REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
--T L E M C E N--

الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
École Supérieure en Sciences Appliquées

المدرسة العليا في العلوم التطبيقية
-تلمسان-

Mémoire de fin d'étude

Pour l'obtention du diplôme Master

Filière : Automatique
Spécialité : Automatique

Présenté par : BARAKA Hodhaifa Abdelghani

Thème

Étude sur les Méthodes d'Optimisation Utilisée dans l'Apprentissage Automatique

Soutenu publiquement, le 27 / 10 / 2020, devant le jury composé de :

M. ABDELLAOUI Ghouti	MCB	ESSA.Tlemcen	Président
M. RIMOUCHE Ali	MCB	ESSA.Tlemcen	Directeur de mémoire
Mme. HANDOUZI Wahida	MCB	Université de Tlemcen	Co- Directrice de mémoire
Mme. GHOMRI Latéfa	MCA	Université de Tlemcen	Examinateur 1
Mme. SEBBAGH Hafidha	MCB	ESSA.Tlemcen	Examinateur 2

Année universitaire : 2019/2020

Je dédie ce modeste travail :

A mes chers parents

….

Hodhaifa

REMERCIEMENT

Avant toute personne, je remercie le bon Dieu de nous avoir prêté vie, santé et volonté pour achever ce travail.

Je tiens à remercier mes encadreurs M. RIMOUCHE Ali et Mme. HANDOUZI Wahida pour tout le temps qu'ils m'ont consacré, pour leurs conseils précieux, pour toute leur aide et leur appui durant la réalisation de ce travail.

Je tiens à remercier, mes chers parents pour leur encouragement et soutien.

Je tiens à remercier chacun des membres du jury pour nous avoir fait l'honneur d'examiner et d'évaluer notre travail.

Résumé : L'objectif étant de définir le meilleur optimiseur en termes de plus haute résolution et de moindre perte, on explique au passage ce qu'un neurone artificiel et ce qu'un réseau de neurones convolutif ; on ira ensuite comparer plusieurs optimiseurs tels qu'AdaGrad, SGD, RMSprop et Adam en utilisant une base de données de chiffres manuscrits à l'aide d'un code Python basé sur l'architecture d'un réseau de neurones convolutif afin de conclure le meilleur algorithme d'optimisation.

Mots-clés : CNN, Optimiseurs, Python.

الملخص: الهدف هو تحديد أفضل مُحسِّن من حيث دقة أعلى وخسارة أقل، نشرح فيما يلي ماهية الخلايا العصبية الاصطناعية وما هية الشبكة العصبية التلافيفية. ننتقل بعد ذلك إلى مقارنة العديد من المحسّنات مثل AdaGrad و SGD و RMSprop و Adam باستخدام قاعدة بيانات للأرقام المكتوبة بخط اليد باستخدام كود Python استنادًا إلى بنية شبكة عصبية تلافيفية من أجل اختيار أفضل خوارزمية تحسين.

الكلمات المفتاحية: CNN، Optimizers، Python.

Abstract : The objective being to define the best optimizer in terms of higher resolution and less loss, we explain in passing what an artificial neuron and what a convolutional neural network is. We will then go on to compare several optimizers such as AdaGrad, SGD, RMSprop and Adam using a database of handwritten figures using a Python code based on the architecture of a convolutional neural network in order to conclude the best optimization algorithm.

Keywords : CNN, Optimizers, Python.

Table des matières

Liste des abréviations

Adam	Moment d'Estimation Adaptatif
CNN	Réseaux de Neurones Convolutifs
IA	Intelligence Artificielle
MNIST	Base de Données de Gravure Manuscrite
ML	Apprentissage Automatique
RMSprop	Propagation quadratique moyenne
SGD	Méthode du Gradient Stochastique
SVM	Séparateur à Vaste Marge

Liste des tableaux

Liste des figures

Introduction

L'apprentissage automatique (ML) a émergé dans la seconde moitié du XXème siècle du domaine de l'intelligence artificielle et correspond à l'élaboration d'algorithme capable d'accumuler de la connaissance et de l'intelligence à partir d'expériences, sans être humainement guidés au cours de leur apprentissage, ni explicitement programmés pour gérer telle ou telle expérience ou donnée spécifique.

Pourtant, les fondements de ces méthodes ne sont pas si récents : l'apprentissage profond a été formalisé en 2007 à partir de nouvelles architectures de réseaux de neurones dont les précurseurs sont McCulloch et Pitts en 1943 qui suivront de nombreux développements comme, les réseaux de neurones convolutifs de Yann Le Cun et Yoshua Bengio en 1998 et les réseaux de neurones profonds qui en découlent en 2012 et ouvrent la voie à de nombreux champs d'application comme la vision, le traitement du langage ou la reconnaissance de la parole.

Dans ce mémoire, nous allons utiliser l'apprentissage automatique (ML) dans le coté optimisation. Tout d'abord nous allons évoquer des états de l'art sur ces méthodes et nous allons voire plusieurs types de ces derniers afin de faire une application sur la classification des chiffres avec un code python en utilisant l'architecture de réseaux de neurones convolutifs. A la fin on va faire une conclusion sur la meilleure méthode.

Dans le but de bien mené cette étude nous avons besoin d'une base de données MNIST pour la classification des chiffres et nous allons explorer l'optimisation en testant plusieurs optimiseurs afin de choisir celui avec la meilleure résolution qui conviendra à notre programme.

Dans le premier chapitre nous allons définir ce que l'apprentissage automatique ainsi que ses différents problèmes et nous allons présenter un état de l'art sur les méthodes d'optimisations dans l'apprentissage automatique.

Dans le deuxième chapitre nous allons expliquer notre méthodologie de travail en explorant la base de données MNIST puis en définissant ce qu'un séparateur a vaste marge (SVM) et ce qu'un réseau de neurones convolutif, on donnera vers la fin de ce chapitre des exemples sur la classification des chiffres.

Dans le troisième chapitre de cet ouvrage vous trouverai l'application, le chapitre contiendra différents codes python de différents optimiseurs, les résultats de ces applications nous permettrons de choisir l'optimiseur le plus convenable à notre travail.

Chapitre 1

Les méthodes d’optimisations dans l’apprentissage automatique

1.1 Introduction :

Ces dernières années, l'intelligence artificielle (IA) a fait l'objet d'une large couverture médiatique. L'apprentissage automatique, l'apprentissage profond et l'intelligence artificielle apparaissent souvent dans d'innombrables articles en plus des publications techniques. Notre avenir promis Chatbots, voitures et assistants virtuels - parfois représentés dans le futur à l'ère de la lumière sinistre et de l'utopie où le travail humain deviendra rare et les activités économiques seront gérées par des robots ou des agents IA. Les praticiens de l'apprentissage automatique y voient la promesse d’une nouvelle révolution industrielle qui bouleversera notre rapport au travail et à la connaissance.

Nous allons présenter dans ce chapitre l’apprentissage automatique en anglais « Machine Learning » et ces différents principes de fonctionnement tout en focalisons notre recherche sur les méthodes d’optimisation utilisées.

1.2 Apprentissage automatique (ML) :

1.2.1 Définition

En terme court l'apprentissage automatique « Machine Learning » est une technique de modélisation qui fait appel à des données. Si on veut plus de détails, cet apprentissage consiste à découvrir un ensemble de données et apprendre comment ce dernier fonctionne et donne à la

fin un modèle général qui peut résoudre un problème similaire à la base de données, ce modèle est conclu après un entrainement avec les données d'entrée.

L'apprentissage automatique traite des problèmes que l'on ne peut pas modéliser avec des équations mathématiques ou des lois physiques. Donc la machine a seulement une base de données pour l'entrainement avec et nous donner un modèle qui peut être appliqué sur les données réelles du terrain. Ce processus est présenté à la figure suivante :

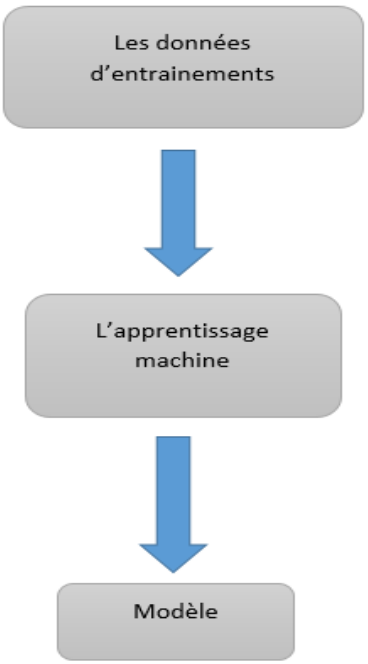


Figure 1 : Processus de l'apprentissage automatique

1.2.2 Les problèmes de l'apprentissage automatique :

a. Overfitting (Sur-apprentissage) :

Le sur-apprentissage c'est le fait que le model conclue par notre machine s'adapte très bien avec les données d'entrainements et même plus. Ce qui cause des problèmes car notre modèle va capturer les détails des données et toutes les caractéristiques, donc il va même capturer des bruits

et toutes les autres variations aléatoires de données, on aboutira donc sur un modèle mal généralisé.

b. Underfitting (Sous-apprentissage) :

Le sous-apprentissage est le contraire du problème précédant, donc notre machine n'arrive même pas à faire des prédictions justes sur les données, ce qui nous donne un modèle pas bien généralisé et qui fait plusieurs erreurs.

1.2.3 Réglage des problèmes de l'apprentissage automatique :

En conclusion notre modèle doit éviter un sur ou un sous-apprentissage, donc on doit être au milieu de ses deux derniers.

Tout d'abord on a deux méthodes à utiliser pour lutter contre les problèmes de l'apprentissage automatique : la régulation et la validation.

