8 Régularisation

Un problème central du ML est de savoir comment créer un algorithme qui fonctionne bien ; non seulement sur l'ensemble d'apprentissage, mais aussi sur l'ensemble de test. De nombreuses stratégies utilisées dans le Deep Learning sont explicitement conçues pour réduire l'erreur de test, éventuellement au détriment d'une augmentation de l'erreur d'apprentissage. Ces stratégies sont connues sous le nom de régularisation. Nous allons nous intéresser à une technique particulièrement puissante : le **Dropout**.

8.1 Dropout

Le Dropout [101] est une technique où des neurones sélectionnés au hasard sont ignorés (temporairement) pendant l'apprentissage. Cela signifie que leur contribution à l'activation des neurones qui leur succède est temporairement supprimé lors de la phase de propagation et toutes les mises à jour de poids ne sont pas appliquées au neurone lors de la phase de retro propagation.

Lorsque des neurones sont supprimés au hasard du réseau pendant l'apprentissage, les autres neurones devront intervenir et gérer la représentation requise pour faire des prédictions pour les neurones manquants.

Lors de la phase d'apprentissage, pour chaque itération, un neurone est gardé avec une probabilité p, sinon il est supprimé.

Lors de la phase de test, tous les neurones sont gardés, nous voulons donc que les sorties des neurones au moment du test soient identiques à leurs sorties au moment de l'apprentissage. Par exemple, dans le cas ou $p = 0.5$, les neurones doivent réduire de moitié leurs sorties au moment du test pour avoir la même sortie que pendant l'apprentissage.

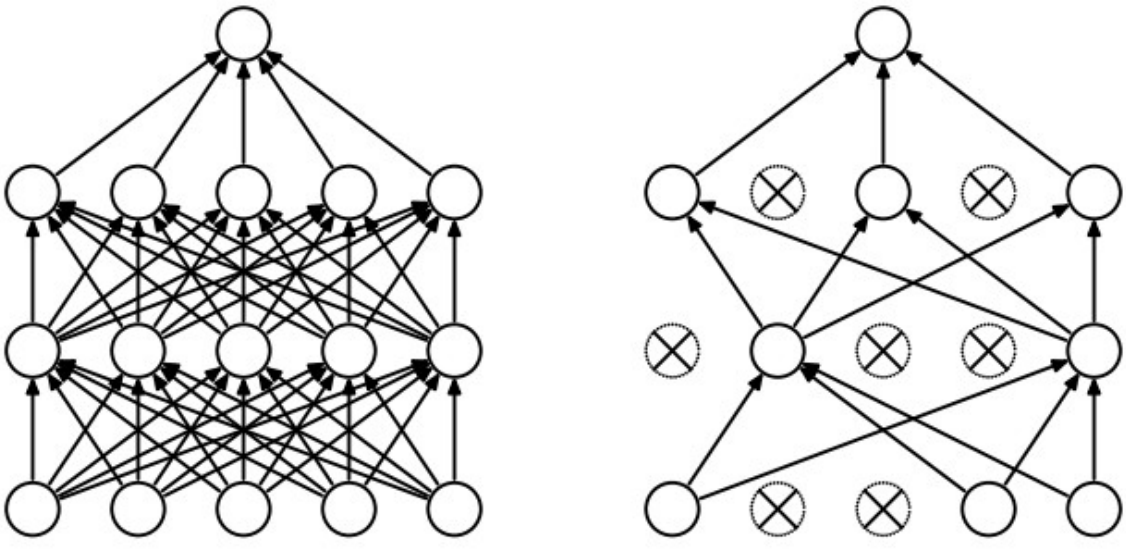

FIGURE 45 – Illustration du dropout lors de l'apprentissage (à droite) et lors du test (à gauche)

Il existe plusieurs interprétation au dropout et les raisons qui font que c'est une technique efficace sont nombreuses :

- Le dropout peut être interprété comme un échantillonnage d'un réseau de neurones dans le réseau de neurones complet (une couche avec h neurones, alors nous avons 2^h réseaux possibles), et mettre à jour seulement les paramètres du réseau échantillonné. La différence c'est que ces sous réseaux partage les paramètres.
- D'autres le considèrent comme une forme d'augmentation des données par corruption artificielle des entrées de chaque couche. Cela élargit considérablement le nombre d'exemples que le modèle utilisera lors de l'apprentissage ce qui peut aider à protéger contre le sur-apprentissage.
- Encore une autre interprétation est qu'il s'agit d'une forme de *bagging* dans lequel un ensemble de modèles ne sont formés que sur un petit sous-ensemble de données.

Maintenant que nous savons comment fonctionne le dropout, nous allons l'introduire dans notre réseau et voir comment il impacte les performances.

FIGURE 46 – Le modèle 2. C'est la même architecture mais régularisé avec la technique du Dropout. **[Max Pool 2x2 s2 / Dropout 0.75]** signifie que chaque neurone de sortie de la couche de pooling à une probabilité $p = 75\%$ d'être gardé. Dans le cas d'une couche convolutive ou de perceptron, le dropout est appliqué après la fonction d'activation.

Voici les nouveaux résultats :

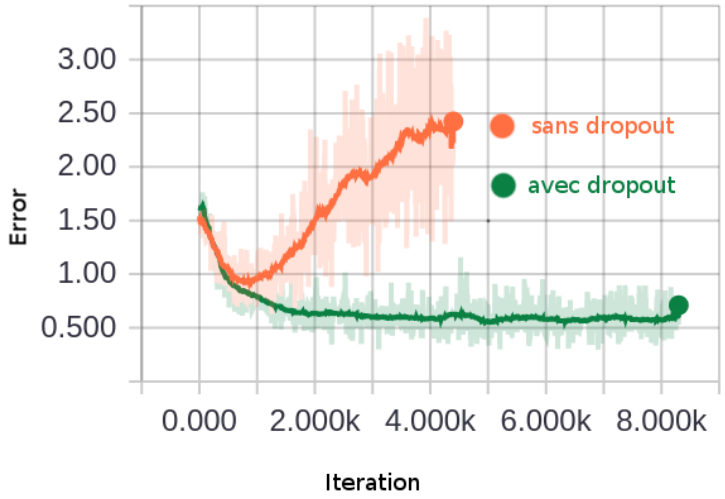

FIGURE 47 – Évolution de l'erreur de test sans dropout (figure 42) et avec dropout (figure 46)

