Nom et Prénom : BATTACHE Nassim.

Master 1 OIVM - Optique Image Vision MultiMedia

Apprentissage supervisé

Abstract :

Ce rapport résume l'Apprentissage par perceptron multi-couches sous sklearnl.

Introduction

Prise en main du modèle

Travail sur le jeu de données Iris

Travail sur le jeu de données MNIST

Conclusion

Introduction

Les réseaux de neurones artificiels ou en bref les ANN sont largement utilisées aujourd'hui dans de nombreuses applications et la classification en fait partie. Il existe également de nombreuses bibliothèques et cadres dédiés à la construction de réseaux de neurones en toute simplicité. Cependant, la plupart de ces frameworks et outils nécessitent de nombreuses lignes de code à implémenter par rapport à une simple bibliothèque de Scikit-Learn.

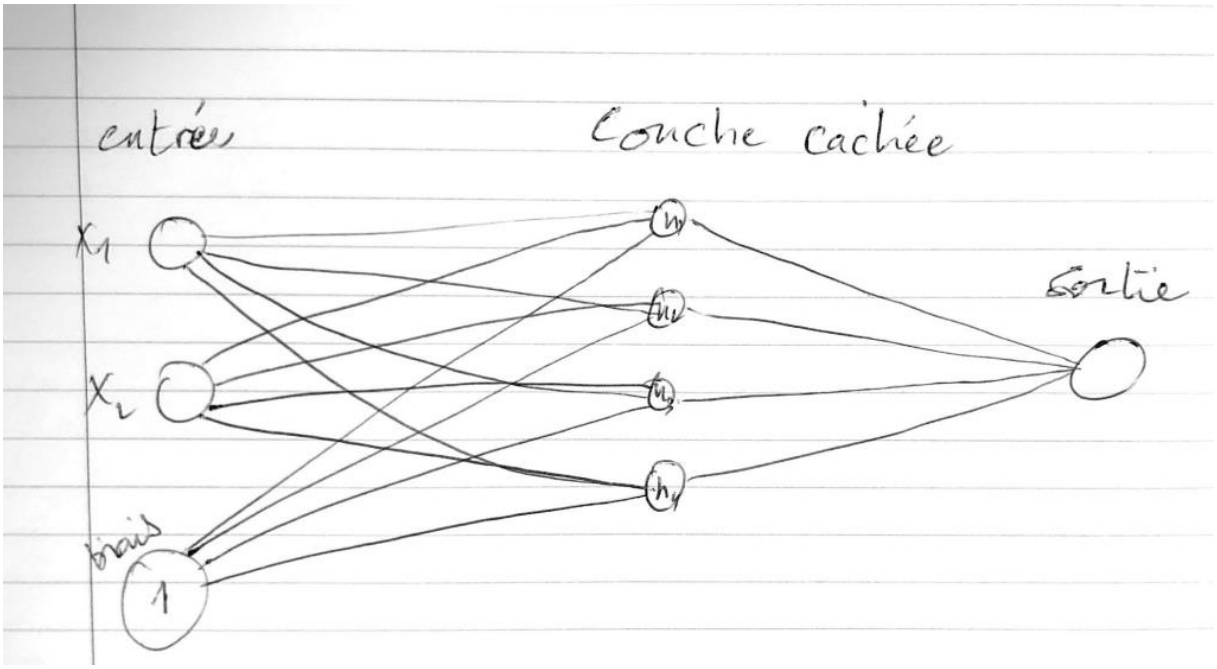
l'un des réseaux de neurones les plus faciles à implémenter pour la classification de Scikit-Learn appelé MLPClassifier.

