



深度學習TensorFlow實務

TensorFlow 常用 Ops

Lab1

-TA-

李偉弘

廖宜健

林佑昌

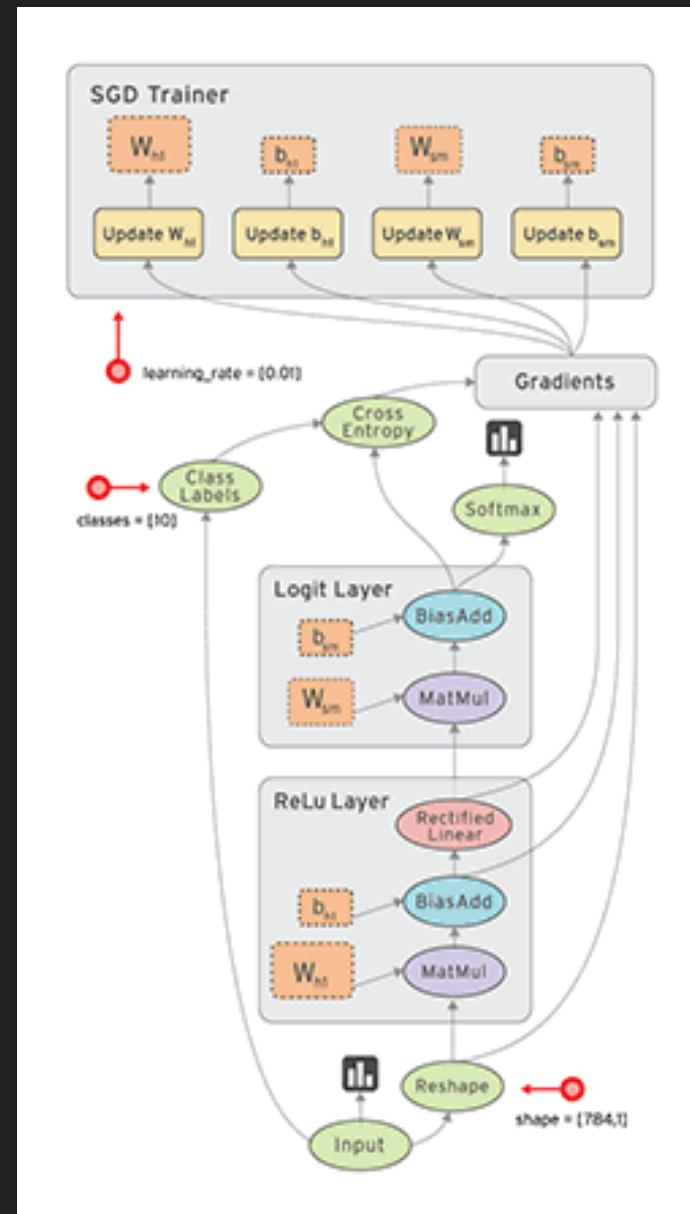
蔡明諺

彭冠偉

1. Basic Operations 介紹

Tensor & Flow

- TensorFlow使用數據流圖來根據各個操作之間的依賴關係來表示您的計算。
 1. 首先定義數據流圖(Dataflow Graph)。
 2. 創立Tensorflow會話(Session)，在本地端運行數據流圖。



張量 — Tensor

- 張量其實就是「陣列」的廣義說法。
- 可以是代表一維陣列(向量)、二維陣列(矩陣)、...、 N 維陣列(張量)。
- N 等於一的時候，就是向量； N 等於二，就是矩陣。意即「向量」和「矩陣」的稱呼，只是張量的特例。

流 — Flow

- 平常我們在寫得程式，也是一種「流」。宣告變數，然後傳遞變數，回傳，再繼續傳遞...最後輸出結果。
- 當程式啟動，資料就開始傳輸，直到程式結束。
- TensorFlow 的「流」和我們所習慣的「流」不太一樣。
TensorFlow 的流，需要把所有的節點設置好了以後，用「`sess.run()`」來啟動計算圖。

