Perceptron與多層感知器(Multi-Layer Perceptrons)

林郁修

Yu-Hsiu Lin
yhlin@ntut.edu.tw
https://sites.google.com/view/aici

Graduate Institute of Automation Technology,
National Taipei University of Technology

Outline

- 機器學習的架構
- 感知器 (Perceptron)
- 感知器的學習
- 感知器 VS. 多層感知器 (Multi-Layer Perceptrons)
- 多層感知器的學習

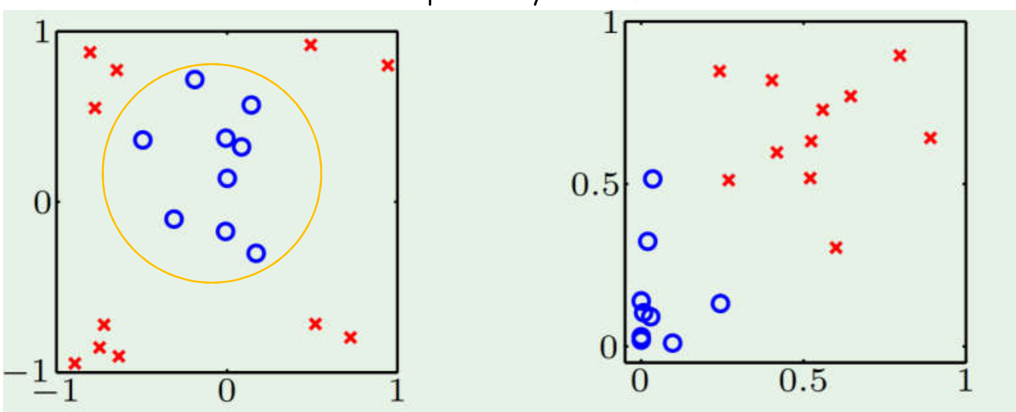
Outline

- 機器學習的架構
- 感知器 (Perceptron)
- 感知器的學習
- 感知器 VS. 多層感知器 (Multi-Layer Perceptrons)
- 多層感知器的學習

(feature data transformation)

- To see, to believe …
  - z-space

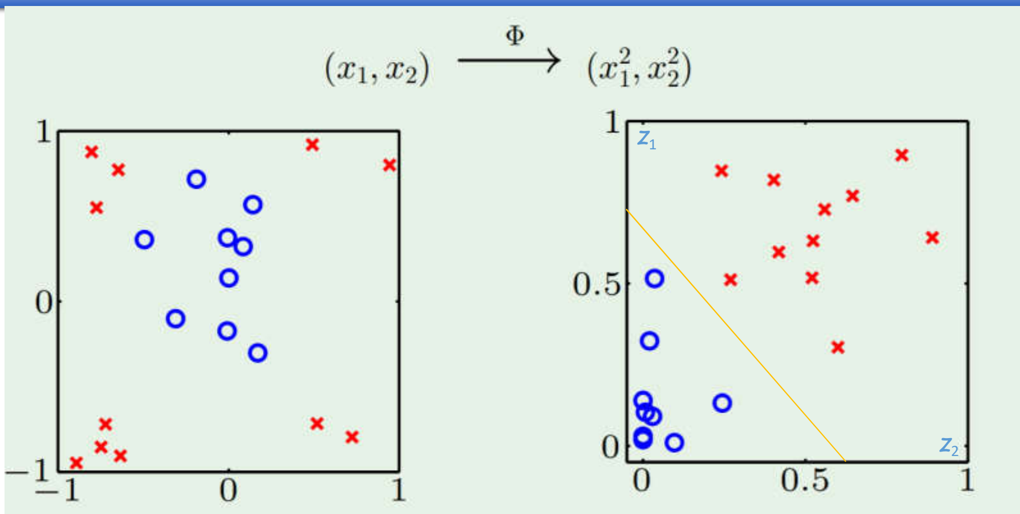
The original data are not linearly separable, but separable by a circle.