Prise en main du modèle

1.Soit le jeu de données S = {([0, 0], 0), ([0, 1], 1)}. Nous cherchons à apprendre un modèle f(x) : R2 ! {0, 1} qui, à chaque exemple, associe une sortie booléenne. Créez ce jeu de données (X et y) en utilisant les tableaux de numpy.

```
x_train=np.array([[0,0],[0,1]])
y_train=np.array([0,1])
```

2. Créez un modèle MLP de classification avec une couche cachée de 4 neurones : dessinez ce modèle sur feuille, créez-le avec MLPClassifier, puis apprenez le avec S, et testez-le avec les entrées [2.,2.], puis [-1.,-2.]. A l'aide des coefficients appris, complétez le dessin du MLP et vérifiez la sortie manuellement.



3. Complétez S en ajoutant les exemples ([1, 1], 0) et ([1, 0], 1). Il s'agit du problème du XOR. Apprenez les deux réseaux dont l'architecture a été présentée en cours avec des perceptrons linéaires a seuil. Les prédictions des réseaux obtenus sont-elles correctes ? Quels sont les coefficients et biais obtenus ?

```
import numpy as np
from sklearn.neural_network import MLPClassifier

x_train=np.array([[0,0],[0,1]])
y_train=np.array([0,1])
x_test=np.array([[2.,2.],[-1.,-2.]])


MLP_clf = MLPClassifier(hidden_layer_sizes=(4), activation='logistic', solver='lbfgs', alpha=0.0001, learning_rate='adaptive'
MLP_clf.fit(x_train, y_train)
y_pred = MLP_clf.predict(x_test)
print(y_pred)
```

On obtient le résultat suivant :

```
[1 0]
```

Oui les prédictions obtenues sont correcte car XOR fonctionne bien sur notre x_test

4. Soit S0 = {([0, 0], [0, 1]), ([1, 1], [1, 1])}. Nous cherchons à apprendre un modèle f(x) : R2 ! {0, 1}2 qui, à chaque exemple, associe deux sorties booléennes distinctes. Au vu de S0, quelles sont les deux fonctions booléennes que nous cherchons à apprendre ? Créez ce jeu de données (X et y) en utilisant les tableaux de numpy.

```
import numpy as np
from sklearn.neural_network import MLPClassifier

x_train=np.array([[0,0],[1,1]])
y_train=np.array([[0, 1],[1, 1]])
x_test=np.array([[2.,2.],[-1.,-2.]])
MLP_clf = MLPClassifier(hidden_layer_sizes=(4), activation='logistic', solver='lbfgs', alpha=0.0001, learning_rate='adaptive'
MLP_clf.fit(x_train, y_train)
y_pred = MLP_clf.predict(x_test)
print("les prediction sont :" +str(y_pred))
print ("les coefficients sont: "+str( MLP_clf.coefs_))
print ("les biais des couche cachees  et la sortie sont :" +str(MLP_clf.intercepts_))
```

```
les prediction sont :[[1 1]
 [0 1]]
les coefficients sont: [array([[ 1.404451  , -2.77703753,  2.4980617 , -2.73815629],
       [ 1.41587339, -2.77495584,  2.49761427, -2.73895053]]), array([[ 2.53183829,  0.15595431],
       [-5.99050145,  0.18102598],
       [ 3.53072429,  0.18911753],
       [-6.05546327,  0.11455512]])]
les biais des couche cachees  et la sortie sont :[array([-1.3454215 ,  3.04150961, -3.23784567,  2.99942232]), array([ 3.303
72972, 10.16717894])]
```

Les deux fonctions booléennes sont le y2=Xnor(x1+x2) et le y1=and

5. Déterminez une architecture MLP la plus simple possible pour l'apprentissage conjoint de ces deux fonctions (i.e. combien de neurones dans l'unique couche cachée?). Le réseau correspondant devra ˆêtre appris et testé sur S0 lui-même.

