

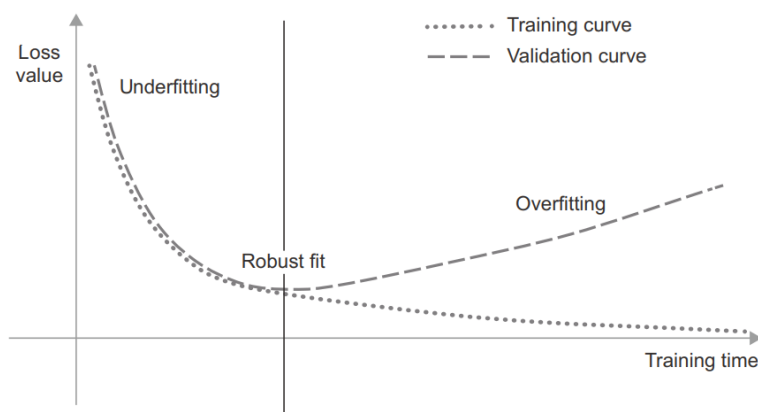
## CHƯƠNG 5: CÁC NGUYÊN TẮC CƠ BẢN CỦA HỌC MÁY

### 1. Mục tiêu của học máy

Mục tiêu cốt lõi của học máy không phải là đạt hiệu suất hoàn hảo trên dữ liệu huấn luyện, mà là khái quát hóa (generalization) – tức là khả năng dự đoán chính xác trên dữ liệu mới, chưa từng thấy trước đó. Một mô hình chỉ đơn thuần "ghi nhớ" dữ liệu huấn luyện mà không thể áp dụng kiến thức vào các tình huống mới thì không có giá trị thực tiễn. Thách thức lớn nhất của học máy nằm ở chỗ: làm thế nào để từ một tập hợp hữu hạn các ví dụ (dữ liệu huấn luyện), mô hình có thể suy ra các quy tắc chung áp dụng được cho các trường hợp chưa biết.

Khái quát hóa không phải là một quá trình ngẫu nhiên hay thần kỳ, mà là kết quả của việc thiết kế mô hình, dữ liệu và quy trình huấn luyện sao cho mô hình học được các mẫu thực sự (true patterns) thay vì các chi tiết ngẫu nhiên hay nhiễu.

#### 1.1 Hiện tượng thiếu khớp (Underfitting) và quá khớp (Overfitting)



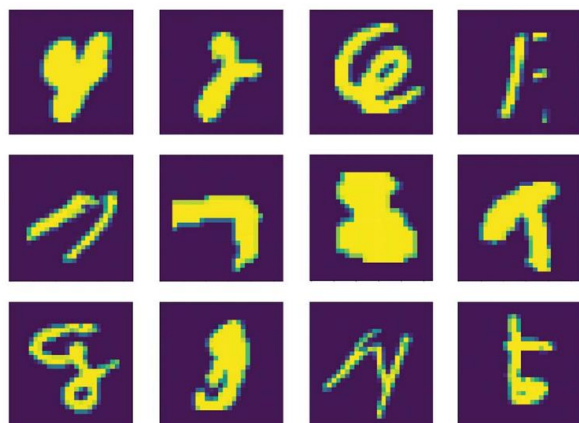
Đối với các mô hình đã thấy, hiệu suất trên dữ liệu kiểm tra giữ lại (held-out validation data) ban đầu sẽ cải thiện khi quá trình huấn luyện diễn ra và sau đó không thể tránh khỏi việc đạt đỉnh trước khi suy giảm. Mô hình này là một quy luật phổ biến. Chúng ta sẽ thấy điều này với bất kỳ loại mô hình nào và bất kỳ tập dữ liệu nào.

Ban đầu, tối ưu hóa và tổng quát hóa có mối tương quan: khi độ mất mát (loss) trên dữ liệu huấn luyện giảm, thì độ mất mát trên dữ liệu kiểm tra cũng giảm. Khi điều này xảy ra, mô hình của chúng ta được coi là underfit (chưa khớp đủ dữ liệu): vẫn còn chỗ để cải thiện, mạng chưa học được tất cả các mẫu quan trọng trong dữ liệu huấn luyện.

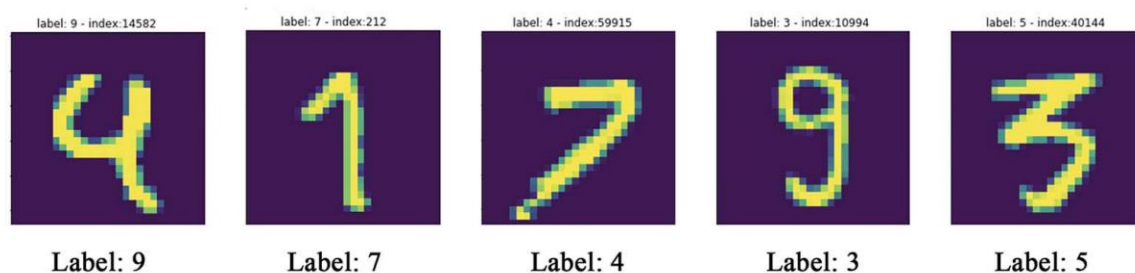
Nhưng sau một số lần lặp nhất định trên dữ liệu huấn luyện, quá trình tổng quát hóa ngừng cải thiện, các chỉ số xác thực bị chững lại và sau đó bắt đầu suy giảm: mô hình bắt đầu overfit (quá khớp). Nghĩa là nó bắt đầu học các mẫu chỉ xuất hiện trong dữ liệu huấn luyện nhưng lại không có ý nghĩa hoặc gây hiểu lầm khi áp dụng cho dữ liệu mới.

##### 1.1.1. Dữ liệu huấn luyện có nhiễu

Trong các tập dữ liệu thực tế, khá phổ biến khi một số đầu vào không hợp lệ. Ví dụ, trong tập dữ liệu MNIST, có thể có một hình ảnh hoàn toàn đen hoặc một thứ gì đó giống như trong hình dưới



Điều tệ hơn nữa là có những đầu vào hoàn toàn hợp lệ nhưng lại bị gán nhãn sai, như trong hình



**Figure 5.3** Mislabeled MNIST training samples

Nếu một mô hình cố gắng học theo những điểm ngoại lai này, hiệu suất tổng quát hóa của nó sẽ suy giảm. Ví dụ, một số 4 trông rất giống với số 4 bị gán nhãn sai trong hình có thể bị phân loại thành số 9.

Nhiều là các yếu tố ngẫu nhiên hoặc sai lệch trong dữ liệu, như nhãn bị gán sai, giá trị ngoại lai (outliers), hoặc dữ liệu không chính xác. Khi mô hình học từ dữ liệu nhiều, nó có thể nhầm lẫn nhiều với các mẫu thực sự.

### 1.1.2. Các đặc trưng mơ hồ

Không phải tất cả nhiễu dữ liệu đều đến từ sự không chính xác — ngay cả dữ liệu hoàn toàn sạch và được gán nhãn cẩn thận cũng có thể chứa nhiễu khi bài toán liên quan đến sự không chắc chắn và tính mơ hồ.

Trong các bài toán phân loại, thường xảy ra trường hợp một số vùng trong không gian đặc trưng đầu vào có thể liên quan đến nhiều lớp cùng một lúc. Ví dụ, giả sử chúng ta đang phát triển một mô hình nhận diện hình ảnh quả chuối và dự đoán xem nó chưa chín, đã chín hay đã hỏng. Các danh mục này không có ranh giới khách quan rõ ràng, vì

vậy cùng một bức ảnh có thể được một người gán nhãn là "chưa chín" trong khi người khác lại cho là "đã chín".

Tương tự, nhiều vấn đề có yếu tố ngẫu nhiên. Chúng ta có thể sử dụng dữ liệu áp suất khí quyển để dự đoán liệu ngày mai có mưa hay không, nhưng cùng một chỉ số áp suất có thể dẫn đến trời mưa hoặc trời quang đãng với một xác suất nhất định.

