



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。

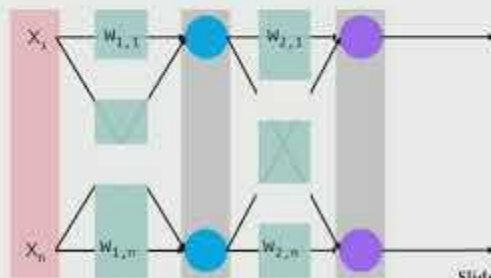
課程內容

1. 對抗過擬和
2. tf.keras API

理論講解：過擬和

Dropout

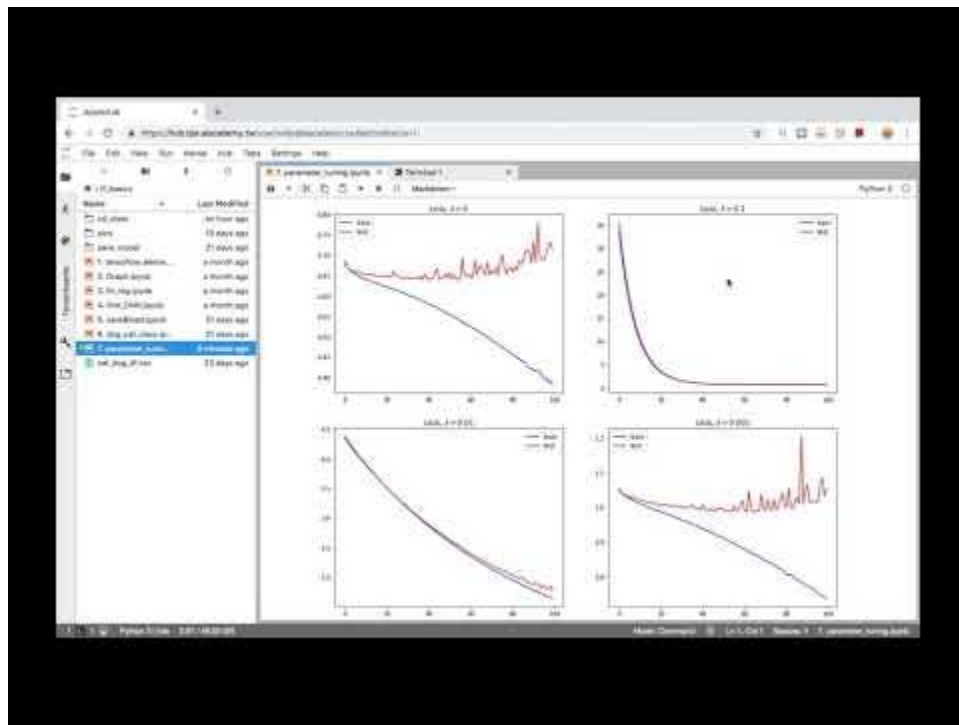
- 原本兩個鄰近 layer 的 neurons 為全連結 (fully connected)
- 在訓練過程中隨機 disable 一定比例的連結 (將weights 暫時設為零)



Slides credit: 台大電機李宏毅



實作對抗 overfitting (7. parameter_tuning.ipynb)

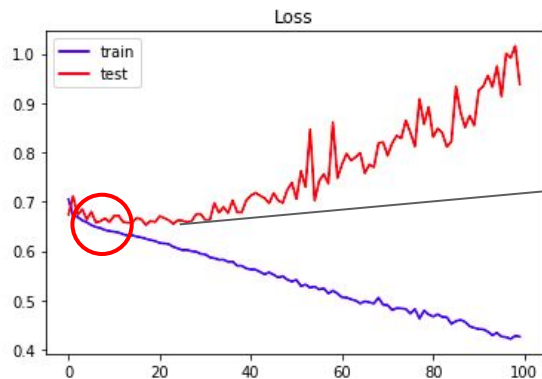


遇到 overfitting 我們可以使用的方法有...