Tensor&flow

C語言

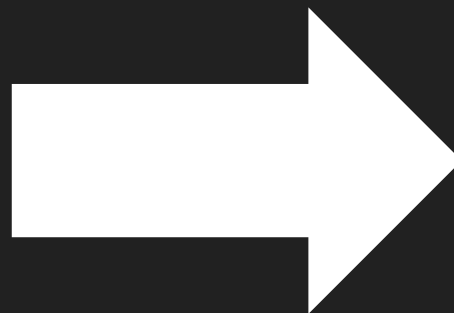
```
# include<stdio.h>
# include<stdlib.h>
int main()
{
    int A = 10;
    int B = 20;
    int C = A + B;
    printf("%d\n", C);
    system("pause");
}
```

TensorFlow

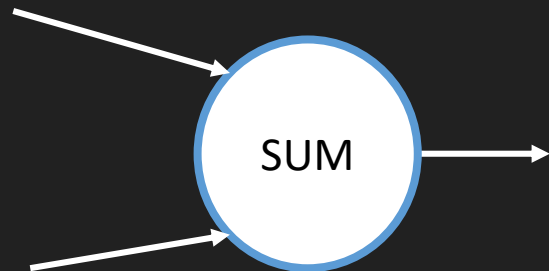
```
import tensorflow as tf
# 宣告常數 A = 50
A = tf.constant(50)
# 宣告常數 B = 100
B = tf.constant(100)
# 相加
C = A + B
# 啟動計算圖
with tf.Session() as sess:
    print(sess.run(C))
```

第一道TensorFlow程式

```
import tensorflow as tf
a = tf.constant(2)
b = tf.constant(3)
x = tf.add(a, b)
writer = tf.summary.FileWriter('./graphs',
tf.get_default_graph())
with tf.Session() as sess:
    print(sess.run(x))
```



?



$$2 + 3 = 5$$

第一道TensorFlow程式

```
import tensorflow as tf
```

```
a = tf.constant(2)
```

```
b = tf.constant(3)
```

建立常量

```
x = tf.add(a, b)
```

建立加法op，將 a 與 b 作為輸入

```
writer = tf.summary.FileWriter('./graphs',
```

圖形定義後寫入

```
tf.get_default_graph())
```

```
with tf.Session() as sess:
```

Session是圖和執行者之間的媒介，
首先透過Session來啟動圖

```
    print(sess.run(x))
```

把 Op 放入 sess.run(Op) 中才會開始執行。

Jupyter開啟第一道TensorFlow程式

- 在 cmd 視窗，輸入以下指令

■ `> activate tensorflow`

```
C:\>activate tensorflow  
(tensorflow) C:\>
```


Jupyter開啟第一道TensorFlow程式

- 在 cmd 視窗，輸入以下指令

■ `> jupyter notebook`

```
C:\>activate tensorflow  
(tensorflow) C:\>jupyter notebook
```

Jupyter開啟第一道TensorFlow程式

```
File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3 
```

```
In [ ]: import tensorflow as tf
        a = tf.constant(2)
        b = tf.constant(3)
        x = tf.add(a, b)
        writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())
        with tf.Session() as sess:
            print(sess.run(x))
```

視覺化-TensorBoard

- 在 cmd 視窗，輸入以下指令


> Tensorboard --logdir="./graphs"


```
C:\>activate tensorflow  
(tensorflow) C:\>tensorboard --logdir="./graphs"
```

視覺化-TensorBoard

TensorBoard **GRAPHS**

Search nodes. Regexes sup...

 Fit to screen

 Download PNG

Run

-

(1)

Session

-

runs (0)

Upload

Choose File

☐ Trace inputs

Color

☒ Structure

☐ Device

☐ XLA Cluster

☐ Compute time

☐ Memory


☐ TPU Compatibility

colors

same substructure

☐ unique substructure

Main GraphAuxiliary Nodes



視覺化-TensorBoard

```
import tensorflow as tf  
a = tf.constant(2)  
b = tf.constant(3)  
x = tf.add(a, b)  
writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())  
with tf.Session() as sess:  
    print(sess.run(x))
```



該如何更改張量名稱??

