國立高雄科技大學
National Kaohsiung University of Science and Technology

深度學習理論與實作
CH4 機器學習的基礎知識

資工系 陳俊豪 教授

Outline

4-1 機器學習的四個分支

4-2 評估機器學習模型

4-3 資料預處理、特徵工程和特徵學習

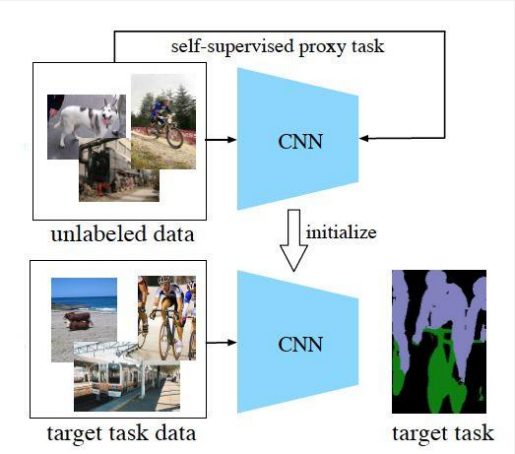
4-4 過度配適、低度配適

4-5 機器學習的運用工作流程

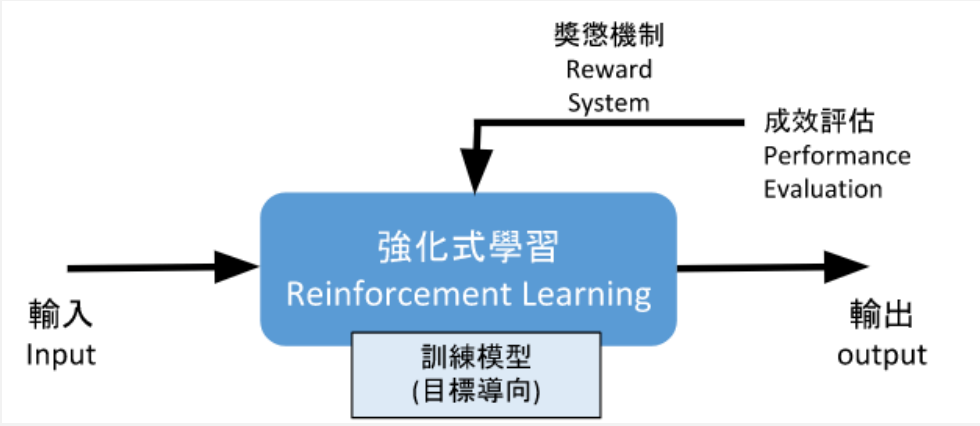
4-1 機器學習的四個分支

監督式學習(Supervised learning)

非監督式學習(Unsupervised Learning)



自監督式學習(Self-supervised Learning)



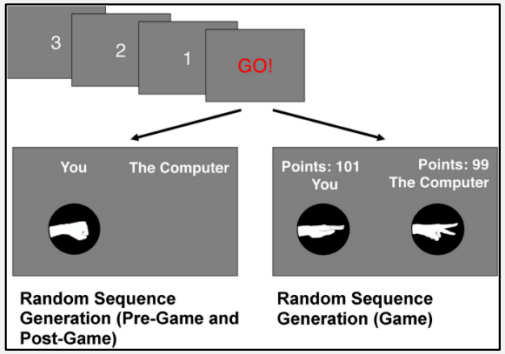
增強式學習(Reinforcement learning)

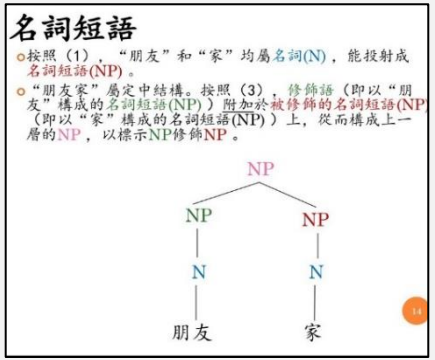
4-1-1 監督式學習 Supervised Learning

- ✓ 訓練資料(training data)都有相對應的人工標註標籤(label)
- ✓ 除分類、迴歸，也包含下列：

語法樹預測(syntax tree prediction)

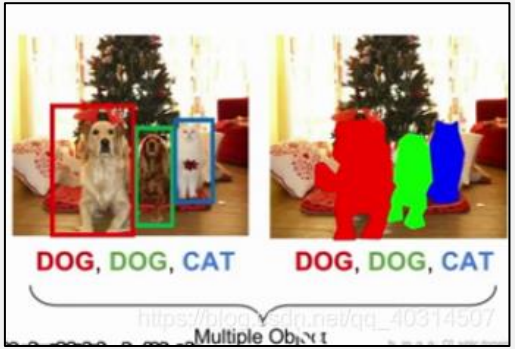
以語意的結構為節點，預測並分解成語法樹

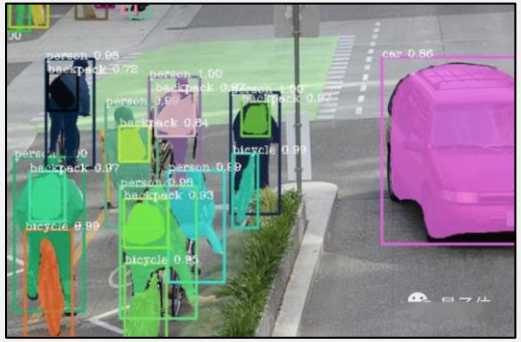




物體偵測(object detection)

繪製邊框來標示圖片內不同的物品





序列生成(sequence generation)