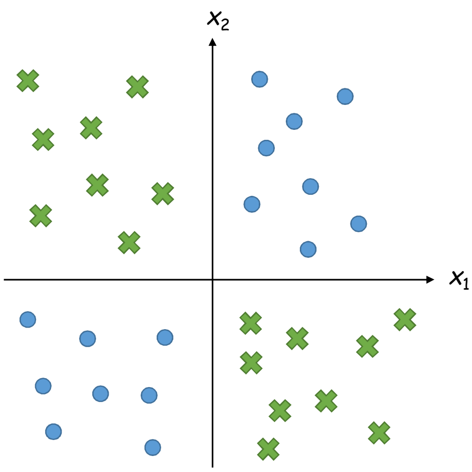
$x_1^2 + x_2^2 = 0.6$

$0.6 + (-1) \cdot \underbrace{x_1^2}_{z_1} + (-1) \cdot \underbrace{x_2^2}_{z_2}$

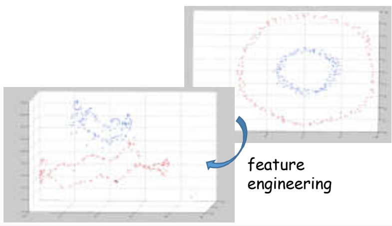
Transform the data nonlinearly



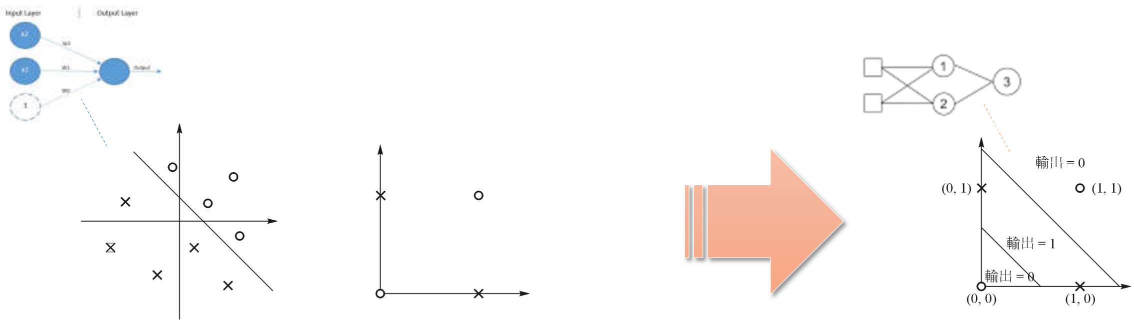
(feature data transformation/feature crossing)



x_3 = ?



感知器 VS. 多層感知器 (Multi-Layer Perceptrons)

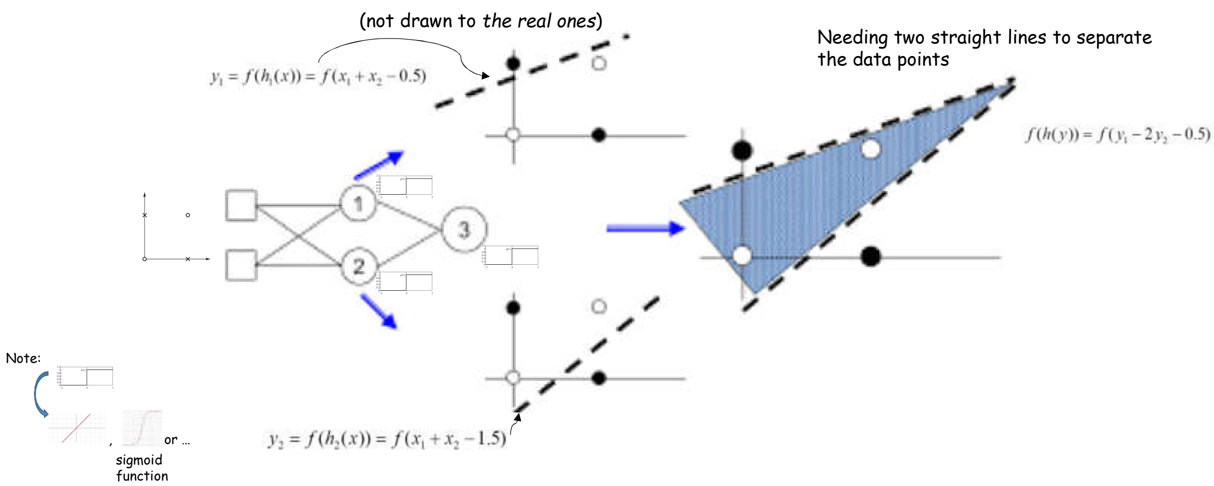


感知器將〇及╳分成兩類別 (資料分佈為線性可分離)

