

# TP2 Etude Multi-layered Neural Network

Entrée [2]:



```
1  #*****
2  #TP2 : Multi-layered Perceptron
3  #*****
4
5  #import necessaires
6  import numpy as np
7  from sklearn import datasets
8  from sklearn.model_selection import train_test_split
9  from sklearn.neural_network import MLPClassifier
10 from sklearn import metrics
11 import time
12
13
14
15 #charger le jeu de données MNIST
16 mnist=datasets.fetch_mldata('MNIST original')
17
18 #randomise Data et target
19 indices = np.random.randint(70000, size=70000)
20 data = mnist.data[indices]
21 target = mnist.target[indices]
22 #pour séparer le dataset en 2 échantillons de trainint et de test
23 # on veut un training set de 49000
24 xtrain, xtest, ytrain, ytest =train_test_split(data, target, train_size=49000)
25
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecatio
n.py:77: DeprecationWarning: Function fetch_mldata is deprecated; fe
tch_mldata was deprecated in version 0.20 and will be removed in ver
sion 0.22
```

```
warnings.warn(msg, category=DeprecationWarning)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecatio
n.py:77: DeprecationWarning: Function mldata_filename is deprecated;
mldata_filename was deprecated in version 0.20 and will be removed i
n version 0.22
```

```
warnings.warn(msg, category=DeprecationWarning)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_
split.py:2179: FutureWarning: From version 0.21, test_size will alwa
ys complement train_size unless both are specified.
```

```
FutureWarning)
```

```
1  #MLP avec 1 hidden layer de 50 neurones
2
3  clf = MLPClassifier(hidden_layer_sizes=(50))
4
5  clf.fit(xtrain, ytrain)
6  predict = clf.predict(xtest)
7  score = clf.score(xtest,ytest)
8  recall = metrics.recall_score(ytest, predict, average = 'macro')
9  precision = metrics.precision_score(ytest, predict, average='macro')
10 loss01 = metrics.zero_one_loss(ytest, predict)
11
12
13 print("Ce modèle de MLP, d'1 couche de 50, à un score de ", score*100, "%.")
14 # Ce modèle de MLP, d'1 couche de 50, à un score de 94.83333333333334 %.
15
16 print("4eme image : prédiction ",predict[3], "reel : ", ytest[3])
17 #4eme image : prédiction 3.0 reel : 3.0
18
19 print ("ce modèle de MLP à une précision de", precision*100, "%.")
20 # Ce modèle de MLP à une précision de 94,83333333333334 %.
21
22 print ("ce modèle de MLP à un recall de",recall*100, "%.")
23
24 print ("ce modèle de MLP à un zero-one_loss de",recall*100, "%.")
25
26
27 #*****
28 # REMARQUE :
29 # Pour average = macro
30 # Score = (TP + TN)/(TP+TN+FP+FN)
31 # Précision = TP/(TP + FP)
32 # recall = TP/(TP+FN)
33 # zero one loss : standard loss function in classification (equivalent of squa
34 # pour average = micro, c'est la même chose.
35 #*****
36
37 #Autre exécution
38 #Ce modèle de MLP, d'1 couche de 50, à un score de 96.64761904761905 %.
39 #4eme image : prédiction 1.0 reel : 1.0
40 #ce modèle de MLP à une précision de 96.64204324487432 %.
41 #ce modèle de MLP à un recall de 96.59646102012928 %.
42 #ce modèle de MLP à un zero-one_loss de 96.59646102012928 %.
43
44 #NOTE : quelqu'un à une précision différente de score
```

Entrée [\*]:



```
1 # faire varier le nombre de couches de 2 à 100 ;
2
3 hidden_layer =(50,)*100
4
5 ResScore =[]
6 ResPrecision = []
7 ResRecall = []
8 ResLoss = []
9
10 for i in range (100):
11     clf = MLPClassifier(hidden_layer_sizes= hidden_layer[0:i])
12     clf.fit(xtrain, ytrain)
13     predict = clf.predict(xtest)
14     score = clf.score(xtest,ytest)
15     precision = metrics.precision_score(ytest, predict, average='macro')
16     recall = metrics.recall_score(ytest, predict, average = 'macro')
17     loss01 = metrics.zero_one_loss(ytest, predict)
18     ResScore.append(score)
19     ResPrecision.append(precision)
20     ResRecall.append(recall)
21     ResLoss.append(loss01)
22     #print("pour ", i ,"hidden Layers, score = ", score*100, "%, précision =",
23     print(i)
24
```