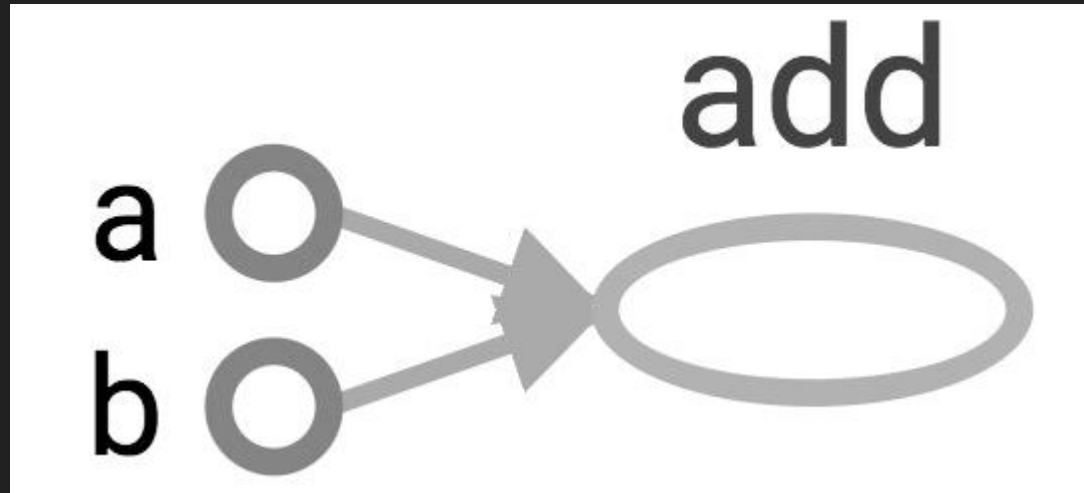
視覚化-TensorBoard

```
import tensorflow as tf

a = tf.constant(2, name='a')
b = tf.constant(3, name='b')
x = tf.add(a, b, name='add')

writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())

with tf.Session() as sess:
    print(sess.run(x))
```



張量(tensor)內指定數值

- 創建一個張量，所有元素都設置為零。返回具有 dtype 類型與 shape 大小的張量。

```
tf.zeros(shape, dtype=tf.float32, name=None)
```

```
tf.zeros([2, 3], tf.int32) ==> [[0, 0, 0], [0, 0, 0]]
```


給定張量內數值

- 建立一個與input_tensor同樣維度大小的張量且input_tensor內容為零

```
tf.zeros_like(input_tensor, dtype=None, name=None, optimize=True)
```

```
# input_tensor is [[0, 1], [2, 3], [4, 5]]
```

```
tf.zeros_like(input_tensor) ==> [[0, 0], [0, 0], [0, 0]]
```

```
tf.ones(shape, dtype=tf.float32, name=None)
```

```
tf.ones_like(input_tensor, dtype=None, name=None, optimize=True) 與上述方法相同
```

練習題

- 建立 x 張量其數值為 $\begin{bmatrix} 0 & -2 & -1 \\ 0 & 1 & 2 \end{bmatrix}$ ，而 y 張量大小形狀與 x 相同且裡面為0，最後以布林判斷 x 與 y 元素。

提示: `tf.constant`、`tf.equal`、`tf.zeros_like`

```
x = tf.constant([[0, -2, -1], [0, 1, 2]])
y = tf.zeros_like(x)
out = tf.equal(x, y)
with tf.Session() as sess:
    print(sess.run(out))
```

判斷 x 與 y 是否相等，輸出boolean

序列

- `tf.linspace(start, stop, num, name=None)`：均分計算，在間隔中生成值。

Ex. `tf.linspace(10.0, 13.0, 4) ==> [10. 11. 12. 13.]`

`tf.range(start, limit=None, delta=1, dtype=None, name='range')`

`tf.range(3, 18, 3) ==> [3 6 9 12 15]`

`tf.range(5) ==> [0 1 2 3 4]`

隨機生成常量

■ `tf.random_normal`：從常態分佈中輸出隨機值

```
tf.random_normal (shape, mean=0.0, stddev=1.0, dtype=tf.float32,  
seed=None, name=None)
```

`shape`：一個一維整數張量或Python數組。代表張量的形狀。

`mean`：資料類型為`dtype`的張量值或Python值。是常態分佈的均值。

`stddev`：資料類型為`dtype`的張量值或Python值。是常態分佈的標準差。

`dtype`：輸出的資料類型。

`seed`：`tf.set_random_seed`功能。

`name`：操作的名稱(可選)

