La Lumière du Savoir

مدرسة نور المعرفة

Année 2022-2023- Grand Oral de Spécialité

Question de Mathématique

Problématique :

Les mathématiques utilisées pour le Deep Learning sont-elles aussi complexes que l'IA est puissante ?

Auteur : Mohamed MAZEN

Introduction :

1. Qu'est-ce que l'IA

L'Intelligence Artificielle (IA) est un domaine de l'informatique qui vise à développer des machines capables de simuler certaines caractéristiques de l'intelligence humaine, comme la perception, la compréhension du langage naturel, la reconnaissance de motifs et la prise de décision. En d'autres termes, l'IA cherche à imiter la manière dont les êtres humains pensent, apprennent et résolvent des problèmes.

2. Qu’est-ce que le Machine Learning

Le Machine Learning (apprentissage automatique) est une sous-catégorie de l'IA qui permet aux ordinateurs d'apprendre et de s'améliorer à partir de l'expérience, sans être explicitement programmés pour chaque tâche. Les algorithmes de Machine Learning identifient des modèles dans les données et utilisent ces informations pour prendre des décisions, ce qui permet aux machines de s'adapter et de résoudre des problèmes de manière autonome.

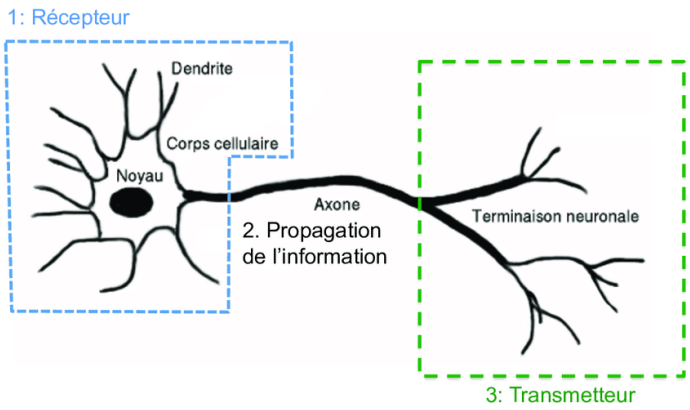
3. Qu’est-ce que le Deep Learning

Le Deep Learning est une méthode avancée d'apprentissage automatique qui utilise des réseaux de neurones artificiels pour reconnaître des motifs complexes dans les données. Il est utilisé dans des domaines tels que la reconnaissance d'images, le traitement du langage naturel et la traduction automatique. Grâce à de grandes quantités de données et de puissance de calcul, le Deep Learning a permis des avancées significatives dans des domaines tels que la reconnaissance vocale, la vision par ordinateur, la médecine et la finance.

Le liens entre les Mathématiques et le Deep Learning :

1. Les premiers réseaux de neurones

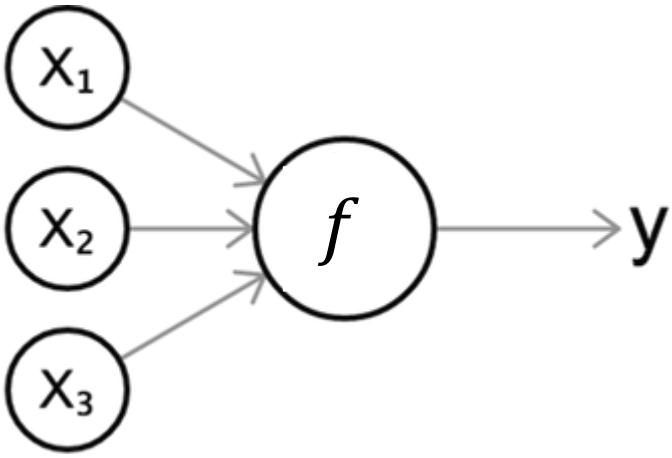
Biologiquement, un neurone est une cellule excitable connectée avec d’autres neurones et ayant pour rôle de transmettre des informations dans le système nerveux :



