



深度學習

許懷中 & 教研處

「版權聲明頁」

本投影片已經獲得作者授權台灣人工智慧學校得以使用於教學用途，如需取得重製權以及公開傳輸權需要透過台灣人工智慧學校取得著作人同意；如果需要修改本投影片著作，則需要取得改作權；另外，如果有需要以光碟或紙本等實體的方式傳播，則需要取得人工智慧學校散佈權。

課程內容

1. 有框架的深度學習

1.1 Tensorflow 的基本語法與概念

1.2 線性迴歸模型

2. 利用Tensorflow建構神經網路

2.1 手寫數字辨識

2.2 儲存及載入模型

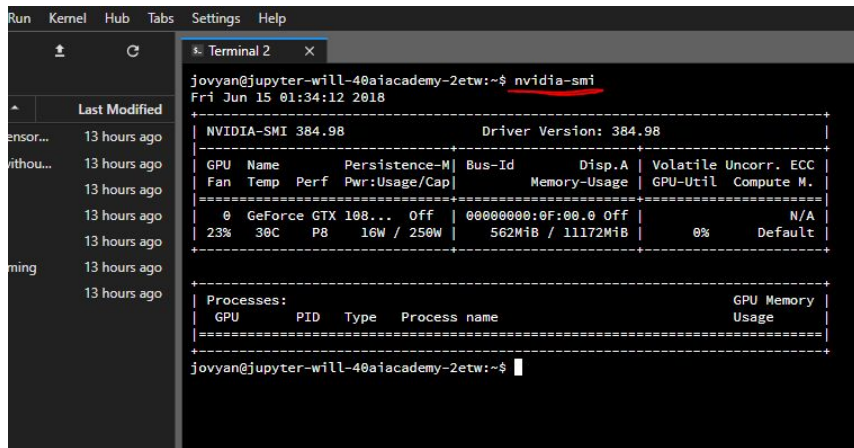
Before we start

- 在本機上安裝 Tensorflow, 若本機端沒有 gpu, 執行 `pip install tensorflow` 會自動安裝 cpu 版本。
- 此後的課程, 是否使用 gpu 會使程式的運行速度有極大的差距, 學員們可以多善用 server 上 gpu 的資源。



Before we start

- 如何知道自己是否使用到 GPU?
在 terminal 輸入指令 `nvidia-smi`

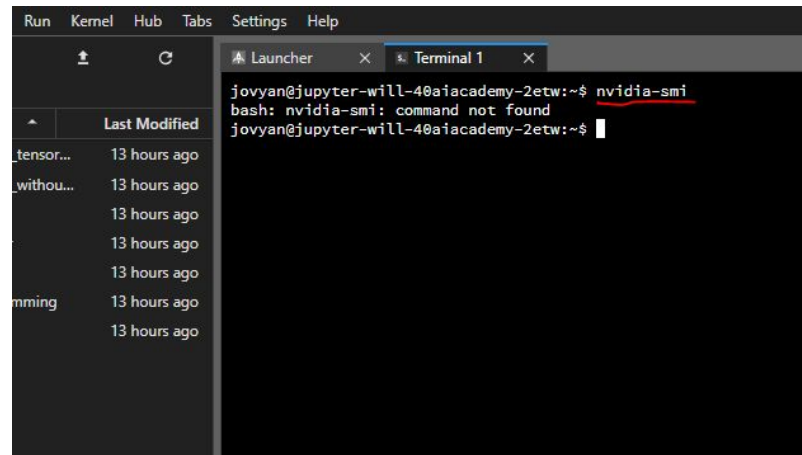


A terminal window titled "Terminal 2" showing the command `nvidia-smi` being executed. The output displays NVIDIA-SMI 384.98 and Driver Version: 384.98, followed by a table of GPU information.

GPU		Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.

GPU	PID	Type	Process name	GPU Memory Usage
0			GeForce GTX 108...	Off
23%	30C	P8	16W / 250W	562MiB / 11172MiB

如果有顯卡且驅動程式安裝成功



A terminal window titled "Terminal 1" showing the command `nvidia-smi` being executed. The output is `bash: nvidia-smi: command not found`, indicating that the command is not recognized.