Entrée [\*]:



```
1 # pour garder une trace des résultats et ne pas runner encore une fois
2
3 print(ResScore)
4 print(ResPrecision)
5 print(ResRecall)
6 print(ResLoss)
7
8
```

Entrée [\*]:



```
1 import matplotlib.pyplot as plt
2
3
4 fig, axarr = plt.subplots(4, sharex=True, figsize=(10,10))
5 axarr[0].plot(range(100), ResScore)
6 axarr[0].set_title('Number of hidden layers from 1 to 99')
7 axarr[0].set_ylabel('Score')
8 axarr[1].plot(range(100), ResPrecision)
9 axarr[1].set_ylabel('Precision')
10 axarr[2].plot(range(100), ResRecall)
11 axarr[2].set_ylabel('Recall')
12 axarr[3].plot(range(100), ResLoss)
13 axarr[3].set_ylabel('Zero-to-one Loss')
14
15 plt.show()
```

```

1  # 5 modèles Mnist
2  # avec entre 1 et 10 couches de 10 à 300 neurones
3
4  # 1 couche, max neurone
5  clf1 = MLPClassifier(hidden_layer_sizes=(300))
6  # 3 couches, aléatoire neurones
7  clf3 = MLPClassifier(hidden_layer_sizes=(20,200,50))
8  # 5 couches, gaussien neurones
9  clf5 = MLPClassifier(hidden_layer_sizes=(50, 100, 200, 100,50))
10 # 7 couches, décroissant neurones
11 clf7 = MLPClassifier(hidden_layer_sizes=(300, 250, 200, 150, 100, 50, 10))
12 # 9 couches, croissant neurones
13 clf9 = MLPClassifier(hidden_layer_sizes=(30, 60, 90, 120, 150, 180, 210, 240,
14
15 ClassifierList = ("clf1", "clf3", "clf5", "clf7", "clf9")
16
17 ResScore2 = []
18 ResPrecision2 = []
19 ResRecall2 = []
20 ResLoss2 = []
21 ResTimeTraining2 = []
22 ResTimePrediction2 = []
23
24 def runClass(clf, i):
25     #Training
26     startTrain = time.time()
27     clf.fit(xtrain, ytrain)
28     endTrain = time.time()
29     #Prediction
30     startpred = time.time()
31     predict = clf.predict(xtest)
32     endpred = time.time()
33     #Metrics
34     score = clf.score(xtest, ytest)
35     precision = metrics.precision_score(ytest, predict, average='macro')
36     recall = metrics.recall_score(ytest, predict, average='macro')
37     loss01 = metrics.zero_one_loss(ytest, predict)
38     timetrain = endTrain - startTrain
39     timePred = endpred - startpred
40     #Append
41     ResScore2.append(score*100)
42     ResPrecision2.append(precision*100)
43     ResRecall2.append(recall)
44     ResLoss2.append(loss01)
45     ResTimePrediction2.append(timePred)
46     ResTimeTraining2.append(timetrain)
47     #print
48     print("pour le ", i, "eme modèle, score = ", score*100, "%, précision =", pr
49     print("      temps apprentissage : ", timetrain, "sec , temps prediction = "
50
51 runClass(clf1, 1)
52 runClass(clf3, 2)
53 runClass(clf5, 3)

