Introduction to TensorFlow

Fully Connected Deep Networks with TensorFlow

郭耀仁

大綱

- 關於 Fully Connected Deep Networks
- 取得資料
- Benchmark
- 建構 TensorFlow 計算圖形
- 訓練
- 隨堂練習

關於 Fully Connected Deep Networks

Fully Connected Deep Networks 多層感知器

單一個感知器（Perceptron）的公式：

$$y = \sigma(WX + b)$$

σ 即所謂的激活函數（activation function）

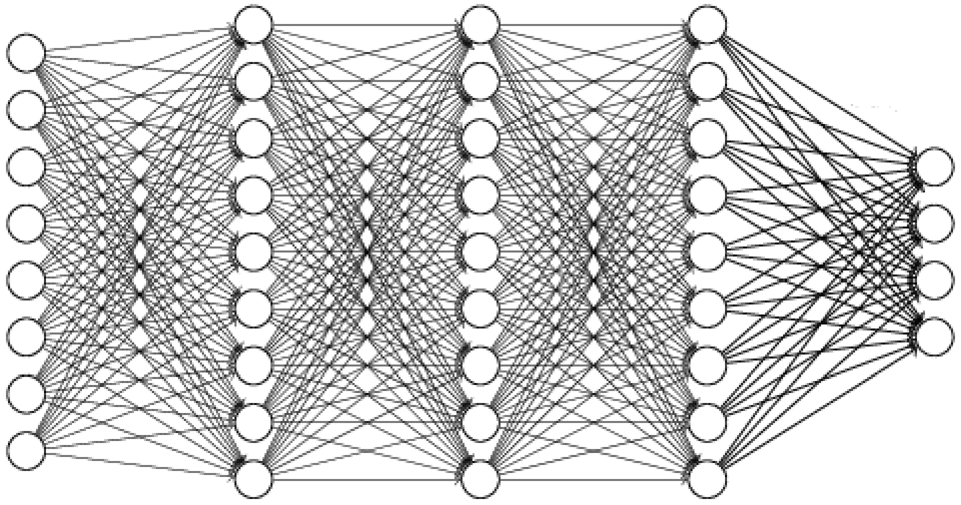
激活函數用來將線性關係映射至非線性關係

常用的激活函數：

- Sigmoid
- TanH
- ReLU

連結多個感知器建構出多層感知器，透過 backpropagation 求出係數

- 輸入層
- 隱藏層
- 輸出層



Source: https://math.stackexchange.com/questions/2048722/a-name-for-layered-directed-graph-as-in-a-fully-connected-neural-network (https://math.stackexchange.com/questions/2048722/a-name-for-layered-directed-graph-as-in-a-fully-connected-neural-network)

取得資料

簡單、作為測試目的即可

Scikit-Learn Breast Cancer 資料集

```
In [1]: from sklearn.datasets import load_breast_cancer

        breast_cancer = load_breast_cancer()
        print(breast_cancer.feature_names)
        print(breast_cancer.DESCR)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter

        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
```

```
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extractio
n
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis a
nd
     prognosis via linear programming. Operations Research, 43(4), pages 570-5
77,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniq
ues
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77
(1994)
     163-171.
```

In [2]:

```
X_arr = breast_cancer.data
y_arr = breast_cancer.target
print(X_arr.shape)
print(y_arr.shape)
```

```
(569, 30)
(569,)
```

```
In [3]: from sklearn.model_selection import train_test_split

        X_train, X_test, y_train, y_test = train_test_split(X_arr, y_arr, test_size=0.3, r
        andom_state=123)
        print(X_train.shape)
        print(X_test.shape)
        print(y_train.shape)
        print(y_test.shape)
```

```
(398, 30)
(171, 30)
(398,)
(171,)
```

Benchmark

In [4]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

clf = LogisticRegression(solver="liblinear")
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
```

In [5]:
```
print(clf.coef_)
print(clf.intercept_)
print(acc)
```

