



Chapter 6

神經網路的 優化與調教

6-1 過擬合 (overfitting) 與欠擬合 (underfitting) 問題

我們常聽到很多人在說這個模型的泛化能力好不好？那甚麼是泛化能力呢？所謂的泛化能力 (generalization ability) 是指一個機器學習算法對於沒有見過的樣本資料有很好的識別能力。也就是此算法在定義的範圍內都能夠使用。

這邊舉一個例子，人們從幼稚園至小學開始，通過不斷的學習，因此漸漸的就可以熟練的掌握加減法，但這是如何做到的呢？第一步可能大人先讓他們知道了有阿拉伯數字的存在，漸漸地開始會從一數到十，再來可能就開始教導他們先拿一顆糖果，再拿一顆糖果，這樣有多少顆這就是相加，然後吃掉了一顆糖果，這樣就是相減，漸漸的從個位運算到十位運算，再慢慢地推廣到了多位數的加減法運算。

對於機器學習算法也是如此，我們通過資料及撰寫的演算法讓機器知道基本知識後，之後算法通過自己的學習 (也就是訓練)，推廣至更多對於相似資料的識別能力，如果機器在不斷的測試中都能夠識別正確，那麼我們認為機器已經總結這類資料的共同特徵與相似處，但如果說機器只能辨識我們給定的資料，而沒辦法識別其他的資料 (例如他只能辨識我們給定的狗照片、但其他的狗照片無法辨識)，那麼我們認為機器只是死記硬背，並沒有學以致用的能力，這樣就是泛化能力非常的低，同時我們也把這種現象叫做這個算法「過擬合 (overfitting)」了。

另外還有一種狀況稱「欠擬合 (Underfitting)」，意思是我們訓練出來的機器算法，在識別資料時所給的答案值與真實答案之間有顯著的落差，造成這個機器算法沒有什麼用處。例如希望機器去學習甚麼是貓跟狗，結果最後你給機器不管是甚麼動物圖片，他都識別成貓跟狗，這個我們就稱為欠擬合 (Underfitting)，因為這個算法連貓狗的特徵都無法學習出來。

那甚麼情況下會產生過擬合與欠擬合的狀況呢？主要有兩個原因：「模型容量」與「訓練的資料多寡」。

● 6-1-1 模型容量

在機器學習的領域中，我們會說容量 (capacity) 是模型擬合複雜函數的能力。一種可以表現模型容量大小的指標為模型的假設空間 (Hypothesis Space) 大小，即模型經由輸入可以得到的映射集的範圍大小。假設空間如果越大，從假設空間中搜索出逼近真實資料的模型的函數也就越有可能；相反的，如果假設空間非常小，就很難從中找到逼近真實模型的函數。通常，越是複雜的模型其容量越高，但是容易造成過擬合。反之，越是簡單的模型其容量越低，卻容易導致欠擬合。這邊我們用一個範例來解釋上面的說明。假設考慮樣本資料點來自於五次多項式，為了讓資料模擬得更接近真實情況，因此這邊加了一些噪聲 (誤差)，其資料分布圖如下 (圖 6-1)。

