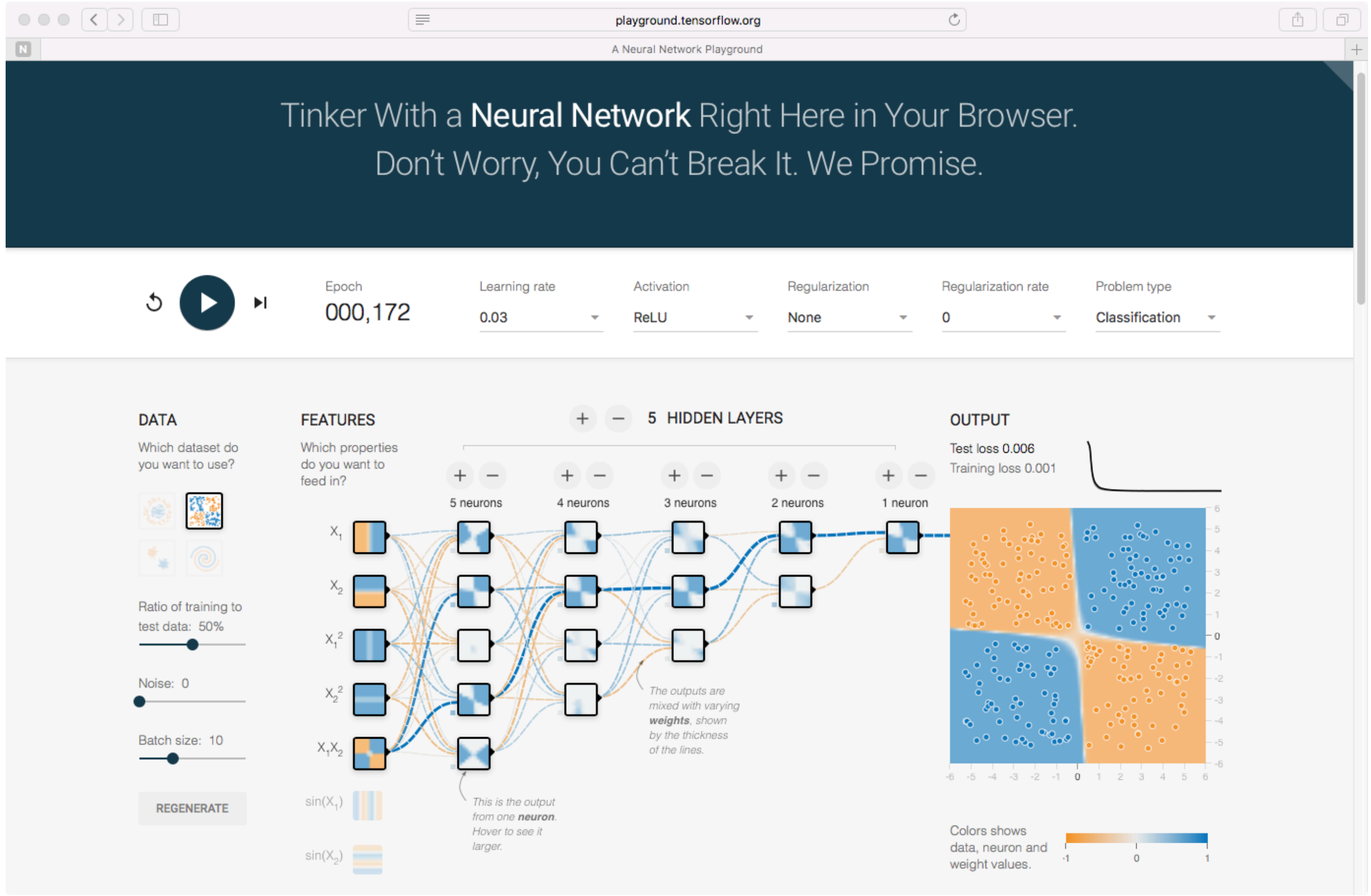
TensorFlow

吳佳諺　老師

TensorFlow

1.TensorFlow
2.TensorFlow的變數
3.圖形和Sessions
4.TensorFlow placeholder
5.範例實作
6.執行計算圖

1.TensorFlow



Tensor張量

```
3 # 維度為0的張量; a scalar with shape []
[1., 2., 3.] # 維度為1的張量; shape [3]
[[1., 2., 3.], [4., 5., 6.]] #維度為2的張量; shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]]
# 維度為3的張量shape [2, 1, 3]
```

張量的形狀shape

- shape代表該維度有幾個元素

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

In [13]: X=np.array([1,2,3])
         X.shape

Out[13]: (3,)

In [11]: Y=np.array(3)
         Y.shape

Out[11]: ()
```

張量Tensor

```
In [14]: Z=np.array([[1., 2., 3.], [4., 5., 6.]])
         Z.shape

Out[14]: (2, 3)

In [16]: K=np.array([[[1., 2., 3.]], [[7., 8., 9.]]])
         K.shape

Out[16]: (2, 1, 3)
```

張量的資料型態

- tf.Variable
- tf.Constant
- tf.Placeholder
- tf.SparseTensor

張量維度為0的變數

```
mammal = tf.Variable("Elephant",
tf.string)
ignition = tf.Variable(451, tf.int16)
floating = tf.Variable(3.14159265359,
tf.float64)
its_complicated = tf.Variable((12.3,
-4.85), tf.complex64)
```

張量維度為1的變數

```
mystr = tf.Variable(["Hello"], tf.string)
cool_numbers  = tf.Variable([3.14159, 2.71828], tf.float32)
first_primes = tf.Variable([2, 3, 5, 7, 11], tf.int32)
its_very_complicated = tf.Variable([(12.3, -4.85), (7.5, -6.23)], tf.complex64)
```

張量維度為2

```
mymat = tf.Variable([[7],[11]], tf.int16)
myxor = tf.Variable([[False, True],[True, False]], tf.bool)
linear_squares = tf.Variable([[4], [9], [16], [25]], tf.int32)
squarish_squares = tf.Variable([ [4, 9], [16, 25] ],
tf.int32)
rank_of_squares = tf.rank(squarish_squares)
mymatC = tf.Variable([[7],[11]], tf.int32)
```

改變張量的shape

```
rank_three_tensor = tf.ones([3, 4, 5])
matrix = tf.reshape(rank_three_tensor, [6, 10])  # 將張量改成6*10的矩陣
matrixB = tf.reshape(matrix, [3, -1])  # matrix. -1 是計算該維度的大小,20.
matrixAlt = tf.reshape(matrixB, [4, 3, -1])  # -1是計算該維度的大小,5.
```

2.TensorFlow的變數

建立張量的變數使用get_variable()函數

```
my_variable = tf.get_variable("my_variable", [1, 2, 3])
```

為三維張量其shape為[1,2,3],資料型態預設為float32

訓練前需初始化變數

tf.global_variables_initializer()
再訓練前需初始化變數使用global_variables_initializer()
函數
session.run(tf.global_variables_initializer())
執行完session.run()後,就初始化變數了

線性模型LinearModel

- xx=tf.placeholder(tf.float32)
- Wi=tf.Variable([0.8],dtype=tf.float32)
- b=tf.Variable([-0.8],dtype=tf.float32)

- LinearModel=Wi*xx+b

執行線性模型

執行時,將x串列帶入placeholder去作運算

- print(sess.run(LinearModel,{xx:[5,6,7,8,9]}))

評估模型的成效

建立好線性模型之後，我們會需要評估模型的成效，所以還要定義一個 placeholder yy，用於儲存正確的答案,所以預估值LinearModel減去原來的yy值,就是它的誤差：

```
yy = tf.placeholder(tf.float32)
```

平方

```
s_delta = tf.square(LinearModel – yy)
```

平方和

```
cost = tf.reduce_sum(s_delta)
```

執行成本函數

- 執行成本函數，並將xx和yy串列帶入運算
- print(sess.run(cost, {xx:[5,6,7,8,9], yy:[-1,-2,-3,-4,-5]}))

assign動態調整參數

可以透過動態調整不同的參數組合，讓誤差值cost最小。若要調整 variable 的值可以使用assign函數。

```
f_Wi = tf.assign(Wi, [–0.5])
f_b = tf.assign(b, [0.5])
sess.run([f_Wi, f_b])
print(sess.run(cost, {xx:[5,6,7,8,9], yy:[–1,–2,–3,–4,–5]}))
```

TensorFlow常數生成函數

函數名稱	功能	範例
tf.zeros	產生全0的陣列	tf.zeros([2,3],int32)->[[0,0,0],[0,0,0]]
tf.ones	產生全1的陣列	tf.ones([2,3],int32)->[[1,1,1],[1,1,1]]
tf.fill	產生一個全部為給定數字的陣列	tf.fill([2,3],9)->[[9,9,9],[9,9,9]]
tf.constant	產生一個給定值的常數	tf.constant([1,2,3])->[1,2,3]

3.圖形和Sessions

- TensorFlow使用圖形來代表計算操作的相依
- 使用TensorFlow的session來平行執行圖形
- Dataflow使用平行運算

輸入TensorFlow

- import tensorflow as tf

設定兩個常數節點

```
>>> import tensorflow as tf
>>> node1 = tf.constant(3.0,
dtype=tf.float32)
>>> node2 = tf.constant(4.0) # also
tf.float32 implicitly
>>> print(node1, node2)
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(),
dtype=float32)
```

資料型態有float32,int32或字串string

“Const:0”為名子

執行Session

```
>>> sess = tf.Session()
>>> print(sess.run([node1,node2]))
[3.0, 4.0]
>>> sess.close()
```

tf.Session()為建立一個執行會話sess

使用sess來執行此運算[node1,node2]
sess.run([node1,node2])

關閉執行會話使用sess.close(),讓使用資源可以被釋放

with上下文管理

```
>>> with tf.Session() as sess:
...     sess.run([node1,node2])
...
[3.0, 4.0]
```

建立一個對話,並透過Python中的上下文管理器來管理這個會話Session

sess.run()則使用這個建立好的會話來執行結果
當with子句上下文結束時,會話關閉和資源釋放也自動完成,因此沒有用close()函數關掉Session會話

eval()函數可以用來取張量的值

```
import tensorflow as tf
x = tf.constant(3.1,name='x')
y = tf.constant(2.5,name='y')
result = x+y

with tf.Session() as sess:
    print(result.eval())
```

```
5.6
```

4.TensorFlow placeholder


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

輸入參數dtype為資料型態,shape維度,name為名稱

placeholder 是一種可以讓計算圖形保留輸入欄位的節點，並允許實際的輸入值留到後來再指定。

TensorFlow placeholder

這種方式像定義一個函數，其接受x與y兩個輸入參數，並且進行加法運算。

x = tf.placeholder(tf.float32)

```
y = tf.placeholder(tf.float32)
z = tf.add(x, y)
```

TensorFlow placeholder

接著在執行計算圖形時，只要在run的feed_dict參數指定輸入的數值，即可進行運算

```
with tf.Session() as sess:
  print(sess.run(z,feed_dict={x:5.5,y:2.1}))
```

placeholder讓我們在計算時才進行設定變數數值

{x:5.5,y:2.1}為字典

TensorFlow基礎數學運算

Arithmetic Operators

TensorFlow provides several operations that you can use to add basic arithmetic operators to your graph.

- `tf.add`
- `tf.subtract`
- `tf.multiply`
- `tf.scalar_mul`
- `tf.div`
- `tf.divide`

TensorFlow基礎數學運算

- `tf.truediv`
- `tf.floordiv`
- `tf.realdiv`
- `tf.truncatediv`
- `tf.floor_div`
- `tf.truncatemod`
- `tf.floormod`
- `tf.mod`
- `tf.cross`

TensorFlow基礎數學運算

TensorFlow provides several operations that you can use to add basic mathematical functions to your g

- `tf.add_n`
- `tf.abs`
- `tf.negative`
- `tf.sign`
- `tf.reciprocal`

TensorFlow基礎數學運算

TensorFlow provides several operations that you can use to add basic mathematical functions to

- `tf.add_n`
- `tf.abs`
- `tf.negative`
- `tf.sign`
- `tf.reciprocal`

TensorFlow基礎數學運算

- `tf.square`
- `tf.round`
- `tf.sqrt`
- `tf.rsqrt`
- `tf.pow`

建立計算圖

In [8]:
```
import tensorflow as tf
x = tf.placeholder(tf.float32,name='x')
y = tf.placeholder(tf.float32,name='y')
z = tf.add(x, y,name='sum')

with tf.Session() as sess:
    print(sess.run(z,feed_dict={x:5.5,y:2.1}))
```

```
7.6
```

FileWriter(‘紀錄的路徑’)

- tf.summary.merge_all()
- filewriter=tf.summary.FileWriter(‘test/sum’,sess.graph)

FileWriter('紀錄的路徑')

```
import tensorflow as tf
x = tf.constant(3.1,name='x')
y = tf.constant(2.5,name='y')
result = x+y

with tf.Session() as sess:
    print(result.eval())
```

```
5.6
```

```
writer = tf.summary.FileWriter("log/simple3_log", tf.get_default_graph())
writer.close()
```

5.範例實作


```
import tensorflow as tf
```

```
xx=tf.placeholder(tf.float32)
Wi=tf.Variable([0.8],dtype=tf.float32)
b=tf.Variable([-0.8],dtype=tf.float32)
```

```
LinearModel=Wi*xx+b
```

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

```
print(sess.run(LinearModel,{xx:[5,6,7,8,9]}))
```

```
[ 3.20000005  4.          4.79999971  5.5999999   6.4000001 ]
```

```
yy = tf.placeholder(tf.float32)
```

範例實作

```
s_delta = tf.square(LinearModel - yy)
cost = tf.reduce_sum(s_delta)
```

```
print(sess.run(cost, {xx:[5,6,7,8,9], yy:[-1,-2,-3,-4,-5]}))
```

```
174.6
```

```
f_Wi = tf.assign(Wi, [-0.5])
f_b = tf.assign(b, [0.5])
sess.run([f_Wi, f_b])
print(sess.run(cost, {xx:[5,6,7,8,9], yy:[-1,-2,-3,-4,-5]}))
```