隨機生成常量

```
1 import tensorflow as tf
2 a = tf.random_normal([3,3])
3 with tf.Session() as sess:
4     print(sess.run(a))
```

-----> 建立隨機生成常量，其大小為3 * 3

```
[[ 0.33148983  0.963401 -0.26759976]
 [ 0.7575312  0.8530798 -0.18600698]
 [ 1.1833237 -2.23758 -1.9750623 ]]
```

```
1 a = tf.random_normal([5,3], 4, 2, tf.float64, 1.0)
2 with tf.Session() as sess:
3     print(sess.run(a))
```

-----> 建立隨機生成常量，其大小為5 * 3

```
[[4.48565509 3.29400356 2.32698442]
 [3.51286196 7.82013658 3.26140554]
 [4.72015467 0.56759923 5.3093726 ]
 [2.49358633 4.32934506 1.97655564]
 [4.15348251 7.03906969 5.52209567]]
```

隨機生成常量

```
1 c = tf.random_normal([1], seed=1.0)
2 print("Session 1")
3 with tf.Session() as sess:
4     print(sess.run(c))
5 print("Session 2")
6 with tf.Session() as sess:
7     print(sess.run(c))
```

Seed 設定為1

Session 1
[-0.8113182]
Session 2
[-0.8113182]

結果相等

練習題

- 建立隨機分佈 x & y 張量，張量大小為300，使用 `tf.cond()` 當平均小於0，輸出 $(x-y)^2$ 的平均值，當平均大於0輸出 $|x-y|$ 的總和

提示：`tf.random_normal`、`tf.reduce_mean`、`tf.square`、`tf.abs`、`tf.cond`

```
x = tf.random_normal([300], mean=5, stddev=1)
y = tf.random_normal([300], mean=5, stddev=1)
average = tf.reduce_mean(x - y) -----> Tensor 取平均
def f1(): return tf.reduce_mean(tf.square(x - y)) -----> Tensor 平方後取平均值
def f2(): return tf.reduce_sum(tf.abs(x - y)) -----> Tensor 絕對值後取平均值
out = tf.cond(average < 0, f1, f2) -----> Tensor 判斷式
```

`tf.cond(pred, fn1, fn2, name=None)`

`pred`: 條件式

`fn1`: 函式

`fn2`: 函式

Operations

- 元素數學運算
- 陣列op
- 矩陣op
- 狀態op
- 神經網路區塊 等

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural network building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Operations

- `tf.abs`
- `tf.negative`
- `tf.sign`
- `tf.reciprocal`
- `tf.square`
- `tf.round`
- `tf.sqrt`
- `tf.rsqrt`
- `tf.pow`
- `tf.exp`

■ 與Python的numpy類似

Operations

```
a = tf.constant([4, 9], name='a', dtype=tf.float32)
b = tf.constant([[0, -1], [2, -4]], name='b', dtype=tf.float32)
```

$\begin{bmatrix} 2 & 2 \end{bmatrix}$ $\begin{bmatrix} 0 & -1 \\ 2 & -3 \end{bmatrix}$

```
with tf.Session() as sess:
```

```
    print(sess.run(tf.abs(b)))
```

-----> $\begin{bmatrix} 0. & 1. \\ 2. & 4. \end{bmatrix}$

```
    print(sess.run(tf.square(b)))
```

-----> $\begin{bmatrix} 0. & 1. \\ 4. & 16. \end{bmatrix}$

```
    print(sess.run(tf.exp(a)))
```

-----> $[54.59815 \ 8103.084]$

```
    print(sess.run(tf.sqrt(a)))
```

-----> $[2. \ 3.]$

```
    print(sess.run(tf.pow(b, 2)))
```

-----> $\begin{bmatrix} 0. & 1. \\ 4. & 16. \end{bmatrix}$

變量(Variables)

- 建立 `tf.Variable` 變量 - `tf.Variable(initializer, name)`

```
s = tf.Variable(2, name="scalar")
m = tf.Variable([[0, 1], [2, 3]], name="matrix")
W = tf.Variable(tf.zeros([784,10]))
```