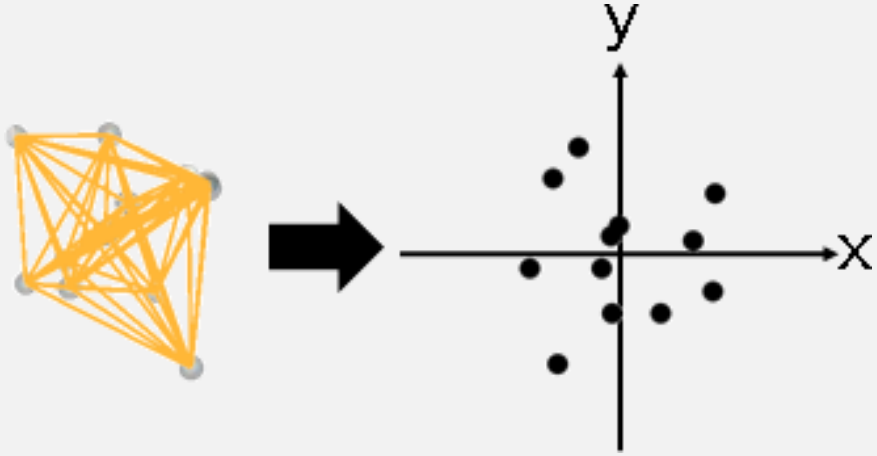
使用連續資料進行預測

圖像分割(image segmentation)

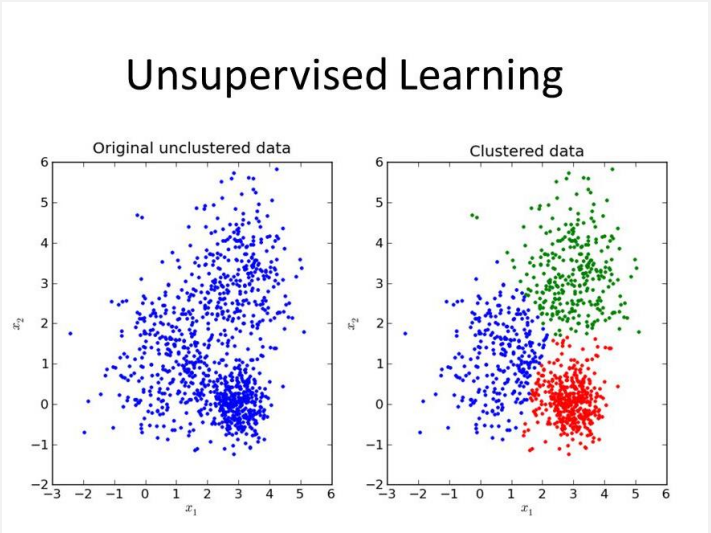
用像素遮罩(pixel-level mask)區別不同的物體

4-1-2 非監督式學習 Unsupervised Learning

- ✓ 輸入資料沒有任何標籤，要機器理解資料，找出有意義的表示法
  - 主要用途：資料視覺化、資料壓縮、資料降噪或理解資料的相關性
- ✓ 範例如下



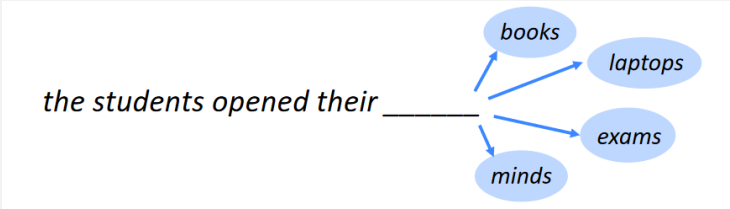
維度降低(dimensionality reduction)



分群(clustering)

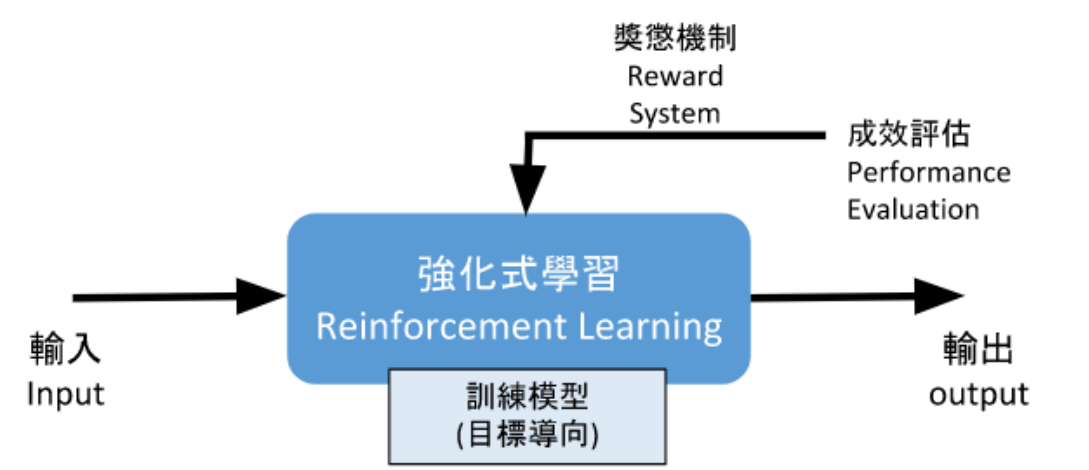
4-1-3 自監督式學習

- ✓ 沒有人工標註標籤的監督式學習，從輸入資料自動生成標籤
  - 通常使用啟發式演算法(heuristic algorithm)
  - 補充：啟發式演算法：受到自然啟發、重複疊代的演算法就被稱作啟發式演算法，如：遺傳演算法
- ✓ 案例
  - 自動編碼器(autoencoders)
  - 時間性監督式學習：由未來資料判定預測結果是否準確，例如：
    - ✓ 從前面的影格，預測下一幅影格
    - ✓ 從文章之前的字詞預測下一個字詞



