深度學習TensorFlow實務

手寫板功能

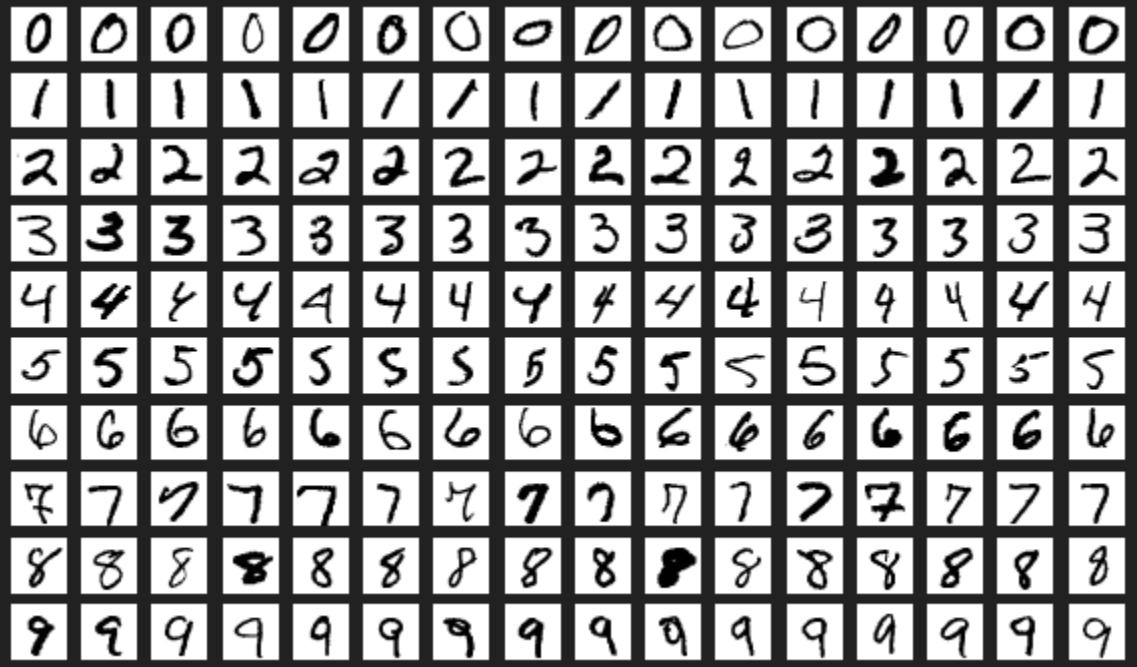
Lab2

-TA-
李偉弘
廖宜健
林佑昌
蔡明諺
彭冠偉

1. MNIST 介紹

MNIST

- 全名: Mixed National Institute of Standards and Technology database
- **手寫數字資料庫**，為網路上著名公開資料集之一
- 機器學習和深度學習研究機構，喜歡拿它來做訓練和測試



MNIST

- 這種公開常用資料集具有兩個功能:
  - 提供了大量的資料作為訓練集和驗證集，為一些學習人員提供了豐富的樣本資訊
  - 基準化分析法(Benchmark)，大家用的資料集都是一樣的，那麼設計出來的網路就可以在這個資料集上不斷互相比較，從而看誰的辨識率更高

MNIST 官方網站

- http://yann.lecun.com/exdb/mnist/

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

please note that your browser may uncompress these files without telling you. If the files you downloaded have a larger size than the above, they have been uncompressed by your browser. Simply rename them to remove the .gz extension. Some people have asked me "my application can't open your image files". These files are not in any standard image format. You have to write your own (very simple) program to read them. The file format is described at the bottom of this page.

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

With some classification methods (particuarly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass. If you do this kind of pre-processing, you should report it in your publications.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained

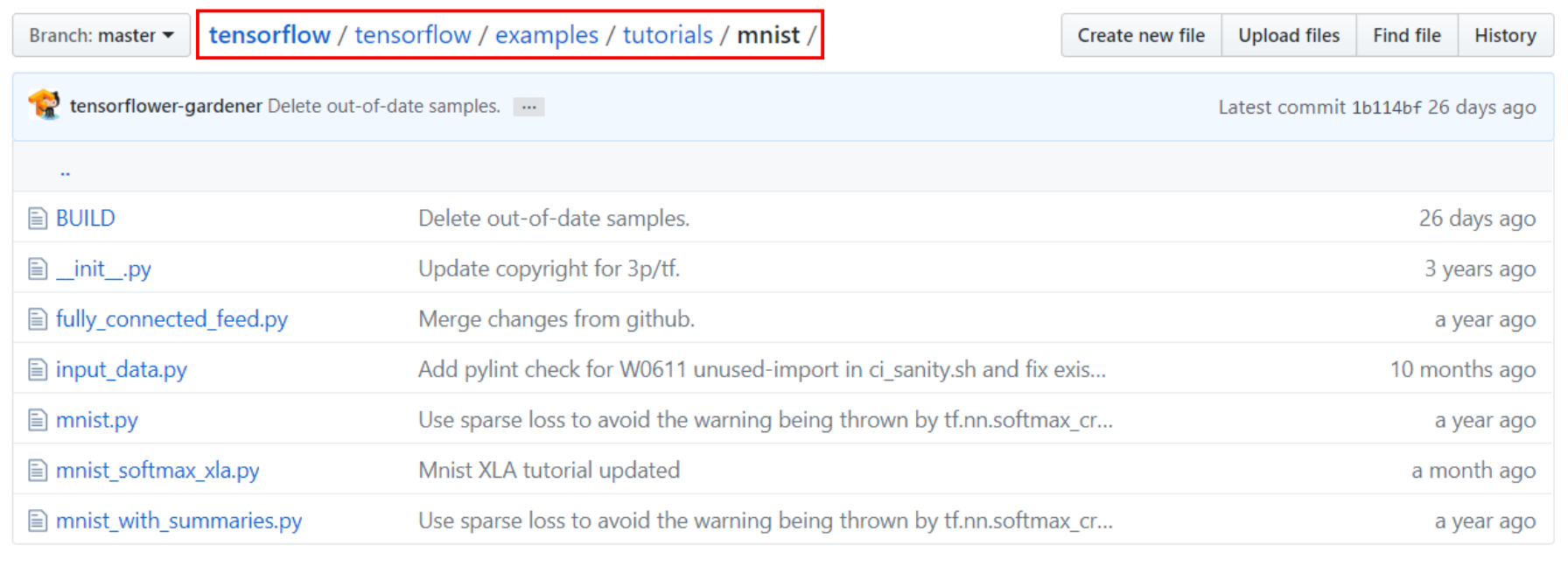
MNIST 資料文件

- 該網站提供 4 個檔案:
  - train-images-idx3-ubyte.gz: training set images (9912422 bytes)
  - train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
  - t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
  - t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)
- 上列依序為訓練圖片、訓練標籤、測試圖片、測試標籤
- 透過訓練集來讓模型學習這些數字，並讓模型在這些測試圖片上能成功辨別出它們來