Một mô hình có thể bị quá khớp với dữ liệu xác suất này bằng cách trở nên quá tự tin khi xử lý các vùng đặc trưng mơ hồ, như trong hình 5.5. Một mô hình khớp tốt hơn sẽ bỏ qua các điểm dữ liệu riêng lẻ và nhìn vào bức tranh tổng thể.

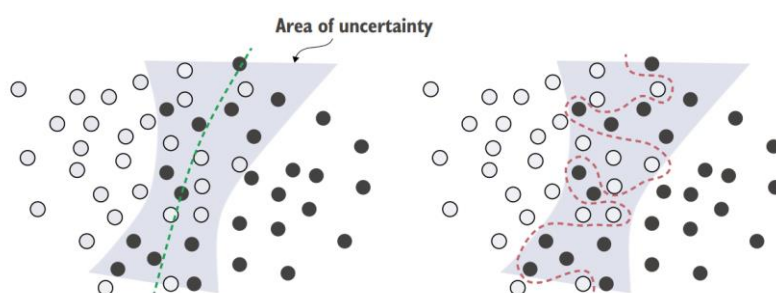


Figure 5.5 Robust fit vs. overfitting giving an ambiguous area of the feature space

Nếu dữ liệu chứa các đặc trưng không đủ rõ ràng để phân biệt giữa các lớp, mô hình sẽ gặp khó khăn trong việc đưa ra dự đoán chính xác trên dữ liệu mới. Ví dụ, trong bài toán phân loại ảnh chó và mèo, nếu chỉ có đặc trưng "màu lông" mà không có "hình dạng tai" hay "kích thước cơ thể", mô hình sẽ dễ nhầm lẫn vì màu lông có thể trùng lặp giữa hai loài.

Diễn giải sâu hơn: Đặc trưng mơ hồ làm tăng tính bất định (uncertainty) trong bài toán. Để khắc phục, cần dữ liệu phong phú hơn hoặc kỹ thuật đặc trưng (feature engineering) để bổ sung thông tin.

### 1.1.3. Các đặc trưng hiếm gặp và mối tương quan giả

Nếu một người chỉ từng thấy hai con mèo màu cam trong đời, và cả hai đều cực kỳ khó gần, chúng ta có thể kết luận rằng mèo cam nói chung có xu hướng khó gần. Đây chính là hiện tượng quá khớp (overfitting): nếu người đó tiếp xúc với nhiều con mèo hơn, bao gồm cả nhiều con mèo cam khác, chúng ta sẽ nhận ra rằng màu lông mèo không thực sự liên quan đến tính cách của chúng.

Tương tự, các mô hình học máy được huấn luyện trên tập dữ liệu có các giá trị đặc trưng hiếm rất dễ bị quá khớp. Trong một bài toán phân loại cảm xúc văn bản, nếu từ "cherimoya" (một loại trái cây có nguồn gốc từ dãy Andes) chỉ xuất hiện trong một văn bản trong tập huấn luyện, và văn bản đó có cảm xúc tiêu cực, thì một mô hình không được điều chuẩn (regularization) tốt có thể đặt trọng số rất cao cho từ này và luôn phân loại các văn bản mới có chứa "cherimoya" là tiêu cực, mặc dù trên thực tế, không có gì tiêu cực về trái cherimoya cả.

Quan trọng hơn, một giá trị đặc trưng không nhất thiết phải xuất hiện rất ít lần để tạo ra tương quan ngẫu tạo. Giả sử một từ nào đó xuất hiện trong 100 mẫu trong tập huấn luyện và được liên kết với cảm xúc tích cực 54% số lần, còn lại 46% là tiêu cực. Sự chênh lệch này có thể hoàn toàn là ngẫu nhiên về mặt thống kê, nhưng mô hình của chúng ta vẫn có khả năng học và tận dụng đặc trưng này để phân loại. Đây là một trong những nguyên nhân phổ biến nhất dẫn đến quá khớp.

### Ví dụ minh họa

Lấy tập dữ liệu MNIST làm ví dụ. Ta tạo một tập huấn luyện mới bằng cách nối thêm 784 chiều chứa nhiễu ngẫu nhiên vào 784 chiều dữ liệu gốc. Ngoài ra, cũng tạo một tập dữ liệu tương đương nhưng thay vì nhiễu, ta nối thêm 784 chiều toàn số 0. Việc thêm các đặc trưng vô nghĩa này không làm thay đổi nội dung thông tin của dữ liệu, vì ta chỉ đang thêm vào một thứ gì đó không liên quan. Nếu con người phân loại, kết quả sẽ không bị ảnh hưởng bởi sự thay đổi này.

### Thực hiện trên Python

```
from tensorflow.keras.datasets import mnist

import numpy as np

(train_images, train_labels), _ = mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255

# Thêm 784 chiều chứa nhiễu ngẫu nhiên
train_images_with_noise_channels = np.concatenate(
    [train_images, np.random.random((len(train_images), 784))], axis=1)

# Thêm 784 chiều toàn số 0
train_images_with_zeros_channels = np.concatenate(
    [train_images, np.zeros((len(train_images), 784))], axis=1)
```

### Huấn luyện mô hình

```
from tensorflow import keras
from tensorflow.keras import layers

def get_model():
    model = keras.Sequential([
        layers.Dense(512, activation="relu"),
        layers.Dense(10, activation="softmax")
    ])
    model.compile(optimizer="rmsprop",
                  loss="sparse_categorical_crossentropy",
                  metrics=["accuracy"])
    return model

# Huấn luyện với dữ liệu chứa nhiễu
model = get_model()
history_noise = model.fit(
    train_images_with_noise_channels, train_labels,
```

```

epochs=10,
batch_size=128,
validation_split=0.2)

# Huấn luyện với dữ liệu chứa toàn số 0
model = get_model()
history_zeros = model.fit(
    train_images_with_zeros_channels, train_labels,
    epochs=10,
    batch_size=128,
    validation_split=0.2)

```

## So sánh độ chính xác trên tập kiểm tra

```

import matplotlib.pyplot as plt

val_acc_noise = history_noise.history["val_accuracy"]
val_acc_zeros = history_zeros.history["val_accuracy"]
epochs = range(1, 11)

plt.plot(epochs, val_acc_noise, "b-", label="Validation accuracy with noise channels")
plt.plot(epochs, val_acc_zeros, "b--", label="Validation accuracy with zeros channels")

plt.title("Effect of noise channels on validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

```

## Kết quả

Mặc dù hai tập dữ liệu chứa cùng lượng thông tin, nhưng mô hình được huấn luyện với các đặc trưng chứa nhiễu có độ chính xác trên tập kiểm tra thấp hơn khoảng 1% so với mô hình dùng đặc trưng toàn số 0. Điều này xảy ra chỉ do ảnh hưởng của các tương quan ngẫu tạo. Càng thêm nhiều đặc trưng nhiễu, độ chính xác của mô hình càng suy giảm.

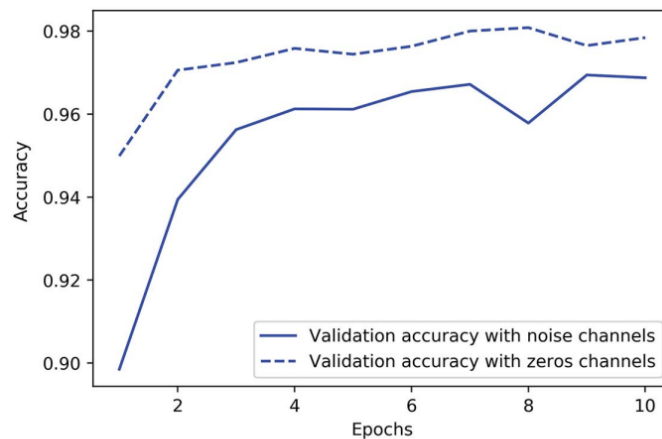


Figure 5.6 Effect of noise channels on validation accuracy

Do các đặc trưng nhiều làm mô hình bị quá khớp, nên khi không chắc chắn liệu một đặc trưng có thực sự hữu ích hay chỉ gây nhiễu, ta thường thực hiện chọn lọc đặc trưng (feature selection) trước khi huấn luyện.