沒有讀取到GPU



Caffe



DL4J
Deeplearning4j

K
KERAS

Microsoft
CNTK

MatConvNet

MINERVA

mxnet

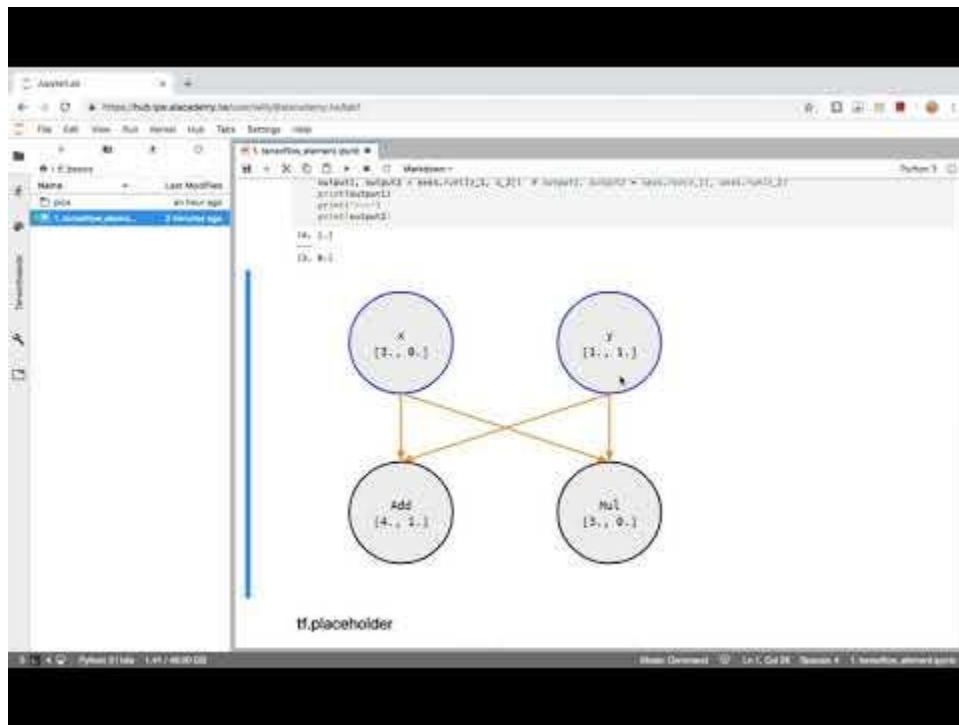


theano



有框架的深度學習之旅

Tensorflow的基本元素 (1. tensorflow_element.ipynb)



為何使用Tensorflow

- 優點

- 由 google 開源及維護
- 有龐大的 community support
- 彈性的 graph and flow 設計
- 易於支援大規模部署

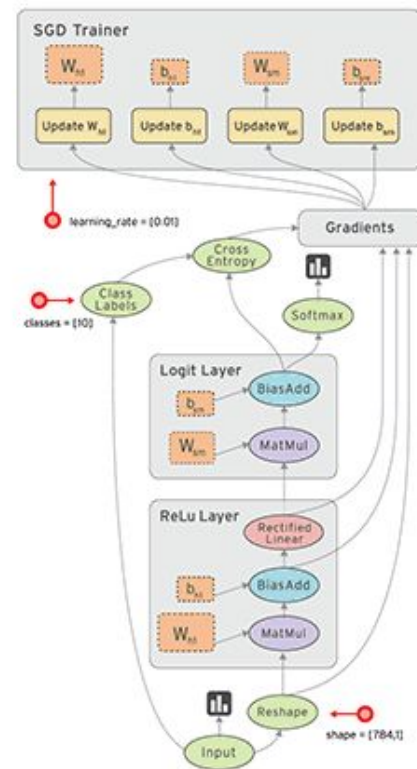
- 缺點

- 概念不易理解
- 屬於底層框架



Tensorflow運行的兩個步驟

- 建立 **graph** (網路設計圖)
 - graph 上的每個物件都由 tensor object 及 tensor operation 組成。
- 啟用 **session**, 運行 **graph**
 - session 如同電插座, 要讓 graph 上的元素互相運算必須插上電, 讓電流在裡面流動。



Graph上的基本元素

- Graph 上的每個元素都是 Tensor, Tensor 是多維向量的泛稱。
- 在 tensorflow 裏, 有三種特殊的 Tensor 可以擁有自己的數值:
 - `tf.constant`: 常數, 創建出來之後就不能改變數值的 Tensor
 - `tf.Variable`: 變數, 此種 Tensor 的值可以不斷更新
 - `tf.placeholder`: 一個預留位置, 可以事後再給定數值
- 在 tensorflow 的世界, 每一個 Tensor 有三個屬性:
`name`, `shape`, `dtype`

```
a = tf.constant([1., 0.], dtype=tf.float32, name='const_a')  
a  
  
<tf.Tensor 'const_a:0' shape=(2,) dtype=float32>
```



Tensor的運算

numpy 語法對應到 tensorflow 語法

- `+, -, *, /` -> `tf.add, tf.subtract, tf.multiply, tf.divide`
- `np.mean, np.sum` -> `tf.reduce_mean, tf.reduce_sum`