2. 使用 TensorFlow 完成實驗

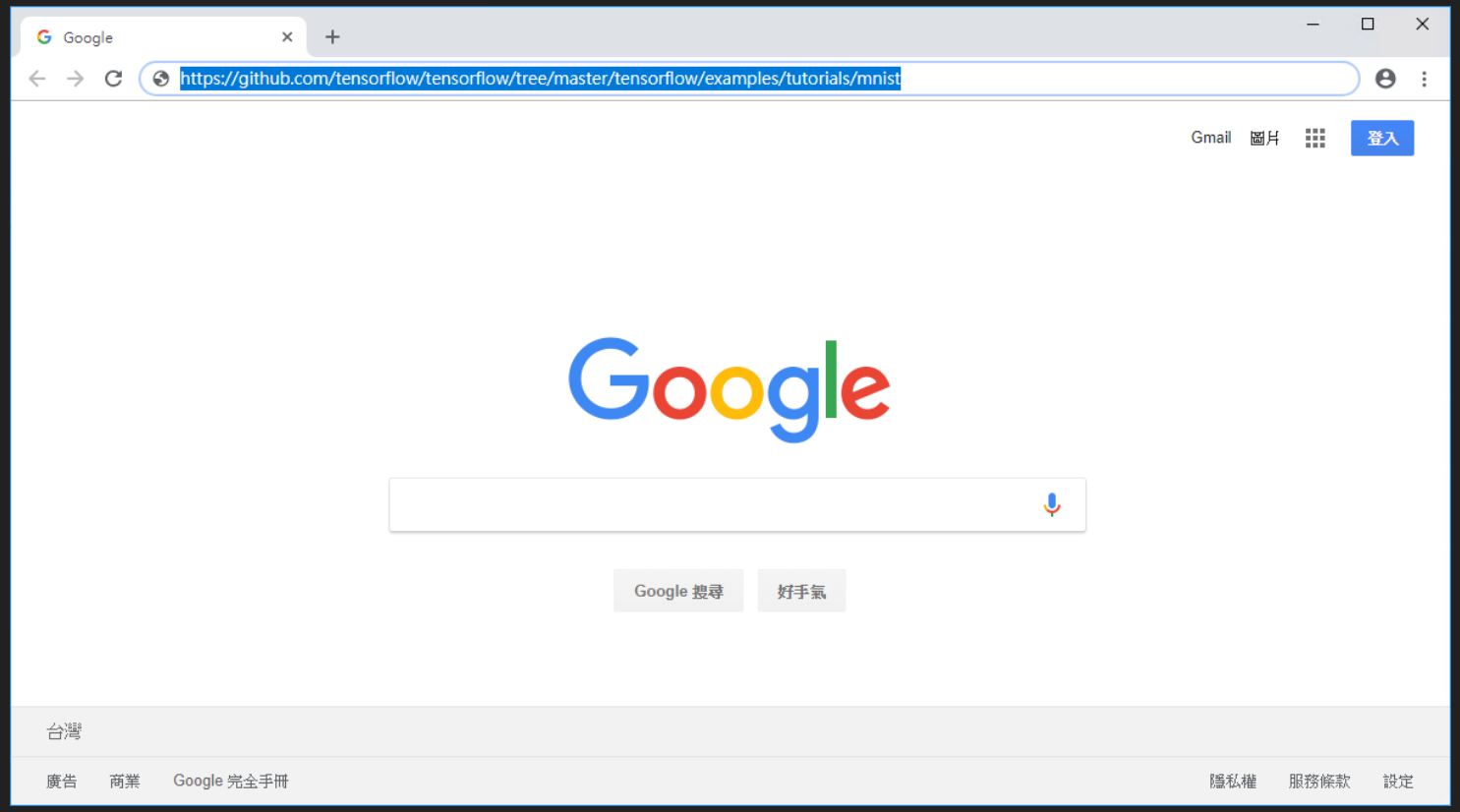
使用 TensorFlow 完成實驗

- 採用全連接神經網路(fully-connected neural network)完成 MNIST 資料集手寫辨識的工作
- TensorFlow 官方的 GitHub 上面有提供此次實驗的程式碼
- https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/tutorials/mnist



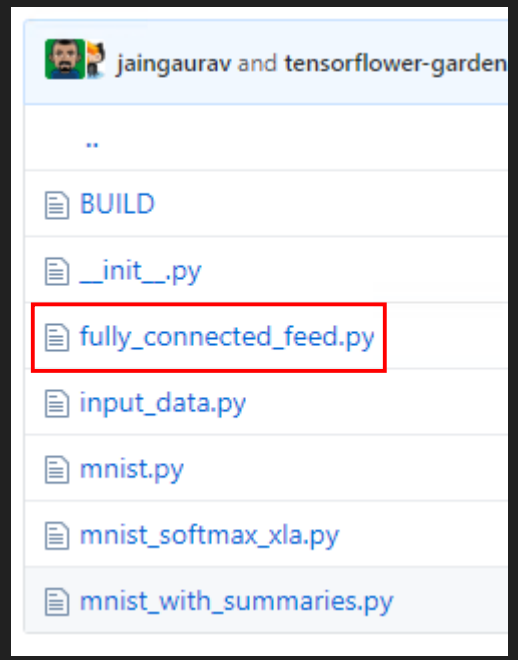
使用 TensorFlow 完成實驗

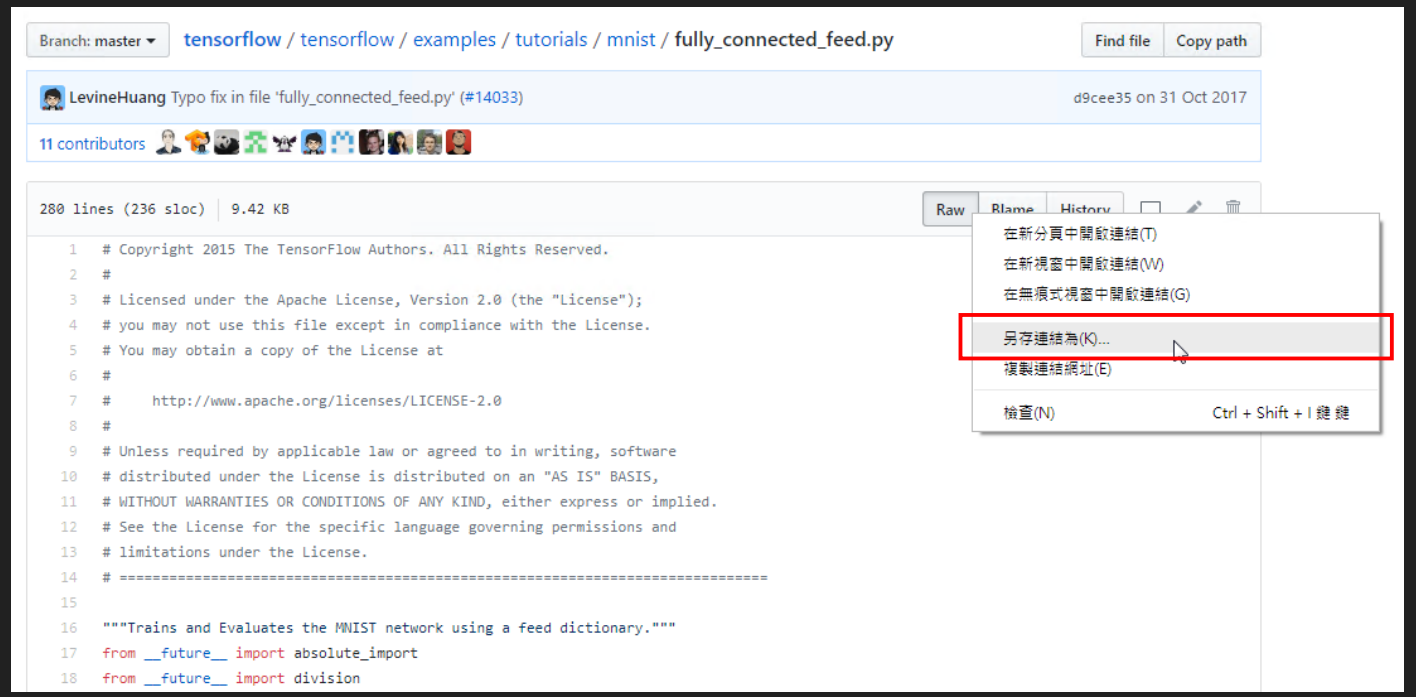
- 瀏覽器輸入以下網址：
- https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/tutorials/mnist



使用 TensorFlow 完成實驗

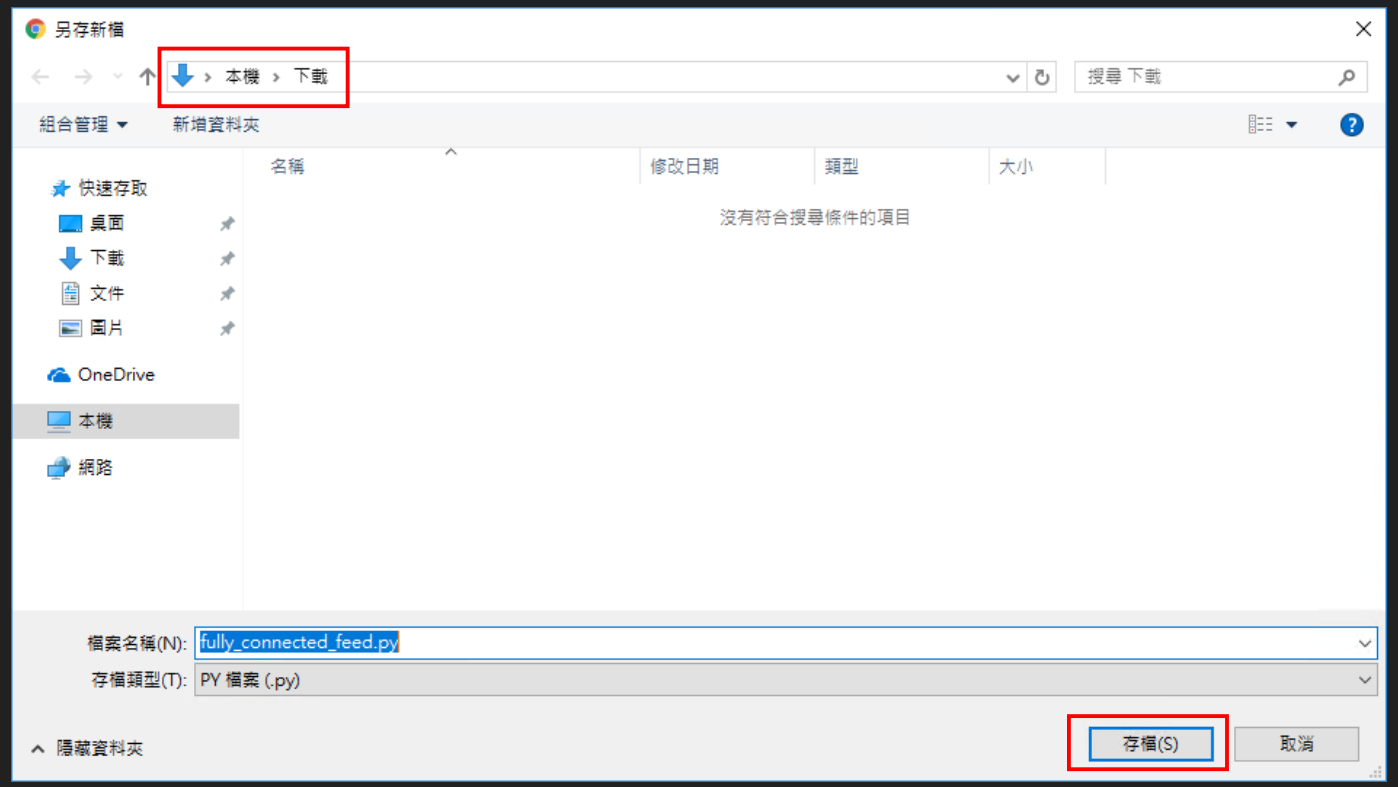
- 找到 fully_connected_feed.py，並點擊該檔案進入程式碼頁面(如右圖)
- 在右圖的 Raw 區域點擊右鍵，另存連結為(K)...