```
0.45
```

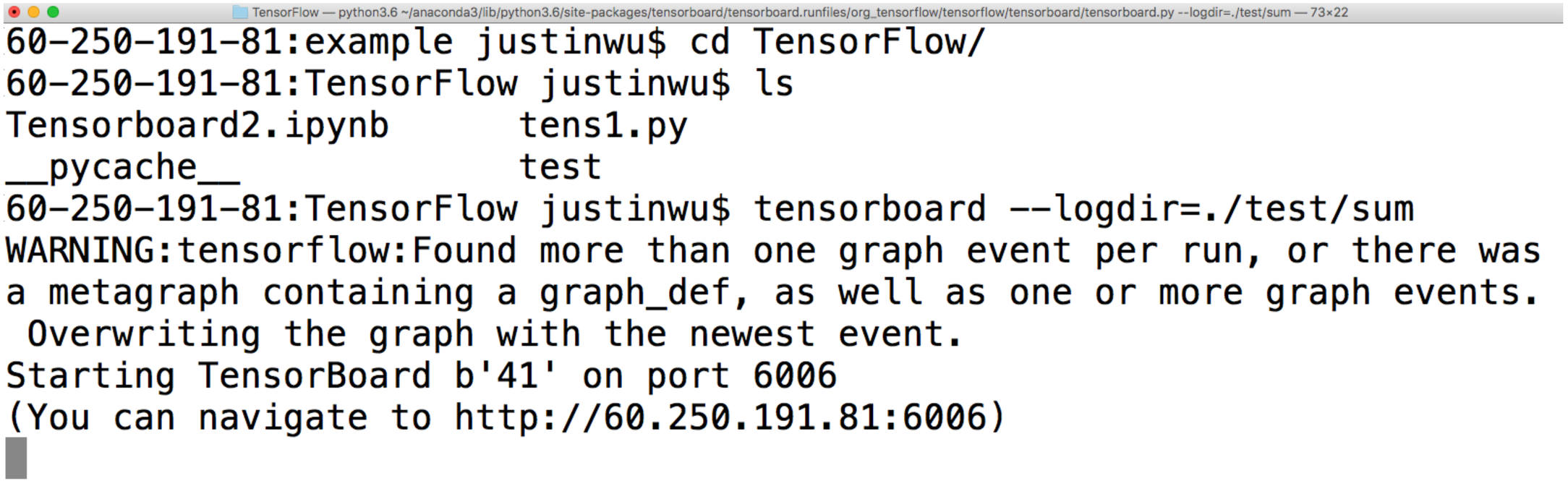
6.執行計算圖

- **執行tensorboard且設定目錄**
- 我們範例檔案所在目錄,含有test目錄
- 將記錄檔存在test/sum目錄下(這是在Mac底下的目錄)
- 執行在test/sum目錄下的summary紀錄
- 打開指定的瀏覽器IP或網址來打開TensorBoard
- 選取Graphs

執行tensorboard且設定目錄

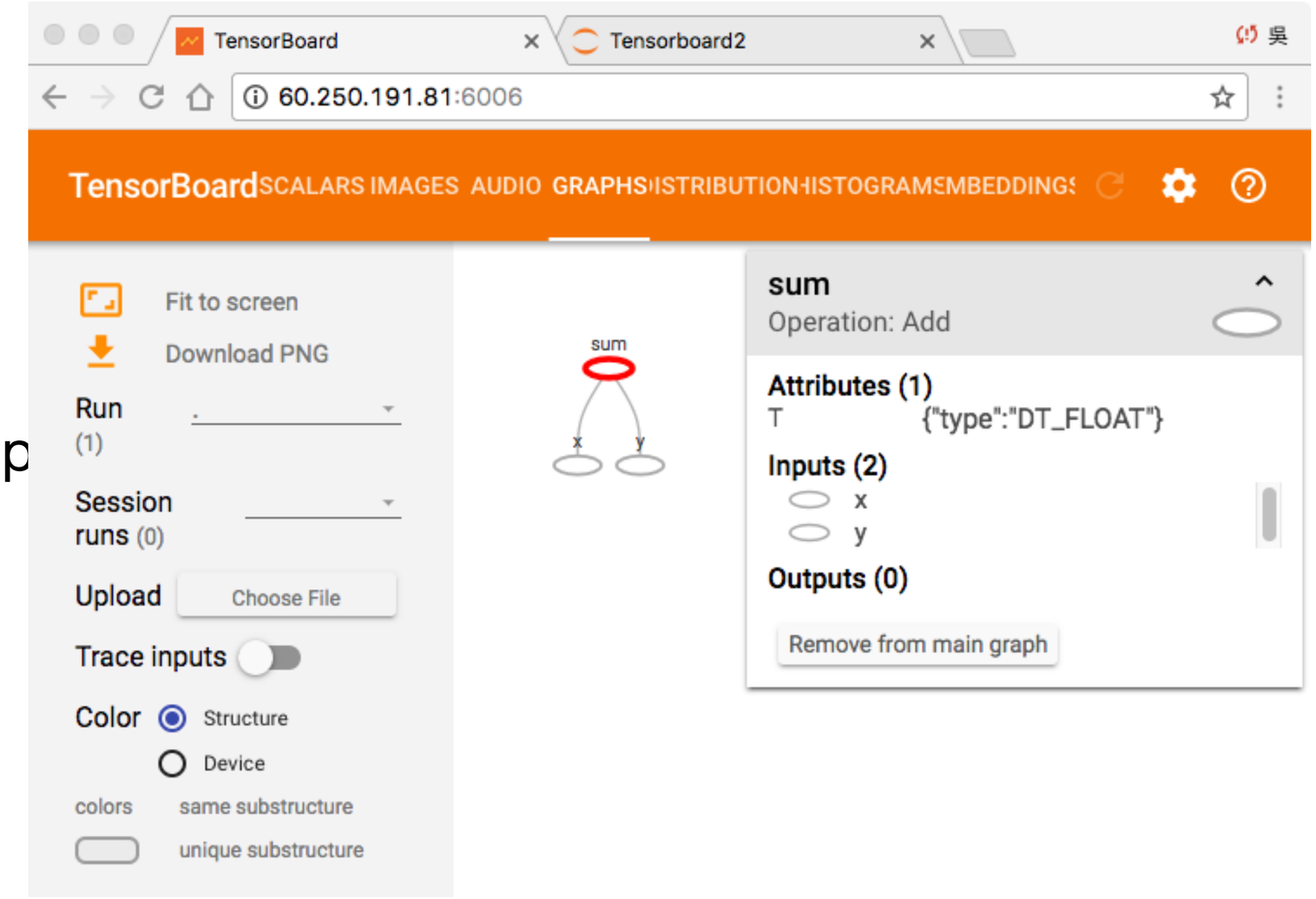
- —logdir為指定紀錄的目錄(這是在Mac環境下的目錄)

- 打開指定的網頁http://IP:連接埠



選取GRAPHS

- pp


TensorBoard
Tensorboard2
60.250.191.81:6006
TensorBoard SCALARS IMAGES AUDIO GRAPHS DISTRIBUTIONS HISTOGRAMS EMBEDDINGS
Fit to screen
Download PNG
Run (1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
colors
same substructure
unique substructure
sum
x
y
sum
Operation: Add
Attributes (1)
T
{"type":"DT_FLOAT"}
Inputs (2)
x
y
Outputs (0)
Remove from main graph

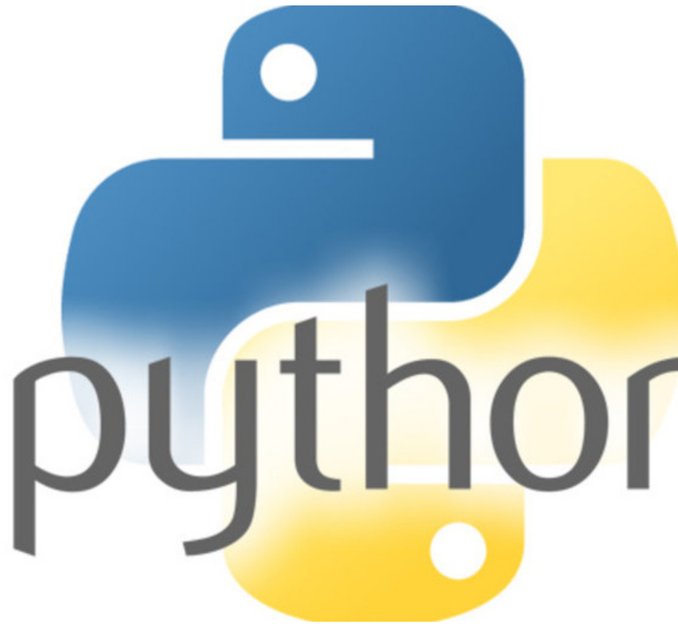
- Thanks.

TensorFlow
MNIST手寫辨識

吳佳諺 老師

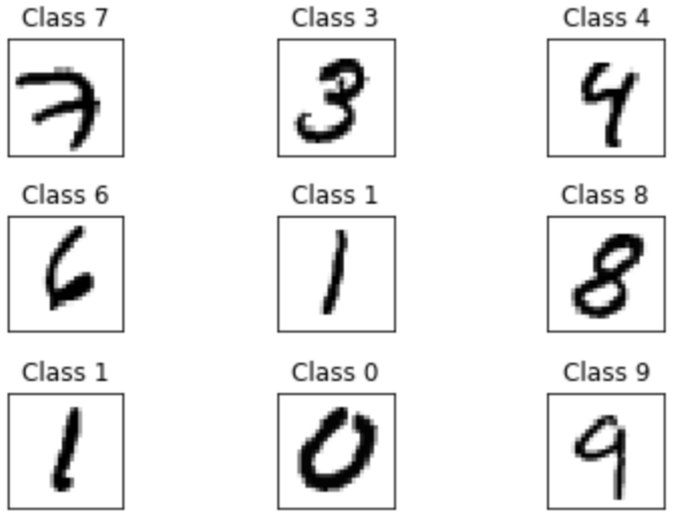
TensorFlow
手寫辨識演算法

1.載入mnist手寫辨識資料
2.W是權重,b是偏差,X是輸入
3.輸入X乘與權重W加bios,經過softmax得到y
4.訓練模型使用交叉熵的成本
5.使用梯度遞減微分求取最小交叉嫡
6.評估模型



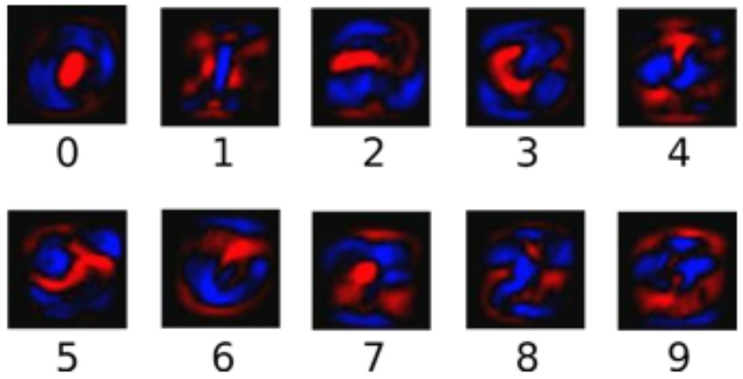
1.載入mnist手寫辨識資料

- 載入mnist手寫辨識資料
- from tensorflow.examples.tutorials.mnist import input_data
- mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)



載入mnist手寫辨識資料

紅色表示負的權重,藍色表示正的權重,手寫的值



2.W是權重,b是偏差,X是輸入

$$evidence_i = \sum_j W_{i,\ j} x_j + b_i$$

softmax()函数是歸一化

$$y = softmax(evidence)$$

softmax()函数是歸一化

$$softmax(evidence)=normalize((evidence))$$

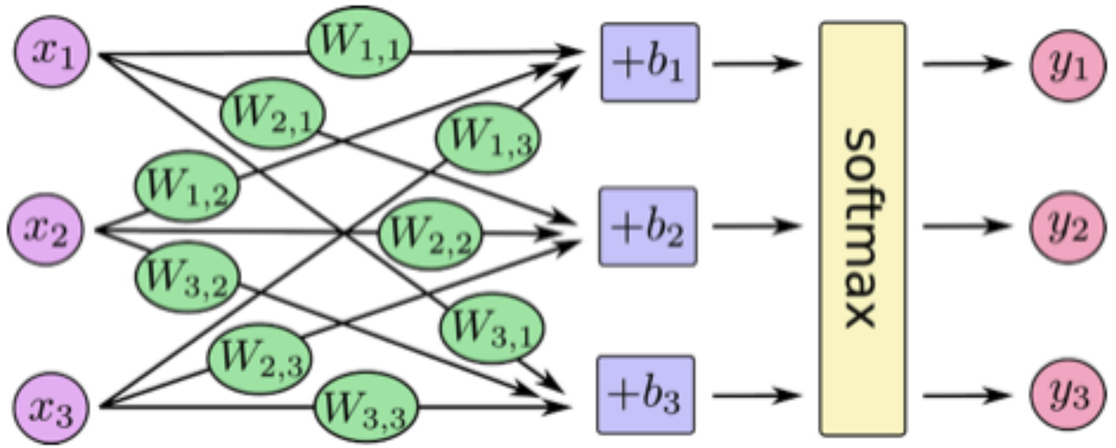
softmax()函数是歸一化

$$softmax(evidence)_i = \frac{(evidence_i)}{\sum_j (evidence_j)}$$

softmax()函数是歸一化

$$y=softmax(Wx+b)$$

3.輸入X乘與權重W加bios,經過softmax得到y



輸入X乘與權重W加bios,經過softmax得到y

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

W矩陣和X向量的內積加b為y

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

實作迴歸

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

4.訓練模型使用交叉熵的成本

- y是預測,y’是真實的標籤

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

使用交叉熵來實作

```
y_為正確的答案
y_ = tf.placeholder(tf.float32, [None, 10])

log(y)是取y對數log
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]))
```

tf.reduce_mean(input_tensor,axis=None,keep_dims=False, name=None,reduction_indices=None)

將tensor取平均，第二個參數代表沿著那一維度取平均
第二個，沿著第0維也就是列取平均得到「1.5，1.5」，
手指沿著row的方向
沿著第1維也就是column取平均得到[1,2]，沿著column方向

5.使用梯度遞減微分
求取最小交叉熵

```
train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

使用交談式會話

```
sess = tf.InteractiveSession()
初始化變數
tf.global_variables_initializer().run()
```

執行1000次訓練

```
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_:
batch_ys})
```

評估模型

```
correct_prediction = tf.equal(tf.argmax(y,1),
tf.argmax(y_,1))
```

y_為正確標籤

```
tf.argmax(y_,1)
```

y為預測標籤

```
tf.argmax(y,1)
```

tf.cast()資料型態轉換

將布林值轉為浮點數

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

測試資料的預測結果

```
print(sess.run(accuracy, feed_dict={x: mnist.test.images,
y_: mnist.test.labels}))
```

- Thanks.

TensorFlow
手寫辨識實作

吳佳諺 老師

TensorFlow

1.TensorFlow函數
2.手寫辨識實作
3.顯示手寫訓練圖片
4.tf.matmul()為矩陣內積相乘
5.Class GradientDescentOptimizer類別

tf.reduce_mean

tf.reduce_mean(input_tensor,axis=None,keep_dims=False, name=None,reduction_indices=None)

將tensor取平均，第二個參數代表沿著那一維度取平均
第二個，沿著第0維也就是列取平均得到「2，2」，手指沿著row的方向
沿著第1維也就是column取平均得到[1,3]，沿著column方向

tf.reduce_mean

```
import tensorflow as tf
```

```
x=tf.Variable([[1,1],[3,3]],dtype=tf.float32)
```

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

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

```
2.0
```

tf.reduce_mean

```
x1=tf.Variable([[1,1],[3,3]],dtype=tf.float32)
```

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

```
#column欄取平均
print(sess.run(tf.reduce_mean(x1,0)))
```

```
[ 2.  2.]
```

tf.reduce_mean

```
x2=tf.Variable([[1,1],[3,3]],dtype=tf.float32)
```

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

```
#列取平均
print(sess.run(tf.reduce_mean(x2,1)))
```

```
[ 1.  3.]
```

2.手寫辨識實作

```
#import numpy as np
import argparse
import sys
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
FLAGS = None


mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

print(mnist.train.images[0])
  # Create the model
```

下載MNIST手寫辨識檔案

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

0到1灰階的值28*28像素

```
   0.
  0.          0.          0.          0.          0.35294119  0.54117
65
  0.92156869  0.92156869  0.92156869  0.92156869  0.92156869  0.92156
869
  0.98431379  0.98431379  0.97254908  0.99607849  0.96078438  0.92156
869
  0.74509805  0.08235294  0.          0.          0.          0.
   0.
  0.          0.          0.          0.          0.          0.
  0.54901963  0.98431379  0.99607849  0.99607849  0.99607849  0.99607
849
  0.99607849  0.99607849  0.99607849  0.99607849  0.99607849  0.99607
849
  0.99607849  0.99607849  0.99607849  0.99607849  0.74117649  0.09019
608
  0.          0.          0.          0.          0.          0.
   0.
  0.          0.          0.          0.88627458  0.99607849  0.81568
```

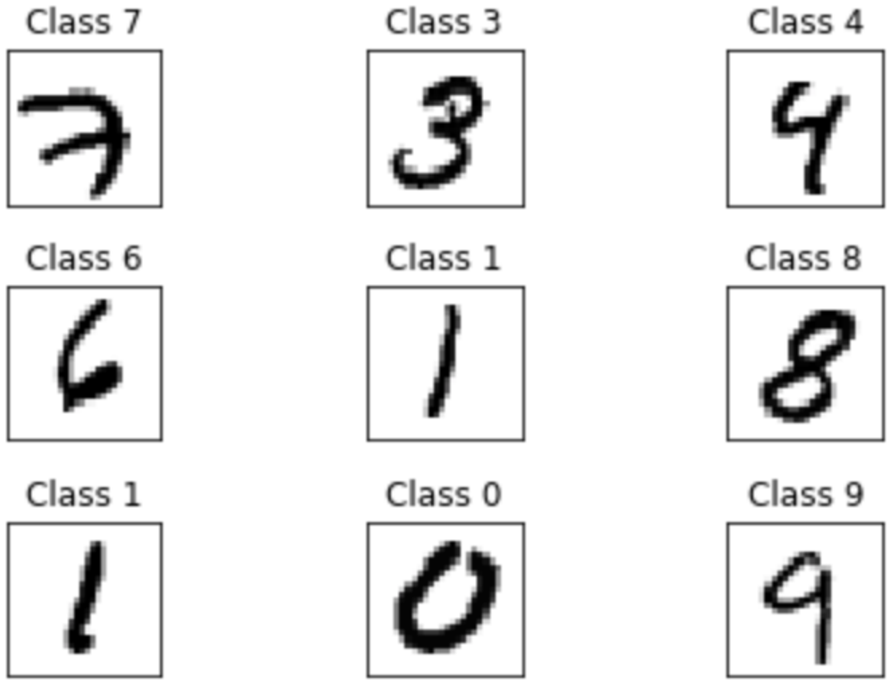
3.顯示手寫訓練圖片

- plt.imshow(mnist.train.images[i].reshape(28,28), cmap='binary',

- ```
import numpy as np

fig = plt.figure()
for i in range(9):
  number=np.argmax(mnist.train.labels[i])
  #print(number),argmax顯示陣列最大值的索引
  plt.subplot(3,3,i+1)
  plt.tight_layout()
  plt.imshow(mnist.train.images[i].reshape(28,28), cmap='binary',
             interpolation='none')
  plt.title("Class {}".format(number))
  plt.xticks([])
  plt.yticks([])
fig
```
```

顯示手寫訓練圖片

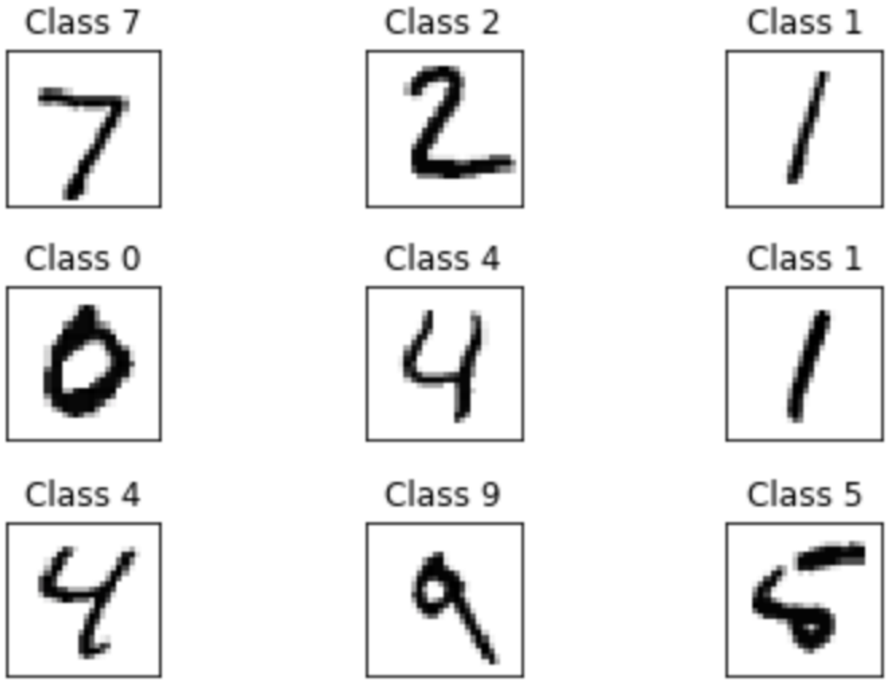

Class 7
Class 3
Class 4
Class 6
Class 1
Class 8
Class 1
Class 0
Class 9

顯示手寫測試圖片

- plt.imshow(mnist.test.images[i].reshape(28,28), cmap='binary', interpolation='none')