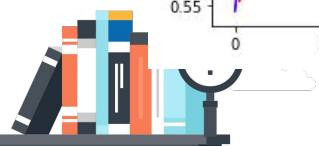
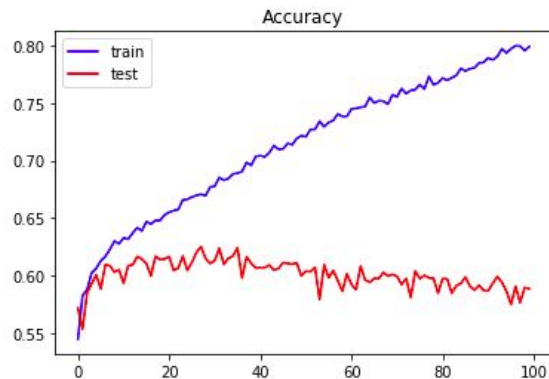
- Early stopping
- Regularization
- Dropout



Early stopping & Checkpoint



如果能在loss最低點
的地方停下來就好
了...



Early stopping & Checkpoint(續)

- Checkpoint: 當模型表現有進步就儲存 weight。
- Early stopping: 隨時監控 validation 的 loss 或準確率, 有 overfitting 的現象就跳出迴圈。
 - 設定一個閾值, N 次迴圈沒有進步就跳出迴圈。
- 跳出迴圈後, 把表現最好的 weight 載入回來使用。



Regularization

- 正規化：將神經網路的權重大小也放進損失函數做考量，降低深度學習的記憶力來達到避免 overfitting。

- $$L(x, y) = L'(x, y) + \lambda \sum_{i=1}^n \theta_i^2$$

原本的
loss function

L2
regularizer



Tensorflow 實作正規化

```
h1 = tf.layers.dense(input_data, 256, activation=tf.nn.relu, name='hidden1',  
                      kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=12))  
h2 = tf.layers.dense(h1, 128, activation=tf.nn.relu, name='hidden2',  
                      kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=12))  
h3 = tf.layers.dense(h2, 64, activation=tf.nn.relu, name='hidden3',  
                      kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=12))  
out = tf.layers.dense(h3, 2, name='output',  
                       kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=12))  
  
cross_loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_true, logits=out),  
                             name='cross_entropy')  
reg = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)  
loss = cross_loss + tf.reduce_sum(reg)
```

- 在 `tf.layers.dense` 的參數中多加上 `kernel_regularizer`
- 利用 `tf.get_collection` 把正規化的項次取出並和原本的 `loss` 合併



Dropout

- 當神經網路的隱藏層使用 dropout 時，**訓練**時此隱藏層的神經元會有一定的比率被捨棄掉，且每次捨棄的神經元都不一定相同，故訓練100次就如同訓練100個不同的神經網路。
- 最後在**測試**的時候，要將所有的神經元都要復原，不可捨棄。
- 由於抽掉了一定比例的神經元，訓練時的神經網路的記憶效果必不如原本的好。Dropout 有點類似 ensemble learning，訓練多個弱分類器，最後集成泛性較強的強分類器。



Tensorflow 實作 dropout

```
input_data = tf.placeholder(tf.float32, shape=[None, picsize*picsize], name='X')
y_true = tf.placeholder(tf.float32, shape=[None, 2], name='y')
dropout = tf.placeholder(tf.float32, shape=[], name='dropout')
training = tf.placeholder(tf.bool, name='training')

input_drop = tf.layers.dropout(input_data, rate=dropout, training=training, name='input_drop')
h1 = tf.layers.dense(input_drop, 256, activation=tf.nn.relu, name='hidden1')
h1 = tf.layers.dropout(h1, rate=dropout, training=training, name='h1_drop')
h2 = tf.layers.dense(h1, 128, activation=tf.nn.relu, name='hidden2')
h2 = tf.layers.dropout(h2, rate=dropout, training=training, name='h2_drop')
h3 = tf.layers.dense(h2, 64, activation=tf.nn.relu, name='hidden3')
out = tf.layers.dense(h3, 2, name='output')
```

- 建立 graph 時多創建兩個 placeholder 用於 dropout 的參數。
- 使用 `tf.layers.dropout` 放入要執行 dropout 的隱藏層、捨棄神經元的比例與訓練狀態。



Tensorflow 實作 dropout(續)

