PROJET ANNUEL

MODELE DE MACHINE LEARNING POUR DE LA CLASSIFICATION D'EXOPLANETE

GAZEUSE - TERRESTRE - TYPE NEPTUNE

PROJET REALISE PAR

TINHINANE ISSAD - AMEL ZITOUN- CAMERON DEBLIQUY

ESGI
école supérieure de génie informatique

Résumé du document

Ce document constitue le rapport du projet annuel pour l'année 2023-2024 de Mme. Tinhinane ISSAD, de Mme. Amel ZITOUN et de M. Cameron DEBLIQUY. Il est destiné au jury de l'ESGI lors de la présentation finale.

Liste de diffusion

Société	Nom	Titre
ESGI	Amel ZITOUN	Etudiant
ESGI	Tinhinane ISSAD	Etudiant
ESGI	Cameron DEBLIQUY	Etudiant
ESGI	Nicolas VIDAL	Examinateur

Rédigé par

Nom	Amel ZITOUN	Tinhinane ISSAD	Cameron DEBLIQUY
Date	11-06-2024	11-06-2024	11-06-2024
Signature			

1. Introduction

Les exoplanètes, ou planètes extrasolaires, sont des planètes situées en dehors de notre système solaire. La classification des exoplanètes est un domaine de recherche crucial en astronomie, car elle permet de mieux comprendre la diversité et les caractéristiques des planètes dans notre galaxie. Ce projet vise à développer et comparer trois modèles de classification pour identifier et classer les exoplanètes : un modèle linéaire, un perceptron multicouche (PMC), et un modèle de fonction de base radiale (RBF).

À travers ce rapport, nous allons présenter le résultat de nos travaux sur la mise en place de modèles de Machine Learning appliqués à la classification multi-classes d'exoplanètes. Nous détaillerons les étapes de collecte et de préparation des données, l'implémentation des algorithmes, ainsi que les résultats obtenus lors des expérimentations.

Ce travail vise à offrir une vision critique et approfondie des outils d'intelligence artificielle utilisés pour la classification d'exoplanètes.

1.1 Contexte

C'est dans le cadre de notre formation en troisième année à l'École Supérieure de Génie Informatique (ESGI) de Paris, que nous avons entrepris un projet ambitieux visant à étudier les performances des algorithmes de Machine Learning appliqués à une problématique de classification complexe.

Initialement, notre projet portait sur la classification de drapeaux. Cependant, désireux de relever un défi scientifique plus important, nous avons décidé de concentrer nos efforts sur la classification d'exoplanètes à partir d'images issues du catalogue de la NASA.

Cette nouvelle orientation nous permet de contribuer à un domaine de recherche actuel et passionnant, tout en mettant en pratique des techniques avancées de Machine Learning.

1.2 Enjeux et objectif

L'objectif principal de ce projet est de développer des modèles de Machine Learning capables de classifier des exoplanètes, en démontrant l'efficacité de différentes approches, allant des modèles les plus simples aux plus complexes. Nous avons choisi d'implémenter et de comparer un modèle linéaire, un Perceptron Multicouche (PMC), un Réseau à Fonction de Base Radiale (RBFN). Tous ces modèles ont été implémentés en langage Rust pour son efficacité et ses performances, et par la suite intégrés sous forme de bibliothèques dynamiques utilisables depuis des scripts Python.

Ce projet vise non seulement à évaluer la précision et la performance de chaque modèle, mais aussi à analyser leur rapidité de convergence et à identifier les phénomènes de sous-apprentissage et de sur-apprentissage. En outre, nous explorerons les biais potentiels présents dans les données et l'impact des différents paramètres sur les performances des modèles. Ce travail se veut une contribution critique et approfondie sur l'application des outils d'intelligence artificielle à la classification d'exoplanètes, offrant des perspectives pour de futures recherches et améliorations.

2. Le jeu de données

Dans le cadre de notre projet, nous avons décidé de réaliser une classification d'exoplanètes en trois catégories distinctes : les géantes gazeuses, les planètes terrestres et les planètes de type Neptune. Chacune de ces classes possède des caractéristiques spécifiques qui permettent de les identifier et de les distinguer.

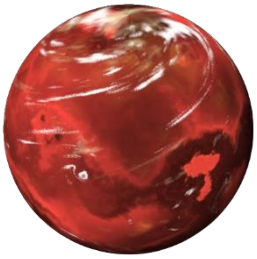
Géantes Gazeuses : Ces exoplanètes sont principalement composées de gaz, principalement d'hydrogène et d'hélium. Elles sont similaires à Jupiter et Saturne dans notre système solaire. Ces planètes ont une grande taille et une faible densité, et elles possèdent souvent des systèmes d'anneaux et de nombreuses lunes.

Planètes Terrestres : Ces exoplanètes ont une composition rocheuse et sont similaires à la Terre et à Mars. Elles sont caractérisées par leur surface solide et leur densité plus élevée. Ces planètes peuvent posséder des atmosphères variées, mais elles sont principalement composées de métaux et de silicates.

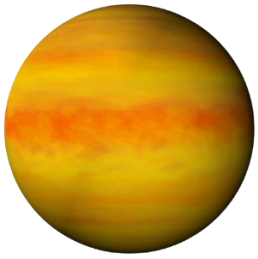
Planètes de Type Neptune : Ces exoplanètes sont intermédiaires entre les géantes gazeuses et les planètes terrestres. Elles sont similaires à Neptune et Uranus, avec des compositions dominées par l'eau, l'ammoniac et le méthane, souvent sous forme de glace. Ces planètes possèdent une atmosphère épaisse et une masse plus grande que celle des planètes terrestres, mais inférieure à celle des géantes gazeuses.

61-Virginis-b (Type Neptune)

K2-343-b (Terrestre)



BD+20-2457-c (Gazeuse)



Toutes les données utilisées dans ce projet proviennent du catalogue de la NASA. Il est important de noter que ces données ne sont pas des images réelles des exoplanètes. En raison des immenses distances impliquées et des limitations technologiques actuelles, il est impossible d'obtenir des images détaillées et précises des exoplanètes. Par conséquent, les images utilisées sont des représentations 3D générées en fonction des caractéristiques physiques et atmosphériques des exoplanètes telles que déterminées par les observations indirectes (comme la spectroscopie, la photométrie et d'autres méthodes de détection).

Ces représentations artistiques sont basées sur les données disponibles et sont utilisées pour visualiser et classifier les exoplanètes en fonction de leurs propriétés distinctives. Elles offrent une approximation visuelle qui aide à la compréhension et à l'analyse des caractéristiques des différentes classes d'exoplanètes.

2.1 Constitution du jeu de données

Pour constituer notre jeu de données, nous avons d'abord mis en place un script pour récupérer les liens vers toutes les exoplanètes répertoriées par la NASA, ce qui

représente exactement 5612 entrées. Chaque lien donne un accès direct aux informations détaillées sur chaque planète.

Ensuite, nous avons développé un deuxième script en Python, utilisant la bibliothèque Selenium, outil permettant de contrôler des navigateurs web à des fins de tests automatisés, en simulant les actions de l'utilisateur comme les clics et les saisies de texte. Notre script visait à générer artificiellement les pages web des exoplanètes pour capturer les images.

Cependant, nous avons rapidement rencontré un problème de complexité. Les exoplanètes ne sont pas représentées par des images statiques, mais par des modèles 3D interactifs. Il n'était donc pas possible de simplement télécharger les images. Nous avons dû prendre des captures d'écran des modèles 3D. Pour cela, il était nécessaire de simuler des clics de souris pour supprimer les ombres potentielles sur les planètes et enlever l'interface utilisateur présente sur les pages. Comme la visualisation 3D gérée par Selenium n'était pas optimisée, le processus de capture d'images était lent, prenant environ 1 minute et 30 secondes par image. De plus, le taux de réussite n'était pas de 100%, certaines captures échouaient en raison d'erreurs diverses.

Pour optimiser ce temps de récupération, nous avons développé un second script en JavaScript utilisant la bibliothèque Puppeteer. Puppeteer est une bibliothèque Node.js qui fournit une API de haut niveau pour contrôler Chrome ou Chromium. Puppeteer offre des performances améliorées par rapport à Selenium pour certaines tâches. En utilisant Puppeteer, nous avons pu réduire le temps de capture à environ 50 secondes par image.

Nous avons décidé de ne pas récolter des planètes provenant d’autres sources car ces images n’étaient pas des représentations 3D mais des illustrations artistiques qui ne reflètent pas fidèlement les caractéristiques des types de planètes. Afin d’éviter un biais potentiel pouvant augmenter la complexité du modèle, nous avons choisi de nous concentrer uniquement sur les planètes répertoriées par la NASA.