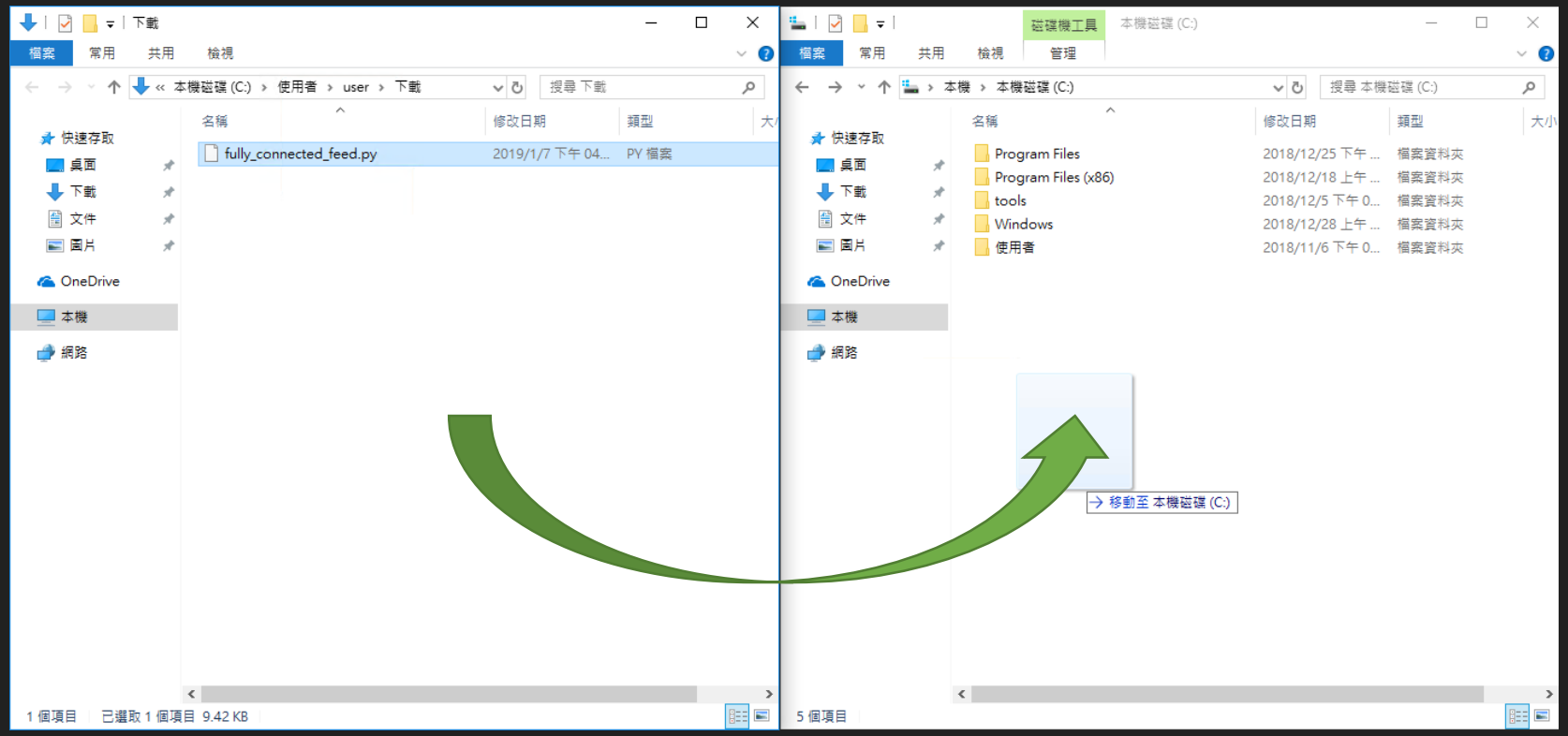
使用 TensorFlow 完成實驗

- 另存新檔，將 fully_connected_feed.py 至「下載」目錄
- 注意副檔名一定要為 .py



使用 TensorFlow 完成實驗

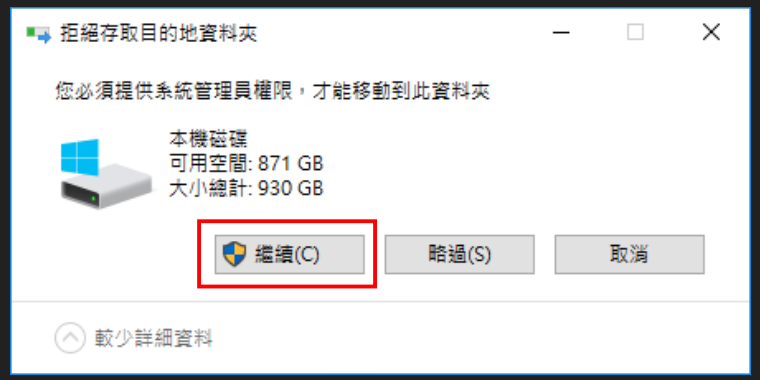
- 再將 fully_connected_feed.py ，由「下載」目錄移動至「本機磁碟(C:) 」

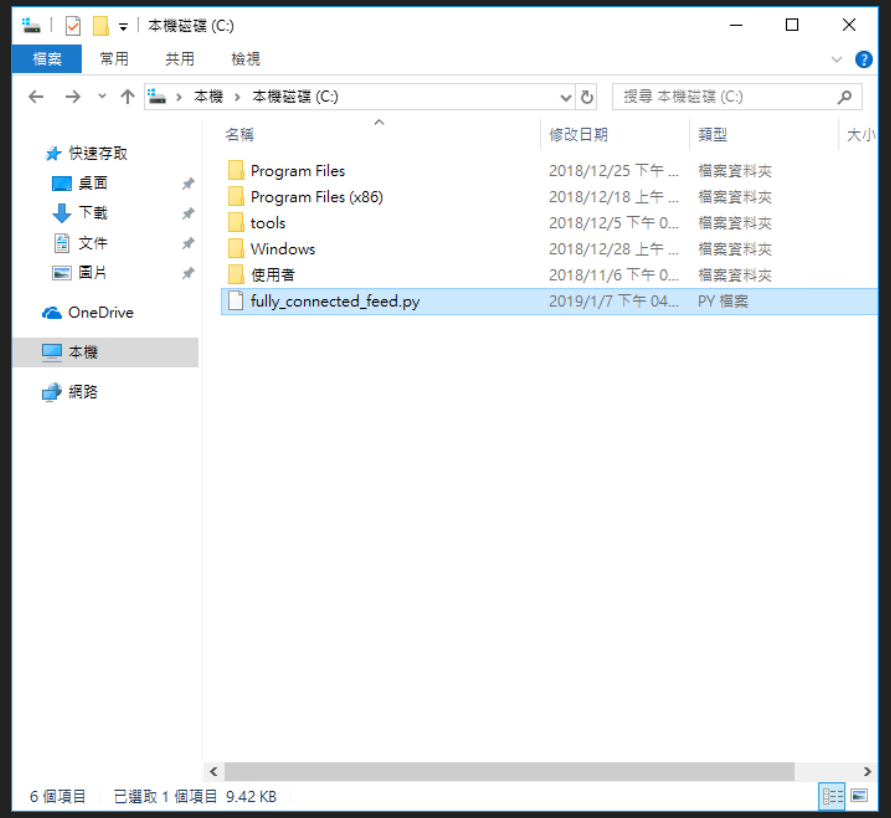


使用 TensorFlow 完成實驗

- 此時有可能會有系統管理員權限問題，點擊 

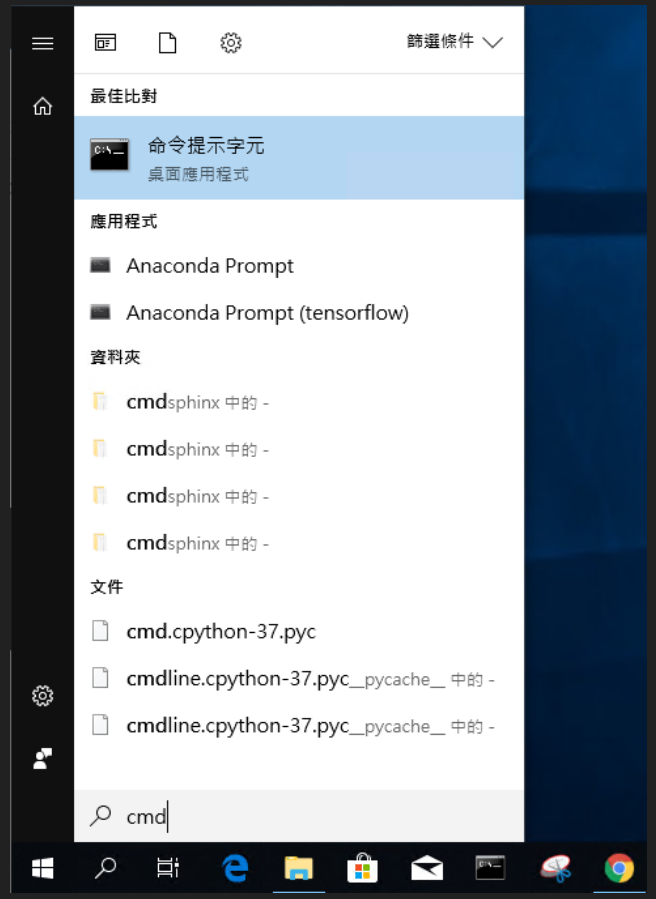
- 確認 fully_connected_feed.py 已於「本機磁碟(C:) 」目錄下