La régulation est une méthode numérique qui essaye de former un modèle prédictif, mais cette méthode ne suffira pas pour construire un modèle, ce qui nécessite une validation qui utilise une partie des données pour contrôler les performances.

Ensuite on doit diviser l'entrainement de la machine en deux parties essentielles : le training set (partie d'entrainement) et la validation set (partie de validation) comme indique la figure suivante :

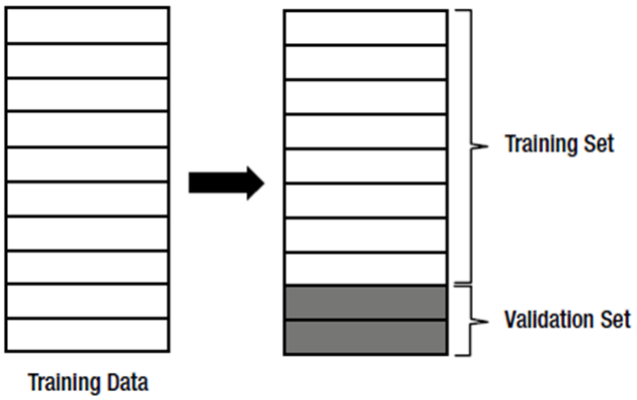


Figure 2 : Les étapes de l'apprentissage automatique

Le processus de l'apprentissage va suivre les étapes suivantes :

1. Deviser les données d'entrainement en deux parties, une pour le training set qui va prendre 80% et l'autre pour la validation set qui va prendre 20% des données.
2. Entraîner le modèle avec l'ensemble d'entraînement.
3. Evaluer la performance du modèle avec la partie de validation (validation set).
4. Si le modèle nous donne des résultats satisfaisants on termine le processus. Sinon on doit répéter les étapes précédentes en modifiant le modèle.

On a encore une autre solution pour les problèmes de l'apprentissage automatique c'est la **validation croisée** qui varie légèrement le processus de validation en divisant encore les données d'entrainement en deux groupes, un pour le training et l'autre pour la validation, mais au lieu de conserver les ensembles des données initialement divisés, elle les change constamment et elle répète cette division plusieurs fois. La validation croisée a pour but de maintenir le caractère aléatoire des données de validation, et détecte mieux l'overfitting. La figure 3 résume le concept de la validation croisée.

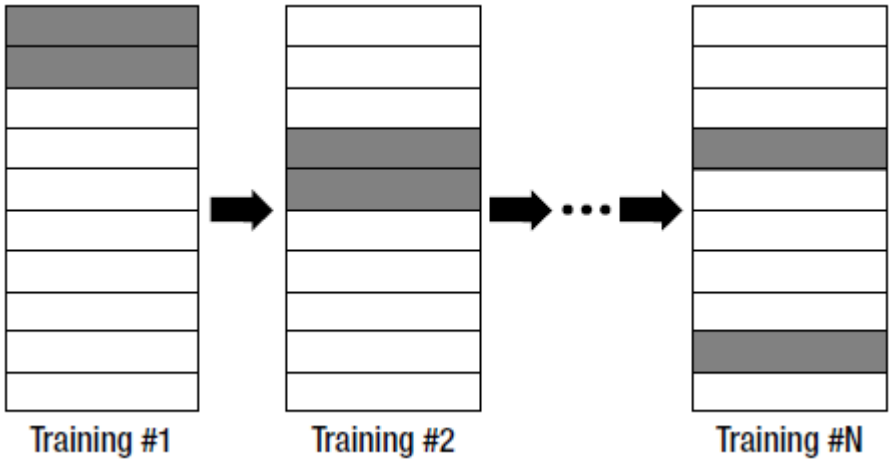


Figure 3 : Validation croisée

1.2.4 Les types d'apprentissage automatique :

Il y'a différents types d'apprentissage pour l'apprentissage automatique qui ont été développés pour résoudre différents problèmes, mais on a trois grandes méthodes connues :

- Supervised learning (Apprentissage Supervisé).
- Unsupervised learning (Apprentissage Non Supervisé).
- Reinforcement learning (Apprentissage Par Renforcement).

a. Supervised Learning (Apprentissage Supervisé) :

Dans l'apprentissage supervisé on a les données d'entrainement comme entrées avec leurs sorties connues, comme pour résoudre un exercice, on compare le résultat avec la solution correcte et si la réponse est fausse, on modifie les connaissances actuelles et on refait les autres exercices. Donc les problèmes et les solutions de l'exercice correspondent aux données de l'entrainement, et les connaissances correspondent au modèle.

Le but de cette méthode est de trouver un modèle prédictif avec la plus grande précision possible, et réduire la différence entre la sortie correcte et la sortie du modèle pour la même entrée, donc si un modèle est parfaitement réalisé, il nous donne un résultat correct qui correspond à l'entrée des données d'entrainement.

La méthode d'apprentissage supervisé traite deux types de problèmes :

- La régression : les données sont dans un ensemble continu
- La classification : les données sont dans un ensemble discret

Parmi les méthodes les plus utilise dans l'apprentissage supervisé est :

- K-plus proches voisins (K-NN)
- L'arbre de décision
- Machine à vecteur de support (SVM)
- Les réseaux de neurones

i. K-plus proche voisin (K-nearest neighbor (K-NN) en anglais) :

C'est l'une des méthodes les plus simples de classification. Le principe de cette méthode est de classer un individu par le vote de ses voisins (k) majoritaires, donc si ces voisins appartiennent majoritairement à un groupe il appartient à ce même groupe.

Ces voisins sont reconnus par une mesure de distance avec l'utilisation de la distance euclidienne (généralement).

ii. Arbre de décision :

Parmi les méthodes de classification les plus utilisé dont le principe est de diviser l'ensemble de données en plusieurs sous-groupes, en fonction d'une variable discriminante ; Comme le montre la figure suivante :

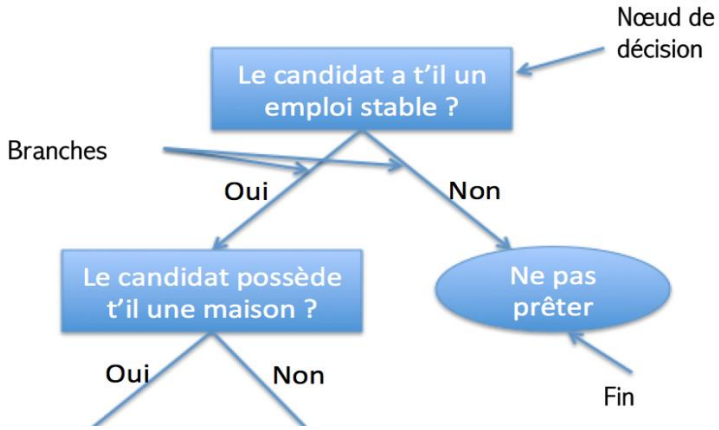


Figure 4 : Arbre de décision

La racine dans l'arbre correspond aux données d'apprentissage et la feuille correspond à la classe.

Pour construire un arbre de décision il faut d'abord choisir la meilleure variable qui sera la racine de l'arbre, et ce choix doit se faire avec des algorithmes basés sur la théorie de l'information et aussi la théorie de Shannon -une théorie probabiliste permettant de quantifier le contenu moyen en information d'un ensemble de messages, dont le codage informatique satisfait une distribution statistique précise-, ces algorithmes feront une évaluation de la qualité de chaque sommet. Ensuite pour crée les sous-ensembles de l'arbre il faudra faire une division récursive pour toutes les données d'apprentissage avec des tests définit à l'aide des variables afin de trouver des différentes classes.

A. Le degré de mélange :

Le degré de mélange est un calcul pour comparer entre les choix des classes. Pour ce calcul on peut utiliser différentes fonctions par exemple :

- Fonction d'entropie :

$$Entropie(p) = -\sum_{k=1}^{c} p\,(k|p) lnp(k|p)$$

Avec :

K : la classe

P : le nœud

- Fonction de Gini :

$$Gini(p) = 1 - \sum_{k=1}^{c} p^2\,(k|p) = 2\sum p(k|p)p(k'|p)$$

Le but de ces fonctions est d'atteindre un minimum ou un maximum, le minimum représente le fait d'avoir un nœud pur c'est-à-dire un nœud à une seule classe qui regroupe tous ces individus, et le maximum représente le fait d'avoir un mélange maximal où les individus sont équirépartis entre les classes.

B. Notion de gain :

C'est une fonction qui cherche le test qui maximise le gain :

$$Gain(p,t) = i(p) - \sum_{j=1}^{n} p_j i(p_j)$$

Avec :

t : le test de la variable

n : nombre de modalités de t

i : la fonction qui mesure le degré de mélange

p_j : la proportion des individus (fixée)

iii. Séparateurs à vaste marge :

La méthode de SVM est appliquée pour les problèmes linéairement séparables (généralement).

Le principe de cette méthode est de changer la dimension de l'espace du problème en une dimension supérieur, le but est de maximiser la marge entre les classes, on cherche un hyperplan séparateur linéaire s'il existe parmi une infinité des hyperplans séparateurs.

