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。

– 台灣人工智慧學校

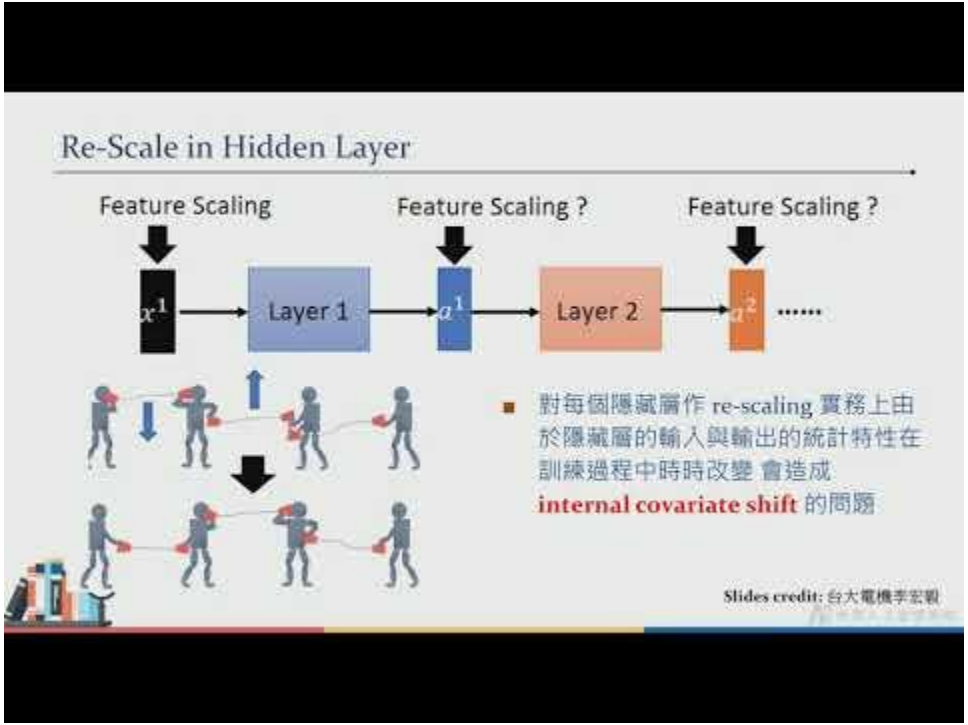
課程內容

神經網路的參數調控

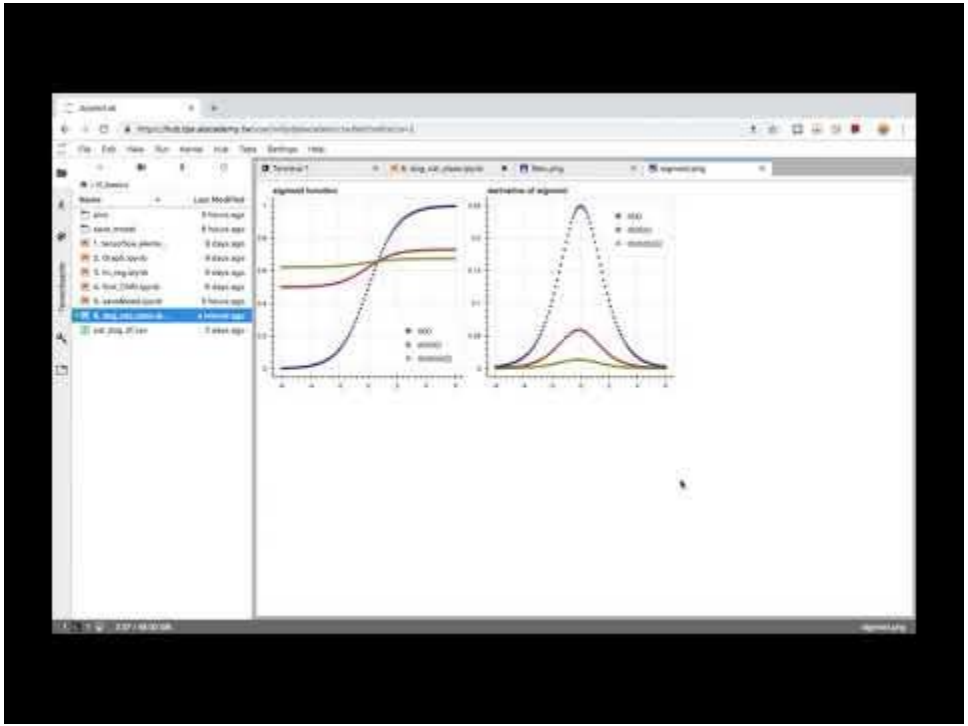
1. 貓狗分類器
2. 參數調控

神經網路的參數調控

理論講解 : 模型調教



貓狗分類器(6. dog_cat_class.ipynb)



資料觀察

	file_name	type
0	/data/examples/cat_dog/train/cat.8852.jpg	0
1	/data/examples/cat_dog/train/cat.11428.jpg	0
2	/data/examples/cat_dog/train/cat.12315.jpg	0
3	/data/examples/cat_dog/train/dog.4577.jpg	1
4	/data/examples/cat_dog/train/cat.3228.jpg	0
5	/data/examples/cat_dog/train/dog.10232.jpg	1
6	/data/examples/cat_dog/train/cat.11895.jpg	0
7	/data/examples/cat_dog/train/cat.1361.jpg	0
8	/data/examples/cat_dog/train/dog.8190.jpg	1
9	/data/examples/cat_dog/train/cat.546.jpg	0

資料路徑以及貓狗種類已經整理成 Dataframe 了

處理圖片檔案

- 利用套件 cv2 來處理圖片的輸入
- 讀取圖片成數字資訊 → `cv2.imread`
  - 為了簡化模型，我們讀取圖片時將圖片改成灰階
- 改變圖片大小 → `cv2.resize`
  - 訓練資料的大小統一才能放進模型訓練

處理圖片檔案(續)

```
def df_to_data(df, picsize):
    img_ls = []
    label_ls = []

    for file_name, label in zip(df.file_name, df.type):
        img = cv2.imread(file_name, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (picsize, picsize))
        img_ls.append(img)

        onehot = np.zeros(2)
        onehot[label] = 1
        label_ls.append(onehot)

    x_data = np.array(img_ls).reshape(len(df), -1)  # flatten image data
    x_data = x_data / 255.  # normalization
    y_data = np.array(label_ls)
    return x_data, y_data