Quelle architecture obtenez-vous : sauvegardez ses paramètres dans un fichier. On utilisera pour cela les composantes du modèle que sont n layers , n outputs , out activation , coefs , intercepts , classes , loss . En combien d'itérations le solveur converge-t-il (cf. Composante n iter) ?

```
from sklearn.model_selection import GridSearchCV

X_train2 = np.array([[0,0],[1,1]])
y_train2 = np.array([[0,1],[1,1]])
X_test2 = np.array([[2.,2.],[-1.,-2.]])
param_grid = {
    'hidden_layer_sizes': [2, 3, 4, 5, 6, 7, 8],

    }

mlp_clf = GridSearchCV(MLPClassifier(activation = 'logistic', solver = 'lbfgs', learning_rate = 'adaptive', max_iter = 100),
mlp_clf.fit(X_train2, y_train2)
y_predict = mlp_clf.predict(X_test2)
print("Les classes prédites pour les données de test {} sont: {} respectivement.".format(X_test, y_predict))
print("La meilleure configuration est: ", mlp_clf.best_params_)
```

```
   [5.4 3.9 1.7 0.4]
   [5.5 2.3 4.  1.3]
   [6.8 3.2 5.9 2.3]
   [7.6 3.  6.6 2.1]] sont: [[1 1]
   [0 1]] respectivement.
  La meilleure configuration est:  {'hidden_layer_sizes': 2}
```

La meilleure configuration est : hidden_layer_size :2

Travail sur le jeu de données Iris

1. Charger ce jeu de données. Toutes les expériences à venir devront ˆêtreé faites en train/test split 1.

```
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
iris = datasets.load_iris()

X= iris.data
y= iris.target
start_time = time.time()

#On conserve 50% du jeu de données pour l'évaluation


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7,random_state=0)
MLP_clf = MLPClassifier(hidden_layer_sizes=(200,150,100), activation='logistic', solver='lbfgs', learning_rate='adaptive')
MLP_clf.fit(X_train, y_train)
y_pred = MLP_clf.predict(X_test)
score = MLP_clf.score(X_test, y_test)
end_time = time.time()
print("le score est :"+str(score))
print('temps ecoule = '+str(end_time - start_time))
```

2. Apprendre cinq modèles de classification des données Iris, avec des réseaux qui ont respectivement de 1 à 5 couches cachées, et des tailles de couches entre 10 et 300 au choix. Quelles sont les performances en taux de bonne classification et en temps d'apprentissage obtenus pour chaque modèle ?

```python
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
iris = datasets.load_iris()

X= iris.data
y= iris.target
start_time = time.time()



def Hiddenlayersizes3(x1,y1,z1):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7,random_state=0)
    MLP_clf = MLPClassifier(hidden_layer_sizes=(x1,y1,z1), activation='logistic', solver='lbfgs', learning_rate='adaptive')
    MLP_clf.fit(X_train, y_train)
    y_pred = MLP_clf.predict(X_test)
    score = MLP_clf.score(X_test, y_test)
    end_time = time.time()
    print("pour les 3 couche de taille : " +str(x1)+"-->"+str(y1)+"-->"+str(z1))
    print('temps ecoule = '+str(end_time - start_time))
    return print("le score est :"+str(score))
def Hiddenlayersizes2(x1,y1):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7,random_state=0)
    MLP_clf = MLPClassifier(hidden_layer_sizes=(x1,y1), activation='logistic', solver='lbfgs', learning_rate='adaptive')
    MLP_clf.fit(X_train, y_train)
    y_pred = MLP_clf.predict(X_test)
    score = MLP_clf.score(X_test, y_test)
    end_time = time.time()
    print("pour les 2 couche de taille : " +str(x1)+"-->"+str(y1))
```

```python
# pour differentes taille des 2 couche de neuron
Hiddenlayersizes2(1,15)
Hiddenlayersizes2(10,100)
Hiddenlayersizes2(200,300)

# pour differentes taille des 3 couche de neuron
Hiddenlayersizes3(1,15,30)
Hiddenlayersizes3(10,100,200)
Hiddenlayersizes3(100,150,300)

# pour differentes taille des 4 couche de neuron
Hiddenlayersizes4(1,15,30,90)
Hiddenlayersizes4(10,100,200,300)
```