```

```

54 runClass(clf7, 4)
55 runClass(clf9, 5)
56
57
58 #1ERE EXECUTION :
59 #pour le 1 eme modèle, score = 97.17142857142858 %, précision = 97.171428571
60 # temps apprentissage : 58.63614535331726 sec , temps prediction = 0.28324
61 #pour le 2 eme modèle, score = 96.26666666666667 %, précision = 96.266666666
62 # temps apprentissage : 43.548506021499634 sec , temps prediction = 0.175
63 #pour le 3 eme modèle, score = 97.78571428571429 %, précision = 97.785714285
64 # temps apprentissage : 38.507989168167114 sec , temps prediction = 0.261
65 #pour le 4 eme modèle, score = 98.18095238095238 %, précision = 98.180952380
66 # temps apprentissage : 290.31491470336914 sec , temps prediction = 0.545
67 #pour le 5 eme modèle, score = 97.88571428571429 %, précision = 97.885714285
68 # temps apprentissage : 116.49138069152832 sec , temps prediction = 0.507
69
70 #2EME EXECUTION
71 #pour le 1 eme modèle, score = 96.53809523809524 %, précision = 96.632511971
72 # temps apprentissage : 86.58849000930786 sec , temps prediction = 0.5075
73 #pour le 2 eme modèle, score = 96.43809523809523 %, précision = 96.395315011
74 # temps apprentissage : 59.65086317062378 sec , temps prediction = 0.1715
75 #pour le 3 eme modèle, score = 97.46666666666667 %, précision = 97.448987332
76 # temps apprentissage : 47.93583607673645 sec , temps prediction = 0.2263
77 #C:\ProgramData\Anaconda3\Lib\site-packages\sklearn\metrics\classification.py:
78 # 'precision', 'predicted', average, warn_for)
79 #pour le 4 eme modèle, score = 11.4 %, précision = 1.1400000000000001 %.
80 # temps apprentissage : 61.67111420631409 sec , temps prediction = 0.5355
81 #pour le 5 eme modèle, score = 97.44761904761904 %, précision = 97.410766602
82 # temps apprentissage : 100.82642769813538 sec , temps prediction = 0.515

```

Entrée [\*]:



```

1 # pour garder une trace des résultats et ne pas runner encore une fois
2 print(ResScore2)
3 print(ResPrecision2)
4 print(ResRecall2)
5 print(ResLoss2)
6 print(ResTimeTraining2)
7 print(ResTimePrediction2)
8

```

```
1 import matplotlib.pyplot as plt
2
3 fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
4 axarr[0].scatter(range(5), ResScore2)
5 axarr[0].set_title('The five classifiers with respectively 1,3,5,7,9 hidden la
6 axarr[0].set_ylabel('Score (%)')
7 axarr[1].scatter(range(5), ResPrecision2)
8 axarr[1].set_ylabel('Precision (%)')
9 axarr[2].scatter(range(5), ResRecall2)
10 axarr[2].set_ylabel('Recall ')
11 axarr[3].scatter(range(5), ResLoss2)
12 axarr[3].set_ylabel('Zero-to-one Loss')
13 axarr[4].scatter(range(5), ResTimeTraining2)
14 axarr[4].set_ylabel('Training Time in sec')
15 axarr[5].scatter(range(5), ResTimePrediction2)
16 axarr[5].set_ylabel('Prediction Time in sec')
17
18 plt.show()
```

```
1 # ETUDE DE CONVERGENCE des algo d'optimisation
2 # LBFGS : an optimizer in the family of quasi-Newton methods.
3 # SGD : stochastic gradient descent.
4 # ADAM : a stochastic gradient-based optimizer proposed by Kingma, Diederik, a
5
6 # Je récupère les tuples utilisés pour les clf1,3,5,7,9
7
8 # TODO : A runner
9
10 # 1 couche, max neurone
11 tuple1 = (300)
12 # 3 couches, aléatoire neurones
13 tuple3 = (20,200,50)
14 # 5 couches, gaussien neurones
15 tuple5 = (50, 100, 200, 100,50)
16 # 7 couches, décroissant neurones
17 tuple7 = (300, 250, 200, 150, 100, 50, 10)
18 # 9 couches, croissant neurones
19 tuple9 = (30, 60, 90, 120, 150, 180, 210, 240, 270)
20
21 ResScore3 = []
22 ResPrecision3 = []
23 ResRecall3 = []
24 ResLoss3 = []
25 ResTimeTraining3 = []
26 ResTimePrediction3 = []
27
28 DataOrder = ["clf1-lbfgs", "clf1-sgd", "clf1-adam", "clf3-lbfgs", "clf3-sgd",
29
30 def runClassSolver(numTuple, solv, i):
31     # Train
32     clf = MLPClassifier(hidden_layer_sizes=numTuple, solver= solv)
33     startTrain =time.time()
34     clf.fit(xtrain, ytrain)
35     endTrain = time.time()
36     # predict
37     startpred= time.time()
38     predict = clf.predict(xtest)
39     endpred = time.time()
40     # Metrics
41     score = clf.score(xtest,ytest)
42     precision = metrics.precision_score(ytest, predict, average='macro')
43     recall = metrics.recall_score(ytest, predict, average = 'macro')
44     loss01 = metrics.zero_one_loss(ytest, predict)
45     timetrain = endTrain - startTrain
46     timePred = endpred - startpred
47     #Append
48     ResScore3.append(score*100)
49     ResPrecision3.append(precision*100)
50     ResRecall3.append(recall)
51     ResLoss3.append(loss01)
52     ResTimePrediction3.append(timePred)
53     ResTimeTraining3.append(timetrain)
```



```

54     #Print
55     print ("SOLVER : ", j)
56     print("pour le ", i,"eme modèle, score = ", score*100, "%, précision =",pr
57     print("      temps apprentissage : ", timetrain, "sec , temps prediction = "
58
59
60     for j in ('lbfgs', 'sgd', 'adam'):
61         runClassSolver(tuple1, j, 1)
62     for j in ('lbfgs', 'sgd', 'adam'):
63         runClassSolver(tuple3, j, 2)
64     for j in ('lbfgs', 'sgd', 'adam'):
65         runClassSolver(tuple5, j, 3)
66     for j in ('lbfgs', 'sgd', 'adam'):
67         runClassSolver(tuple7, j, 4)
68     for j in ('lbfgs', 'sgd', 'adam'):
69         runClassSolver(tuple9, j, 5)
70
71

