TP Perceptron Multi-Couche

M. Tami, T. Thonet,
E. Gaussier

L'objectif de ce TP est 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 : `http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`.

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.]] # Entrees
y = [0, 0, 0, 1] # Resultats attendus
```

Après spécification d'un `classifier` 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 # Entrees
y = dataset.target # Resultats attendus
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, solver...). 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) # Resultats predits
print("Exactitude :", accuracy_score(test_y, test_y_pred))
```