Ví dụ, trong bài toán phân loại cảm xúc IMDB, ta chỉ giữ lại 10.000 từ phổ biến nhất, đó là một cách chọn lọc đặc trưng đơn giản.

Phương pháp chọn lọc đặc trưng điển hình là tính toán điểm hữu ích của từng đặc trưng, chẳng hạn như độ thông tin tương hỗ (mutual information) giữa đặc trưng và nhãn. Sau đó, chỉ giữ lại những đặc trưng có điểm hữu ích cao hơn một ngưỡng nhất định. Nếu áp dụng phương pháp này, ta có thể loại bỏ các chiều chứa nhiễu trong ví dụ MNIST trên, giúp giảm nguy cơ quá khớp.

Khi một đặc trưng chỉ xuất hiện trong một vài mẫu huấn luyện và tình cờ tương quan với nhãn, mô hình có thể "tin" rằng đặc trưng đó quan trọng, dù nó không đại diện cho thực tế. Đây là mối tương quan giả – một hiện tượng phổ biến trong học máy.

## **1.2 Bản chất của khái quát hóa trong học sâu**

### **1.2.1. Giả thuyết đa tạp**

Đầu vào của một bộ phân loại MNIST (trước khi tiền xử lý) là một mảng  $28 \times 28$  gồm các số nguyên từ 0 đến 255. Tổng số giá trị đầu vào có thể có do đó là  $256^{784}$ —lớn hơn số nguyên tử trong vũ trụ. Tuy nhiên, rất ít trong số các đầu vào này trông giống như các mẫu MNIST hợp lệ: các chữ số viết tay thực sự chỉ chiếm một phần rất nhỏ của không gian chứa tất cả các mảng uint8  $28 \times 28$  có thể có. Hơn nữa, không gian con này không chỉ đơn thuần là một tập hợp các điểm rải rác một cách ngẫu nhiên trong không gian cha—nó có cấu trúc cao.

Trước hết, không gian con của các chữ số viết tay hợp lệ có tính liên tục: nếu chúng ta lấy một mẫu và thay đổi nó một chút, nó vẫn sẽ được nhận diện là cùng một chữ số viết tay. Hơn nữa, tất cả các mẫu trong không gian con hợp lệ đều được kết nối bởi các đường dẫn mượt mà chạy qua không gian con. Điều này có nghĩa là nếu chúng ta lấy hai chữ số MNIST ngẫu nhiên A và B, sẽ tồn tại một chuỗi các hình ảnh “trung gian” biến đổi từ A thành B, sao cho hai chữ số liên tiếp rất gần nhau. Có thể sẽ có một số hình dạng mơ hồ gần ranh giới giữa hai lớp, nhưng ngay cả những hình dạng này vẫn trông rất giống chữ số.

Về mặt kỹ thuật, có thể nói rằng các chữ số viết tay tạo thành một đa tạp trong không gian của tất cả các mảng uint8  $28 \times 28$  có thể có. Đây là một thuật ngữ lớn, nhưng khái niệm này khá trực quan. Một “đa tạp” là một không gian con có số chiều thấp hơn trong một không gian cha, nhưng tại mỗi điểm, nó trông giống một không gian Euclid tuyến tính. Ví dụ:

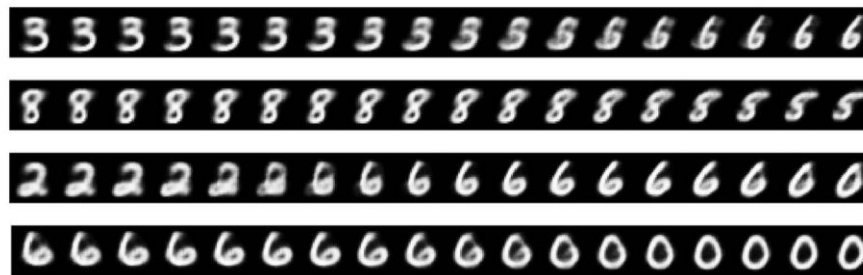


Figure 5.7 Different MNIST digits gradually morphing into one another, showing that the space of handwritten digits forms a “manifold.” This image was generated using code from chapter 12.

Nói một cách tổng quát hơn, giả thuyết đa tạp cho rằng tất cả dữ liệu tự nhiên đều nằm trên một đa tạp có số chiều thấp trong không gian chiều cao nơi nó được mã hóa. Đây là một tuyên bố khá mạnh mẽ về cấu trúc thông tin trong vũ trụ. Theo những gì chúng ta biết, nó là chính xác và nó là lý do tại sao deep learning hoạt động. Điều này đúng với chữ số MNIST, nhưng cũng đúng với khuôn mặt con người, hình dạng của cây cối, âm thanh giọng nói con người và thậm chí cả ngôn ngữ tự nhiên.

Giả thuyết đa tạp ngụ ý rằng:

Các mô hình machine learning chỉ cần khớp với các không gian con có số chiều thấp, có cấu trúc cao trong không gian đầu vào tiềm năng của chúng (tức là các đa tạp tiềm ẩn).

Bên trong một đa tạp như vậy, luôn có thể nội suy giữa hai đầu vào, tức là biến đổi một đầu vào thành một đầu vào khác thông qua một đường dẫn liên tục mà tất cả các điểm đều nằm trên đa tạp.

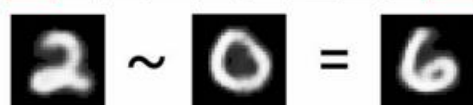
### 1.2.2. Nội suy là một nguồn gốc của tổng quát hóa

Nếu chúng ta làm việc với các điểm dữ liệu có thể được nội suy, chúng ta có thể bắt đầu hiểu được ý nghĩa của những điểm mà chúng ta chưa từng thấy trước đây bằng cách liên hệ chúng với các điểm khác nằm gần trên mặt đa tạp. Nói cách khác, chúng ta có thể hiểu toàn bộ không gian chỉ bằng cách sử dụng một mẫu của không gian đó. Chúng ta có thể dùng nội suy để điền vào những khoảng trống.

Lưu ý rằng nội suy trên mặt đa tạp tiềm ẩn khác với nội suy tuyến tính trong không gian gốc. Chẳng hạn, trung bình của các pixel giữa hai chữ số MNIST thường không phải là một chữ số hợp lệ.

Điều quan trọng là, trong khi học sâu đạt được sự tổng quát hóa thông qua nội suy trên một xấp xỉ đã học của mặt đa tạp dữ liệu, sẽ là sai lầm nếu cho rằng nội suy là tất cả những gì liên quan đến tổng quát hóa. Nó chỉ là phần nổi của tảng băng chìm. Nội suy chỉ có thể giúp chúng ta hiểu những thứ rất gần với những gì chúng ta đã thấy trước đây:





Manifold interpolation  
(intermediate point  
on the latent manifold)



Linear interpolation  
(average in the encoding space)

Nó cho phép tổng quát hóa cục bộ. Nhưng điều đáng chú ý là con người đối mặt với những điều mới mẻ cực đoan mọi lúc, và họ vẫn làm tốt. Chúng ta không cần phải được huấn luyện trước với vô số ví dụ về mọi tình huống mà chúng ta sẽ gặp phải trong đời. Mỗi ngày của chúng ta đều khác với bất kỳ ngày nào chúng ta đã trải qua trước đó, và khác với bất kỳ ngày nào mà bất kỳ ai đã trải qua kể từ buổi bình minh của loài người. Chúng ta có thể chuyển đổi giữa việc dành một tuần ở NYC, một tuần ở Thượng Hải, và một tuần ở Bangalore mà không cần hàng nghìn kiếp để học và diễn tập cho từng thành phố.