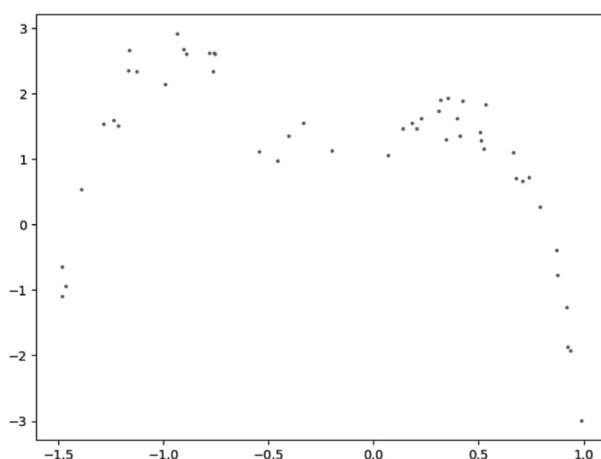


圖 6-1 資料散布情形

接下來我們利用多次多項式來找到一條能夠較好地逼近真實數據的分佈。首先一開始找一次多項式，即 $y = ax + b$ ，如圖 6-2，從圖中可以發現假設空間大於一次多項式的空間，因此接下來稍微增大假設空間，令假設空間為所有的 3 次多項式函數，即 $y = ax^3 + bx^2 + cx + d$ ，如圖 6-3，從圖中發現雖然他比一次多項式模型表達了更廣的假設空間，但很明顯的還是表現得不夠好。

因此再次加大假設空間，使得可以搜索的函數為 5 次多項式，即 $y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$ ，在此假設空間中，可以搜索到一個較好的函數，如圖 8.4 中 5 次多項式所示。

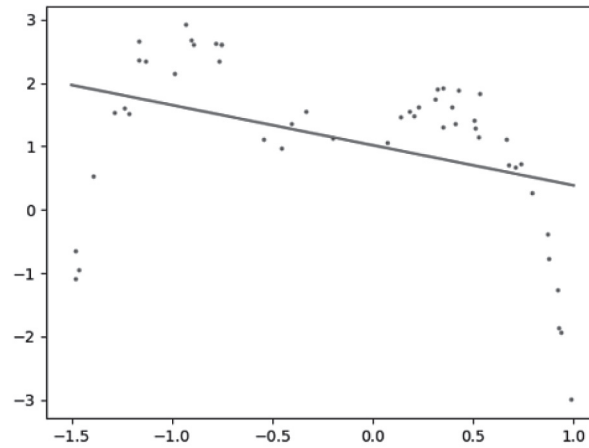


圖 6-2 一次函數容量表示圖

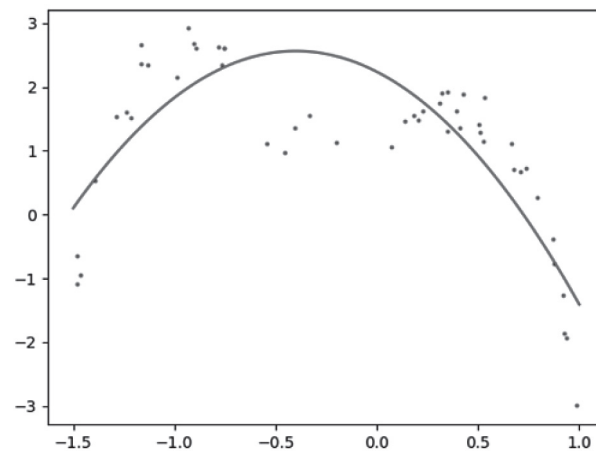


圖 6-3 三次函數容量表示圖

因此再次加大假設空間，使得可以搜索的函數為 5 次多項式，即 $y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$ ，在此假設空間中，可以搜索到一個較好的函數，如圖 6-4 中 5 次多項式所示。

從上面範例中可以發現，函數的假設空間如果太小則無法找出一個函數模型能逼近樣本，而造成欠擬合 (如圖 6-5(a)) 的情形，所以函數的假設空間越大，就越有可能找到一個函數能夠更好地逼近真實分佈的函數模型。但是這樣也有缺點，因為較大的假設空間會增加搜尋的難度與時間。此外，雖然較大的假設空間可以找到更好的函數模型，但由於樣本躁聲 (誤差) 的存在，較大的假設空間中也包含了表達能力過強的函數 (因為把訓練樣本的躁聲誤差跟著學習進來)，造成了過擬合 (如圖 6-5(b))，傷害了模型的泛化能力。因此怎麼挑選合適容量的學習模型來完成適度的擬合 (如圖 6-5(c)) 也是一個非常大的挑戰。