```
[[ 1.93815613e+00  1.95132301e-01 -2.79314889e-03 -9.98660343e-03
  -1.31191303e-01 -3.94395839e-01 -5.87981725e-01 -3.08507409e-01
  -2.34633450e-01 -2.07557569e-02 -1.29122502e-03  1.25273443e+00
   8.82364969e-03 -8.58922482e-02 -1.44638024e-02 -2.16113982e-04
  -4.51757012e-02 -3.74554453e-02 -3.87686384e-02  6.76008093e-03
   1.00073745e+00 -3.88446001e-01 -1.28867607e-01 -1.78769588e-02
  -2.50482381e-01 -1.09777946e+00 -1.45824047e+00 -6.35683325e-01
  -7.02842419e-01 -9.58336644e-02]]
[0.41220102]
0.9824561403508771
```

Benchmark 完整程式碼

```
In [6]: from sklearn.datasets import load_breast_cancer
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score

        breast_cancer = load_breast_cancer()
        X_arr = breast_cancer.data
        y_arr = breast_cancer.target
        X_train, X_test, y_train, y_test = train_test_split(X_arr, y_arr, test_size=0.3, r
        andom_state=123)
        clf = LogisticRegression(solver="liblinear")
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
```

建構 TensorFlow 計算圖形

準備 Placeholders 作為 Input layers

In [7]:
```
import tensorflow as tf

n_features = X_train.shape[1]
with tf.name_scope("input-layer"):
  X = tf.placeholder(tf.float32, (None, n_features)) # ?
  y = tf.placeholder(tf.float32, (None,))            # ?
```

Placeholders 的觀測值個數設定為 None 有什麼好處?

- 多層感知器的應用常會使用 **Mini-batching** 的技巧
- 訓練結束之後可以餵入 X_test 獲得 y_pred_arr

準備 Variables 作為 Hidden layers

In [8]:
```
import numpy as np

n_classes = np.unique(y_train).size
n_neurons = int((n_features + n_classes) / 2)
with tf.name_scope("hidden-layer"):
  W = tf.Variable(tf.random_normal((n_features, n_neurons))) # (30, 16)
  b = tf.Variable(tf.random_normal((n_neurons,)))            # (16,)
  X_hidden = tf.nn.relu(tf.matmul(X, W) + b)                 # (398, 30) x (30, 1
6) + (16,)
```

如何決定 hidden-layer 的層數與神經元數量?

- 絕大多數情境：使用一層 hidden-layer 就足夠
- 神經元數量：(輸入層神經元個數 + 輸出層神經元個數) / 2

```
n_neurons = int((n_features + n_classes) / 2)
```

準備 Variables 作為 Output layers

In [9]:
```
with tf.name_scope("output-layer"):
  W = tf.Variable(tf.random_normal((n_neurons, 1))) # (16, 1)
  b = tf.Variable(tf.random_normal((1,)))           # (1,)
  y_logit = tf.squeeze(tf.matmul(X_hidden, W) + b)  # (398, 16) x (16, 1) + (1,)
  y_one_prob = tf.sigmoid(y_logit)
  y_pred = tf.round(y_one_prob)
```

寫下成本函數的公式

```
In [10]: with tf.name_scope("loss"):
           entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit, labels=y)
           loss = tf.reduce_sum(entropy)
```

宣告 Optimizer 與學習速率

In [11]:
```
learning_rate = 0.01
with tf.name_scope("optimizer"):
  opt = tf.train.AdamOptimizer(learning_rate)
  optimizer = opt.minimize(loss)
```

建構 TensorFlow 計算圖形完整程式碼

In [12]:
```
import tensorflow as tf
import numpy as np

tf.reset_default_graph()
n_features = X_train.shape[1]
n_classes = np.unique(y_train).size
n_neurons = int((n_features + n_classes) / 2)
learning_rate = 0.0001