大部分 numpy 矩陣的操作 function 都能在 tensorflow 找到對應的語法

- `np.matmul, np.concatenate` -> `tf.matmul, tf.concat`

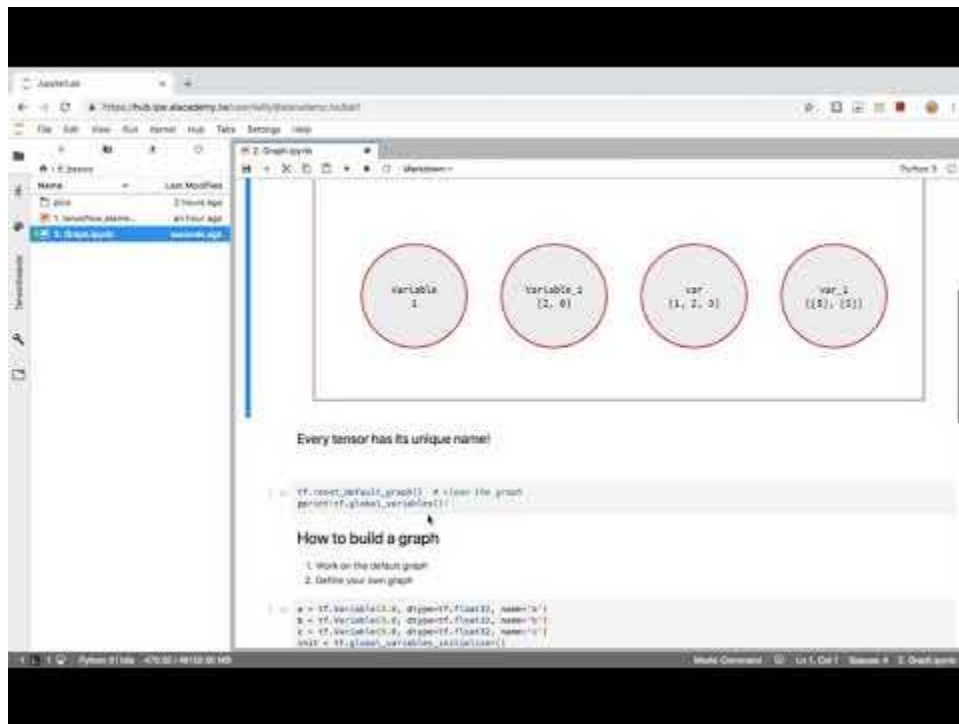


Session

- 所有的 Tensor object 都是看不到值的，想取出值或執行運算都必須要開啟 session 來執行。
- 用完的 session 記得要 `sess.close()`，
或者利用 `with tf.Session() as sess:` 的語法讓 session 自動關閉。



Graph的本質 (2. Graph.ipynb)

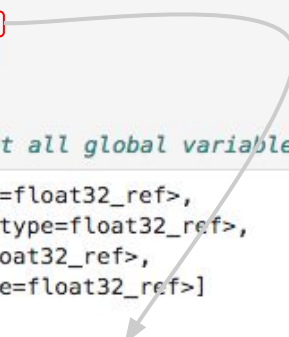


Tensorflow的命名系統

```
a = tf.Variable(1.)
a = tf.Variable([2., 0.])
a = tf.Variable([1., 2., 3.], name='var')
a = tf.Variable([[5.], [5.]], name='var')

pprint(tf.global_variables()) # print out all global variables in this graph
```

[<tf.Variable 'Variable:0' shape=() dtype=float32_ref>,
<tf.Variable 'Variable_1:0' shape=(2,) dtype=float32_ref>,
<tf.Variable 'var:0' shape=(3,) dtype=float32_ref>,
<tf.Variable 'var_1:0' shape=(2, 1) dtype=float32_ref>]



- a 不是變數名稱，var 才是！
- 每 assign 一次 Tensor, 就會在graph上多新建一個 Tensor 節點, 而不是覆蓋掉舊的 Tensor ！！(重要觀念)