Voici les résultats obtenus :

```
pour les 2 couche de taille : 1-->15
temps ecoule = 0.06194806098937988
le score est :0.638095238095238
pour les 2 couche de taille : 10-->100
temps ecoule = 0.286989688873291
le score est :0.9619047619047619

pour les 2 couche de taille : 200-->300
temps ecoule = 2.9593069553375244
le score est :0.9619047619047619

pour les 3 couche de taille : 1-->15-->30
```

```
temps ecoule = 3.196627140045166
le score est :0.8
pour les 3 couche de taille : 10-->100-->200
temps ecoule = 4.3012378215789795
le score est :0.9714285714285714
pour les 3 couche de taille : 100-->150-->300
temps ecoule = 6.9399309158325195
le score est :0.9619047619047619
pour les 4 couche de taille : 1-->15-->30-->90
temps ecoule = 6.977614641189575
le score est :0.3142857142857143
pour les 4 couche de taille : 10-->100-->200-->300
temps ecoule = 7.2141172885894775
le score est :0.3142857142857143


nombre de n_iter : 200
```

Remarque :

Quand le nombre de couche augmente le temps de repense augmente

Quand la taille des couches augmente le temps de repense augmente

Le score est stable à partir de la 3emme couche

3. Comparer les résultats avec une classifieur `a noyau de type SVM avec noyaux polynomial : clsvm = svm.SVC(kernel='poly').

```python
from sklearn import svm
clsvm = svm.SVC(kernel='poly')
clsvm.fit(X_train, y_train)
y_pred = clsvm.predict(X_test)
score = clsvm.score(X_test, y_test)
print ("le score avec un classifieur a noyau de type SVM ="+str(score))
```

On obtient le résultat suivant :

```
le score avec un classifieur a noyau de type SVM =0.9428571428571428
```

Dans ce cas par exemple la classification des réseaux de neurone est mieux avec un nombre de couche cacher qui est égale a 3 ou 4 par exemple .

```python
from statistics import mean, stdev
for i in range(len(X)):
    print("la valeur moyenne de l'attribut {} est: {}".format(i+1,mean(X[i])))
    print("l'écart-type de l'attribut {} est: {}".format(i+1,stdev(X[i])))
```

```
la valeur moyenne de l'attribut 1 est: 2.55
l'écart-type de l'attribut 1 est: 2.1794494717703365
la valeur moyenne de l'attribut 2 est: 2.375
l'écart-type de l'attribut 2 est: 2.036950334855189
la valeur moyenne de l'attribut 3 est: 2.35
l'écart-type de l'attribut 3 est: 1.997498435543818
la valeur moyenne de l'attribut 4 est: 2.35
l'écart-type de l'attribut 4 est: 1.9122412679017953
la valeur moyenne de l'attribut 5 est: 2.55
l'écart-type de l'attribut 5 est: 2.1563858652847823
la valeur moyenne de l'attribut 6 est: 2.85
l'écart-type de l'attribut 6 est: 2.230844384233617
la valeur moyenne de l'attribut 7 est: 2.425
l'écart-type de l'attribut 7 est: 1.9362764954072718
la valeur moyenne de l'attribut 8 est: 2.525
l'écart-type de l'attribut 8 est: 2.109304782781916
la valeur moyenne de l'attribut 9 est: 2.225
l'écart-type de l'attribut 9 est: 1.8227726133558184
la valeur moyenne de l'attribut 10 est: 2.4
```

2. Réapprendre et tester les cinq modèles MLP et le SVM après avoir normalisé les données en entrées. Observez-vous des améliorations ?

```
pour les 2 couche de taille : 1-->15
temps ecoule = 0.12578940391540527
le score est :0.9619047619047619
pour les 2 couche de taille : 10-->100
temps ecoule = 0.20779061317443848
le score est :0.9523809523809523
pour les 2 couche de taille : 200-->300
temps ecoule = 1.0955638885498047
le score est :0.9619047619047619
pour les 3 couche de taille : 1-->15-->30
temps ecoule = 1.3302545547485352
le score est :0.9619047619047619
pour les 3 couche de taille : 10-->100-->200
temps ecoule = 1.8461182117462158
le score est :0.9523809523809523
pour les 3 couche de taille : 100-->150-->300
temps ecoule = 2.8674685955047607
le score est :0.9619047619047619
```

oui on remarque que la normalisation apporte des améliorations au niveau du score .