4-1-4 增強式學習 Reinforcement learning

✓ 代理人(agent)接收環境相關資訊，並學會選擇最大獎勵的行動



案例：

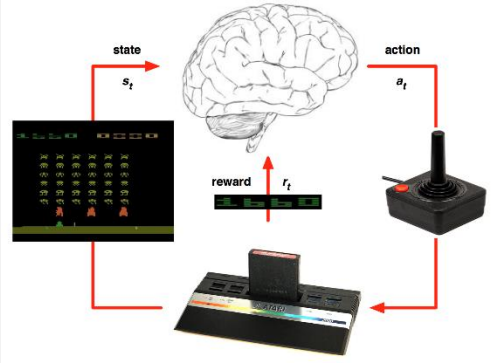
✓ 遊戲

✓ 自駕車

✓ 機器人技術

✓ 資源管理

Atari 遊戲



4-1 複習：分類和迴歸詞彙表

- ✓ 樣本(sample)或輸入(input) ➔ 模型的輸入
- ✓ 預測(prediction)或輸出(output) ➔ 模型的輸出
- ✓ 目標(target) ➔ 正確答案
- ✓ 真實狀況(ground-truth)或標註集(annotations)
  - 資料集所對應的目標(所有正確答案)
- ✓ 標籤(label) ➔ 監督式學習的分類問題中，伴隨的答案
- ✓ 類別(class) ➔ 貓狗分類
- ✓ 預測誤差(prediction error)或損失值(loss) ➔ 預測與目標的差距

4-1 複習：分類和迴歸詞彙表

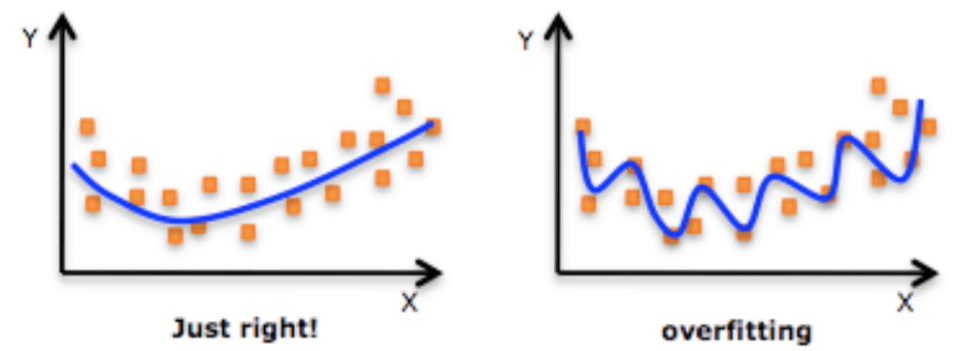
- ✓ 二元分類(binary classification) ➔ 評論正評負評
- ✓ 多類別分類(multiple classification) ➔ 手寫數字0~9
- ✓ 多標籤分類(multilabel classification)
  - 一張圖片有多個類別，同時存在貓狗影像的圖片
- ✓ 純量迴歸(scalar regression) ➔ 預測房價，目標是連續的純量數值
- ✓ 向量迴歸(vector regression) ➔ 目標是一組連續的數值
- ✓ 小批次量(mini-batch)或批次量(batch)
  - 模型同時處理的一組樣本，通常為2的次方倍

4-2 評估機器學習模型

- ✓ 過度配適(overfitting)
  - 模型在訓練資料的表現，隨著訓練的進行不斷改善，但模型在前所未見的測試資料上，表現則出現停滯或惡化的現象，造成模型的普遍適用(generalize)能力較差
- ✓ 評估機器學習模型：如何減輕過度配適? 如何達到最大普適化?
  - 將模型切為訓練集(training set)、驗證集(validation set)、測試集(testing set)

4-2-1 把資料分成訓練集，驗證集和測試集

- ✓ 模型開發過程中，要調整模型結構，如：
  - 模型超參數(hyperparameters)：模型要有多少層(深度)，每一層規模要多大(寬度)
  - 權重參數(weight parameters)：權重可以被看做神經元之間的連接強度
- ✓ 按照驗證集的回饋資訊進行參數調整時，即使模型從未直接用驗證集進行訓練，也可能很快導致驗證集過度配適，導致**資訊洩漏**
  - 資訊洩漏(information leak)：根據每次模型在驗證集上面的表現調整參數時，驗證集資料的資訊就會洩漏到模型中，這樣所訓練出來的模型，普適化程度降低



4-2-1 把資料分成訓練集，驗證集和測試集

✓ 簡單拆分驗證(simple hold-out verification)

```
num_validation_samples = 10000 #設定預計的驗證資料數據筆數：10000 筆
np.random.shuffle(data) #打散資料，重新洗牌

validation_data = data[:num_validation_samples] #定義前 10000 筆(0~9999)為驗證集
data = data[num_validation_samples:] #定義第 10000 筆後(10000~)為訓練集
training_data = data[:]

model = get_model()
model.train(training_data)
validation_score = model.evaluate(validation_data) #以訓練集資料進行模型訓練，並以驗證集資料進行模型評估
----------------------對模型進行調整，再進行訓練與評估，再調整----------------------------
model=get_model()
model.train(np.concatenate([training_daa,validation_data])) #當完成超參數調整後，通常會從頭再用所有資料訓練一次
test_score=model.evaluate(test_data)
```