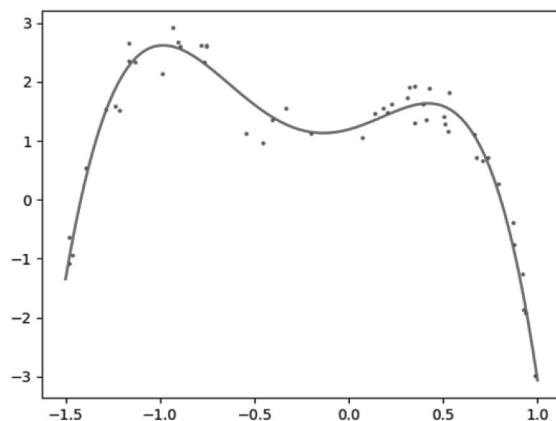


圖 6-4 三次函數容量表示圖

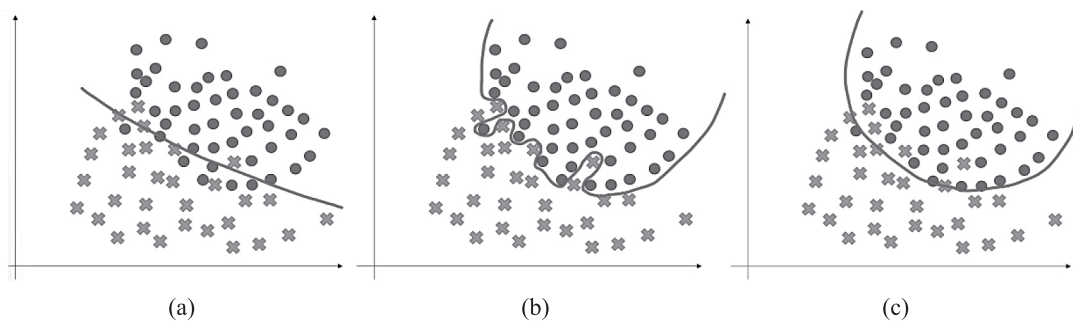


圖 6-5 (a) 欠擬合 (b) 過擬合 (c) 適度擬合

● 6-1-2 如何識別模型過擬合

對於機器學習或深度學習的模型而言，我們不僅要求它對訓練數據集有比較小的訓練誤差，同時也希望它可以對未知數據（也就是我們用的測試集）也有很好的擬合結果（泛化能力），這樣所產生的測試誤差被稱為泛化誤差（如圖 6-6 過擬合區的泛化誤差曲線）。而怎麼去測量泛化能力的好壞，最直觀的觀察與計算方式就是去分析模型的過擬合 (overfitting) 和欠擬合 (underfitting) 的情形。一般來說，訓練過程中模型複雜度與誤差會是如圖 6-6 所示的一個曲線圖。

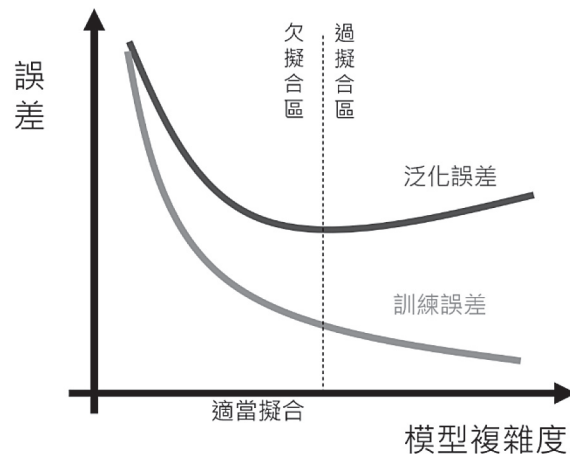


圖 6-6 模型複雜度與誤差關係圖

基本上，我們可以從模型的訓練與測試的準確度或者式誤差圖表的顯示，來識別出是否有過擬合或者式欠擬合的情形。從圖 6-6 可知，當給定一個訓練數據集，如果模型的複雜度過低，很容易出現欠擬合狀況；如果模型複雜度過高，很容易出現過擬合狀況。因此解決欠擬合和過擬合的一個辦法是針對數據集選擇合適複雜度的模型（中間那條虛線）。

