TP sur les réseaux de neurones

1. Utilisation de ScikitLearn avec Python 3

On suppose ici que anaconda(3) est installé.

Nous utiliserons la librairie ScikitLearn de Python 3.

Nous importons d'abord les librairies utiles :

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

2. Expériences de prédiction de cancer de la poitrine

2.1 Chargement de la base de données `Breast Cancer Data`

Pour nos premières expériences, nous allons utiliser la base de données `Breast Cancer Data` qui comporte des descriptions de tumeurs avec une étiquette précisant si la tumeur est maligne ou bénigne. La base de données contient 569 exemples décrits à l'aide de 30 descripteurs.

Nous chargeons d'abord la base, puis nous examinons l'un des exemples qu'elle contient. Nous donnons les commandes en lignes de commandes, qui sont précédées de `>>` . Nous chargeons les descriptions des exemples dans la variable `X` et les étiquettes dans la variable `y`.

```
>>> from sklearn.datasets import load_breast_cancer
>>> cancer = load_breast_cancer()
>>> cancer['data'].shape
(569, 30)

>>> X = cancer['data']
>>> y = cancer['target']
```

2.2 Préparation des ensembles d'apprentissage et de test

On divise les données de la base `Breast Cancer Data` en un ensemble d'exemples pour l'apprentissage et un autre ensemble, disjoint du premier, d'exemples de test. La fonction `train_test_split` de ScikitLearn fait cela pour nous.

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X, y)
```

2.3 Préparation des données

Les réseaux de neurones apprennent mieux si les données dont préalablement normalisées, c'est-à-dire si on pris soin que la variance des valeurs soit la même pour tous les descripteurs. Il faut bien sûr veiller à ce que cette même normalisation soit appliquée aux données de test.

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
# Fit only to the training data
>>> scaler.fit(X_train)
StandardScaler(copy=True, with_mean=True, with_std=True)

# Now apply the transformations to the data:
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
```

2.4 Premier apprentissage

Nous allons utiliser ici un apprentissage par réseaux de neurones. Nous illustrons ici avec un réseau de neurones à trois couches cachées, chacune comportant 10 neurones.

```
>>> from sklearn.neural_network import MLPClassifier
>>> mlp = MLPClassifier(hidden_layer_sizes=(30,30,30))

>>> mlp.fit(X_train,y_train)
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(30, 30, 30), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

Chacun des paramètres apparaissant dans le retour de l'appel de la fonction `mlp.fit` peut être précisé lors de l'appel de la fonction `MLPClassifier`.

2.5 Évaluation de la performance sur l'ensemble de test

```
>>> predictions = mlp.predict(X_test)

>>> from sklearn.metrics import classification_report,confusion_matrix
>>> print(confusion_matrix(y_test,predictions))
[[50  3]
 [ 0 90]]
```

Ici, nous avons d'abord demandé que la prédiction des étiquettes soit réalisée sur les données de test, puis nous avons fait calculer la matrice de confusion.

Pour avoir un tableau de bord des résultats en terme de *précision*, *rappel*, et *F1-socre*, on fait appel à la commande suivante :

```
>>> print(classification_report(y_test, predictions))
             precision    recall  f1-score   support

          0       1.00      0.94      0.97        53
          1       0.97      1.00      0.98        90

avg / total       0.98      0.98      0.98       143
```

1. Vous essaierez de voir si vous pouvez améliorer la performance en prédiction, déjà remarquable, en choisissant d'autres architectures de réseau de neurones : en faisant varier le nombre de couches et le nombre de neurones par couche.
2. Vous regarderez aussi ce qui se passe si vous changez le nombre d'itérations avant arrêt de l'apprentissage. Pour cela vous pourrez préciser le nombre maximal d'itérations dans l'appel de la fonction. Dans l'exemple suivant, ce nombre est fixé à 20.

```
>>> mlp = MLPClassifier(hidden_layer_sizes=(30,30,30),max_iter=20))
```

2.6 Évaluation de la performance par validation croisée

Une méthode courante d'évaluation d'un apprentissage supervisé de concept est la validation croisée. Ici, nous utilisons une validation croisée à trois plis. Une valeur courante est 10 plis.

La fonction `cross_val_score` calcule les erreurs associées à chaque pli. Vous pouvez ensuite en calculer la moyenne.

```
>>> from sklearn.model_selection import cross_val_score

>>> cross_val_score(mlp, X_train, y_train, cv=3, scoring="accuracy")
Out[71]: array([ 0.9654 ,  0.9634 ,  0.96615])
```

Rq : Les chiffres donnés ici sont certainement différents de ceux que vous trouverez.

3. Expériences de reconnaissance de chiffres manuscrits

3.1 Chargement de la base de données MNIST

Pour nos premières expériences, nous allons utiliser la base de données MNIST qui comporte les descriptions de chiffres manuscrits en format 28×28, soit 784 pixels, et les étiquettes associées. La base contient en tout 70 000 exemples, c'est-à-dire 7000 exemples de chaque chiffre.

Nous chargeons d'abord la base, puis nous examinons l'un des exemples qu'elle contient.