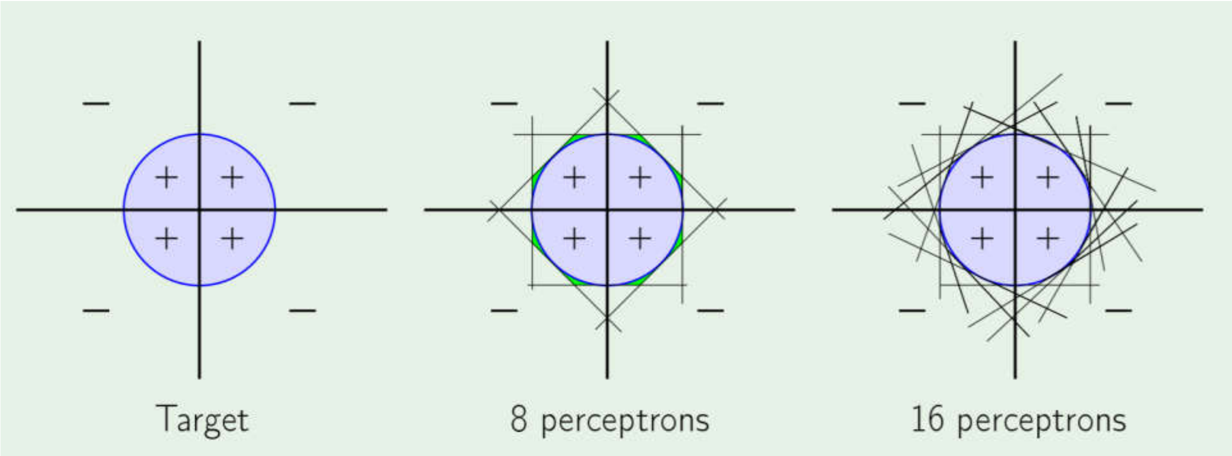
感知器無法解決簡單的*互斥或* (XOR)問題

多層感知器(多層前向式網路)可以解決XOR問題

(feature data transformation)

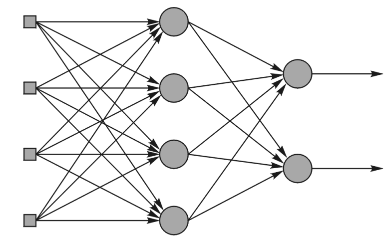


(定義成本函數 (X->Y擬合誤差最小) → 負的梯度方向於權重係數更新)



多層感知器

- The multilayer perceptron/MLP was presented by Rumelhart and McClelland in 1986.
  - 多層前饋式類神經網路 **(Multilayer Feedforward Networks)**
  - 所使用的學習演算法為誤差倒傳遞**(Back-propagation, BP)**演算法



Note: 定義成本函數 (X->Y擬合誤差最小) → 負的梯度方向於權重係數更新

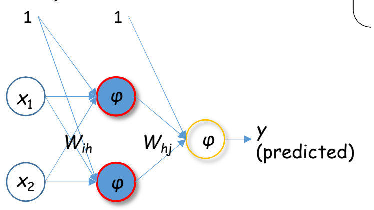
https://numpy.org/doc/stable/reference/generated/numpy.matmul.html

An example

patterns:

x_1	x_2	y (desired)
.1	.1	.1
.1	.95	.95
.95	.1	.95
.95	.95	.1

- A well-trained simple MLP



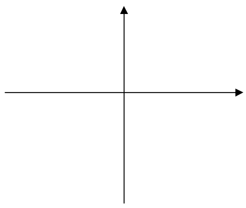
$$[1 \;\; x_1 \;\; x_2] \quad W_{ih} = \begin{pmatrix} 5.32 & 3.13 \\ -3.73 & -6.67 \\ -3.72 & -6.61 \end{pmatrix} \quad W_{hj} = \begin{pmatrix} -3.08 \\ 7.30 \\ -7.30 \end{pmatrix}$$

$\varphi(v)$: sigmoid activation function

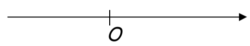
1/(1 + exp(-v))

S_h Hidden neuron output signals?

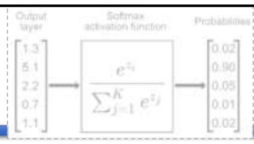
[1 , ,]



S_y Output neuron signal?