Con người có khả năng tổng quát hóa cực độ, điều này được hỗ trợ bởi các cơ chế nhận thức khác ngoài nội suy: sự trừu tượng, các mô hình biểu tượng của thế giới, lập luận, logic, tri thức thông thường, các tiên nghiệm bẩm sinh về thế giới—những gì chúng ta thường gọi là lý trí, trái ngược với trực giác và nhận diện mẫu. Trực giác và nhận diện mẫu phần lớn mang tính nội suy, nhưng lý trí thì không. Cả hai đều thiết yếu cho trí thông minh

### 1.2.3. Vậy tại sao học sâu hoạt động?

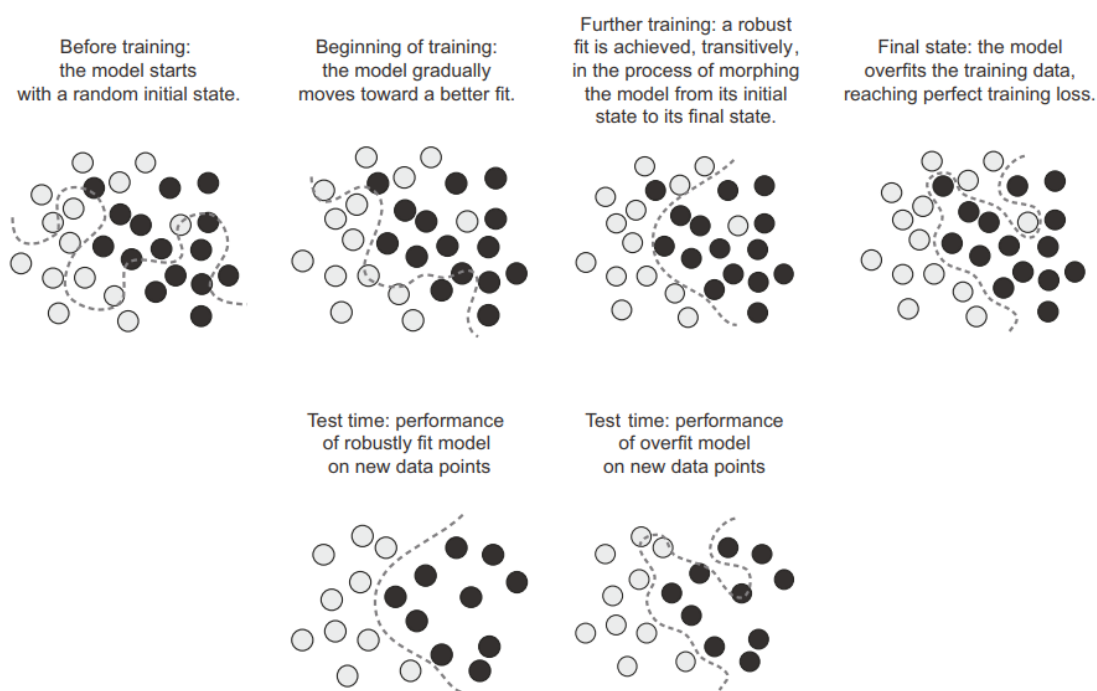
Mô hình học sâu thực chất là một đường cong có số chiều rất cao—một đường cong trơn tru và liên tục (với các ràng buộc bổ sung về cấu trúc xuất phát từ các tiên nghiệm của kiến trúc mô hình), bởi vì nó cần phải khả vi. Đường cong này được điều chỉnh sao cho khớp với các điểm dữ liệu thông qua quá trình lan truyền ngược bằng cách sử dụng gradient descent, diễn ra một cách trơn tru và dần dần. Về bản chất, học sâu là quá trình lấy một đường cong phức tạp lớn—một đa tạp—và điều chỉnh tham số của nó một cách từ từ cho đến khi nó khớp với dữ liệu huấn luyện.

Mô hình học sâu có số lượng tham số đủ lớn để có thể khớp với bất cứ thứ gì—thực tế, nếu chúng ta để mô hình huấn luyện quá lâu, nó sẽ chỉ đơn thuần ghi nhớ dữ liệu huấn luyện và không thể tổng quát hóa. Tuy nhiên, dữ liệu mà mô hình đang khớp không phải là những điểm rời rạc được phân bố rải rác trong không gian đầu vào. Dữ liệu của chúng ta hình thành một đa tạp có cấu trúc cao, có số chiều thấp trong không gian đầu vào—đây chính là giả thuyết đa tạp. Và vì việc khớp mô hình với dữ liệu diễn ra dần dần và



trơn tru trong suốt quá trình tối ưu hóa bằng gradient descent, sẽ có một giai đoạn trung gian trong quá trình huấn luyện mà mô hình gần như xấp xỉ đa tạp tự nhiên của dữ liệu

Di chuyển dọc theo đường cong mà mô hình đã học tại thời điểm đó sẽ gần như tương đương với việc di chuyển dọc theo đa tạp tiềm ẩn thực sự của dữ liệu—do đó, mô hình sẽ có khả năng hiểu các đầu vào chưa từng thấy trước đó thông qua nội suy giữa các đầu vào huấn luyện.



Bên cạnh thực tế hiển nhiên rằng chúng có đủ khả năng biểu diễn, có một số đặc điểm của mô hình học sâu giúp chúng đặc biệt phù hợp để học các đa tạp tiềm ẩn:

- + Mô hình học sâu thực hiện một ánh xạ trơn tru và liên tục từ đầu vào đến đầu ra. Điều này là bắt buộc vì mô hình phải khả vi (nếu không, không thể thực hiện gradient descent). Tính trơn tru này giúp chúng mô phỏng các đa tạp tiềm ẩn, vốn cũng tuân theo những tính chất tương tự.

- + Mô hình học sâu có xu hướng được cấu trúc theo cách phản ánh "hình dạng" của thông tin trong dữ liệu huấn luyện (thông qua các tiên nghiệm về kiến trúc). Nhìn chung, mạng nơ-ron sâu cấu trúc các biểu diễn học được theo cách phân cấp và mô-đun, tương tự như cách dữ liệu tự nhiên được tổ chức.

## 2. Đánh giá mô hình học máy

Chúng ta chỉ có thể kiểm soát những gì chúng ta quan sát được. Vì mục tiêu của chúng ta là phát triển các mô hình có thể tổng quát hóa thành công trên dữ liệu mới, nên điều quan trọng là phải đo lường đáng tin cậy khả năng tổng quát hóa của mô hình. Trong

phần này, tôi sẽ giới thiệu chính thức các cách khác nhau để đánh giá mô hình học máy. Chúng ta đã thấy hầu hết các phương pháp này trong chương trước.

## ***2.1. Training, validation, and test sets***

Việc đánh giá một mô hình luôn quy về việc chia tập dữ liệu có sẵn thành ba phần: tập huấn luyện (training set), tập kiểm tra (validation set), và tập đánh giá (test set). Chúng ta sẽ huấn luyện mô hình trên tập huấn luyện và đánh giá nó trên tập kiểm tra. Khi mô hình đã sẵn sàng để đưa vào thực tế, chúng ta sẽ kiểm tra lần cuối trên tập đánh giá, vốn được thiết kế để giống với dữ liệu thực tế nhất có thể. Sau đó, chúng ta có thể triển khai mô hình vào môi trường sản xuất.

Chúng ta có thể tự hỏi: Tại sao không chỉ chia thành hai tập: tập huấn luyện và tập đánh giá? Chúng ta có thể huấn luyện trên tập huấn luyện và đánh giá trực tiếp trên tập đánh giá – cách này đơn giản hơn nhiều!

Lý do là việc phát triển một mô hình luôn đi kèm với việc tinh chỉnh cấu hình của nó, chẳng hạn như chọn số lượng lớp hoặc kích thước của từng lớp (gọi là siêu tham số – hyperparameters để phân biệt với tham số – parameters, là trọng số của mạng). Chúng ta thực hiện việc tinh chỉnh này bằng cách sử dụng hiệu suất của mô hình trên tập kiểm tra làm tín hiệu phản hồi. Bản chất của quá trình này cũng là một hình thức học tập: tìm kiếm một cấu hình tối ưu trong không gian tham số

Hệ quả là, nếu chúng ta điều chỉnh mô hình dựa trên hiệu suất của nó trên tập kiểm tra, mô hình có thể quá khớp (overfitting) với tập kiểm tra, dù nó không trực tiếp được huấn luyện trên đó.

