Programmation avancée - Mini projet
Introduction au Deep Learning (Perceptron)

Table des matières

I – Introduction

L'objectif de ce mini projet est de mettre en pratique ce que nous avons appris durant la matière Programmation Avancée, dans la branche Deep Learning de l'Intelligence Artificielle. Il faut ainsi réaliser un réseau de neurones simple (ou Perceptron simple) qui prédit si un élève est admis ou non en fonction des notes qu'il a reçu dans différentes matières.

Dans la partie II sera présentée la création des données.
Puis dans la partie III, sera présentée l'entraînement du modèle réseau de neurones avec le dataset crée précédemment.
Ensuite, dans la partie IV sera testé le réseau de neurones dans le cas d'un dataset avec 3 features.
Enfin, dans la partie V, on teste ce dernier avec un dataset contenant 5 features.

II- Création des données

La création du dataset constitue la première étape d'une création du modèle. Cette étape est importante car c'est sur ce dernier que le modèle va s'entraîner, et ainsi les performances de ce dernier en est impacté.

Dans notre cas, nous allons créer un dataset avec **m** échantillons et **n** features. Chaque échantillon représente un étudiant. Et chaque échantillon est composé d'un ensembles de features, dans notre cas, chaque élève possède une note dans n matières différentes. Comme nous sommes dans un cas d'apprentissage supervisé, chaque échantillon de ce dataset sera associée à une étiquette (ou label) qu'on nomme **y**. Dans notre cas, chaque échantillon est une variable binaire (0 ou 1) qui correspond à l'étudiant admis (**y=0**) ou non (**y=1**).

Ainsi, nous allons en premier lieu créer un dataset avec 1000 échantillons et 2 features, qui se traduit par un dataset composé de 1000 étudiants ayant une note dans 2 matières. Les deux matières seront le **français** et les **mathématiques**.
De plus, pour rendre ce dataset plus « proche de la réalité », chaque note aura un coefficient en fonction de la matière, plus précisément les notes en mathématiques auront un coefficient de 2 et les notes en français auront un coefficient de 1.

Pour créer ce dataset, j'ai utilisé une boucle, dans laquelle je générais 2 notes aléatoires grâce à la méthode np.random.uniform qui permet d'obtenir des nombres floatant à partir d'une distribution uniforme.
Ensuite, je calcule la moyenne des notes obtenus de l'élève en tenant compte des coefficients de chaque note en fonction de sa matière. Le résultat obtenu me permet de créer le label associé à l'étudiant, qui est de 1 si la moyenne est supérieure à 10, sinon 0.

Une fois le dataset crée dont la taille est (1000,2), je divise ce dernier en 2.
Cela permet d'avoir un **dataset d'entraînement** (20% de la taille du dataset) qui va être donné au modèle durant la phase d'entraînement, et d'un **dataset de test** (20% de la taille du dataset) pour comparer les labels que le modèle a prédis par rapport aux vrais labels, une fois son entraînement terminé.

III- Création et optimisation du modèle Perceptron simple

Une fois notre dataset crée, il reste à créer un percreptron ou réseau de neurones artificiels, qui dans notre cas est composé d'un unique neurone.

J'ai choisi d'utiliser une classe pour définir ce réseau de neurones.
La première méthode **Initialisation**, permet d'initialiser le poids de chaque note, ainsi que le biais pour le neurone.
Ensuite, on définit la fonction **Cout** (LogLoss) de notre modèle, permettant de mesurer la performance de notre modèle pour un biais et un poids donné.
Vient la méthode **Gradients** qui permet de calculer les dérivées partielles de la fonction **Cout**, selon le poids et le biais.
Deux autre méthodes ont été créées, où la première, **Update,** met à jour les coefficients poids et biais de notre modèle d'après le gradient, et la deuxième, **Model**, qui permet de calculer la fonction d'activation, utilisé par le modèle pour prédire si un étudiant est admis ou non, en calculant cette probabilité d'admission. On considère qu'un étudiant est admis si la fonction d'activation retourne un chiffre supérieur ou égal à 0.5, ce qui veut dire qu'on est au minimum 50% sûr que l'élève est admis.

Enfin, on crée une autre méthode **Fit**, correspondant à la Descente de Gradients de la fonction coût. Cela permet d'optimiser les paramètres poids, biais de notre modèle au fils des itérations, en fonction du dataset qu'on lui a fournis en entrée.
En sortie, nous avons un poids et un biais optimisé, ainsi qu'un modèle efficace pour prédire si un élève est admis ou non en fonction des notes de ce dernier en français et en mathématiques.