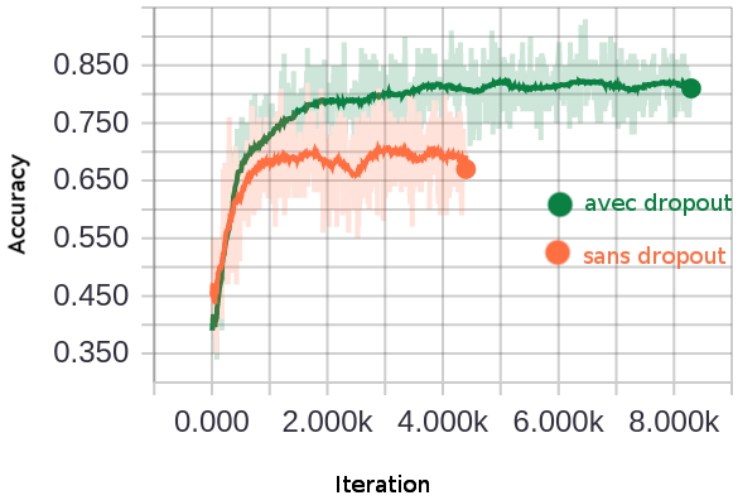

FIGURE 48 – Évolution de la precision de test sans dropout et avec dropout

	erreur	precision %
sans dropout	2.5	70
avec dropout	0.6	80

TABLE 4 – tableau récapitulatif des résultats précédent après 45 époque

A partir des figure 47 et 48 on voit tout de suite que le modèle avec dropout réalise une performance largement meilleure que le modèle sans dropout. Avec le modèle 2 (46), l'erreur est tout aussi rapidement minimisée même avec des neurones en moins lors de l'apprentissage (l'effet du dropout) à la seul différence que : avec dropout, l'erreur continue d'être minimisée pour atteindre sa plus faible valeur après 20 époques pour ne plus diverger après. L'effet se voit tout aussi bien sur la precision où le pourcentage de bonne classification atteint et de 80% après 45 époques. Cette precision continue d'augmenter doucement jusqu'à atteindre 82% après 50 époques.

Maintenant regardons le comportent du réseau lors de l'apprentissage :

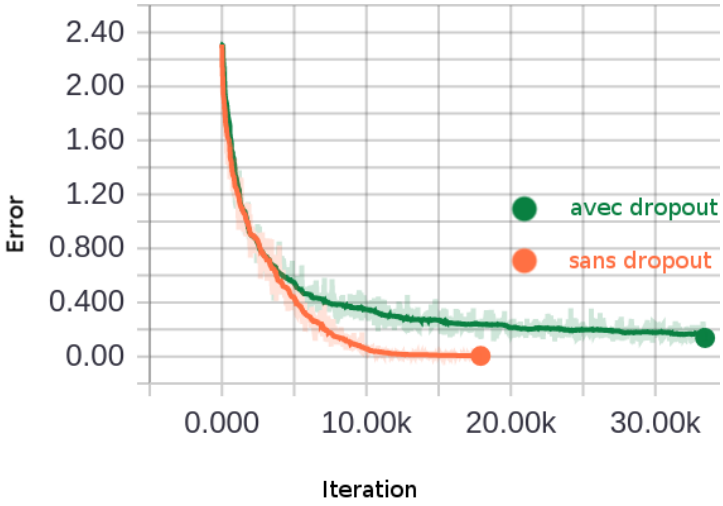

FIGURE 49 – Évolution de l'erreur d'apprentissage avec et sans dropout

	erreur
sans dropout	$5e^{-3}$
avec dropout	0.24

TABLE 5 – Récapitulatif de l'erreur d'apprentissage après 45 époques

Le fait de supprimer aléatoirement et temporairement des neurones ralentit considérablement l'apprentissage et le modèle 2 n'arrive pas à minimiser l'erreur d'apprentissage aussi bien que le faisait le modèle 1 même après 75 époques. Même si le plus important est fait (minimisation de l'erreur de test) on peut penser que les données d'apprentissage contiennent encore quelques caractéristiques qui peuvent aider le réseau lors de l'inférence et qui valent la peine d'être apprises. Pour cela, on a plusieurs solution :

- Laisser durer l'apprentissage
- Augmenter les probabilités du dropout : Cette stratégie n'est pas sure car le fait d'augmenter les probabilités de garder les neurones pourrait ne pas protéger aussi efficacement du sur-apprentissage.
- Changer quelques paramètres du réseau.

9 Exponential Linear Units

Actuellement, la fonction d'activation la plus populaire pour les réseaux de neurones est (**ReLu**). La fonction d'activation ReLu est l'identité pour des entrées positives et zéro sinon. L'avantage principal de ReLu, c'est qu'elle règle le problème du **vanishing gradient**. Toutefois ReLu n'est pas négative et, par conséquent, a une activation moyenne supérieure à zéro. Les unités qui ont une activation moyenne différente de zéro agissent comme biais pour la couche suivante. Si ces unités ne s'annulent pas, l'apprentissage provoque un décalage de biais pour les unités de la prochaine couche. [102]

Exponential Linear Units (ELU) [102] tout comme ReLu, règle le problème du **vanishing gradient** grâce à l'identité pour les entrées positives. Cependant, comparé à ReLu, ELU améliore l'apprentissage car elle a des valeurs négatives qui lui permettent de pousser les activations moyennes de l'unité plus près de zéro ce qui accélère l'apprentissage et conduit à une meilleur precision lors de la classification. ELU est :

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right) & \text{if } x \leq 0 \end{cases}$$

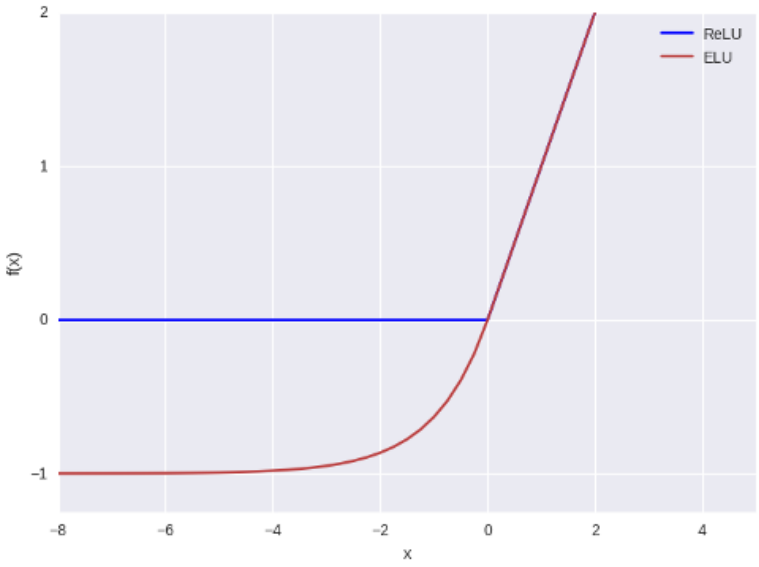

FIGURE 50 – ReLu et ELU (ELU, $\alpha = 1.0$)

Nous allons introduire ELU dans notre réseau :