On commence par faire une transformation non-linéaire ∅ appliquer sur le vecteur d'entré x et on introduit la fonction suivante pour trouver un hyperplan séparateur optimal :

$$h(x)=\alpha\emptyset(x)+\beta$$

La méthode SVM est souvent très utilise pour de nombreux problème de classification telle que la classification des données, la classification des documents, la reconnaissance faciale, la reconnaissance d'image et la reconnaissance de la parole.

iv. Les réseaux de neurones :

a) Définition :

Un réseau de neurones est un modèle d'apprentissage pour l'apprentissage automatique comme indique la figure ci-dessous:

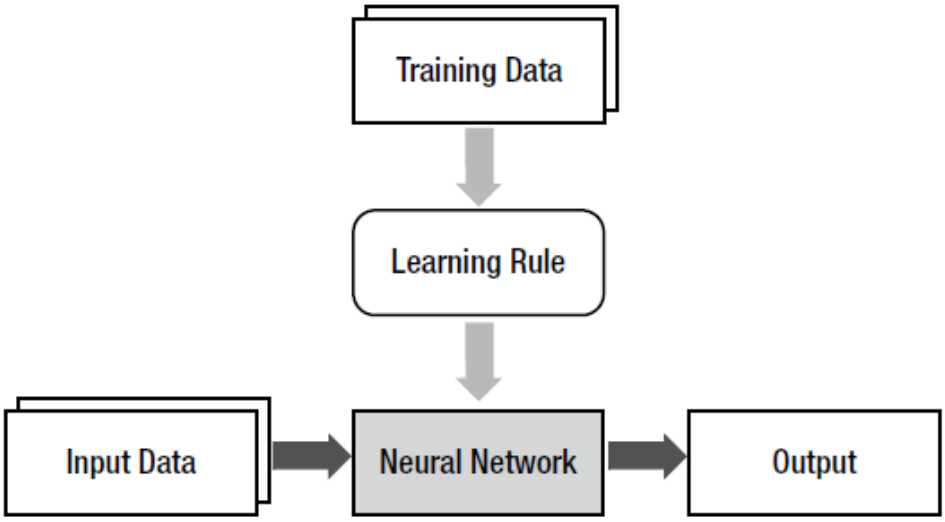


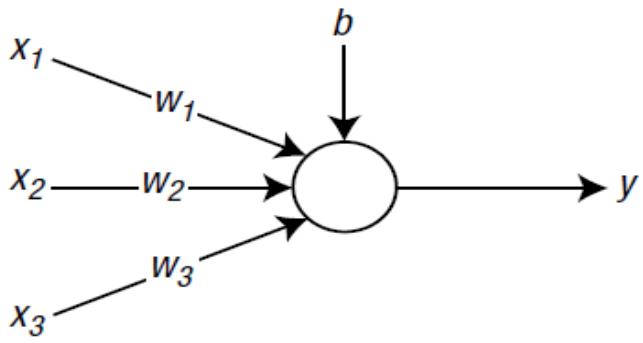
Figure 5 : Schéma explicatif d'un réseau de neurone

Le réseau de neurones est inspiré du cerveau humain, il a plusieurs nœuds connectés entre eux qui transmettent des informations.

b) Les composantes d'un réseau de neurone :

- Un neurone :

Le neurone artificiel dont il est question dans notre étude est une conception des neurologues Warren McCulloch et Walter Pitts où ils montrèrent que des réseaux de neurones formels simples peuvent théoriquement réaliser des fonctions logiques, arithmétiques et symboliques complexes comme un automate doté d'une fonction de transfert qui transforme ses entrées en sortie selon des règles précises. Pour faire plus simple ; un neurone est une composition d'entrées, de sorties, de poids et d'un biais comme indique la figure suivante :



X : entre ; W : poids ; b : biais ; y : sortie

Figure 6: Exemple d'un neurone artificiel.

Si on multiplie les entrées par les poids et qu'on y additionne le biais, on a notre somme pondérée comme indique l'équation suivante :

$$v = (w_1 \times x_1) + (w_2 \times x_2) + (w_3 \times x_3) + b$$

On peut mettre cette fonction sous forme matricielle :

$$v = wx + b$$

Avec : $w = [w_1 \quad w_2 \quad w_3] \quad x = \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix}$

- Fonction d'activation :

C'est une fonction qui sert à introduire une non-linéarité dans le fonctionnement du neurone.

Elle a trois intervalles :

- En dessous du seuil ou le réseau n'est pas active.
- Aux alentours du seuil (une transition).
- Au-dessus du seuil ou le réseau est active.

Exemple :

- Fonction sigmoïde.
- Fonction tangente hyperbolique.
- Les couches :

Le réseau de neurone a plusieurs couches qui contiennent plusieurs nœuds connectés entre eux avec des arcs.

Il y'a trois types des couches :

- Les couches d'entrée.
- Les couches cachées (au milieu).
- Les couches de sorties.

La figure suivante présente les couches d'un réseau de neurones :

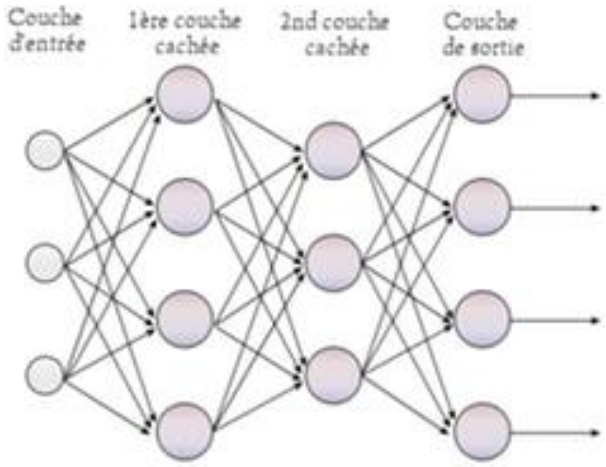


Figure 7: Les différentes couches d'un réseau de neurones

c) Utilité et principe de fonctionnement :

Le réseau de neurones est système capable d'apprendre par l'expérience, en utilisant l'apprentissage supervisé seulement par les étapes suivantes :

- ✓ Initialiser les poids avec des valeurs adéquates.
- ✓ Prendre les données d'entrées comme des données d'entrainement et calculer l'erreur à partir de la sortie.
- ✓ Ajuster les poids pour réduire l'erreur.
- ✓ Répéter les deux étapes précédentes avec toutes les données d'entrainement

d) L'entrainement d'un seul neurone (règle de delta) :

Pour entrainer notre réseau de neurones on doit calculer l'erreur de sortie et réinitialiser les poids des arcs comme avec les formules suivantes :

L'erreur : $e_{i=}d_i - y_i$

La réinitialisation : $w_{ij} + \alpha e_i x_i$

Avec :

e_i : L'erreur

d_i : La sortie désirée

y_i : La sortie réelle

w_{ij} : Le poids de l'arc

α : Taux d'apprentissage, il est toujours entre 0 et 1

x_i : L'entrée

Pour l'entrainement on doit suivre les étapes suivantes :

a) Initialiser les poids à des valeurs adéquates.
b) Calculer l'erreur.
c) Calculer la mise à jour des poids.
d) Ajuster les poids.
e) Effectuer les étapes b à d pour toutes les données relatives à l'entrainement.
f) Effectuer les étapes b à e jusqu'à ce que l'erreur atteigne un niveau acceptable (niveau de tolérance).

v. Le réseau de neurones convolutif (CNN) :

Le CNN et un réseau de neurones profond qui contient plusieurs couches cachées et qui est utiliser pour la reconnaissance des images et les problèmes de classification des images. Ce réseau est un type de modèle d'apprentissage profond presque universellement utilisé dans les applications de vision par ordinateur.

Le principe consiste à extraire les caractéristiques des images et de les classifier. Ce réseau a deux types de couche :

- La couche de convolution qui convertit l'image en utilisant l'opération de convolution avec une collection de filtre numérique.

- La couche de mise en commun qui réduit la dimension de l'image en regroupant les pixels voisins en un seul pixel.

Pour le coté optimisation dans ce réseau on a plusieurs optimiseurs a utilisé tel que le SGD, Adam et RMSprop…etc. Tous ces optimiseurs sont base sur le SGD, le but de cette méthode est de trouver le minimum d'une fonction différenciable (un point où la dérivée est nul), pour le réseau de neurones convolutif, cela signifie qu'il faut trouver analytiquement la combinaison des valeurs de poids qui donnent la plus petite fonction de pertes possible.

La fonction de perte et aussi nécessaire pour le réseau de neurones convolutif, le but de cette fonction est de trouver la quantité qu'un modèle doit chercher à minimiser durant le processus d'entrainement.

b. Unsupervised learning (Apprentissage Non Supervisé) :

Contrairement à l'apprentissage supervisé, l'apprentissage non supervisé a ces données d'entrainement comme des entrées sans sorties connues. Ce type d'apprentissage est utilisé pour certain type de problèmes comme étudier les caractéristiques et le prétraitement des données.

c. Reinforcement learning (Apprentissage Par Renforcement) :

L'apprentissage par renforcement est utilisé pour les observations. Il nous décrit un comportement après l'observation de son environnement. Ce type d'apprentissage utilise comme données d'entrainement des entrées, quelques sorties et des notes pour ces derniers.

1.3 L'optimisation dans l'apprentissage automatique :

L'idée de l'apprentissage automatique mathématiquement est de minimiser la fonction objective en trouvant un extremum. Une foi la fonction objective déterminé, des méthodes d'optimisation numériques ou analytiques sont généralement utilisées pour résoudre ce problème de l'optimisation.

1.3.1 L'optimisation dans l'apprentissage supervisé :

L'apprentissage supervisé ou supervised learning est généralement utilisé pour des problèmes de régressions et de classifications.