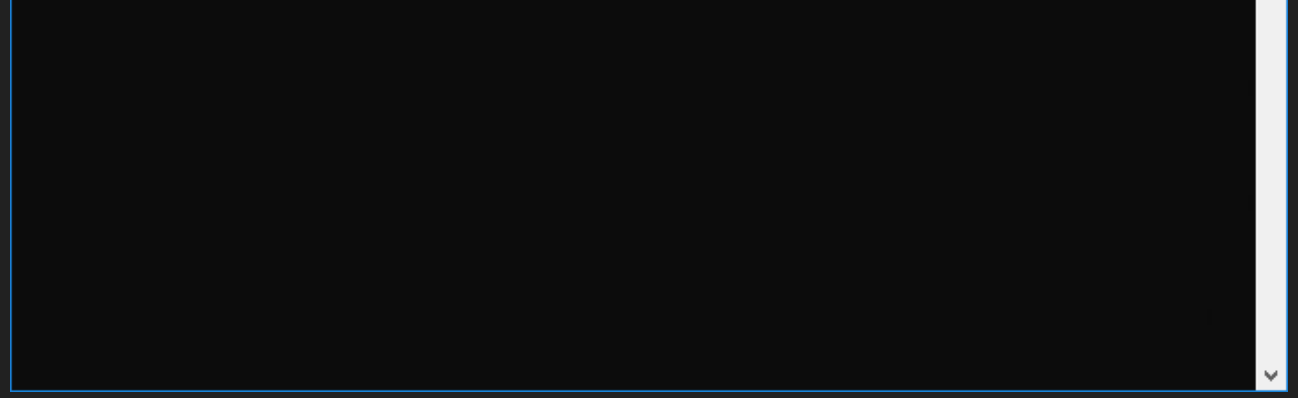




使用 TensorFlow 完成實驗

- 在搜尋輸入 cmd，開啟命令提示字元



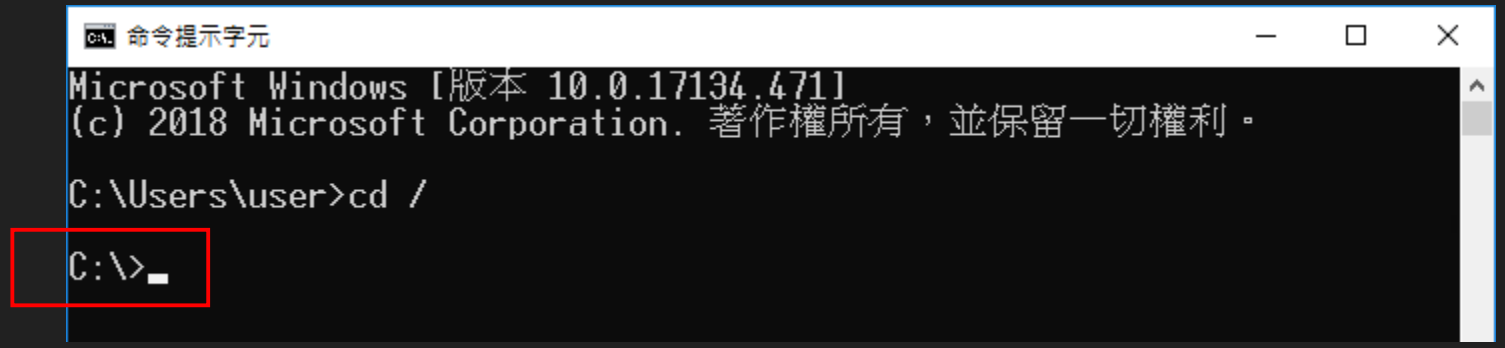


使用 TensorFlow 完成實驗

- 在 cmd 視窗，輸入以下指令

```
> cd \
```

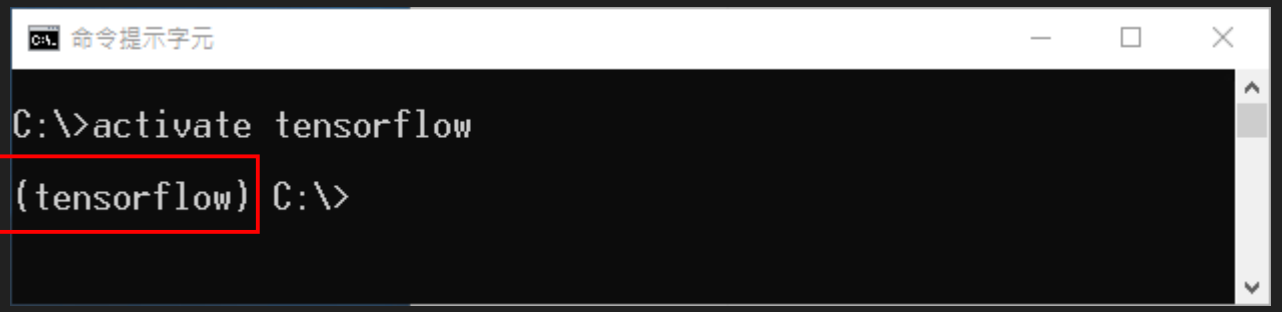
- 工作路徑由 C:\Users\user 變成 C:\



使用 TensorFlow 完成實驗

- 在 cmd 視窗，輸入以下指令

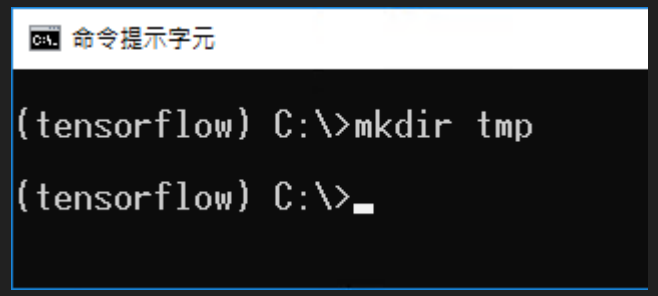
```
> activate tensorflow      # 之前已經建立該虛擬環境
```