兩種建立graph的方法

1. 在預設的 graph 上作業
2. 建立一個 `tf.Graph` 物件
 - a. 利用 `with graph.as_default():` 的語法來建構網路
 - b. 開啟 session 時, 要傳入指定的 graph



兩種建立graph的方法(續)

method 1

```
a = tf.Variable(2.0, dtype=tf.float32, name='a')
b = tf.Variable(3.0, dtype=tf.float32, name='b')
c = tf.Variable(5.0, dtype=tf.float32, name='c')
init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)
print(sess.run((a + b) * c))
sess.close()
```

method 2

```
my_graph = tf.Graph() # create a graph object

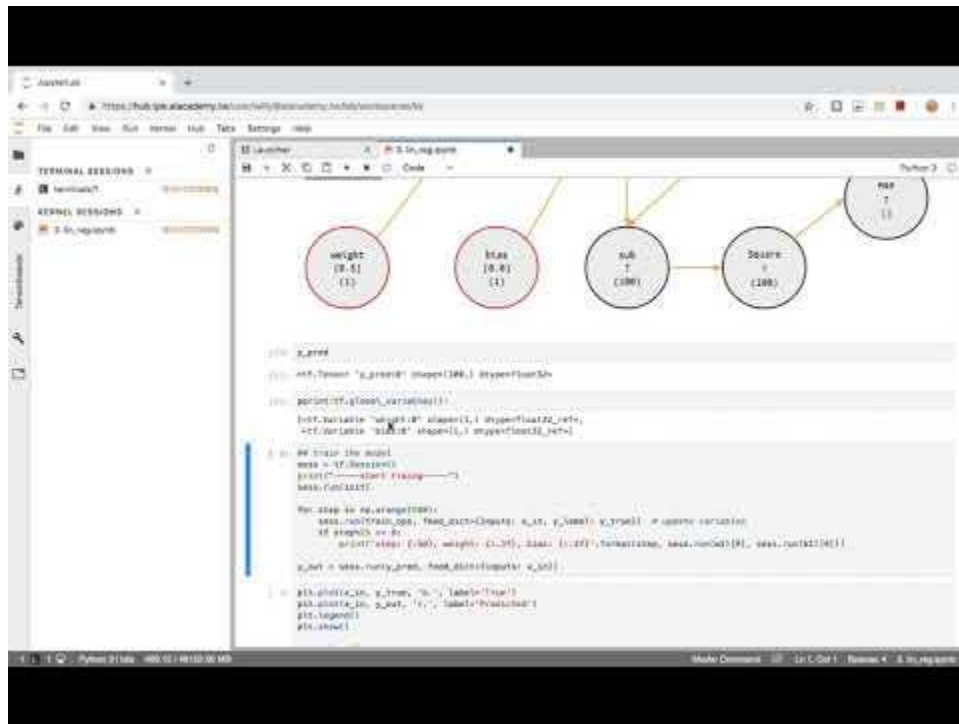
with my_graph.as_default():

    d = tf.Variable(1.0, dtype=tf.float32, name='d')
    e = tf.Variable(3.0, dtype=tf.float32, name='e')
    f = tf.Variable(7.0, dtype=tf.float32, name='f')
    init = tf.global_variables_initializer()

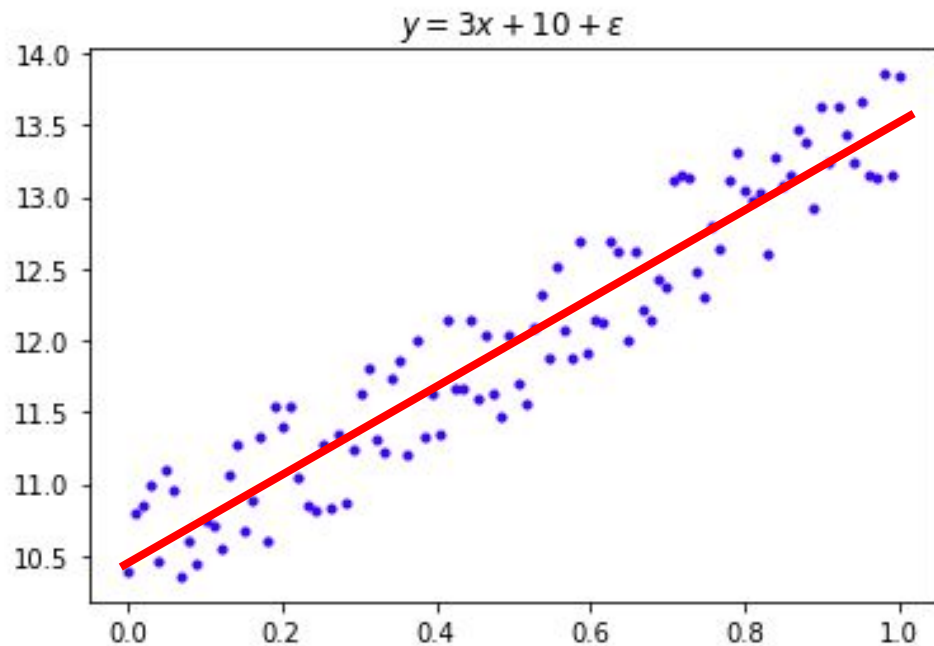
sess = tf.Session(graph=my_graph) # pass a specific graph
sess.run(init)
print(sess.run((d + e) * f))
sess.close()
```