```

讀取 jpg 檔成二維 numpy 向量，並改變長寬大小

二維的圖片向量拉長成一維向量，才能餵進 DNN 模型

處理圖片檔案(續)

- 資料準備完成了，可以來建構神經網路囉！

```
picsize = 64
X_train, y_train = df_to_data(train_df, picsize)
X_test, y_test = df_to_data(test_df, picsize)

print('size of training data:', X_train.shape, y_train.shape)
print('size of training data:', X_test.shape, y_test.shape)
```

```
size of training data: (20000, 4096) (20000, 2)
size of training data: (5000, 4096) (5000, 2)
```

建立Graph

```
tf.reset_default_graph()

with tf.name_scope('placeholder'):
    input_data = tf.placeholder(tf.float32, shape=[None, picsize*picsize], name='X')
    y_true = tf.placeholder(tf.float32, shape=[None, 2], name='y')

with tf.variable_scope('network'):
    h1 = tf.layers.dense(input_data, 256, activation=tf.nn.sigmoid, name='hidden1')
    h2 = tf.layers.dense(h1, 128, activation=tf.nn.sigmoid, name='hidden2')
    h3 = tf.layers.dense(h2, 64, activation=tf.nn.sigmoid, name='hidden3')
    out = tf.layers.dense(h3, 2, name='output')

with tf.name_scope('loss'):
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_true, logits=out), name='loss')

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(tf.nn.softmax(out), 1), tf.argmax(y_true, 1))
    compute_acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.name_scope('opt'):
    update = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(loss)

init = tf.global_variables_initializer()
```