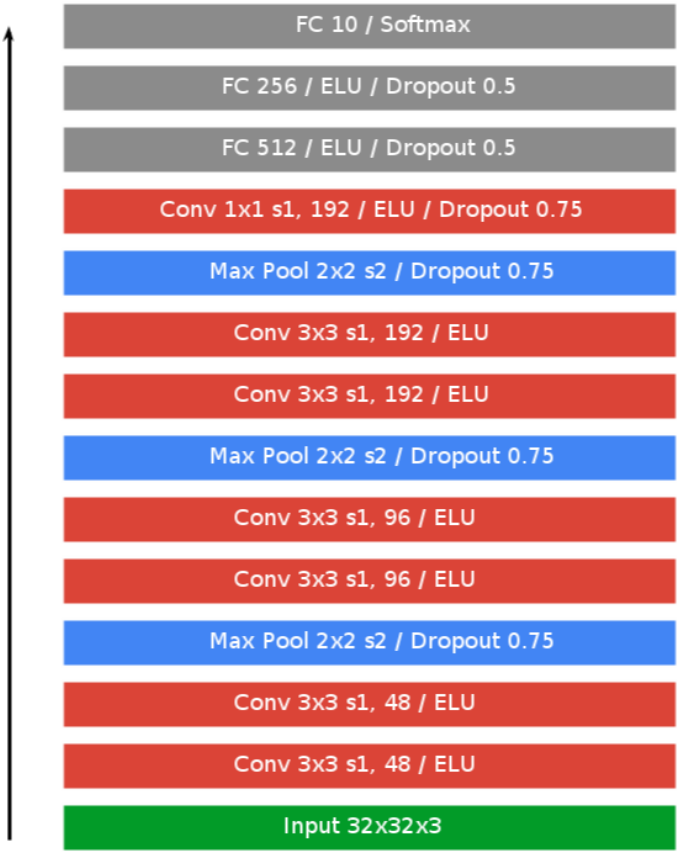

FIGURE 51 – Le modèle 3. Il s'agit de remplacer la fonction d'activation ReLu par ELU

Nous allons comparer les performances du modèle 2 (figure 46) et celui du modèle 3 (figure 51) et voir quels sont les changements qui vont se produire

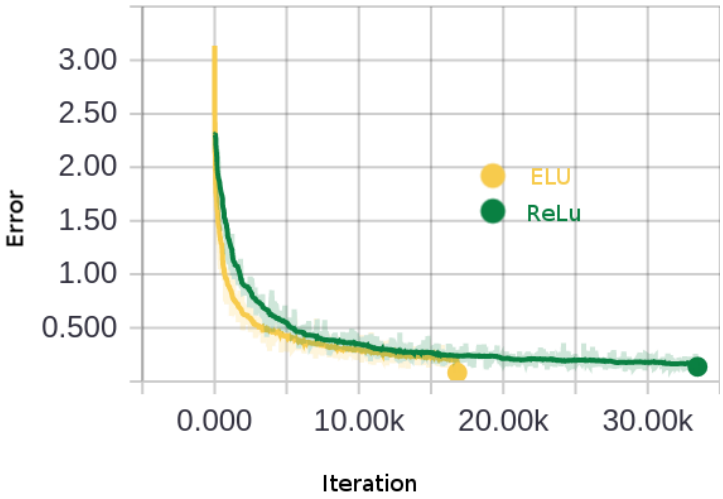

FIGURE 52 – évolution de l'apprentissage du modèle 2 et 3

	erreur
ReLu	0.24
ELU	0.19

TABLE 6 – Récapitulatif de l'erreur d'apprentissage après 45 époques

Avec ses sorties négatives, ELU diminue le décalage des biais car les activations moyennes sont plus proches de zéro et un décalage des biais moindre accélère l'apprentissage. C'est exactement ce qui se passe avec notre réseau. Que ce soit avec ReLu ou ELU, les deux fonctions d'activation mènent à une convergence mais ELU commence plus tôt.

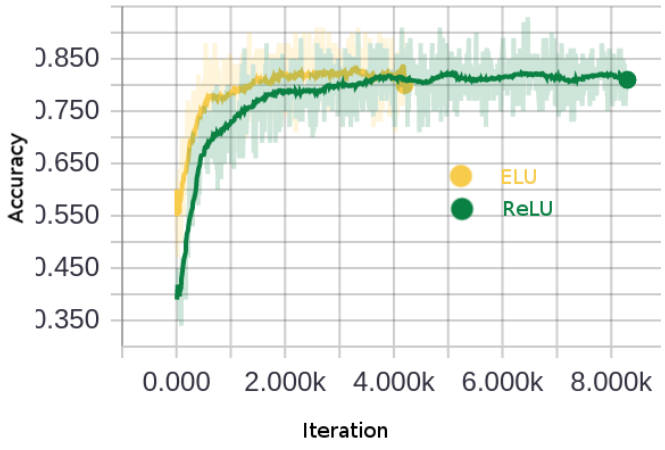

FIGURE 53 – évolution du taux de bonne classification en phase de test des modèles 2 et 3

	precision %
ReLu	80
ELU	83

TABLE 7 – Récapitulatif de la precision en phase de test après 45 époques

Le graphe 53 montre la capacité de ELU à généraliser où le score de 83% est atteint après 33 époques

Voyons si ELU minimise l'erreur d'apprentissage aussi bien que le faisait le modèle 1 (figure 42).

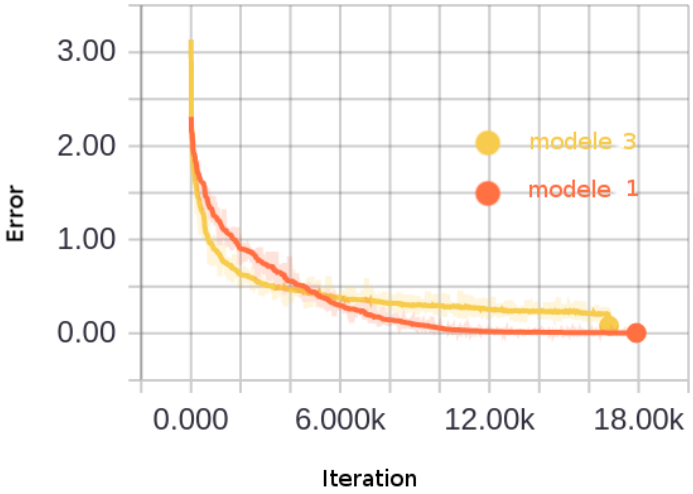

FIGURE 54 – évolution de la courbe d'apprentissage des modèle 1 et 3

	erreur
modele 1	$5e^{-3}$
modele 3	0.19

TABLE 8 – comparaison entre l'erreur des modèle 1 et 3 après 45 époques

Clairement non. Le gap entre les 2 courbes est encore important, mais la fonction d'activation ELU a au moins le mérite d'accélérer l'apprentissage par rapport à ReLu et d'améliorer la capacité du réseau à généraliser. La méthode qui va suivre va définitivement régler ce problème et bien plus encore.

10 Batch normalization

Il est connu depuis longtemps que l'apprentissage du réseau converge plus rapidement si ses entrées sont centrées réduites (moyenne = 0, variance = 1) [103]. Comme chaque couche observe des entrées produites par des couches qui la précède, il serait avantageux d'obtenir des entrées centrées et réduites pour chaque couche.

Batch normalization [104]est une composante qui se trouve entre les couches du réseau de neurones et qui prend continuellement la sortie d'une couche particulière et la normalise avant de l'envoyer à la couche suivante comme entrée.

Soit H un mini batch d'activations de la couche à normaliser, disposé en une matrice avec les activations pour chaque exemple apparaissant dans une ligne de la matrice. Pour normaliser H :

$$H' = \frac{H - \mu}{\sigma} \tag{48}$$

Où μ est un vecteur contenant la moyenne de chaque neurone et σ un vecteur contenant l'écart type de chaque neurone.