4-2-1 把資料分成訓練集，驗證集和測試集

✓ K折驗證(K-fold validation)

```
k = 4 #設定 4 個區塊
num_validation_samples = len(data) // k #將資料筆數除以區塊數(此例為 4)
np.random.shuffle(data)

validation_scores = []
for fold in range(k):
    validation_data = data[num_validation_samples * fold:  #選擇驗證資料區塊
                           num_validation_samples * (fold + 1)]
    training_data = data[:num_validation_samples * fold] +
                    data[num_validation_samples * (fold + 1):] #使用剩餘的資料做為訓練資料，注意到+運算子是指串列(list)的連接，而不是加總
    model = get_model()  #建立一個全新的模型(未經訓練)
    model.train(training_data)
    validation_score = model.evaluate(validation_data)
    validation_scores.append(validation_score)
validation_score = np.average(validation_scores)  #驗證分數：k 折驗證分數的平均值

model = get_model()
model.train(data)
test_score = model.evaluate(test_data)  #以所有可用的非測試資料訓練最終模型
```

4-2-1 把資料分成訓練集，驗證集和測試集

- ✓ 多次洗牌的K折驗證(Iterated K-fold validation with shuffling)
  - 比賽中常用，每次分割前把資料把打散，如果執行P回，那就是會評估過P*K次，運算成本較高

4-2-2 驗證的注意事項

✓ 資料代表性

- 如果資料的排序有其規律存在，要隨機打散資料

✓ 時間方向性

- 預測未來狀態(例如：股票走勢)，不應該再拆分資料前隨機打散資料，會造成時間漏失

✓ 資料中的重複現象

- 資料集中的資料有重複，又剛好被切分成訓練集與驗證集，會造成訓練出來的模型不可信

4-3-1 神經網路資料的預處理

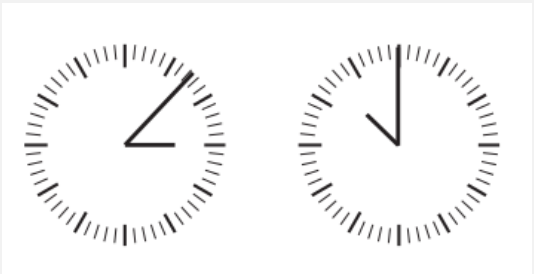
- ✓ 向量化(vectorization)：神經網路中，無論是什麼資料，需要先轉成張量
- ✓ 數值正規化(normalization/feature scaling)
  - 不同類型的資料，資料呈現的區間不同(年齡、收入...)，差距太大會不利於神經網路收斂
    - ✓ 通常資料整理到0~1之間的浮點數
    - ✓ 同質性：所有特徵都採用大致相同的數值範圍
    - ✓ 單獨正規化每個特徵，使平均值為0
    - ✓ 單獨正規化每個特徵，使標準差為1
    - ✓ 數字圖片分類就不需要正規化
- ✓ 處理缺失值(missing values)：如果0是沒有定義的值，就可以補0
  - 如果知道訓練樣本中沒有缺失值，但是測試樣本中有缺失值，這樣就需要人工複製一些訓練樣本，並將其對應的值刪除

4-3-2 特徵工程 (Feature Engineering)

✓ 訓練模型前

- 透過自身的知識對手邊資料與機器學習演算法的理解，直接以人工的方式去轉換資料(非經由機器學習)
- 良好的特徵允許用更少的資源解決問題，深度學習仰賴大量樣本，當樣本數不足時，特徵就顯得更重要

原始資料



較佳特徵(轉換成座標)

Better features: clock hands' coordinates	{x1: 0.7, y1: 0.7} {x2: 0.5, y2: 0.0}	{x1: 0.0, y2: 1.0} {x2: -0.38, 2: 0.32}

最佳特徵(轉化成角度)

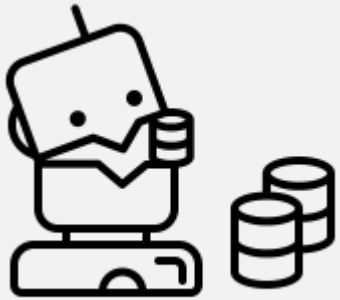
Even better features: angles of clock hands	theta1: 45 theta2: 0	theta1: 90 theta2: 140

用於讀取時鐘時間的特徵工程

4-4 過度配適、低度配適

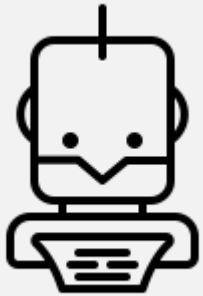
✓ 為了防止模型在訓練資料中學到錯誤或不相關的模式：

最好的解決方案



取得更多的訓練資料

次佳的解決方案



調配模型儲存的資訊量或者限制儲存資訊的類型或數值

補充：

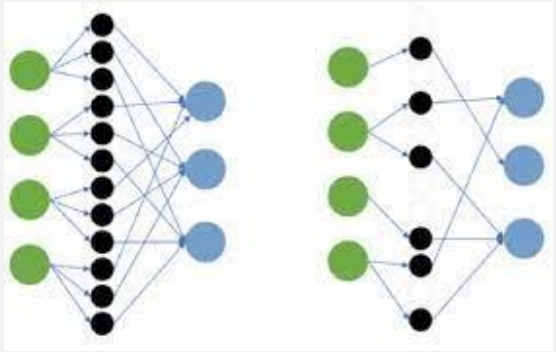
調整參數以對抗overfitting的方式稱為**常規化(regularization)**