線性迴歸模型 (3. lin_reg.ipynb)



目標：找出紅色的回歸線段 ($y = 3x + 10$)



實作Gradient descent

- step 1. 建立 forward pass network
- step 2. 算出 loss function
- step 3. 挑選優化器優化 loss function
- step 4. 開啟 session 更新參數



實作Gradient descent(續)

```
# step 1
inputs = tf.placeholder(dtype=tf.float32, shape=[100], name='X')
y_label = tf.placeholder(dtype=tf.float32, shape=[100], name='label')

w1 = tf.Variable([0.5], dtype=tf.float32, name='weight')
b1 = tf.Variable([0.0], dtype=tf.float32, name='bias')
y_pred = tf.add(tf.multiply(w1, inputs), b1, name='y_pred')

# step 2
loss = tf.reduce_mean(tf.square(y_pred - y_label), name='mse')

# step 3
optim = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_ops = optim.minimize(loss)

# step 4
sess = tf.Session()
sess.run(init)

for step in np.arange(500):
    sess.run(train_ops, feed_dict={inputs: x_in, y_label: y_true})
```

回歸模型的損失函數
選用均方誤差



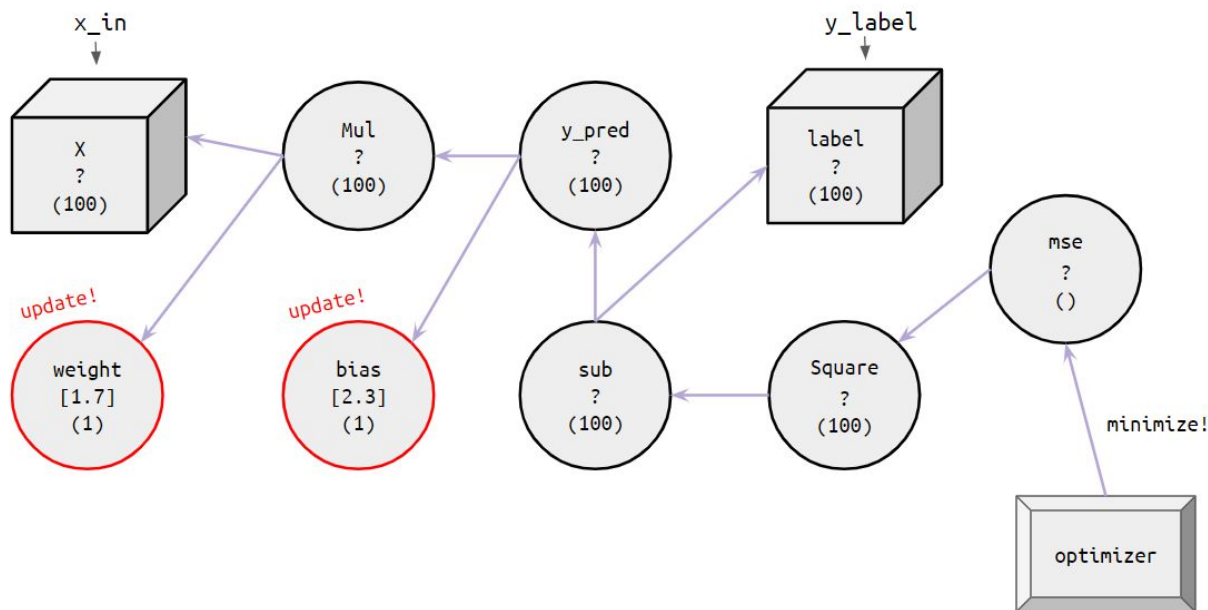
Optimizers in Tensorflow

- 梯度下降的優化器都放在 `tf.train` 下面的類別
- 優化器的種類繁多，例如：
 - `tf.train.GradientDescentOptimizer`
 - `tf.train.AdagradOptimizer`
 - `tf.train.MomentumOptimizer`
 - `tf.train.AdamOptimizer`
- 用法都是創立一個優化器物件，再使用 `minimize` 這個方法。

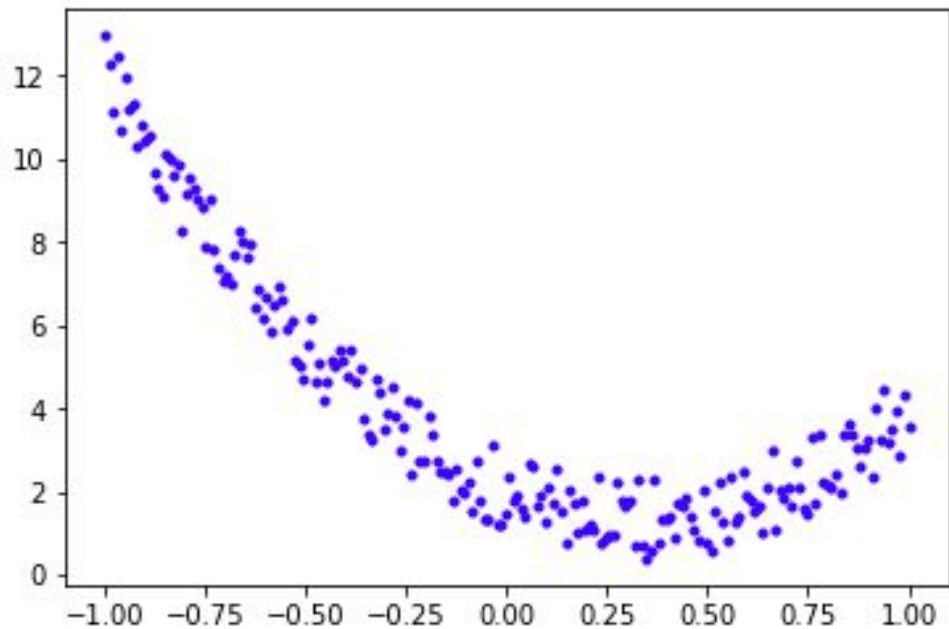


Optimizers in Tensorflow (續)

- 當使用 `minimize(loss)` 這個方法時，優化器會對組成 loss 上游全部的變數進行梯度下降。



練習：找出的回歸線段($y = 6x^2 - 4x + 1$)

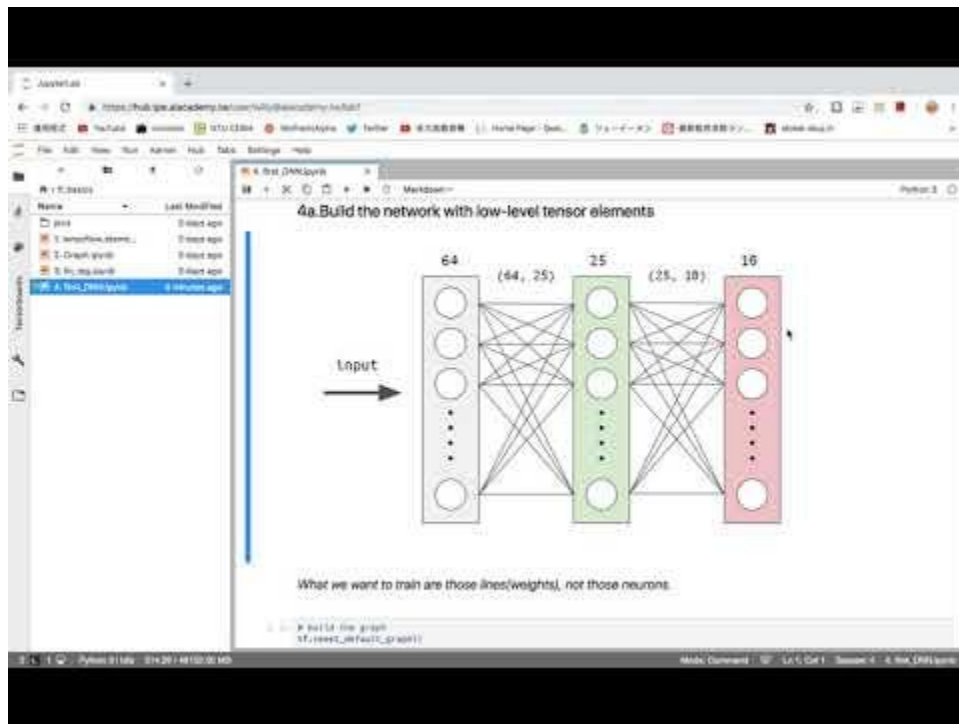


0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9



利用Tensorflow建構神經網路

神經網路處理手寫數字辨識 (4. first_DNN.ipynb)