```
fig = plt.figure()
for i in range(9):
    #test資料
  number=np.argmax(mnist.test.labels[i])
  #print(number),argmax顯示陣列最大值的索引
  plt.subplot(3,3,i+1)
  plt.tight_layout()
  plt.imshow(mnist.test.images[i].reshape(28,28), cmap='binary',
             interpolation='none')
  plt.title("Class {}".format(number))
  plt.xticks([])
  plt.yticks([])
fig
```

4.顯示手寫測試圖片


Class 7
Class 2
Class 1
Class 0
Class 4
Class 1
Class 4
Class 9
Class 5

4.tf.matmul()為矩陣內積相乘

```
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

  # 定義成本和最佳化函數
y_ = tf.placeholder(tf.float32, [None, 10])
```

手寫圖形無限張,784個維度向量

- [None,784],為手寫圖形無限張,784個維度向量,由28*28像素得到.

- [784,10]為784維度,10個類

```
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

  # 定義成本和最佳化函數
y_ = tf.placeholder(tf.float32, [None, 10])
```

交叉熵函數

```
softmax_cross_entropy_with_logits(
    _sentinel=None,
    labels=None,
    logits=None,
    dim=-1,
    name=None
)
```

```
cross_entropy = tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
train_step = tf.train.GradientDescentOptimizer(
                        0.5).minimize(cross_entropy)

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

5.Class GradientDescentOptimizer類別

- 使用梯度下降法為最小化損失函數,增加優化.來最小化交叉熵
- train_step = tf.train.GradientDescentOptimizer(
- 0.5).minimize(cross_entropy)

6.執行預測結果91.88%

```
  # Train
for _ in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

  # Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images,
                                    y_: mnist.test.labels}))
```

```
0.9188
```

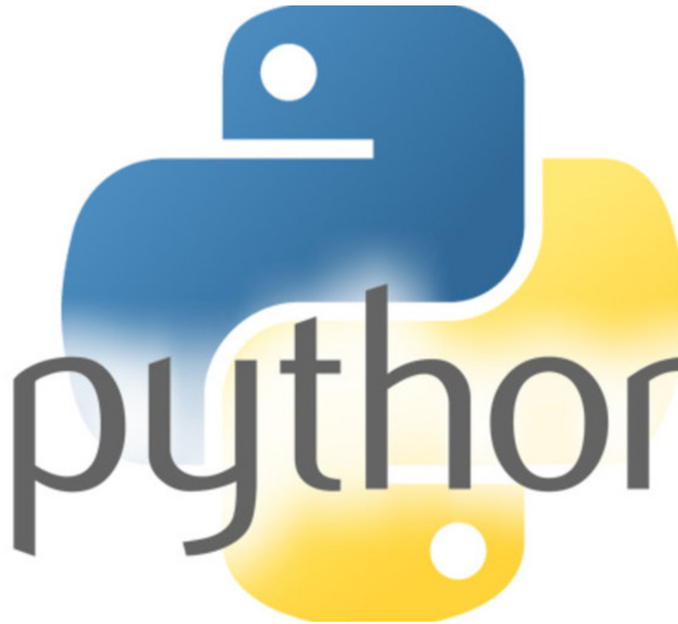
- Thanks.

TensorFlow
卷積深度學習手寫辨識

吳佳諺 老師

TensorFlow
卷積深度學習手寫辨識

1.輸入tensorflow函式庫
2.定義卷積conv2d和最大池化
3.深度學習函數deepnn(x)
4.第二卷積層對應 32 特徵向量到64
5.交叉熵最佳化
6.平行計算會議
7.準確度98.59%
8.TensorBoard的計算圖Graphs



1.輸入tensorflow函式庫


```
import argparse
import sys
import tempfile
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf
```

2.定義卷積conv2d和最大池化max_pool_2x2

```python
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')


def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)


def bias_variable(shape):
  initial = tf.constant(0.2, shape=shape)
  return tf.Variable(initial)
```

卷積-圖片影像特徵抽取

- tf.nn.conv2d

```
conv2d(
    input,
    filter,
    strides,
    padding,
    use_cudnn_on_gpu=True,
    data_format='NHWC',
    name=None
)
```

卷積-圖片影像特徵抽取

- tf.nn.conv2d(x,W,strides=[1,1,1,1],padding='same')
- 卷積參數主要有四個：
- 第一個參數為影像輸入:需要做卷積的輸入圖像,它是一個張量,[batch圖片數量,in_height圖片高度,in_width圖片寬度,in_channels圖片通道數]的維度,資料型態是float32或float64
- 第二個參數影像過濾器:為卷積核心,一個張量[filter_heitht影像過濾器高,filter_width影像過濾器寬,in_channels特徵圖像通道數,out_channels輸出通道]
- 第三個參數strides步寬:卷積圖像每一維移動的步寬,這是一個一維向量,長度4
- 第四個參數padding填充:預設是'SAME'或'VALID'
- 回傳一個張量,影像特徵圖feature map

池化縮減矩陣尺寸
降低計算複雜度

- tf.nn.max_pool
- max_pool(

```
    value,
    ksize,
    strides,
    padding,
    data_format='NHWC',
    name=None
)
```

池化縮減矩陣尺寸
降低計算複雜度

- max pooling是CNN的最大值池化
- 參數有四個：
- 第一個參數：特徵影像池化的輸入，池化層接在卷積層後面，所以輸入是影像特徵圖，是[batch, height, width, channels]的維度
- 第二個參數：池化矩陣的大小長height寬width，一個四維向量，[1, height, width, 1]，第0個和第三個參數預設為1
- 第三個參數：矩陣在每一個維度上移動的距離，[1, stride,stride, 1]
- 第四個參数padding：可以取'VALID' 或者'SAME'
- 回傳一個張量，shape仍然是[batch, height, width, channels]

3.深度學習函數deepnn(x)
x:輸入784維度手寫數字,回傳y是張量維度10

```python
def deepnn(x):

    #x:輸入784維度手寫數字
    #回傳y是張量維度10
  with tf.name_scope('reshape'):
    x_image = tf.reshape(x, [-1, 28, 28, 1])

  # 第一卷積層 - maps one grayscale image to 32 feature maps.
  with tf.name_scope('conv1'):
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

  # 池化層 - downsamples by 2X.
  with tf.name_scope('pool1'):
    h_pool1 = max_pool_2x2(h_conv1)
```

4.第二卷積層對應 32輸入特徵圖到輸出64特徵圖到第二池化層

```python
# 第二卷積層maps 32 feature maps to 64.
with tf.name_scope('conv2'):
  W_conv2 = weight_variable([5, 5, 32, 64])
  b_conv2 = bias_variable([64])
  h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)

# 第二池化層pooling layer.
with tf.name_scope('pool2'):
  h_pool2 = max_pool_2x2(h_conv2)
```

全連接層,在兩次池化層矩陣28*28縮小維度為7*7*64(過濾器),對應到1024個神經元

```
  #全連接層,在兩次池化層矩陣28*28縮小維度為7*7*64(過濾器),對應到1024個特徵
with tf.name_scope('fc1'):
  W_fc1 = weight_variable([7 * 7 * 64, 1024])
  b_fc1 = bias_variable([1024])

  h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
  h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

with tf.name_scope('dropout'):
  keep_prob = tf.placeholder(tf.float32)
  h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

使用Dropout來減少Overffiting

對應1024個神經元到10個類別

```
  #對應1024個特徵到10個類別
with tf.name_scope('fc2'):
  W_fc2 = weight_variable([1024, 10])
  b_fc2 = bias_variable([10])

  y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
return y_conv, keep_prob
```

儲存訓練準確值的串列

- loglist=[]
- loglist.append(train_accuracy)

建立卷積深度學習的網路

```
def main(_):
  # 輸入資料
  mnist = input_data.read_data_sets("MNIST_data/",one_hot=True)

  x = tf.placeholder(tf.float32, [None, 784])

  y_ = tf.placeholder(tf.float32, [None, 10])

  # 建立深度學習的網路
  y_conv, keep_prob = deepnn(x)
```

5.交叉熵最佳化


```
with tf.name_scope('loss'):
  cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                        logits=y_conv)

cross_entropy = tf.reduce_mean(cross_entropy)

with tf.name_scope('adam_optimizer'):
  train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

tf.cast(,tf.float32)將布林值轉成float32浮點數

```
with tf.name_scope('accuracy'):
  correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
  correct_prediction = tf.cast(correct_prediction, tf.float32)
accuracy = tf.reduce_mean(correct_prediction)
```

將計算圖輸出到 log/cnn99_log目錄

```
train_writer = tf.summary.FileWriter("log/cnn99_log",
                                     tf.get_default_graph())
train_writer.close()
```

6.平行計算會議

- train_step.run()將使用AdamOptimizer()來更新參數,使用feed_dict來取代placeholder的張量x和y_

```
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(5000):
    batch = mnist.train.next_batch(100)
    if i % 100 == 0:
      train_accuracy = accuracy.eval(feed_dict={
          x: batch[0], y_: batch[1], keep_prob: 0.8})
      loglist.append(train_accuracy)
      print('step %d, training accuracy %g' % (i, train_accuracy))
    train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.8})
```

測試結果預測

```
print('test accuracy %g' % accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 0.8}))
```

執行main(1),載入MNIST資料

```
main(1)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

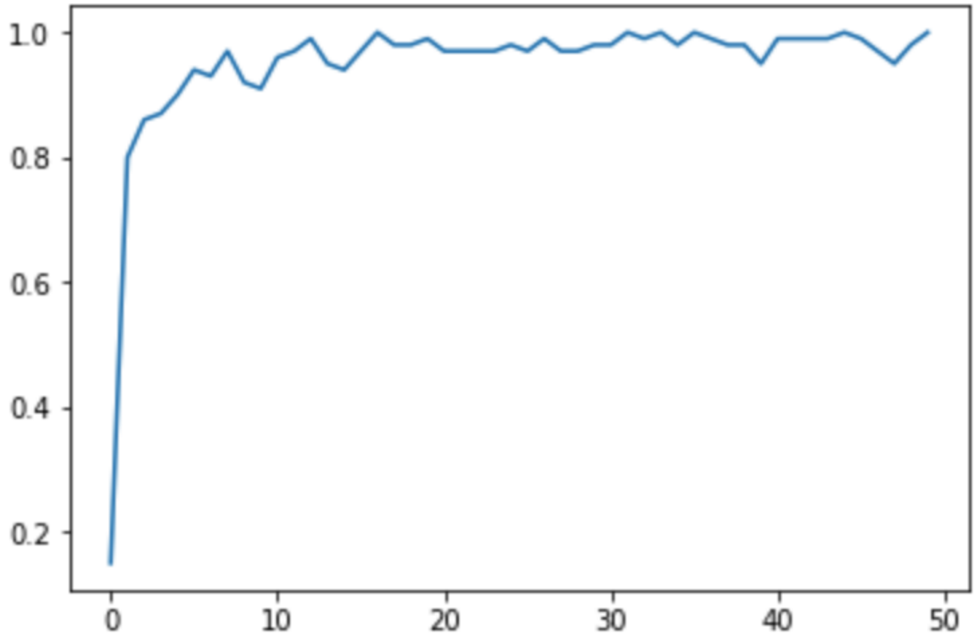
TensorFlow手寫辨識測試準確度

98.59%

```
step 3500, training accuracy 1
step 3600, training accuracy 0.99
step 3700, training accuracy 0.98
step 3800, training accuracy 0.98
step 3900, training accuracy 0.95
step 4000, training accuracy 0.99
step 4100, training accuracy 0.99
step 4200, training accuracy 0.99
step 4300, training accuracy 0.99
step 4400, training accuracy 1
step 4500, training accuracy 0.99
step 4600, training accuracy 0.97
step 4700, training accuracy 0.95
step 4800, training accuracy 0.98
step 4900, training accuracy 1
test accuracy 0.9859
```

7.準確度98.59%

```
import matplotlib.pyplot as plt
plt.plot(loglist)
plt.show()
```



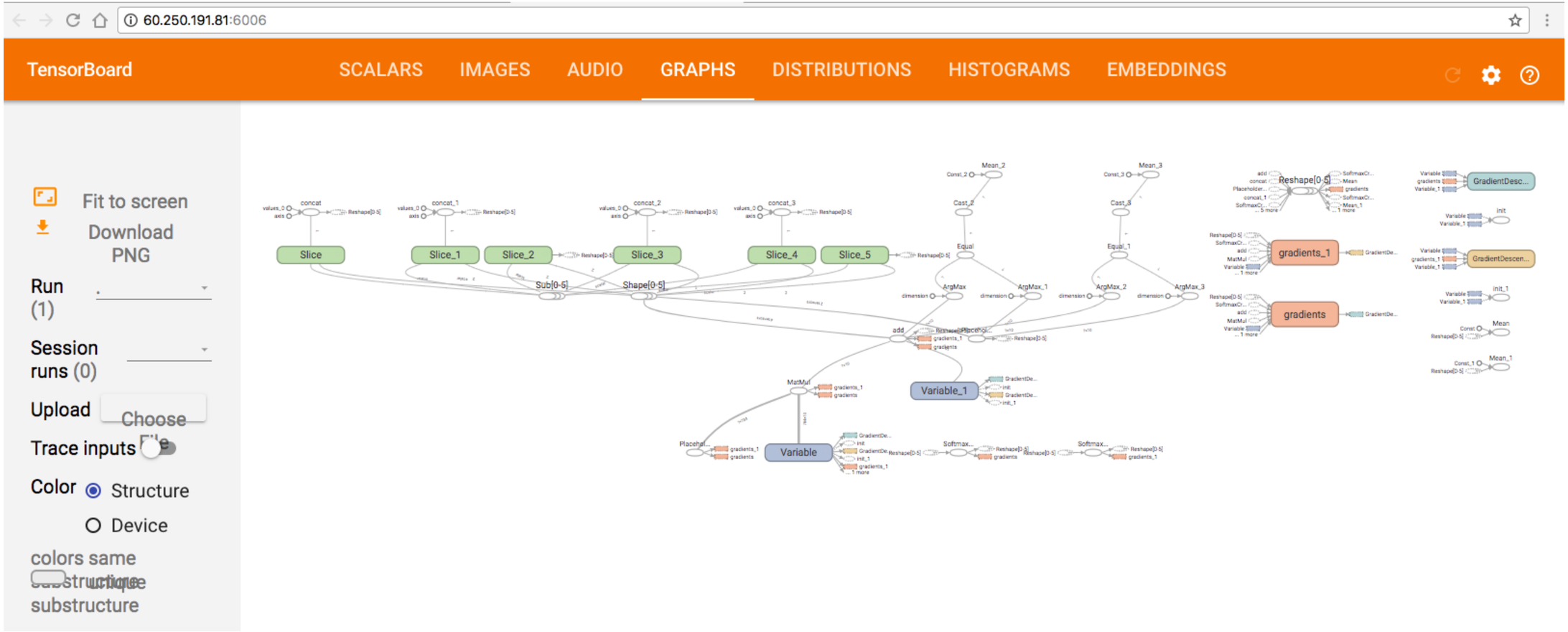
8.TensorBoard的計算圖Graphs

- 打開Tensorboard

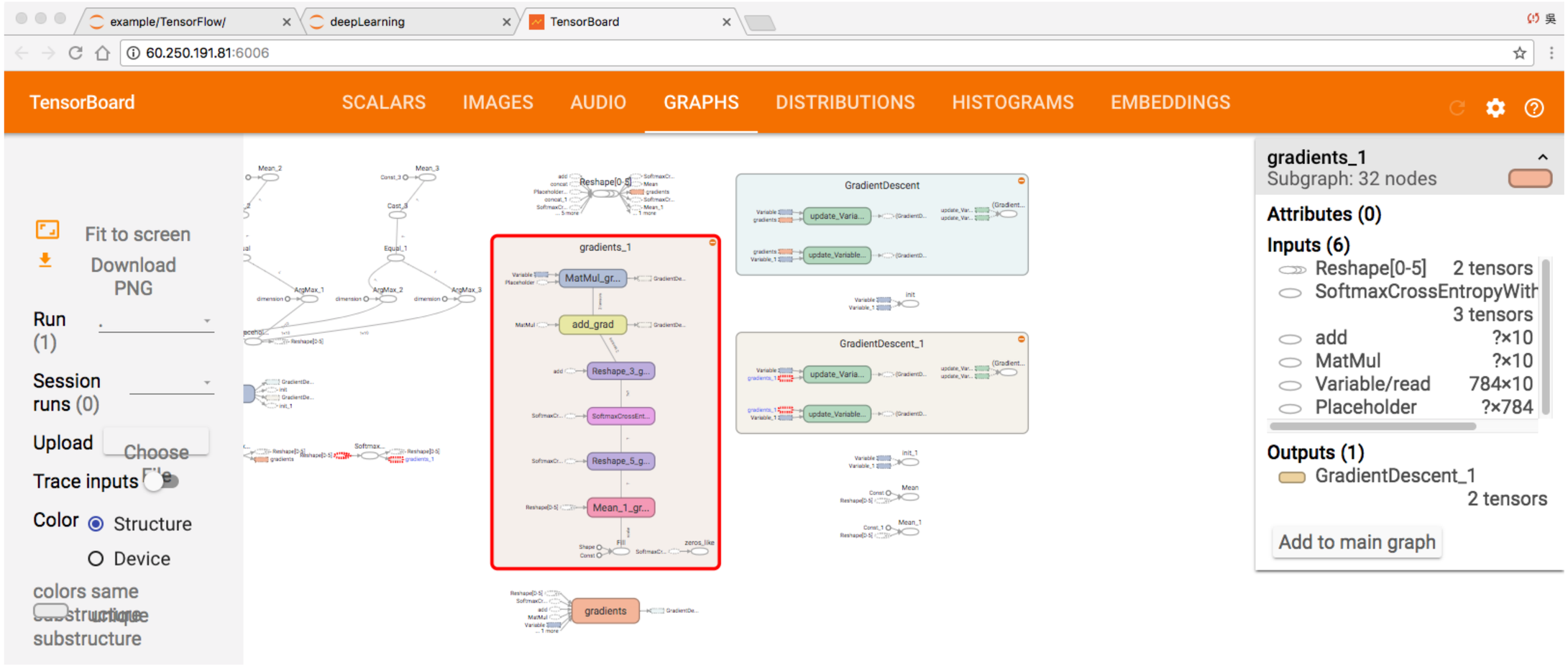
```
$ tensorboard –logdir=./log/simple99_log
```