Trung tâm của hiện tượng này là khái niệm rò rỉ thông tin (information leak). Mỗi khi chúng ta tinh chỉnh một siêu tham số dựa trên hiệu suất của mô hình trên tập kiểm tra, một phần thông tin về tập kiểm tra sẽ bị "rò rỉ" vào mô hình. Nếu chúng ta chỉ làm điều này một lần với một tham số, lượng thông tin rò rỉ là không đáng kể, và tập kiểm tra vẫn có thể đáng tin cậy để đánh giá mô hình. Nhưng nếu chúng ta lặp lại quá trình này nhiều lần—chạy một thử nghiệm, đánh giá trên tập kiểm tra, rồi điều chỉnh mô hình—thì lượng thông tin rò rỉ ngày càng lớn.

Cuối cùng, chúng ta sẽ có một mô hình thể hiện rất tốt trên tập kiểm tra, vì nó đã được tối ưu hóa để làm điều đó. Tuy nhiên, điều chúng ta quan tâm là hiệu suất trên dữ liệu hoàn toàn mới, không phải trên tập kiểm tra. Vì vậy, chúng ta cần một tập dữ liệu hoàn toàn khác, chưa từng được mô hình tiếp cận, để đánh giá mô hình: tập đánh giá (test set).

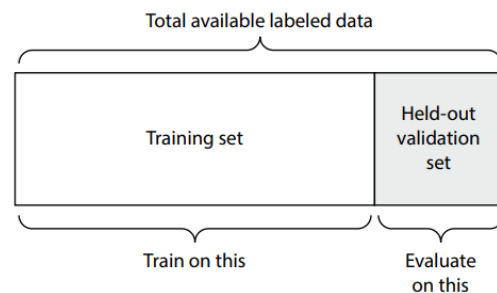
Mô hình của chúng ta không nên có bất kỳ thông tin nào về tập đánh giá, dù là gián tiếp. Nếu có bất kỳ yếu tố nào của mô hình được điều chỉnh dựa trên hiệu suất trên tập đánh giá, thì phép đo về khả năng tổng quát hóa của mô hình sẽ bị sai lệch.

Việc chia dữ liệu thành tập huấn luyện, kiểm tra và đánh giá có vẻ đơn giản, nhưng có một số phương pháp nâng cao có thể hữu ích khi dữ liệu bị hạn chế. Hãy cùng xem xét ba cách đánh giá cổ điển: phép chia ngẫu nhiên (holdout validation), kiểm tra chéo K-fold (K-fold validation), và kiểm tra chéo K-fold lặp lại với xáo trộn (iterated K-fold validation with shuffling). Chúng ta cũng sẽ thảo luận về việc sử dụng các đường cơ sở hợp lý (common-sense baselines) để đảm bảo rằng quá trình huấn luyện đang đi đúng hướng.

## 2.2. Xác thực giữ lại đơn giản

Chúng ta sẽ tách một phần dữ liệu của mình làm tập kiểm tra (test set), huấn luyện mô hình trên phần dữ liệu còn lại, và đánh giá mô hình trên tập kiểm tra. Như đã đề cập trước đó, để tránh rò rỉ thông tin, chúng ta không nên điều chỉnh mô hình dựa trên tập kiểm tra. Vì vậy, chúng ta cũng nên dành riêng một tập kiểm tra (validation set) để thực hiện quá trình tinh chỉnh.

Về mặt sơ đồ, phương pháp xác thực giữ lại có thể được minh họa như trong hình dưới.



**Figure 5.12** Simple holdout validation split

Đoạn mã sau đây mô tả cách triển khai đơn giản của phương pháp này:

```
num_validation_samples = 10000 # Số lượng mẫu dành cho tập kiểm tra
np.random.shuffle(data) # Xáo trộn dữ liệu để đảm bảo tính ngẫu nhiên

# Tách tập kiểm tra và tập huấn luyện
validation_data = data[:num_validation_samples]
training_data = data[num_validation_samples:]

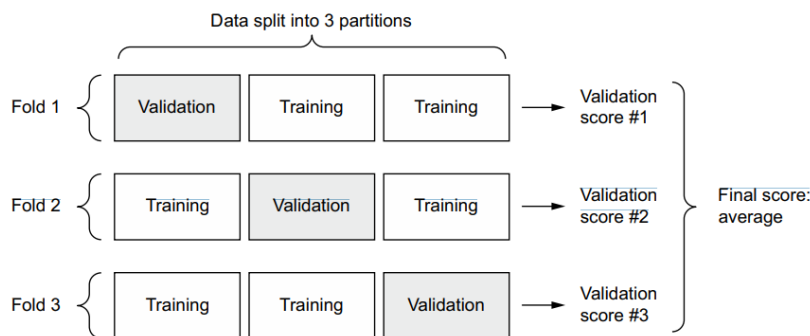
# Huấn luyện mô hình trên tập huấn luyện và kiểm tra trên tập kiểm tra
model = get_model()
model.fit(training_data, ...)
validation_score = model.evaluate(validation_data, ...)

# Sau khi chọn được mô hình tốt nhất, huấn luyện lại trên toàn bộ dữ liệu
model = get_model()
model.fit(np.concatenate([training_data, validation_data]), ...)
test_score = model.evaluate(test_data, ...)
```

**K-Fold Cross-Validation:** Với phương pháp này, chúng ta chia dữ liệu thành K phần có kích thước bằng nhau. Đối với mỗi phần i:

- + Huấn luyện mô hình trên K-1 phần còn lại.
- + Đánh giá mô hình trên phần i.
- + Tính điểm số cuối cùng bằng trung bình của K điểm số thu được.

Phương pháp này đặc biệt hữu ích khi hiệu suất của mô hình thay đổi đáng kể tùy thuộc vào cách phân chia tập huấn luyện và tập kiểm tra. Tuy nhiên, giống như holdout validation, phương pháp này không thay thế việc sử dụng một tập kiểm tra riêng biệt để đánh giá tổng quát hóa của mô hình.



**Figure 5.13 K-fold cross-validation with K=3**

```
k = 3 # Số lần chia dữ liệu
num_validation_samples = len(data) // k # Số mẫu trong mỗi tập kiểm tra
np.random.shuffle(data) # Xáo trộn dữ liệu để đảm bảo tính ngẫu nhiên

validation_scores = [] # Danh sách để lưu điểm số kiểm tra

for fold in range(k):
    # Chia dữ liệu thành tập kiểm tra và tập huấn luyện
    validation_data = data[num_validation_samples * fold :
num_validation_samples * (fold + 1)]
    training_data = np.concatenate([
        data[:num_validation_samples * fold],
        data[num_validation_samples * (fold + 1):]
    ])

    # Huấn luyện và đánh giá mô hình
    model = get_model()
    model.fit(training_data, ...)
    validation_score = model.evaluate(validation_data, ...)
    validation_scores.append(validation_score)

# Tính trung bình điểm số từ K lần huấn luyện
final_validation_score = np.average(validation_scores)

# Huấn luyện lại trên toàn bộ tập dữ liệu trước khi đánh giá trên tập kiểm
tra cuối cùng
model = get_model()
model.fit(data, ...)
test_score = model.evaluate(test_data, ...)
```

### 3. Cải thiện độ khớp của mô hình

Vì chúng ta không biết trước ranh giới nằm ở đâu, chúng ta phải vượt qua nó để tìm ra. Do đó, mục tiêu ban đầu khi bắt đầu giải quyết một vấn đề là xây dựng một mô hình có khả năng tổng quát hóa nhất định và có thể overfit. Khi có được một mô hình như vậy, chúng ta sẽ tập trung vào việc cải thiện khả năng tổng quát hóa bằng cách chống lại overfitting.

Có ba vấn đề phổ biến mà chúng ta có thể gặp phải ở giai đoạn này:

Huấn luyện không khởi động được: độ mất mát trong huấn luyện không giảm theo thời gian.

Huấn luyện bắt đầu tốt nhưng mô hình không tổng quát hóa đáng kể: chúng ta không thể vượt qua ngưỡng cơ sở hợp lý mà chúng ta đã đặt ra.

Độ mất mát trong huấn luyện và kiểm tra đều giảm theo thời gian, và chúng ta vượt qua ngưỡng cơ sở, nhưng chúng ta không thể overfit, điều này cho thấy mô hình vẫn đang underfitting.