```
>>> from sklearn.datasets import fetch_mldata

>>> mnist = fetch_mldata('MNIST original')

>>> mnist
```

```
Out[8]:
{'COL_NAMES': ['label', 'data'],
 'DESCR': 'mldata.org dataset: mnist-original',
 'data': array([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=uint8),
 'target': array([ 0.,  0.,  0., ...,  9.,  9.,  9.])}

>>> X, y = mnist["data"], mnist["target"]

>>> X.shape
Out[10]: (70000, 784)
>>> y.shape
Out[12]: (70000,)
>>> %matplotlib inline

>>> import matplotlib

>>> import matplotlib.pyplot as plt

>>> some_digit = X[36000]

>>> some_digit_image = some_digit.reshape(28, 28)

>>> plt.imshow(some_digit_image, cmap = matplotlib.cm.binary, interpolation="nearest")
```

La dernière instruction devrait produire une image telle que :

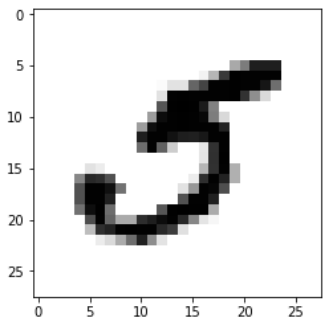

FIGURE 1 – Un exemple de '5' de la base MNIST.

La figure 2 montre un échantillon de chiffres contenus dans la base MNIST.

3.2 Premier apprentissage

Nous sélectionnons 60 000 exemples pour l'apprentissage et 10 000 pour le test (la base MNIST est organisée de cette manière).

FIGURE 2 – Un échantillon de chiffres de la base MNIST.

Pour les tests à venir, il est important de garantir que les exemples sont aléatoirement répartis dans la base d'apprentissage afin que tout tirage d'un sous-ensemble comporte à peu près la même proportion de chaque chiffre.

```
>>> X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

>>> shuffle_index = np.random.permutation(60000)

>>> X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

Nous allons d'abord apprendre des classifieurs binaires, à deux classes. Ici, nous choisissons d'entraîner un classifieur à apprendre à reconnaître les 5 contre tous les autres chiffres.

```
>>> y_train_5 = (y_train == 5)
>>> y_test_5 = (y_test == 5)
```

Nous utilisons maintenant un réseau de neurones à deux couches (20, 20).

```
>>> mlp = MLPClassifier(hidden_layer_sizes=(20,20))
>>> mlp.fit(X_train,y_train_5)

>>> predictions = mlp.predict(X_test)

>>> print(confusion_matrix(y_test_5,predictions))
[[9058   50]
 [  56  836]]
```

```
>>> print(classification_report(y_test_5,predictions))
             precision    recall  f1-score   support

      False       0.99      0.99      0.99      9108
       True       0.94      0.94      0.94       892

avg / total       0.99      0.99      0.99     10000
```

On peut de même calculer la performance par validation croisée, ici avec trois plis. Attention si vous augmentez le nombre de plis, vous multipliez le temps de calcul qui peut être assez long pour les réseaux de neurones.

```
>>> cross_val_score(mlp, X_train, y_train_5, cv=3, scoring="accuracy")
Out[350]: array([ 0.98595,  0.986  ,  0.98685])
```

Ici, les résultats sont obtenus avec un réseau à deux couches (10,10).

1. Vous essaierez de voir si vous pouvez améliorer la performance en prédiction, déjà remarquable, en choisissant d'autres architectures de réseau de neurones : en faisant varier le nombre de couches et le nombre de neurones par couche.
2. Vous regarderez aussi ce qui se passe si vous changez le nombre d'itérations avant arrêt de l'apprentissage. Pour cela vous pourrez préciser le nombre maximal d'itérations dans l'appel de la fonction.

3.3 Limite de ce qui est appris

Vous aurez sans doute obtenu des performances remarquables sur la reconnaissance des chiffres manuscrits dans la base MNIST.

Mais que se passerait-il si les mêmes exemples de validation subissaient une rotation horaire (ou anti-horaire) de θ degrés tout en conservant la même base d'apprentissage ?

1. Vous écrirez un programme qui fait « tourner » de θ degré l'exemple de chiffre mis en entrée.
2. Vous ferez subir à toute la base de test une rotation (horaire ou anti-horaire) de 5 degrés, puis de 10 degrés, etc.
   Vous mesurerez les performances obtenues pour chaque angle de rotation testé tant en terme de taux d'erreur que par la matrice de confusion (Attention : la base d'exemples reste la même, sans rotation). Qu'observez-vous ?