TP Multi-Layer Perceptron
Introduction à l'IA et à la Science des Données - M1 Info

Ce TP contient deux parties. La première a pour objectif d'étudier les réseaux de neurones artificiels de type perceptron multi-couche (*Multi-Layer Perceptron – MLP*). Vous utiliserez la fonction **MLPClassifier** définie dans le module **sklearn.neural_network**, qui fournit une implémentation du perceptron multi-couche pour la classification. MLPClassifier comprend plusieurs paramètres à ajuster tels que le nombre de neurones pour chaque couche cachée (**hidden_layer_sizes**), la fonction d'activation (**activation**) et l'algorithme d'optimisation (**solver**). Consultez la documentation pour plus de détails sur ces paramètres :

scikit-learn.org/stable/modules/generated/sklearn. neural_network.MLPClassifier.html

La deuxième partie a pour objectif de définir un réseau de neurones convolutionnel (CNN) en utilisant PyTorch.

Partie I : MLP dans sklearn

1 Apprentissage de fonctions booléennes

Dans cet exercice, on s'intéresse à l'apprentissage des fonctions booléennes AND, OR et XOR par un MLP. Nous rappelons la définition de ces opérateurs ci-dessous :

Entrée		Sortie		
X_1	X_2	X_1 AND X_2	X_1 OR X_2	X_1 XOR X_2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

En Python, on utilisera une variable x contenant l'ensemble des entrées fournies au MLP et une variable y contenant l'ensemble des résultats attendus correspondants aux différentes entrées. Par exemple, pour l'opérateur AND :

```
X = [ [0., 0.], [0., 1.], [1., 0.], [1., 1.] ] # Entrées
y = [0, 0, 0, 1] # Résultats attendus
```

Après spécification d'un classifieur à partir de la fonction **MLPClassifier**, celui-ci sera entrainé sur les couples (x, y) en appelant **classifier.fit(x, y)**. Le classifier pourra ensuite être utilisé pour prédire la sortie à renvoyer pour une liste d'entrées *x_test* (différentes ou non de celles de x) en appelant **classifier.predict(x_test)**.

1. Définissez un classifieur MLP pour apprendre l'opérateur AND. Vous n'utiliserez aucune couche cachée (**hidden_layer_sizes = ()**), une activation linéaire (fonction **identity**) et un solver de type **lbfgs**. Vérifiez que les résultats prédits par le classifieur sont corrects.
2. Définissez un classifieur MLP pour apprendre l'opérateur OR. Vous n'utiliserez aucune couche cachée (**hidden_layer_sizes = ()**), une activation linéaire (fonction **identity**) et un solver de type **lbfgs**. Vérifiez que les résultats prédits par le classifieur sont corrects.
3. Définissez un classifieur MLP pour apprendre l'opérateur XOR :
   (a) Vous n'utiliserez aucune couche cachée (**hidden_layer_sizes = ()**), une activation linéaire (fonction **identity**) et un solver de type **lbfgs**. Est-ce que les résultats prédits par le classifieur sont corrects ? Comment l'expliquez-vous ?
   (b) Vous utiliserez deux couches cachées composées de 4 neurones (première couche) et 2 neurones (deuxième couche), des activations linéaires (fonction **identity**) et un solver de type **lbfgs**. Est-ce que les résultats prédits par le classifieur sont corrects ? Comment l'expliquez-vous ?
   (c) Vous utiliserez deux couches cachées composées de 4 neurones (première couche) et 2 neurones (deuxième couche), des activations non-linéaires de type tangente hyperbolique (fonction **tanh**) et un solver de type **lbfgs**. Recommencez l'apprentissage plusieurs fois après avoir constaté les résultats prédits par le classifieur. Est-ce que les résultats prédits par le classifieur sont corrects ? Comment l'expliquez-vous ?

2 Classification d'images

On s'intéresse cette fois à un problème d'apprentissage plus difficile : l'identification du chiffre (manuscrit) contenu dans une image. La collection de données **digits** fournie par **scikit-learn** sera utilisée ici. 90% de la collection servira pour l'ensemble d'apprentissage (sur lequel le classifieur sera entrainé) et 10% pour l'ensemble de test (sur lequel la capacité de généralisation du classifieur sera évaluée) :

```
from sklearn.datasets import load_digits dataset = load_digits ()
X = dataset.data # Inputs
y = dataset.target # Associated outputs
train_X,test_X,train_y,test_y = train_test_split(X,y,test_size=0.1)
```

Entrainez un classifieur sur **(train_X, train_y)**. Jouez avec les paramètres de **MLPClassifier** (nombre de couches cachées, nombre de neurones par couche, fonction d'activation, solveur,

etc.). Les performances du classifieur peuvent être évaluées sur l'ensemble de test en termes d'exactitude de classification pour identifier quelle configuration de MLP fonctionne le mieux :

```
from sklearn.metrics import accuracy_score
test_y_pred = classifier.predict(test_X) # Predicted results
print("Accuracy : ",accuracy_score(test_y,test_y_pred))
```

Quel classifieur se comporte le mieux ? Commenter les résultats obtenus.

Partie II : PyTorch et CNN

La deuxième partie de ce TP consiste en l'implémentation d'un CNN en utilisant PyTorch. Pour cela, vous devez tout d'abord suivre le tutoriel PyTorch disponible sous `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`. Il vous faudra ensuite installer `torch` et `torchvision` (voir `https://github.com/pytorch/pytorch#installation` pour les instructions d'installation). Le plus simple en général est d'installer directement les binaires (voir `https://pytorch.org/`). Suivez ensuite les quatre parties du tutoriel :

- `https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html` : notions de base sur les tenseurs (vous pouvez sauter la dernière partie sur les tenseurs CUDA)
- `https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html` : autograd pour la différentiation automatique
- `https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html` : réseaux de neurones de base
- `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html` : classification à base de CNN pour Cifar10 (vous pouvez sauter la dernière partie sur les GPU).

Pour chaque partie, vous pouvez soit télécharger le notebook correspondant, soit l'exécuter sur Google colab. Exécutez tous les éléments de code et jouez avec eux pour comprendre comment fonctionnent les procédures.

Enfin, dans un nouveau notebook à créer, :

1. Implémentez le réseau LeNet, tel que défini dans la quatrième partie du tutoriel, sur les données MNIST (disponibles dans `torchvision`) ;
2. Calculez la précision (accuracy) et comparez les résultats à ceux obtenus avec un MLP ;
3. Modifiez le réseau (taille des feature maps, taille du noyau) et commentez les résultats obtenus ;
4. Modifiez le taux d'apprentissage (learning rate) et commentez les résultats obtenus ;
5. Bonus : implémentez un régularisation dropout sur l'apprentissage.

Pour chaque point, il est important de fournir le code et les commentaires.