L'idée est de minimiser l'erreur de la fonction objective suivante :

$$min\frac{1}{N}\sum_{i=1}^{N} L(y^i, f(x^i, \theta))$$

Avec :

N : Nombre d'échantillon d'entrainement

L : La perte

y^i : L'étiquette correspondante au i^{ème} échantillon

x^i : Le vecteur caractéristique de l'i^{ème} échantillon

θ : Le paramètre de la fonction qu'on veut minimiser

Il existe plusieurs types de fonctions de perte dans l'apprentissage, comme le carré de la distance euclidienne qui est utilisé pour les problèmes de régressions, mais la performance de cette fonction n'est pas nécessairement bien. Généralement des termes de régulation son ajouté pour la fonction objective comme suite :

$$min\frac{1}{N}\sum_{i=1}^{N} L\left(y^i, f(x^i, \theta)\right) + \gamma\|\theta\|$$

Avec : γ est le paramètre de compromis qui l'on peut déterminer avec une validation croisée

1.3.2 L'optimisation dans l'apprentissage non-supervisé :

Le principe dans l'apprentissage non-supervisé ou unsupervised learning est de diviser les échantillons dans des groupes différents. Dans ce problème on veut minimiser les pertes de la fonction objective suivante :

$$min \sum_{k=1}^{K} \sum_{x \in s_k} \|x - \mu_k\|_2^2$$

Avec :

K : Le nombre de groupe

x : Le vecteur caractéristique

s_k : Les échantillons du groupe k

μ_k : Le centre de k

1.3.3 L'optimisation dans l'apprentissage par renforcement :

Le but de l'apprentissage par renforcement ou reinforcement learning est de trouver une fonction de stratégie qui donne un résultat qui varie en fonction de l'environnement. Dans ce type de problème on doit maximiser le rendement cumulatif après l'exécution d'une série d'actions qui sont déterminées par la fonction suivante (Policy function) :

$$\max V_\pi(s) \quad \text{Avec : } V_\pi(s) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s]$$

V_π : La fonction de valeur

E : l'environnement

s : L'état

r : la récompense

γ : facteur d'actualisation

1.3.4 Les méthodes d'optimisation dans l'apprentissage automatique :

Dans l'apprentissage de la machine on a trois étapes principales :

- Construire un modèle.
- Définir la fonction objective.
- Trouver le maximum et le minimum de la fonction objective.

Dans la dernière étape on doit résoudre le modèle avec une des méthodes d'optimisation. Dans ce qui vient on va présenter quelques méthodes d'optimisation à utiliser.

a. La méthode du Gradient :

La méthode du gradient est la méthode d'optimisation la plus ancienne et la plus courante. Le but de cette méthode est de trouver les extrémums en déterminant un vecteur de direction basé sur le gradient avec une mise à jour des paramètres qui est effectuée pour la convergence progressive vers une valeur optimale de la fonction objective. Donc le gradient de descente minimise une fonction objective $j(\theta)$ avec son modèle de paramètreθ.

On rappelle que le gradient est un vecteur de dérivé partiel d'une fonction par rapport à ces entrées comme indique la fonction suivante :

$$\nabla_\theta J(\theta) = [\frac{\partial J(\theta)}{\partial \theta_1}, \dots, \frac{\partial J(\theta)}{\partial \theta_n}]$$

On introduit le gradient dans la fonction objective ce qui nous donne la fonction suivante :

$$\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta L(f(x^i, \theta), y^i)$$

b. La méthode du Batch et le mini-Batch :

Les algorithmes d'apprentissage automatique dans ces deux méthodes décomposent la fonction objective en somme d'exemples d'apprentissage. Les algorithmes d'optimisation pour l'apprentissage automatique calculent généralement chaque mise à jour de paramètre en fonction de la valeur attendue d'une fonction de coût estimé en utilisant uniquement un sous-ensemble des conditions de la fonction de coût complet.

Les algorithmes d'optimisation qui utilisent tous les ensembles d'entrainement sont appelés méthodes de gradients prédéterminées par 'Batch' car ils traitent tous les exemples d'apprentissage simultanément dans un grand lot.

Cette méthode calcule la moyenne des mises à jour des poids calculés pour toutes les erreurs d'entrainement pour ajuster les poids. Parmi les inconvénients de cette méthode est l'utilisation de toutes les données d'entrainement et de ne les mettre à jour qu'une seule foi, mais aussi la consommation du temps pour le calcul de la moyenne des mises à jour des poids est trop élevée. La méthode du Batch est utilisé à la base de la méthode du gradient on utilise un vecteur de direction qui a un paramètre qui indique le taux d'apprentissage comme indique la fonction suivante :

$$d = \epsilon_t \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L(f(x^i, \theta), y^i)$$

Avec :

d : Vecteur de direction

ϵ_t : Taux d'apprentissage à l'itération t

La mise à jour est : $\theta \leftarrow \theta - d$

Le mini-Batch est une méthode d'optimisation également basée sur la méthode du gradient mais contrairement à la méthode du Batch elle fait une mise à jour pour chaque exemple d'entrainement. Cette méthode réduit la variance des mises à jour des paramètres, ce qui peut conduire à une convergence plus stable.

c. La méthode du gradient stochastique (SGD) :

L'idée de cette méthode est d'utiliser un échantillon au hasard pour mettre à jour le gradient par itération, au lieu de calculer directement la valeur exacte du gradient.

Le gradient de la méthode stochastique (SGD) et ses variantes sont probablement les algorithmes d'optimisation les plus couramment utilisés pour l'apprentissage automatique général, et en particulier pour l'apprentissage profond. Il est possible d'obtenir une évaluation

objective du gradient en prenant un gradient moyen sur un mini-lot d'échantillons prélevés dans la distribution génératrice de données. L'algorithme SGD montre comment suivre cette estimation de chute de gradient. Le paramètre clé de l'algorithme SGD est la vitesse d'apprentissage. Le SGD utilise une vitesse d'apprentissage constante, mais dans la pratique, il est nécessaire de réduire progressivement la vitesse d'apprentissage au fil du temps, donc maintenant nous déterminons la vitesse d'apprentissage à l'itération. Ceci est dû au fait que l'estimateur de gradient SGD introduit une source de bruit (échantillonnage aléatoire d'exemples d'apprentissage), qui ne disparaît pas même après avoir atteint le minimum. À titre de comparaison, le véritable gradient de la fonction de coût total devient petit, puis 0 lorsque nous approchons et atteignons le minimum à l'aide d'un gradient par lots.

On résume l'algorithme du SGD :

Algorithm: Stochastic gradient descent (SGD) update:

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$

Require : Initial parameter θ

$k \leftarrow 1$

While stopping criterion not met **do**

Sample a minibatch of m examples from the training set$\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding trgets $y^{(i)}$.

Copute gradient estimate: $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^i; \theta), y^i)$

Apply update: $\theta \leftarrow \theta - \epsilon_k \hat{g}$

$k \leftarrow k + 1$

End while

Cette méthode peut converger avant de traiter toutes les données.

d. Le momentum :

Bien que la méthode du gradient stochastique soit toujours une méthode d'optimisation populaire, elle peut parfois être lente à apprendre. La méthode de Momentum vise à accélérer l'apprentissage, en particulier lorsque vous rencontrez de grandes courbures, des gradients petits mais constants ou des gradients bruyants. L'algorithme de Momentum accumule la moyenne mobile des gradients passés, qui décroît de manière exponentielle et continue de se déplacer.

Momentum vise principalement à résoudre deux problèmes : le mauvais état de la matrice Hessienne et la variance du gradient Stochastique.

L'algorithme du Momentum introduit une variable qui joue un rôle de vitesse : c'est la direction et la vitesse du paramètre se déplaçant dans l'espace. La vitesse est réglée sur la valeur moyenne d'un gradient négatif, qui chute de façon exponentielle.

Algorithm : Stochastic gradient descent (SGD) with momentum

Require : Learning rate ϵ, momentum parameter α

Require : Initial parameter θ, initial velocity v

While stopping criterion not met do

Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ withe corresponding targets $y^{(i)}$.

Compute gradient estimate : $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^i; \theta), y^i)$

Compute velocity : $v \leftarrow \alpha v - \epsilon g$.

Apply update : $\theta \leftarrow \theta + v$

End While

e. Le Nesterov Momentum :

Sutskever et al. (2013) ont introduit une variante de l'algorithme de Momentum, inspirée de la méthode du gradient accéléré de Nesterov (Nesterov, 1983, 2004).

Les paramètres α et ϵ sont similaires à la méthode standard du momentum. La différence entre le momentum de Nesterov et le momentum standard est l'endroit où le gradient est évalué. En utilisant la méthode de Nesterov, le gradient est évalué après l'application de la vitesse du courant. Par conséquent, le momentum de Nesterov peut être interprété comme une tentative d'ajouter un facteur de correction à la méthode standard actuelle. L'algorithme est représenté dans ce qui suit :

Algorithm : Stochastic gradient descent (SGD) with Nesterov momentum

Require : Learning rate ϵ, momentum parameter α

Require : Initial parameter θ, initial velocity v

While stopping criterion not met do

Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots., x^{(m)}\}$ withe corresponding labels $y^{(i)}$.

Apply interim update : $\tilde{\theta} \leftarrow \theta + \alpha v$.

Compute gradient (at interim point) : $g \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x^i; \tilde{\theta}), y^i)$

Compute velocity : $v \leftarrow \alpha v - \epsilon g$.

Apply update : $\theta \leftarrow \theta + v$

End While

f. AdaGrad :

L'algorithme AdaGrad ajuste individuellement le taux d'apprentissage de tous les paramètres du modèle par mise à l'échelle inversement proportionnelle à la racine carrée de la somme de toutes les valeurs quadratiques historiques du gradient. Le taux d'apprentissage du paramètre avec la plus grande dérivée partielle de la perte diminuera en conséquence rapidement, tandis que la baisse du taux d'apprentissage du paramètre avec la plus petite dérivée partielle sera relativement faible. Dans le contexte de l'optimisation convexe, l'algorithme AdaGrad présente des caractéristiques théoriques idéales. Cependant, empiriquement, pour la formation de modèles de réseaux neuronaux profonds, l'accumulation de gradients carrés dans les premiers stades de la formation entraînera une réduction prématurée et excessive du taux d'apprentissage effectif. AdaGrad convient à certains modèles d'apprentissage en profondeur, mais pas à tous.