- 建立 `tf.get_variable` 變量 - `tf.get_variable(name, shape, initializer)`

```
s = tf.get_variable("scalar", initializer=tf.constant(2))
m = tf.get_variable("matrix", initializer=tf.constant([[0, 1], [2, 3]]))
W = tf.get_variable("big_matrix", shape=(784, 10), initializer=tf.zeros_initializer())
```

因為`tf.get_variable()` 會檢查當前命名空間下是否存在同樣name的變量，可以方便共享變量。而`tf.Variable` 每次都會新建一個變量。

變量(Variables)類別

- `tf.Variable` 擁有多個ops:

```
x = tf.Variable(...)
```

```
x.initializer # init op
```

```
x.value() # read op
```

```
x.assign(...) # write op
```

```
x.assign_add(...) # and more
```

初始化變量

■ 初始化所有變量:

```
with tf.Session() as sess:  
    sess.run(tf.global_variables_initializer())
```

■ 初始化變量的子集合(subset):

```
with tf.Session() as sess:  
    sess.run(tf.variables_initializer([a, b]))
```

■ 初始化單一變量

```
W = tf.Variable(tf.zeros([784,10]))  
with tf.Session() as sess:  
    sess.run(W.initializer)
```

練習題

- x為10、y為20，使用Variable輸出x與y的相加值。

```
x = tf.Variable(10, name='x')  
y = tf.Variable(20, name='y')  
z = tf.add(x, y)
```

結果相等

Tensor 相加

```
with tf.Session() as sess:
```

```
    sess.run(tf.global_variables_initializer())
```

```
    print(sess.run(z))
```

Variable 變量初始化

佔位符(Placeholders)

`tf.placeholder(dtype, shape=None, name=None)`

- 是一種常量，但是需在調用run方法傳遞的，其不像constant直接給定數值並使用，需要傳遞常數值。

佔位符(Placeholders)

`tf.placeholder(dtype, shape=None, name=None)`

- 建立大小為3的佔位符(placeholder), 類型為float32

```
a = tf.placeholder(tf.float32, shape=[3])
```

```
b = tf.constant([5, 5, 5], tf.float32)
```

```
c = a + b                                # short for tf.add(a, b)
```

```
with tf.Session() as sess:
```

```
    print(sess.run(c))
```


佔位符(Placeholders)

`tf.placeholder(dtype, shape=None, name=None)`

- 建立大小為3的佔位符(placeholder), 類型為float32

```
a = tf.placeholder(tf.float32, shape=[3])
```

```
b = tf.constant([5, 5, 5], tf.float32)
```

```
c = a + b                                # short for tf.add(a, b)
```

```
with tf.Session() as sess:
```

```
    print(sess.run(c))
```



`InvalidArgumentError: a doesn't an actual value`

佔位符(Placeholders)

`tf.placeholder(dtype, shape=None, name=None)`

- 建立大小為3的佔位符(placeholder), 類型為float32

```
a = tf.placeholder(tf.float32, shape=[3])
b = tf.constant([5, 5, 5], tf.float32)
c = a + b                # short for tf.add(a, b)
with tf.Session() as sess:
    print(sess.run(c, feed_dict={a: [1, 2, 3]}))
```

佔位符(Placeholders)

`tf.placeholder(dtype, shape=None, name=None)`

- 建立大小為3的佔位符(placeholder), 類型為float32

```
a = tf.placeholder(tf.float32, shape=[3])
```

```
b = tf.constant([5, 5, 5], tf.float32)
```

```
c = a + b # short for tf.add(a, b)
```

```
with tf.Session() as sess:
```

```
    print(sess.run(c, feed_dict={a: [1, 2, 3]}))
```



[6, 7, 8]

佔位符(Placeholders)

`tf.placeholder(dtype, shape=None, name=None)`

- 建立大小為3的佔位符(placeholder), 類型為float32

```
a = tf.placeholder(tf.float32, shape=[3])
```

```
b = tf.constant([5, 5, 5], tf.float32)
```

```
c = a + b # short for tf.add(a, b)
```

```
with tf.Session() as sess:
```

```
    print(sess.run(c, {a: [1, 2, 3]}))
```