此外，以訓練集跟驗證集的訓練週數與準確度來說，如圖 6-7。由於模型在訓練過程中對於訓練資料反覆學習，因此準確度會逐漸上升，但驗證資料集不會參與訓練，因此過了某個週期後，如圖 6-7 中間那條虛線，準確度會開始下降，而這個狀況也是代表模型開始產生過擬合，因此可以提早停止訓練避免過擬合。

由於現代深度神經網路能夠表達的模型能力非常強，但往往訓練數據集資料量不夠，導致無法學到更多資料的泛化特性，也因此很容易出現過擬合現象。那麼如何有效的檢測並減少過擬合現象呢？在實務上有許多有效方法可以來避免模型過擬合，以下便提出了一些常見的方法，並會在各節中詳述並實際演練。

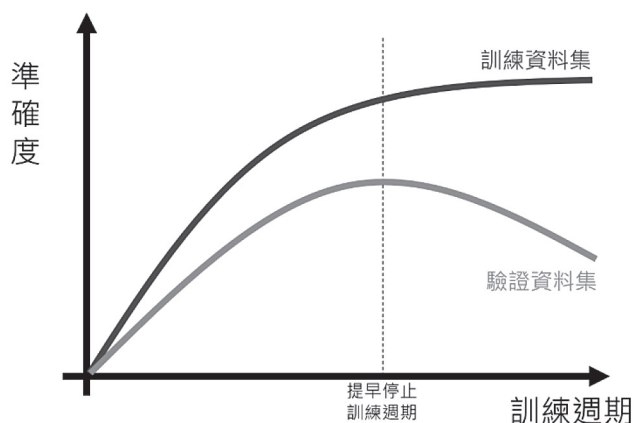


圖 6-7 訓練週期次數與準確度關係圖

6-2 數據集劃分

在訓練一個模型的時候，我們所搜集到資料並不能全部拿來當作訓練集做訓練，必須保留一些資料當作測試資料來評估模型最後訓練的結果好壞。一般最簡單的狀況下會把資料會被切分成測試集 (training) 跟訓練集 (test) 兩種。但由於有時候希望在訓練的過程中能夠有一些資料不參與訓練但是卻要拿來驗證模型參數的好壞，因此這時候會把資料劃分會分成三種，分別是：訓練集 (training)、驗證集 (validation) 和測試集 (test)。

這裡舉個學習英文語言的範例來描述三個數據集：

一、訓練集 (training)

舉例來說就是補習班上課學習英文，藉由不斷的學習英文讓自己對於英文的能力不斷提升。所以訓練集 (Training Set) 主要用在訓練階段，用於模型擬合，直接參與了模型參數調整的過程。

二、驗證集 (validation)

舉例來說就是補習班的模擬考，這時你會根據模擬考的成績去推斷你的英文哪邊還需要加強、或者調整學習方式重新學習。因此驗證集 (Validation Set) 在整個訓練的過程中是用於評估模型的擬合能力與超參數調整的依據。但是驗證集並非一定需要，不像訓練集和測試集。如果不需要調整超參數，就可以不使用驗證集。

三、測試集 (test)

就像是參加英文檢定考試，用來評估你最終英文學習結果。因此測試集是用來評估模型最終的泛化能力。這邊要注意的是測試集是爲了能評估模型真正的擬合能力，因此測試集不應該變爲調整參數或選擇特徵等依據。

訓練集、驗證集和測試集可以按著自己定義的比例來劃分，比如常見的 60% : 20% : 20% 的劃分，圖 6-8 演示了 Fashion-MNIST 圖像分類數據集的劃分示意圖。