建立一個三層的神經網路

檢查使用變數

```
[<tf.Variable 'network/hidden1/kernel:0' shape=(4096, 256) dtype=float32_ref>,
 <tf.Variable 'network/hidden1/bias:0' shape=(256,) dtype=float32_ref>,
 <tf.Variable 'network/hidden2/kernel:0' shape=(256, 128) dtype=float32_ref>,
 <tf.Variable 'network/hidden2/bias:0' shape=(128,) dtype=float32_ref>,
 <tf.Variable 'network/hidden3/kernel:0' shape=(128, 64) dtype=float32_ref>,
 <tf.Variable 'network/hidden3/bias:0' shape=(64,) dtype=float32_ref>,
 <tf.Variable 'network/output/kernel:0' shape=(64, 2) dtype=float32_ref>,
 <tf.Variable 'network/output/bias:0' shape=(2,) dtype=float32_ref>]
```

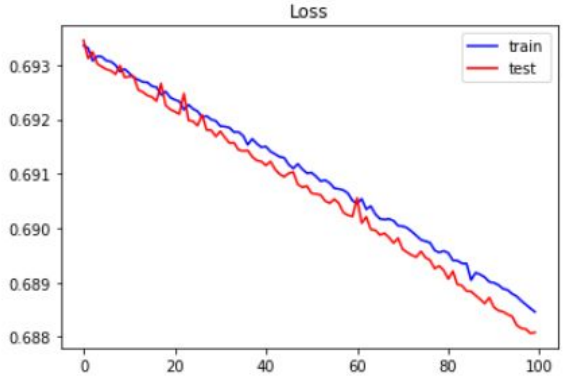
- 四組 dense layer，因此有八組的 weight。

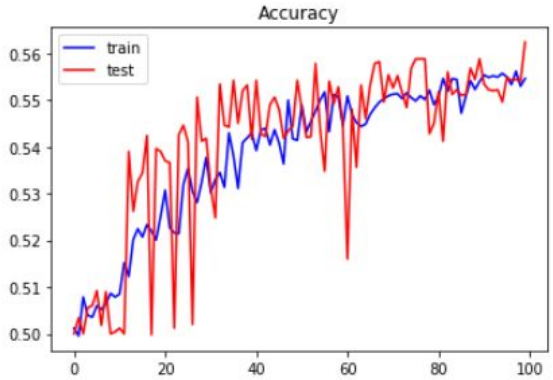
訓練模型囉～

```
epoch = 100
bs = 32

train_loss_epoch, train_acc_epoch = [], []
test_loss_epoch, test_acc_epoch = [], []

sess = tf.Session()
sess.run(init)

for i in tqdm_notebook(range(epoch)):

#     training part
    train_loss_batch, train_acc_batch = [], []

    total_batch = len(X_train) // bs

    for j in range(total_batch):

        X_batch = X_train[j*bs:(j+1)*bs]
        y_batch = y_train[j*bs:(j+1)*bs]
        batch_loss, batch_acc, _ = sess.run([loss, compute_acc, update],
                                            feed_dict={input_data: X_batch, y_true: y_batch})

        train_loss_batch.append(batch_loss)
        train_acc_batch.append(batch_acc)

    train_loss_epoch.append(np.mean(train_loss_batch))
    train_acc_epoch.append(np.mean(train_acc_batch))

#     validation part
    batch_loss, batch_acc = sess.run([loss, compute_acc],
                                     feed_dict={input_data: X_test, y_true: y_test})

    test_loss_epoch.append(batch_loss)
    test_acc_epoch.append(batch_acc)

    X_train, y_train = shuffle(X_train, y_train)
```

收集每個 batch 的 loss 及 accuracy

訓練結果(經過100個 epoch)





- loss 下降緩慢

- 準確率不及六成，只比隨機模型好一些

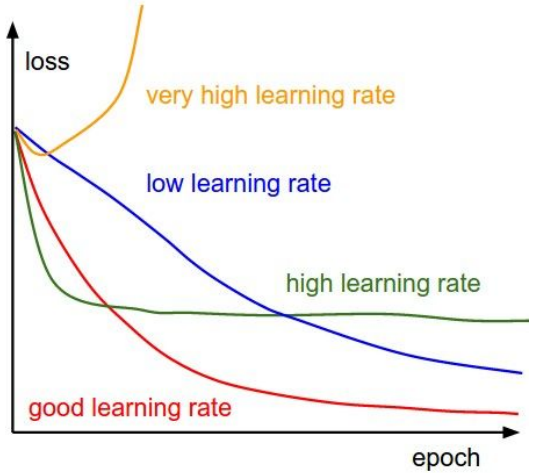
當 training set 訓練不太起來時...