當模型過度偏向訓練資料，甚至把訓練資料中的雜訊當成學習對象，學了不該學的，常規化就如同學校校規，避免學生發生一些異常的舉止！

4-4 過度配適、低度配適防止方法

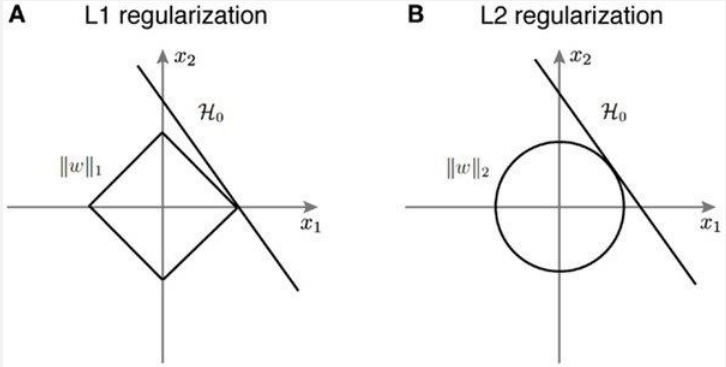
✓ 防止過度配適、低度配適的方法：

1. 縮減神經網路的大小



減少模型可用來學習的參數數量

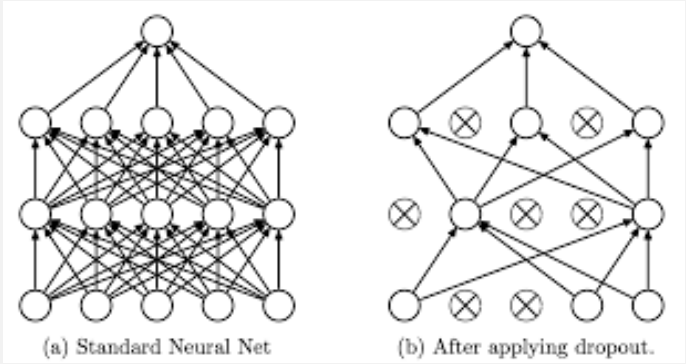
2. 加入權重常規化 weight regularization



對損失函數中較大的權重加上代價(cost)項目

- L1常規化：cost項和權重係數的絕對值成正比
- L2常規化=權重衰減(weight decay)：cost項和權重係數的平方成正比

3. 丟棄法 dropout



主要訓練期間隨機丟棄(dropping out) layer的一些輸出特徵，把feature值設為零

1. 縮減神經網路的大小

✓ 利用IMDB電影資料為例，下載資料集建立訓練與測試資了

```
1.  from keras.datasets import imdb
2.  import numpy as np
3.  (train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
4.  def vectorize_sequences(sequences, dimension=10000):
5.      results = np.zeros((len(sequences), dimension))
6.      for i, sequence in enumerate(sequences):
7.          results[i, sequence] = 1.
8.      return results

9.  x_train = vectorize_sequences(train_data)
10. x_test = vectorize_sequences(test_data)
11. y_train = np.asarray(train_labels).astype('float32')
12. y_test = np.asarray(test_labels).astype('float32')
```

1. 縮減神經網路的大小(Cont.)

✓ 原始模型與縮小版模型

```
1.  #原始模型， 16 個單元
2.  from keras import models
3.  from keras import layers
4.  original_model = models.Sequential()
5.  original_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
6.  original_model.add(layers.Dense(16, activation='relu'))
7.  original_model.add(layers.Dense(1, activation='sigmoid'))
8.  original_model.compile(optimizer='rmsprop‘, loss='binary_crossentropy‘, metrics=['acc'])

9.  #容量較低的模型版本， 4 個單元
10. smaller_model = models.Sequential()
11. smaller_model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))
12. smaller_model.add(layers.Dense(4, activation='relu'))
13. smaller_model.add(layers.Dense(1, activation='sigmoid'))
14. smaller_model.compile(optimizer='rmsprop‘, loss='binary_crossentropy‘, metrics=['acc'])
```

1. 縮減神經網路的大小(Cont.)

✓ 開始訓練網路，請分別接在原始模型與較低模型程式碼之後

```
#原始模型， 16 個單元
original_hist = original_model.fit(x_train, y_train,
                                   epochs=20,
                                   batch_size=512,
                                   validation_data=(x_test, y_test))


#容量較低的模型版本， 4 個單元
smaller_model_hist = smaller_model.fit(x_train, y_train,
                                       epochs=20,
                                       batch_size=512,
                                       validation_data=(x_test, y_test))
```

1. 縮減神經網路的大小(Cont.)

✓ 訓練結果

#原始模型， 16 個單元

```
49/49 [==============================] - 1s 14ms/step - loss: 0.0456 - acc: 0.9877 - val_loss: 0.5482 - val_acc: 0.8586
Epoch 18/20
49/49 [==============================] - 1s 14ms/step - loss: 0.0393 - acc: 0.9894 - val_loss: 0.5943 - val_acc: 0.8568
Epoch 19/20
49/49 [==============================] - 1s 15ms/step - loss: 0.0339 - acc: 0.9920 - val_loss: 0.6030 - val_acc: 0.8574
Epoch 20/20
49/49 [==============================] - 1s 14ms/step - loss: 0.0329 - acc: 0.9919 - val_loss: 0.6262 - val_acc: 0.8553
```

#容量較低的模型版本， 4 個單元

```
49/49 [==============================] - 1s 15ms/step - loss: 0.0927 - acc: 0.9728 - val_loss: 0.3772 - val_acc: 0.8730
Epoch 18/20
49/49 [==============================] - 1s 16ms/step - loss: 0.0870 - acc: 0.9739 - val_loss: 0.3931 - val_acc: 0.8711
Epoch 19/20
49/49 [==============================] - 1s 15ms/step - loss: 0.0811 - acc: 0.9761 - val_loss: 0.4080 - val_acc: 0.8700
Epoch 20/20
49/49 [==============================] - 1s 14ms/step - loss: 0.0760 - acc: 0.9782 - val_loss: 0.4286 - val_acc: 0.8691
```