Algorithm : The AdaGrad algorithm

Require : Global learning rate ϵ

Require : Initial parameter θ

Require : Small constant δ, perhaps 10^{-7}, for numerical stability

Initialize gradient accumulation variable **r = 0**

While stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ withe corresponding targets $y^{(i)}$.

Compute gradient : $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^i; \theta), y^i)$

Accumulate squared gradient :$r \leftarrow r + g \odot g$

Compute update : $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$. (Division and square root applied element-wise)

Apply update : $\theta \leftarrow \theta + \Delta\theta$

End While

g. RMSprop :

L'algorithme RMSprop modifie AdaGrad pour obtenir de meilleures performances dans un environnement non convexe en changeant l'accumulation de gradient en une moyenne mobile pondérée de manière exponentielle. AdaGrad est conçu pour converger rapidement lorsqu'il est appliqué à une fonction convexe. Lorsqu'elle est appliquée à une fonction non convexe pour entraîner un réseau neuronal, la trajectoire d'apprentissage peut traverser de nombreuses structures différentes et éventuellement atteindre une région qui est un bol localement convexe. AdaGrad réduit le taux d'apprentissage en fonction de toute l'histoire du gradient quadratique et peut avoir rendu le taux d'apprentissage trop faible avant d'atteindre une telle structure convexe. RMSprop utilise un moyen en décroissance exponentielle pour éliminer l'histoire du passé extrême afin qu'il puisse rapidement converger vers la fin d'un bol convexe comme s'il s'agissait d'une instance de l'algorithme AdaGrad initié dans ce bol.

Algorithm : The RMSProp algorithm

Require : Global learning rate ϵ, decay rate ρ

Require : Initial parameter θ

Require : Small constant δ, perhaps 10^{-6}, used to stabilize division by smallnumbers

Initialize gradient accumulation variable **r = 0**

While stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ withe corresponding targets $y^{(i)}$.

Compute gradient : $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^i; \theta), y^i)$

Accumulate squared gradient :$r \leftarrow \rho r + (1-\rho) g \odot g$

Compute update : $\Delta\theta \leftarrow -\frac{\epsilon}{\delta+\sqrt{r}} \odot g$ ($\frac{\epsilon}{\delta+\sqrt{r}}$applied element-wise)

Apply update : $\theta \leftarrow \theta + \Delta\theta$

End While

h. Adam :

Adam ou l'estimation du moment adaptatif est une méthode qui calcule les taux d'apprentissage adaptatif pour chaque paramètre et stocke une moyenne à décroissance exponentielle des gradients carrés passés ‘vt’ comme les méthodes Adadelta et RMSprop, Adam conserve également une moyenne des gradients passés qui décroît de façon exponentielle ‘mt’, semblable à le momentum. Mt et vt sont exprimées comme suivent :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Et on a :

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

Les deux paramètres précédents sont utilisés pour la mise à jour comme indique l’équation suivante :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon} \widehat{m_t}$$

Adam est l'abréviation de "Adaptive Moment Estimation", dans le contexte des algorithmes précédents, cela est mieux vu comme une variante de la combinaison de RMSprop et de Momentum avec quelques différences importantes, et c'est une autre façon d'utiliser les gradients passés pour calculer les gradients actuels. Adam utilise également le concept de moment adaptatif en ajoutant des fractions de gradients précédents au gradient actuel. Cet optimiseur est devenu assez répandu, et est pratiquement accepté pour être utilisé dans les réseaux neuronaux d'entraînement.

Il est facile de se perdre dans la complexité de certains de ces nouveaux optimiseurs. Il suffit de se rappeler qu'ils ont tous le même objectif : minimiser notre fonction de perte. Adam est généralement considéré comme assez robuste dans la sélection d'hyper paramètres, bien que la vitesse d'apprentissage doive parfois être modifiée par rapport à la valeur par défaut suggéré.

L'algorithme de cette méthode est présenté dans ce qui suit :

Algorithme : the ADAM algorithme
Require : Step size ϵ (Suggested default : 0.001)
Require : Exponential decay rates for moment estimates, ρ_1 and ρ_2 in [0,1). (Suggested defaults : 0.9 and 0.999 respectively)
Require : Small constant δ used for numerical stabilization (Suggested default :10^{-8})
Require : Initial parameters θ
Initialize 1st and 2nd moment variables s = 0, r = 0
Initialize time step t = 0
While stopping criterion not met **do**
Sample a minibatch of m examples from the training set{x(1), . . ., x(m)} with corresponding targets $y^{(i)}$.
Compute gradient : $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^i; \theta), y^i)$
$t \leftarrow t + 1$
Update bias first moment estimate : $s \leftarrow \rho_1 s + (1 - \rho_1) g$
Update biased second moment estimate : $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
Correct bias in first moment : $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
Correct bias in second moment : $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
Compute update : Compute update : $\Delta\theta \leftarrow -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$ (operation applied element-wise)
Apply update : $\theta \leftarrow \theta + \Delta\theta$
End While

1.3.5 Etat de l’art pour les méthodes d’optimisations :

a. Document 1 : Shiliang Sun, Zehui Cao, Han Zhu et Jing Zhao (2019)

Cet article a introduit et résumé les méthodes d'optimisation avec une étude de leurs applications dans divers domaines de l'apprentissage machine. Ils ont d'abord décrit les bases théoriques des méthodes d'optimisation de premier ordre, de haut ordre et les aspects sans dérivés, ainsi que les progrès de la recherche le long de ces dernières années. Ensuite, ils ont décrit les applications des méthodes d'optimisation dans différents scénarios d'apprentissage machine et les approches visant à améliorer leurs performances dans le matériel complémentaire. Enfin, ils ont discuté certains défis et problèmes ouverts dans les méthodes d'optimisation de l'apprentissage machine.

b. Document 2 : Soham De, Abhay Yadav, David Jacobs et Tom Goldstein (2017) :

Dans cette étude il nous propose des systèmes SGD alternatifs "big batch" qui permettent d'augmenter la taille des lots de manière adaptative sur le temps pour maintenir un rapport signal/bruit presque constant dans l'approximation du gradient. Les résultats ont des taux de convergence similaires à ceux des SGD classiques et n'exigent pas de convexité de l'objectif.

Les gradients d’haute-fidélité permettent une sélection automatique du taux d'apprentissage et ne nécessitent pas de dégradation par paliers. Les méthodes "big batch" sont donc facilement automatisées et peuvent fonctionner avec peu ou pas de surveillance.

c. Document 3 : Michael P. Perrone, Haidar Khan, Changhoan Kim, Anastasios Kyrillidis, Jerry Quinn et Valentina Salapura (2019)

Ce document présente une méthodologie pour sélectionner la taille des mini-batchs qui minimise le gradient stochastique (SGD) et le temps d'apprentissage pour les problèmes d'un seul et de plusieurs apprenants. En dissociant les questions d'analyse algorithmique des détails de mise en œuvre du matériel et des logiciels, ils révèlent une loi empirique inverse robuste entre la taille des mini lots et le nombre moyen des mises à jour du SGD qui doivent converger vers un seuil d'erreur spécifié. En combinant cette loi empirique inverse avec les performances

mesurées du système, ils créent un modèle précis et de forme fermée de la durée moyenne de formation et montrent comment ce modèle peut être utilisé pour identifier les implications quantifiables des aspects algorithmiques et matériels de l'apprentissage machine.

d. Document 4 : Diederik P. Kingma, Jimmy Lei Ba (2015)

Dans cette étude Diederik P. Kingma et Jimmy Lei Ba ont introduit un algorithme simple et efficace sur le plan du calcul pour l'optimisation par gradient des fonctions objectives stochastiques. Cette méthode est destinée aux problèmes d'apprentissage machine avec de grands ensembles de données et/ou des espaces de paramètres à haute dimension. La méthode combine les avantages de deux méthodes d'optimisation récemment populaires : la capacité d'AdaGrad à traiter des gradients clairsemés, et la capacité du RMSprop à traiter des objectifs non stationnaires. La méthode est simple à mettre en œuvre et ne nécessite que peu de mémoire. Les expériences confirment l'analyse sur le taux de convergence dans les problèmes convexes. Dans l'ensemble, nous avons constaté qu'Adam était robuste et bien adapté à un large éventail.

e. Résumé de l’état de l’art

Le tableau suivant résume les états de l’art ce qui permet éventuellement de comparer les différents documents :

Auteur et année	Objective	Méthode utilisée	Résultats
Shiliang Sun Zehui Cao Han Zhu Jing Zhao **Année** : 2019	Résumer les méthodes d'optimisation avec une étude de leurs applications dans divers domaines de l'apprentissage machine	La méthode du gradient (GD) Et la méthode du gradient stochastique (SGD)	Résume les méthodes d'optimisation avec une discussion des certains défies et problèmes
Soham De Tom Goldstein **Année** : 2017	Proposer des systèmes SGD alternatifs "big batch" qui permettent d'augmenter la taille	SGD "big batch"	Les résultats ont des taux de convergence similaires à ceux des SGD classiques et n'exigent pas de convexité de l'objectif
Michael P. Perrone Haidar Khan Changhoan Kim Anastasios Kyrillidis Jerry Quinn Valentina Salapura **Année** : 2019	Méthodologie pour sélectionner la taille des mini-batchs	Mini-batch	Une loi empirique inverse robuste entre la taille des mini lots et le nombre moyen des mises à jour du SGD
Diederik P. Kingma Jimmy Lei Ba **Année** : 2015	Introduire un algorithme simple et efficace sur le plan du calcul	ADAM	Adam est robuste et bien adapté à un large éventail

Tableau 1 : Synthèse de l'état de l'art

1.4 Conclusion :

Dans ce chapitre, nous avons présenté les différents concepts généraux de l'apprentissage automatique. En second lieu nous avons démontré les problèmes liés à la complexité de l'entrainement de la machine. Ensuite on a abordé la notion d'optimisation qui constitue le principal objectif de notre travail, a ce point nous avons abordé différentes méthodes

d’optimisation avec leurs algorithmes, où nous avons aussi donné quelques exemples de méthodes d’optimisation. Par la suite, nous allons présenter, la méthodologie suivie dans notre étude.