L'idée est de normaliser toute la matrice H, donc les éléments H_{ij} seront normalisés en soustrayant μ_j et en divisant par σ_j. Le reste du réseau opère donc sur H' exactement de la même manière que sur H :

En phase d'apprentissage :

$$\mu = \frac{1}{m} \sum_i H_{i,:} \tag{49}$$

et

$$\sigma = \sqrt{\epsilon + \frac{1}{m} \sum_i (H - \mu)_i^2} \tag{50}$$

En phase de test, les entrées sont aussi normalisées mais μ et σ seront remplacés par les moyennes mobiles qui ont été collectées durant la phase d'apprentissage. Cela permet au modèle d'être évalué sur un seul exemple sans avoir besoin à utiliser la définition de μ et σ qui dépendent du mini batch.

La normalisation de la moyenne et de l'écart type d'une unité peut réduire la puissance expressive du réseau de neurones contenant cette unité. Afin de maintenir le pouvoir expressif du réseau, il est fréquent de remplacer le batch d'activation des neurones cachées H par $\gamma H' + \beta$ au lieu de simplement H' normalisé. Les variables γ et β sont des paramètres entraînables qui permettent à la nouvelle variable d'avoir n'importe quelle moyenne et écart type.

Nous allons introduire la Batch normalization dans notre réseau comme suit :

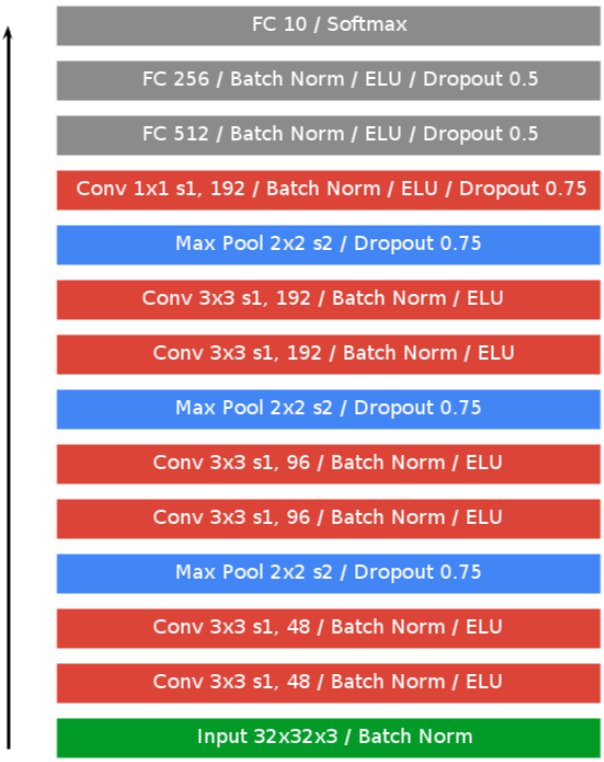

FIGURE 55 – Le modèle 4. Une couche de normalisation entre chaque couche de convolution et son activation. Même chose pour la partie perceptron.

Le nouveau comportement du réseau est montré dans les figures suivantes :

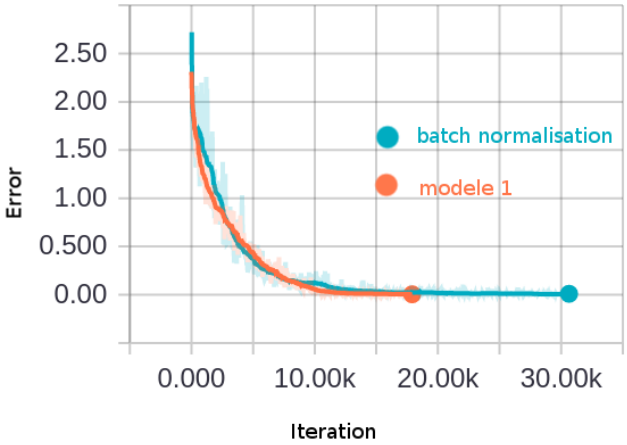

FIGURE 56 – Evolution de la courbe d'apprentissage des modèles 1 (figure 42) et 4 (figure 55)

	erreur	temps
modele 1	$5e^{-3}$	1h15
batch normalization	0.02	1h32

TABLE 9 – Comparaison entre l'erreur des modèle 1 et 4 après 45 époques

Cette fois si l'erreur d'apprentissage est aussi vite et bien minimisée comparée au modèle 1 et la différence est négligeable. Le modèle avec batch normalization requiert un plus grand temps d'exécution. Ceci s'explique par le fait que nous introduisons plus de calculs dans le réseau et aussi par l'introduction de 2 nouveaux paramètres γ et β qui sont entraînables par la retro propagation.

Maintenant comparons la performance en phase de test du modèle 4 (55) avec la performance du modèle qui nous a permis d'atteindre le meilleur résultat jusqu'ici c.à.d le modèle 3 (figure 51) :

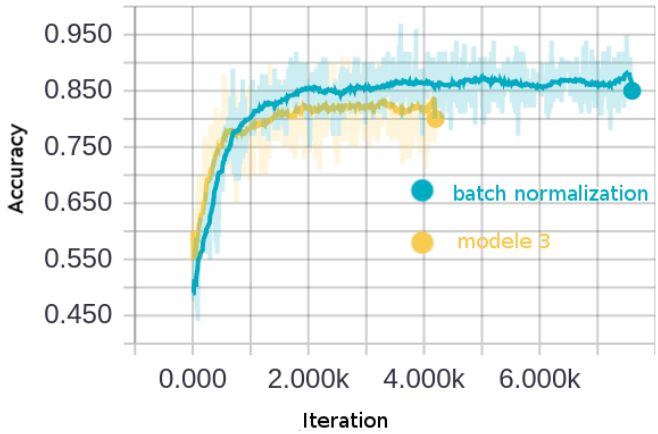

FIGURE 57 – Évolution de la courbe de precision des modèles 3 (figure 51) et 4 (figure 55)

	precision %
modele 3	83
batch normalization	87

TABLE 10 – Comparaison entre la precision des modèles 3 et 4 après 45 époques

Le fait de normaliser les entrées par la moyenne et l'écart type collecté à travers l'apprentissage se montre très efficace et augmente la precision du modèle de 3%.

11 Global average pooling

Un réseau de neurones convolutif traditionnel opère une convolution durant ses premières couches. Pour la classification, la dernière couche convolutive est aplatie puis injectée dans la première couche du perceptron. Cette structure relie la partie convolution à la partie classifieur qui est le perceptron. Les couches convolutives agissent comme des extracteurs de caractéristiques et le résultat est classé de manière traditionnelle.

Cependant le perceptron multi couche est sujet au sur-apprentissage, entravant ainsi la capacité de généralisation du réseau. Dropout fut proposé comme régularisateur.

Global Average Pooling (GAP) [105] fut proposé pour remplacer la partie perceptron multi couches. L'idée c'est de générer une feature map pour chaque catégorie correspondante. Au lieu d'ajouter un perceptron après les feature map, on prend la moyenne de chacune d'elles et le résultat est injecté dans la fonction softmax. Un avantage du GAP par rapport aux couches de perceptron est qu'il n'y a aucun paramètres à entraîner, par conséquent, le sur-apprentissage est évité.