[6, 7, 8]

練習題

- A為 $2+5$ 、B為 $a*3$ ，試試 feed_dict 另 $a=15$ 會有甚麼結果？

```
a = tf.add(2, 5)      -----> Tensor 相加
b = tf.multiply(a, 3) -----> Tensor 相乘

with tf.Session() as sess:
    print(sess.run(b))
    print(sess.run(b, feed_dict={a: 15}))
```

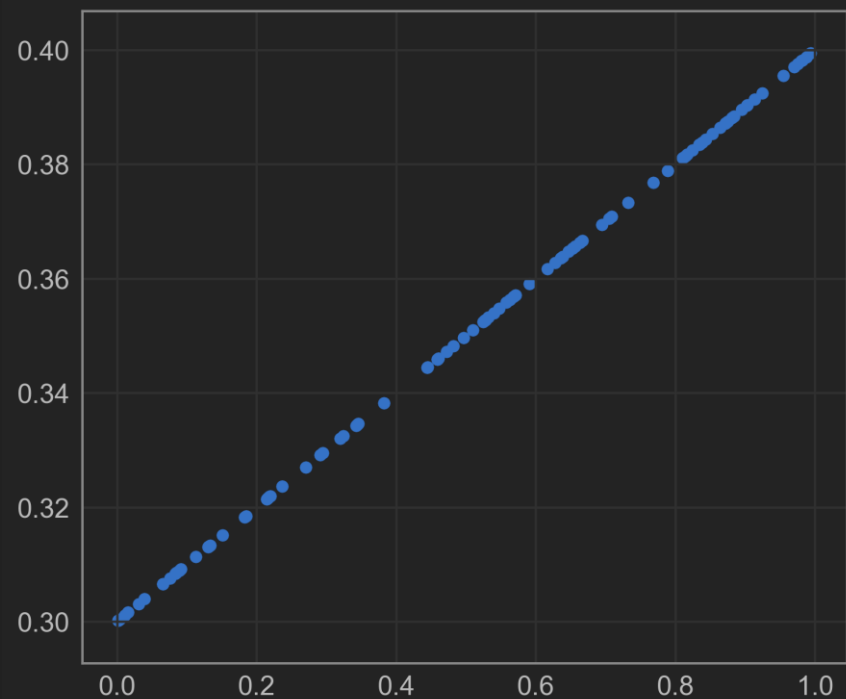
實作題

- 此線性方程式為： $Y=0.1*X+0.3$ ，建立神經元並訓練使其能得出與該方程式相同結果。

提示：

優化器：`tf.train.GradientDescentOptimizer(0.5)`

損失：`tf.reduce_mean(tf.square(y - y_data))`



```
0 [0.4482718] [0.13406843]
20 [0.18202826] [0.25157037]
40 [0.1203346] [0.28799444]
60 [0.10504089] [0.29702386]
80 [0.10124964] [0.29926223]
100 [0.10030977] [0.29981712]
120 [0.10007677] [0.29995468]
140 [0.10001902] [0.29998878]
160 [0.10000471] [0.29999724]
180 [0.10000114] [0.29999933]
200 [0.10000028] [0.29999986]
```

實作題分析

```
# create data
```

```
x_data = np.random.rand(100).astype(np.float32)
```

```
y_data = x_data * 0.1 + 0.3
```

```
Weights = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
```

```
biases = tf.Variable(tf.zeros([1]))
```

```
y = Weights*x_data + biases
```

```
loss = tf.reduce_mean(tf.square(y - y_data))
```

```
optimizer = tf.train.GradientDescentOptimizer(0.5)
```

```
train = optimizer.minimize(loss)
```

```
init = tf.global_variables_initializer()
```

```
with tf.Session() as sess:
```

```
    sess.run(init)          # Very important
```

```
    for step in range(201):
```

```
        sess.run(train)
```

```
        if step % 20 == 0:
```

```
            print(step, sess.run(Weights), sess.run(biases))
```