1. Pour chacun des cinq modèles précédemment appris après normalisation, et pour chaque solveur disponible, indiquez : le temps de convergence, le nombre d'itérations pour converger, et les performances du modèle appris. Résumez ces informations au sein d'un tableau récapitulatif.

```
tab = ("lbfgs", "sgd", "adam")
for i in tab:
    print('--------------------',i,'--------------------')
    start_time = time.time()
    clf1 = MLPClassifier(solver=i,hidden_layer_sizes=(200),max_iter=n,random_state=0)
    clf1.fit(X_train,y_train)
    end_time = time.time()
    print(clf1.score(X_test,y_test))
    print('temps ecoulé pour une couche cachée = '+str(end_time - start_time))
    print('n_iter',clf1.n_iter_)
    start_time = time.time()
    clf2 = MLPClassifier(solver =i,hidden_layer_sizes=(100,200),max_iter=n,random_state=0)
    clf2.fit(X_train,y_train)
    end_time = time.time()
    print(clf2.score(X_test,y_test))
    print('temps ecoulé pour 2 couche cachée = '+str(end_time - start_time))
    print('n_iter',clf2.n_iter_)
    start_time = time.time()
    clf3 = MLPClassifier(solver =i,hidden_layer_sizes=(100,200,300),max_iter=n,random_state=0)
    clf3.fit(X_train,y_train)
    end_time = time.time()
    print(clf3.score(X_test,y_test))
    print('temps ecoulé 3 couche cachée = '+str(end_time - start_time))
    print('n_iter',clf3.n_iter_)
    start_time = time.time()
    clf4 = MLPClassifier(solver =i,hidden_layer_sizes=(100,150,250,300),max_iter=n,random_state=0)
    clf4.fit(X_train,y_train)
    end_time = time.time()
    print(clf4.score(X_test,y_test))
    print('temps ecoulé 4 couche cachée = '+str(end_time - start_time))
    print('n_iter',clf4.n_iter_)
    start_time = time.time()
    clf5= MLPClassifier(solver =i,hidden_layer_sizes=(50,100,150,200,250,300),max_iter=n,random_state=0)
    clf5.fit(X_train,y_train)
    end_time = time.time()
```

```
-------------------- lbfgs --------------------
0.9619047619047619
temps ecoulé pour une couche cachée = 0.03703498840332031
n_iter 20
0.9619047619047619
temps ecoulé pour 2 couche cachée = 0.1288442611694336
n_iter 21
0.9619047619047619
temps ecoulé 3 couche cachée = 0.5603196620941162
n_iter 25
0.9619047619047619
temps ecoulé 4 couche cachée = 0.9386575222015381
n_iter 26
0.9523809523809523
temps ecoulé 5 couche cachée = 4.265818357467651
n_iter 91
```

```
-------------------- sgd --------------------
```

```
0.8476190476190476
temps ecoulé pour une couche cachée = 0.9529769420623779
n_iter 899
0.8952380952380953
temps ecoulé pour 2 couche cachée = 1.7930026054382324
n_iter 1000
0.9333333333333333
temps ecoulé 3 couche cachée = 3.788252592086792
n_iter 979
0.9523809523809523
temps ecoulé 4 couche cachée = 4.6702985763549805
n_iter 890
0.9523809523809523
temps ecoulé 5 couche cachée = 5.430448055267334
n_iter 758
_____________________________________________________________________
-------------------- adam --------------------
0.9619047619047619
temps ecoulé pour une couche cachée = 0.5190973281860352
n_iter 455
0.9428571428571428
temps ecoulé pour 2 couche cachée = 0.44471025466918945
n_iter 229
0.9428571428571428
temps ecoulé 3 couche cachée = 0.45997023582458496
n_iter 108
0.9428571428571428
temps ecoulé 4 couche cachée = 0.4910881519317627
n_iter 72
0.9428571428571428
temps ecoulé 5 couche cachée = 0.4994850158691406
n_iter 53
_____________________________________________________________________
```