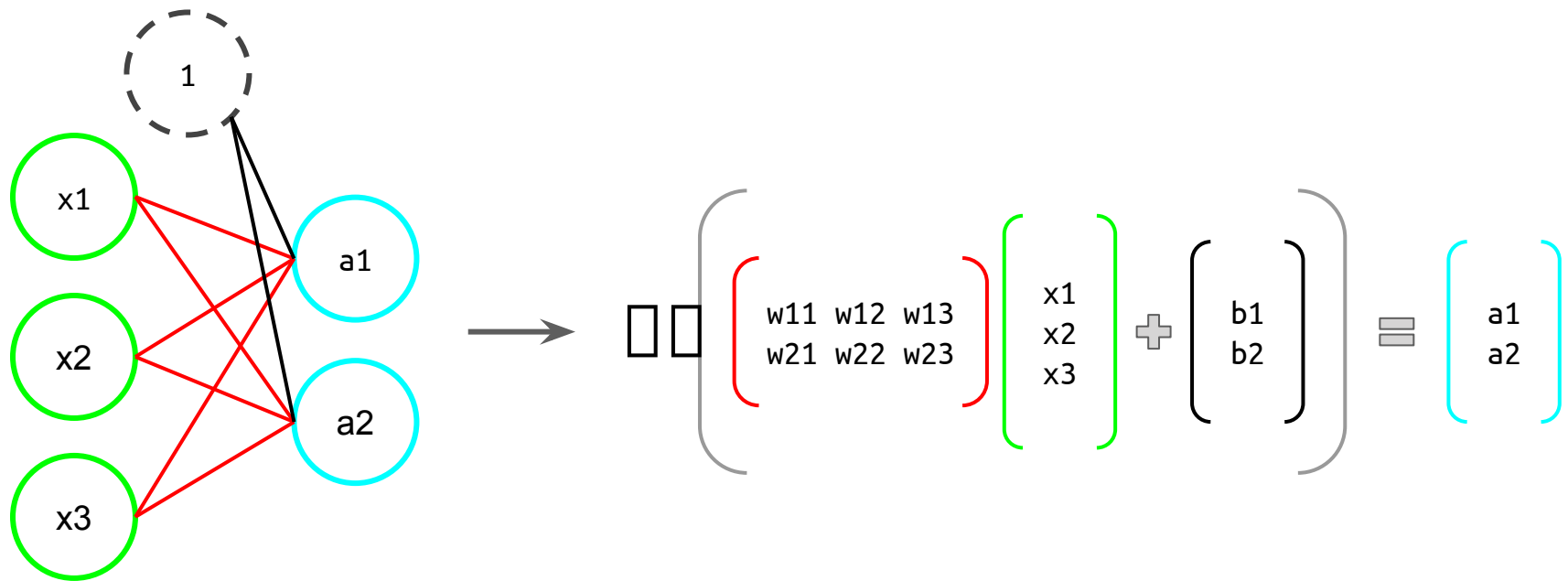


神經網路的本質

- 將輸入資料乘上矩陣(weight)，並通過非線性層得到新的轉換維度。
- 神經網路疊了好多層：乘了好多次矩陣！
- 利用梯度下降法得到可使 loss 最小的矩陣

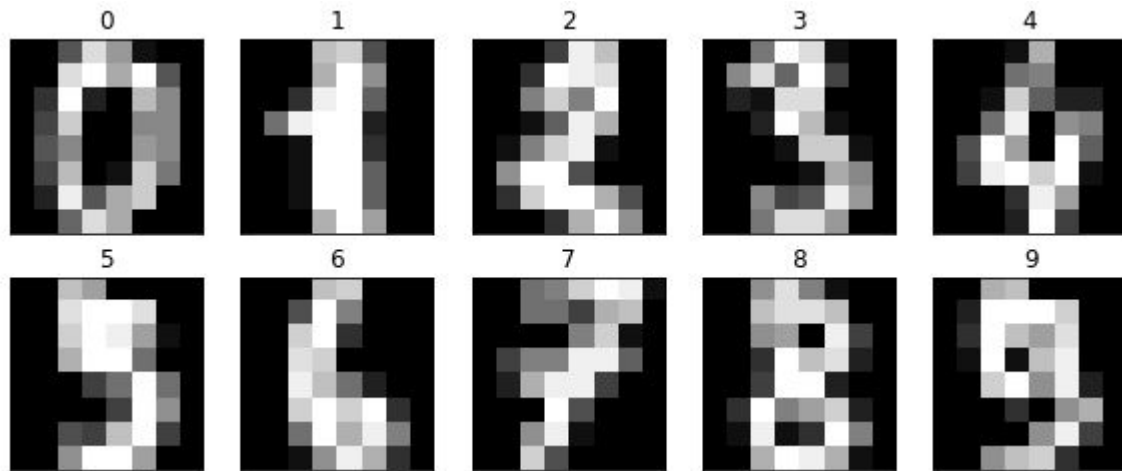


神經網路的本質(續)