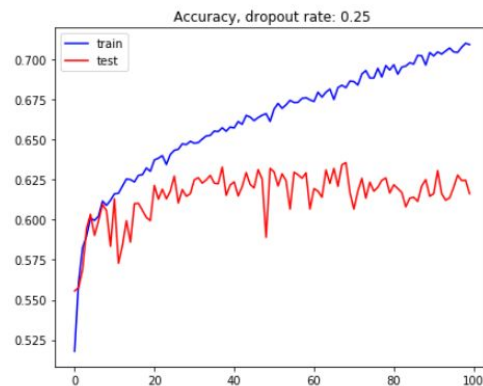
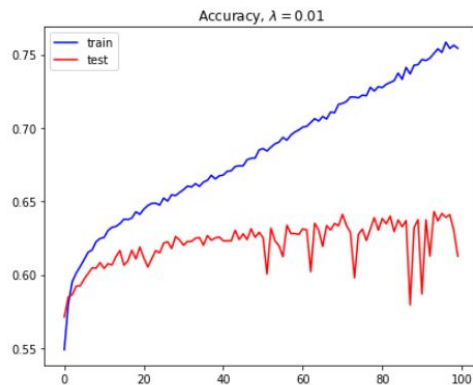
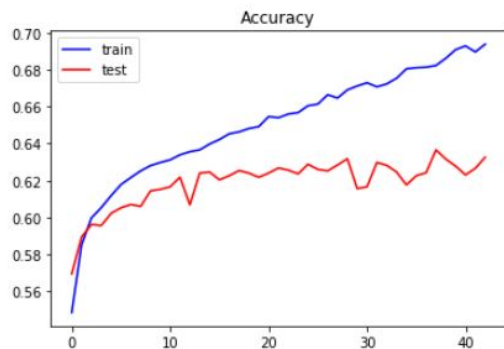
```
# training
batch_loss, batch_acc, _ = sess.run([loss, compute_acc, update],
                                     feed_dict={input_data: X_batch, y_true: y_batch,
                                                dropout: droprate, training: True})

# testing
batch_loss, batch_acc = sess.run([loss, compute_acc],
                                  feed_dict={input_data: X_test, y_true: y_test,
                                             dropout: droprate, training: False})
```

- 跑 session 時，訓練模型要把訓練狀態調成 True 神經網路才會 dropout。
反之訓練時應該要將訓練狀態調成 False



經過各種實驗後...

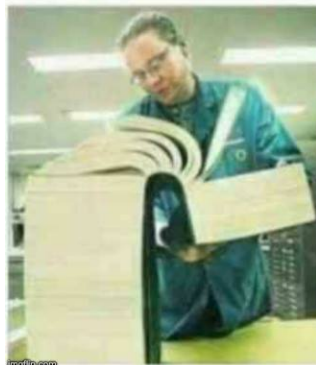


- 雖然 overfitting 的現象有減弱, 但是 testing 的準確率還是只有六成上下 ...
- 不用灰心, 之後的課程還可以嘗試其他種的神經網路 !