```
Starting TensorBoard b'41' on port 6006
(You can navigate to http://IP:6006)
```

TensorBoard的計算圖Graphs



選取計算圖Graphs



- Thanks.

Python
類神經網路深度學習

吳佳諺 老師

Python
類神經網路深度學習

1.安裝Tensorflow
2.安裝Keras
3.類神經網路圖形辨識MNIST
4.類神經深度學習
5.繪製實際和預測結果的手寫辨識

1.安装Tensorflow

- pip install tensorflow

2.安装Keras

- pip install keras

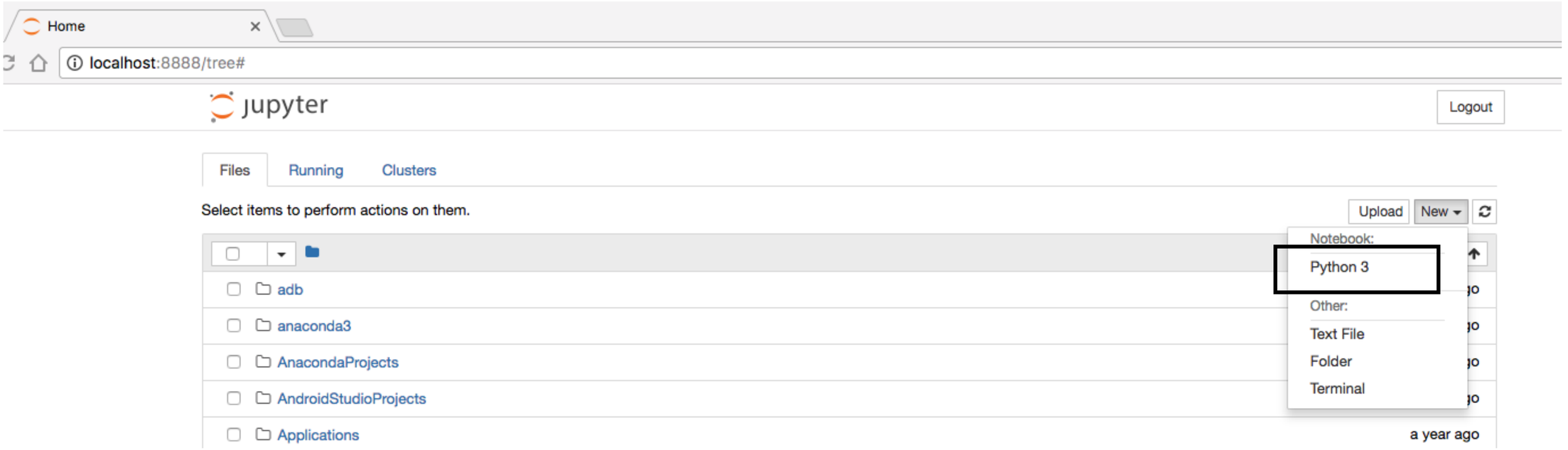
3.啟動jupyter

```
$ jupyter notebook
```

選取New


Home
localhost:8888/tree#
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Name
Last Modified
adb
a year ago
anaconda3
20 hours ago
AnacondaProjects
19 days ago
AndroidStudioProjects
a year ago
Applications
a year ago
clip_android_temp
a year ago
Desktop
a minute ago
Documents
10 days ago
Downloads
4 hours ago

選取Python 3


Home
localhost:8888/tree#
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Notebook:
Python 3
Other:
Text File
Folder
Terminal
adb
anaconda3
AnacondaProjects
AndroidStudioProjects
Applications
a year ago

觀看Tensorflow的版本

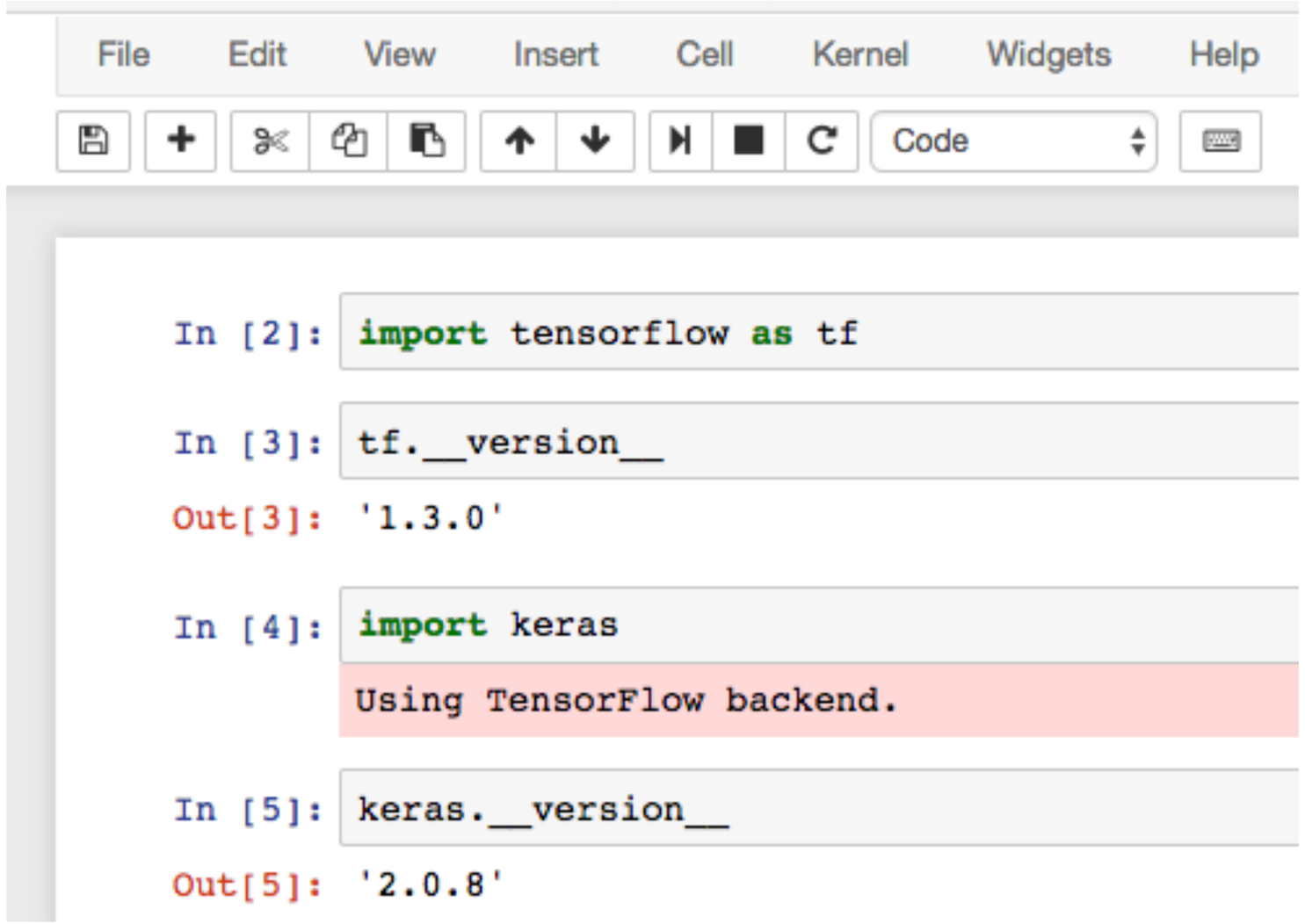
- import tensorflow as tf
- tf.__version__

輸入import tensorflow和執行

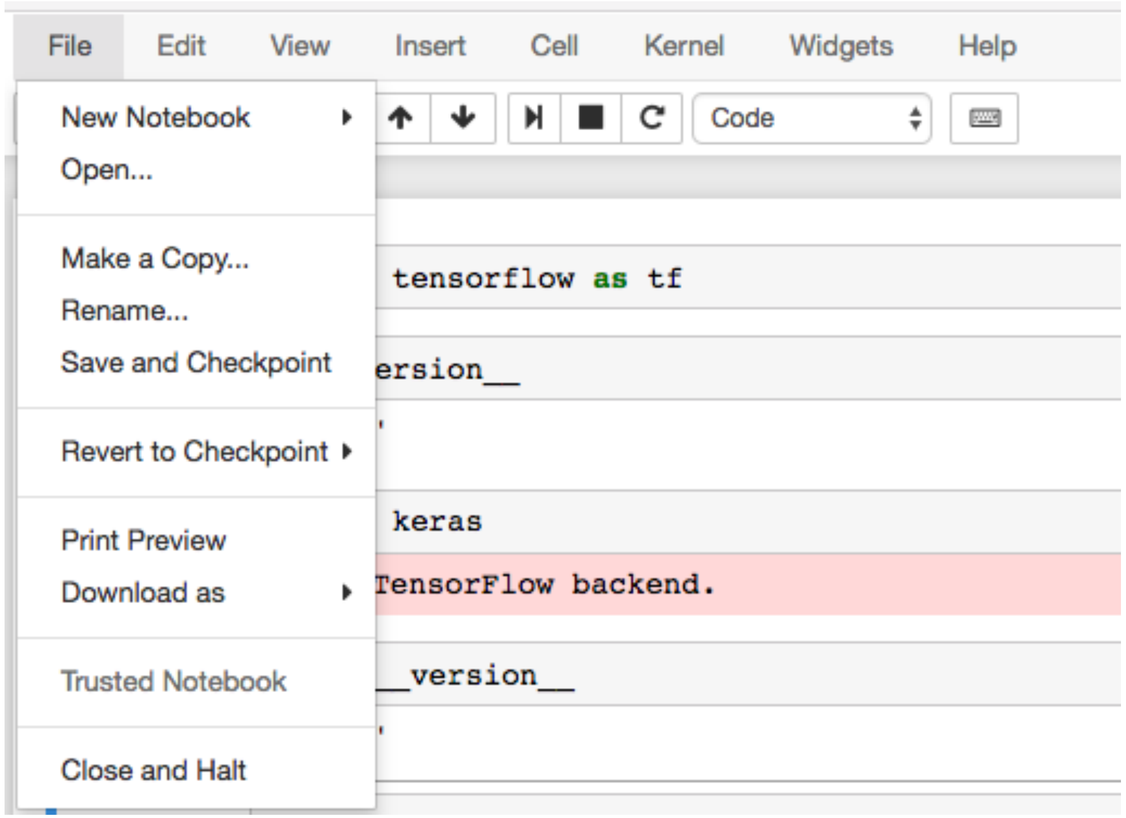
```
In [2]: import tensorflow as tf

In [3]: tf.__version__
Out[3]: '1.3.0'
```

顯示keras版本,使用TensorFlow當後端

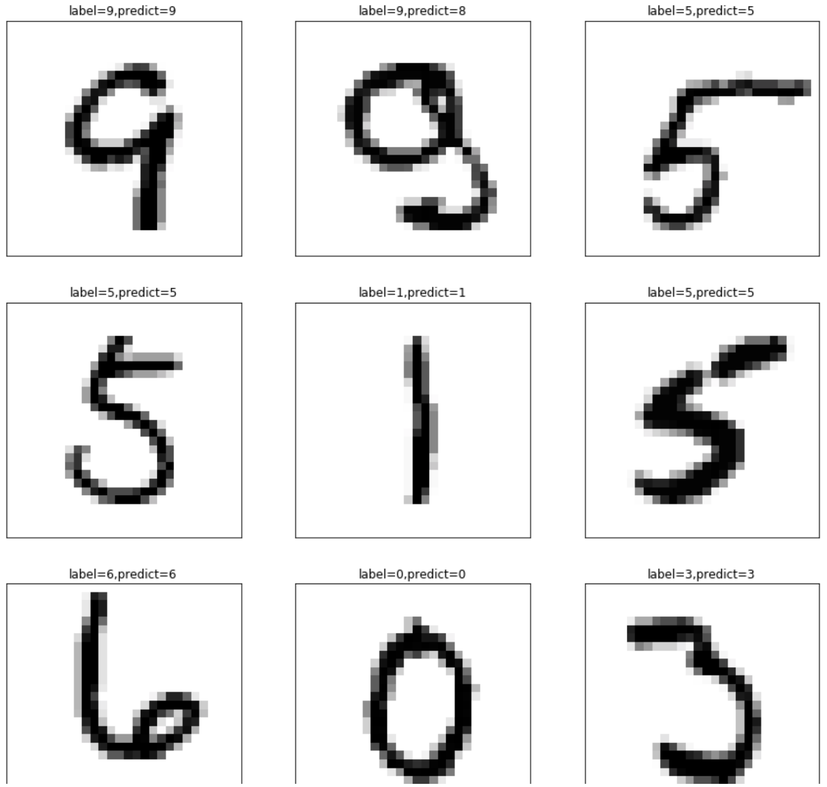


Save儲存檔案


File
Edit
View
Insert
Cell
Kernel
Widgets
Help
New Notebook
Open...
Make a Copy...
Rename...
Save and Checkpoint
Revert to Checkpoint
Print Preview
Download as
Trusted Notebook
Close and Halt
Code
tensorflow as tf
ersion__
keras
TensorFlow backend.
__version__

3.類神經網路圖形辨識MNIST

```
In [174]: plot_figures_labels(X_test,y_test,predicted_classes,idx=150,num=9)
```



輸入Keras模組

- 輸入keras的資料datasets,mnist手寫數字
- 輸入keras的模型models,sequential循序模型
- 輸入keras類神經核心的Dense,Dropout,和啟動Activation

In [42]:
```
import numpy as np
import matplotlib.pyplot as plt

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
```

載入MNIST資料

- http://yann.lecun.com/exdb/mnist/
- 為辨識手寫資料的網址
- mnist.load_data()為載入手寫資料

```
In [189]: nb_classes = 10
          (X_train, y_train), (X_test, y_test) = mnist.load_data()
          print("X_train original shape", X_train.shape)
          print("y_train original shape", y_train.shape)

          X_train original shape (60000, 28, 28)
          y_train original shape (60000,)
```

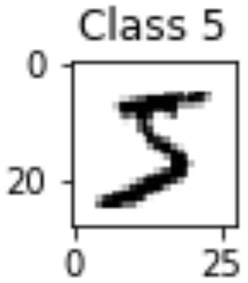
繪製圖形函數

- matplotlib.pyplot.gc()是得到目前的圖形
- matplotlib.pyplot.imshow是顯示圖片

In [190]:
```
import matplotlib.pyplot as plt
def plot_image(image,i):
    fig = plt.gcf()
    fig.set_size_inches(3, 3)
    plt.imshow(image, cmap='binary')
    plt.title("Class {}".format(y_train[i]))
    plt.show()
```

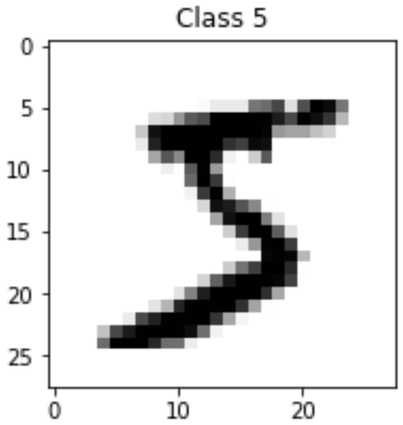
- 顯示手寫圖片第一張

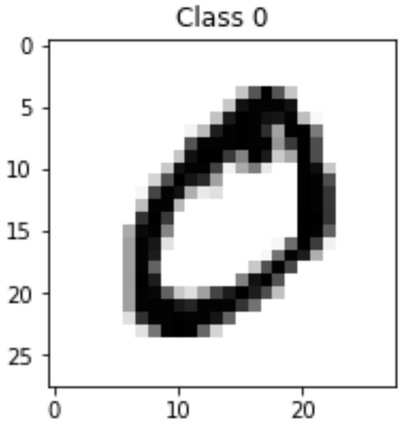
```
In [191]: plot_image(X_train[0],0)
```



觀看訓練的手寫資料

```
In [192]: for i in range(2):
              plot_image(X_train[i],i)
```





訓練,測試資料的正規化

- 訓練60000筆資料,784維度,28*28
- 測試10000筆資料,並將資料數值除以255,成為0-1的數值

In [7]:
```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print("訓練矩陣維度 shape", X_train.shape)
print("測試矩陣維度 shape", X_test.shape)
```

```
訓練矩陣維度 shape (60000, 784)
測試矩陣維度 shape (10000, 784)
```

utils.to_categorical將標籤轉換成0到9的陣列

- 建立10,nb_classes個元素的向量,在第y_train[0]元素值為1,其它向量元素為0.y_train[0]為0到9的數字.