2. Choisissez le modèle qui propose de meilleurs résultats, et tentez d'améliorer ces résultats en faisant varier la magnitude de la régularisation L2 (paramètre). Parvenez-vous à battre le SVM? Si oui, avec quels hyperparamètres finaux de votre MLP ?

```
param_grid = {
    'alpha': [0.0001, 0.001, 0.1, 1]
    }

mlp_clf = GridSearchCV(MLPClassifier(hidden_layer_sizes = (2),activation = 'logistic', solver = 'lbfgs', learning_rate = 'ada
mlp_clf.fit(X_train, y_train)
print("Le score est: ", mlp_clf.score(X_test, y_test))
print("La meilleure valeur pour alpha = {}".format(mlp_clf.best_estimator_.alpha))
```

On obtient le résultat suivant :

```
Le score est:  0.9619047619047619
La meilleure valeur pour alpha = 0.001
```

Travail sur le jeu de données MNIST

pour cette partie j'ai essayé d'utiliser le gridsearch qui a pris beaucoup de temps

```
from sklearn.model_selection import GridSearchCV

#Appliquons un GridSearchCV pour trouver la meilleure configuration possible
param_grid = {
    'hidden_layer_sizes': [2, 30, 40, 50],
    'activation':['logistic','relu'],
    'solver':['lbfgs', 'sgd', 'adam'],
    'learning_rate':['adaptive'],
    'alpha': [0.0001, 0.001,0.01]
    }

mlp_clf = GridSearchCV(MLPClassifier(), param_grid, cv=3, n_jobs=4, verbose=1)
mlp_clf.fit(X_train_flat, y_train)
y_predict = mlp_clf.predict(X_test_flat)
print("Meilleur score : " + str(svm_clf.best_estimator_.score(X_test_flat, y_test)))
print("La meilleure configuration est: " + str(svm_clf.best_params_))
```

mais J'ai réussi à trouver une configuration qui mon permis de trouver un meilleur de 98% sur les données d'apprentissage et 97% sur les données de test

voici mon code

J'avais des difficultés à ouvrir l'ensemble de données mnist car dans la dernière version de scikit-learn (version 0.23.1), il n'y a rien nommé "fetch_mldata()". (Cela était présent dans la version précédente de 0.19.)

Dans la dernière version, jai utiliser fetch_openml() pour le faire.

```
import matplotlib.pyplot as plt
import time
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import fetch_openml
mnist = fetch_openml("mnist_784")

# rescale the data, use the traditional train/test split
X, y = mnist.data / 255., mnist.target
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]

# mlp = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=400, alpha=1e-4,
#                     solver='sgd', verbose=10, tol=1e-4, random_state=1)

start_time = time.time()

mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha=1e-4,
                    solver='sgd', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)

mlp.fit(X_train, y_train)
end_time = time.time()
print("le score d'entrainement est : %f" % mlp.score(X_train, y_train))
print("le score de test est : %f" % mlp.score(X_test, y_test))
print('temps ecoulé = '+str(end_time - start_time))
```

Voici le résultat obtenu :

```
le score d'entrainement est : 0.986800
le score de test est : 0.970000
temps ecoulé = 65.73171329498291
```

Conclusion

On peut dire sur les MLP que :

- le Classifieur est très précis (si bien paramétré)
- Incrémentalité
- Scalabilité (capacité à être mis en œuvre sur de grandes bases)

Cependant on a aussi quelques défauts qui sont :

- On risque un danger de sur-apprentissage pour cause trop de neurones dans la couche cachée par exemple.
- Difficulté de paramétrage c'est très difficile de prédire le nombre de neurones dans la couche cachée
- On a aussi le problème de convergence (optimum local).