En raison de la complexité du processus de collecte des données, nous avons initialement récolté 1500 images correctes. Après des expérimentations avec nos modèles, nous avons constaté que ce nombre d’images était suffisant pour obtenir de bons résultats et permettre une convergence appropriée selon les modèles. Cette quantité d’images nous a permis de valider nos hypothèses et de démontrer l’efficacité des modèles de classification utilisés.

2.2 Caractéristique et manipulation du jeu de données

Nos images sont caractérisées par différentes informations importantes, telles que la taille de l'image, la taille du fichier, la profondeur des bits et la résolution.

- **Taille de l'image :** La taille d'une image est définie par ses dimensions en pixels. Elle est calculée comme suit : hauteur x largeur x dpi (dots per inch). Cette mesure détermine la qualité et le niveau de détail de l'image.
- **Taille du fichier :** La taille du fichier d'une image dépend de ses dimensions, de sa profondeur de bits et de sa résolution.
- **Profondeur des bits :** La profondeur des bits indique le nombre de bits utilisés pour représenter la couleur de chaque pixel de l'image. Plusieurs options sont possibles :
    - **Bitonal :** Utilise 1 bit par pixel, permettant 2 couleurs (souvent noir et blanc).
    - **Nuances de gris :** Utilise 8 bits par pixel, permettant 256 niveaux de gris (de 0 à 255).
    - **Couleurs :** Utilise 24 bits par pixel, permettant de représenter 16,7 millions de couleurs (chaque canal de couleur, rouge, vert et bleu, utilisant 8 bits).
- **Résolution :** La résolution d'une image est le nombre de pixels par unité de surface, généralement mesuré en dpi. Elle indique le niveau de détail et la clarté de l'image. Plus la résolution est élevée, plus l'image est détaillée.

Une fois les données récoltées, nous avons d'abord mis en place un script pour rogner toutes les images afin de les rendre au format carré. Ensuite, selon les besoins de nos tests, nous avons manipulé les images en jouant sur leur taille, en les réduisant au maximum, et en ajustant la composante couleur, en les convertissant soit en niveaux de gris, soit en RGB.

Nous avons également normalisé notre jeu de données. En effet, les images ne sont que des vecteurs de vecteurs de valeurs variant de 0 à 255. Cependant, pour améliorer l'efficacité de nos modèles, il est essentiel de normaliser ces données. La normalisation permet de ramener les valeurs à une plage comprise entre -1 et 1, en utilisant la formule suivante : (valeur - moyenne des valeurs) / écart-type des valeurs. Cette transformation

aide à accélérer la convergence des modèles d'apprentissage et à améliorer leur performance.

3. Les modèles

Une fois notre jeu de données constitué, nous avons mis en place différents modèles de Machine Learning pour la classification des exoplanètes. Nous avons utilisé un modèle linéaire, un perceptron multicouche, et un modèle de fonction de base radiale (RBF).

3.1 Le modèle linéaire

En Machine Learning, un modèle linéaire est couramment utilisé pour résoudre des problèmes de régression et de classification. Ce type de modèle cherche à établir une relation linéaire entre les caractéristiques d'un jeu de données et la variable cible à prédire. Le modèle linéaire est un algorithme d'apprentissage supervisé qui vise à trouver la meilleure droite (ou hyperplan dans des dimensions supérieures) pour séparer les différentes classes.

Il fonctionne en ajustant un ensemble de coefficients pour chaque caractéristique du jeu de données afin de minimiser la différence entre les prédictions du modèle et les valeurs réelles. Cette minimisation est généralement effectuée en utilisant une méthode de descente de gradient pour optimiser les coefficients.

Dans notre projet, nous avons utilisé un modèle de régression logistique pour la classification des exoplanètes. Ce modèle permet de prédire la probabilité qu'une observation appartienne à une classe particulière en utilisant une fonction sigmoïde. La simplicité et l'interprétabilité du modèle linéaire en font un choix efficace pour une première approche de la classification.

3.1.1 Caractéristiques

La bibliothèque implémentée en Rust pour le modèle linéaire a été conçue pour fournir une interface efficace et performante pour l'entraînement et la prédiction sur de la classification linéaire. Elle inclut plusieurs fonctions essentielles pour initialiser, entraîner, mettre à jour les poids, prédire et gérer la mémoire du modèle. Grâce à une

interface en C, cette bibliothèque peut être utilisée facilement dans des scripts Python, permettant ainsi de combiner la performance de Rust avec la flexibilité de Python.

3.1.2 Les fonctions du modèle linéaire

3.1.2.1 La structure « LinearModel »

La structure « LinearModel » contient les différents hyperparamètres utilisés par le modèle pour l'apprentissage et la prédiction.

- **Learning Rate :** Un taux d'apprentissage de type float 64 bits, utilisé lors de la mise à jour des poids et du biais.
- **Weights :** Un vecteur de float 64bits représentant les poids associés à chaque caractéristique du neurone.
- **Bias :** Le biais du modèle, également de type float 64 bits, utilisé pour la caractéristique du neurone.
- **Activation :** Une chaîne de caractères (string) contenant le nom de la fonction d'activation à utiliser dans le modèle (par exemple, Sigmoïde, Tanh ou Relu).

3.1.2.2 La fonction « init »

Cette fonction initialise un nouveau modèle linéaire avec les paramètres spécifiés. Le but est de créer et initialiser une instance de « LinearModel » avec les paramètres d'apprentissage et la fonction d'activation choisis. Elle retourne par la suite une instance de type « LinearModel »

- **learning_rate** : Taux d'apprentissage qui est un float 64 bits.
- **weights :** Vecteur de 64 bits contenant les poids initiaux.
- **bias :** Biais initial de type float 64 bits.
- **activation :** Fonction d'activation de type string.

3.1.2.3 La fonction « train »

Cette fonction entraîne le modèle sur les données d'entrée avec les hyperparamètres spécifiées pour un nombre donné d'époques, c'est-à-dire de répétition. L'objectif est

d'optimiser les poids et le biais du modèle en minimisant l'erreur de prédiction à travers plusieurs itérations d'entraînement.

- **X :** Vecteur de vecteurs de float 64 bits représentant les données d'entraînement.
- **y :** Vecteur de float 64 bits représentant les cibles correspondantes.
- **epochs :** Nombre d'époques d'entraînement à effectuer de type entier.

3.1.2.4 Fonction « update_weights »

Cette fonction met à jour les poids et le biais du modèle en utilisant l'erreur calculée et le taux d'apprentissage. Le but étant d'ajuster les poids et le biais du modèle pour réduire l'erreur de prédiction et assurer une convergence du modèle.

- **inputs :** Vecteur de float 64 bits représentant les caractéristiques d'une observation.
- **error :** Erreur calculée pour l'observation de type float 64 bits.

3.1.2.5 Fonction « predict »

Cette fonction fait une prédiction pour une observation donnée en utilisant les poids et le biais actuels du modèle. Cela permet de calculer la sortie du modèle pour une entrée spécifique en utilisant les poids et le biais actuels et ainsi afficher la prédiction.

- **inputs :** Vecteur de float 64 bits représentant les caractéristiques d'une observation.

3.1.2.6 Fonction « LM_init »

Cette fonction initialise un le modèle via une interface C, en prenant en entrée des pointeurs vers les paramètres. Il prend notamment un pointeur de vecteur « weights_ptr » et sa taille « weights_len » dans le but de reconstruire le tableau initiale

- **learning_rate :** Taux d'apprentissage de type float 64 bits.
- **weights_ptr :** Pointeur vers les poids de type pointeur de float 64 bits (*const c_double).
- **weights_len :** Longueur du vecteur de poids de type entier.
- **bias :** Biais de type float 64 bits.

- **activation :** Fonction d'activation de type string.

3.1.2.7 Fonction « LM_train »

Cette fonction entraîne le modèle linéaire via une interface C. Tout comme la fonction précédente, il se sert des pointeurs des vecteurs et de leurs tailles, dans les paramètres de la fonction, pour reconstruire les vecteurs avant d'appeler la fonction « train »

- **model :** Pointeur vers le modèle « LinearModel ».
- **X :** Pointeur de float 64 bits vers les données d'entraînement
- **y :** Pointeur de float 64 bits vers les cibles d'entraînement
- **n_samples :** Nombre d'échantillons d'entraînement de type entier.
- **n_features :** Nombre de caractéristiques par échantillon de type entier.
- **epochs :** Nombre d'époques d'entraînement de type entier.

3.1.2.8 Fonction « LM_predict »

Cette fonction fait des prédictions via une interface C.