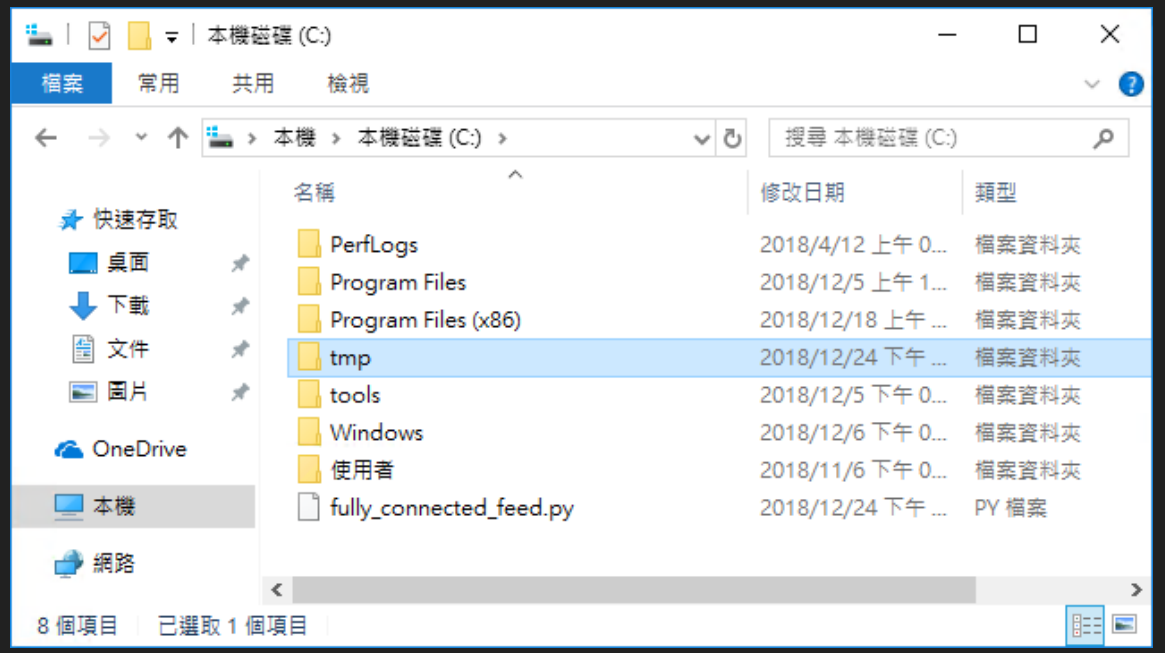
使用 TensorFlow 完成實驗

- 在 cmd 視窗，輸入以下指令

```
> mkdir tmp     # 建立目錄
```



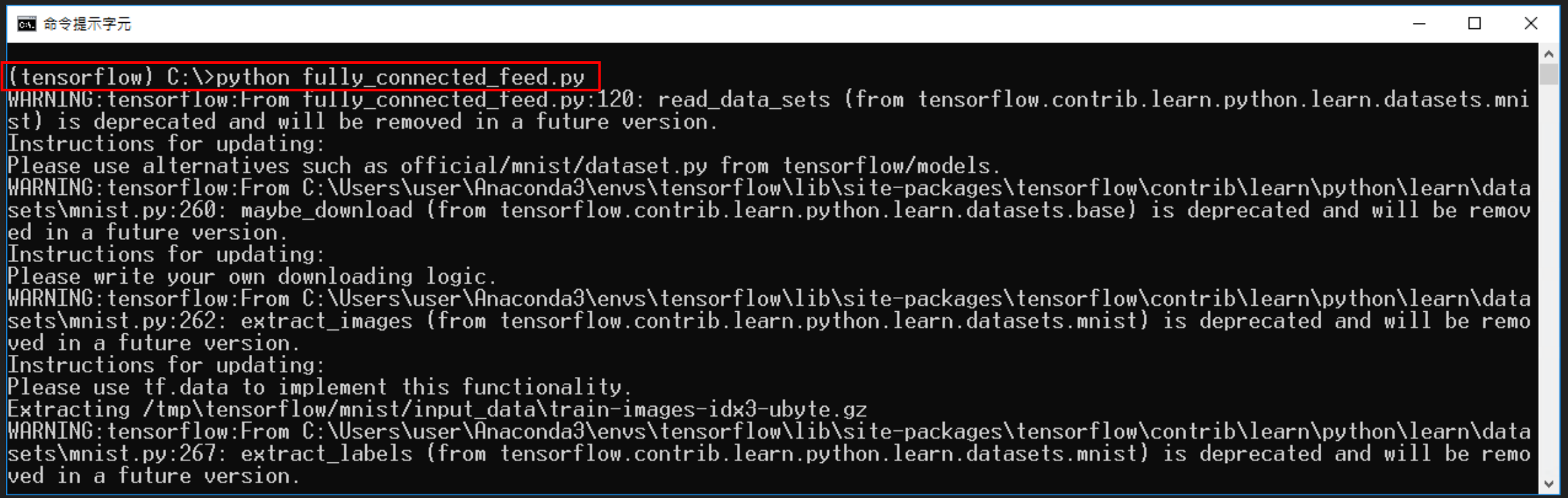
tmp 為放置輸入資料的目錄



使用 TensorFlow 完成實驗

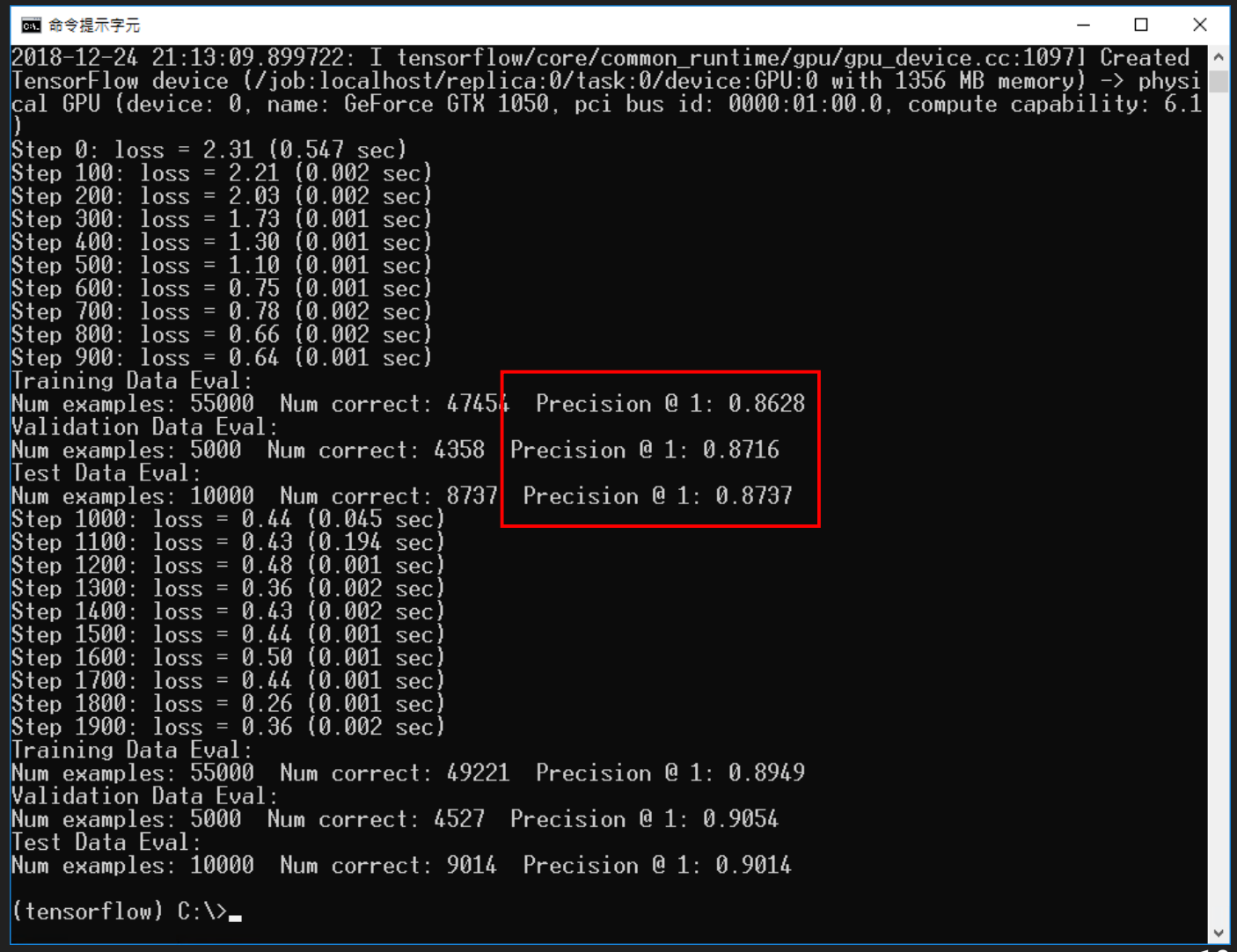
- 在 cmd 視窗，輸入以下指令

```
> python fully_connected_feed.py
```



使用 TensorFlow 完成實驗

- 出現這個畫面~成功
- max_steps = 2000
- 準確率(Precision)



3. fully_connected_feed.py 程式碼解析

fully_connected_feed.py 程式碼解析

- 解析命令列(command)，並啟動 TensorFlow

```
225 if __name__ == '__main__':
226   parser = argparse.ArgumentParser()
227   parser.add_argument(
228       '--learning_rate',
229       type=float,
230       default=0.01,
231       help='Initial learning rate.'
232   )
233   parser.add_argument(
234       '--max_steps',
235       type=int,
236       default=2000,
237       help='Number of steps to run trainer.'
238   )
239   parser.add_argument(
240       '--hidden1',
241       type=int,
242       default=128,
243       help='Number of units in hidden layer 1.'
244   )
245   parser.add_argument(
246       '--hidden2',
247       type=int,
248       default=32,
249       help='Number of units in hidden layer 2.'
250   )
```

```
251   parser.add_argument(
252       '--batch_size',
253       type=int,
254       default=100,
255       help='Batch size.  Must divide evenly into the dataset sizes.'
256   )
257   parser.add_argument(
258       '--input_data_dir',
259       type=str,
260       default=os.path.join(os.getenv('TEST_TMPDIR', '/tmp'),
261                            'tensorflow/mnist/input_data'),
262       help='Directory to put the input data.'
263   )
264   parser.add_argument(
265       '--log_dir',
266       type=str,
267       default=os.path.join(os.getenv('TEST_TMPDIR', '/tmp'),
268                            'tensorflow/mnist/logs/fully_connected_feed'),
269       help='Directory to put the log data.'
270   )
271   parser.add_argument(
272       '--fake_data',
273       default=False,
274       help='If true, uses fake data for unit testing.',
275       action='store_true'
276   )
277
278   FLAGS, unparsed = parser.parse_known_args()
279   tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

fully_connected_feed.py 程式碼解析

- 主程式

```
FLAGS, unparsed = parser.parse_known_args()
tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

啟動 TensorFlow 後首先調用 main 函數

```
def main(_):
  if tf.gfile.Exists(FLAGS.log_dir):
    tf.gfile.DeleteRecursively(FLAGS.log_dir)
  tf.gfile.MakeDirs(FLAGS.log_dir)
  run_training()
```

判斷 log_dir 目錄是否存在，已存在就刪除，不存在就建立

開始訓練 MNIST 資料

fully_connected_feed.py 程式碼解析

```python
def run_training():
  """Train MNIST for a number of steps."""
  # Get the sets of images and labels for training, validation, and
  # test on MNIST.
  data_sets = input_data.read_data_sets(FLAGS.input_data_dir, FLAGS.fake_data)

  # Tell TensorFlow that the model will be built into the default Graph.
  with tf.Graph().as_default():
    # Generate placeholders for the images and labels.
    images_placeholder, labels_placeholder = placeholder_inputs(
        FLAGS.batch_size)

    # Build a Graph that computes predictions from the inference model.
    logits = mnist.inference(images_placeholder,
                             FLAGS.hidden1,
                             FLAGS.hidden2)

    # Add to the Graph the Ops for loss calculation.
    loss = mnist.loss(logits, labels_placeholder)

    # Add to the Graph the Ops that calculate and apply gradients.
    train_op = mnist.training(loss, FLAGS.learning_rate)

    # Add the Op to compare the logits to the labels during evaluation.
    eval_correct = mnist.evaluation(logits, labels_placeholder)

    # Build the summary Tensor based on the TF collection of Summaries.
    summary = tf.summary.merge_all()
```