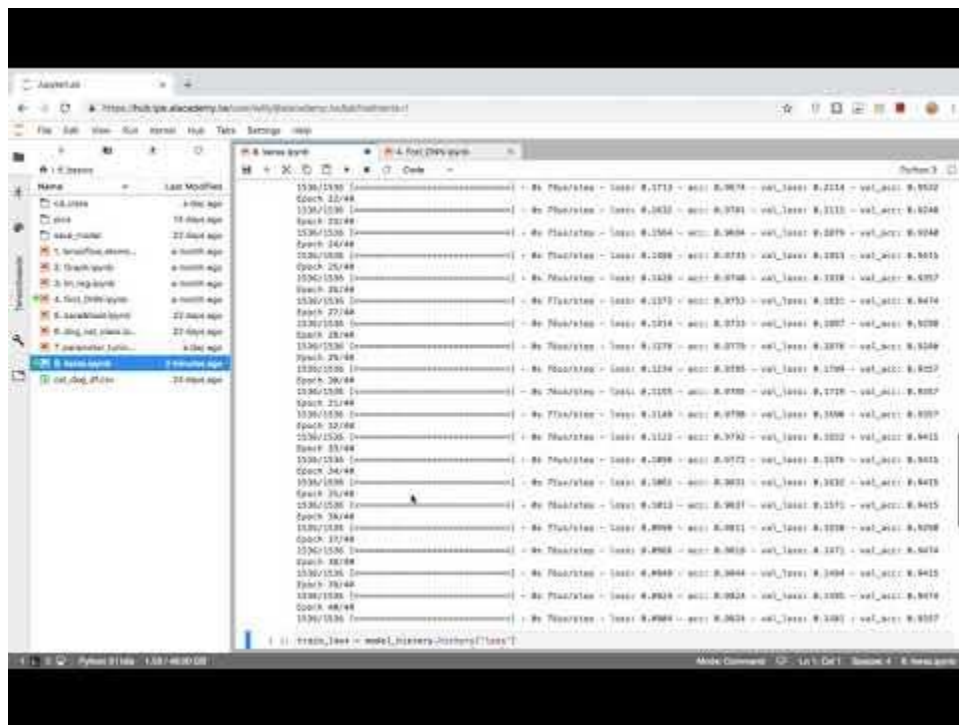
THE MATHS
BEHIND DEEP LEARNING

`import keras`



tf.keras API

tf.keras (8. keras.ipynb)



```
1530/1530 [====] - 6s 700u/step - loss: 0.3719 - acc: 0.3678 - val_loss: 0.2214 - val_acc: 0.3522
Epoch 12/48
1530/1530 [====] - 6s 700u/step - loss: 0.3632 - acc: 0.3781 - val_loss: 0.3110 - val_acc: 0.3248
Epoch 13/48
1530/1530 [====] - 6s 700u/step - loss: 0.3564 - acc: 0.3804 - val_loss: 0.2879 - val_acc: 0.3048
Epoch 14/48
1530/1530 [====] - 6s 700u/step - loss: 0.3498 - acc: 0.3773 - val_loss: 0.3051 - val_acc: 0.3615
Epoch 15/48
1530/1530 [====] - 6s 700u/step - loss: 0.3438 - acc: 0.3748 - val_loss: 0.3338 - val_acc: 0.3357
Epoch 16/48
1530/1530 [====] - 6s 700u/step - loss: 0.3373 - acc: 0.3753 - val_loss: 0.3232 - val_acc: 0.3474
Epoch 17/48
1530/1530 [====] - 6s 700u/step - loss: 0.3314 - acc: 0.3733 - val_loss: 0.3007 - val_acc: 0.3098
Epoch 18/48
1530/1530 [====] - 6s 700u/step - loss: 0.3279 - acc: 0.3778 - val_loss: 0.2878 - val_acc: 0.3248
Epoch 19/48
1530/1530 [====] - 6s 700u/step - loss: 0.3234 - acc: 0.3805 - val_loss: 0.3784 - val_acc: 0.3257
Epoch 20/48
1530/1530 [====] - 6s 700u/step - loss: 0.3185 - acc: 0.3788 - val_loss: 0.3496 - val_acc: 0.3307
Epoch 21/48
1530/1530 [====] - 6s 700u/step - loss: 0.3123 - acc: 0.3792 - val_loss: 0.3032 - val_acc: 0.3415
Epoch 22/48
1530/1530 [====] - 6s 700u/step - loss: 0.3058 - acc: 0.3772 - val_loss: 0.3478 - val_acc: 0.3612
Epoch 23/48
1530/1530 [====] - 6s 700u/step - loss: 0.3083 - acc: 0.3833 - val_loss: 0.3430 - val_acc: 0.3415
Epoch 24/48
1530/1530 [====] - 6s 700u/step - loss: 0.3013 - acc: 0.3827 - val_loss: 0.3157 - val_acc: 0.3415
Epoch 25/48
1530/1530 [====] - 6s 700u/step - loss: 0.2999 - acc: 0.3831 - val_loss: 0.3278 - val_acc: 0.3398
Epoch 26/48
1530/1530 [====] - 6s 700u/step - loss: 0.2968 - acc: 0.3818 - val_loss: 0.3373 - val_acc: 0.3474
Epoch 27/48
1530/1530 [====] - 6s 700u/step - loss: 0.2949 - acc: 0.3844 - val_loss: 0.3494 - val_acc: 0.3415
Epoch 28/48
1530/1530 [====] - 6s 700u/step - loss: 0.2939 - acc: 0.3824 - val_loss: 0.3300 - val_acc: 0.3474
Epoch 29/48
1530/1530 [====] - 6s 700u/step - loss: 0.2984 - acc: 0.3824 - val_loss: 0.3381 - val_acc: 0.3357
```