- **model :** Pointeur vers le modèle « LinearModel ».
- **x :** Pointeur de float 64 bits vers les données d'entraînement
- **n_samples :** Nombre d'échantillons d'entrée de type entier.
- **n_features :** Nombre de caractéristiques par échantillon de type entier.
- **predictions :** Pointeur de float 64 bits vers l'emplacement où les prédictions seront stockées

3.1.2.9 Fonction « LM_free »

Cette fonction libère la mémoire allouée pour le modèle linéaire via une interface C afin de prévenir les fuites de mémoire.

- **model :** Pointeur vers le modèle « LinearModel ».

3.1.3 Résultat

Tester notre modèle linéaire est une étape cruciale dans le processus, car cela permet de vérifier la validité et la performance du modèle sur des données réelles. Les tests permettent d'évaluer si le modèle est capable de généraliser correctement, c'est-à-dire

de faire des prédictions précises sur des données qu'il n'a jamais vues auparavant. En outre, les tests aident à identifier les problèmes de sous-apprentissage et de sur-apprentissage, à ajuster les hyperparamètres et à optimiser le modèle pour de meilleures performances.

3.1.3.1 Cas de test simple

Dans un premier temps nous avons testé notre modèle avec des cas de test simples afin de vérifier sa viabilité. En effet, si le modèle n'arrive pas à converger correctement sur des cas de test simples, cela signifie qu'il n'est pas fonctionnel et qu'il ne pourra pas fonctionner correctement sur notre jeu de données. Ces tests initiaux sont essentiels pour s'assurer que les fondamentaux du modèle sont solides avant de passer à des données plus complexes.

Cas de test « Linear multiple »

Les données d'entraînement sont constituées de deux groupes de points générés aléatoirement :

- Le premier groupe de 50 points est centré autour des coordonnées [1,1]
- Le second groupe de 50 points est centré autour des coordonnées [2,2].

Les labels associés à ces points sont :

- 1.0 pour les points du premier groupe.
- −1.0 pour les points du second groupe.

Ces données sont utilisées pour entraîner le modèle à distinguer entre les deux groupes de points.

Le modèle linéaire est initialisé avec les hyperparamètres suivants :

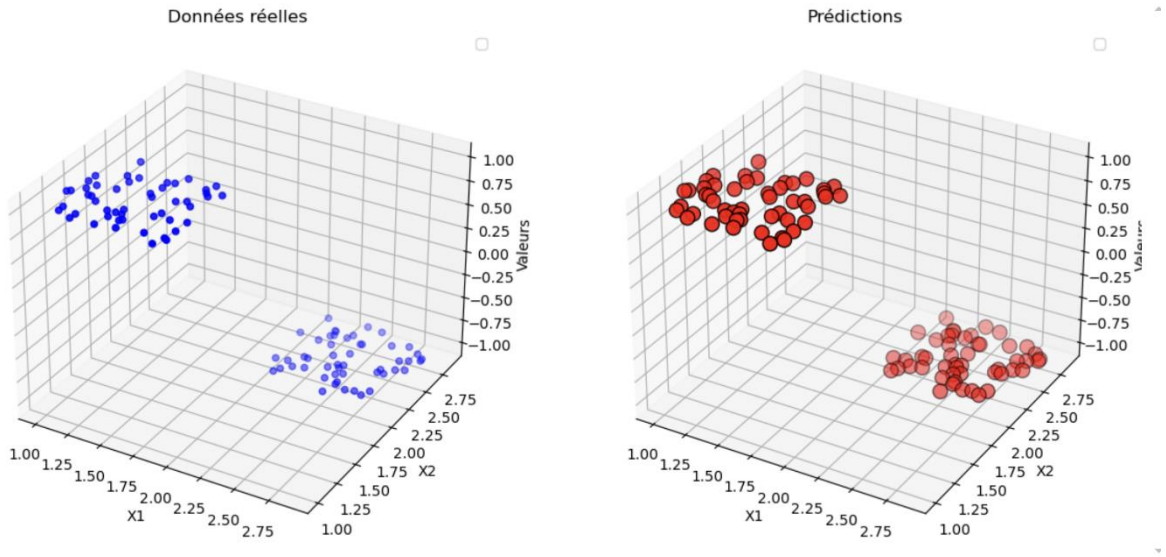
- **Learning Rate :** 0.01
- **Weights :** [0.1,−0.4,0.6,−0.24]
- **Bias :** 0.2
- **Fonction d'Activation :** Tanh
- **Nombre d'Époques :** 1,000,000

Après l'entraînement, le modèle est utilisé pour faire des prédictions sur les mêmes données d'entraînement. Les prédictions sont comparées aux valeurs cibles pour évaluer la performance du modèle. Deux graphiques 3D sont générés pour visualiser les résultats

1. **Données réelles** :

- Les points d'entraînement sont affichés en bleu.
- Les prédictions du modèle sont affichées en rouge.
- Les axes représentent les deux caractéristiques (X1 et X2) et les valeurs cibles.

Ces visualisations permettent de comparer directement les données réelles avec les prédictions du modèle et d'observer comment le modèle a appris à séparer les deux classes de points.



L'évaluation du modèle se fait en calculant l'erreur moyenne quadratique (Mean Squared Error) pour les ensembles d'entraînement et de validation. Nous avons obtenu les résultats suivants :

- **Train Error** : $3.853216883823483\times 10^{-7}$
- **Validation Error** : $1.1095267042967379 \times 10^{-14}$

Ces valeurs d'erreur extrêmement faibles indiquent que le modèle a très bien appris à séparer les deux classes de données. Une erreur très faible sur les deux ensembles (entraînement et validation) montre que le modèle généralise bien et qu'il n'y a pas de sur-apprentissage ni de sous-apprentissage. Le modèle est capable de prédire avec précision

non seulement sur les données d'entraînement mais également sur les données de validation, ce qui démontre une bonne performance globale du modèle.

Cas de test « Xor »

Dans ce cas de test les entrées sont les combinaisons binaires possibles de deux variables, représentées par les points suivants :

- [1, 0]
- [0, 1]
- [0, 0]
- [1, 1]

Les labels associés à ces points sont :

- 1.0 pour les points [1, 0] et [0, 1], représentant les cas où exactement l'une des variables est 1.
- -1.0 pour les points [0, 0] et [1, 1], représentant les cas où les deux variables sont identiques (soit toutes deux 0, soit toutes deux 1).

Ces données sont utilisées pour entraîner un modèle à distinguer entre les deux groupes de points basés sur l'opération logique XOR.

Le modèle linéaire est initialisé avec les hyperparamètres suivants :

- Learning Rate : 0.05
- Weights : [0.1, -0.4, 0.6, -0.24]
- Bias : 0.2
- Fonction d'Activation : Tanh
- Nombre d'Époques : 100,000

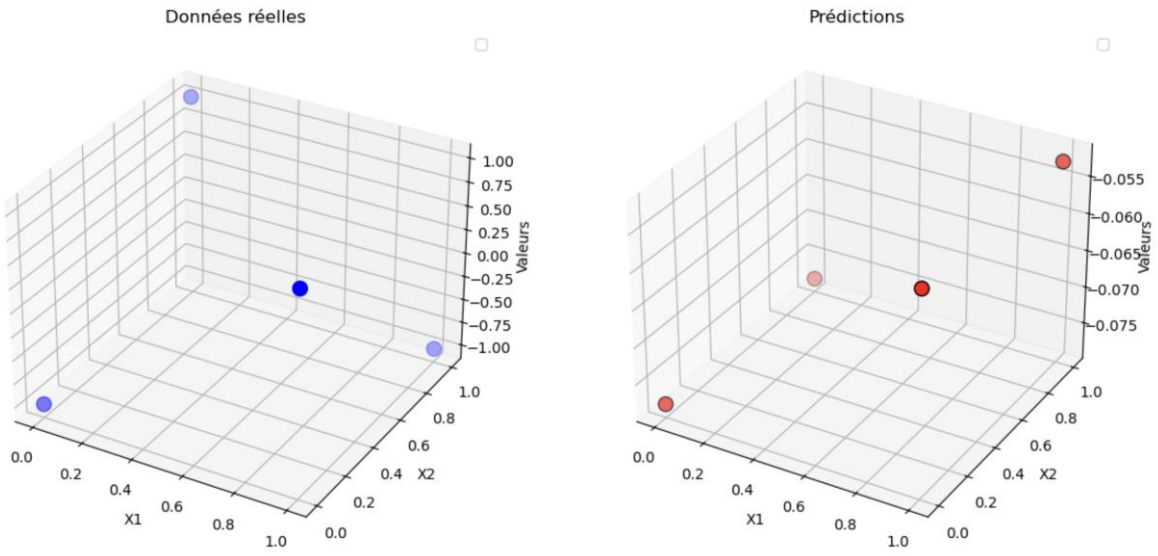
Après l'entraînement, le modèle est utilisé pour faire des prédictions sur les mêmes données d'entraînement. Les prédictions obtenues sont :

- [-0.05258049, -0.07811046, -0.07811046, -0.05258049]

Ces prédictions sont comparées aux valeurs cibles pour évaluer la performance du modèle. Deux graphiques 3D sont générés pour visualiser les résultats :

- Les points d'entraînement sont affichés en bleu.