Hãy cùng tìm hiểu cách giải quyết những vấn đề này để đạt được cột mốc quan trọng đầu tiên của một dự án máy học: có được một mô hình có khả năng tổng quát hóa nhất định (có thể đánh bại một baseline đơn giản) và có thể overfit.

#### 3.1. *Điều chỉnh các tham số chính của gradient descent*

Đôi khi quá trình huấn luyện không bắt đầu hoặc dừng lại quá sớm. Hàm mất mát bị mắc kẹt. Đây luôn là vấn đề có thể khắc phục được: hãy nhớ rằng chúng ta có thể huấn luyện một mô hình với dữ liệu ngẫu nhiên. Ngay cả khi không có gì về bài toán của chúng ta có ý nghĩa, chúng ta vẫn có thể huấn luyện được một thứ gì đó—dù chỉ là ghi nhớ dữ liệu huấn luyện.

Khi điều này xảy ra, nguyên nhân luôn liên quan đến việc cấu hình quá trình gradient descent: lựa chọn bộ tối ưu hóa, phân phối các giá trị khởi tạo trong trọng số của mô hình, tốc độ học (learning rate) hoặc kích thước batch. Tất cả các tham số này có sự phụ thuộc lẫn nhau, vì vậy thông thường chỉ cần điều chỉnh tốc độ học và kích thước batch trong khi giữ nguyên các tham số còn lại.

Hãy cùng xem một ví dụ cụ thể: chúng ta sẽ huấn luyện mô hình MNIST từ chương 2 với một tốc độ học không phù hợp, có giá trị là 1.

+ Huấn luyện mô hình MNIST với tốc độ học quá cao

```
(train_images, train_labels), _ = mnist.load_data()  
train_images = train_images.reshape((60000, 28 * 28))  
train_images = train_images.astype("float32") / 255
```

```

model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])

model.compile(optimizer=keras.optimizers.RMSprop(1.),
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

model.fit(train_images, train_labels,
          epochs=10,
          batch_size=128,
          validation_split=0.2)

```

Mô hình nhanh chóng đạt được độ chính xác trên tập huấn luyện và tập kiểm tra trong khoảng 30%–40%, nhưng không thể vượt qua mức đó. Hãy thử giảm tốc độ học xuống một giá trị hợp lý hơn là  $1e-2$ :

```

model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])

model.compile(optimizer=keras.optimizers.RMSprop(1e-2),
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

model.fit(train_images, train_labels,
          epochs=10,
          batch_size=128,
          validation_split=0.2)

```

Nếu chúng ta gặp tình huống tương tự, hãy thử:

Giảm hoặc tăng tốc độ học (learning rate):

Tốc độ học quá cao có thể khiến mô hình bỏ lỡ điểm hội tụ phù hợp, giống như ví dụ trên.

Tốc độ học quá thấp có thể làm cho quá trình huấn luyện quá chậm, khiến nó có vẻ như bị dừng lại.

Tăng kích thước batch: Một batch có nhiều mẫu hơn sẽ giúp giảm nhiễu trong gradient và cung cấp thông tin chính xác hơn, giúp quá trình học ổn định hơn.

Cuối cùng, chúng ta sẽ tìm được cấu hình phù hợp để bắt đầu quá trình huấn luyện thành công.

#### 4. Cải thiện khái quát hóa

Khi mô hình của chúng ta đã thể hiện khả năng tổng quát hóa và cũng có thể bị overfit, đã đến lúc tập trung vào việc tối đa hóa khả năng tổng quát hóa

#### **4.1. Xây dựng và quản lý tập dữ liệu**

Chúng ta đã biết rằng khả năng tổng quát hóa trong deep learning đến từ cấu trúc tiềm ẩn của dữ liệu. Nếu dữ liệu của chúng ta có thể được nội suy mượt mà giữa các mẫu, chúng ta có thể huấn luyện một mô hình học sâu có khả năng tổng quát hóa tốt. Nhưng nếu bài toán của chúng ta quá nhiều hoặc mang tính rời rạc, chẳng hạn như sắp xếp danh sách, deep learning sẽ không hữu ích. Deep learning thực chất chỉ là khớp đường cong (curve fitting), không phải phép màu.

Vì vậy, điều quan trọng là đảm bảo rằng chúng ta đang làm việc với một tập dữ liệu phù hợp. Việc đầu tư nhiều thời gian và tiền bạc vào thu thập dữ liệu chất lượng thường mang lại hiệu quả cao hơn so với việc cố gắng tinh chỉnh một mô hình phức tạp.

Những điều cần chú ý khi xây dựng tập dữ liệu:

Đảm bảo có đủ dữ liệu:

Chúng ta cần một tập dữ liệu có độ phủ dày đặc trên không gian đầu vào - đầu ra.

Càng có nhiều dữ liệu, mô hình càng có khả năng tổng quát tốt hơn.

Một số vấn đề tưởng chừng như không thể giải quyết lại trở nên khả thi khi có thêm dữ liệu.

Giảm thiểu lỗi gán nhãn:

Trực quan hóa dữ liệu đầu vào để kiểm tra các mẫu bất thường.

Kiểm tra lại nhãn để tránh sai sót trong việc gán nhãn dữ liệu.

Làm sạch dữ liệu và xử lý giá trị thiếu:

Đây là một bước quan trọng, sẽ được đề cập chi tiết hơn trong chương tiếp theo.

Lựa chọn đặc trưng (feature selection):

Nếu chúng ta có quá nhiều đặc trưng và không chắc đặc trưng nào thực sự quan trọng, hãy thực hiện feature selection để loại bỏ các đặc trưng không cần thiết.

#### **4.2. Kỹ thuật tạo đặc trưng**

Kỹ thuật tạo đặc trưng (Feature engineering) là quá trình sử dụng kiến thức của chúng ta về dữ liệu và thuật toán machine learning hiện có (trong trường hợp này là một mạng



nơ-ron) để giúp thuật toán hoạt động tốt hơn bằng cách áp dụng các phép biến đổi cứng (không được học) lên dữ liệu trước khi đưa vào mô hình. Trong nhiều trường hợp, không hợp lý khi kỳ vọng một mô hình machine learning có thể học từ dữ liệu hoàn toàn ngẫu nhiên. Dữ liệu cần phải được trình bày theo cách giúp mô hình thực hiện công việc dễ dàng hơn.

Hãy cùng xem một ví dụ trực quan. Giả sử chúng ta đang cố gắng phát triển một mô hình có thể nhận một hình ảnh của một chiếc đồng hồ làm đầu vào và xuất ra thời gian trong ngày



Raw data: pixel grid		
Better features: clock hands' coordinates	$\{x1: 0.7,$ $y1: 0.7\}$ $\{x2: 0.5,$ $y2: 0.0\}$	$\{x1: 0.0,$ $y2: 1.0\}$ $\{x2: -0.38,$ $y2: 0.32\}$
Even better features: angles of clock hands	theta1: 45 theta2: 0	theta1: 90 theta2: 140

Figure 5.16 Feature engineering for reading the time on a clock

Nếu chúng ta chọn sử dụng các pixel thô của hình ảnh làm dữ liệu đầu vào, chúng ta sẽ gặp một bài toán machine learning khó khăn. Chúng ta sẽ cần một mạng nơ-ron tích chập để giải quyết nó, và chúng ta sẽ phải tiêu tốn khá nhiều tài nguyên tính toán để huấn luyện mạng.

Nhưng nếu chúng ta đã hiểu vấn đề ở mức độ cao (chúng ta hiểu cách con người đọc giờ trên mặt đồng hồ), chúng ta có thể đưa ra những đặc trưng đầu vào tốt hơn cho thuật toán machine learning: chẳng hạn, rất dễ dàng để viết một đoạn mã Python gồm năm dòng để theo dõi các pixel đen của kim đồng hồ và xuất ra tọa độ (x, y) của đầu kim. Khi đó, một thuật toán machine learning đơn giản có thể học cách liên kết các tọa độ này với thời gian trong ngày.