Another example



patterns:

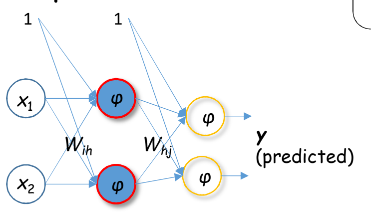
x_1	x_2	y_1	y_2
.1	.1	.95	.1
.1	.95	.1	.95
.95	.1	.1	.95
.95	.95	.95	.1

y (desired)

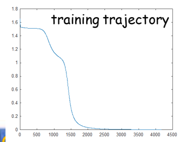
等同於labeling為 [1 0]

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

- A well-trained simple MLP



$$[1 \;\; x_1 \;\; x_2] \quad W_{ih} = \begin{pmatrix} 2.73 & 3.51 \\ -6.37 & 7.89 \\ 6.60 & -7.92 \end{pmatrix} \quad W_{hj} = \begin{pmatrix} -8.03 & 8.44 \\ 5.71 & -5.59 \\ 5.55 & -5.44 \end{pmatrix}$$

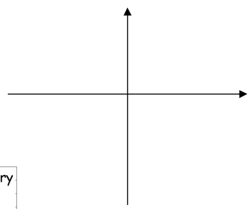


$\varphi(v)$: sigmoid activation function

1/(1 + exp(-v))

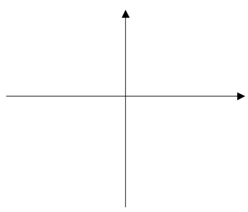
S_h Hidden neuron output signals?

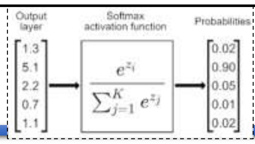
[1 , ,]



S_y Output neuron signal?

[,]





- 分類 (離散型)

VS.

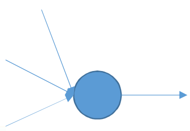
- 函數近似 (連續型)
  - 分類： Decision thresholds / Winner-take-all rule / Probabilities (e.g., by Softmax)

Perceptron() -- Single Layer Perceptron (VS. MLP) -- I

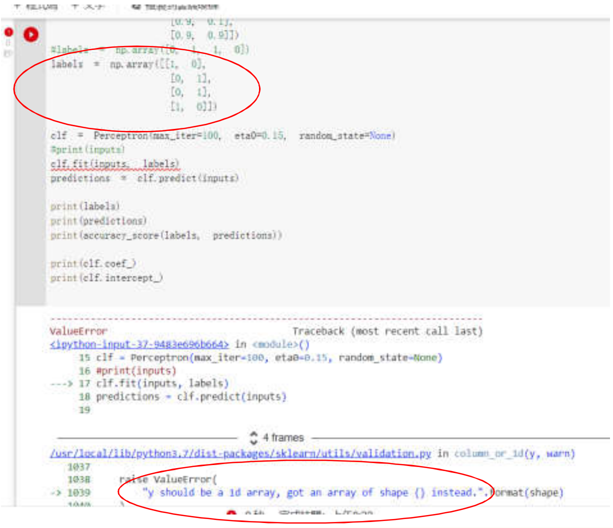
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

```
• import numpy as np
• from sklearn.linear_model import Perceptron
• from sklearn.metrics import accuracy_score

• inputs = np.array([[0.1, 0.1],
•         [0.1, 0.9],
•         [0.9, 0.1],
•         [0.9, 0.9]])
• labels = np.array([0, 1, 1, 0])

• clf = Perceptron(max_iter=100, eta0=0.15, random_state=None)
• print(inputs)
• clf.fit(inputs, labels)
• predictions = clf.predict(inputs)

• print(labels)
• print(predictions)
• print(accuracy_score(labels, predictions))

• print(clf.coef_)
• print(clf.intercept_)
```

```
[0 1 1 0]
[1 1 1 0]
0.75
[[-0.105 -0.105]]
[0.15]
```



Perceptron() -- Single Layer Perceptron (VS. MLP) -- I (Cont'd)



https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

Perceptron() -- Single Layer Perceptron (VS. MLP) -- II

- from sklearn import datasets
- from sklearn.preprocessing import StandardScaler
- from sklearn.model_selection import train_test_split
- from sklearn.linear_model import Perceptron
- from sklearn.metrics import accuracy_score