準備訓練、驗證和測試資料集。這裡 TensorFlow 提供了內建模組可以直接操作下載 MNIST datasets 資料集

使用預設圖（graph），TensorFlow 裡使用圖來表示運算任務，圖中的節點被稱為 Op（operation）

建立圖片和標籤的佔位符（placeholder），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

建立網路 Op、loss Op、gardients Op、evaluation Op（後續會說明 mnist.py 程式碼）

合併所有的 summary Op 為一個 Op

tf.summary

- TensorFlow 裡所有出現 summary 程式碼的地方都是在建立 summary Op，用來儲存訓練過程中你想要紀錄資料。比如：

```
tf.summary.histogram('histogram', var)
tf.summary.scalar('loss', loss)
```

- 如果需要記錄的資料很多，就會建立很多 summary Op，這時候使用 `tf.summary.merge_all` 來合併所有的 summary Op，就會方便很多
- 在訓練完畢後可以啟動 Tensorboard，輸入以下指令

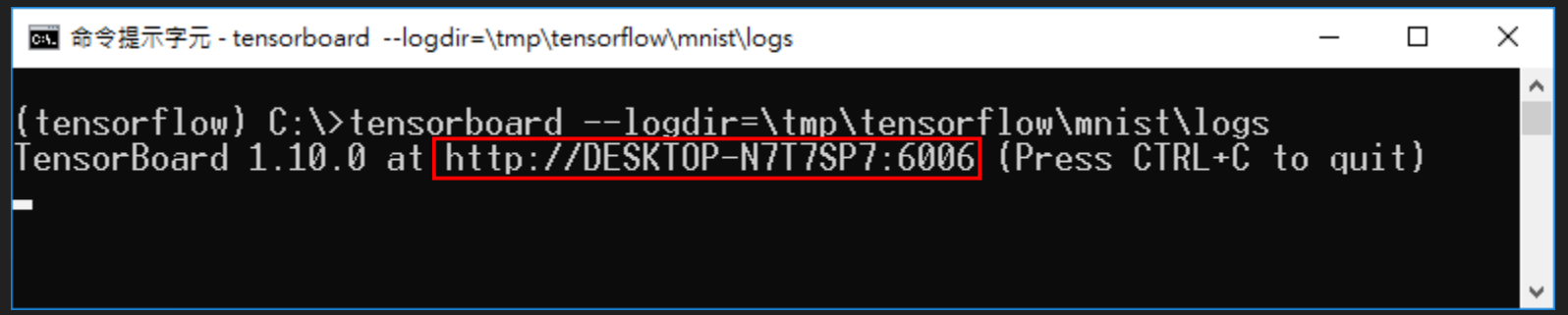
```
> tensorboard --logdir=path/to/logs
```

Tensorboard

- 本次程式的 logs 目錄是在 \tmp\tensorflow\mnist\logs
- tensorboard 的指令參數要改成以下：

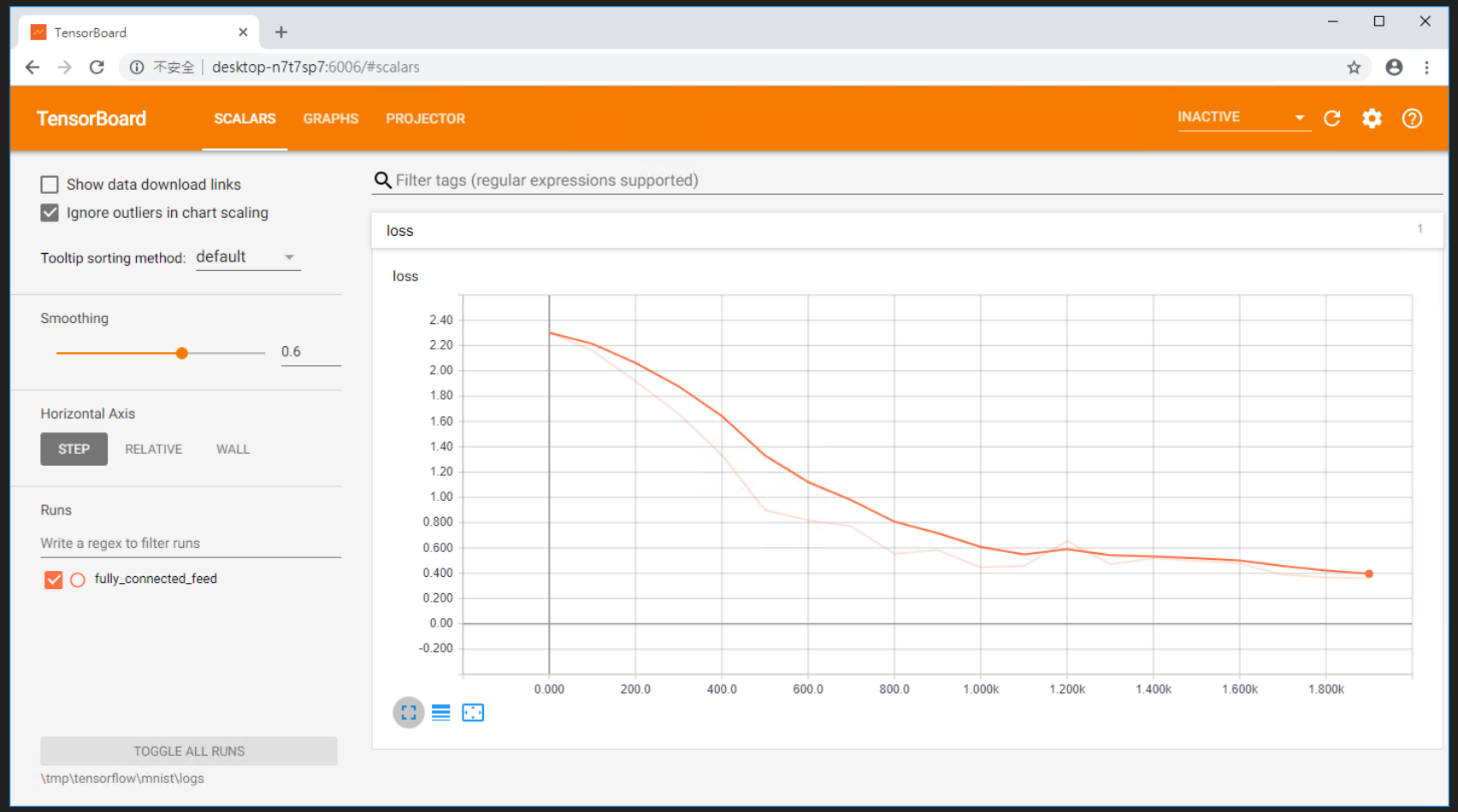
```
> tensorboard --logdir=\tmp\tensorflow\mnist\logs
```