Dans notre réseau, le GAP est introduit comme ceci :

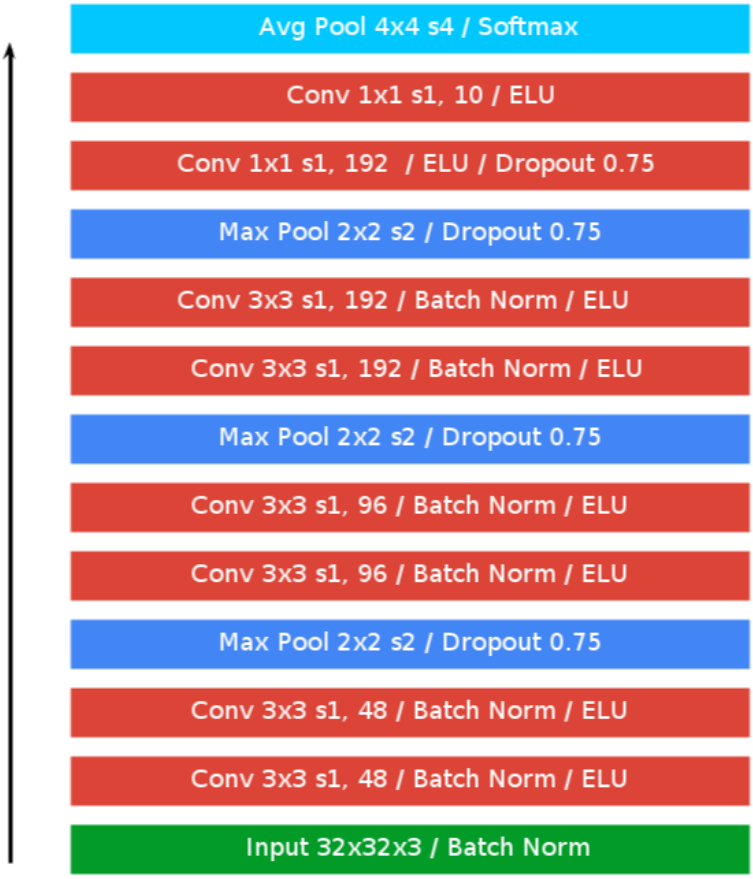

FIGURE 58 – Le modele 5. La partie MLP est remplacée par une couche convolutive avec 10 filtres (un filtre pour chaque classe) puis Avg Pool qui fait la moyenne de chaque feature map.

A fin de voir l'avantage du GAP par rapport au MLP, nous allons reprendre le modèle 4 (figure 55) mais sans régularisation dans la partie MLP, comme ceci :

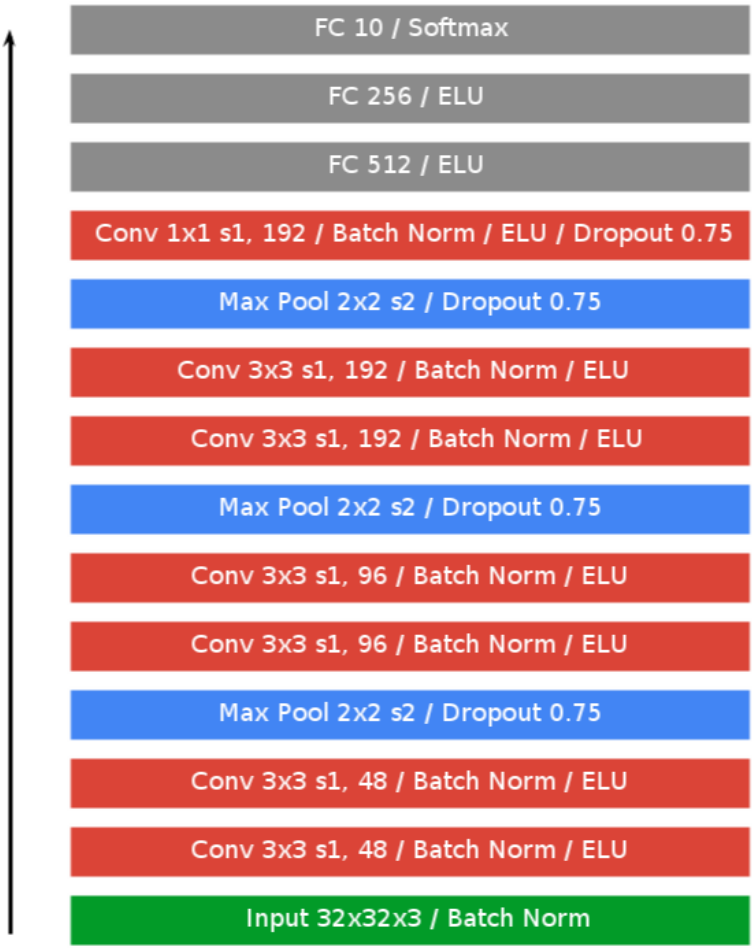

FIGURE 59 – Une version du modèle 4. sans régularisation dans la partie MLP

Et voici le comportement des deux modèles :

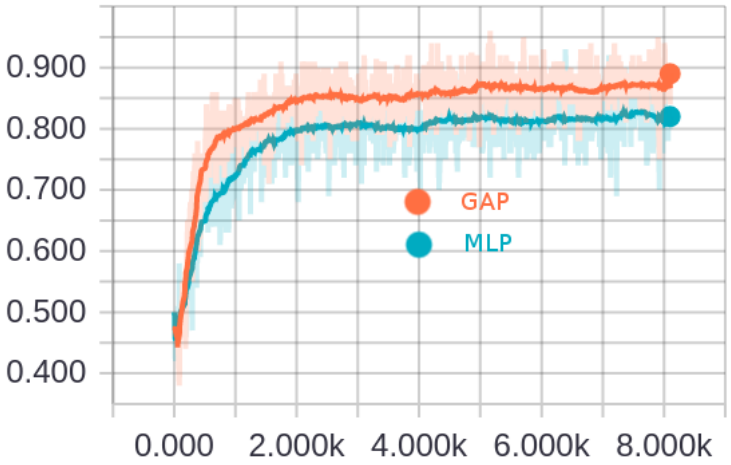

FIGURE 60 – Évolution de la precision sur l'ensemble de test pour le modèle 5 et le modèle 4 avec MLP non régularisé

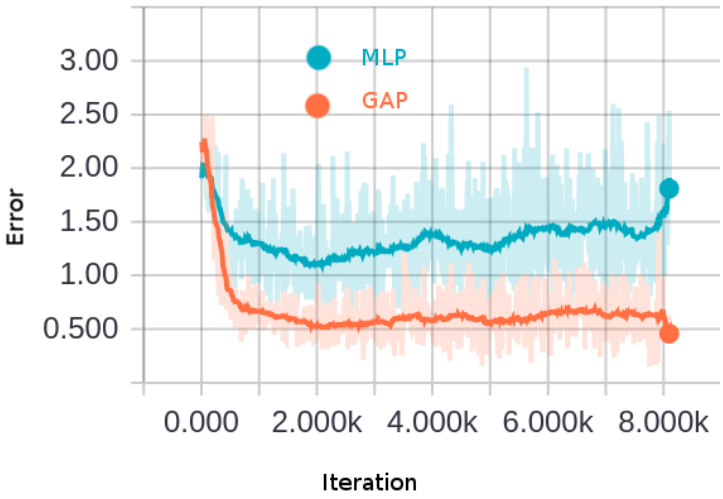

FIGURE 61 – Évolution de l'erreur sur l'ensemble de test pour le modèle 5 et le modèle 4 avec MLP non régularisé

	erreur	precision %
MLP	1.6	82
GAP	0.6	87

TABLE 11 – Comparaison entre l'erreur et la precision des 2 modèles sur l'ensemble de test après 80 époques