Au niveau de la partie du récepteur, il y a la synapse qui se trouve entre le bouton synaptique d'un neurone présynaptique et la dendrite ou le corps cellulaire d'un neurone postsynaptique. Deux types de signaux y sont réceptionnés : un signal excitateur que l’on peut traduire par « $+1$ » ou un signal inhibiteur que l’on peut traduire par « -1 ». Si le signal excitateur est supérieur au signal inhibiteur ou l’inverse, le neurone emmène un signal électrique qui passe par l’axone pour finir vers les terminaisons neuronales qui ont pour rôle de transmettre ce signal vers un autre neurone.

Ainsi, les scientifiques McCulloch et Pitts ont chercher à modéliser le neurone pour effectuer des problèmes booléens où le neurone est représenté comme une fonction de transfère avec des signaux x en entre et un signal y en sors. il y a plusieurs étapes : d’abord l’agrégation où l’on fait la somme de toute les entrées des neurones et en les multipliant par un coefficient w qui représentent l’activité

synaptique qui permet de savoir si le signal est excitateur dont $w = +1$ ou inhibiteur $w = -1$. Puis, il y a l'étape de l'activation où si le résultat y dépasse un seuil il s'active sinon il ne s'active pas.



Agrégation : $f = x_1w_1 + x_2w_2 + x_3w_3$

Activation : $\begin{cases} y = 1 \; si \; f \geq 0 \\ y = 0 \; sinon \end{cases}$

Ce neurone artificiel a été nommée plus tard *THESHOLD LOGIC UNIT* qui était conçu seulement pour traiter des entrées logiques telles que and ou or. McCulloch et Pitts ont remarqué qu'au même titre que les neurones naturels si on liait plusieurs neurones artificiels on pourrait normalement résoudre n'importe quel problème de type booléen ($True$ / $False$) malheureusement, le modèle n'avait pas d'algorithme d'apprentissage donc on ne pouvait pas s'en servir pour des applications du monde réel.

2. Le Perceptron de Frank Rosenblatt

En 1957 Frank Rosenblatt invente le Perceptron, un algorithme d'apprentissage basé sur le modèle de neurone de McCulloch et Pitts, à la seule différence qu'en plus le Perceptron possédait aussi un algorithme d'apprentissage pour trouver les valeur w. Frank Rosenblatt se basa sur la théorie de Hebb où il postule que « lorsque 2 neurones biologiques sont excités conjointement alors il renforce leurs liens synaptiques ». Sur ce modèle, il entraine un neurone artificiel sur des données de référence (X, y) pour que celui-ci renforce ses paramètres W à chaque fois qu'une entrée X est activée en même temps que la sorties y présente dans ces données ;

$$W = W + \alpha(y_{true} - y)X$$

- y_{true} : sorties de référence
- y : sorties produit par le neurone
- X : entrées du neurone
- a : vitesse d'apprentissage

Plus précisément, on peut apporter les développements suivants :

Le perceptron est un modèle mathématique utilisé dans le domaine du Deep Learning et de l'apprentissage automatique (machine learning). Il est considéré comme le bloc de construction fondamental des réseaux de neurones artificiels.

Le perceptron est basé sur le concept des neurones biologiques et vise à simuler leur fonctionnement dans un modèle mathématique. Il est utilisé pour résoudre des problèmes de classification binaire, c'est-à-dire des problèmes où il faut séparer des données en deux catégories distinctes.