- 在瀏覽器中輸入以下紅框部分的網址



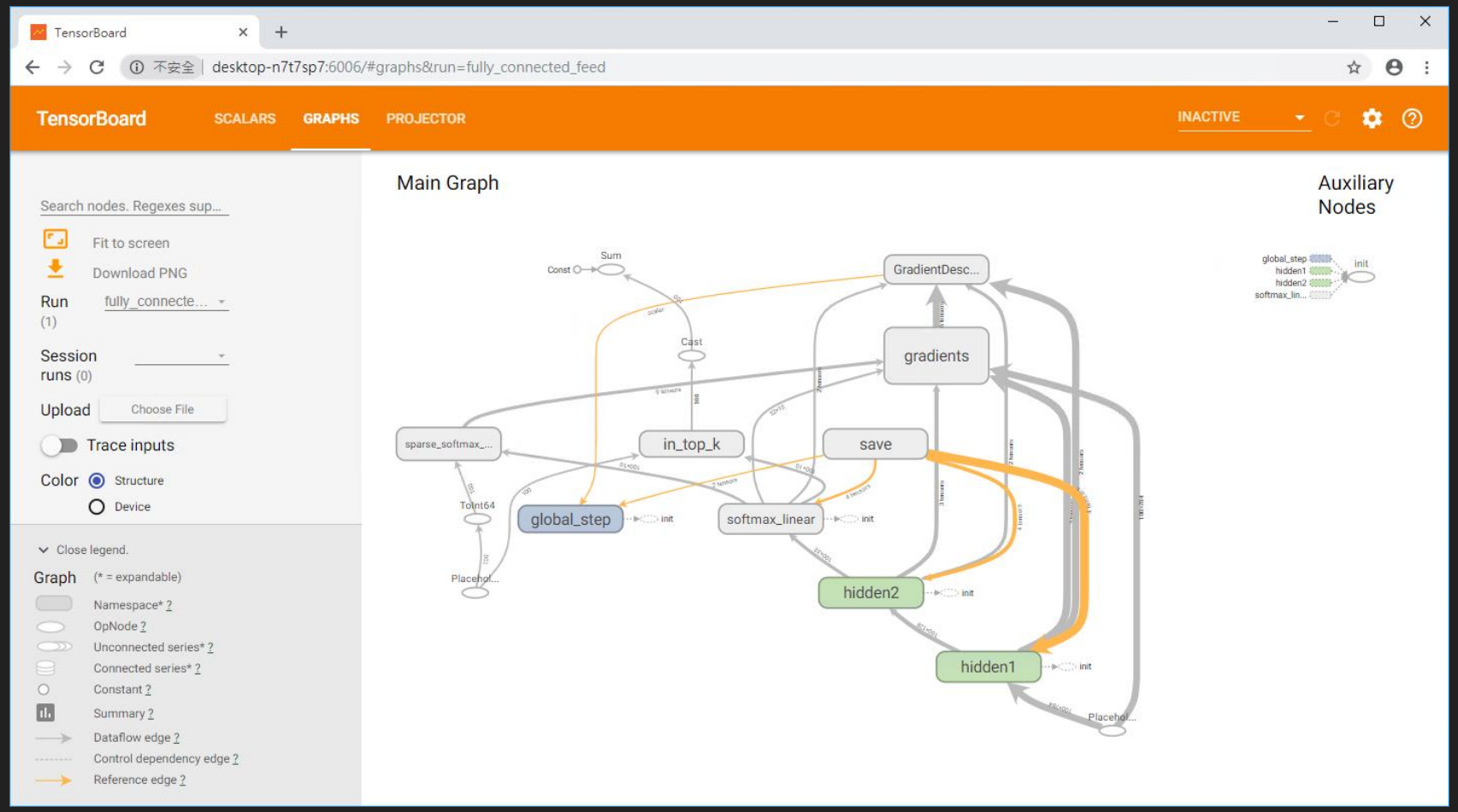
Tensorboard

- Scalars



Tensorboard

- Graphs



fully_connected_feed.py 程式碼解析

```
# Add the variable initializer Op.
init = tf.global_variables_initializer()

# Create a saver for writing training checkpoints.
saver = tf.train.Saver()

# Create a session for running Ops on the Graph.
sess = tf.Session()

# Instantiate a SummaryWriter to output summaries and the Graph.
summary_writer = tf.summary.FileWriter(FLAGS.log_dir, sess.graph)

# And then after everything is built:

# Run the Op to initialize the variables.
sess.run(init)
```

建立初始化變數 Op

建立 saver 來儲存模型

建立工作階段（session）上下文，圖需要在工作階段中執行

建立 summary FileWriter，把 summary Op 返回的資料寫到磁碟

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。

fully_connected_feed.py 程式碼解析

- 建置好所有參數與設定後，即可開始訓練網路(也就是學習)

```
# Start the training loop.
for step in xrange(FLAGS.max_steps):
  start_time = time.time()

  # Fill a feed dictionary with the actual set of images and labels
  # for this particular training step.
  feed_dict = fill_feed_dict(data_sets.train,
                             images_placeholder,
                             labels_placeholder)

  # Run one step of the model.  The return values are the activations
  # from the `train_op` (which is discarded) and the `loss` Op.  To
  # inspect the values of your Ops or variables, you may include them
  # in the list passed to sess.run() and the value tensors will be
  # returned in the tuple from the call.
  _, loss_value = sess.run([train_op, loss],
                           feed_dict=feed_dict)

  duration = time.time() - start_time
```

開始訓練循環，總共執行 FLAGS.max_steps 個 step

取一個 batch 訓練資料，使用真實資料填充圖片和標籤佔位符（placeholder）

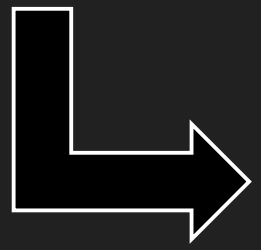
把一個 batch 資料放入模型進行訓練，得到 train_op 和 loss Op 的返回值。用不到 train_op 所以將其忽略掉，返回值變數設定為 _

計算執行一個 step 花費的時間

fully_connected_feed.py 程式碼解析

- 每 100 個 step 把 summary 訊息寫入硬碟一次

```python
# Write the summaries and print an overview fairly often.
if step % 100 == 0:
  # Print status to stdout.
  print('Step %d: loss = %.2f (%.3f sec)' % (step, loss_value, duration))
  # Update the events file.
  summary_str = sess.run(summary, feed_dict=feed_dict)
  summary_writer.add_summary(summary_str, step)
  summary_writer.flush()
```



```
Step 0: loss = 2.30 (0.487 sec)
Step 100: loss = 2.12 (0.002 sec)
Step 200: loss = 1.95 (0.002 sec)
Step 300: loss = 1.51 (0.001 sec)
Step 400: loss = 1.21 (0.002 sec)
Step 500: loss = 0.98 (0.001 sec)
Step 600: loss = 0.85 (0.001 sec)
Step 700: loss = 0.73 (0.002 sec)
Step 800: loss = 0.63 (0.002 sec)
Step 900: loss = 0.55 (0.002 sec)
```

fully_connected_feed.py 程式碼解析

- 每 1000 個 step 儲存一下模型，並且列印在訓練、驗證、測試資料集上的準確度

```python
# Save a checkpoint and evaluate the model periodically.
if (step + 1) % 1000 == 0 or (step + 1) == FLAGS.max_steps:
  checkpoint_file = os.path.join(FLAGS.log_dir, 'model.ckpt')
  saver.save(sess, checkpoint_file, global_step=step)
  # Evaluate against the training set.
  print('Training Data Eval:')
  do_eval(sess,
          eval_correct,
          images_placeholder,
          labels_placeholder,
          data_sets.train)
  # Evaluate against the validation set.
  print('Validation Data Eval:')
  do_eval(sess,
          eval_correct,
          images_placeholder,
          labels_placeholder,
          data_sets.validation)
  # Evaluate against the test set.
  print('Test Data Eval:')
  do_eval(sess,
          eval_correct,
          images_placeholder,
          labels_placeholder,
          data_sets.test)
```

```
Step 0: loss = 2.30 (0.487 sec)
Step 100: loss = 2.12 (0.002 sec)
Step 200: loss = 1.95 (0.002 sec)
Step 300: loss = 1.51 (0.001 sec)
Step 400: loss = 1.21 (0.002 sec)
Step 500: loss = 0.98 (0.001 sec)
Step 600: loss = 0.85 (0.001 sec)
Step 700: loss = 0.73 (0.002 sec)
Step 800: loss = 0.63 (0.002 sec)
Step 900: loss = 0.55 (0.002 sec)
Training Data Eval:
Num examples: 55000  Num correct: 47499  Precision @ 1: 0.8636
Validation Data Eval:
Num examples: 5000  Num correct: 4353  Precision @ 1: 0.8706
Test Data Eval:
Num examples: 10000  Num correct: 8722  Precision @ 1: 0.8722
Step 1000: loss = 0.53 (0.046 sec)
```

fully_connected_feed.py 程式碼解析

- 建置好所有參數與設定後，即可開始訓練網路(也就是學習)

```
# Start the training loop.
for step in xrange(FLAGS.max_steps):
  start_time = time.time()

  # Fill a feed dictionary with the actual set of images and labels
  # for this particular training step.
  feed_dict = fill_feed_dict(data_sets.train,
                             images_placeholder,
                             labels_placeholder)

  # Run one step of the model.  The return values are the activations
  # from the `train_op` (which is discarded) and the `loss` Op.  To
  # inspect the values of your Ops or variables, you may include them
  # in the list passed to sess.run() and the value tensors will be
  # returned in the tuple from the call.
  _, loss_value = sess.run([train_op, loss],
                           feed_dict=feed_dict)

  duration = time.time() - start_time
```

開始訓練循環，總共執行 FLAGS.max_steps 個 step

取一個 batch 訓練資料，使用真實資料填充圖片和標籤佔位符（placeholder）

把一個 batch 資料放入模型進行訓練，得到 train_op 和 loss Op 的返回值。用不到 train_op 所以將其忽略掉，返回值變數設定為 _

計算執行一個 step 花費的時間

4. mnist.py 程式碼解析

Review 回顧

```python
def run_training():
  """Train MNIST for a number of steps."""
  # Get the sets of images and labels for training, validation, and
  # test on MNIST.
  data_sets = input_data.read_data_sets(FLAGS.input_data_dir, FLAGS.fake_data)

  # Tell TensorFlow that the model will be built into the default Graph.
  with tf.Graph().as_default():
    # Generate placeholders for the images and labels.
    images_placeholder, labels_placeholder = placeholder_inputs(
        FLAGS.batch_size)

    # Build a Graph that computes predictions from the inference model.
    logits = mnist.inference(images_placeholder,
                             FLAGS.hidden1,
                             FLAGS.hidden2)

    # Add to the Graph the Ops for loss calculation.
    loss = mnist.loss(logits, labels_placeholder)

    # Add to the Graph the Ops that calculate and apply gradients.
    train_op = mnist.training(loss, FLAGS.learning_rate)

    # Add the Op to compare the logits to the labels during evaluation.
    eval_correct = mnist.evaluation(logits, labels_placeholder)

    # Build the summary Tensor based on the TF collection of Summaries.
    summary = tf.summary.merge_all()
```