Avec ses 71% de paramètres en moins, le réseau de neurones avec GAP surpasse largement le modèle avec MLP non régularisé grâce à la capacité du GAP à mieux généraliser que le MLP par le fait qu'il n'y a aucun paramètre à entraîner. Nous pouvons même distinguer dans la figure 61 un sur-apprentissage dans la courbe du MLP malgré une partie convolutive moins enclins au sur-apprentissage de par sa nature et régularisé malgré tout avec Dropout.

Maintenant nous avons obtenu notre modèle final. Puisque nous comptons tirer le maximum de notre réseau, nous prévoyons de laisser durer l'apprentissage très longtemps et faire une sauvegarde des paramètres à chaque époque jusqu'à ce qu'il n'y ait plus aucune amélioration de performance sur l'ensemble de test, puis restaurer le modèle à son état où il a offert la meilleure performance.

Comme l'apprentissage va durer longtemps, on peut se permettre de baisser les probabilités du dropout afin d'augmenter la capacité de généralisation du réseau.

- methode d'apprentissage : ADAM
- taux d'apprentissage : 10^{-3}
- initialisation des parametres : Glorot initialisation

erreur	0.38
precision %	90.35
temps d'apprentissage	18h50mn

TABLE 12 – Performance finale de notre modèle sur l'ensemble de test. L'apprentissage a duré 561 époques et la meilleure performance a été enregistrée à la 488ème époque

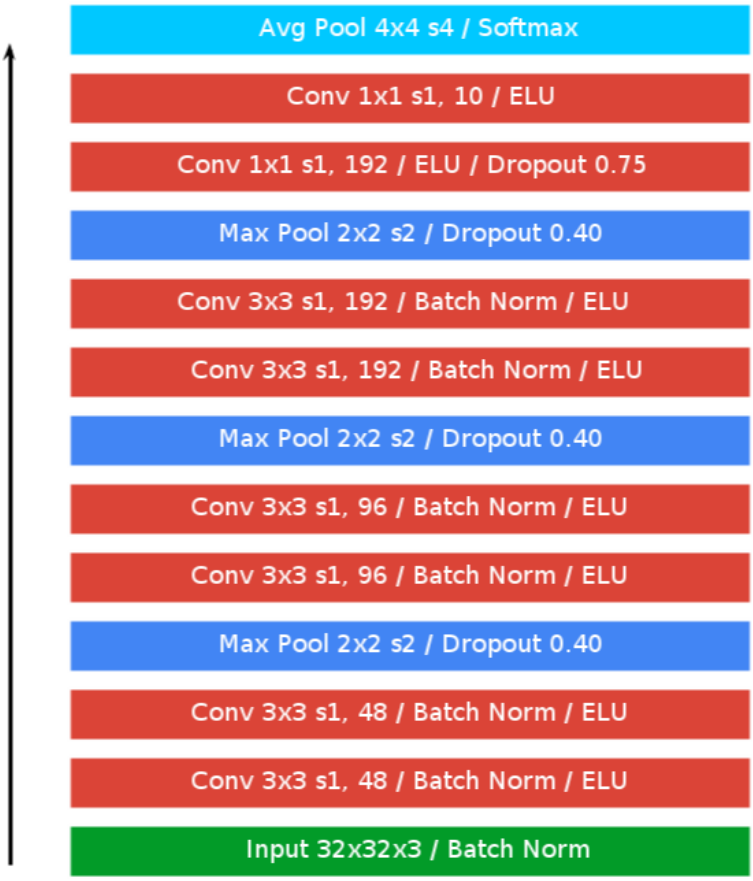

FIGURE 62 – Le modèle final

12 Interface

Nous avons tenu à introduire notre réseau de neurones dans une interface graphique afin de pouvoir visualiser les performances de notre réseau de neurones facilement sans être obligé de passer à chaque fois par les lignes de codes. Les détails de l'interface sont montrés dans la figure suivante :

- **1** : Choisir l'ensemble sur lequel tester le réseau (apprentissage, test, nouveau) avec taux de bonne classification et matrice de confusion en plus d'enregistrer les exemples mal classés.
- **2** : recherche d'un exemple en particulier en entrant le numéro.
- **3** :affichage de l'exemple en question.
- **4 et 5** :Exemple suivant / Exemple précédent. Ces deux boutons déclenchent automatiquement la prédiction du réseau. Tout comme le bouton 2
- **6** : sortie désirée
- **7** : sortie prédite
- **8** : les feature map de l'exemple en question seront enregistrés en tant qu'image et pourront être visualisés.
- **9** : affichage des scores obtenus en fonction de chaque catégorie