- iris = datasets.load_iris()

- *#features = iris.data*
- features = iris.data[:,2:4]
- target = iris.target

- features_train, features_test, target_train, target_test = train_test_split(features, target, test_size=0.2)

(Cont'd)

- *#sc = StandardScaler()*
- *#sc.fit(features_train)*

- *#features_train_std = sc.transform(features_train) # feature transformation*
- *#features_test_std = sc.transform(features_test)*

(Cont'd)

- clf = Perceptron(max_iter=100, eta0=0.15, random_state=None)
- *#clf.fit(features_train_std, target_train)*
- clf.fit(features_train, target_train)

(Cont'd)

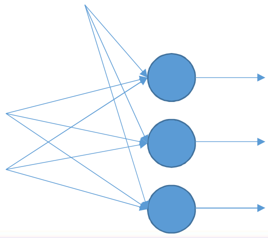
- *#target_pred = clf.predict(features_test_std)*
- target_pred = clf.predict(features_test)
- print(accuracy_score(target_test, target_pred))

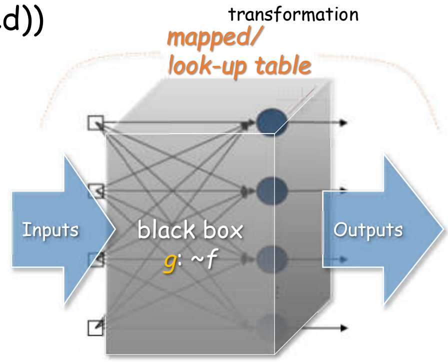
- print(clf.coef_)
- print(clf.intercept_)

相較於Perceptron，Single Layer Perceptron實踐較複雜的feature data transformation

```
#target_pred = clf.predict(features_test_std)
target_pred = clf.predict(features_test)
print(accuracy_score(target_test, target_pred))


print(clf.coef_)
print(clf.intercept_)
```

```
0.8333333333333334
[[-0.105 -0.33 ]
 [ 1.245 -2.475]
 [-0.225  2.97 ]]
[ 0.6  -1.65 -4.2 ]
```



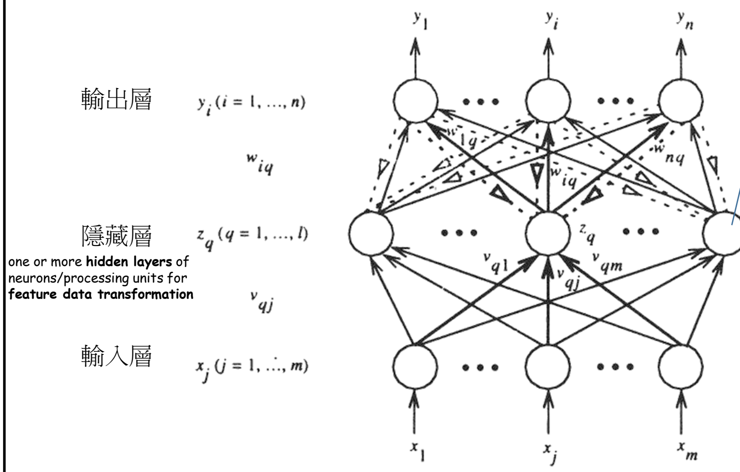


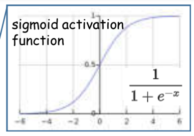
Outline

- 機器學習的架構
- 感知器 (Perceptron)
- 感知器的學習
- 感知器 VS. 多層感知器 (Multi-Layer Perceptrons)
- 多層感知器的學習

多層感知器的學習

- 倒傳遞類神經網路 (Back-propagation, BP)





$$z_q = a(net_q) = a(\sum_{j=1}^{m} v_{qj} x_j)$$

$$E(\mathrm{w}) = \frac{1}{2}\sum_{i=1}^{n}(d_i - y_i)^2 = \frac{1}{2}\sum_{i=1}^{n}[d_i - a(net_i)]^2 = \frac{1}{2}\sum_{i=1}^{n}[d_i - a(\sum_{q=1}^{l} w_{iq} z_q)]^2$$

$$\Delta w_{iq} = -\eta\left[\frac{\partial E}{\partial y_i}\right]\left[\frac{\partial y_i}{\partial net_i}\right]\left[\frac{\partial net_i}{\partial w_{iq}}\right] = \eta[d_i - y_i][a'(net_i)][z_q] \underline{\underline{\Delta}} \eta\delta_{oi} z_q$$