```

Entrée [\*]:



```

1  # pour garder une trace des résultats et ne pas runner encore une fois
2  print(ResScore3)
3  print(ResPrecision3)
4  print(ResRecall3)
5  print(ResLoss3)
6  print(ResTimeTraining3)
7  print(ResTimePrediction3)
8

```

Entrée [\*]:



```

1  fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
2  axarr[0].scatter(range(15), ResScore3)
3  axarr[0].set_title('The five classifiers with 3 training methods : lbfgs, sgd,
4  axarr[0].set_ylabel('Score (%)')
5  axarr[1].scatter(range(15), ResPrecision3)
6  axarr[1].set_ylabel('Precision (%)')
7  axarr[2].scatter(range(15), ResRecall3)
8  axarr[2].set_ylabel('Recall ')
9  axarr[3].scatter(range(15), ResLoss3)
10 axarr[3].set_ylabel('Zero-to-one Loss')
11 axarr[4].scatter(range(15), ResTimeTraining3)
12 axarr[4].set_ylabel('Training Time in sec')
13 axarr[5].scatter(range(15), ResTimePrediction3)
14 axarr[5].set_ylabel('Prediction Time in sec')
15
16 plt.show()

```

```

1  # VARIATION DES FONCTIONS D'ACTIVATION
2
3
4  ResScore4 = []
5  ResPrecision4 = []
6  ResRecall4 = []
7  ResLoss4 = []
8  ResTimeTraining4 = []
9  ResTimePrediction4 = []
10
11 def runClassActiv(numTuple, activ, i):
12     # Train
13     clf = MLPClassifier(hidden_layer_sizes=numTuple, activation= activ)
14     startTrain = time.time()
15     clf.fit(xtrain, ytrain)
16     endTrain = time.time()
17     # Predict
18     startpred= time.time()
19     predict = clf.predict(xtest)
20     endpred = time.time()
21     # Metrics
22     score = clf.score(xtest,ytest)
23     precision = metrics.precision_score(ytest, predict, average='macro')
24     recall = metrics.recall_score(ytest, predict, average = 'macro')
25     loss01 = metrics.zero_one_loss(ytest, predict)
26     timetrain = endTrain - startTrain
27     timePred = endpred - startpred
28     #Append
29     ResScore4.append(score*100)
30     ResPrecision4.append(precision*100)
31     ResRecall4.append(recall)
32     ResLoss4.append(loss01)
33     ResTimePrediction4.append(timePred)
34     ResTimeTraining4.append(timetrain)
35     #Print
36     print ("SOLVER : ", j)
37     print("pour le ", i,"eme modèle, score = ", score*100, "%, précision =",pr
38     print("      temps apprentissage : ", timetrain, "sec , temps prediction = "
39
40
41 for j in ('identity', 'logistic', 'tanh', 'relu'):
42     runClassActiv(tuple1, j, 1)
43 for j in ('identity', 'logistic', 'tanh', 'relu'):
44     runClassActiv(tuple3, j, 2)
45 for j in ('identity', 'logistic', 'tanh', 'relu'):
46     runClassActiv(tuple5, j, 3)
47 for j in ('identity', 'logistic', 'tanh', 'relu'):
48     runClassActiv(tuple7, j, 4)
49 for j in ('identity', 'logistic', 'tanh', 'relu'):
50     runClassActiv(tuple9, j, 5)
51