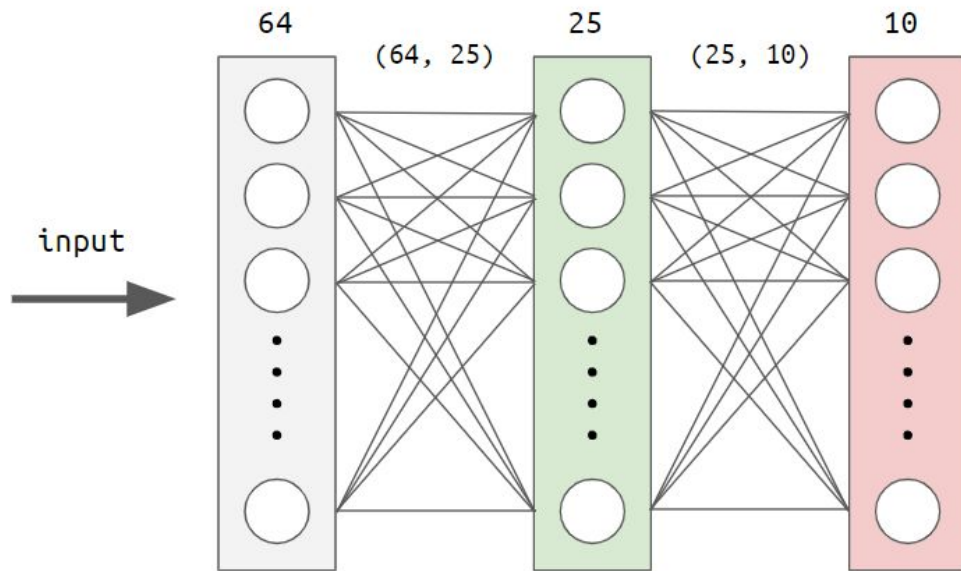
架構網路前：準備資料

- 觀察資料：數字資料有64維



模型大綱

- 64(圖片維度) -> 25(隱藏層) -> 10(分類類別)



架構 forward pass 網路(方法一)

- 手動創建矩陣變數

```
with tf.name_scope('input'):
    x_input = tf.placeholder(shape=(None, x_train.shape[1]), name='x_input', dtype=tf.float32)
    y_out = tf.placeholder(shape=(None, y_train.shape[1]), name='y_label', dtype=tf.float32)
```

```
with tf.variable_scope('hidden_layer'):
```

```
    w1 = tf.Variable(tf.truncated_normal(shape=[x_train.shape[1], 25], stddev=0.1),
                      name='weight1',
                      dtype=tf.float32)
    b1 = tf.Variable(tf.constant(0.0, shape=[25]),
                      name='bias1',
                      dtype=tf.float32)
```

```
    z1 = tf.add(tf.matmul(x_input, w1), b1) # (None, 64) × (64, 25) + (None, 25) = (None, 25)
    a1 = tf.nn.relu(z1, name='h1_out')
```

需要形狀(64, 25)的矩陣當
weight
和長度(25)的向量當bias

```
with tf.variable_scope('output_layer'):
```

```
    w2 = tf.Variable(tf.truncated_normal(shape=[25, y_train.shape[1]], stddev=0.1),
                      name='weight2',
                      dtype=tf.float32)
    b2 = tf.Variable(tf.constant(0.0, shape=[y_train.shape[1]]),
                      name='bias2',
                      dtype=tf.float32)
```

```
    output = tf.add(tf.matmul(a1, w2), b2, name='output')
```

需要形狀(25, 10)的矩陣當
weight
和長度(10)的向量當bias



架構 forward pass 網路(方法二)

- 利用 tf.layers 的 API

```
with tf.name_scope('input'):
    x_input = tf.placeholder(shape=(None, x_train.shape[1]),
                             name='x_input',
                             dtype=tf.float32)
    y_out = tf.placeholder(shape=(None, y_train.shape[1]),
                           name='y_label',
                           dtype=tf.float32)

with tf.variable_scope('hidden_layer'):
    x_h1 = tf.layers.dense(inputs=x_input, units=25, activation=tf.nn.relu)

with tf.variable_scope('output layer'):
    output = tf.layers.dense(x_h1, 10, name='output')
```

tf.layers.dense 會自動
創建形狀相符的 weight &
bias



計算損失函數

- 多類別的分類問題通常會使用交叉熵(cross entropy)來當作損失函數
- 交叉熵的計算需要兩個機率分布才能運算：
 - label (1, 0, 0) \leftrightarrow predict (0.8, 0.1, 0.1)
loss = $-\ln(1 \cdot 0.8) = 0.223$
 - label (1, 0, 0) \leftrightarrow predict (0.3, 0.5, 0.2)
loss = $-\ln(1 \cdot 0.3) = 1.204$
- 預測的分布與正確答案越像, loss 越小



計算損失函數(續)

- tensorflow 中的 `tf.nn.softmax_cross_entropy_with_logits_v2` 會自動將輸入的預測經過 softmax, 將預測轉換成機率分布型態, 因此不需要事先做 softmax。

Wrong!!

```
output = tf.layers.dense(x_h1, 10, name='output', activation=tf.nn.softmax)
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=output, labels=y_out))
```

Right!!

```
output = tf.layers.dense(x_h1, 10, name='output')
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=output, labels=y_out))
```