Tensorflow 的缺點

- Tensorflow 的觀念對初學者來說不易理解
- 明明是在寫 Python 的程式, 而 Python 的哲學就是 Simple is better than complex.
難道沒有更容易的方法建立 DNN 模型了嗎？
- 有的！那就是利用 tf.keras！



Sequential model

- 利用 `tf.keras.Sequential` 快速建模
- 搭配 `add` 的方法來堆疊模型

In keras:

```
model = tf.keras.Sequential()  
model.add(Dense(25, activation='relu', input_shape=(64,)))  
model.add(Dense(10, activation='softmax'))
```

`tf.keras` 在算 loss 的時候
不會自動通過 softmax,
必須設定讓最後一層通
過 softmax。

In tensorflow:

```
x_input = tf.placeholder(shape=(None, 64), name='x_input', dtype=tf.float32)  
y_out = tf.placeholder(shape=(None, 10), name='y_label', dtype=tf.float32)  
x_h1 = tf.layers.dense(inputs=x_input, units=25, activation=tf.nn.relu)  
output = tf.layers.dense(x_h1, 10, name='output')
```



Sequential model(續)

- 使用 compile 的方法來設定 loss 及優化器

In keras:

```
model.compile(loss='categorical_crossentropy',  
              optimizer=tf.keras.optimizers.Adam(),  
              metrics=['accuracy'])
```

In tensorflow:

```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=output, labels=y_out), name='loss')  
train_step = tf.train.AdamOptimizer().minimize(loss)  
correct_prediction = tf.equal(tf.argmax(tf.nn.softmax(output), 1), tf.argmax(y_out, 1))  
compute_acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```



Sequential model (續)

- 使用 fit 的方法來訓練模型

In keras:

```
model_history = model.fit(x=x_train, y=y_train,  
                           batch_size=batch_size,  
                           epochs=epochs,  
                           validation_split=1 - train_ratio)
```

In tensorflow:

```
train_loss_epoch, valid_loss_epoch = [], []  
train_acc_epoch, valid_acc_epoch = [], []  
  
sess = tf.Session()  
  
sess.run(tf.global_variables_initializer())  
  
for i in tqdm_notebook(range(epochs)):  
  
    total_batch = len(x_train) // batch_size  
    train_loss_in_batch, train_acc_in_batch = [], []  
  
    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})  
  
        train_loss_in_batch.append(this_loss)  
        train_acc_in_batch.append(this_acc)  
  
    valid_acc, valid_loss = sess.run([compute_acc, loss],  
                                     feed_dict={x_input: x_valid, y_out : y_valid})  
  
    valid_loss_epoch.append(valid_loss)  
    valid_acc_epoch.append(valid_acc)  
    train_loss_epoch.append(np.mean(train_loss_in_batch))  
    train_acc_epoch.append(np.mean(train_acc_in_batch))  
  
    x_train, y_train = shuffle(x_train, y_train)
```

兩者的 code 量天壤之別...



Sequential model(續)

- 最後使用 `predict` 的方法來用模型預測

In keras:

```
y_predict = model.predict(x_test)
```

In tensorflow:

```
y_predict = sess.run(tf.nn.softmax(output), feed_dict={x_input:x_test})
```



儲存載入模型

- 對 code 簡化最注重的 tf.keras, 模型的儲存及載入也相當簡單

```
# save model
model.save('my_model.h5')

# Load model
another_model = tf.keras.models.load_model('my_model.h5')
```



小總結

- tf.keras 的 Sequential model 讓神經網路的架構更簡單，概念也更直白。
- 雖然在 tf.keras 的程式裡沒有看到 graph session，但其實程式底層運作的架構都還是由 Tensorflow 最基礎的元素運作。



練習時間

- 利用 tf.keras 的 Sequential model 來建立模型訓練貓狗分類器