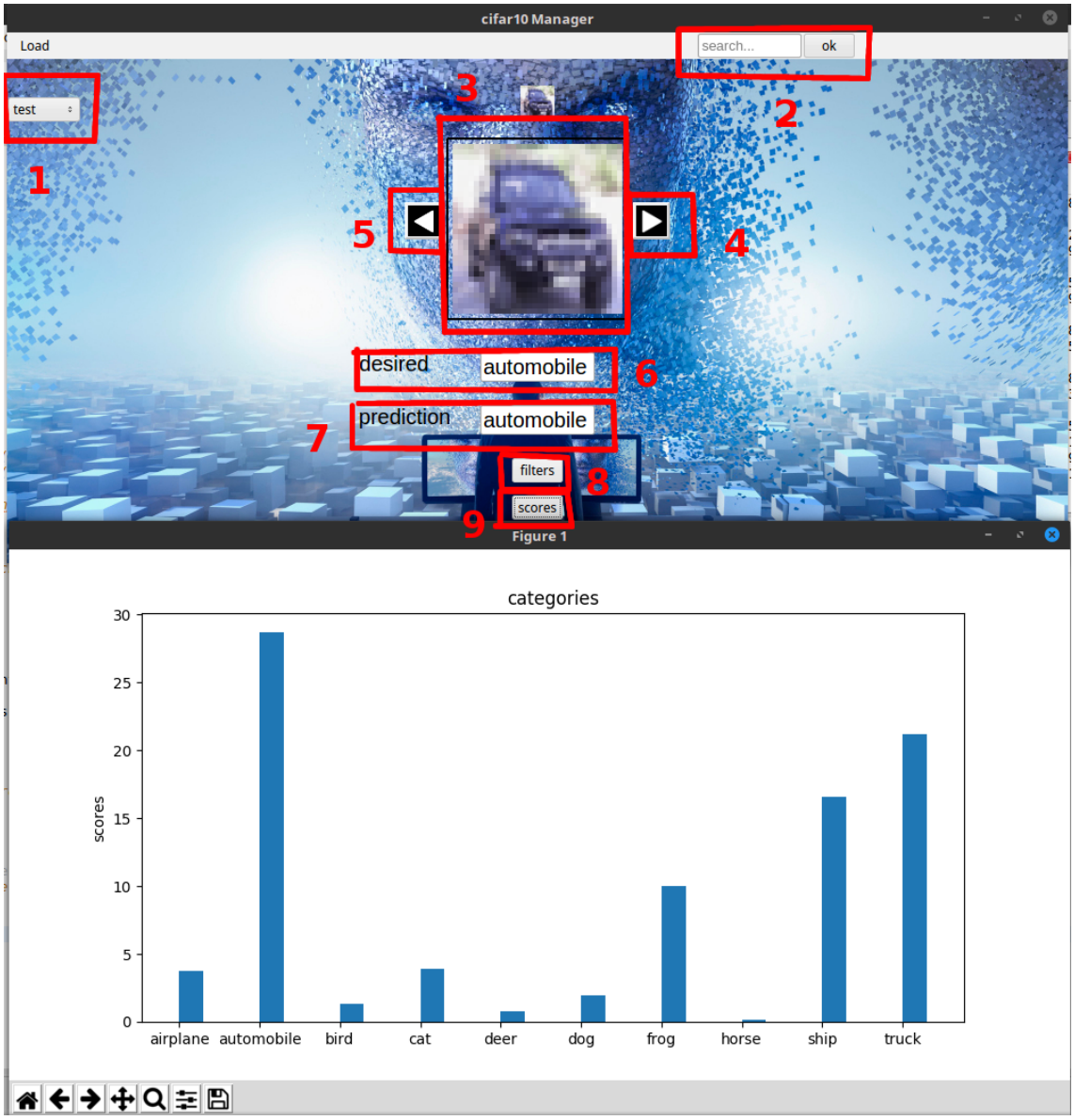

FIGURE 63 – L'interface graphique qui sert à tester le réseau

13 Conclusion

Dans cette partie, nous avons décrit la démarche suivie afin d'obtenir le classifieur le plus précis possible. Nous avons commencé par un réseau de neurones convolutif classique et analysé ses défauts en observant son comportement grâce aux graphes et nous avons fait changer les paramètres un à un afin de comprendre l'impacte de chacun sur la performance finale. Après avoir trouvé l'architecture qui nous convient le mieux, nous avons décidé de tirer le maximum du réseau en laissant l'apprentissage durer longtemps jusqu'à obtenir la performance maximale que peut atteindre notre modèle et qui est l'une des meilleures performances enregistrées sur la base cifar-10

Conclusion et perspectives

Grâce aux Deep Learning, l'avenir de l'intelligence artificielle est prometteur.

Dans ce travail, nous avons exploré le domaine de la classification d'image qui comme tous les autres domaines de l'intelligence artificielle ont connue une évolution majeure depuis l'apparition du Deep Learning.

Afin d'aboutir à ces résultats, nous avons passé beaucoup de temps à lire et à étudier les publications et les articles pour voir ce qui se fait de mieux en matière de classification et pour pouvoir concevoir notre propre modèle.

Pour finir, avant de passer aux perspectives, ce travail nous a permis de mettre en pratique nos connaissances sur les réseaux de neurones et d'en acquérir d'autres et le temps passé à lire des articles nous a servi d'une bonne initiation à la recherche.

Comme perspectives nous pouvons citer :

- Implémentation des techniques qui font le succès des modèles qui participent au challenge Imagenet (ResNet ...).
- Tester sur de nouvelles bases
- Segmentation et génération des images.

Bibliographie

[1] H. Wang, B. Raj, and E. P. Xing, "On the origin of deep learning," *arXiv preprint arXiv :1702.07800*, 2017.

[2] R. Kurzweil, R. Richter, R. Kurzweil, and M. L. Schneider, *The age of intelligent machines*, vol. 579. MIT press Cambridge, 1990.

[3] R. Bellman, *An introduction to artificial intelligence : Can computers think ?* Thomson Course Technology, 1978.

[4] H. Patrick, "Winston. artificial intelligence," 1992.

[5] P. David, M. Alan, and G. Randy, "Computational intelligence : A logical approach," 1998.

[6] G. Bradski, A. Kaehler, and V. Pisarevsky, "Learning-based computer vision with intel's open source computer vision library.," *Intel Technology Journal*, vol. 9, no. 2, 2005.

[7] P. K. Chan and S. J. Stolfo, "Toward scalable learning with non-uniform class and cost distributions : A case study in credit card fraud detection.," in *KDD*, vol. 1998, pp. 164–168, 1998.

[8] X. D. Huang, W. H. Gates, E. J. Horvitz, J. T. Goodman, B. A. Brunell, S. T. Dumais, G. W. Flake, T. J. Griffin, and O. Hurst-Hiller, "Web-based targeted advertising in a brick-and-mortar retail establishment using online customer information," June 29 2006. US Patent App. 11/427,764.

[9] I. Kononenko, "Machine learning for medical diagnosis : history, state of the art and perspective," *Artificial Intelligence in medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[10] S. Russell and P. Norvig, *Artificial Intelligence : A Modern Approach*. Upper Saddle River, NJ, USA : Prentice Hall Press, 3rd ed., 2009.

[11] K. Hechenbichler and K. Schliep, "Weighted k-nearest-neighbor techniques and ordinal classification," 2004.

[12] T. Hong and D. Tsamis, "Use of knn for the netflix prize," *CS229 Projects*, 2006.

[13] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine learning*, vol. 29, no. 2, pp. 103–130, 1997.

[14] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, pp. 41–46, IBM New York, 2001.

[15] Shannon and Weaver 1949.

[16] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[17] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, ACM, 1992.

[18] V. N. Vapnik and S. Kotz, *Estimation of dependences based on empirical data*, vol. 40. Springer-Verlag New York, 1982.

[19] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations," *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, vol. 209, pp. 415–446, 1909.

[20] M. Minsky and S. Papert, "Perceptrons.," 1969.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., DTIC Document, 1985.

[22] L. Deng, D. Yu, *et al.*, "Deep learning : methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[23] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[24] Y. Bengio, A. Courville, and P. Vincent, "Representation learning : A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[25] J. Schmidhuber, "Deep learning in neural networks : An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[26] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[27] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep machine learning-a new frontier in artificial intelligence research [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 5, no. 4, pp. 13–18, 2010.

[28] J. Schmidhuber, "Deep learning.," *Scholarpedia*, vol. 10, no. 11, p. 32832, 2015.

[29] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes, *et al.*, "The human splicing code reveals new insights into the genetic determinants of disease," *Science*, vol. 347, no. 6218, p. 1254806, 2015.

[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[31] F. Buggenthin, F. Buettner, P. S. Hoppe, M. Endele, M. Kroiss, M. Strasser, M. Schwarzfischer, D. Loeffler, K. D. Kokkaliaris, O. Hilsenbeck, *et al.*, "Prospective identification of hematopoietic lineage choice by deep learning," *Nature Methods*, vol. 14, no. 4, pp. 403–406, 2017.