```
In [80]: Y_train = np_utils.to_categorical(y_train, nb_classes)
         Y_test = np_utils.to_categorical(y_test, nb_classes)
```

```
In [82]: y_train[0]
Out[82]: 5
```

```
In [81]: Y_train[0]
Out[81]: array([ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.])
```

4.類神經深度學習

- Keras的核心為模型，最主要也是最常使用的是Sequential這個模型，Sequential可以讓我們按照順序將神經網路串起。深度學習為隱藏層有兩層或兩層以上.

```
In [60]: model = Sequential()
         model.add(Dense(512, input_shape=(784,)))
         model.add(Activation('relu'))
         model.add(Dropout(0.2))
         model.add(Dense(512))
         model.add(Activation('relu'))
         model.add(Dropout(0.2))
         model.add(Dense(units=500,activation='relu'))
         model.add(Dropout(0.2))
         model.add(Dense(10))
         model.add(Activation('softmax'))
```

類神經深度學習

- add()我們可以一層一層的將神經網路疊起。在每一層之中我們只需要設定每層的大小(units)與啟動函數(activation function)。
- 第一層輸入向量大小、最後一層為units要等於輸出的向量大小。
- 最後一層的啟動函數(activation function)為softmax。
- softmax()為歸一化指數函數,將向量的值歸一化為0到1之間。

keras編譯模型model.compile()

- 使用Adam做為優化器，成本函數使用categorical_crossentropy。

- model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

訓練和建立模型model.fit()

- 訓練為80%,用來驗證的設為20%,反覆8次執行,每次128筆

```
In [51]: train_data=model.fit(X_train, Y_train,
                   validation_split=0.2, epochs=8,
                   batch_size=128, verbose=2)

Train on 48000 samples, validate on 12000 samples
Epoch 1/8
 - 9s - loss: 0.2820 - acc: 0.9159 - val_loss: 0.1211 - val_acc: 0.9624
Epoch 2/8
 - 8s - loss: 0.1131 - acc: 0.9656 - val_loss: 0.0986 - val_acc: 0.9702
Epoch 3/8
 - 8s - loss: 0.0788 - acc: 0.9750 - val_loss: 0.0877 - val_acc: 0.9737
Epoch 4/8
 - 8s - loss: 0.0593 - acc: 0.9815 - val_loss: 0.0819 - val_acc: 0.9743
Epoch 5/8
 - 9s - loss: 0.0477 - acc: 0.9840 - val_loss: 0.0756 - val_acc: 0.9785
Epoch 6/8
 - 8s - loss: 0.0380 - acc: 0.9876 - val_loss: 0.0796 - val_acc: 0.9778
Epoch 7/8
 - 8s - loss: 0.0355 - acc: 0.9884 - val_loss: 0.0737 - val_acc: 0.9802
Epoch 8/8
 - 8s - loss: 0.0306 - acc: 0.9898 - val_loss: 0.0815 - val_acc: 0.9796
```

model.fit()傳回History物件

```
In [74]: train_data.history.keys()
Out[74]: dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

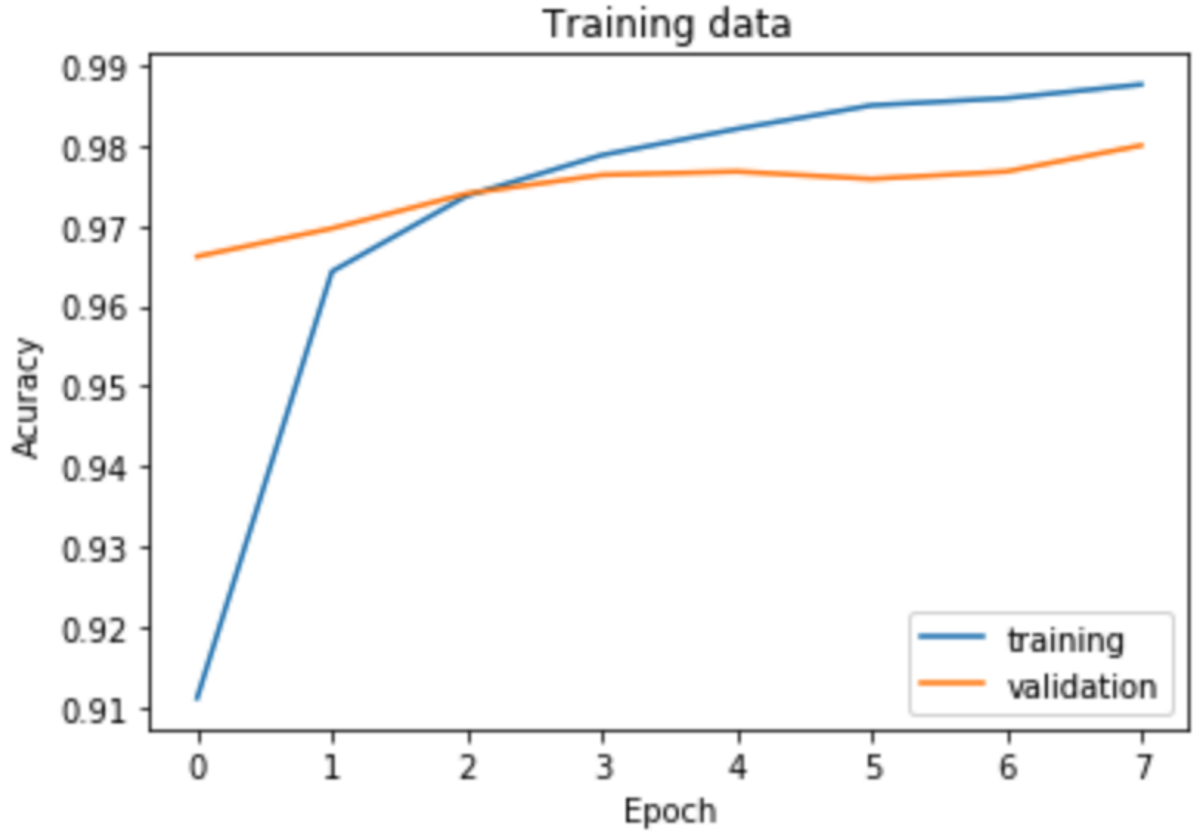
自訂繪製圖形

```
def show_TFigure(training,validation_data,training_data,loc):
    plt.xlabel('Epoch')
    plt.ylabel('Acuracy')
    plt.title('Training data')
    plt.plot(training_data.history[training])
    plt.plot(training_data.history[validation_data])

    if loc == 1 :
        plt.legend(['training','validation'],loc='lower right')
        plt.show()
    else:
        plt.legend(['training','validation'],loc='upper right')
        plt.show()
```

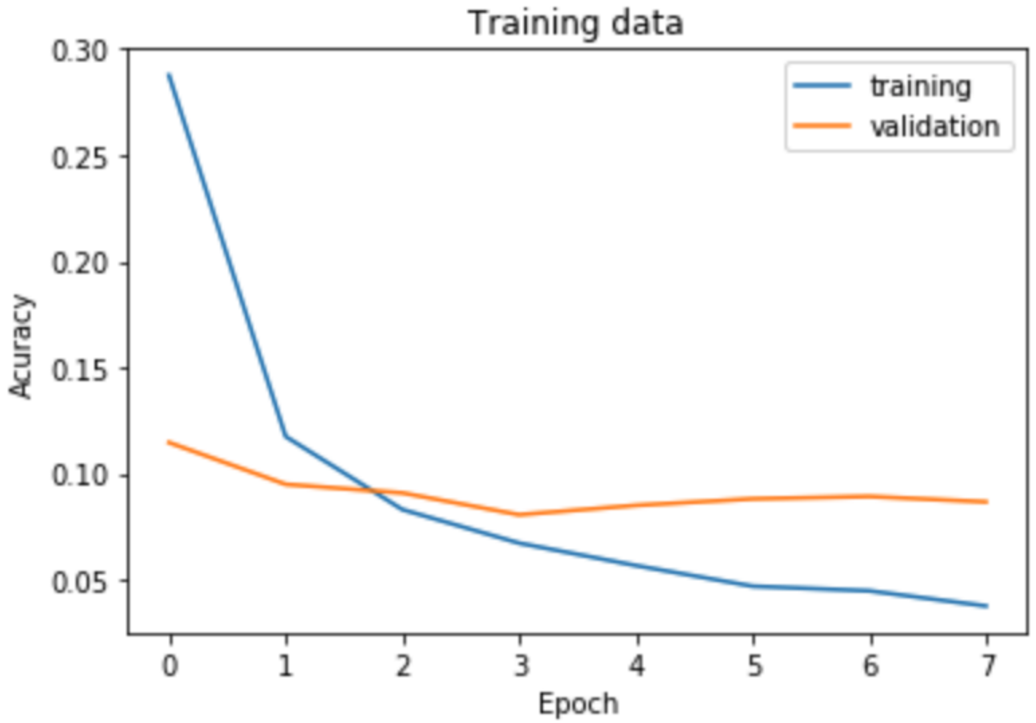
這是訓練和驗證的結果

```
show_TFigure('acc','val_acc',train_data,1)
```



這是訓練和驗證的成本函數

```
show_TFigure('loss','val_loss',train_data,2)
```



驗證模型準確性使用evaluate()

In [63]:
```
score = model.evaluate(X_test, Y_test,)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
   32/10000 [..............................] - ETA: 1s
  512/10000 [>.............................] - ETA: 1s
  896/10000 [=>............................] - ETA: 1s
 1376/10000 [===>..........................] - ETA: 0s
 1888/10000 [====>.........................] - ETA: 0s
 2432/10000 [======>.......................] - ETA: 0s
 2976/10000 [=======>......................] - ETA: 0s
 3520/10000 [=========>....................] - ETA: 0s
 4064/10000 [==========>...................] - ETA: 0s
 4608/10000 [============>.................] - ETA: 0s
 5152/10000 [==============>...............] - ETA: 0s
 5696/10000 [================>.............] - ETA: 0s
 6208/10000 [=================>............] - ETA: 0s
 6752/10000 [===================>..........] - ETA: 0s
 7296/10000 [====================>.........] - ETA: 0s
 7808/10000 [======================>.......] - ETA: 0s
 8352/10000 [========================>.....] - ETA: 0s
 8864/10000 [=========================>....] - ETA: 0s
 9408/10000 [===========================>..] - ETA: 0s
 9920/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - ETA: 0s
Test score: 0.0790858687627
Test accuracy: 0.9781
```

model.predict_classes得到驗證的結果

- correct_indices為預測正確的索引,incorrect_indices為預測錯誤的索引

- predicted_classes = model.predict_classes(X_test)

- correct_indices = np.nonzero(predicted_classes == y_test)[0]

- incorrect_indices = np.nonzero(predicted_classes != y_test)[0]

model.predict_classes得到驗證的結果,0-9的手寫數值

```
In [166]: predicted_classes

Out[166]: array([7, 2, 1, ..., 4, 5, 6])
```

5.繪製實際和預測結果的手寫辨識

```
def plot_Flabels(X_data,y_data,predict,numtuple):
    figure=plt.gcf()
    figure.set_size_inches(15,15)
    index,number=numtuple
    for j in range(0,number):
            ax=plt.subplot(3,3,j+1)
            ax.imshow(X_data[index].reshape(28,28),cmap='binary')
            name='label='+str(y_data[index])
            if len(predict)>0:
                name+=",predict="+str(predict[index])
            index+=1
            ax.set_xticks([])
            ax.set_yticks([])
            ax.set_title(name,fontsize=12)
    plt.show()
```

28*28的圖形,正規化成0-1的數值

```
In [11]: X_test[0]
```

```
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.06666667, 0.25882354, 0.05490196, 0.26274511,
        0.26274511, 0.26274511, 0.23137255, 0.08235294, 0.9254902 ,
        0.99607843, 0.41568628, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.32549021, 0.99215686, 0.81960785, 0.07058824,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.08627451, 0.9137255 ,
        1.        , 0.32549021, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.50588238, 0.99607843, 0.93333334, 0.17254902,
```

y_test為0到9的數值

```
In [173]: X_test,y_test

Out[173]: (array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
                  [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                  [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                  ...,
                  [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                  [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                  [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32),
           array([7, 2, 1, ..., 4, 5, 6], dtype=uint8))
```

這是辨識錯誤的圖形編號

```
In [176]: incorrect_indices

Out[176]: array([ 151,  247,  321,  445,  447,  448,  449,  450,  456,  582,  610,
                  619,  646,  659,  684,  691,  720,  883,  890,  938,  944, 1014,
                 1039, 1112, 1156, 1166, 1178, 1182, 1226, 1232, 1242, 1247, 1260,
                 1315, 1319, 1356, 1393, 1438, 1500, 1522, 1530, 1531, 1549, 1554,
                 1600, 1609, 1611, 1621, 1626, 1681, 1751, 1790, 1828, 1901, 1982,
                 1984, 2004, 2024, 2053, 2070, 2098, 2109, 2118, 2129, 2130, 2135,
                 2185, 2189, 2293, 2299, 2326, 2369, 2387, 2406, 2414, 2447, 2455,
                 2488, 2560, 2607, 2618, 2648, 2654, 2713, 2877, 2921, 2927, 2939,
                 2952, 2953, 2995, 3030, 3060, 3062, 3108, 3117, 3251, 3377, 3405,
                 3451, 3475, 3503, 3520, 3533, 3558, 3559, 3597, 3681, 3776, 3780,
                 3796, 3808, 3811, 3893, 3941, 3943, 3976, 3985, 3995, 4027, 4065,
                 4078, 4156, 4176, 4199, 4224, 4248, 4289, 4294, 4317, 4363, 4382,
                 4425, 4497, 4528, 4536, 4567, 4751, 4807, 4823, 4860, 4880, 4966,
                 5068, 5138, 5331, 5457, 5573, 5600, 5634, 5642, 5676, 5734, 5936,
                 5945, 5955, 5973, 6009, 6011, 6023, 6045, 6046, 6071, 6093, 6166,
                 6173, 6426, 6555, 6574, 6576, 6597, 6641, 6651, 6735, 6744, 6783,
                 6817, 7049, 7216, 7459, 8059, 8091, 8094, 8246, 8273, 8277, 8290,
                 8325, 8413, 8504, 8522, 8527, 9009, 9024, 9211, 9253, 9280, 9587,
                 9634, 9664, 9669, 9692, 9700, 9729, 9735, 9745, 9749, 9755, 9768,
                 9770, 9779, 9792, 9832, 9839, 9867, 9904, 9922, 9944, 9982])
```

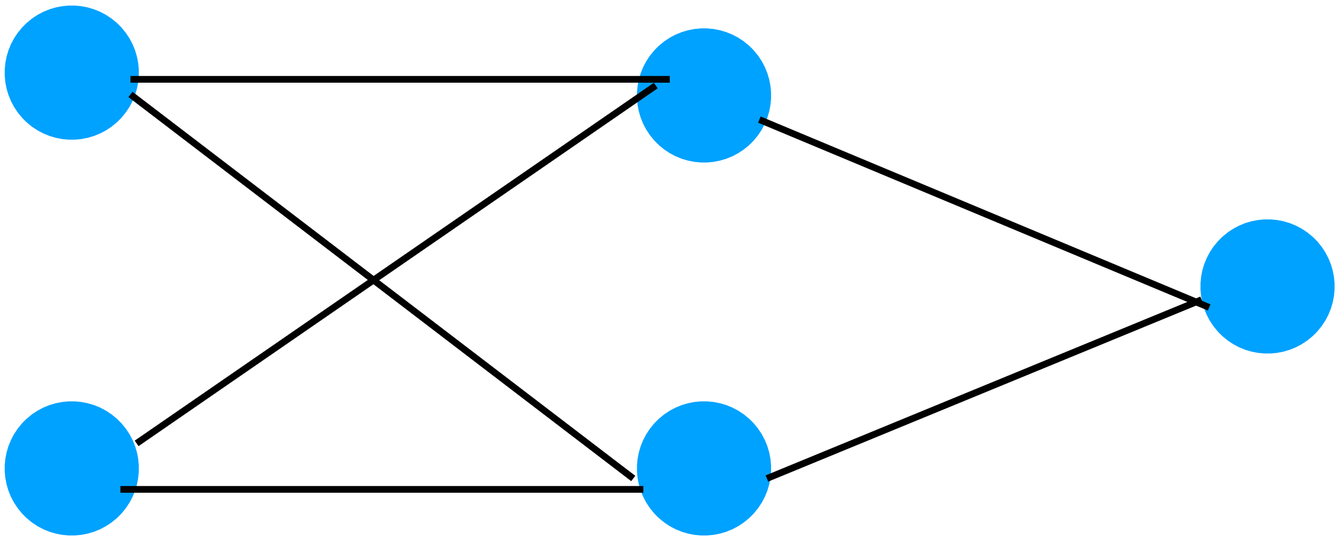
- Thanks.