- Les prédictions du modèle sont affichées en rouge.
- Les axes représentent les deux caractéristiques (X1 et X2) et les valeurs cibles.



Les visualisations ainsi que les prédictions permettent d'observer que les prédictions sont proches de 0, indiquant que le modèle n'a pas bien appris à distinguer les classes, probablement en raison de la nature non linéaire du problème XOR, qui n'est adapté à un modèle linéaire simple avec une seule couche.

3.2 Modèle linéaire de régression

En Machine Learning, les modèles de régression linéaire sont couramment utilisés pour établir une relation entre les caractéristiques d'un jeu de données et une variable cible continue. Ce type de modèle cherche à ajuster un ensemble de coefficients pour chaque caractéristique afin de minimiser l'erreur entre les prédictions du modèle et les valeurs réelles. Dans notre projet, nous avons utilisé un modèle de régression linéaire pour prédire des valeurs continues, en utilisant la pseudo-inverse pour calculer les poids optimaux.

3.2.1 Les fonctions

3.2.1.1 La structure « LinearRegressionModel »

La structure « LinearRegressionModel » contient les hyperparamètres nécessaires à l'apprentissage et à la prédiction.

- **weights :** Un DVector de float 64 bits représentant les poids de chaque caractéristique du modèle.

3.2.1.2 La fonction « new »

Cette fonction initialise un nouveau modèle de régression linéaire avec des poids nuls, créant ainsi une instance de « LinearRegressionModel » avec des poids initialisés à zéro.

3.2.1.3 La fonction « train »

La fonction train permet d'entraîner le modèle en utilisant la fonction « pseudo-inverse » de la matrice des caractéristiques pour calculer les poids optimaux.

- **X :** Pointeur d'un DVector de float 64 bits représentant les données
- **Y :** Pointeur d'un DVector de float 64 bits représentant les labels

3.2.1.4 La fonction « predict »

La fonction « predict » permet de faire des prédictions pour une matrice de nouvelles observations en utilisant les poids actuels du modèle. Elle calcule la sortie du modèle pour une entrée donnée.

- **X :** Pointeur d'un DVector de float 64 bits représentant les données

3.2.1.5 La fonction pseudo_inverse

Cette fonction calcule la pseudo-inverse d'une matrice via la décomposition en valeurs singulières Cela permet de résoudre des systèmes d'équations même lorsque la matrice est non carrée ou singulière.

- **matrix :** Pointeur d'un DVector de float 64 bits représentant les données

3.2.2 Les résultats

Cas de test simple

Les données d'entraînement sont constituées de trois points :

- [1, 1], [2, 2] ,[3, 3]

Les labels associés à ces points sont :

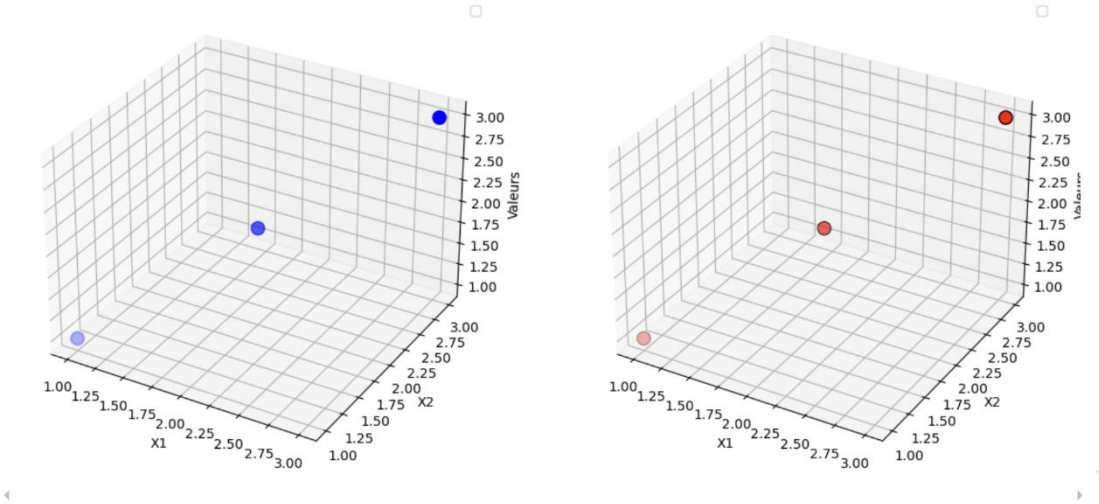
- 1 pour [1, 1]
- 2 pour [2, 2]
- 3 pour [3, 3]

Ces données sont utilisées pour entraîner un modèle de régression linéaire à prédire une valeur cible continue basée sur deux caractéristiques.

Le modèle de régression linéaire est initialisé sans hyperparamètres spécifiques à ajuster dans ce cas. Le modèle va simplement apprendre les poids et le biais à partir des données d'entraînement.

Ces prédictions sont comparées aux valeurs cibles pour évaluer la performance du modèle. Deux graphiques 3D sont générés pour visualiser les résultats :

- Les points d'entraînement sont affichés en bleu.
- Les prédictions du modèle sont affichées en rouge.
- Les axes représentent les deux caractéristiques (X1 et X2) et les valeurs cibles.



Les résultats montrent que le modèle a correctement appris à prédire les valeurs cibles pour les points d'entraînement, ce qui est attendu pour un modèle de régression linéaire simple avec des données linéaires.

3.3 Le Perceptron Multi-Couches

Le perceptron multicouche est une architecture de réseau de neurones artificiels composée de plusieurs couches de neurones. Chaque couche de neurones est connectée à la couche suivante, permettant au modèle de capturer des relations complexes et non linéaires dans les données. Les PMC sont utilisés pour résoudre des problèmes de classification et de régression complexes grâce à leur capacité à modéliser des relations non linéaires entre les caractéristiques d'entrée et la variable cible.

3.3.1 Caractéristiques

Le perceptron multicouche se compose de trois types de couches :

- **Couche d'entrée** : Les entrée de notre jeu de données.
- **Couches cachées** : Effectuent des transformations non linéaires sur les données d'entrée. Chaque neurone dans une couche cachée applique une fonction d'activation, telle que Sigmoid, Tanh ou relu, pour introduire de la non-linéarité dans le modèle.
- **Couche de sortie** : Génère les prédictions finales du modèle. Dans une tâche de classification, elle utilise généralement une fonction d'activation comme Tanh ou Softmax pour produire des probabilités pour chaque classe.

L'entraînement d'un PMC se fait par rétropropagation de l'erreur. Ce processus comprend les étapes suivantes :

- **Propagation avant (forward propagation)** : Les données d'entrée passent à travers les couches du réseau, et chaque neurone calcule sa sortie en appliquant une fonction d'activation.
- **Calcul de l'erreur** : L'erreur entre les prédictions du modèle et les valeurs réelles est calculée.
- **Rétropropagation de l'erreur (backpropagation)** : L'erreur est propagée en arrière à travers le réseau, et les gradients des poids sont calculés.

- **Mise à jour des poids** : Les poids du réseau sont ajustés en utilisant les gradients et le taux d'apprentissage pour minimiser l'erreur.

Dans notre modèle, nous avons intégré des fonctionnalités supplémentaires qui n'étaient pas initialement demandées, telles que l'intégration de la fonction d'activation Softmax en couche de sortie sur la classification multi-classe et l'utilisation du mini-batch gradient au lieu du stochastique gradient. Ces améliorations apportent des avantages significatifs au modèle.

La fonction d'activation Softmax est utilisée dans la couche de sortie pour les tâches de classification multi-classes. Elle transforme les scores de sortie du modèle en probabilités, permettant une interprétation plus facile des résultats. Le Softmax garantit que la somme des probabilités de toutes les classes est égale à 1, ce qui est particulièrement utile pour les problèmes où chaque entrée doit être assignée à une seule classe.

- **Avantage** : En utilisant le Softmax, le modèle peut fournir des probabilités pour chaque classe, facilitant ainsi l'interprétation et la prise de décision basée sur la confiance des prédictions.