Chapitre 2

Méthodologie

2.1 Introduction :

Pour appliquer les méthodes d'optimisation présentée dans le chapitre précédant il nous faut une base de données (MNIST) que nous allons présenter dans ce chapitre, suivi de la méthodologie utilisée et l'architecture de notre modèle.

2.2 La base MNIST

La base de données de gravure manuscrite du MNIST contient un kit d'entrainement de 60 000 exemples et un kit de test de 10 000 exemples. Il s'agit d'un sous-ensemble de l'ensemble plus grand disponible dans le NIST. Les chiffres ont été normalisés et centrés sur l'image de taille fixe. Il s'agit d'une bonne base de données pour les personnes qui souhaitent essayer des techniques et des méthodes d'apprentissage pour reconnaître des modèles de données réels, avec un minimum d'effort de prétraitement et de formatage. Pour la base MNIST les quatre fichiers essentiels suivants sont disponible à téléchargé :

train-images-idx3-ubyte.gz: training set images (9912422 bytes)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

Nous notons que notre navigateur peut décompresser ces fichiers sans nous en informer. Si les fichiers téléchargés sont plus volumineux que ceux indiqués ci-dessus, ils ont été décompressés par notre navigateur. Nous avons juste besoin de changer leur nom pour supprimer l'extension .gz. Certaines personnes demandent "mon application ne peut pas ouvrir ces fichiers image".

Ces fichiers n'ont pas de format d'image standard. Nous devons écrire notre propre programme (très simple) pour les lire. Le format de fichier est décrit au bas de cette page.

Les images originales en noir et blanc (à deux niveaux) du format NIST ont été normalisées pour tenir dans le champ de 20 x 20 pixels tout en conservant leurs proportions. Les images résultantes contiennent des niveaux de gris résultant de la technique d'anticrénelage utilisée par l'algorithme de normalisation. Les images ont été centrées sur l'image 28 x 28 pixels en calculant le centre de masse des pixels et en décalant l'image pour placer ce point au centre du champ 28x28 pixels.

Pour certaines méthodes de classification (en particulier les méthodes basées sur un modèle telles que SVM et les KNN), le taux d'erreurs s'améliore lorsque les nombres sont centrés par la boîte englobante plutôt que par le centre de masse. Si nous effectuons ce type de prétraitement, nous devons l'indiquer dans notre mémoire.

La base de données MNIST est construite à partir de bases de données spéciales NIST 3 et 1, qui contiennent des images binaires de chiffres manuscrits. Le NIST a désigné SD-3 initialement comme module d’entrainement et SD-1 comme module de test. Cependant, SD-3 est plus propre et plus facile à identifier que SD-1. La raison en est que SD-3 est collecté auprès des employés du Census Bureau - Le Bureau du recensement des États-Unis-, tandis que SD-1 est collecté auprès des lycéens. Afin de tirer des conclusions raisonnables de l'expérience d'apprentissage, les résultats doivent être indépendants du choix de l'ensemble de formation et du test dans tous les échantillons. Par conséquent, il est nécessaire de créer une nouvelle base de données en mélangeant des ensembles de données NIST.

Le module de formation MNIST comprend 30 000 modèles de SD-3 et également 30 000 modèles de SD-1. Notre testeur comprend 5 000 SD-3 et 5 000 SD-1. Les 60 000 modèles de formation contiennent des exemples d'environ 250 auteurs.

Le SD-1 contient 58 527 images de nombres écrits par 500 auteurs différents. Contrairement à SD-3, où les blocs de données de chaque appareil d'enregistrement apparaissent en séquence,

les données SD-1 sont encodées. Les identités des auteurs du SD-1 sont disponibles et les créateurs ont utilisé ces informations pour déchiffrer les auteurs. Ensuite, ils ont divisé SD-1 en deux parties : les personnages écrits par les 250 premiers auteurs ont été inclus dans le nouveau jeu d'entraînement. Les 250 auteurs restants ont été inclus dans le kit de test. C'est pourquoi ils avaient deux séries de près de 30 000 exemplaires. La nouvelle série de formation a été complétée par suffisamment d'exemples SD-3, du modèle 0 pour créer une série complète de 60000 modèles de formation. De même, la nouvelle série de tests a été complétée par des exemples de SD-3 du modèle n ° 35 000, créant une série complète de 60 000 modèles de test. Le site ne fournit qu'un sous-ensemble de 10 000 images de test (SD-1 fournit 5 000, SD-3 fournit 5 000). 60 000 échantillons de formation sont disponibles.

De nombreuses méthodes ont été testées sur cet ensemble de formation et l'ensemble de test. Voici quelques exemples ;

- Certaines expériences ont utilisé une version de base de données où l'image d'entrée était décalée (en calculant le grand axe de la forme le plus proche de la direction verticale et en la déplaçant tout droit pour la rendre verticale).

- Dans d'autres expériences, l'ensemble d'apprentissage a également ajouté une version artificiellement déformée de l'échantillon d'apprentissage d'origine. La distorsion est une combinaison aléatoire de décalage, zoom, inclinaison et compression.

2.3 Les séparateurs à vaste marge (SVM) :

SVM est une méthode pour résoudre des problèmes de classification, ce dernier appartient à la catégorie des classificateurs linéaires (utilisant la séparation linéaire des données), et il a sa propre méthode pour trouver les limites entre les catégories. Pour que le SVM trouve la frontière, il est nécessaire de fournir des données d'apprentissage, le SVM estimera l'emplacement le plus raisonnable de la frontière : c'est la période d'apprentissage, pour tout algorithme d'apprentissage automatique.

Le séparateur en question ne sépare pas seulement les points dans le plan, Il peut en fait séparer des points dans n'importe quel espace dimensionnel. Par exemple, si nous essayons de classer les fleurs par espèce, lorsque nous connaîtrons la taille de la fleur, le nombre de pétales et le diamètre de la tige, nous travaillerons sur la dimension 3. Un autre exemple est la reconnaissance d'image : une image noire et blanche de 28 * 28 pixels contiennent 784 pixels, il s'agit donc d'un objet de taille 784. Par conséquent, il fonctionne généralement dans un espace de milliers de tailles. Fondamentalement, SVM ne recherchera que des hyperplans capables de séparer les deux types de problèmes.

Dans un espace vectoriel de dimension finie n, l'hyperplan est un sous-espace vectoriel de dimension $n - 1$. Par conséquent, dans un espace de taille 2, un hyperplan sera une ligne, dans un espace de taille 3, un hyperplan sera un plan, et ainsi de suite.

L'objectif des SVM est de trouver une séparation linéaire permettant de distinguer les deux classes. Le classifieur se présente sous la forme d'une combinaison linéaire des variables comme indique la fonction suivante :

$$f(x) = x^T\beta + \beta_0$$

$$f(x) = x_1\beta_1 + x_2\beta_2 + \cdots + \beta_0$$

Avec : β les paramètres à estimer.

L'hyperplan optimal maximise la distance entre la frontière de séparation et les points de chaque classe qui lui sont le plus proche comme indique la figure suivante :

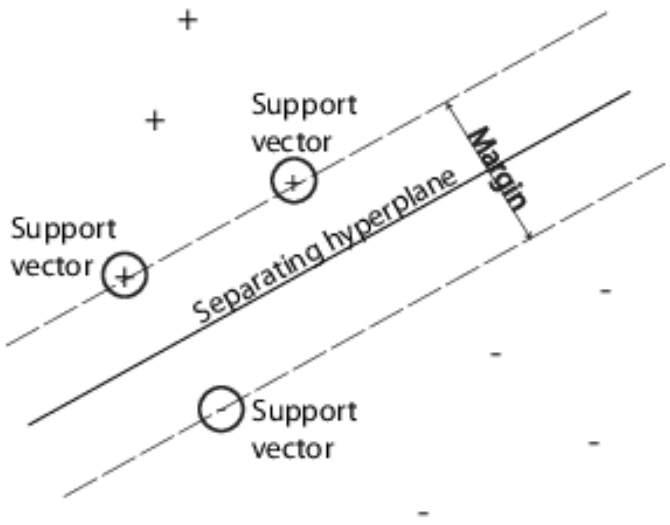


Figure 8 : Hyperplan optimal

On doit calculer les paramètres suivants :

- Distance d'un point quelconque x avec la frontière : $d = \frac{|x^T\beta+\beta_0|}{\|\beta\|}$
- La marge maximale est égale à : $\delta = \frac{2}{\|\beta\|}$
- Maximiser la marge revient à minimiser la norme du vecteur de paramètres β : $max\frac{2}{\|\beta\|} \Leftrightarrow min\|\beta\|$

2.4 Le réseau de neurones convolutif (CNN) :

Les réseaux convolutifs sont une forme particulière de réseau neuronal multicouches dont l'architecture des connexions est inspirée de celle du cortex visuel des mammifères, Leur conception suit la découverte de mécanismes visuels dans les organismes vivants.

Ces réseaux de neurones artificiels (aussi baptisés réseau de neurones à convolution, ou CNN) sont capables de catégoriser les informations des plus simples aux plus complexes. Ils consistent en un empilage multicouche de neurones, des fonctions mathématiques à plusieurs paramètres ajustables, qui pré-traitent de petites quantités d'informations. En apprentissage automatique, un réseau de neurones convolutifs est un type de réseau de neurones artificiels acycliques C'est l'une des méthodes les plus pratiques pour la classification comme on a dit précédemment.