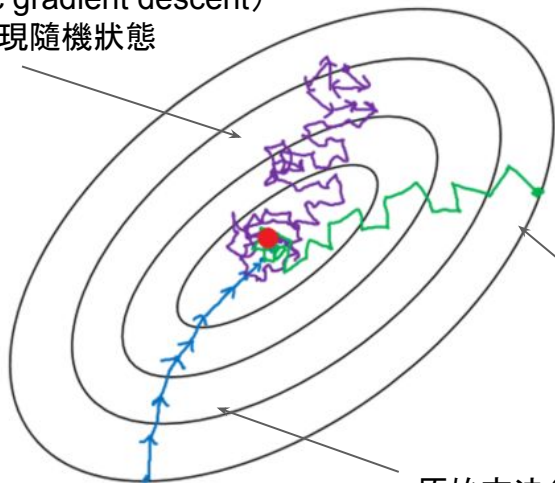
批次訓練

- 最基本的梯度下降法是看過所有的資料，算出一個 total loss，在對 loss 做梯度下降。
 - 看過所有資料所需的運算量太大，且更新速度慢。
- 替代方案之一：每讀一筆資料就算出 loss 並更新一次，雖然梯度下降的方向不是最佳的，但有機會最後走到 loss 最低的點。
 - 有一千筆資料，全部丟進網路就可以更新一千次，但穩定度低。
- 替代方案之二：每讀N筆資料就更新一次。
 - 前面兩個方法的折衷，有更新速度快以及穩定度較高的特性。



批次訓練示意圖

替代方案一（又稱 stochastic gradient descent）
梯度下降次數多次但方向呈現隨機狀態



替代方案二（又稱 mini-batch gradient descent）
梯度下降次數多次且方向較穩定

原始方法（又稱 batch gradient descent）
梯度下降的方向為最佳方向



訓練網路

```
sess = tf.Session()

sess.run(tf.global_variables_initializer())

for i in tqdm_notebook(range(epochs)):

    total_batch = len(x_train) // batch_size

    for j in range(total_batch):
```

```
        batch_idx_start = j * batch_size
        batch_idx_stop = (j+1) * batch_size
```

```
        x_batch = x_train[batch_idx_start : batch_idx_stop]
        y_batch = y_train[batch_idx_start : batch_idx_stop]
```

```
        this_loss, this_acc, _ = sess.run([loss, compute_acc, train_step],
                                           feed_dict={x_input: x_batch, y_out: y_batch})
```

```
x_train, y_train = shuffle(x_train, y_train)
```

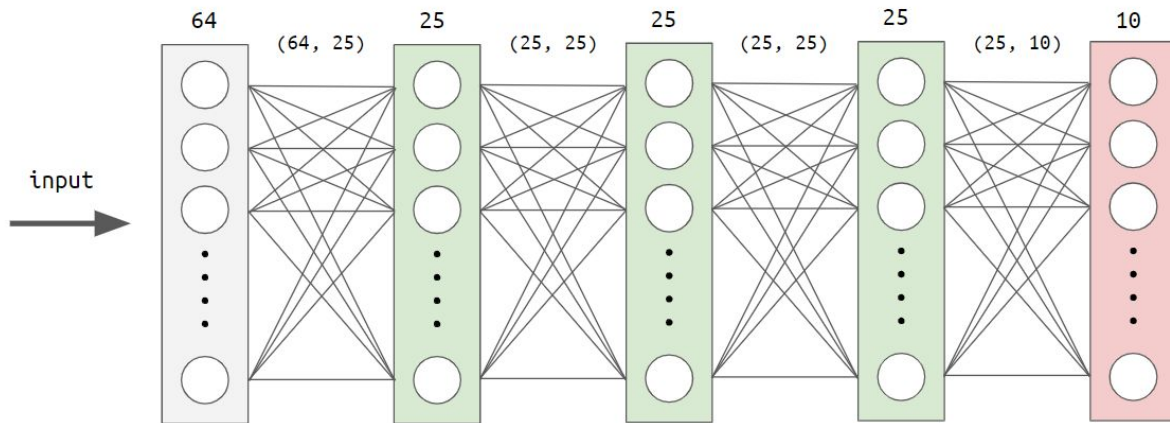
神經網路通常使用 mini-batch 的方法訓練，
程式中使用雙重迴圈：
裡面的迴圈將所有的 mini-batch 都訓練一遍，
稱作一個 epoch。
外面的迴圈則重複執行多個 epoch。

每訓練完所有的資料
就要重新打亂一次避免過擬和



練習時間

- 嘗試利用方法一疊出三層隱藏層都為25維的神經網路
- 嘗試利用方法二疊出三層隱藏層都為25維的神經網路



Save the model

- 在 graph 上多創建一個 saver 物件

```
saver = tf.train.Saver()
```

- 模型訓練完畢後，執行 saver

```
saver.save(sess, "./save_model/checkpoint_weight.ckpt")
```

訓練好的 session

檔案存放的位置



Load the model

- 載入模型有兩種方法：
 - 載入模型的權重
 - 必須將 graph 重寫一遍
 - 載入整個模型的 graph
 - 執行 graph 時必須先得知原本 graph 某些元素的名稱



載入權重

- 將原本的 graph 結構完整地寫一遍
 - 不可新增或刪減變數
 - 變數名稱也不可更變
- 開啟一個新的 session 並執行

```
saver.restore(sess, "./save_model/checkpoint_weight.ckpt")
```



載入 graph

- 開啟新的 session 並執行

```
filename = './save_model/checkpoint_weight.ckpt'  
loader = tf.train.import_meta_graph(filename + '.meta')  
loader.restore(sess, filename)
```

- 利用 `sess.graph.get_tensor_by_name` 取得 graph 上的元素



練習時間

- 嘗試將之前訓練過的手寫數字辨識模型儲存下來，並重新載入確認
流程正確

