Toyceptron

Un vrai réseau de neurones, sans magie ni libs externes - v1.2

Introduction

L'objectif de ce projet est de **comprendre la structure d'un perceptron multi-couches**, aussi appelé "réseau de neurones". Et quelle meilleure façon de comprendre qu'en faisant ?

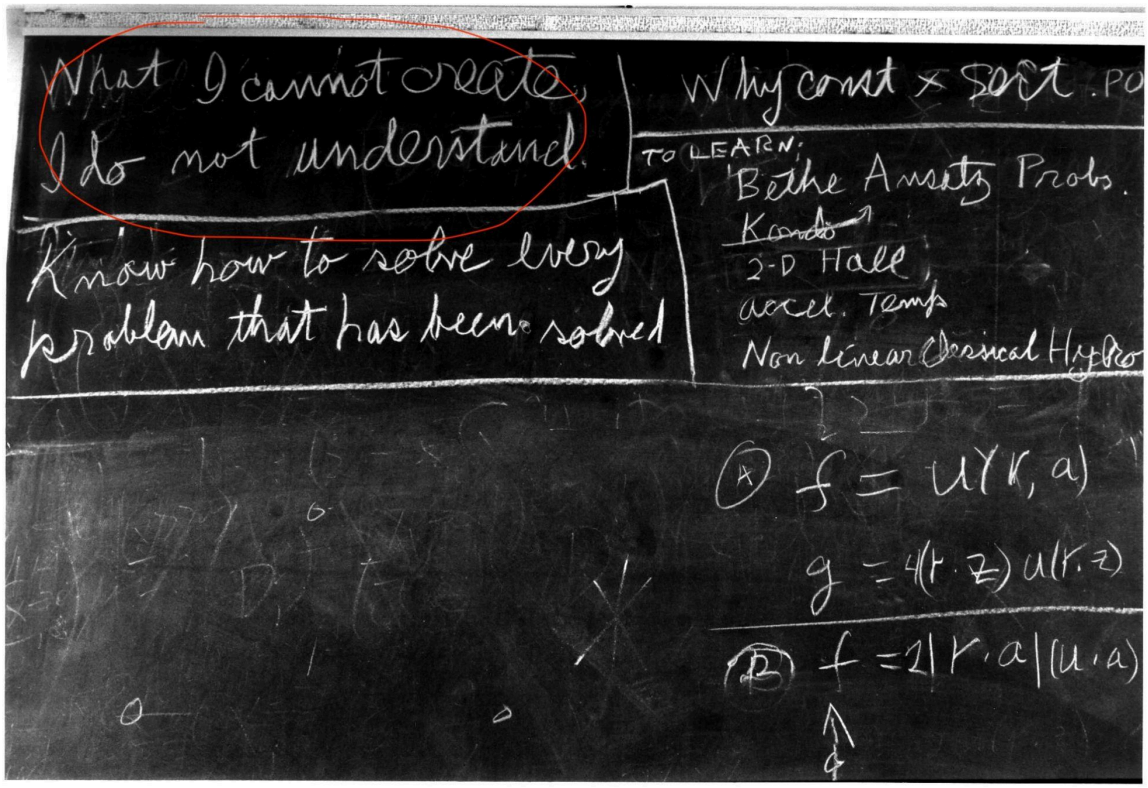


Tableau noir de Richard Feynman à sa mort (1988) : "What I cannot create I do not understand". Il faut construire pour comprendre.

L'objectif de votre perceptron est de **démystifier et comprendre les principes d'un réseau de neurones**.

Le projet

Vous allez **créer un perceptron en Python**, un modèle de réseau de neurones simple qui prend une entrée (un vecteur) et produit une sortie (un scalaire). Le comportement de votre réseau de neurones dépendra uniquement de ses paramètres initiaux (vous n'aurez pas à entraîner votre modèle).

À la fin de la semaine, votre réseau devra pouvoir être initialisé aléatoirement ou avec des paramètres pré-définis, et **faire une forward pass depuis l'entrée jusqu'à la sortie**. Et c'est tout !

Contraintes

Le langage utilisé sera Python. Aucune bibliothèque externe n'est autorisée (`numpy`, `pytorch`, `sklearn`...). **Vous utiliserez les listes de Python comme vecteurs** (inefficientes mais conceptuellement simples).

Le `main.py` de test est fourni. Votre rôle sera uniquement d'implémenter les classes demandées pour faire fonctionner ce main.

