Exercice : Perceptron

Objectif :

L'objectif de cet exercice est de construire, entraîner et évaluer pas à pas un **Perceptron Multicouche (MLP) très simple** avec une seule couche cachée pour la classification des chiffres manuscrits du jeu de données MNIST. Vous apprendrez les concepts fondamentaux du *Machine Learning* en manipulant les données et les calculs neurone par neurone.

Contexte du problème

La reconnaissance de chiffres manuscrits est un défi classique en informatique. Le jeu de données MNIST contient des images de 28×28 pixels de chiffres manuscrits (de 0 à 9). Votre défi sera de créer un programme capable de deviner quel chiffre est représenté sur chaque image.

Concepts Clés à comprendre et implémenter

Le Perceptron Multicouche (MLP) : Imaginez un réseau de "neurones" connectés en couches. Chaque neurone reçoit des informations, fait un calcul et transmet un résultat.

Couche d'entrée : C'est là que vos données (les images) entrent dans le réseau. Pour MNIST, chaque image de 28×28 pixels sera "aplatie" en une longue liste de 784 nombres (un nombre par pixel). Chaque nombre sera une entrée pour un "neurone d'entrée".

On peut dire que les valeurs des pixels (après normalisation) **sont les activations** des neurones d'entrée. Elles sont directement passées à la première couche "réelle" de calcul (la couche cachée).

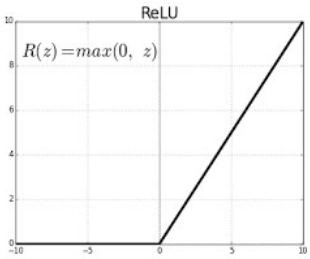
Couche cachée : Une couche intermédiaire de neurones qui effectue des calculs sur les informations reçues de la couche d'entrée. C'est ici que la "magie" opère pour apprendre des motifs plus complexes. Vous choisirez combien de neurones elle aura (par exemple, 128 ou 256).

Couche de sortie : C'est la couche finale qui donne la réponse de votre réseau. Pour chaque chiffre possible (de 0 à 9), il y aura un neurone de sortie. Le neurone qui donne la valeur la plus élevée indiquera la prédiction du modèle.

Poids et Biais : Chaque connexion entre neurones a un "poids", un nombre qui indique l'importance de cette connexion. Chaque neurone a aussi un "biais", un nombre qui l'aide à décider quand s'activer. Ces poids et biais sont ce que le réseau "apprend".

Fonction d'activation (ReLU) : Après avoir calculé la somme des entrées pondérées et du biais, chaque neurone passe ce résultat dans une fonction d'activation. Pour la couche cachée, nous utiliserons ReLU (Rectified Linear Unit). Si le résultat est positif, ReLU le garde tel quel. S'il est négatif, ReLU le transforme en 0. Elle permet au réseau d'apprendre des relations complexes.

$$\mathbf{ReLU(z) = max(0,z)}$$



Normalisation des données : Les valeurs de pixels sont entre 0 et 255. Il est préférable de les transformer en nombres entre 0 et 1 en les divisant par 255.0. Cela aide le réseau à mieux apprendre.

Étapes d'implémentation détaillées

1. Chargement et prétraitement des données

- **Charger les données** :
  - Les fichiers mnist_handwritten_train.json (60 000 exemples) et mnist_handwritten_test.json (10 000 exemples) contiennent chacun un tableau d'objets. Chaque objet a un champ image (tableau de 784 éléments de pixels [0,255]) et un champ label (le chiffre).
  - Lisez ces fichiers JSON. Vous obtiendrez une liste d'images (tableaux de 784 nombres) et une liste d'étiquettes (nombres entre 0 et 9).
- **Normaliser les images** :
  - Pour chaque pixel dans chaque image, divisez sa valeur par 255.0. Par exemple, si un pixel vaut 128, il deviendra $128/255.0 \approx 0.50$.

2. Définition de l'architecture et Initialisation

- **Définir la taille des couches** :
  - **Couche d'entrée** : 784 "neurones" (les 784 pixels de l'image).
  - **Couche cachée** : Choisissez un nombre de neurones, par exemple **128 neurones**.
  - **Couche de sortie** : 10 neurones (un pour chaque chiffre de 0 à 9).

- **Initialiser les poids et les biais** :
    - Créez des tableaux pour stocker les **poids** et les **biais**.
    - **Poids (entrée vers cachée)** : Un tableau de 784×128 nombres. Initialisez-les avec de très petites valeurs aléatoires (par exemple, entre -0.1 et 0.1).
    - **Biais (cachée)** : Un tableau de 128 nombres. Initialisez-les à 0.
    - **Poids (cachée vers sortie)** : Un tableau de 128×10 nombres. Initialisez-les avec de très petites valeurs aléatoires.
    - **Biais (sortie)** : Un tableau de 10 nombres. Initialisez-les à 0.