```

Entrée [\*]:



```
1 # pour garder une trace des résultats et ne pas runner encore une fois
2 print(ResScore4)
3 print(ResPrecision4)
4 print(ResRecall4)
5 print(ResLoss4)
6 print(ResTimeTraining4)
7 print(ResTimePrediction4)
8 len(ResScore4)
```

Entrée [\*]:



```
1 # ordre des données :
2 # Clf1 Id, Logistic, tanh, relu | Clf3 Id, ... | Clf5 ... | Clf7 ... | Clf9 ..
3
4 fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
5 axarr[0].scatter(range(20), ResScore4)
6 axarr[0].set_title('The five classifiers with different activation functions :
7 axarr[0].set_ylabel('Score (%)')
8 axarr[1].scatter(range(20), ResPrecision4)
9 axarr[1].set_ylabel('Precision (%)')
10 axarr[2].scatter(range(20), ResRecall4)
11 axarr[2].set_ylabel('Recall ')
12 axarr[3].scatter(range(20), ResLoss4)
13 axarr[3].set_ylabel('Zero-to-one Loss')
14 axarr[4].scatter(range(20), ResTimeTraining4)
15 axarr[4].set_ylabel('Training Time in sec')
16 axarr[5].scatter(range(20), ResTimePrediction4)
17 axarr[5].set_ylabel('Prediction Time in sec')
18
19 plt.show()
```

```
1  # VARIATION DU alpha (VALEUR DE REGULISATION L2)
2  import numpy as np
3
4  alphas = np.logspace(-5,3,5)
5
6
7  ResScore5 = []
8  ResPrecision5 = []
9  ResRecall5 = []
10 ResLoss5 = []
11 ResTimeTraining5 = []
12 ResTimePrediction5 = []
13
14 def runClassActiv(numTuple, a, i):
15     # Train
16     clf = MLPClassifier(hidden_layer_sizes=numTuple, alpha= a)
17     startTrain = time.time()
18     clf.fit(xtrain, ytrain)
19     endTrain = time.time()
20     # Predict
21     startpred = time.time()
22     predict = clf.predict(xtest)
23     endpred = time.time()
24     # Metrics
25     score = clf.score(xtest,ytest)
26     precision = metrics.precision_score(ytest, predict, average='macro')
27     recall = metrics.recall_score(ytest, predict, average='macro')
28     loss01 = metrics.zero_one_loss(ytest, predict)
29     timetrain = endTrain - startTrain
30     timePred = endpred - startpred
31     #Append
32     ResScore5.append(score*100)
33     ResPrecision5.append(precision*100)
34     ResRecall5.append(recall)
35     ResLoss5.append(loss01)
36     ResTimePrediction5.append(timePred)
37     ResTimeTraining5.append(timetrain)
38     #Print
39     print ("ALPHA : ", a)
40     print("pour le ", i,"eme modèle, score = ", score*100, "%, précision =",pr
41     print("      temps apprentissage : ", timetrain, "sec , temps prediction = "
42
43
44 for j in alphas:
45     runClassActiv(tuple1, j, 1)
46 for j in alphas:
47     runClassActiv(tuple3, j, 2)
48 for j in alphas:
49     runClassActiv(tuple5, j, 3)
50 for j in alphas:
51     runClassActiv(tuple7, j, 4)
52 for j in alphas:
53     runClassActiv(tuple9, j, 5)
```