建立網路 Op、loss Op、gardients Op、evaluation Op
（後續會說明 mnist.py 程式碼）

minst.py 程式碼解析

- MNIST dataset

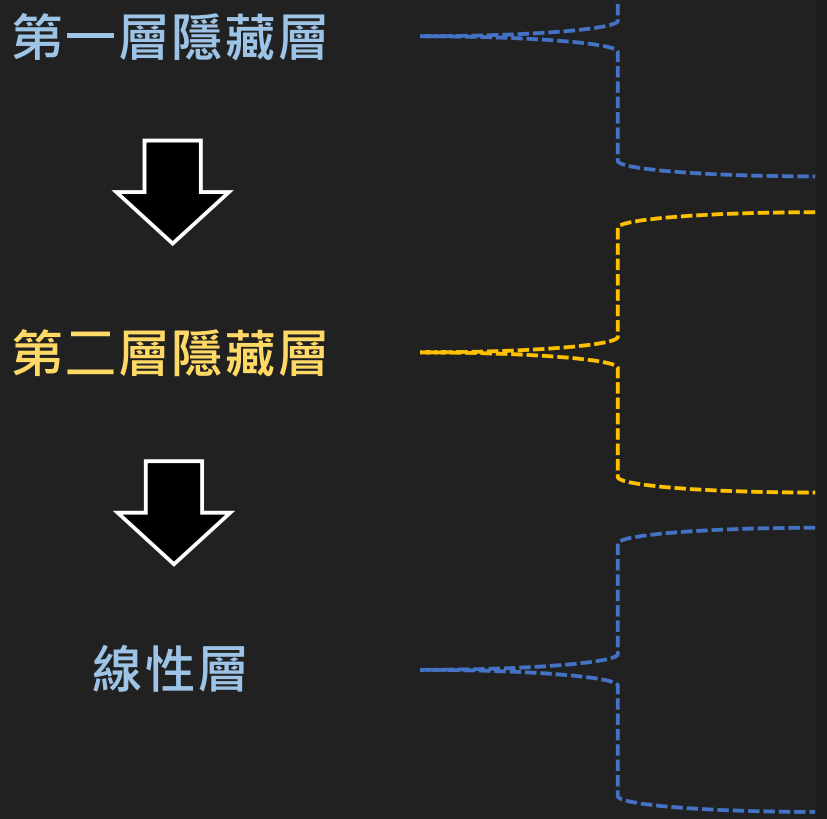
```
# The MNIST dataset has 10 classes, representing the digits 0 through 9.
NUM_CLASSES = 10

# The MNIST images are always 28x28 pixels.
IMAGE_SIZE = 28
IMAGE_PIXELS = IMAGE_SIZE * IMAGE_SIZE
```

總共 10 類別，0 ~ 9 的手寫數字圖片

每張圖片像素 28 x 28 = 784

mnist.py 程式碼解析

- 建構神經網路




```python
def inference(images, hidden1_units, hidden2_units):
  # Hidden 1
  with tf.name_scope('hidden1'):
    weights = tf.Variable(
        tf.truncated_normal([IMAGE_PIXELS, hidden1_units],
                            stddev=1.0 / math.sqrt(float(IMAGE_PIXELS))),
        name='weights')
    biases = tf.Variable(tf.zeros([hidden1_units]),
                         name='biases')
    hidden1 = tf.nn.relu(tf.matmul(images, weights) + biases)
  # Hidden 2
  with tf.name_scope('hidden2'):
    weights = tf.Variable(
        tf.truncated_normal([hidden1_units, hidden2_units],
                            stddev=1.0 / math.sqrt(float(hidden1_units))),
        name='weights')
    biases = tf.Variable(tf.zeros([hidden2_units]),
                         name='biases')
    hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
  # Linear
  with tf.name_scope('softmax_linear'):
    weights = tf.Variable(
        tf.truncated_normal([hidden2_units, NUM_CLASSES],
                            stddev=1.0 / math.sqrt(float(hidden2_units))),
        name='weights')
    biases = tf.Variable(tf.zeros([NUM_CLASSES]),
                         name='biases')
    logits = tf.matmul(hidden2, weights) + biases
  return logits
```

minst.py 程式碼解析

- 第一層隱藏層

```
def inference(images, hidden1_units, hidden2_units):
  # Hidden 1
  with tf.name_scope('hidden1'):
    weights = tf.Variable(
        tf.truncated_normal([IMAGE_PIXELS, hidden1_units],
                            stddev=1.0 / math.sqrt(float(IMAGE_PIXELS))),
        name='weights')
    biases = tf.Variable(tf.zeros([hidden1_units]),
                         name='biases')
    hidden1 = tf.nn.relu(tf.matmul(images, weights) + biases)
```

設定命名空間，不同的命名空間中變數名不衝突

建立變數和權重，使用截斷常態分布進行初始化

建立變數和偏移量，全部初始化為 0

對線性運算結果，套用激活函數 ReLU

$$X_{100\times784} \otimes W_{784\times128} = Y_{100\times128}$$

minst.py 程式碼解析

- 第二層隱藏層

```python
# Hidden 2
with tf.name_scope('hidden2'):
  weights = tf.Variable(
      tf.truncated_normal([hidden1_units, hidden2_units],
                          stddev=1.0 / math.sqrt(float(hidden1_units))),
      name='weights')
  biases = tf.Variable(tf.zeros([hidden2_units]),
                       name='biases')
  hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
```

$$X_{100\times784} \times W_{784\times128} = Y_{100\times128}$$

$$X_{100\times128} \times W_{128\times32} = Y_{100\times32}$$

minst.py 程式碼解析

- 線性層

```
# Linear
with tf.name_scope('softmax_linear'):
  weights = tf.Variable(
      tf.truncated_normal([hidden2_units, NUM_CLASSES],
                          stddev=1.0 / math.sqrt(float(hidden2_units))),
      name='weights')
  biases = tf.Variable(tf.zeros([NUM_CLASSES]),
                       name='biases')
  logits = tf.matmul(hidden2, weights) + biases
return logits
```

只有線性運算，無使用激活函數

$$X_{100\times784} \times W_{784\times128} = Y_{100\times128}$$

$$X_{100\times128} \times W_{128\times32} = Y_{100\times32}$$

$$X_{100\times32} \times W_{32\times10} = Y_{100\times10}$$

minst.py 程式碼解析

- 建立 loss op

```
# Add to the Graph the Ops for loss calculation.
loss = mnist.loss(logits, labels_placeholder)
```

```
def loss(logits, labels):
  """Calculates the loss from the logits and the labels.

  Args:
    logits: Logits tensor, float - [batch_size, NUM_CLASSES].
    labels: Labels tensor, int32 - [batch_size].

  Returns:
    loss: Loss tensor of type float.
  """
  labels = tf.to_int64(labels)
  return tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)
```

每張圖片只允許被標記為一個類別

minst.py 程式碼解析

- 建立 training Ops

```
# Add to the Graph the Ops that calculate and apply gradients.
train_op = mnist.training(loss, FLAGS.learning_rate)
```

```
def training(loss, learning_rate):
  """Sets up the training Ops."""
  # Add a scalar summary for the snapshot loss.
  tf.summary.scalar('loss', loss)
  # Create the gradient descent optimizer with the given learning rate.
  optimizer = tf.train.GradientDescentOptimizer(learning_rate)
  # Create a variable to track the global step.
  global_step = tf.Variable(0, name='global_step', trainable=False)
  # Use the optimizer to apply the gradients that minimize the loss
  # (and also increment the global step counter) as a single training step.
  train_op = optimizer.minimize(loss, global_step=global_step)
  return train_op
```

紀錄 loss 變化數值

使用梯度下降優化器，傳入學習率

建立變數來記錄全域步數

使用優化器之目的是最小化 loss

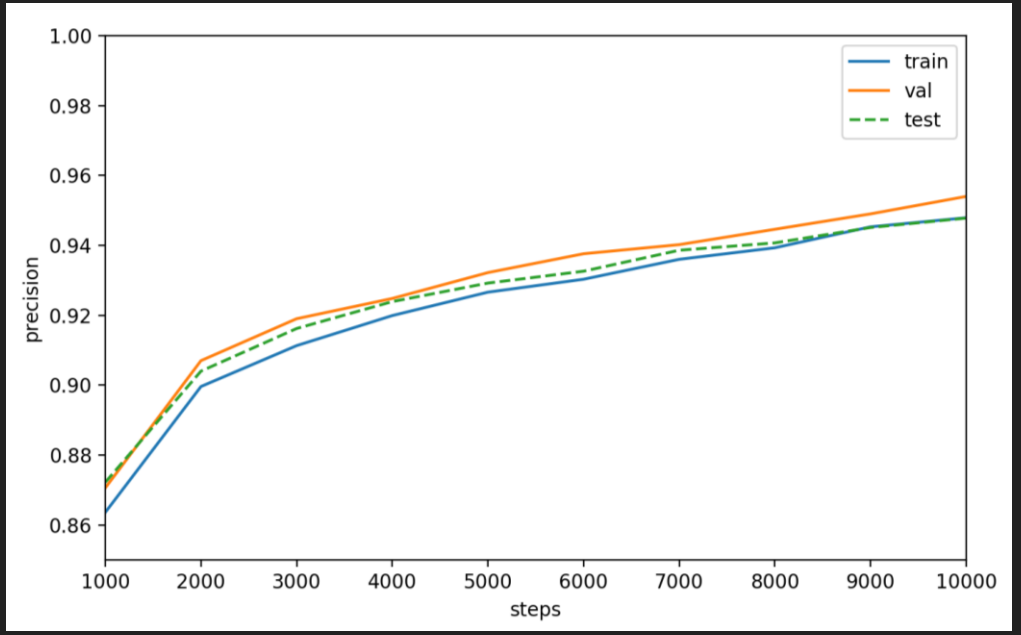
5. 練習題

改變訓練的 max_steps

- 在 cmd 視窗，輸入以下指令

```
> python fully_connected_feed.py --max_steps=10000
```

- Jupyter notebook 上完成右圖：
  - numpy
  - matplotlib



-END-