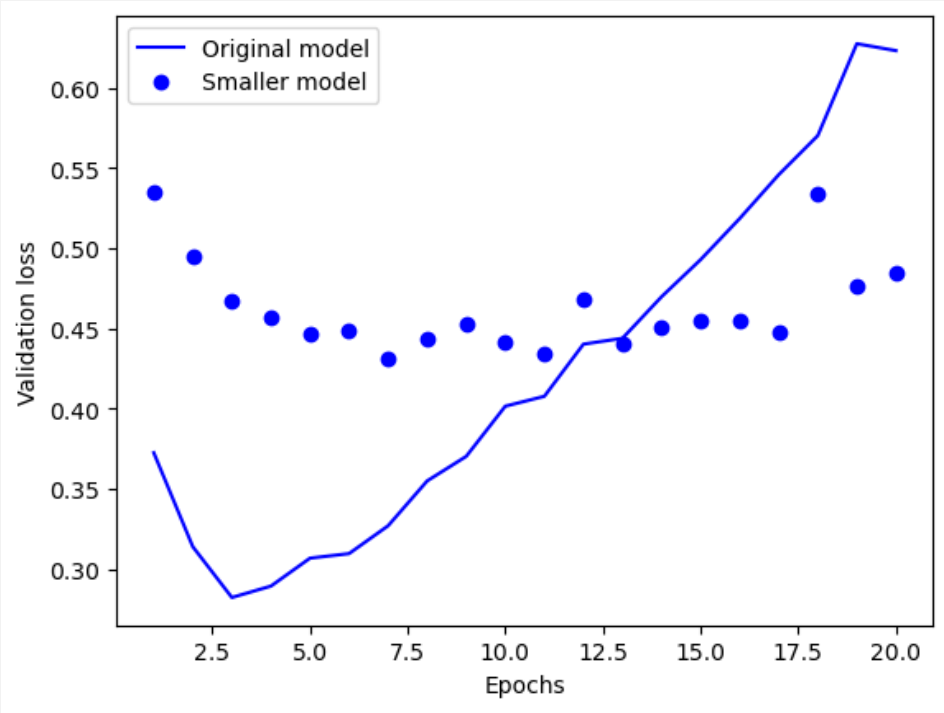
容量較低版本的模型，效果更好 ➔ 小問題用小模型，大問題再用大模型

1. 縮減神經網路的大小(Cont.)

✓ 繪製相關圖表

```
1.  epochs = range(1, 21)
2.  original_val_loss = original_hist.history['val_loss']
3.  smaller_model_val_loss =
    smaller_model_hist.history['val_loss']

4.  import matplotlib.pyplot as plt
5.  plt.plot(epochs, original_val_loss, 'b-',
    label='Original model')
6.  plt.plot(epochs, smaller_model_val_loss, 'bo',
    label='Smaller model')
7.  plt.xlabel('Epochs')
8.  plt.ylabel('Validation loss')
9.  plt.legend()
10. plt.show()
```



2. 加入權重常規化

✓ 使用同樣資料，模型建置如下:

```
1. from keras import regularizers


2. l2_model = models.Sequential()
3. #加入 L2 權重常規化並將學習率設為 0.001
4. l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
5.                           activation='relu', input_shape=(10000,)))
6. l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
7.                           activation='relu'))
8. l2_model.add(layers.Dense(1, activation='sigmoid'))
```

2. 加入權重常規化(Cont.)

✓ 編譯模型

```
9.  l2_model.compile(optimizer='rmsprop',
10.                  loss='binary_crossentropy',
11.                  metrics=['acc'])
```

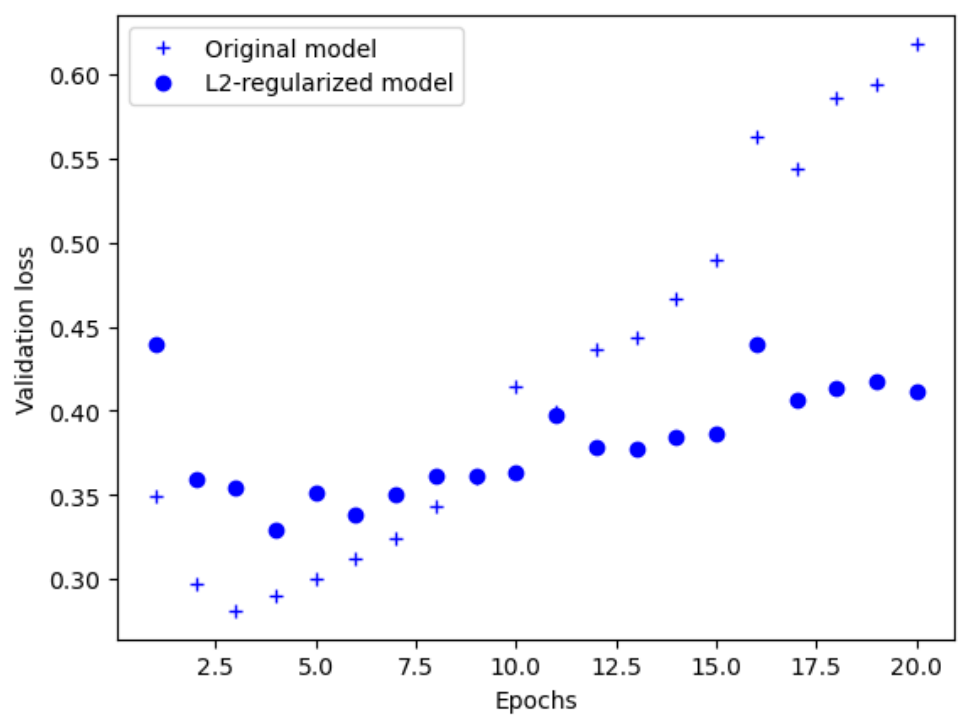
✓ 開始訓練模型

```
12. l2_model_hist = l2_model.fit(x_train, y_train,
13.                              epochs=20,
14.                              batch_size=512,
15.                              validation_data=(x_test, y_test))
```

2. 加入權重常規化(Cont.)