建立資料

建立亂數變量

建立變量

主要方程式

計算損失

優化器

找最小損失

變量初始化

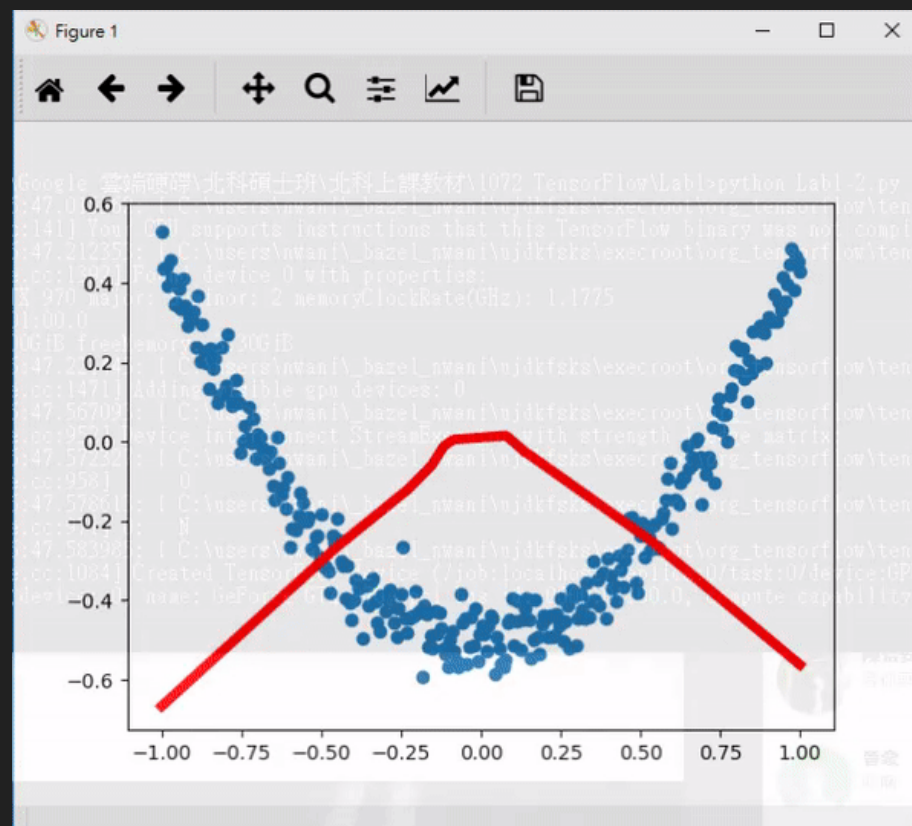
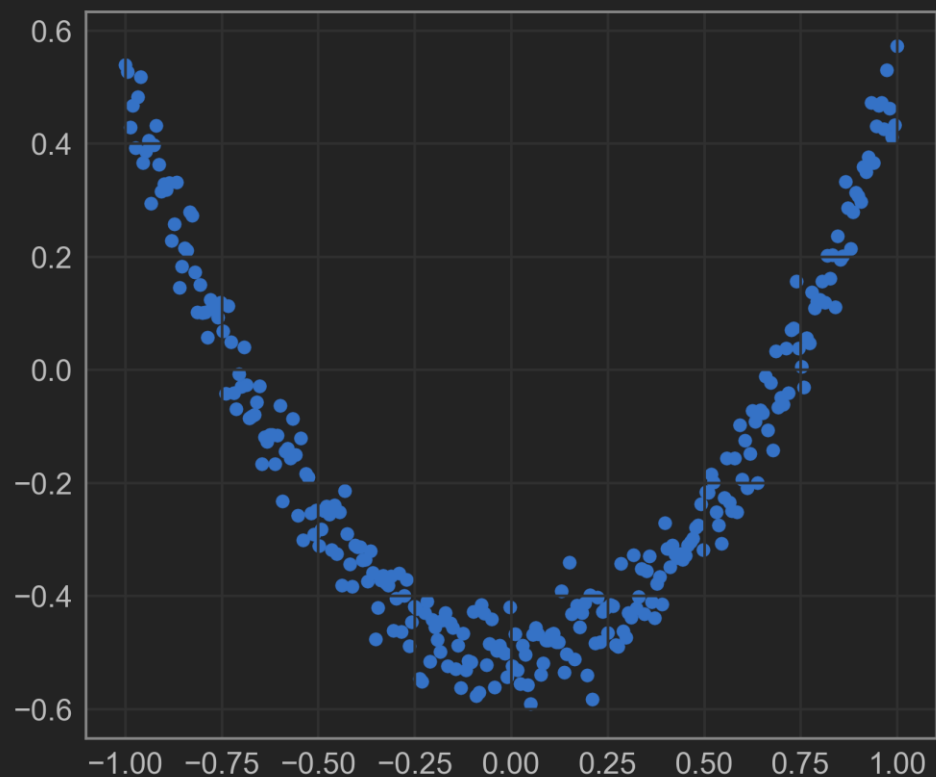
執行初始化所有變數，之前建立的 Op 只是描述了資料是怎樣流動或者是怎麼運算，沒有真正開始執行運算，只有把 Op 放入 sess.run(Op) 中才會開始執行。

