TP 7 : Réseaux de neurones

1. Introduction

Un MLP (Multilayer Perceptron) est un algorithme d'apprentissage automatique supervisé (ML) qui appartient à la classe des réseaux de neurones artificiels à propagation avant (feedforward artificial neural networks). L'algorithme est essentiellement entraîné sur les données afin d'apprendre une fonction. Étant donné un ensemble de caractéristiques et une variable cible (par exemple, des étiquettes), il apprend une fonction non linéaire pour la classification ou la régression. Dans ce TP, nous nous concentrerons uniquement sur le cas de la classification.

2. L'ensemble de données (dataset)

Pour cet exemple pratique, nous utiliserons l'ensemble de données MNIST. La base de données MNIST est une célèbre base de données de chiffres manuscrits utilisée pour entraîner plusieurs modèles en apprentissage automatique. Il y a des images manuscrites de 10 chiffres différents, donc le nombre de classes est de 10 (voir Figure 1).

Figure 1: Quelques exemples de l'ensemble de données.

Notes :

- Comme nous travaillons avec des images, celles-ci sont représentées par des matrices 2D et la dimension initiale des données est de 28 par 28 pour chaque image (28x28 pixels). Les images 2D sont ensuite aplaties et donc représentées par des vecteurs à la fin. Chaque image 2D est transformée en un vecteur 1D de dimension [1, 28x28] = [1, 784]. Finalement, notre ensemble de données comporte 784 caractéristiques/variables/colonnes.
- Lien pour télécharger les données:

https://datahub.io/machine-learning/mnist_784/r/mnist_784.csv

3. Importation et préparation des données

Question:

Ecrire les commandes Python qui permettent de :

- Charger les données à partir du dataset mnist
- Séparer les données d'entrées et cibles
- Normaliser l'intensité des images pour la ramener dans l'intervalle [0,1].

Rappelez-vous que chaque image 2D est maintenant transformée en un vecteur 1D de dimension [1, 28x28] = [1, 784]. Vérifions cela maintenant.

Question:

Ecrire la commande Python qui permet de vérifier les dimensions ainsi que les intervalles des données.

Cela renvoie : (70000, 784). Nous avons 70 000 images aplaties (échantillons), chacune contenant 784 pixels (28*28=784) (variables/caractéristiques). Ainsi, la matrice de poids de la couche d'entrée aura une forme de 784 x #neurons_in_1st_hidden_layer. La matrice de poids de la couche de sortie aura une forme de #neurons_in_3rd_hidden_layer x #number_of_classes.

4. Entraînement du modèle

Question:

Maintenant, construire le modèle de réseaux de neurones, entraîner-le et effectuer la classification. Utiliser 3 couches cachées avec respectivement 50, 20 et 10 neurones. De plus, nous fixerer le nombre maximal d'itérations à 100 et le taux d'apprentissage à 0,1.

Rappelez-vous que chaque image 2D est maintenant transformée en un vecteur 1D de dimension [1, 28x28] = [1, 784].

5. Évaluation du modèle

Question:

Maintenant, évaluer le modèle. Vous allez estimer la précision moyenne des données et des étiquettes d'entraînement et de test.

6. Visualisation de l'évolution de la fonction de coût

Question:

Afficher un graphe afin de visulaiser à quelle vitesse l'erreur a-t-elle diminué pendant l'entraînement.