De plus nous avons mis en place un Mini-Batch Gradient Descent, qui contrairement au Stochastique Gradient Descent, qui met à jour les poids du modèle après chaque échantillon, le Mini-Batch Gradient Descent met à jour les poids après avoir calculé le gradient sur un petit lot (batch) d'échantillons. Cette approche combine certains des avantages du Batch Gradient Descent (meilleure estimation du gradient) et du Stochastique Gradient Descent (meilleure généralisation).

- **Avantage** : Le Mini-Batch Gradient Descent réduit la variance des mises à jour des gradients, ce qui conduit à des trajectoires d'optimisation plus stables et plus rapides. De plus il à l'avantage de permettre si nous le souhaitons de faire de la Stochastique Gradient Descent ou un Batch Gradient Descent simplement en modifiant un hyperparamètre

Nous avons également intégré TensorBoard pour la visualisation des métriques d'entraînement, ce qui permet de suivre et d'analyser les performances du modèle au fil du temps.

- **Fonctionnement** : TensorBoard est une suite d'outils de visualisation qui permet de visualiser différents aspects de l'entraînement de modèles de

machine learning. Nous avons utilisé TensorBoard pour suivre des métriques telles que la perte d'entraînement, l'accuracy, et les gradients des poids.

- **Avantage** : La visualisation avec TensorBoard facilite l'analyse des performances du modèle et la détection des problèmes potentiels. Elle permet de suivre les progrès de l'entraînement en temps réel et d'ajuster les hyperparamètres en conséquence pour optimiser le modèle.

3.3.2 Les fonctions

3.3.2.1 Fonction « init »

Cette fonction initialise un nouveau modèle PMC avec les paramètres spécifiés. Il initialise une instance de « MlpModel » avec les tailles des neurones par couche et le taux d'apprentissage choisis.

- **neurons_size :** Vecteur contenant la taille des neurones pour chaque couche.
- **learning_rate :** Taux d'apprentissage de type float 64 bits.

3.3.2.2 Fonction « new » fichier neural_matrix.rs

Cette fonction initialise une nouvelle couche neuronale avec des poids aléatoires et des biais. Cela permet de créer et d'initialiser une instance de « NeuralMatrix » avec des poids et des biais aléatoires pour la couche neuronale.

- **input_size :** Taille des entrées de la couche de type entier.
- **output_size :** Taille des sorties de la couche de type entier.

3.3.2.3 Fonction « propagate »

Cette fonction effectue la propagation des valeurs d'entrée à travers le réseau de neurones. Calculer les sorties du modèle en passant les données d'entrée à travers toutes les couches du réseau.

- **inputs :** Slice de f64 représentant les caractéristiques d'entrée.
- **is_classification :** Booléen indiquant si la tâche est une classification.

3.3.2.4 Fonction forward

Cette fonction effectue une propagation avant (forward) des valeurs d'entrée à travers toutes les couches du réseau, et renvoie les activations de chaque couche. Il calcul et renvois les activations de chaque couche du réseau pour une entrée donnée.

- **inputs :** Pointeur de float 64 bits représentant les caractéristiques d'entrée.
- **is_classification :** Booléen indiquant si la tâche est une classification.

3.3.2.5 Fonction backward

Cette fonction effectue la rétropropagation de l'erreur à travers le réseau de neurones, ajustant les poids et les biais en conséquence. Cela permet de mettre à jour les poids et les biais du réseau en minimisant l'erreur de prédiction par rétropropagation.

- **activations :** Pointeur de vecteur de vecteur de float 64 bits représentant les activations de chaque couche.
- **target :** Pointeur de vecteur de float 64 bits représentant les labels de sortie.
- **is_classification :** Booléen indiquant si la tâche est une classification.
- **weight_gradients :** Pointeur mutable d'un triple vecteur de efloat 64 bits qui permet d'accumuler les gradients des poids, important pour le mini-batch.
- **bias_gradients :** Pointeur vecteur de vecteur de float 64 bits pour accumuler les gradients des biais.
- **is_last_chunk :** Booléen indiquant si c'est le dernier lot de la rétropropagation.
- **batch_len :** Longueur du lot de données de type float 64 bits.

3.3.2.6 Fonction save

Cette fonction sauvegarde le modèle et ses poids ainsi que ses gradients et ses biais actuel dans un fichier json. Le but est de sauvegarder l'état actuel du modèle dans un fichier pour une récupération ultérieure.

- **filename :** Référence à une chaîne de caractères représentant le nom du fichier.

3.3.2.7 Fonction load

Cette fonction charge un modèle à partir d'un fichier et ainsi reprendre un modèle déjà pré-entrainé.

- **filename :** Référence à une chaîne de caractères représentant le nom du fichier.

3.3.2.8 Fonction train

Cette fonction entraîne le modèle PMC sur les données d'entrée et les cibles spécifiées pour un nombre donné d'époques. Il optimise les poids et les biais du modèle en minimisant l'erreur de prédiction à travers plusieurs itérations d'entraînement.

- **x :** Pointeur de vecteur de vecteur de float 64 bits représentant les données d'entraînement.
- **y :** Pointeur de vecteur de vecteur de float 64 bits représentant les cibles correspondantes.
- **epochs :** Nombre d'époques d'entraînement de type entier.
- **batch_size :** Taille du batch de données de type entier .
- **is_classification :** Booléen indiquant si la tâche est une classification.
- **callback :** Fonction de rappel pour le suivi de la progression et permettre une barre de progression dans le notebook jupyter.
- **callback_interval :** Intervalle de rappel de type entier.
- **checkpoint_enable :** Booléen indiquant si les points de contrôle sont activés.
- **checkpoint_interval :** Intervalle de point de contrôle de type entier.
- **log_enable :** Booléen indiquant si la journalisation est activée à travers tensorboard.
- **tag :** Référence à une chaîne de caractères représentant le nom pour le fichier de journalisation de tensorboard.

3.3.2.9 Fonction predict

Cette fonction fait une prédiction pour une observation donnée en utilisant les poids et le biais actuels du modèle.

- **inputs :** Pointeur de float 64 bits.
- **is_classification :** Booléen indiquant si la tâche est une classification.

3.3.3 Résultat

3.3.3.1 Cas de test simple

Comme les modèles précédents nous avons testé notre modèle avec des cas de test simples pour vérifier sa viabilité.

Le « Xor »

Dans ce cas de test, nous reprenons le cas de test du Xor que nous avons essayé avec le modèle linéaire, les entrées sont donc des combinaisons binaires possibles de deux variables, représentées par les points suivants :

- [1, 0]
- [0, 1]
- [0, 0]
- [1, 1]

Les labels associés à ces points sont :

- 1.0 pour les points [1, 0] et [0, 1], représentant les cas où exactement l'une des variables est 1.
- -1.0 pour les points [0, 0] et [1, 1], représentant les cas où les deux variables sont identiques (soit toutes deux 0, soit toutes deux 1).

Le modèle est initialisé avec les hyperparamètres suivants :

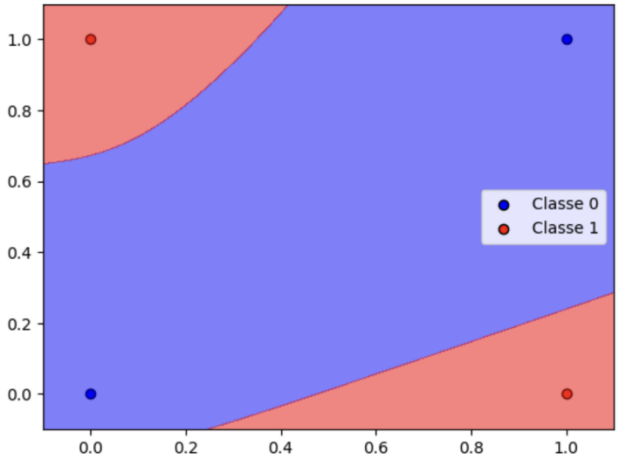
- Learning Rate : 0.01
- Neuronnes : [2, 3, 1]
- Batch size : 1 (ce qui équivaut à la Stochastique Gradient)
- Classification : True
- Nombre d'Époques : 1,000,000

Après l'entraînement, le modèle est utilisé pour faire des prédictions sur les mêmes données d'entraînement. Les prédictions obtenues sont :

- [-0.99984962, 0.99984726, 0.99985508, -0.99985493]

Ces prédictions sont comparées aux valeurs cibles pour évaluer la performance du modèle.

- Les points d'entraînement sont affichés en bleu ou en rouge selon la classe.
- Les prédictions du modèle sont affichées par une couleur de background en rouge ou bleu selon la classe



Les résultats montrent que, contrairement au modèle linéaire, le Perceptron Multi Couch a correctement appris à prédire les valeurs cibles pour les points d'entraînement.