得到 train 返回值，
也就是 loss 值

得到 Weights 與 biases 返回值，
也就是 權重與偏差值

實作題

- 此二次方程式為 $x^2 - 0.5$ ，建立神經元並訓練使其能得出與該方程式相同結果。



實作題分析

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

導入函式庫

```
def add_layer(inputs, in_size, out_size, activation_function=None):
    Weights = tf.Variable(tf.random_normal([in_size, out_size]))
    biases = tf.Variable(tf.zeros([1, out_size]) + 0.1)
    Wx_plus_b = tf.matmul(inputs, Weights) + biases
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs
```

定義 function

隨機常數變量

偏差變量

隨機常數變量

激勵函數，使得線性
方程式複雜化，引入
非線性方程式。

實作題分析

```
## Data
```

```
x_data = np.linspace(-1,1,300, dtype=np.float32)[: , np.newaxis]  
noise = np.random.normal(0, 0.05, x_data.shape).astype(np.float32)  
y_data = np.square(x_data) + noise
```

建立資料

```
xs = tf.placeholder(tf.float32, [None, 1])  
ys = tf.placeholder(tf.float32, [None, 1])
```

建立輸入與輸出佔位符，當後面真正
使用時會進行資料填充，這裡只是預
先告知資料的形狀和類型

```
l1 = add_layer(xs, 1, 10, activation_function=tf.nn.relu)  
prediction = add_layer(l1, 10, 1, activation_function=None)  
loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - prediction),  
                                reduction_indices=[1]))  
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
```

建立神經元層

計算實際與預測損失

隨機常數變量

```
init = tf.global_variables_initializer()
```

建立初始化變數 Op

```
sess = tf.Session()  
sess.run(init) # Very important
```

執行初始化所有變數，之前建立的 Op 只是
描述了資料是怎樣流動或者是怎麼運算，沒
有真正開始執行運算，只有把 Op 放入
sess.run(Op) 中才會開始執行。

實作題分析

```
# plot the real data
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(x_data, y_data)
plt.ion()
```

- > 定義繪圖窗口
- > 新建子圖
- > 將資料以散點方式繪入
- > 開啟交互模式，使每個pyplot命令都會重新繪製

```
# writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())
```

```
for i in range(1000):
```

```
    # training
```

```
    sess.run(train_step, feed_dict={xs: x_data, ys: y_data})
```

- > 餵入資料使神經元開始訓練

```
    if i % 50 == 0:
```

```
        print (sess.run(loss, feed_dict={xs: x_data, ys: y_data}))
```

- > 餵入資料並得到 loss 返回值

```
        # to visualize the result and improvement
```

```
        try:
```

```
            ax.lines.remove(lines[0])
```

- > 移除上一條曲線

```
        except Exception:
```

```
            pass
```

```
        prediction_value = sess.run(prediction, feed_dict={xs: x_data})
```

- > 返回預測值

```
        #plot the prediction
```

```
        lines = ax.plot(x_data, prediction_value, 'r-', lw=5)
```

- > 將預測結果以曲線畫出

```
        plt.pause(0.5)
```

```
plt.ioff()
```

- > 暫停0.5秒

```
plt.show()
```

- > 關閉交互模式

- > 將最後一張圖顯示出來

-END-