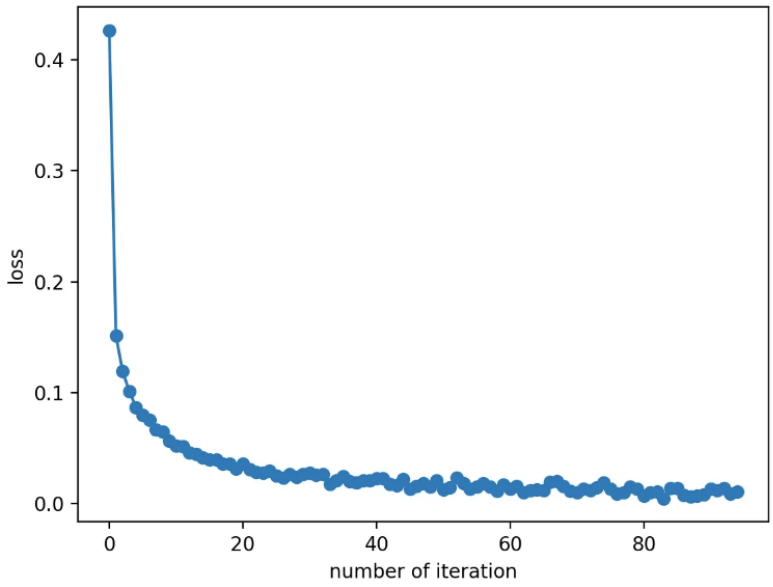

Figure 2: Évolution de la perte au fil des itérations d'entraînement.

Ici, nous voyons que l'erreur diminue très rapidement pendant l'entraînement et elle se stabilise après la 40ème itération (rappelez-vous que nous avons défini 100 itérations au maximum en tant qu'hyperparamètre).

7. Visualisation des poids estimés

Ici, nous devons d'abord comprendre comment les poids (paramètres appris du modèle pour chaque couche) sont stockés.

Selon la documentation, l'attribut classifier.coefs_ est une liste de forme (n_layers-1,) de tableaux de poids, où la matrice de poids à l'indice i représente les poids entre la couche i et la couche i+1. Dans cet exemple, nous avons défini 3 couches cachées et nous avons également la couche d'entrée et de sortie. Nous nous attendons donc à avoir 4 tableaux de poids pour les poids entre les couches (entrée-L1, L1-L2, L2-L3 et L3-sortie).

De même, classifier.intercepts_ est une liste de vecteurs de biais, où le vecteur à l'indice i représente les valeurs de biais ajoutées à la couche i+1.

Question:

Ecrire la commande Python qui permet de vérifier cela.

Rappel : la matrice de poids de la couche d'entrée aura une forme de 784 x #neurones_dans_la_1ère_couche_cachée. La matrice de poids de la couche de sortie aura une forme de #neurones_dans_la_3ème_couche_cachée x #nombre_de_classes.

Visualisation des poids appris de la couche d'entrée

Question:

Ecrire les commandes Python qui permettent de visualiser les poids estimés.

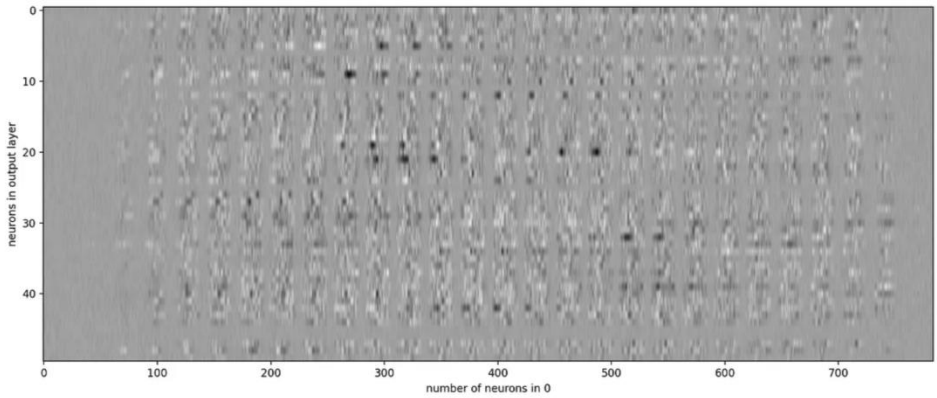

Figure 3 : Visualisation des poids appris des neurones entre l'entrée et la 1ère couche cachée.

Question:

Ecrire la commande Python qui permet de reformater les poids et les représenter en tant qu'images 2D à nouveau.

8. Conclusion

Le classificateur MLP est un modèle de réseau neuronal très puissant qui permet l'apprentissage de fonctions non linéaires pour des données complexes. La méthode utilise la propagation avant pour construire les poids et calcule ensuite la fonction d'erreur. Ensuite, la rétropropagation est utilisée pour mettre à jour les poids de manière à réduire l'erreur. Cela se fait de manière itérative et le nombre d'itérations est un hyperparamètre d'entrée. D'autres hyperparamètres importants sont le nombre de neurones dans chaque couche cachée et le nombre total de couches cachées. Ils doivent être affinés.