Dans cette partie on va parler des deux couches qui construisent notre CNN.

2.4.1 Convolution Layer (la couche de convolution) :

La couche de convolution est la pierre angulaire du réseau convolutif qui effectue la majeure partie du travail de calcul, vue d'ensemble et intuition sans cervelle. Regardons d'abord ce que la couche CONV calcule sans l'analogie cerveau / neurone. Les paramètres de la couche CONV consistent en un ensemble de filtres d'apprentissage. Comprend des filtres qui convertissent les images. Nous appellerons ces filtres des filtres de convolution. Chaque filtre a un petit espace (largeur et hauteur) mais s'étend sur toute la profondeur du volume d'entrée. La couche de convolution génère de nouvelles images appelées « cartes de caractéristiques ». La carte met en évidence les caractéristiques uniques de l'image originale. Cette Couche fonctionne complètement différemment des autres couches du réseau neuronal elle n'utilise pas le poids du lien et la somme pondérée, et le processus d'introduction de l'image à travers les filtres de convolution produit une carte de caractéristiques, la figure suivante explique le processus de la couche de convolution :

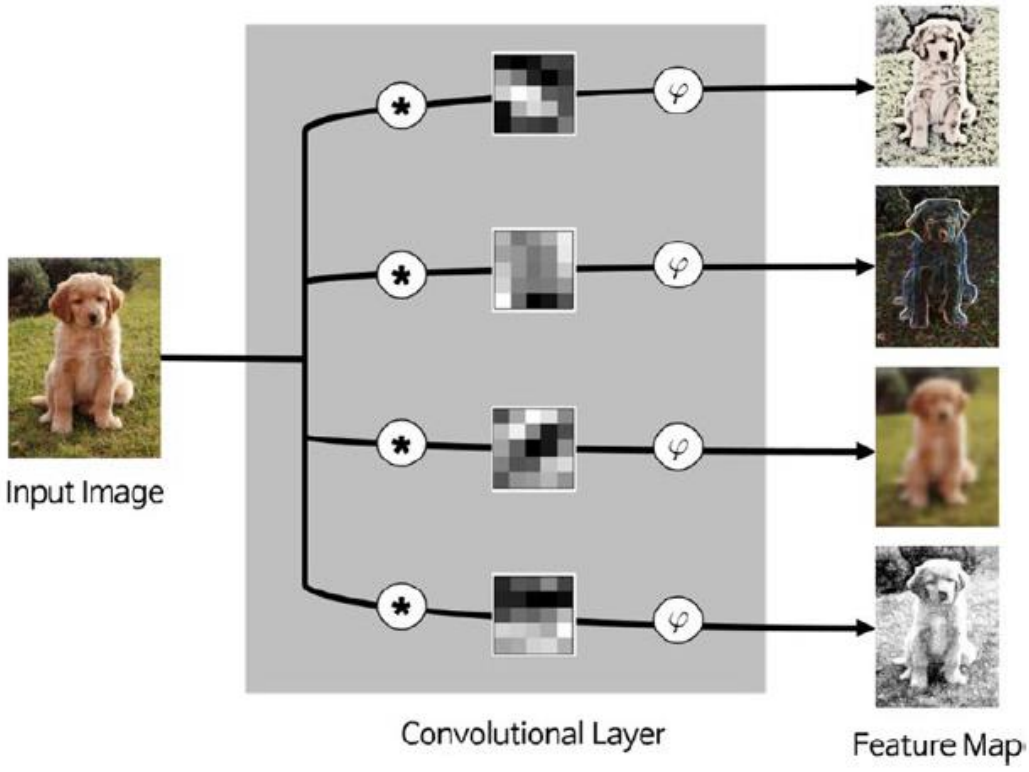


Figure 9 : Exemple d’une couche de convolution

2.4.2 Pooling layer :

C’est une couche de regroupement qui réduit la taille de l'image car elle combine des images de pixels adjacentes d'une zone spécifique de l'image en une seule valeur représentative. Cette technique est typiquement adoptée par de nombreux autres systèmes de traitement d'images. Afin d'opérer dans la couche de pooling, nous devons déterminer comment choisir les pixels groupés de l'image et comment définir le représentant valeur. Sélectionner généralement les pixels adjacents dans une matrice carrée, puis le nombre de pixels fusionnés varie en fonction du problème. La valeur représentative est généralement définie comme la valeur moyenne ou maximale des pixels sélectionnés. Le fonctionnement de la couche de pooling est très simple.

Prenons un exemple. Prenons une image d'entrée de 4×4 pixels, représenté par la matrice illustrée dans la figure ci-dessous :

1	1	1	3
4	6	4	8
30	0	1	5
0	2	2	4

Figure 10 : Matrice représentatrice de l'image d'entrée

Nous fusionnons les pixels de l'image d'entrée dans une matrice 2×2 sans éléments qui se chevauchent. Une fois que l'image d'entrée a traversé la couche de regroupement, Il est réduit à une image de 2×2 pixels. La figure suivante montre la fusion des résultats en moyen pooling (mean pooling) et en max pooling.

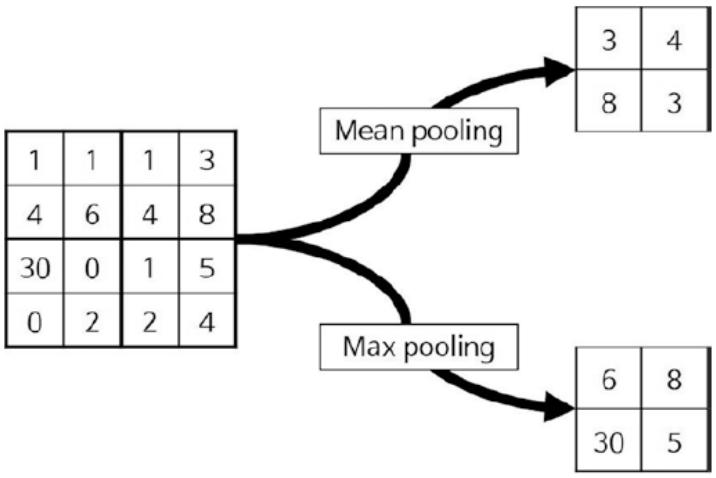


Figure 11 : Fusion des résultats en moyen pooling et en max pooling.

2.5 Exemple de classification des chiffres

Dans cette partie en va appliquer ce qu'on a vu précédemment sur la classification des chiffres en utilisant le CNN et les SVM.

2.5.1 Classification des chiffres avec le CNN :

On utilise un CNN comme une architecture pour classer les chiffres de la base MNIST avec une pile de Couches Conv2D et MaxPooling2D comme indique le code du python dans la figure suivante :

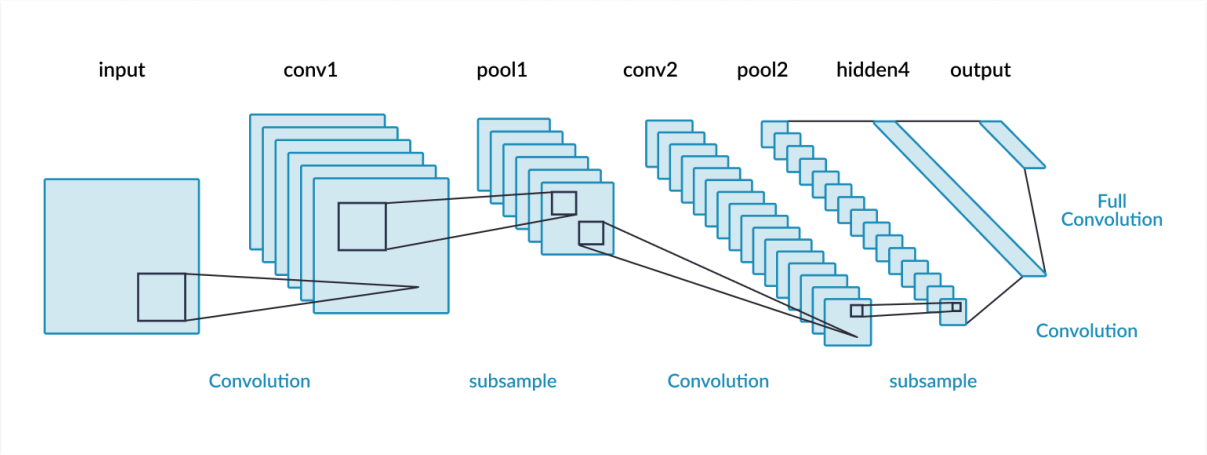


Figure 12 : Exemple d'une architecture d'un modèle CNN

Il est important de noter qu'un CNN prend en entrée des tenseurs de forme (hauteur image, largeur image, canal de l'image) (sans compter la dimension du batch). Dans ce cas, nous allons configurer le CNN pour traiter les entrées de taille (28, 28, 1), qui est le format d'image du MNIST. Nous allons le faire en passant l'argument input_shape= (28, 28, 1) à la première couche.

On va avoir une sortie de chaque couche Conv2D et MaxPooling2D comme un tenseur 3D forme (hauteur, largeur, canal). Les dimensions de largeur et de hauteur ont tendance à diminuer au fur et à mesure que nous approfondissons le réseau. L'étape suivante consiste à introduire le

dernier tenseur de sortie (format (3, 3,64)) dans une couche réseau de classificateurs connectés. Ces classificateurs traitent des vecteurs unidimensionnels et la sortie actuelle est un Tenseur 3D. Nous devons d'abord aplatir la sortie 3D en 1D, puis ajouter des couches denses Au sommet. Nous allons faire une classification à 10 voies, en utilisant une couche finale avec 10 sorties et une activation Softmax.