with tf.name_scope("input-layer"):
  X = tf.placeholder(tf.float32, (None, n_features))
  y = tf.placeholder(tf.float32, (None,))
with tf.name_scope("hidden-layer"):
  W = tf.Variable(tf.random_normal((n_features, n_neurons))) # (30, 16)
  b = tf.Variable(tf.random_normal((n_neurons,)))            # (16,)
  X_hidden = tf.nn.relu(tf.matmul(X, W) + b)                 # (398, 30) x (30, 1
6) + (16,)
with tf.name_scope("output-layer"):
  W = tf.Variable(tf.random_normal((n_neurons, 1)))          # (16, 1)
  b = tf.Variable(tf.random_normal((1,)))                    # (1,)
  y_logit = tf.squeeze(tf.matmul(X_hidden, W) + b)           # (398, 16) x (16, 1)
+ (1,)
  y_one_prob = tf.sigmoid(y_logit)
  y_pred = tf.round(y_one_prob)
with tf.name_scope("loss"):
  entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit, labels=y)
  loss = tf.reduce_sum(entropy)
with tf.name_scope("optimizer"):
  opt = tf.train.AdamOptimizer(learning_rate)
  optimizer = opt.minimize(loss)
```

訓練

In [13]:
```
n_steps = 50000
file_writer_path = "./graphs/fully-connected-deep-networks"
loss_history = []

with tf.Session() as sess:
  sess.run(tf.global_variables_initializer()) # 初始化所有的變數張量!
  train_writer = tf.summary.FileWriter(file_writer_path, tf.get_default_graph())
  for i in range(n_steps):
    feed_dict = {
      X: X_train,
      y: y_train
    }
    _, loss_ = sess.run([optimizer, loss], feed_dict=feed_dict)
    loss_history.append(loss_)
    if i % 1000 == 0:
      print("step {}, loss: {}".format(i, loss_))
  w_final, b_final = sess.run([W, b])
  y_pred_arr = sess.run(y_pred, feed_dict={X: X_test})
```

```
step 0, loss: 414025.875
step 1000, loss: 158248.84375
step 2000, loss: 2681.51953125
step 3000, loss: 2467.9853515625
step 4000, loss: 2411.2861328125
step 5000, loss: 2320.51806640625
step 6000, loss: 2178.854736328125
step 7000, loss: 1970.0159912109375
step 8000, loss: 1686.59521484375
step 9000, loss: 1358.921875
step 10000, loss: 987.6146850585938
step 11000, loss: 609.17138671875
step 12000, loss: 372.76959228515625
step 13000, loss: 255.7460021972656
step 14000, loss: 155.87393188476562
step 15000, loss: 118.18053436279297
step 16000, loss: 94.63573455810547
step 17000, loss: 73.91920471191406
```

```
In [14]: import matplotlib.pyplot as plt

         plt.plot(range(n_steps), loss_history)
         plt.title("Loss Summary")
         plt.xlabel("Epochs")
         plt.ylabel("Loss")
         plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

```
In [15]: acc = accuracy_score(y_test, y_pred_arr)
         print(w_final)
         print(b_final)
         print(acc)
```

```
[[-0.7065138 ]
 [-0.66401774]
 [-2.1829813 ]
 [ 1.1565275 ]
 [-0.40028438]
 [ 0.9624974 ]
 [-0.2515118 ]
 [-0.6297551 ]
 [-1.0843177 ]
 [ 0.5629655 ]
 [ 1.0997338 ]
 [-1.6978433 ]
 [ 1.5582514 ]
 [-0.769502  ]
 [ 1.513353  ]
 [-2.228517  ]]
[0.8738572]
0.9824561403508771
```

隨堂練習

使用 titanic 資料集並利用 TensorFlow 建立一個 Fully Connected Deep Networks Classifier

In [16]:

```
import pandas as pd

train_url = "https://s3-ap-northeast-1.amazonaws.com/kaggle-getting-started/titani
c/train.csv"
test_url = "https://s3-ap-northeast-1.amazonaws.com/kaggle-getting-started/titani
c/test.csv"
train_df = pd.read_csv(train_url)
test_df = pd.read_csv(test_url)
```