Les cas de test multiclass

Dans ce cas de test, nous avons testé des entrées multi-classes, les entrées sont donc des combinaisons de trois variables

Contrairement à d'habitude les labels suivants sont compris entre 0.0 et 1.0, en effet maintenant que nous sommes en multi-classe nous n'utilisons plus Tanh en couche de sortie mais la Softmax. Les labels associés à ces points sont donc :

- Pour la classe 1 les points [1.0, 0.0, 0.0]
- Pour la classe 2 les points [0.0, 1.0, 0.0]
- Pour la classe 3 les points [0.0, 0.0, 1.0]

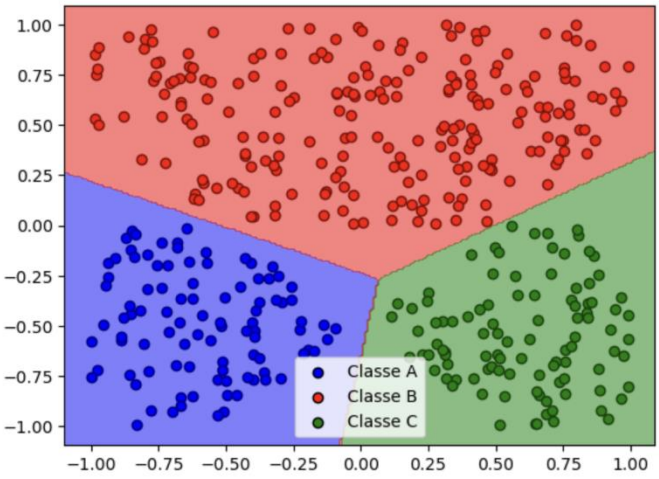
Le modèle est initialisé avec les hyperparamètres suivants :

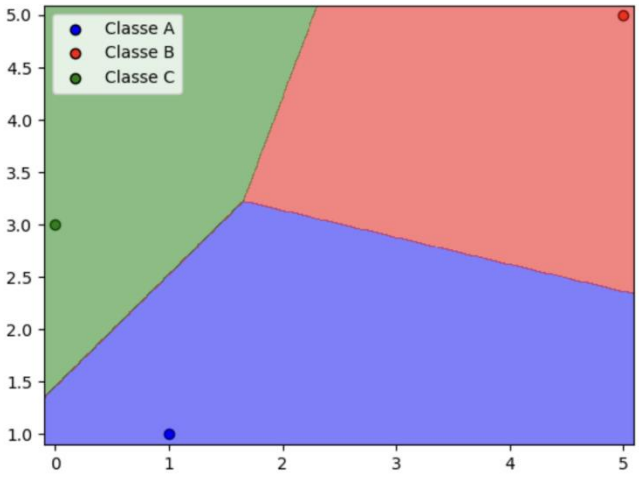
- Learning Rate : 0.01
- Neuronnes : [3, 3] et [418, 3]

- Batch size : 1 (ce qui équivaut à la Stochastique Gradient)
- Classification : True
- Nombre d'Époques : 1,000,000

Tout comme les autres cas ces prédictions sont comparées aux valeurs cibles pour évaluer la performance du modèle.

- Les points d'entraînement sont affichés en bleu ou en rouge ou vert selon la classe.
- Les prédictions du modèle sont affichées par une couleur de background en rouge ou bleu ou vert selon la classe





Les résultats montrent que le Perceptron Multi Couch a correctement appris à prédire les valeurs cibles pour les points d'entraînement.

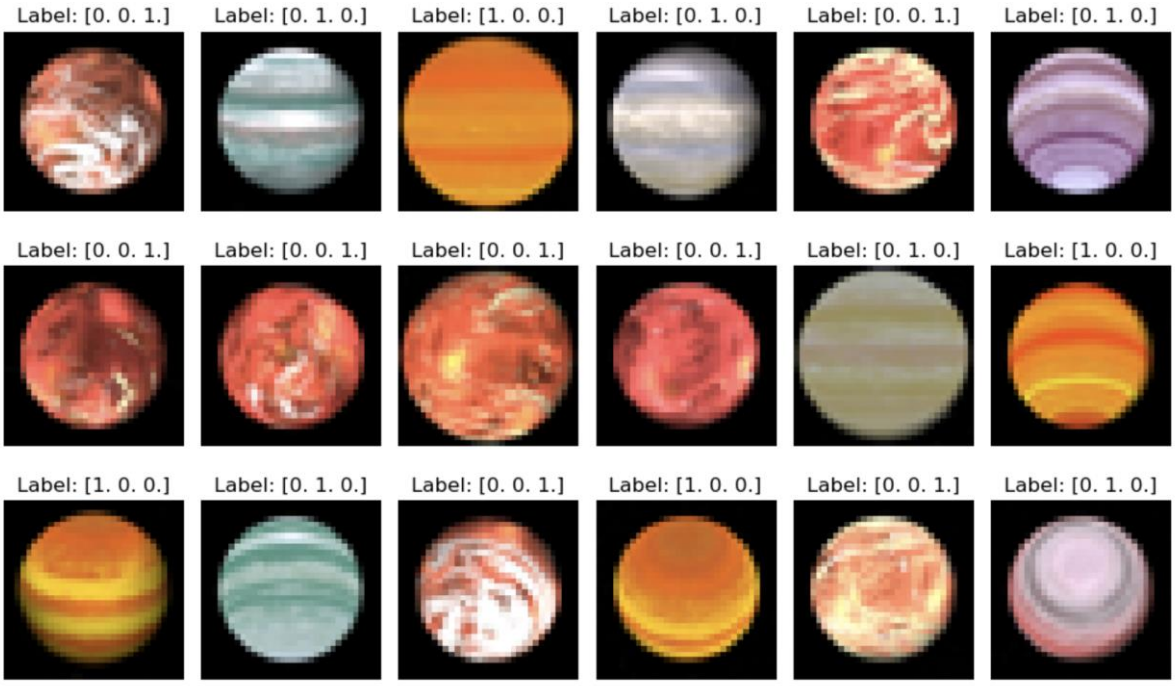
3.3.3.2 Entrainement sur le jeu de données

Après avoir testé notre modèle sur des cas de test simples, nous avons appliqué notre jeu de données directement à notre modèle

Les images sont chargées depuis trois dossiers différents, chacun correspondant à une classe spécifique d'exoplanètes (gas giant, neptune-like, super earth). Par la suite, les images sont converties en RGB et redimensionnées à une taille de 32x32 pixels puis aplaties en vecteurs 1D pour être utilisées comme entrée du modèle.

Chaque image est associée à un label correspondant à sa classe :

- [1.0, 0.0, 0.0] pour les gas giants
- [0.0, 1.0, 0.0] pour les neptune-like
- [0.0, 0.0, 1.0] pour les super earth.



Les données sont divisées en ensembles d'entraînement et de test dans une proportion de 80/20, afin d'évaluer la performance du modèle sur des données non vues pendant l'entraînement.

Afin d'améliorer les performances du modèle, nous avons normalisées les images en soustrayant la moyenne et en divisant par l'écart-type des valeurs des pixels de l'ensemble d'entraînement.

Entrainement performant numéro 1

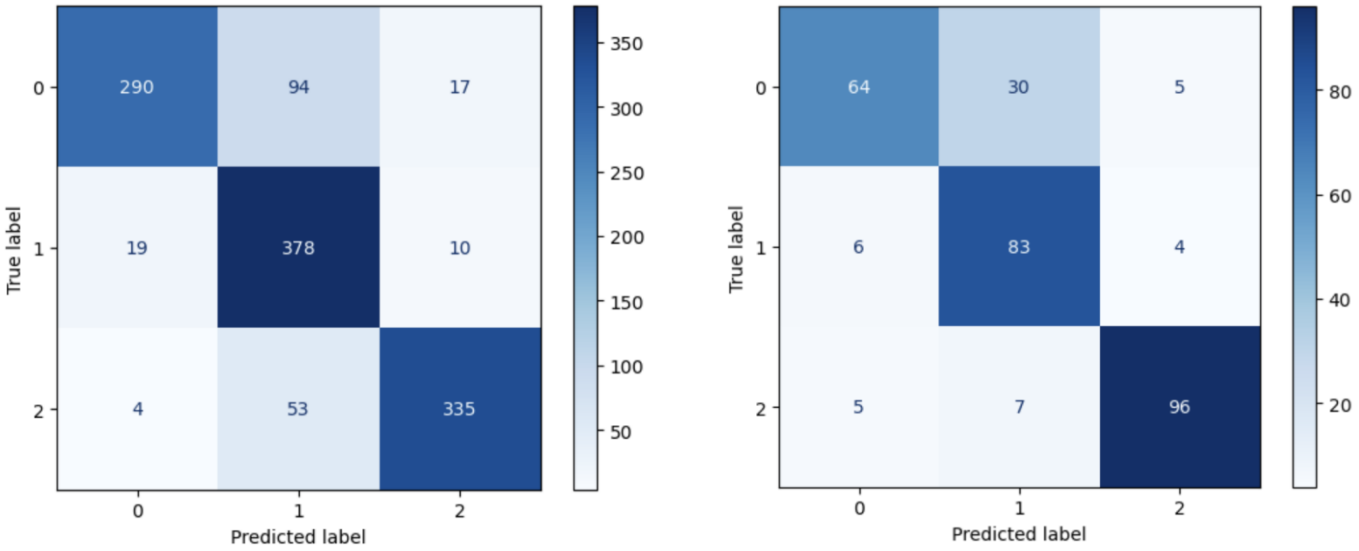
Dans ce cas de test le modèle MLP est initialisé avec les hyperparamètres suivants :

- **Learning Rate :** 0.005
- **Neurones :** [3072, 32, 16, 3]
- **Nombre d'Époques :** 1000
- **Taille du Batch :** 32

Après l'entraînement, nous avons obtenu une accuracy de 78,83% pour les données d'entraînement et de 76,67% pour les données de test, ce qui indique une bonne performance générale du modèle.