Python深度學習
類神經網路

吳佳諺 老師

1.類神經網路
2.微分chain rule
3.類神經網路深度學習
4.啟動函數
5.CNN,卷積類神經網路
6.RNN遞迴式神經網路

1.類神經網路

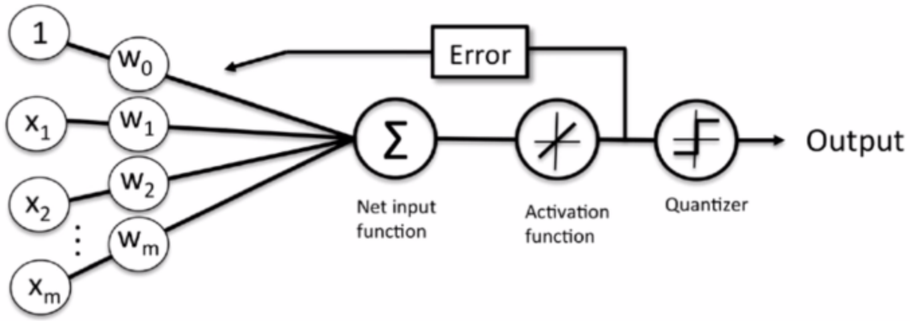


輸入　　隱藏層　　輸出

類神經網路

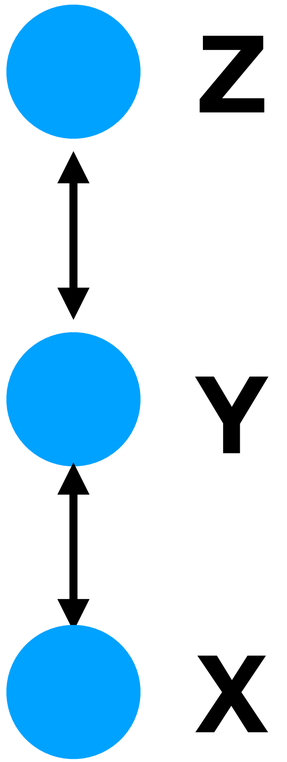
- 兩個輸入
- 兩個隱藏節點
- 一個輸出

類神經元



$$z_i = \sum w_{ij} x_i$$

2.微分chain rule



$$\Delta z = \frac{\partial z}{\partial y}\Delta y$$

$$\Delta y = \frac{\partial y}{\partial x}\Delta x$$

$$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

- 我們由 Δx 的改變來得到 Δy

$$\Delta y = \frac{\partial y}{\partial x} \Delta x$$

- 我們由 Δy 的改變來得到 Δz

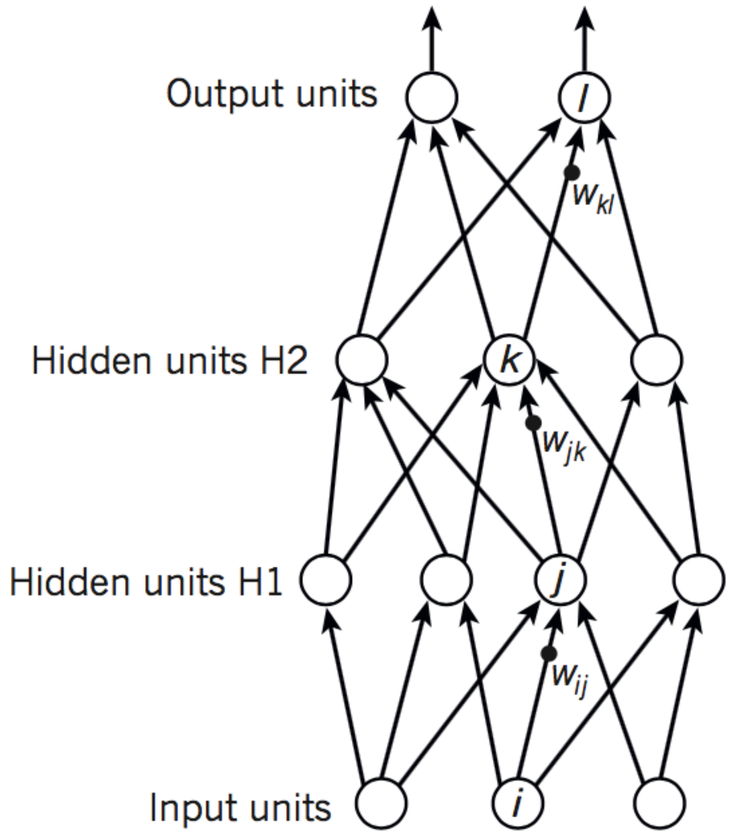
$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

微分ChainRule

- 我們由 Δx 的改變來得到 Δz

$$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$$

3.類神經網路深度學習



$$y_l = f(z_l)$$

$$z_l = \sum_{k \,\varepsilon\, \text{H2}} w_{kl}\, y_k$$

$$y_k = f(z_k)$$

$$z_k = \sum_{j \,\varepsilon\, \text{H1}} w_{jk}\, y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \,\varepsilon\, \text{Input}} w_{ij}\, x_i$$

類神經網路深度學習

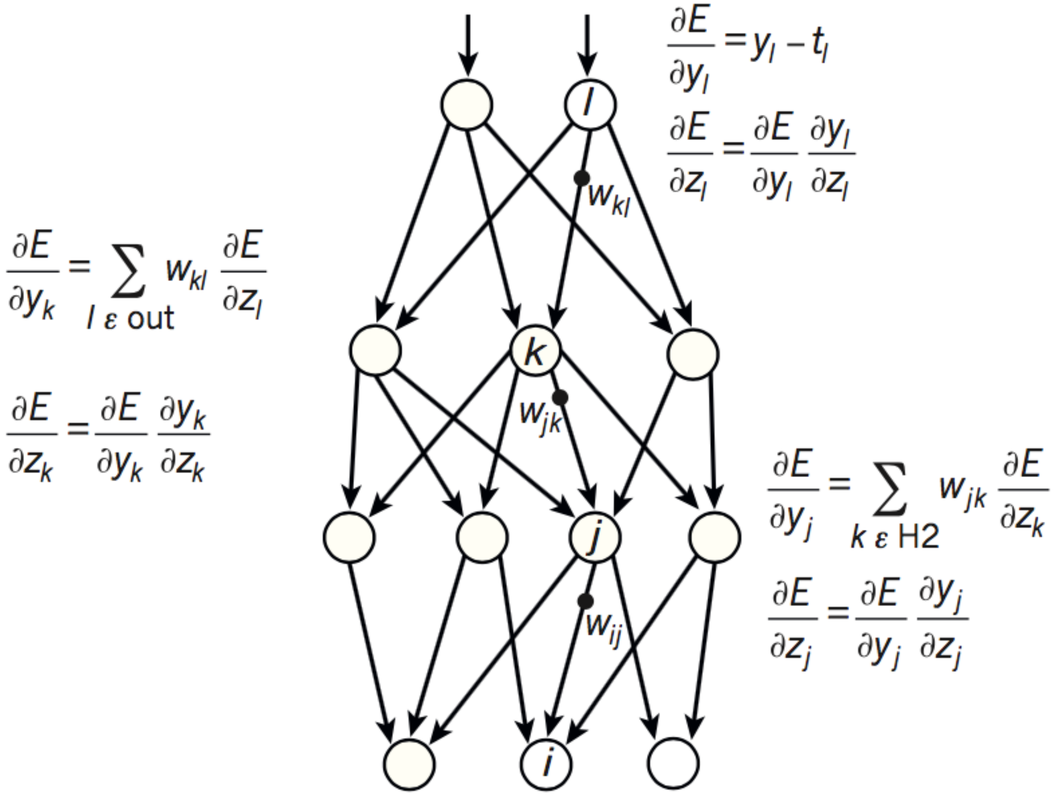
- Forward Pass向前傳遞
- 一層輸入層
- 兩層隱藏層
- 一層輸出層
- 每一個隱藏層的節點可以被逆推導微分梯度gradient

類神經網路深度學習

- 計算每一個節點的輸入z,節點前面的每一個節點加權的和
- 使用非線性的rectified調整函數relu計算z,f(z)=max(0,z)