Ensuite nous allons entrainer le CNN sur la base de chiffres du MNIST et nous allons le refaire pour plusieurs optimiseurs tel que le SGD, Adam… etc.

2.5.2 Classification des chiffres avec les SVM :

Dans cette partie on va utiliser la classification des chiffres sur la base du MNIST avec les SVM. On importe au début les bibliothèques nécessaires, les ensembles de données, les classificateurs et les mesures de performance, Après ça on importe la base MNIST « Dataset ». Le champ de données est un tableau de 70k x 784, chaque ligne représente les pixels de l'image 28x28=784. On Crée un classificateur SVM et à la fin on a la partie d'entrainement et prédiction.

2.6 Conclusion :

Nous avons présenté dans ce chapitre la méthodologie de notre application, au début on a commencé par définir la base de données MNIST et les séparateurs a vaste marge (SVM), au passage on a expliqué ce qu'un réseau de neurones convolutif et ensuite on s'est retourné vers l'application dont la classification des chiffres en utilisant les CNN et les SVM ainsi que leur code python. Dans le chapitre suivant nous allons donner les résultats de l'application.

Chapitre 3

Application

3.1 Introduction

Dans ce chapitre nous allons présenter les résultats de l'application des méthodes d'optimisation dans l'apprentissage automatique sur l'exemple de la classification des chiffres, afin de discuter les résultats obtenus.

3.2 Résultats

Nous allons tester plusieurs méthodes d'optimisation en changeant l'optimiseur dans le modèle présenté dans le chapitre précédant pour chaque méthode.

3.2.1 SGD

L'application de l'optimiseur SGD nous donne une précision de 0.9708 (97.08%) et une perte de 0.0940 (9.40%).

La méthode du gradient stochastique est un principe plus général, dans lequel la direction de la mise à jour est une variable aléatoire, dont l'espérance est le véritable gradient qui nous intéresse. La SGD peut être beaucoup plus rapide que la méthode du gradient ordinaire, parce que le nombre de mises à jour est beaucoup plus grand et donc la convergence est plus rapide mais elle l'est moins par rapport aux autres méthodes qui sont plus développé.

3.2.2 AdaGrad

L'application de l'optimiseur AdaGrad nous donne une précision de 0.9929 (99.29%) et une perte de 0.0238 (2.38%).

AdaGrad est une amélioration de la méthode SGD qui détermine automatiquement un taux d'apprentissage pour chaque paramètre, c’est la raison pour laquelle AdaGrad donne de meilleurs résultats par rapport à la SGD.

3.2.3 RMSprop

L’application de l’optimiseur RMSprop nous donne une précision de 0.9941 (99.41%) et une perte de 0.0200 (2%).

RMSprop est également une méthode développée par rapport à AdaGrad dans laquelle le taux d'apprentissage est adapté pour chacun des paramètres. L'idée est de diviser le taux d'apprentissage pour un poids par une moyenne mobile des amplitudes des gradients récents pour ce poids. Ainsi, la moyenne mobile est d'abord calculée en termes de moyenne carrée.

RMSprop a montré une bonne adaptation du taux d'apprentissage dans différentes applications et elle a aussi donné de bons résultats dans notre cas.

3.2.4 Adam

L’application de l’optimiseur Adam nous donne une précision de 0.9926 (99.26%) et une perte de 0.0226 (2.26%).

Adam est une mise à jour de l'optimiseur RMSprop. Dans cet algorithme d'optimisation, les moyennes courantes des gradients et des seconds moments des gradients sont utilisées. On remarque que cette méthode donne presque les mêmes résultats que RMSprop.

3.2.5 Résumé des résultats

Méthode utilisé	Accuracy (Précision)	Loss (Perte)	Commentaires
SGD	0.9708 (97.08%)	0.0940 (9.40%)	L'algorithme SGD est le plus basic parmi tous les autres algorithmes. C'est pourquoi la perte est la plus grande par rapport aux autres méthodes.
AdaGrad	0.9929 (99.29%)	0.0238 (2.38%)	L'accumulation de gradients carrés dans les premiers stades de l'entrainement entraînera une réduction prématurée et excessive du taux d'apprentissage effectif. Ce qui explique la perte qui a baissé dans cette méthode par rapport au SGD.
RMSprop	0.9941 (99.41%)	0.0200 (2%)	L'algorithme RMSprop modifie AdaGrad pour obtenir de meilleures performances dans un environnement non convexe en changeant l'accumulation de gradient en une moyenne mobile pondérée de manière exponentielle.
Adam	0.9926 (99.26%)	0.0226 (2.26%)	Adam est une combinaison de RMSprop et Momentum avec quelques différences importantes. Premièrement, dans Adam, le Momentum est inclus directement en tant qu'estimation du moment de premier ordre (avec poids exponentiel) du gradient. Deuxièmement, Adam inclut des corrections d'erreur dans les moments du premier ordre (angles) et des estimations du second ordre (non focalisées) pour tenir compte de leur initialisation initiale.

Tableau 2 : Tableau des résultats

3.3 Discussions

On remarque que ces méthodes d'optimisation utilisée pour la classification des chiffres ont donné des bons résultats au terme de perte et de précision. Nous avons testé plusieurs optimiseurs dans cette partie qui nous ont donné différents résultats. Le SGD a donné une précision de 0.9708 (97.08%) et une perte de 0.0940 (9.40%), comme on peut remarquer, cette méthode donne les moins bons résultats car elle est ancienne et elle fait moins de mises à jour par rapport aux autres méthodes. AdaGrad est la deuxième méthode qu'on a testée et nous a donné une précision de 0.9929 (99.29%) et une perte de 0.0238 (2.38%), car c'est une amélioration de la méthode SGD. RMSprop est la troisième méthode qu'on a testée et qui a donné une précision de 0.9941 (99.41%) et une perte de 0.0200 (2%), on remarque clairement la différence entre le SGD et cette méthode qui a donné de bons résultats grâce à la différence de calculs qui cause une convergence plus rapide. Adam est la dernière méthode qu'on a appliquée et qui nous a donné une Précision de 0.9926 (99.26%) et une perte de 0.0226 (2.26%), La seule grande différence entre ces méthodes c'est le SGD qui a donné un grand taux de pertes par rapport aux autres car c'est la méthode la plus basic. RMSprop et Adam ont donné les meilleures performances étant donné que ce sont la version la plus développé des optimisateurs.

3.4 Conclusion

Nous avons vue dans ce chapitre les résultats obtenus de l'application des méthodes d'optimisation sur la classification des chiffres en employant différents optimiseurs, on les a ensuite comparés pour conclure qu'Adam et RMSprop sont les deux méthodes les plus convenables pour notre étude.

Conclusion

L’intelligence artificielle est une révolution scientifique qui va changer le monde, c’est l’avenir de la technologie, un futur fascinant. Dans notre recherche nous nous sommes concentrés sur le côté d’optimisation d’un point de vue mathématique et algorithmique et nous avons appliqué plusieurs méthodes sur la classification des chiffres.

Nous avons vue différents optimiseurs mais pas tous ; il y’a d’autre optimiseurs que nous n’avons pas utilisés tels que Mini-Batch et Momentum et qui donnent d’assez bons résultats. Nous avons aussi brièvement exploré la partie mathématique de l’optimisation sans s’approfondir ; nous nous sommes plutôt concentrés sur la partie programmation.

Durant le développement de n’importe quel algorithme qui nécessite l’utilisation des réseaux de neurones, Il est important de savoir modifier ses paramètres et de préciser les critères de comparaison entre les résultats obtenus.

La porte de la recherche sur les applications de l’intelligence artificielle est toujours ouverte à de nouveaux algorithmes et propositions. Nous pensons que les ingénieurs doivent aller plus loin dans ce domaine et doivent acquérir davantage de connaissance afin de mettre en place plus de théories et de les convoiter.

Bibliographie

[1] Kim, P. (2017). Matlab deep learning. *With Machine Learning, Neural Networks and Artificial Intelligence*, *130*, 21.

[2] Chollet, F. (2018). Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek. MITP-Verlags GmbH & Co. KG.

[3] Sun, S., Cao, Z., Zhu, H., & Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*.

[4] De, S., Yadav, A., Jacobs, D., & Goldstein, T. (2017, April). Automated inference with adaptive batches. In *Artificial Intelligence and Statistics* (pp. 1504-1513).

[5] Perrone, M. P., Khan, H., Kim, C., Kyrillidis, A., Quinn, J., & Salapura, V. (2019). Optimal Mini-Batch Size Selection for Fast Gradient Descent. *arXiv preprint arXiv:1911.06459*.

[6] Diederik, P. K., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization International Conference On Learning Representations, 2015.

[7] Cornuéjols, A. (2002). Une nouvelle méthode d'apprentissage : Les SVM. Séparateurs à vaste marge. *Bulletin de l'AFIA*, *51*, 14-23.

[8] Sun, S., Cao, Z., Zhu, H., & Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. IEEE transactions on cybernetics.

[9] TAFFAR, M. INITIATION AL'APPRENTISSAGE.

[10] Shalev-Shwartz, S., & Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, *14*(Feb), 567-599.

[11] Van Gysel, C., Kanoulas, E., & de Rijke, M. (2017, April). Pyndri: a python interface to the indri search engine. In *European Conference on Information Retrieval* (pp. 744-748). Springer, Cham.

[12] URL : https://keras.io/

[13] Roosa, K., Lee, Y., Luo, R., Kirpich, A., Rothenberg, R., Hyman, J. M., ... & Chowell, G. (2020). Real-time forecasts of the COVID-19 epidemic in China from February 5th to February 24th, 2020. *Infectious Disease Modelling*, *5*, 256-263.