Nous avons utilisé une matrice de confusion est utilisée pour évaluer la précision du modèle dans la classification des différentes classes d'exoplanètes. Elle permet de visualiser les performances de prédiction du modèle en montrant les vrais positifs, faux positifs, vrais négatifs et faux négatifs pour chaque classe.

Grâce à cette matrice de confusion, nous remarquons un phénomène très intéressant, le modèle semble avoir du mal à différencier deux classes, ce qui peut laisser supposer que potentiellement, le jeu de données contient des images qui se ressemble grandement.



Cependant le modèle a bien convergé et a démontré une capacité à classer correctement les exoplanètes avec une précision satisfaisante sur l'ensemble des données. Ces résultats montrent que le modèle est capable de généraliser relativement bien sur des données non vues. L'utilisation de la normalisation des données et la structuration en batchs ont permis une meilleure stabilité et une convergence plus rapide du modèle.

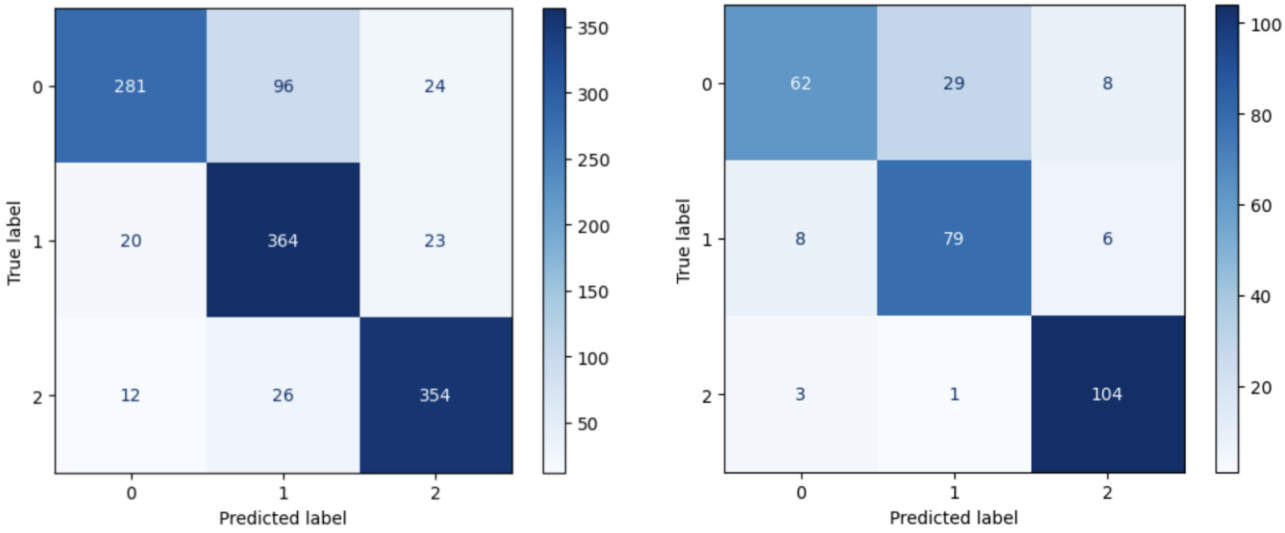
Entrainement performant numéro 2

Dans ce cas de test le modèle MLP est initialisé avec les hyperparamètres suivants :

- **Learning Rate :** 0.005
- **Neurones :** [3072,16, 3]
- **Nombre d'Époques :** 1000
- **Taille du Batch :** 32

Après l'entraînement, nous avons obtenu une accuracy de 83.25% pour les données d'entraînement et de 81.67% pour les données de test, ce qui indique une bonne performance générale du modèle.

Grâce à la matrice de confusion, nous remarquons encore une fois le même phénomène, ce qui confirme l'hypothèse précédente.



Néanmoins le modèle a encore mieux convergé avec une précision très satisfaisante sur l'ensemble des données

3.4 Le Radial Basis Fonction

Le réseau de neurones à base de fonctions radiales (RBF) est un type spécifique de réseau de neurones conçu pour capturer les caractéristiques non linéaires des données. Contrairement aux réseaux de neurones traditionnels, les réseaux RBF utilisent des fonctions d'activation

radiale pour traiter les entrées, ce qui les rend particulièrement efficaces pour les problèmes de régression et de classification.

3.4.1 Caractéristiques

Le RBF se compose de trois couches : une couche d'entrée, une couche cachée de neurones à fonctions radiales (généralement des gaussiennes), et une couche de sortie. Les neurones de la couche cachée sont activés en fonction de la distance entre l'entrée et un centre prédéfini, ce qui permet de modéliser des transformations complexes des données.

L'entraînement d'un RBF implique deux étapes principales : déterminer les centres des fonctions radiales en utilisant des algorithmes comme k-means, et ajuster les poids de la couche de sortie via des techniques de régression. Ce double processus permet au RBF de s'adapter efficacement aux données et de capturer des patterns non linéaires. Grâce à cette capacité,

3.4.2 Fonctions

3.4.2.1 Structure « RBFModel »

La structure RBFModel comprend plusieurs hyperparamètres essentiels qui seront par la suite utilisé par les différentes fonctions :

- **centers:** Un vecteur de points représentant les centres des noyaux RBF déterminés par k-means.
- **gamma:** Un coefficient déterminant l'influence de chaque noyau RBF.
- **weights:** Un vecteur de poids associés à chaque noyau RBF.

3.4.2.2 Fonction « init »

La fonction initialise un nouveau modèle RBF avec les paramètres spécifiés. Elle crée une instance de RBFModel en utilisant les données d'entraînement fournies.

- **points:** Vecteur de points représentant les données d'entraînement.
- **y_train:** Vecteur de cibles associées aux points d'entraînement.
- **k:** Nombre de noyaux RBF.
- **max_iter:** Nombre maximal d'itérations pour l'algorithme de k-means.

- **gamma:** Coefficient déterminant l'influence de chaque noyau RBF.

3.4.2.3 Fonction « train_rbf_classification »

Cette fonction entraîne le modèle RBF pour la classification en ajustant les poids des noyaux pour minimiser l'erreur de prédiction.

- **x_train:** Vecteur de points représentant les données d'entraînement.
- **y_train:** Vecteur de cibles associées aux points d'entraînement.
- **k:** Nombre de noyaux RBF.
- **max_iter:** Nombre maximal d'itérations pour l'algorithme de k-means.
- **gamma:** Coefficient déterminant l'influence de chaque noyau RBF.

3.4.2.4 Fonction « train_rbf_regression »

Similaire à la fonction de classification, cette fonction entraîne le modèle RBF pour la régression en ajustant les poids pour minimiser l'erreur de prédiction.

- **x_train:** Vecteur de points représentant les données d'entraînement.
- **y_train:** Vecteur de cibles associées aux points d'entraînement.
- **k:** Nombre de noyaux RBF.
- **max_iter:** Nombre maximal d'itérations pour l'algorithme de k-means.
- **gamma:** Coefficient déterminant l'influence de chaque noyau RBF.

3.4.2.5 Fonction « predict »

Cette fonction fait des prédictions pour de nouvelles observations en utilisant les poids et les noyaux RBF actuels du modèle.

- **x_test:** Vecteur de points représentant les données de test.

3.4.3 Résultat

Afin de vérifier la validité et les performances de notre RBF, tester le modèle est essentiel. Les tests évaluent la capacité du modèle à généraliser et à faire des prédictions précises sur des

données non vues auparavant, en identifiant les problèmes de sous-apprentissage et de sur-apprentissage.