圖 6-8 圖像依訓練集 - 驗證集 - 測試集分類

以下提供了三種數據集切割方式：

1. 方法一：利用 `sklearn` 內的 `train_test_split()` 函式，範例如下：

程式範例 | ch06_1

```
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

(train_image, train_label), (test_image, test_label) = \
    tf.keras.datasets.fashion_mnist.load_data()

print("train_image.shape :", train_image.shape)
print("train_label.shape :", train_label.shape)
```

程式輸出

```
train_image.shape : (60000, 28, 28)
train_label.shape : (60000,)
train_data.shape : (48000, 28, 28)
Valid_data.shape : (12000, 28, 28)
```

當分割完成後，訓練時可以在 `validation_data` 參數做設定，如下範例。

```
model = Sequential([
    layers.Flatten(input_shape=(28, 28)), # 將輸入資料從 28x28 攤平成 784
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax') # output 為 10 個 class
])
# model 每層定義好後需要經過 compile
# sparse_categorical_crossentropy 的標籤是 integer
```

```

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
history = model.fit(train_data, train_labelNew,
                  # 驗證集可以在這邊設定
                  validation_data=0.2, # 訓練集切出20%給驗證集
                  epochs=500,
                  batch_size=128, # 設定批次大小
                  shuffle=True) # 是否打散

```

2. 方法二：利用 `tf.split()` 函式

`tf.split()` 函式定義如下：

```
tf.split(value, num_or_size_splits, axis=0, num=None, name='split')
```

參數解釋：

- (1) `value`：準備切分的張量
- (2) `num_or_size_splits`：準備切成幾份
- (3) `axis`：準備在第幾個維度上進行切割

程式範例 | ch06_2

```

import tensorflow as tf

(train_image, train_label), (test_image, test_label) = \
    tf.keras.datasets.fashion_mnist.load_data()

print("train_image.shape :", train_image.shape)
print("train_image.shape :", train_label.shape)
train_data, valid_data, test_data = tf.split(train_image,
                                             [36000, 12000, 12000], axis=0)
print("train_data.shape :", train_data.shape)
print("valid_data.shape :", valid_data.shape)
print("test_data.shape :", test_data.shape)

```

程式輸出

```
train_data.shape : (36000, 28, 28)
valid_data.shape : (12000, 28, 28)
test_data.shape : (12000, 28, 28)
```

3. 方法三：利用 fit() 函數訓練時直接指定分割

```
history = model.fit(train_data, train_labelNew,
                    # 驗證集可以在這邊設定
                    validation_data=0.2, # 訓練集切出20%給驗證集
                    epochs=500,
                    batch_size=128, # 設定批次大小
                    shuffle=True) # 是否打散
```

6-3 提前停止 (Early stopping)

在圖 6-7 中可以看到，在訓練的過程中，即使在相同的網路設定下，隨著訓練的進行，可能觀測到不同的過擬合、欠擬合狀況。在訓練的前期，隨著訓練的進行，模型的訓練資料準確率和測試資料準確率都呈現增大的趨勢，此時並沒有出現過擬合現象；但在訓練後期，由於模型的實際容量發生改變，因此開始會觀察到了過擬合的現象，所以這時候就應該要停止訓練，Early stopping 是一種應用於機器學習、深度學習的提早停止訓練的技巧。在進行監督式學習的過程中，這樣的方式很有可能可以找到模型收斂時機點的方法。

當一個模型訓練太久時，模型就會發生所謂的 Overfitting(過擬合)，因為在訓練的過程中模型過度地去擬合它的訓練資料。當然，這個模型在我們的訓練資料上會表現得很好，可是在其他的資料、也就是所謂的測試資料上卻會顯得效果很差，也就是這個模型的泛化性不好。

在程式範例 ch06_3 中說明如何在 TensorFlow 2 中實現 early stopping。