- 將模型加大加厚！
  - 不一定是個好主意，容易增加模型過擬和的機會。

- 模型收斂有問題，改變一些參數讓模型容易收斂：
  - learning rate
  - activation function
  - optimizer

Learning rate

- 神經網路的更新步伐決定了學習的效率
- 調參時應該一次改變一個數量級(0.001, 0.01, 0.1 ...)

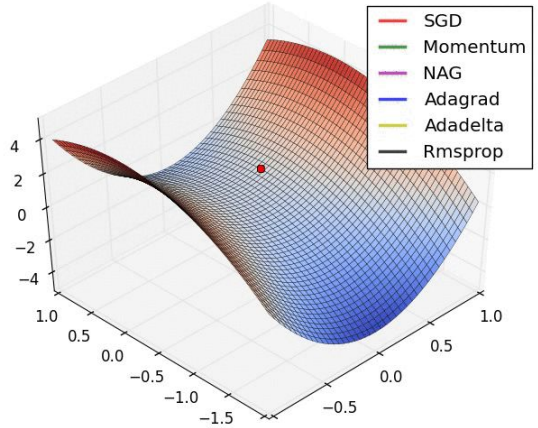


Activation function

- 非線性層使用不當可能會造成梯度消失
  - 以 sigmoid function 為例，通過多層的 sigmoid 容易使前幾層網路的梯度變小，難以更新。
  - 解決方法：可以使用 ReLU function 或其他函數來避免梯度消失。
  - 但這並不代表 sigmoid function 就是無用的，當要控制輸出映射在一定區間時，sigmoid function 還是相當有效。

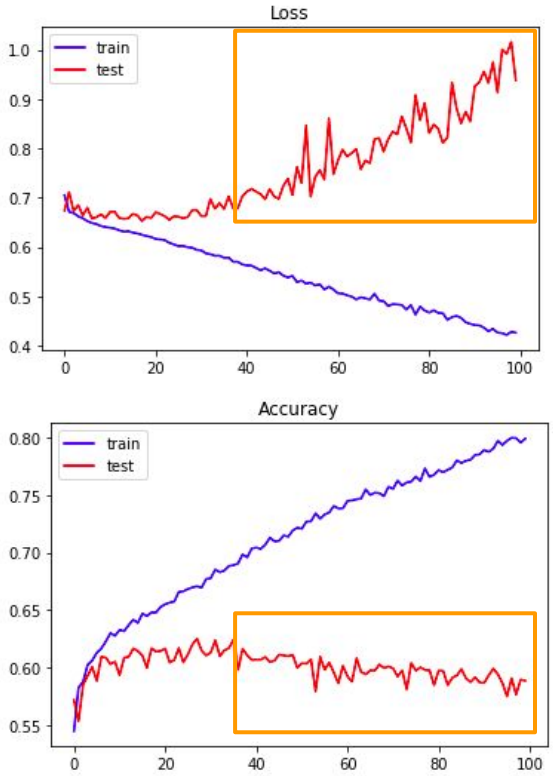
Optimizer

- 神經網路的向量空間相當複雜，在 loss function 中會有非常多的極值，但收斂到那些極值表現不一定好。
- 除了普通的梯度下降法，也可以使用其他較複雜的演算法來讓神經網路收斂快速且有效。（例如 Adam、RMSProp 等）



一張簡單的動畫圖展現
不同的演算法在處理
saddle point 的更新狀態

在經過各種調參後...



Training set 已經可以成功訓練了！
但是無法避免 overfitting...

練習時間

- 思考一下，overfitting 是不是代表我們參數使用的太多？
- 試試如果把層數或者神經元數減少，是不是就能訓練得更好？