3.4.3.1 Cas de Test Simple

Nous avons, tout comme les modèles précédents, fait des tests simples pour vérifier la fonctionnalité de base du modèle. Ces tests initiaux sont cruciaux pour s'assurer que les fondations du modèle sont solides.

Cas de Test "RBF Regression "

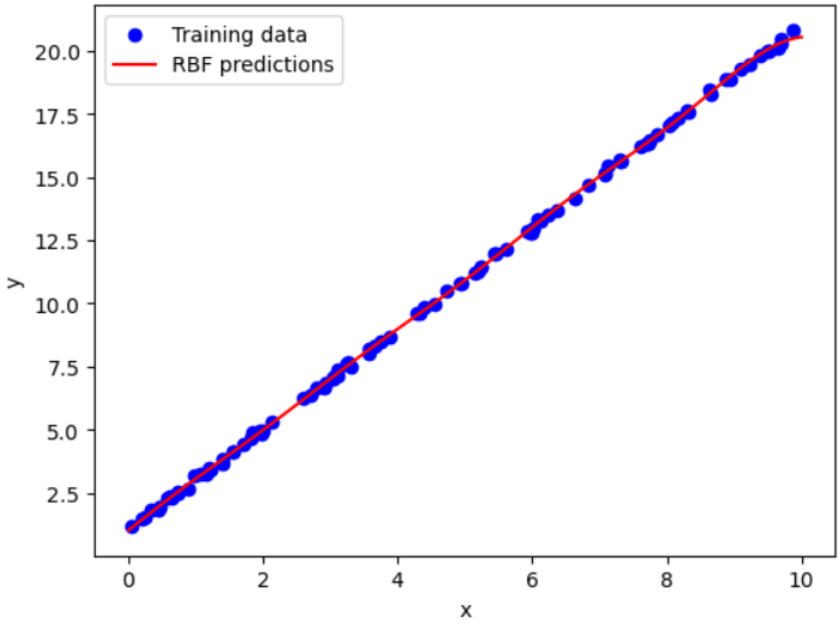
Dans ce cas de test, nous avons testé un ensemble de points linéaire afin valider la régression du modèle.

Le modèle est initialisé avec les hyperparamètres suivants :

- **k :** 6
- **gamma :** 0.1
- **nombre d' époques :** 1,000,000

Les prédictions sont comparées et représenté graphiquement pour évaluer la performance du modèle

- Les points bleus représentent les données d'entraînement. Chaque point bleu correspond à une paire (x, y) des données que vous avez utilisées pour entraîner votre modèle RBF.
- La ligne rouge représente les prédictions du modèle RBF. Elle montre la relation que le modèle a appris à partir des données d'entraînement



4. L'application

Concernant l'application, nous avons été très ambitieux en décidant d'implémenter une interface web complète permettant de manipuler nos modèles de Machine Learning en « blueprint ». Cette approche offre une représentation visuelle et intuitive des différentes étapes et composants de nos modèles, facilitant ainsi leur configuration et leur gestion.

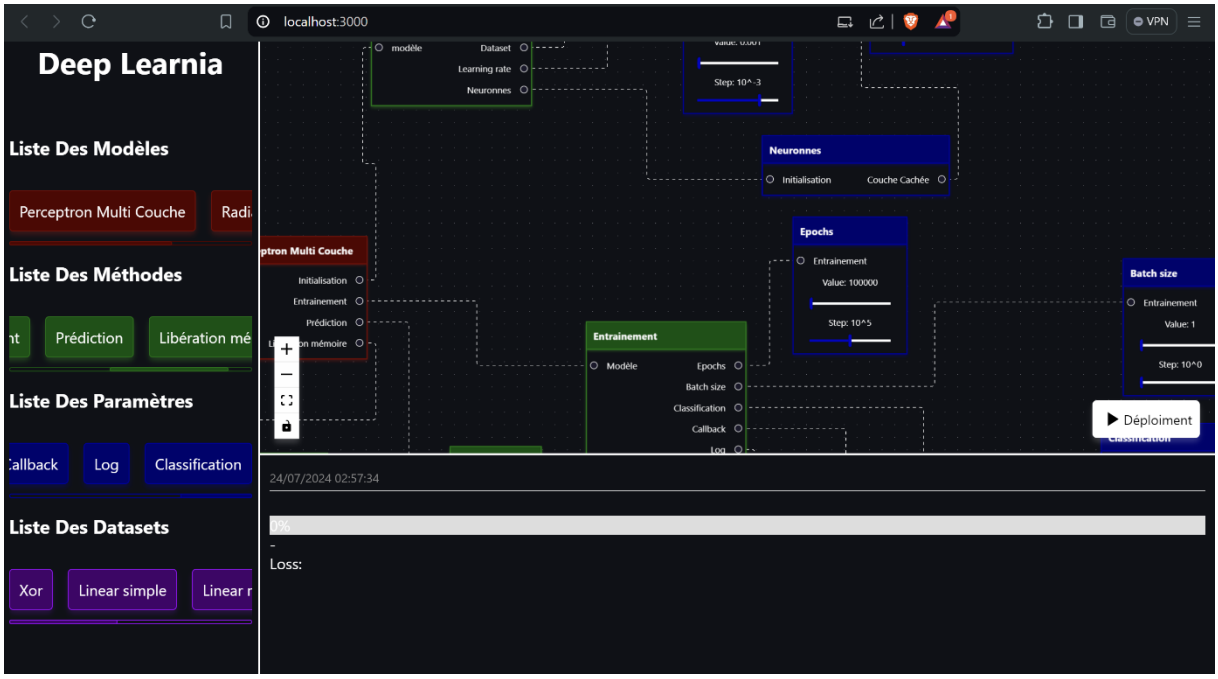
4.1 Technologie Utilisée

Pour mettre en place cette application, nous avons utilisé React.js pour la partie frontend et Django pour le serveur backend.

React.js est une bibliothèque JavaScript développée par Facebook, utilisée pour construire des interfaces utilisateur. Elle permet de créer des composants réutilisables et interactifs, ce qui facilite le développement et la maintenance de grandes applications web. Dans notre projet, React.js est utilisé pour construire l'interface utilisateur de l'application web, offrant une expérience fluide et réactive.

Django est un framework web Python de haut niveau qui encourage le développement rapide et la conception propre et pragmatique. Il est utilisé pour construire le backend de notre application, gérant la logique de l'application, la base de données, et l'interaction avec les modèles de machine learning.

Pour améliorer l'expérience utilisateur, nous avons intégré l'utilisation des WebSockets pour afficher les barres de progression en temps réel. Les WebSockets permettent une communication bidirectionnelle entre le client et le serveur, ce qui est essentiel pour des mises à jour en temps réel sans avoir à recharger la page.



Grâce à l'intégration de React.js et Django, ainsi qu'à l'utilisation des WebSockets pour les mises à jour en temps réel, nous avons développé une application web robuste et interactive. Cette application permet de manipuler et de visualiser facilement les modèles de machine learning, offrant ainsi un outil puissant pour les utilisateurs souhaitant explorer et analyser les données d'exoplanètes.

5. Conclusion

Notre projet visant à classifier des exoplanètes à l'aide de divers modèles de machine learning, notamment le modèle linéaire, le perceptron multicouche (PMC) et le réseau de

fonctions de base radiale (RBF), a été couronné de succès. Les résultats obtenus sur notre jeu de données ont montré des performances satisfaisantes et prometteuses pour nos modèles.

Le modèle linéaire a fourni une base solide et interprétable pour la classification, tandis que le perceptron multicouche a démontré une capacité accrue à capturer des non-linéarités grâce à ses couches cachées et à l'intégration de fonctions d'activation avancées telles que le softmax et le tanh. Le réseau de fonctions de base radiale a également montré de bonnes performances, particulièrement pour les données avec des relations non linéaires complexes.

Notre ambition d'implémenter une interface web complète a également été réalisée bien qu'elle n'est pas encore entièrement terminé elle offre une plateforme interactive et user-friendly pour manipuler et visualiser les modèles de machine learning.

Nous avons eu aussi l'ambition d'intégrer des fonctionnalités supplémentaires, telles que le Mini-Batch Gradient Descent, qui a permis une optimisation plus stable et rapide des modèles, ainsi que la visualisation des métriques d'entraînement avec TensorBoard, offrant des insights précieux sur les performances et les progrès du modèle.

En somme, ce projet nous a permis de mettre en pratique une large gamme de compétences, allant de l'implémentation de modèles de Machine Learning à la construction d'une application web sophistiquée. Nous sommes fiers des résultats obtenus et enthousiastes quant aux possibilités futures d'amélioration et d'expansion de ce travail.