Durant la descente de gradients, i.e l'entraînement du modèle, nous avons aussi stocké le coût de ce dernier avant chaque modification des paramètres poids et biais.
Avec cette liste de coûts, on peut tracer une courbe qui montre l'évolution des erreurs effectuées par le modèle avec un poids et un biais donné, au cours des itérations.

Pour mesurer l'efficacité de ce modèle, j'ai demandé au modèle de prédire les labels du dataset d'entraînement, puis utilisé une fonction de scklearn, **accuracy_score**, qui calcule le score de précision entre les labels prédis par le modèle et les véritables labels.
Pour un learning_rate (coefficient de descente durant la descente de gradient) de 0.1 et un nombre d'itérations maximum de 2000, j'ai obtenu un accuracy_score d'environ 0.985, ce qui veut dire que le modèle a prédit de manière exacte l'admission ou non de 98,5% des étudiants.
Ensuite, j'ai prédit les labels du dataset de tests, et j'ai obtenu un accuracy_score d'environ 98%.
Néanmoins, lorsque j'affichais l'évolution de la fonction, on pouvait constater que le cout pouvait remonter. Ce phénomène est dû au fait qu'avec le pas utilisé (ou learning_rate), on dépassait le minimum de la fonction coût, dont nous savons qu'elle est concave et qu'elle admet un minimum globale sans minimum local.
En mettant le learning_rate à 0.05, il est possible de réduire ce phénomène sans impacter grandement les performances du mdoèle.

On peut en conclure que notre modèle est bon pour prédire l'admission ou non d'un étudiant dans notre cas.

Enfin, comme nous avons de bon coefficients, on peut tracer la frontière de décision qui

permet de visualiser le taux de confiance pour chaque label du dataset, i.e selon le modèle, la probabilité pour qu'un étudiant soit admis ou non.

IV- Dataset avec 3 features

Dans cette dernière partie, nous allons entraîner et tester un modèle de perceptron simple avec non plus 2 features, mais 3 features, c'est-à dire que chaque étudiant aura une note dans 3 matières, à savoir les mathématiques (coeff 2), le français (coeff 1) et la physique (coeff 2).

On utilise la même logique que dans la partie précédente pour créer un dataset de dimensions (1000,3).
On divise le dataset en 2, dont l'un va permettre d'entraîner le modèle, et l'autre qui va permettre de tester la performance de ce dernier.
On entraîne le modèle avec cette fois un learning_rate de 0.025 et un nombre d'itérations maximum de 10000 car il y a plus de features.
On obtient un score de précision d'environ 0,975 pour le training_set et le test_set, ce qui montre qu'un modèle de réseau de neurones permet aussi de prédire de manière précise l'admission d'un étudiant avec plusieurs features.

Enfin, on représente dans un plan 3d, le dataset, ainsi que la frontière de décision.

V- Dataset avec 5 features

Dans cette dernière partie, nous allons entraîner et tester un modèle de perceptron simple avec non plus 2 features, mais 5 features, c'est-à dire que chaque étudiant aura une note dans 5 matières, à savoir les mathématiques (coeff 3), le français (coeff 2), l'histoire (coeff 2), la physique (coeff 3) et le sport (coeff 1).

On utilise la même logique que dans la partie précédente pour créer un dataset de dimensions (1000,3).
On divise le dataset en 2, dont l'un va permettre d'entraîner le modèle, et l'autre qui va permettre de tester la performance de ce dernier.
On entraîne le modèle avec cette fois un learning_rate de 0.025 et un nombre d'itérations maximum de 15000 car il y a plus de features.
On obtient un score de précision d'environ 0,99 pour le training_set et de 0,98 le test_set,

Enfin, comme on ne peut représenter le dataset du fait qu'il y a 5 features, on peut utiliser l'algorithme TSNE qui permet d'effectuer une réduction de dimension pour visualiser le dataset dans un plan en 2 dimensions.
On remarque qu'il y a une frontière, cependant, elle n'est pas parfaite. Il faudrait utiliser un algorithme type ACP qui permet de savoir quelles sont les features qui ont le plus de poids, réduire la dimension de notre dataset en éliminant ainsi les features les moins impactantes.