[32] E. Gibney, "Google reveals secret test of ai bot to beat top go players.," *Nature*, vol. 541, no. 7636, p. 142, 2017.

[33] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv :1604.07316*, 2016.

[34] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[35] H. C. Hazlett, "Early brain development in infants at high risk for autism spectrum disorder," in *Biological Psychiatry*, vol. 73, pp. 115S–115S, ELSEVIER SCIENCE INC 360 PARK AVE SOUTH, NEW YORK, NY 10010-1710 USA, 2013.

[36] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv :1508.06576*, 2015.

[37] R. Dechter and J. Pearl, *The cycle-cutset method for improving search performance in AI applications*. University of California, Computer Science Department, 1986.

[38] I. Aizenberg, N. N. Aizenberg, and J. P. Vandewalle, *Multi-Valued and Universal Binary Neurons : Theory, Learning and Applications*. Springer Science & Business Media, 2013.

[39] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in neural information processing systems*, pp. 2643–2651, 2013.

[40] R. Collobert and J. Weston, "A unified architecture for natural language processing : Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, ACM, 2008.

[41] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda, "Subject independent facial expression recognition with robust face detection using a convolutional neural network," *Neural Networks*, vol. 16, no. 5, pp. 555–559, 2003.

[42] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[45] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[46] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[47] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[48] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv :1602.07261*, 2016.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[50] M. C. Mozer, "A focused backpropagation algorithm for temporal," *Backpropagation : Theory, architectures, and applications*, p. 137, 1995.

[51] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural networks*, vol. 1, no. 4, pp. 339–356, 1988.

[52] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks.," *ICML (3)*, vol. 28, pp. 1310–1318, 2013.

[53] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[54] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv :1406.1078*, 2014.

[55] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model.," in *Interspeech*, vol. 2, p. 3, 2010.

[56] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 5528–5531, IEEE, 2011.

[57] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 2011.

[58] S. Liu, N. Yang, M. Li, and M. Zhou, "A recursive recurrent neural network for statistical machine translation," 2014.

[59] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[60] M. Auli, M. Galley, C. Quirk, and G. Zweig, "Joint language and translation modeling with recurrent neural networks.," in *EMNLP*, vol. 3, p. 0, 2013.

[61] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks.," in *ICML*, vol. 14, pp. 1764–1772, 2014.

[62] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3128–3137, 2015.

[63] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell : A neural image caption generator," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3156–3164, 2015.

[64] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634, 2015.

[65] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers : A comparison of logistic regression and naive bayes," *Advances in neural information processing systems*, vol. 2, pp. 841–848, 2002.

[66] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[67] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*, pp. 791–798, ACM, 2007.

[68] G. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, vol. 9, no. 1, p. 926, 2010.

[69] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[70] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," in *Artificial Intelligence and Statistics*, pp. 448–455, 2009.

[71] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," in *Advances in neural information processing systems*, pp. 2222–2230, 2012.

[72] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[73] E. L. Denton, S. Chintala, R. Fergus, *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in neural information processing systems*, pp. 1486–1494, 2015.

[74] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv :1511.06434*, 2015.

[75] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski, "Deep generative stochastic networks trainable by backprop," in *International Conference on Machine Learning*, pp. 226–234, 2014.

[76] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv :1511.05644*, 2015.

[77] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in neural information processing systems*, pp. 2933–2941, 2014.

[78] R. S. Sutton, "Two problems with backpropagation and other steepest-descent learning procedures for networks," in *Proc. 8th annual conf. cognitive science society*, pp. 823–831, Erlbaum, 1986.

[79] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[80] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, 2013.

[81] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.

[82] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8624–8628, IEEE, 2013.

[83] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[84] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, pp. 1223–1231, 2012.

[85] L. Clark, "Google's artificial brain learns to find cat videos," *Wired UK, www. wired. com*, 2012.

[86] J. Pennington, R. Socher, and C. D. Manning, "Glove : Global vectors for word representation.," in *EMNLP*, vol. 14, pp. 1532–1543, 2014.

[87] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini–batch gradient descent," 2012.

[88] D. Kingma and J. Ba, "Adam : A method for stochastic optimization," *arXiv preprint arXiv :1412.6980*, 2014.

[89] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[90] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[91] A. Blum and R. L. Rivest, "Training a 3-node neural network is np-complete," in *Proceedings of the 1st International Conference on Neural Information Processing Systems*, pp. 494–501, MIT Press, 1988.

[92] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[93] H. Wu, "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions," *Information Sciences*, vol. 179, no. 19, pp. 3432–3441, 2009.

[94] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[95] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks.," in *Aistats*, vol. 15, p. 275, 2011.

[96] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, 2013.

[97] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers : Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[98] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[99] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Aistats*, vol. 9, pp. 249–256, 2010.

[100] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers : Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[101] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[102] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv :1511.07289*, 2015.

[103] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks : Tricks of the trade*, pp. 9–48, Springer, 2012.

[104] S. Ioffe and C. Szegedy, "Batch normalization : Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv :1502.03167*, 2015.

[105] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv :1312.4400*, 2013.

التعلم المتعمق

التعلم العميق هو عبارة عن مجموعة من أساليب التعلم الألي محاولا لنمذجة مع مستوى عال من تجريد البيانات من خلال أبنية مفصلية من مختلف التحولات غير الخطية. اثبتت الاكتشافات في هذا المجال تقدما كبيرا وسريعا وفعالية في العديدمن المجالات منها التعرف على الوجه، التعرف على الكلام، الرؤية الحاسوبية، ومعالجة اللغات الطبيعية

في هذا العمل، تمكنا من الحصول على نتائج جيدة، كما قدمنا عدة تحسينات من خلال تقديم العديد من التغيرات لفهم تأثير كل واحد على النتيجة النهائية.

ذكاء اصطناعي, تعلم آلي ,التعلم المتعمق , التصنيف

L'**apprentissage profond**

L'**apprentissage profond** est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires. Ces techniques ont permis des progrès importants et rapides dans les domaines de l'analyse du signal sonore ou visuel et notamment de la reconnaissance faciale, de la reconnaissance vocale, de la vision par ordinateur, du traitement automatisé du langage.

Dans ce travail, nous avons réussi à reproduire les résultats de l'état de l'art, aussi nous avons réalisé plusieurs optimisations en proposant divers paramétrage de l'architecture du réseau convolutif afin de comprendre l'impact de chacun d'eux sur le résultat final.

l'intelligence artificielle, l'apprentissage automatique, l'apprentissage profond, la classification

Deep learning

Deep learning is a set of automatic learning methods attempting to model with a high level of abstraction of data through the articulated architectures of different nonlinear transformations. These techniques have enabled significant and rapid progress in the fields of sound and visual signal analysis, including facial recognition, speech recognition, computer vision, and automated language processing.

In this work, we succeeded in reproducing the results of the state of the art, so we made several optimizations by proposing various parameters of the architecture of the convolutional network in order to understand the impact of each of them on the final result.

Artificial intelligence, machine learning, deep learning, classification