Entrée [\*]:



```
1 # pour garder une trace des résultats et ne pas runner encore une fois
2 print(ResScore5)
3 #[97.74761904761905, 96.91428571428573, 97.49047619047619, 96.46190476190476,
4 print(ResPrecision5)
5 #[97.73118107142945, 96.96515283161631, 97.51034153327748, 96.48304373340724,
6 print(ResRecall5)
7 #[0.9771299357141972, 0.9684367263157749, 0.9744505493947285, 0.96430507191253
8 print(ResLoss5)
9 #[0.0225238095238095, 0.030857142857142805, 0.02509523809523806, 0.03538095238
10 print(ResTimeTraining5)
11 #[62.78313994407654, 70.90343022346497, 83.05095362663269, 124.70159530639648,
12 print(ResTimePrediction5)
13 #[0.29720568656921387, 0.3341064453125, 0.365023136138916, 0.29421257972717285
14
15 len(ResScore5)
```

Entrée [\*]:



```
1 # ordre des données :
2 # Clf1 alpha : 1^-5, 0.001, 0.1, 10, 1000 | Clf3 ... | Clf5 ... | Clf7 ... | C
3
4 fig, axarr = plt.subplots(6, sharex=True, figsize=(10,10))
5 axarr[0].scatter(range(25), ResScore5)
6 axarr[0].set_title('The five classifiers with different alpha : 1^-5, 0.001,
7 axarr[0].set_ylabel('Score (%)')
8 axarr[1].scatter(range(25), ResPrecision5)
9 axarr[1].set_ylabel('Precision (%)')
10 axarr[2].scatter(range(25), ResRecall5)
11 axarr[2].set_ylabel('Recall ')
12 axarr[3].scatter(range(25), ResLoss5)
13 axarr[3].set_ylabel('Zero-to-one Loss')
14 axarr[4].scatter(range(25), ResTimeTraining5)
15 axarr[4].set_ylabel('Training Time in sec')
16 axarr[5].scatter(range(25), ResTimePrediction5)
17 axarr[5].set_ylabel('Prediction Time in sec')
18
19 plt.show()
```

Entrée [\*]:



```
1 # CHOISIR LE MODELE AVEC LES MEILLEURS RESULTATS
2
3 #TODO + rejouer avec les param sélectionnés
```

Entrée [\*]:



```
1 # MEILLEUR MLP
2 # réseau 3 (50, 100, 200, 100,50)
3 # Relu et Adam (par défaut)
4 # petit alpha 0,1
5
6 #randomise Data et target
7 indices = np.random.randint(70000, size=70000)
8 data = mnist.data[indices]
9 target = mnist.target[indices]
10 # on veut un training set de 49000
11 xtrain, xtest, ytrain, ytest =train_test_split(data, target, train_size=49000)
12
13 clf3 = MLPClassifier(hidden_layer_sizes=(50, 100, 200, 100,50),alpha= 0.1) # a
14 startTrain =time.time()
15 clf3.fit(xtrain, ytrain)
16 endTrain = time.time()
17 # Predict
18 startpred= time.time()
19 predict = clf3.predict(xtest)
20 endpred = time.time()
21 score = clf3.score(xtest,ytest)
22
23 recall = metrics.recall_score(ytest, predict, average ='macro')
24 precision = metrics.precision_score(ytest, predict, average='macro')
25 loss01 = metrics.zero_one_loss(ytest, predict)
26 timetrain = endTrain - startTrain
27 timePred = endpred - startpred
28
29
30 print("Ce modèle de MLP, d'1 couche de 50, à un score de ", score*100, "%.")
31 print("4eme image : prédiction ",predict[3], "reel : ", ytest[3])
32 print ("ce modèle de MLP à une précision de", precision*100, "%.")
33 print ("ce modèle de MLP à un recall de",recall*100, "%.")
34 print ("ce modèle de MLP à un zero-one_loss de",recall*100, "%.")
35 print("    temps apprentissage : ", timetrain, "sec , temps prediction = ", ti
36
```

Entrée [\*]:



```
1 # AVANTAGES ET INCONVEGNIENS :
2 # Optimalité ?
3 # tps de calcul ?
4 # Passage à l'échelle ?
```

Entrée [ ]:



```
1
```