Implémentation

Vous devrez en tout et pour tout définir **trois classes :**

1. La classe `Neuron` (dans `neuron.py`)

Un neurone représente une unité de calcul élémentaire. Il doit pouvoir *a minima* :

- **stocker** ses poids et son biais,
- **calculer sa sortie** à partir d'un vecteur d'entrées.

Les poids et le biais seront aléatoires **ou** fixés à la création. La fonction d'activation sera définie au niveau du réseau : pas besoin de la stocker dans chaque neurone.

2. La classe `Layer` (dans `layer.py`)

Une **couche** est une **collection ordonnée de neurones identiques**. Elle doit pouvoir :

- créer/stocker plusieurs `Neuron`,
- appliquer à tous ses neurones un vecteur d'entrée et produire un vecteur de sortie.

Tous les neurones d'une couche auront le même nombre d'entrées. On suppose le réseau totalement relié (fully-connected).

3. La classe `Network` (dans `network.py`)

Le **réseau** est une **composition de couches**. Il doit :

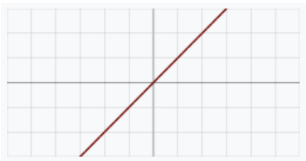
- créer une architecture complète à partir d'hyperparamètres (fournis),
- faire circuler un vecteur d'entrée à travers toutes les couches et retourner la sortie finale.

Lors de la création du réseau, on doit pouvoir spécifier au minimum :

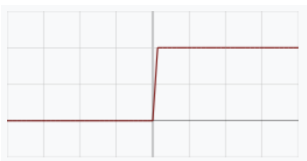
- le nombre de couches,
- la taille de chaque couche,
- la fonction d'activation,
- l'initialisation des poids et biais (valeurs fixes ou règles simples)

Ces hyperparamètres seront passés au constructeur de `Network`.

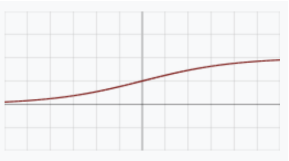
Vous devrez proposer au moins ces **quatre fonctions d'activation simples** : identité, seuil, sigmoïde, ReLU (la sigmoïde vous est donnée dans le main).



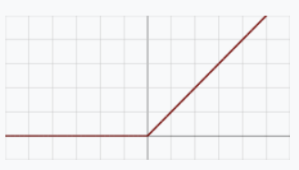
Identité



Seuil



Sigmoïde



ReLU

Bonus

Ces bonus sont grosso modo rangés par ordre croissant de difficulté, mais vous pouvez les aborder dans l'ordre de votre choix. Ils ne sont à faire qu'une fois la partie obligatoire réalisée !

- Ajouter une méthode `summary()` dans `Network` qui affiche son architecture (nombre de couches, tailles, activation)
- Ajouter des vérifications d'erreurs claires, notamment à la création si les tailles sont incompatibles entre couches
- Permettre des activations différentes par `Layer`, voire par `Neuron`
- Ajouter une couche d'entrée explicite (qui ne fait que récupérer l'input)
- Implémenter une fonction `forward_debug()` qui affiche toutes les sorties intermédiaires
- Implémenter un perceptron binaire (= dont l'output est un scalaire) qui approxime les fonctions AND et OR, avec des poids fixés à la main
- Montrer qu'un réseau sans couche cachée ne peut pas représenter la fonction XOR
- Ajouter une notion de *batch* (liste de vecteurs en entrée), pour accélérer le traitement
- Utiliser `numpy` sur tout le projet (attention, ceci seulement si vous avez réussi avec des listes simples)
- Implémenter une couche spéciale identité ("pass-through"), pour illustrer la "profondeur inutile" dans un RN
- Ajouter une sérialisation simple du réseau dans un fichier externe, pour une sauvegarde/chargement des poids. Le fichier peut avoir le format

de votre choix : n'importe quel format texte fonctionne très bien (txt, csv...), mais vous êtes libre de choisir un format binaire. Pas de lib autorisée pour la sérialisation !

Compétences visées

- Python
- POO
- Machine Learning

Rendu

Le projet est à rendre sur votre github :

https://github.com/prenom-nom/toyceptron

Base de connaissances

- main.py de test
- Article du perceptron
- Vidéo de 3blue1brown sur les réseaux de neurones
- Un playground pour jouer avec un réseau simple
- W3Schools "Learning machines to imitate human intelligence"