Chúng ta còn có thể làm tốt hơn nữa: thực hiện một phép đổi hệ tọa độ, và biểu diễn tọa độ (x, y) dưới dạng tọa độ cực với tâm của hình ảnh làm gốc. Khi đó, đầu vào của chúng ta sẽ trở thành góc theta của từng kim đồng hồ. Ở bước này, đặc trưng của chúng ta làm cho bài toán trở nên đơn giản đến mức không cần machine learning nữa; một phép làm tròn đơn giản và tra cứu từ điển là đủ để xác định thời gian gần đúng trong ngày.

Đó chính là bản chất của kỹ thuật tạo đặc trưng: làm cho một bài toán trở nên dễ dàng hơn bằng cách biểu diễn nó theo một cách đơn giản hơn. Làm cho không gian tiềm ẩn trở nên mượt mà hơn, đơn giản hơn, có tổ chức hơn. Làm được điều đó thường yêu cầu sự hiểu biết sâu sắc về bài toán.

Trước khi deep learning ra đời, kỹ thuật tạo đặc trưng từng là phần quan trọng nhất trong quy trình machine learning, bởi vì các thuật toán nông (shallow algorithms) cổ điển

không có không gian giả thuyết đủ phong phú để tự học ra các đặc trưng hữu ích. Cách chúng ta trình bày dữ liệu cho thuật toán hoàn toàn mang tính quyết định đối với sự thành công của mô hình. Ví dụ, trước khi các mạng nơ-ron tích chập (CNN) trở nên phổ biến trong bài toán phân loại chữ số MNIST, các giải pháp thường dựa trên các đặc trưng được thiết lập cứng như số vòng lặp trong hình ảnh chữ số, chiều cao của từng chữ số trong ảnh, biểu đồ tần suất của giá trị pixel, v.v.

May mắn thay, deep learning hiện đại đã loại bỏ phần lớn nhu cầu về kỹ thuật tạo đặc trưng, vì các mạng nơ-ron có khả năng tự động trích xuất các đặc trưng hữu ích từ dữ liệu thô. Nhưng điều đó có nghĩa là chúng ta không cần quan tâm đến kỹ thuật tạo đặc trưng khi sử dụng deep learning nữa không? Câu trả lời là không, vì hai lý do sau:

Các đặc trưng tốt giúp chúng ta giải quyết vấn đề một cách gọn gàng hơn với ít tài nguyên hơn. Ví dụ, sẽ là phi lý nếu sử dụng một mạng nơ-ron tích chập để giải quyết bài toán đọc giờ trên mặt đồng hồ.

Các đặc trưng tốt cho phép chúng ta giải quyết vấn đề với ít dữ liệu hơn. Khả năng của các mô hình deep learning trong việc tự học đặc trưng phụ thuộc vào việc có một lượng lớn dữ liệu huấn luyện. Nếu chúng ta chỉ có một số ít mẫu dữ liệu, giá trị thông tin trong các đặc trưng của chúng trở nên cực kỳ quan trọng.

#### **4.3. Sử dụng *early stopping***

Trong deep learning, chúng ta luôn sử dụng các mô hình có số lượng tham số lớn hơn nhiều so với mức tối thiểu cần thiết để khớp với không gian tiềm ẩn của dữ liệu. Việc mô hình bị quá tham số (overparameterization) không phải là một vấn đề, vì chúng ta sẽ không bao giờ huấn luyện một mô hình deep learning đến mức hoàn toàn khớp với dữ liệu. Nếu làm vậy, mô hình sẽ không thể tổng quát hóa tốt. Chúng ta luôn phải dừng quá trình huấn luyện trước khi đạt đến mức mất mát huấn luyện (training loss) nhỏ nhất có thể.

Việc xác định chính xác thời điểm trong quá trình huấn luyện khi mô hình đạt đến mức khớp tổng quát hóa tốt nhất—ranh giới giữa mô hình chưa khớp (underfitting) và mô hình khớp quá mức (overfitting)—là một trong những cách hiệu quả nhất để cải thiện khả năng tổng quát hóa của mô hình.

Trong các ví dụ ở chương trước, chúng ta thường bắt đầu bằng cách huấn luyện mô hình lâu hơn mức cần thiết để xác định số epoch mang lại kết quả đánh giá tốt nhất trên tập validation. Sau đó, chúng ta sẽ huấn luyện lại một mô hình mới đúng với số epoch đó. Đây là một cách làm tiêu chuẩn, nhưng nó yêu cầu thực hiện công việc dư thừa, đôi khi tốn kém về tài nguyên.

Tất nhiên, chúng ta có thể lưu trạng thái mô hình sau mỗi epoch, và khi tìm ra epoch tốt nhất, chúng ta có thể sử dụng lại mô hình đã lưu gần nhất. Trong Keras, cách làm phổ biến để thực hiện điều này là sử dụng EarlyStopping callback, một kỹ thuật sẽ tự động

dừng quá trình huấn luyện ngay khi các chỉ số đánh giá trên tập validation không còn cải thiện nữa, đồng thời ghi nhớ trạng thái tốt nhất của mô hình

#### 4.4. Điều chuẩn (Regularizing) mô hình

Các kỹ thuật điều chuẩn là tập hợp các phương pháp giúp hạn chế khả năng của mô hình trong việc khớp hoàn hảo với dữ liệu huấn luyện. Mục tiêu của việc này là giúp mô hình hoạt động tốt hơn trên tập kiểm định (validation). Quá trình này được gọi là “điều chuẩn” (regularizing) mô hình, bởi vì nó giúp mô hình trở nên đơn giản hơn, “chuẩn” hơn, làm cho đường khớp trở nên mượt mà hơn, ít đặc thù với tập huấn luyện hơn, và từ đó có thể tổng quát hóa tốt hơn bằng cách mô phỏng chính xác hơn cấu trúc tiềm ẩn của dữ liệu.

Hãy nhớ rằng điều chuẩn mô hình phải luôn được hướng dẫn bởi một quy trình đánh giá chính xác. Chúng ta chỉ có thể đạt được khả năng tổng quát hóa nếu chúng ta có thể đo lường nó đúng cách.

Bây giờ, chúng ta sẽ cùng xem xét một số kỹ thuật điều chuẩn phổ biến nhất và áp dụng chúng vào thực tế để cải thiện mô hình phân loại phim

##### 4.4.1 Giảm kích thước mạng

Chúng ta đã học rằng một mô hình quá nhỏ sẽ không bị overfit. Cách đơn giản nhất để giảm overfitting là giảm kích thước mô hình (số lượng tham số có thể học được trong mô hình, được xác định bởi số lượng tầng và số lượng đơn vị trên mỗi tầng). Nếu mô hình có tài nguyên ghi nhớ hạn chế, nó sẽ không thể chỉ đơn giản ghi nhớ dữ liệu huấn luyện; do đó, để giảm thiểu hàm mất mát, nó sẽ phải học các biểu diễn nén có khả năng dự đoán mục tiêu—chính xác là loại biểu diễn mà chúng ta quan tâm. Đồng thời, hãy nhớ rằng mô hình của chúng ta cần có đủ tham số để không bị underfit: mô hình của chúng ta không nên bị thiếu tài nguyên ghi nhớ. Cần có sự cân bằng giữa quá nhiều và quá ít khả năng lưu trữ.

Thật không may, không có công thức kỳ diệu nào để xác định số tầng hoặc kích thước phù hợp cho từng tầng. Chúng ta cần đánh giá nhiều kiến trúc khác nhau (trên tập kiểm định, không phải trên tập kiểm tra) để tìm ra kích thước mô hình phù hợp cho dữ liệu của mình. Quy trình chung để tìm kích thước mô hình thích hợp là bắt đầu với ít tầng và ít tham số, sau đó tăng kích thước tầng hoặc thêm tầng mới cho đến khi nhận thấy lợi ích giảm dần đối với hàm mất mát trên tập kiểm định.

Hãy thử điều này trên mô hình phân loại đánh giá phim. Đoạn mã sau đây hiển thị mô hình ban đầu của chúng ta.