逆傳遞Back Propagation

目標和預測的差值為修正值



逆傳遞更新權重W與偏差b

- 逆傳遞Back propagation
- 對於每一個隱藏層我們計算成本Error的微分梯度
- 對於節點l,成本Cost函數最小化

$$Cost = \frac{(y_l - t_l)^2}{2}$$

- t_l 是目標值
- 更新權重W與偏差b

4.啟動函數

- 線性
- sigmoid啟動函數
- relu啟動函數

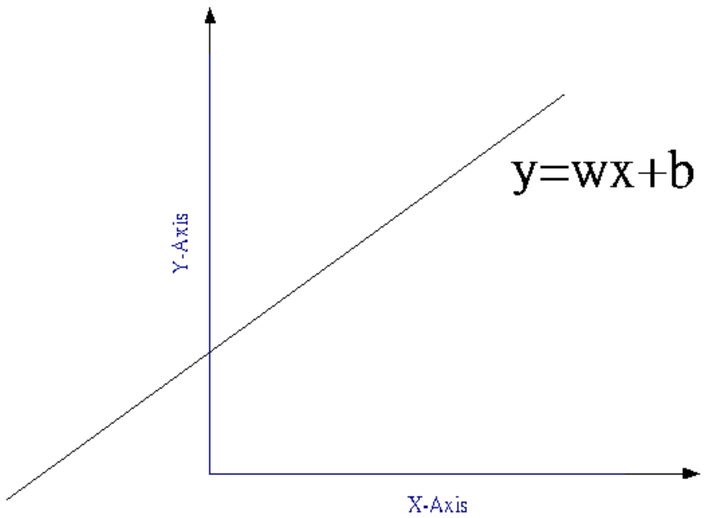
$$y=wx+b$$

$$\sigma(z)=\frac{1}{1+e^{-z}}$$

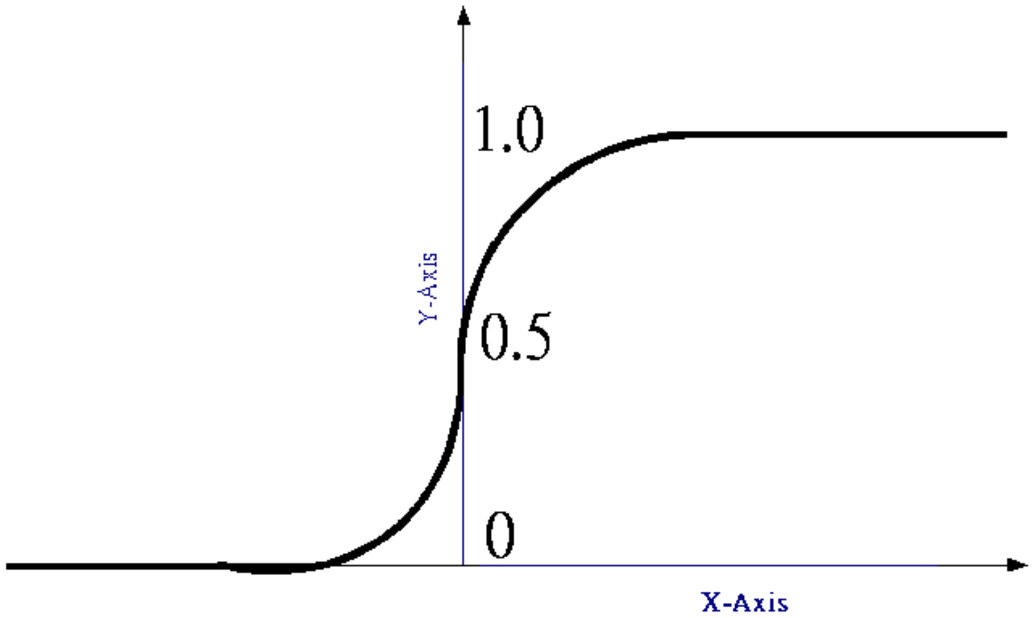
$$f(z)=max(0,z)$$

$$\mathrm{F}(z)=ln(1+e^{z})$$

線性

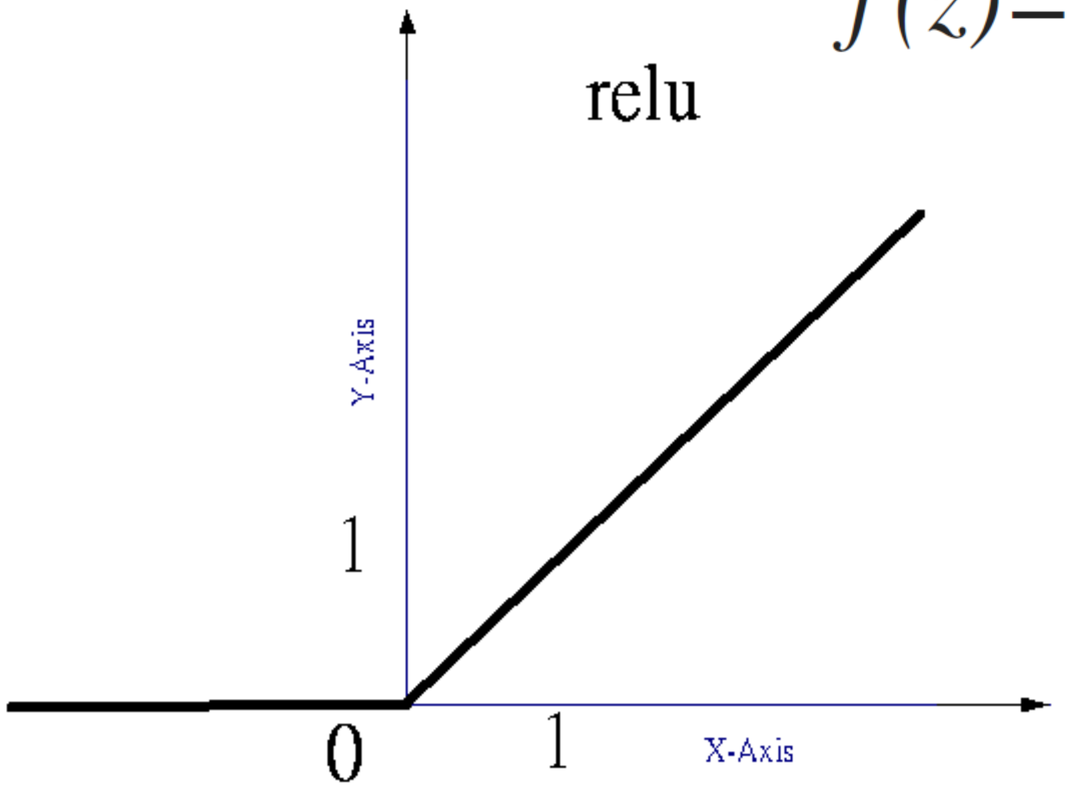


sigmoid啟動函數

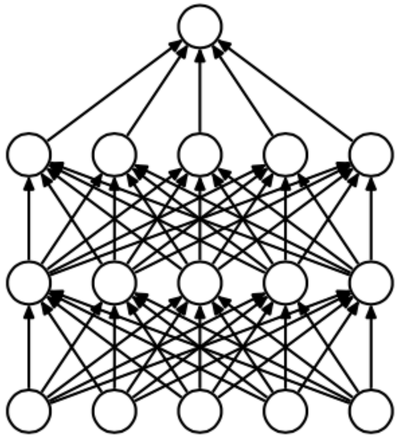


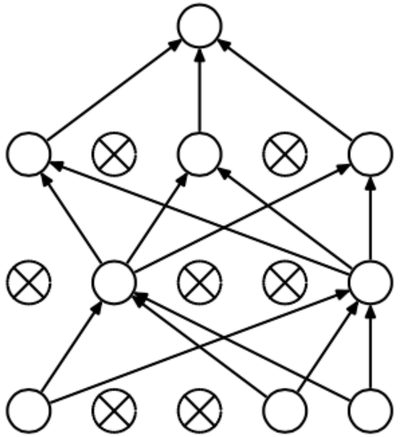
relu啟動函數

$$f(z)=max(0,z)$$

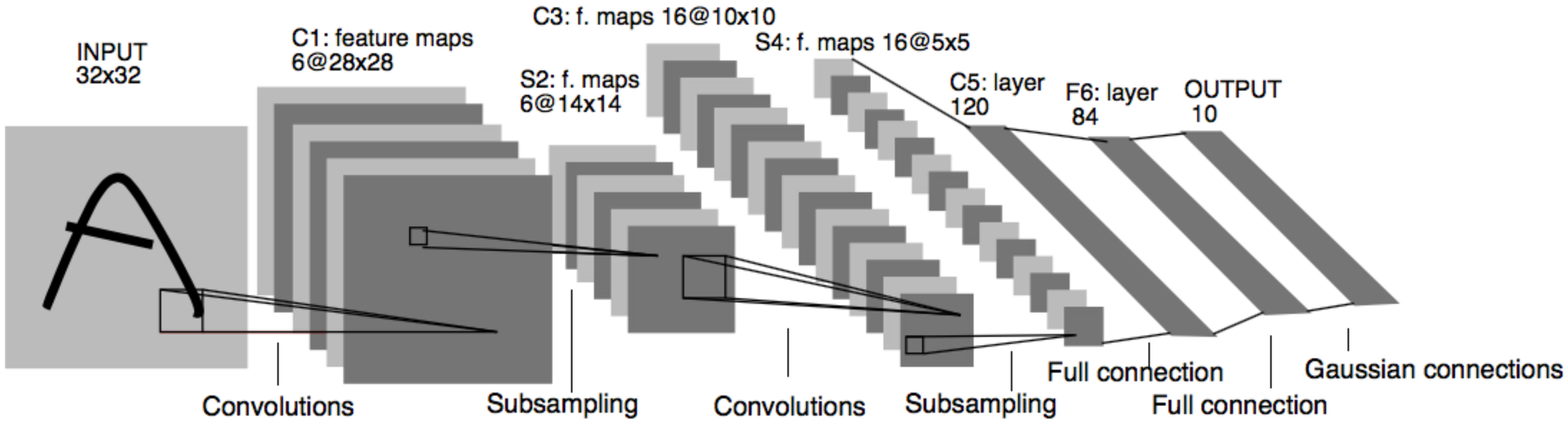


Dropout修剪神經元來減少Overfitting

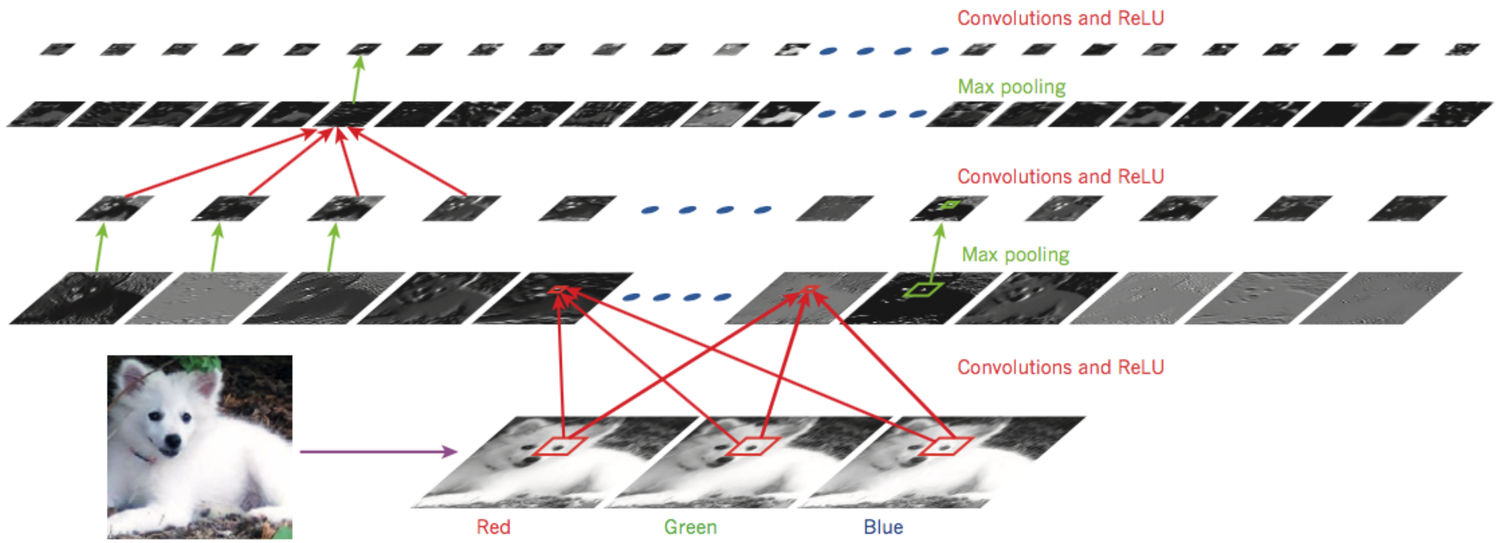




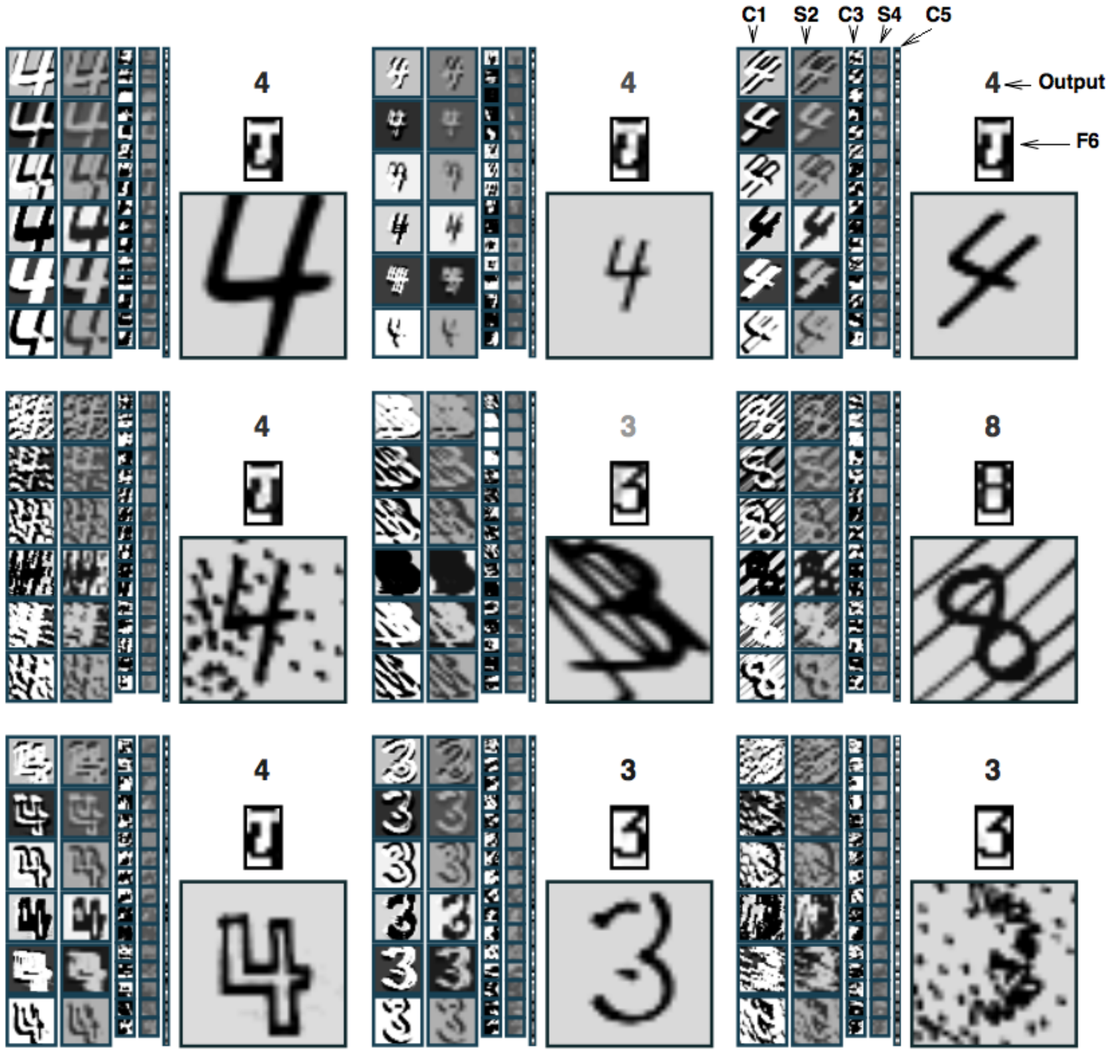
5.CNN,卷積類神經網路



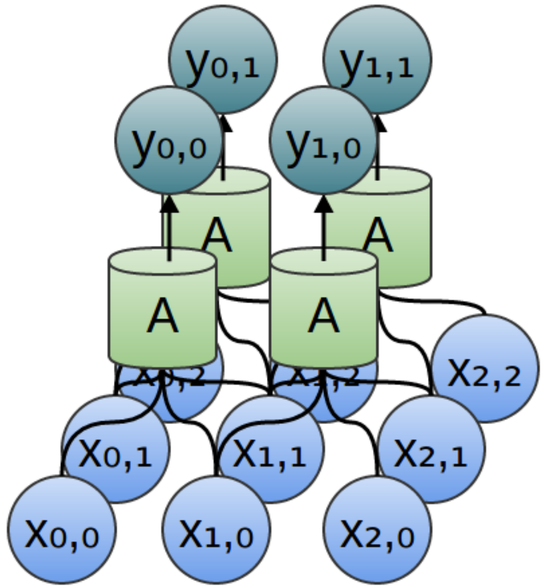
CNN,Convolutions Neural network



CNN,特徴取様

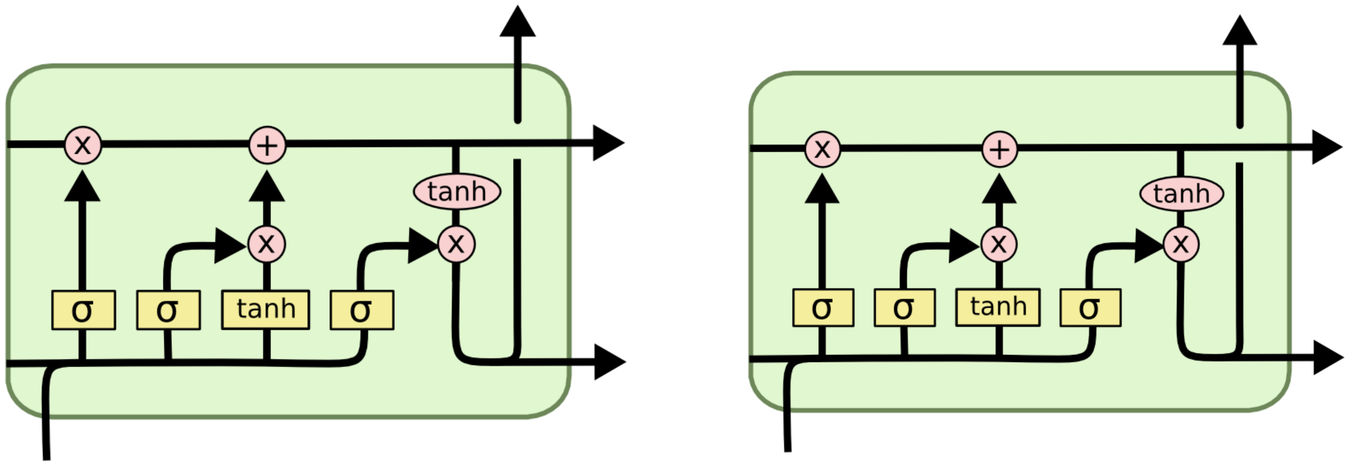


CNN,卷積類神經網路



6.RNN,遞迴神經網路Recurrent Neural Network(LSTM長短記憶)

輸出



輸入

RNN,遞迴神經網路Recurrent Neural Network(LSTM長短記憶)

- 運用在下列的地方:
- 序列樣本(Sequential patterns)
- 語言翻譯
- 語音辨識
- 聲音
- 影片滿意度調查
- 和時間有關的Pattern

- Thanks.

TensorFlow+Keras
CNN卷積深度學習影像辨識

吳佳諺 老師

TensorFlow+Keras
CNN卷積深度學習Cifar圖形辨識

1.安裝Tensorflow
2.Cifar-10圖片集
3.範例:cifar10_data_ok
4.預測測試10000筆的準確度73%
5.範例:cifar10_kk

1.安装Tensorflow

- pip install tensorflow

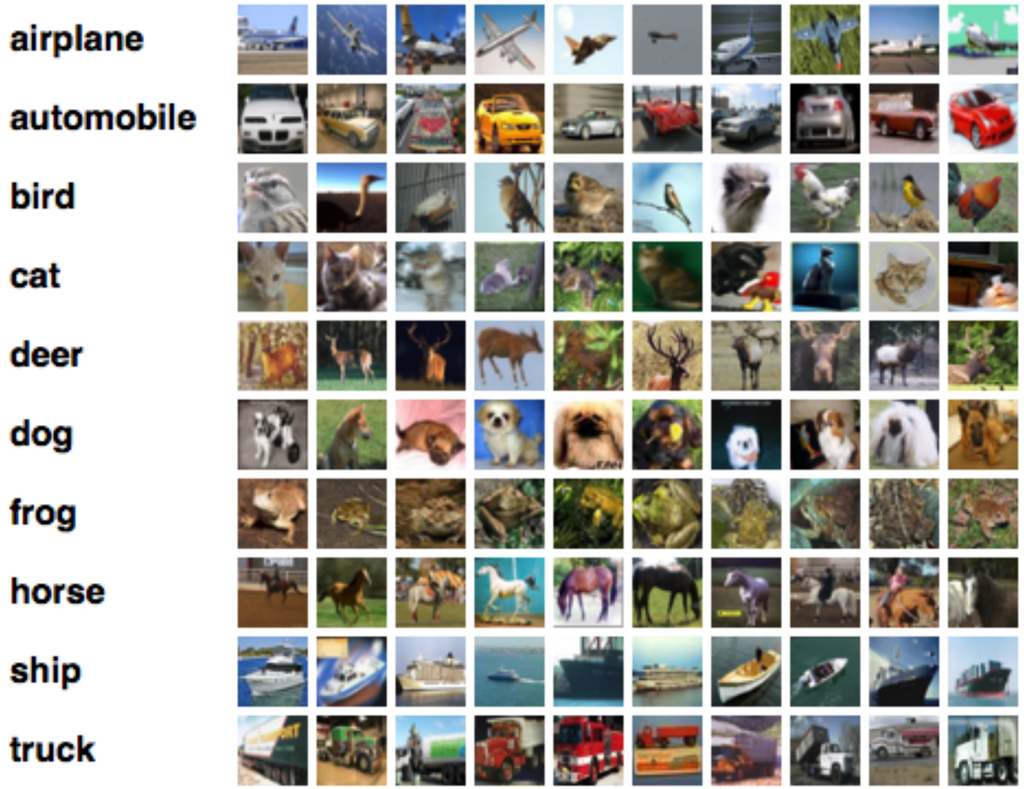
安装Keras

- pip install keras

2.Cifar-10圖片集

- https://www.cs.toronto.edu/~kriz/cifar.html

Here are the classes in the dataset, as well as 10 random images from each:



3.範例:cifar-10_data_ok

- 輸入keras的資料datasets,cifar-10圖片集
- 輸入keras的模型models,sequential循序模型
- 輸入keras類神經核心的Dense,Dropout,和啟動Activation

```
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os
import matplotlib.pyplot as plt
```

cifar10.load_data()
載入訓練和測試資料

- cifar10.load_data()為載入手寫資料,50000張圖,每一張32*32像素,RGB三顏色

```
batch_size = 32
num_classes = 10
epochs = 50
num_predictions = 20

# The data, shuffled and split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
10000 test samples
```

image_dictionary為字典,
數字索引圖的名稱

```
print(x_train[0])
print(y_train[0])
image_dictionary={0:'airplane',1:'automobile',2:'bird',3:'cat',
       4:'deer',5:'dog',6:'frog',7:'horse',8:'ship',9:'truck'}
print(image_dictionary[float(y_train[0])])
```

```
[[[ 59  62  63]
  [ 43  46  45]
  [ 50  48  43]
  ...,
  [158 132 108]
  [152 125 102]
  [148 124 103]]
```

plt.imshow()顯示繪製圖形

- **matplotlib.pyplot.imshow是顯示圖片**

- X_train[1]第二張圖,cmap為顏色,binary

```
fig=plt.figure()
fig.set_size_inches(6,6)
for i in range(0,9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_train[i],cmap='binary')
    plt.title(image_dictionary[float(y_train[i])])
    plt.xticks([])
    plt.yticks([])

fig
```

圖片的顯示

數位影像，是二維圖像用有限數字數值像素的表示。

RGB(紅藍綠)圖像：圖像中每個像素紅藍綠顏色可以由0(黑)到255(白)的亮度值(Intensity)表示。0–255之間表示不同的紅藍綠顏色值。

各種不一樣的圖案有不一樣的特徵

觀看各種圖片

python

TensorFlow

K

python

TensorFlow

K

python

TensorFlow

K

將數字轉換成向量

```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

- 將數字6轉換成二進位的向量,num_classes為有多少類

- 建立10,nb_classes個元素的向量,在第y_train[0]元素值為1,其它向量元素為0.y_train[0]為6的數字.