3. Propagation avant (Comment le réseau fait une prédiction)

Pour chaque image :

- **Calcul de la couche cachée** :
    - Pour **chaque neurone de la couche cachée** (de 0 à 127) :
        - Initialisez une somme_ponderee à 0.
        - Pour **chaque pixel d'entrée** (de 0 à 783) :
            - Ajoutez pixel_actuel * poids_entre_ce_pixel_et_ce_neurone_cache à somme_ponderee.
        - Ajoutez le biais_de_ce_neurone_cache à somme_ponderee.
        - Appliquez la fonction **ReLU** : Si somme_ponderee est négative, le résultat pour ce neurone caché est 0. Sinon, c'est somme_ponderee.
        - Stockez ce résultat (l'activation) du neurone caché.
- **Calcul de la couche de sortie** :
    - Pour **chaque neurone de la couche de sortie** (de 0 à 9) :
        - Initialisez une somme_ponderee à 0.
        - Pour **chaque activation de la couche cachée** (de 0 à 127) :
            - Ajoutez activation_neurone_cache * poids_entre_ce_neurone_cache_et_ce_neurone_sortie à somme_ponderee.
        - Ajoutez le biais_de_ce_neurone_sortie à somme_ponderee.
        - Stockez ce résultat. C'est la "score" ou "logit" pour cette classe.

- **Décision finale** :
    - Trouvez le neurone de sortie qui a la valeur la plus élevée. L'indice de ce neurone (par exemple, si le neurone 3 a la valeur la plus élevée, la prédiction est 3) est la prédiction de votre modèle. Vous pouvez utiliser la fonction max de votre langage.

4. Entraînement : Apprendre de ses erreurs (Règle d'apprentissage simplifiée)

C'est la partie où le réseau ajuste ses poids et biais pour s'améliorer. Au lieu de calculs complexes de dérivées, nous allons utiliser une règle simple.

- **Boucle d'entraînement** : Répétez ce processus pour un certain nombre d'**époques** (par exemple, 5 ou 10). Une époque signifie que le réseau a vu toutes les images d'entraînement une fois.
- **Pour chaque image d'entraînement :**
    1. **Effectuez la propagation avant** comme décrit ci-dessus pour obtenir la prédiction.
    2. **Comparez la prédiction à la vraie étiquette** (le chiffre correct).
    3. **Ajustez les poids et les biais** (avec un petit taux_apprentissage, par exemple 0.01) :
        - **Si la prédiction est correcte :**
            - Félicitez légèrement les neurones : Augmentez très légèrement les poids et biais qui ont contribué à cette bonne prédiction.
                - Pour le neurone de sortie correct : Augmentez un peu son biais. Pour chaque neurone caché qui a contribué positivement à ce neurone de sortie, augmentez un peu le poids de la connexion.
                - Pour les neurones cachés : S'ils ont eu une activation positive et ont contribué à la bonne décision, vous pouvez légèrement augmenter leurs poids d'entrée. (Ceci est la partie la plus simplifiée et peut être une simple augmentation si la prédiction est bonne.)

- **Si la prédiction est incorrecte :**
  - **Pour le neurone de sortie qui aurait dû gagner (la vraie étiquette) :** Augmentez son biais et les poids des connexions qui lui arrivent depuis la couche cachée. (L'idée est de le rendre plus fort pour la prochaine fois).
  - **Pour le neurone de sortie qui a gagné par erreur :** Diminuez son biais et les poids des connexions qui lui arrivent depuis la couche cachée. (L'idée est de le rendre moins fort).
  - **Pour la couche cachée :** C'est plus délicat ici. Une approche simple serait de :
    - Si un neurone caché a fortement contribué à un *bon* neurone de sortie (celui qui aurait dû gagner) : Ajustez légèrement ses poids d'entrée pour qu'il s'active plus.
    - Si un neurone caché a fortement contribué à un *mauvais* neurone de sortie (celui qui a gagné par erreur) : Ajustez légèrement ses poids d'entrée pour qu'il s'active moins.
    - (Cette partie est très heuristique. L'idée est de montrer qu'on propage une forme de correction.)

- **Taux d'apprentissage** : Multipliez toujours les ajustements de poids et de biais par un petit nombre (le taux_apprentissage) pour ne pas trop modifier le réseau d'un coup.

5. Évaluation du modèle

- Après l'entraînement, utilisez le **jeu de données de test** (les 10 000 images de mnist_handwritten_test.json).
- Pour chaque image de test :
  1. Effectuez une propagation avant pour obtenir la prédiction.
  2. Comparez la prédiction à la vraie étiquette.
  3. Comptez combien de prédictions sont correctes.
- Calculez la **précision (accuracy)** : (Nombre de prédictions correctes / Nombre total d'images de test) * 100%.