Voici comment fonctionne un perceptron :

1. Entrées : Le perceptron reçoit un ensemble d'entrées, représenté par un vecteur de valeurs. Chaque valeur d'entrée est pondérée par un poids correspondant à l'importance de cette entrée pour la tâche de classification.
2. Sommation pondérée : Les entrées pondérées sont sommées pour obtenir une valeur unique, appelée somme pondérée. Cette somme pondérée représente une combinaison linéaire des entrées et des poids.
3. Fonction d'activation : La somme pondérée est ensuite passée à une fonction d'activation, qui introduit une non-linéarité dans le modèle. La fonction d'activation est généralement une fonction seuil, comme la fonction d'échelon (*step function*) ou la fonction sigmoïde. Elle décide si le neurone doit être activé ou non en fonction de la valeur de la somme pondérée.
4. Sortie : La sortie du perceptron est déterminée par la fonction d'activation. Si le neurone est activé, la sortie est généralement définie comme 1. Sinon, la sortie est définie comme 0.
5. Apprentissage : L'apprentissage du perceptron consiste à ajuster les poids des entrées pour améliorer sa capacité à classer correctement les données d'entrée. Cela est généralement réalisé à l'aide d'un algorithme d'apprentissage supervisé, tel que l'algorithme de rétropropagation du gradient.

L'idée principale derrière le perceptron est de trouver les poids appropriés qui permettent au modèle de séparer efficacement les deux catégories de données. Cela se fait en ajustant les poids de manière itérative à partir des exemples d'entraînement jusqu'à ce que le modèle atteigne une performance satisfaisante.

Il convient de noter que le perceptron simple est limité aux problèmes de classification linéairement séparables, c'est-à-dire des problèmes où les deux catégories de données peuvent être séparées par une frontière de décision linéaire. Cependant, en utilisant des architectures plus complexes, comme les réseaux de neurones multicouches, il est possible de résoudre des problèmes de classification plus complexes et non linéairement séparables.

Voici le détail des équations et fonctions mathématiques utilisées dans le perceptron :

1. Sommation pondérée :
   La sommation pondérée est calculée en multipliant chaque entrée par son poids correspondant et en prenant la somme de ces produits. Mathématiquement, cela peut être représenté comme suit :

$$z = \sum_{i=1}^{n} x_i \cdot w_i + b$$

   où :

   - z est la somme pondérée,
   - x_i est la valeur de la i-ème entrée,
   - w_i est le poids correspondant à la i-ème entrée,
   - b est le biais (un poids supplémentaire qui est ajouté à la somme pondérée).

2. Fonction d'activation :
   La fonction d'activation est appliquée à la somme pondérée pour introduire une non-linéarité dans le modèle. Voici deux exemples de fonctions d'activation couramment utilisées :

- Fonction d'échelon (Step function) :
  La fonction d'échelon est une fonction seuil qui renvoie 1 si la valeur d'entrée est supérieure à un certain seuil et 0 sinon. Mathématiquement, cela peut être représenté comme suit :

$$a = \begin{cases} 1, & si\ z \geq seuil \\ 0, & sinon \end{cases}$$

- Fonction sigmoïde :
  La fonction sigmoïde est une fonction couramment utilisée en raison de ses propriétés de différentiation douce et de sa sortie dans la plage [0, 1]. Mathématiquement, elle est définie comme suit :

$$a = \frac{1}{1 + e^{-z}}$$

3. Apprentissage :
   L'apprentissage du perceptron est généralement réalisé à l'aide d'un algorithme d'apprentissage supervisé tel que l'algorithme de rétropropagation du gradient. Cet algorithme ajuste les poids en utilisant la descente de gradient afin de minimiser une fonction de perte (loss function) qui mesure l'écart entre les sorties prédites par le modèle et les sorties réelles. Les détails spécifiques de l'algorithme de rétropropagation du gradient sont plus complexes et dépassent peut-être le cadre de cette réponse. Mais globalement, il utilise le gradient de la fonction de perte par rapport aux poids pour mettre à jour les poids du perceptron de manière itérative.

Ces équations et fonctions mathématiques sont les éléments de base du perceptron et du Deep Learning en général. En utilisant ces concepts fondamentaux, il est possible de construire des architectures plus complexes, telles que les réseaux de neurones multicouches, qui peuvent résoudre des problèmes de classification plus complexes et non linéairement séparables.