```
[6]
[ 0.  0.  0.  0.  0.  0.  1.  0.  0.  0.]
```

3-1Keras的核心為模型

- Keras的核心為模型，最主要也是最常使用的是Sequential這個模型，Sequential可以讓我們按照順序將神經網路串起。卷積和池化各有四層。

卷積和池化各有四層


```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

add()我們可以一層一層的將神經網路疊起。在每一層之中我們只需要設定每層的大小(units)與啟動函數(activation function)。

- 第一層輸入向量大小、最後一層為units要等於輸出的向量大小。
- 最後一層的啟動函數(activation function)為softmax。
- softmax()為歸一化指數函數,將向量的值歸一化為0到1之間。

```
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

最佳化隨機梯度下降

- 使用隨機梯度下降做為優化器，成本函數使用categorical_crossentropy交叉熵。

- opt = keras.optimizers.SGD(lr=0.01, decay=1e-6,

  momentum=0.9, nesterov=True)

訓練模型,使用交叉熵


```
# 訓練模型
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

3-2建立模型model.fit()

history.history.keys()傳回History物件的字典
loss是成本,acc是準確度
model.fit()傳回History物件

- history=model.fit(x_train, y_train,

  batch_size=batch_size,

  epochs=epochs,

  validation_data=(x_test, y_test),

  shuffle=True)

Cifar-10,77.26%訓練準確度

- Train on 50000 samples, validate on 10000 samples

```
acc: 0.7728
49856/50000 [============================>.] - ETA: 0s - loss: 0.6607 -
acc: 0.7728
49888/50000 [============================>.] - ETA: 0s - loss: 0.6610 -
acc: 0.7728
49920/50000 [============================>.] - ETA: 0s - loss: 0.6611 -
acc: 0.7727
49952/50000 [============================>.] - ETA: 0s - loss: 0.6612 -
acc: 0.7727
49984/50000 [============================>.] - ETA: 0s - loss: 0.6613 -
acc: 0.7726
50000/50000 [==============================] - ETA: 0s - loss: 0.6613 -
acc: 0.7726 - val_loss: 0.7772 - val_acc: 0.7341
```

測試1000筆資料,準確度72.8%

•驗證模型準確性使用evaluate()

- scores = model.evaluate(x_test[:1000], y_test[:1000], verbose=1)

  print('Test loss:', scores[0])

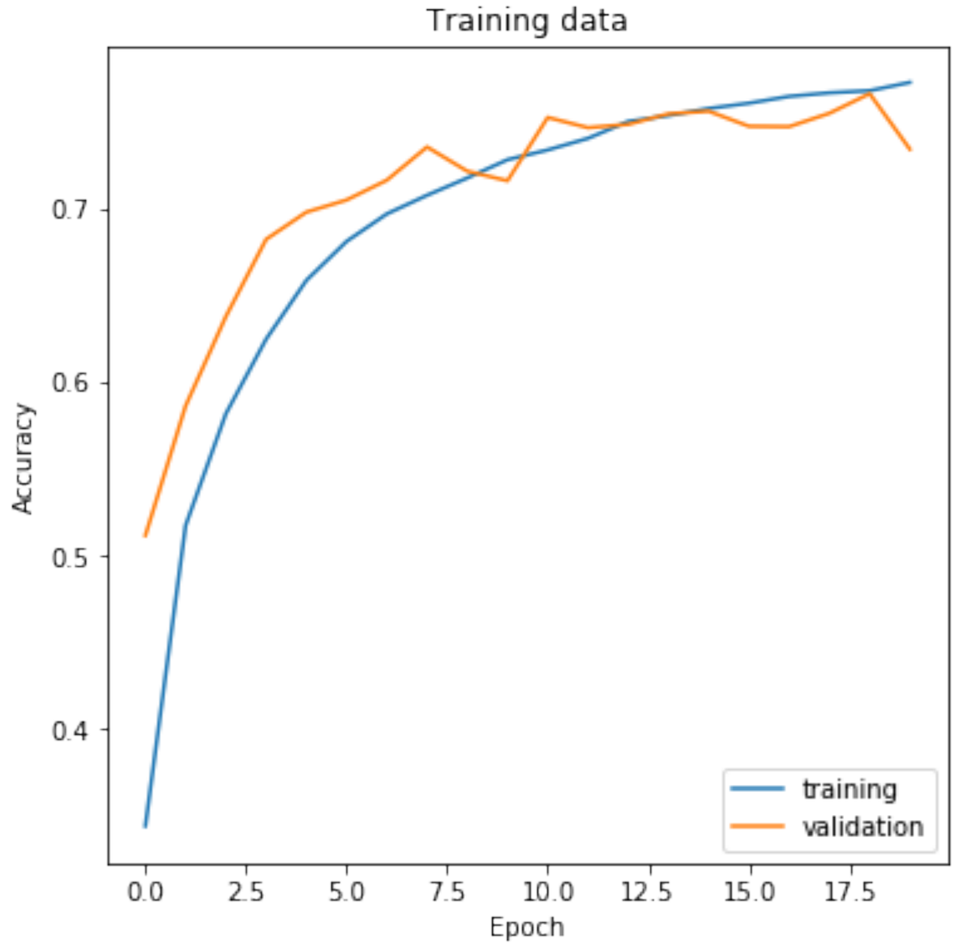
  print('Test accuracy:', scores[1])

```
Test loss: 0.801029728413
Test accuracy: 0.728
```

顯示訓練和驗證資料

- plt.clf()
- plt.xlabel('Epoch')
- plt.ylabel('Accuracy')
- plt.title('Training data')
- plt.plot(history.history['acc'])
- plt.plot(history.history['val_acc'])
- plt.legend(['training','validation'],loc='lower right')
- plt.show()
- plt.clf()

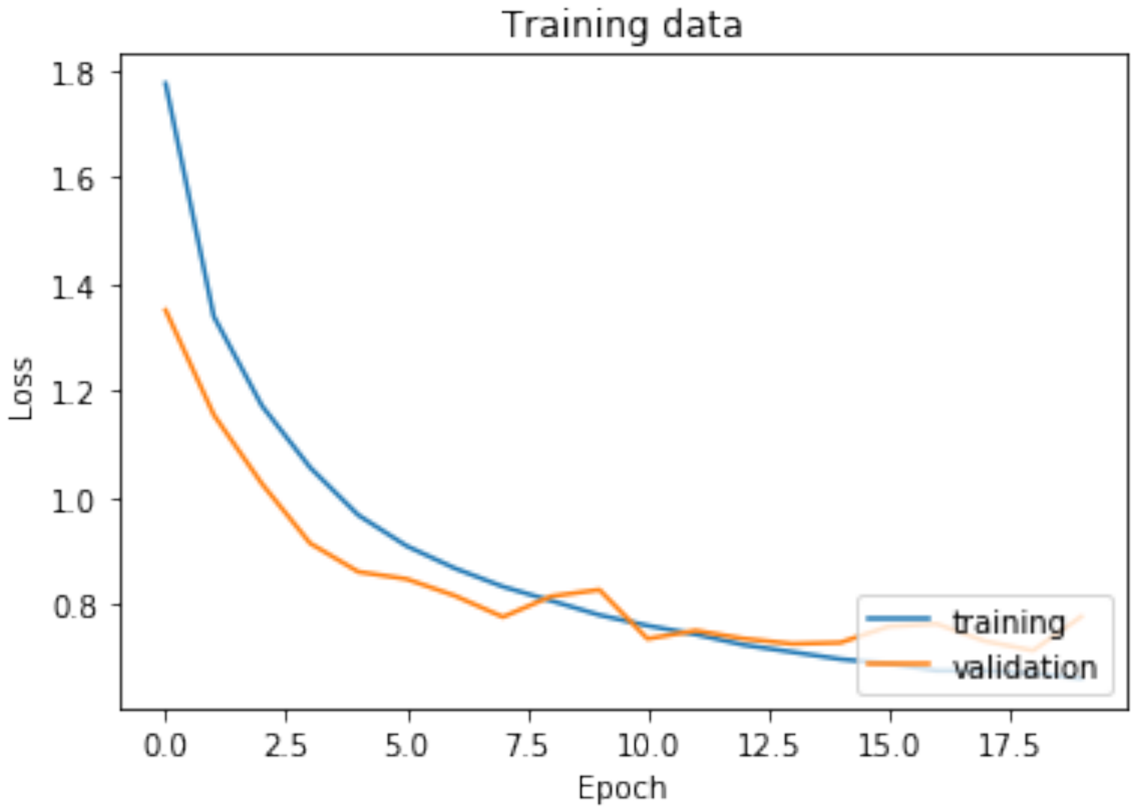
足夠多的Training data
抽取圖像特徵



顯示成本Loss下降

- plt.clf()
- plt.xlabel('Epoch')
- plt.ylabel('Loss')
- plt.title('Training data')
- plt.plot(history.history['loss'])
- plt.plot(history.history['val_loss'])
- plt.legend(['training','validation'],loc='lower right')
- plt.show()

足夠多的Training data
抽取圖像特徵



預測1000筆cifar-10測試資料

- model.predict_classes得到驗證的結果

- predicted_classes=model.predict_classes(x_test[:1000])

- 預測值

```
array([3, 8, 8, 8, 4, 6, 1, 6, 4, 9, 0,
9, 4, 7, 9, 6, 5, 5, 8, 6, 7, 0, 0,
        9, 4, 4, 4, 0, 9, 6, 6, 5, 4, 6,
9, 8, 4, 1, 9, 5, 4, 6, 7, 6, 0, 9,…..
```

matplotlib.pyplot.figure
將回傳圖形實體

rcParams定義預設數值

```
# 設定圖形長6寬6
plt.rcParams['figure.figsize'] = (6,6)

fig = plt.figure()
```

真實的圖和預測

```
import numpy as np
fig=plt.figure()
fig.set_size_inches(6,6)
for i in range(0,9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[i],cmap='binary')
    plt.title(image_dictionary[np.argmax(y_test[i])]+':'
              +image_dictionary[predicted_classes[i]])
    plt.xticks([])
    plt.yticks([])

fig
```

真實的圖和預測


cat:cat
ship:ship
ship:ship
airplane:ship
frog:deer
frog:frog
automobile:automobile
frog:frog
cat:deer

4.預測測試10000筆的準確度

73%

- scores2 = model.evaluate(x_test, y_test, verbose=1)
- print('Test loss:', scores2[0])
- print('Test accuracy:', scores2[1])

```
Test loss: 0.777192718601
Test accuracy: 0.7341
```

5.範例:cifar10_kk


```
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os
import matplotlib.pyplot as plt
```

用1000筆資料建立模型


```
history=model.fit(x_train[:1000], y_train[:1000],
                  batch_size=batch_size,
                  epochs=epochs,
                  validation_data=(x_test[:1000], y_test[:1000]),
                  shuffle=True)
```

訓練時間比較短


```
c: 0.9111
 864/1000 [========================>.....] - ETA: 0s - loss: 0.2405 - ac
c: 0.9086
 896/1000 [=========================>....] - ETA: 0s - loss: 0.2395 - ac
c: 0.9096
 928/1000 [==========================>...] - ETA: 0s - loss: 0.2327 - ac
c: 0.9127
 960/1000 [===========================>..] - ETA: 0s - loss: 0.2374 - ac
c: 0.9115
 992/1000 [============================>.] - ETA: 0s - loss: 0.2411 - ac
c: 0.9103
1000/1000 [==============================] - ETA: 0s - loss: 0.2405 - ac
c: 0.9100 - val_loss: 3.1695 - val_acc: 0.3720
```

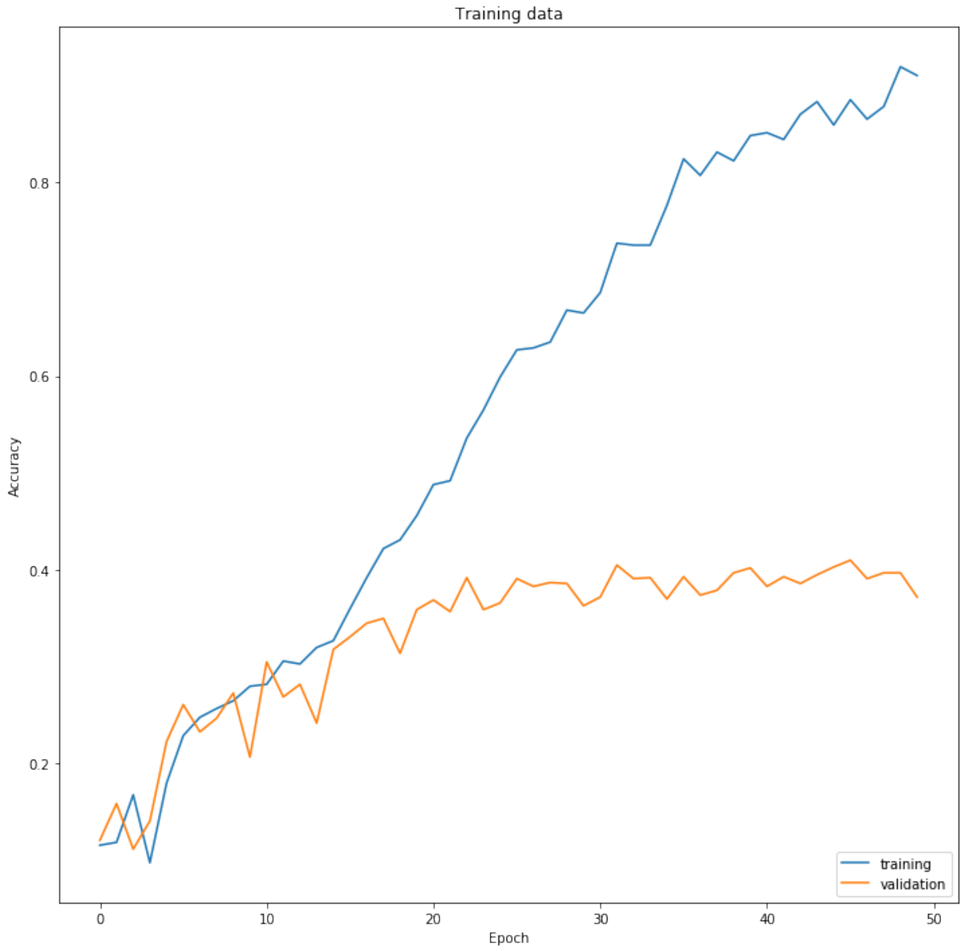
1000筆測試資料


```
# 測試資料.
scores = model.evaluate(x_test[:1000], y_test[:1000], verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

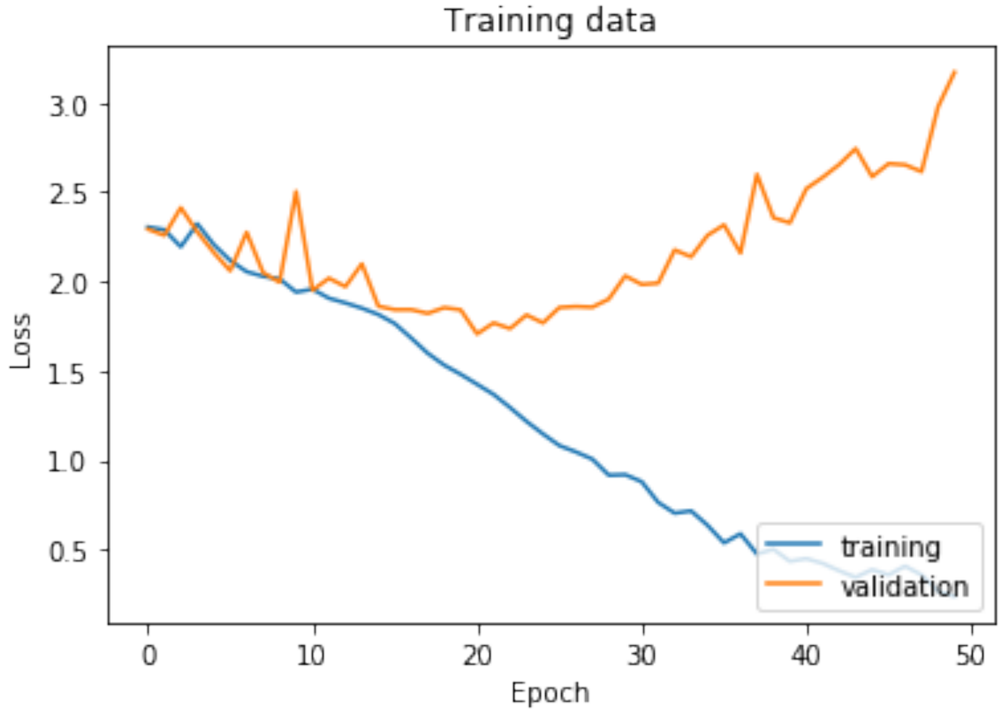
測試結果


```
 672/1000 [===================>.........] - ETA: 0s
 704/1000 [====================>........] - ETA: 0s
 736/1000 [=====================>.......] - ETA: 0s
 768/1000 [======================>......] - ETA: 0s
 800/1000 [=======================>.....] - ETA: 0s
 832/1000 [========================>.....] - ETA: 0s
 864/1000 [=========================>....] - ETA: 0s
 896/1000 [==========================>...] - ETA: 0s
 928/1000 [===========================>..] - ETA: 0s
 960/1000 [============================>.] - ETA: 0s
 992/1000 [=============================>.] - ETA: 0s
1000/1000 [==============================] - ETA: 0s
Test loss: 3.16952228546
Test accuracy: 0.372
```

訓練資料過少,驗證準確度較低



訓練資料少,特徵擷取少,
訓練成本和驗證成本不同步



模型預測

- predicted_classes=model.predict_classes(x_test[:1000])

```
array([5, 1, 8, 8, 4, 6, 3, 6, 6, 1, 0,
9, 6, 7, 6, 0, 5, 7, 8, 6, 6, 4, 4,
       9, 4, 5, 6, 0, 2, 6, 5, 4, 6, 6,
7, 6, 7, 9, 9, 2, 7, 2, 5, 6, 8, 9,
       6, 6, 2, 4, 9, 8, 5, 6, 8, 0, 7,
9, 2, 4, 4, 6, 6, 2, 6, 5, 9, 0, 3,….
```

顯示真實圖形和預測結果

```
fig=plt.figure()
fig.set_size_inches(6,6)
for i in range(0,9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[i],cmap='binary')
    plt.title(image_dictionary[np.argmax(y_test[i])]+':'
              +image_dictionary[predicted_classes[i]])
    plt.xticks([])
    plt.yticks([])

fig
```

顯示真實圖形和預測結果

- Thanks.