✓ 繪製原始模型與L2模型的驗證資料損失值

```
1. import matplotlib.pyplot as plt
2. l2_model_val_loss =
   l2_model_hist.history['val_loss']
3. plt.plot(epochs, original_val_loss, 'b+',
   label='Original model')
4. plt.plot(epochs, l2_model_val_loss, 'bo',
   label='L2-regularized model')
5. plt.xlabel('Epochs')
6. plt.ylabel('Validation loss')
7. plt.legend()
8. plt.show()
```



2. 加入權重常規化(Cont.)

- ✓ Keras 提供不同的權重常規化 ➔ L1 、 L2常規化或同時使用
- ✓ 上述範例是使用L2, 如使用L1或同時使用，部分程式碼可改寫如下：

```
1. from keras import regularizers

2. regularizers.l1(0.001)  #L1 常規化

3. regularizers.l1_l2(l1=0.001, l2=0.001)  #同時使用 L1 及 L2 常規化
```

3. 丟棄法 dropout

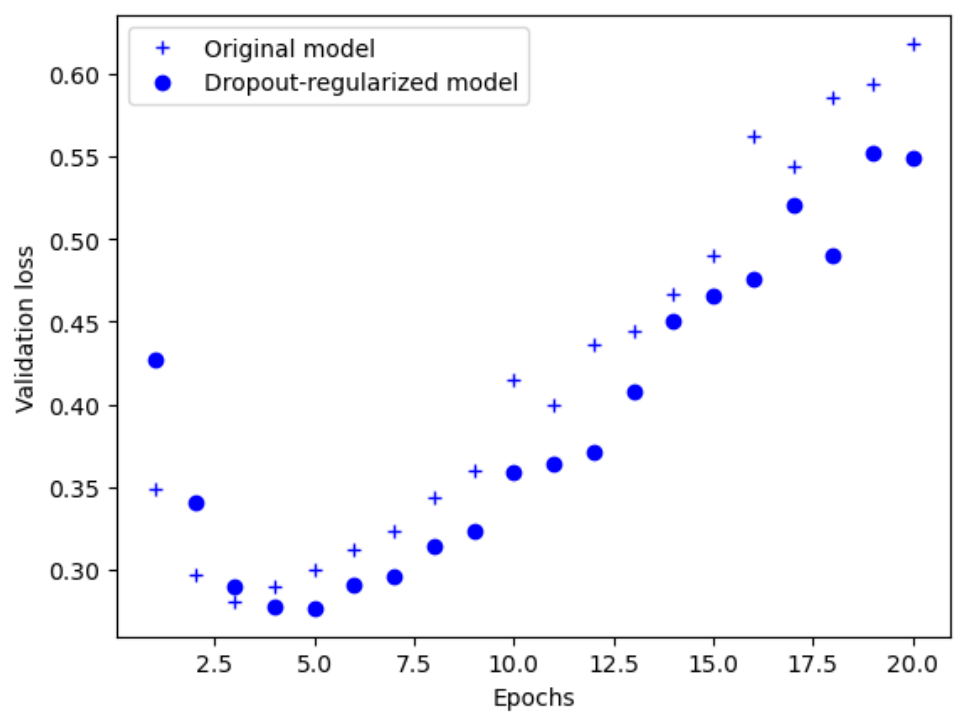
- ✓ 將 Dropout 層添加到 IMDB 神經網路
- ✓ 模型建置、編譯與訓練程式碼如下：

```
1.  dpt_model = models.Sequential()
2.  dpt_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
3.  dpt_model.add(layers.Dropout(0.5))
4.  dpt_model.add(layers.Dense(16, activation='relu'))
5.  dpt_model.add(layers.Dropout(0.5))
6.  dpt_model.add(layers.Dense(1, activation='sigmoid'))
7.  dpt_model.compile(optimizer='rmsprop', loss='binary_crossentropy',
8.                    metrics=['acc'])
9.  dpt_model_hist = dpt_model.fit(x_train, y_train,
10.                               epochs=20,
11.                               batch_size=512,
12.                               validation_data=(x_test, y_test))
```

3. 丟棄法 dropout

✓ 繪製原始模型與丟棄法模型的驗證資料損失值

1. dpt_model_val_loss = dpt_model_hist.history['val_loss']
2. plt.plot(epochs, original_val_loss, 'b+', label='Original model')
3. plt.plot(epochs, dpt_model_val_loss, 'bo', label='Dropout-regularized model')
4. plt.xlabel('Epochs')
5. plt.ylabel('Validation loss')
6. plt.legend()
7. plt.show()