$$\delta_{oi} \underline{\underline{\Delta}} - \frac{\partial E}{\partial net_i} = -\left[\frac{\partial E}{\partial y_i}\right]\left[\frac{\partial y_i}{\partial net_i}\right] = [d_i - y_i][a'(net_i)]$$

$$\Delta v_{qj} = \eta\sum_{i=1}^{n}[\delta_{oi} w_{iq}] a'(net_q) x_j = \eta\delta_{hq} x_j$$

$$\delta_{hq} \underline{\underline{\Delta}} - \frac{\partial E}{\partial net_q} = -\left[\frac{\partial E}{\partial z_q}\right]\left[\frac{\partial z_q}{\partial net_q}\right] = a'(net_q)\sum_{i=1}^{n}\delta_{oi} w_{iq}$$

- BP

- https://github.com/Jaewan-Yun/optimizer-visualization

An example -- Iris flowers classification based on MLPClassifier()

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

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score


iris_dataset = load_iris()
data = pd.DataFrame(iris_dataset.data, columns=iris_dataset.feature_names)
data['target'] = iris_dataset.target


features = data.drop('target', axis=1)
x_train, x_test, y_train, y_test = train_test_split(features, data['target'], test_size=0.25)


clf = MLPClassifier(hidden_layer_sizes=(8,), activation='tanh', solver='adam', learning_rate='constant', learning_rate_init=0.2)
clf.fit(x_train, y_train)
```



```
print(accuracy_score(y_test, clf.predict(x_test)))


print(y_train[1:6])