Voici la formulation mathématique plus détaillée de l'algorithme de rétropropagation du gradient :

1. Initialisation :
   Initialisez les poids du réseau de neurones avec des valeurs aléatoires ou pré-définies.

2. Forward Pass :
   - Pour chaque exemple d'entraînement, effectuez une propagation avant (forward pass) pour calculer les sorties prédites du réseau de neurones.
   - Calculez l'erreur entre les sorties prédites et les sorties réelles à l'aide d'une fonction de perte (loss function).

3. Calcul du gradient :
   Calculez les dérivées partielles de la fonction de perte par rapport aux poids du réseau de neurones à l'aide de la règle de la chaîne (règle du gradient).

4. Backward Pass :

- Effectuez une propagation arrière (backward pass) pour propager les gradients calculés à travers le réseau de neurones.
- Mettez à jour les poids en utilisant une règle d'optimisation, telle que la descente de gradient, en utilisant les gradients calculés.

5. Répétez les étapes 2 à 4 jusqu'à ce qu'un critère d'arrêt soit atteint (par exemple, un nombre fixe d'itérations ou une précision suffisamment élevée).

Voici un exemple d'implémentation simplifiée de l'algorithme de rétropropagation du gradient en Python, en utilisant la bibliothèque numpy pour les calculs matriciels :

```
# Programme python
import numpy as np

# Définition de la fonction d'activation sigmoïde
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Définition de la dérivée de la fonction sigmoïde
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))

# Fonction d'apprentissage du réseau de neurones
def train_network(inputs, targets, learning_rate, num_iterations):
    # Initialisation des poids avec des valeurs aléatoires
    weights1 = np.random.randn(inputs.shape[1], 4)  # Poids de la première couche
    weights2 = np.random.randn(4, targets.shape[1])  # Poids de la deuxième couche

    for _ in range(num_iterations):
        # Forward pass
        hidden_layer = sigmoid(np.dot(inputs, weights1))
        output_layer = sigmoid(np.dot(hidden_layer, weights2))

        # Calcul de l'erreur
        output_error = targets - output_layer

        # Calcul du gradient et de la mise à jour des poids
        output_delta = output_error * sigmoid_derivative(output_layer)
        hidden_error = np.dot(output_delta, weights2.T)
        hidden_delta = hidden_error * sigmoid_derivative(hidden_layer)

        weights2 += learning_rate * np.dot(hidden_layer.T, output_delta)
        weights1 += learning_rate * np.dot(inputs.T, hidden_delta)

    return weights1, weights2

# Exemple d'utilisation
inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
targets = np.array([[0], [1], [1], [0]])

learning_rate = 0.1
num_iterations = 10000

# Entraînement du réseau de neurones
weights1, weights2 = train_network(inputs, targets, learning_rate, num_iterations)

# Fonction de prédiction du réseau de neurones
def predict(inputs):
    hidden_layer = sigmoid(np.dot(inputs, weights1))
    output_layer = sigmoid(np.dot(hidden_layer, weights2))
    return output_layer
```

```
# Test du réseau de neurones
predictions = predict(inputs)
print(predictions)
```

Dans cet exemple, nous avons un réseau de neurones avec une couche cachée de 4 neurones. Les poids des deux couches (`weights1` pour la couche cachée et `weights2` pour la couche de sortie) sont mis à jour à chaque itération en utilisant la descente de gradient. La fonction `predict` permet de faire des prédictions à partir des entrées après l'entraînement du réseau.

Conclusion

Comme on peut le constater dans mon exposé, les mathématiques utilisées en Intelligence Artificielle et en particulier pour le Deep Learning sont très simples. D'un niveau de classe de Terminales, elles font appel aux fonctions numériques, aux fonctions logarithmiques, aux dérivées et aux probabilités. Ce qui fait réellement la puissance de l'Intelligence Artificielle c'est la capacité à gérer des volumes énormes de données (on parle de Big Data) et la puissance de calcul mise en œuvre pour entrainer les modèles de réseau de neurones. Cette puissance ne va donc pas tarder à poser d'énormes problèmes éthiques.