4-5 機器學習的運用工作流程

Step1: 定義問題並建立資料集

Step2: 選擇一種評量成功的準則

Step3: 決定驗證程序

Step4: 準備資料

Step5: 開發出基於基準(baseline)的模型

Step6: 擴大規模：開發一個過度配適的模型

Step7: 常規化模型並調整超參數

4-5-1 定義問題並建立資料集

- ✓ 輸入資料是什麼?
- ✓ 面臨什麼類型的問題?
- ✓ 假設機器可以根據給定的輸入預測出結果
- ✓ 假設手上的資料能提供足夠資訊，讓機器能學習到輸入與輸出間的關係

- ✓ 需要注意：非平穩的問題(nonstationary problems)
  - 例如：建構服裝銷售推薦系統時，要注意人們購買衣服的種類會隨季節而變化，所以服裝購買在幾個月內是**非平穩現象**，嘗試建模的內容會隨著時間而變化，故分析資料就要在相對平穩的時間區(具有規律的週期間)蒐集資料

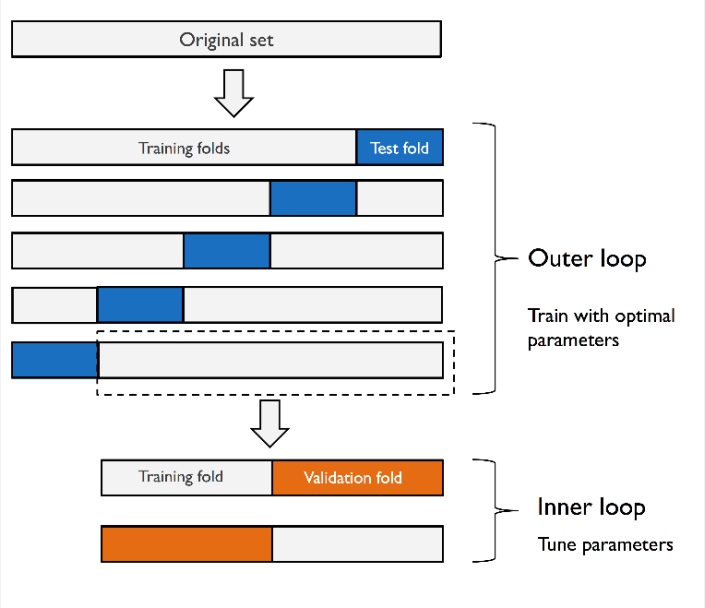
4-5-2 選擇一種評量成功的準則

問題種類	準則
平衡分類 (二元分類)	問題：每個類別具備同等的偏好度 準確度(accuracy) ROC AUC(area under the receiver operating characteristic curve)
類別不均 (class-imbalanced)	精密度(precision)和召回率(recall)
排名問題 多標籤分類	平均精度
不常見的問題	自定義

4-5-3 決定驗證程序

✓ 三種常用方式(先前介紹過)：

- 簡單拆分驗證(simple hold-out verification)
- K折驗證(K-fold validation)
- 多次洗牌的K折驗證(Iterated K-fold validation with shuffling)



4-5-4 準備資料

- ✓ 需將資料整理成可以餵給神經網路的格式
- ✓ 以深度學習神經網路為例，步驟：
  - 將資料轉換為張量
  - 張量壓縮到較小的值，如：[-1, 1]或[0, 1]
  - 如果不同的特徵採用不同範圍的值(異質資料)，則應對資料進行正規化
  - 特徵工程(尤其小資料的問題)

4-5-5 開發出基於基準的模型

- ✓ 基準模型(baseline) ➔ 陽春、射飛鏢的方法，如：亂猜、簡單加總平均等
- ✓ 統計功效(statistical power) ➔ 開發出能夠勝過基準能力baseline的小型模型
  - 統計功效模型的基礎假設：
    - ✓ 可以根據給定的輸入預測出結果
    - ✓ 假設收集的資料能提供足夠的資訊，讓機器學習到輸入與輸出間的關係
    - ✓ 如果嘗試很多次仍無法優於基準能力，要合理懷疑輸入資料可能沒有要的答案
- ✓ 三個建構模型的關鍵：
  - 啟動函數：將為神經網路建立輸出的形式
  - 損失函數：應該與嘗試解決的問題類型相匹配(否則會答非所問)
  - 優化器設定：將使用哪種優化器?學習率多少?

4-5-5 開發出基於基準的模型

- ✓ 為模型選擇正確的輸出層啟動函數和損失函數
- ✓ 下表為問題類型與建議的選項

問題類型	啟動函數	損失函數
Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

4-5-6 擴大規模：開發一個過度配適的模型

- ✓ 一旦模型超越基準(baseline)，問題會變成
  - 你的模型是否足夠強大？是否有足夠的layer和參數來正確模擬手上的問題?
    為了弄清楚需要多大的模型，必須開發一個過度配適的模型：
    - ✓ 添加更多的layer
    - ✓ 讓每一層更寬
    - ✓ 訓練更多週期
    - ✓ 當看到模型在驗證資料上的性能開始下降時，就已經發生過度配適了!

4-5-7 常規化模型並調整超參數

✓ 可以嘗試以下做法，反覆修改模型：

- 使用丟棄法dropout
- 嘗試不同的架構：添加或刪除layer
- 添加L1或L2常規化 regularization(也可以同時使用)
- 嘗試重作特徵工程：添加新特徵或刪除似乎沒用的特徵

總結

- ✓ 目前為止，應該知道的內容：
  - 定義手上的問題以及將用來訓練的資料
  - 針對要解決的問題，考慮如何評量成功與否? 要在驗證資料上監控哪些指標?
  - 確定驗證程序方式
    - ✓ 簡單拆分驗證?
    - ✓ K折驗證?
    - ✓ 應該使用哪部分的資料進行驗證?
  - 開發出比基準能力更好的第一個模型
  - 根據驗證資料的表現，對模型進行常規化並調整其超參數