print(clf.coefs_)
print(clf.intercepts_)
```

```
0.9736842105263158
131    2
23     0
46     0
133    2
6      0
Name: target, dtype: int64
[array([[-0.8161652 , -1.19753879,  0.01145573,  1.02520302,  1.54298311,
         1.24343254, -0.95058383, -1.39859438],
       [-0.67494661, -0.58498357, -1.31826164,  2.13243429,  0.59138936,
         0.72801573, -0.73936458, -1.38636554],
       [-0.50538357, -1.17837592,  1.49475955,  1.84635876,  1.08004668,
         0.38946904,  1.39109698, -0.33248272],
       [ 0.00880586, -0.44441948,  2.01590339,  1.53603984,  0.57910507,
         0.0076436 ,  1.93626005, -1.27232472]]), array([[-0.65091511, -0.60257541, -0.34867521],
       [ 1.91965168, -1.2501208 ,  1.05542322],
       [-4.95386823,  1.83581283,  1.98822023],
       [ 0.67015812, -1.0986102 ,  1.06318613],
       [-0.13448371, -0.11026113, -0.22961601],
       [ 0.27504222,  0.17916816, -0.10053396],
       [-3.52703483, -1.64380386,  6.01350257],
       [ 0.01873228, -0.88516726,  1.63203101]])]
[array([-1.73661696, -0.32512209, -1.59832743,  1.8314011 ,  1.67534562,
        0.52805801, -1.62242794, -0.490211  ]), array([ 0.12558752, -0.1860496 ,  0.71661654])]
```

Install User Guide API Examples Community More

Prev Up Next

scikit-learn 1.0.2
Other versions

Please **cite us** if you use the software.

sklearn.neural_network.MLPClassifier
Examples using sklearn.neural_network.MLPClassifier

常見之
超參數

sklearn.neural_network.MLPClassifier

class sklearn.neural_network.MLPClassifier(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) [source]

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*
The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*
The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Prev Up Next

scikit-learn 1.0.2
Other versions

Please **cite us** if you use the software.

sklearn.neural_network.MLPClassifier
Examples using sklearn.neural_network.MLPClassifier

alpha : *float, default=0.0001*
L2 penalty (regularization term) parameter.

batch_size : *int, default='auto'*
Size of minibatches for stochastic optimizers. If the solver is 'lbfgs', the classifier will not use minibatch. When set to "auto", `batch_size=min(200, n_samples)`.

learning_rate : *{'constant', 'invscaling', 'adaptive'}, default='constant'*
Learning rate schedule for weight updates.

- 'constant' is a constant learning rate given by 'learning_rate_init'.
- 'invscaling' gradually decreases the learning rate at each time step 't' using an inverse scaling exponent of 'power_t'. effective_learning_rate = learning_rate_init / pow(t, power_t)
- 'adaptive' keeps the learning rate constant to 'learning_rate_init' as long as training loss keeps decreasing. Each time two consecutive epochs fail to decrease training loss by at least tol, or fail to increase validation score by at least tol if 'early_stopping' is on, the current learning rate is divided by 5.

Only used when `solver='sgd'`.

learning_rate_init : *float, default=0.001*
The initial learning rate used. It controls the step-size in updating the weights. Only used when solver='sgd' or 'adam'.

power_t : *float, default=0.5*
The exponent for inverse scaling learning rate. It is used in updating effective learning rate when the learning_rate is set to 'invscaling'. Only used when solver='sgd'.

max_iter : *int, default=200*
Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd', 'adam'), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.

shuffle : *bool, default=True*
Whether to shuffle samples in each iteration. Only used when solver='sgd' or 'adam'.

隨機梯度下降法(Stochastic gradient descent, SGD)：一次輸入一個訓練樣本或是小批次(mini-batch)的樣本, 然後計算一次梯度或是小批次梯度的平均更新一次神經網路的權重; 該樣本或是小批次的樣本是隨機抽取/輸入的

Toggle Menu

- MLPClassifier() API參數重點摘要/補充說明如下：
  - solver='lbfgs'：與梯度下降法(sgd、adam)相關的API參數，即：batch_size、learning_rate、power_t、momentum、nesterovs_momentum、beta_1、beta_2、epsilon，無效（API將該些參數自動屏蔽/disabled）
  - solver='lbfgs'：透過調整API參數max_fun (vs. max_iter)、tol (with n_iter_no_change)，決定MLP模型的學習（多少次的迭代及其loss變化量小到何種程度）
  - API參數learning_rate內文敘述 "Only used when solver='sgd'."應該是針對當learning_rate='invscaling'or'adaptive'的情況下
  - 不管solver='sgd'or'adam'：API參數batch_size（batch_size=min(200, n_samples)）預設值為200（n_samples為資料總筆數）。當設定的batch_size介於1與n_samples之間，solver即為mini-batch stochastic gradient-based weight optimizer (for sgd or adam)。於此同時，所設定的max_iter即為模型的總學習次數epochs（每一筆資料將被模型反覆學習達多少次）（epochs即為loss歷程曲線training trajectory的橫軸），且在每一epoch模型的權重會被更新共batch_size次（亦即，達batch_size個iterations or gradient steps）

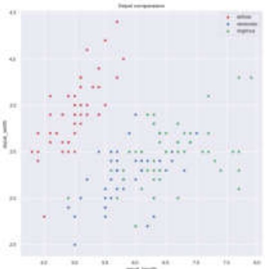
    ps., 確切地說，n_samples應為訓練資料總筆數

    ps., 若batch_size為資料總筆數，其中：資料總筆數小於200，則每一epoch涉及共一次的iteration/gradient step（一整個資料集批次學習）。且，所設定的max_iter即為模型的總學習次數epochs（此學習形式即為minibatch，且其涉及mean gradient）

    ps., 觀念上（MLPClassifier()應無法進行逐筆資料的學習），若batch size為1，則每一epoch涉及資料總筆數個iterations (i.e., gradient steps)
  - 不管solver='lbfgs'or'sgd'or'adam'：可避免模型過度學習的API參數包含alpha、early_stopping (with validation_fraction)。若不清楚該些參數所相關的機制，則預設值即可
  - 當solver='lbfgs'，API參數beta_1、beta_2、epsilon的設定會決定其成效。若設定solver='lbfgs'，則建議去了解其最佳化權重的機制於該些參數的設定（通常是case by case）

Assignment #2

- 基於MLP，分類作業#1之iris flowers資料
- Notes
  - 特徵選取於"越不具有代表性"之二維特徵變量使用：e.g., sepal length vs. sepal width or …



  - 輸出層神經元個數: 3（隱藏層1~2層即可）

- 承Assignment #1實作經驗，完整MLP分類器的學習流程
- 說明所訓練之MLP網路架構與其之超參數設定
- 視覺化所完成訓練之MLP的輸出資料散佈圖 (以不同的顏色顯示不同類別的資料)

  (共150筆資料)

- Due：5/17(三) 23:59上傳i學園
- Oral：5/18(四)