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), _ = imdb.load_data(num_words=10000)
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

```
train_data = vectorize_sequences(train_data)
model = keras.Sequential([ layers.Dense(16, activation="relu"),
layers.Dense(16, activation="relu"), layers.Dense(1, activation="sigmoid")
]) model.compile(optimizer="rmsprop", loss="binary_crossentropy",
metrics=["accuracy"]) history_original = model.fit(train_data,
train_labels, epochs=20, batch_size=512, validation_split=0.4)
```

Now let's try to replace it with this smaller model.

```
model = keras.Sequential([
layers.Dense(4, activation="relu"),
layers.Dense(4, activation="relu"),
layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
loss="binary_crossentropy",
metrics=["accuracy"])
history_smaller_model = model.fit(
train_data, train_labels,
epochs=20, batch_size=512, validation_split=0.4)
```

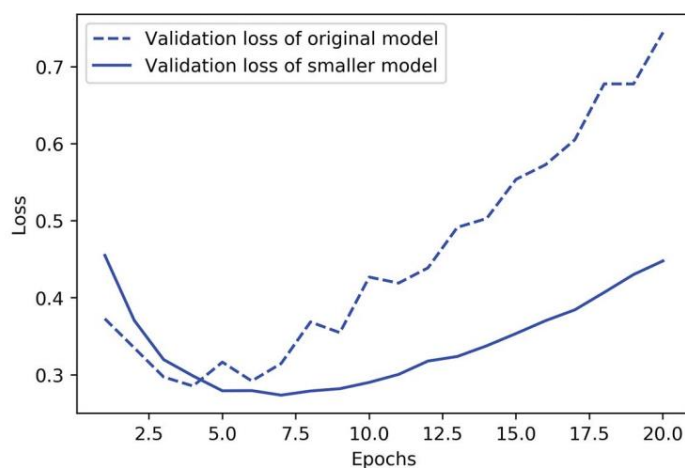


Figure 5.17 Original model vs. smaller model on IMDB review classification

Như chúng ta có thể thấy, mô hình nhỏ hơn bắt đầu bị overfitting muộn hơn so với mô hình tham chiếu (sau sáu epoch thay vì bốn), và hiệu suất của nó suy giảm chậm hơn khi bắt đầu overfitting.

Bây giờ, hãy thêm vào bài đánh giá của chúng ta một mô hình có dung lượng lớn hơn nhiều—vượt xa nhu cầu thực tế của bài toán. Mặc dù việc làm việc với các mô hình có số lượng tham số lớn hơn đáng kể so với những gì chúng cần học là điều phổ biến, nhưng vẫn có giới hạn về mức độ ghi nhớ quá mức. Chúng ta sẽ biết mô hình của mình quá lớn nếu nó bắt đầu overfitting ngay lập tức và nếu đường cong mất mát trên tập kiểm định trông gập ghềnh với độ biến động cao (mặc dù các số liệu kiểm định dao động mạnh cũng có thể là dấu hiệu của một quy trình kiểm định không đáng tin cậy, chẳng hạn như tập kiểm định quá nhỏ).

## Version of the model with higher capacity

```
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(512, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"])
history_larger_model = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

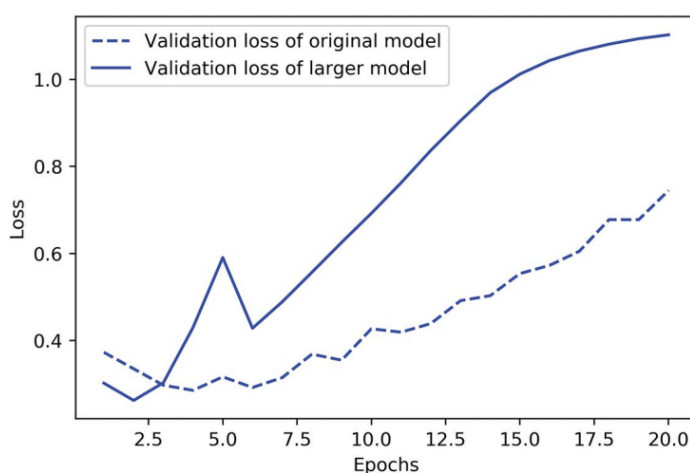


Figure 5.18 Original model vs. much larger model on IMDB review classification

Mô hình lớn hơn bắt đầu overfitting gần như ngay lập tức, chỉ sau một epoch, và nó bị overfitting nghiêm trọng hơn nhiều. Mất mát trên tập kiểm định của nó cũng nhiều hơn. Mô hình đạt mất mát trên tập huấn luyện gần bằng không rất nhanh. Mô hình có dung lượng càng lớn thì càng nhanh chóng có thể mô hình hóa dữ liệu huấn luyện (dẫn đến mất mát huấn luyện thấp), nhưng đồng thời cũng dễ bị overfitting hơn (dẫn đến sự chênh lệch lớn giữa mất mát trên tập huấn luyện và tập kiểm định).

### 4.4.2 Thêm regularization trọng số

Chúng ta có thể đã quen thuộc với nguyên tắc dao cạo Occam: giữa hai lời giải thích cho một vấn đề, lời giải thích đơn giản hơn—tức là giả định ít hơn—có nhiều khả năng đúng hơn. Nguyên tắc này cũng áp dụng cho các mô hình học bằng mạng nơ-ron: với cùng một tập dữ liệu huấn luyện và một kiến trúc mạng, có nhiều tập giá trị trọng số khác nhau (nhiều mô hình khác nhau) có thể giải thích dữ liệu. Mô hình đơn giản hơn ít có khả năng bị overfitting hơn so với mô hình phức tạp.

Một mô hình đơn giản trong ngữ cảnh này là một mô hình mà phân phối giá trị tham số có ít entropy hơn (hoặc một mô hình có ít tham số hơn, như chúng ta đã thấy trong phần

trước). Vì vậy, một cách phổ biến để giảm overfitting là đặt giới hạn về độ phức tạp của mô hình bằng cách ép các trọng số chỉ nhận giá trị nhỏ, giúp phân phối trọng số trở nên đều hơn. Điều này được gọi là regularization trọng số, được thực hiện bằng cách thêm vào hàm mất mát của mô hình một chi phí liên quan đến việc có trọng số lớn.

Chi phí này có hai dạng:

L1 regularization – Chi phí thêm vào tỷ lệ thuận với giá trị tuyệt đối của các hệ số trọng số (chuẩn L1 của trọng số).

L2 regularization – Chi phí thêm vào tỷ lệ thuận với bình phương giá trị của các hệ số trọng số (chuẩn L2 của trọng số). L2 regularization còn được gọi là weight decay trong bối cảnh mạng nơ-ron. Đừng để tên gọi khác nhau làm chúng ta nhầm lẫn: weight decay về mặt toán học giống hệt L2 regularization.

Trong Keras, regularization trọng số được thêm vào bằng cách truyền các đối tượng regularizer vào các lớp như các tham số từ khóa. Hãy thêm L2 regularization vào mô hình phân loại đánh giá phim ban đầu của chúng ta.

```
from tensorflow.keras import regularizers
model = keras.Sequential([
    layers.Dense(16,
        kernel_regularizer=regularizers.l2(0.002),
        activation="relu"),
    layers.Dense(16,
        kernel_regularizer=regularizers.l2(0.002),
        activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"])
from tensorflow.keras import regularizers
model = keras.Sequential([
    layers.Dense(16,
        kernel_regularizer=regularizers.l2(0.002),
        activation="relu"),
    layers.Dense(16,
        kernel_regularizer=regularizers.l2(0.002),
        activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"])
```

Trong đoạn mã trước, `l2(0.002)` có nghĩa là mỗi hệ số trong ma trận trọng số của lớp sẽ thêm vào tổng hàm mất mát của mô hình một giá trị bằng  $0.002 * \text{weight\_coefficient\_value}^2$ .

Lưu ý rằng hình phạt này chỉ được áp dụng trong quá trình huấn luyện, do đó, hàm mất mát của mô hình sẽ cao hơn nhiều trong quá trình huấn luyện so với khi kiểm tra.

Hình dưới minh họa tác động của hình phạt L2 regularization. Như chúng ta có thể thấy, mô hình sử dụng L2 regularization trở nên kháng overfitting tốt hơn nhiều so với mô hình tham chiếu, mặc dù cả hai mô hình có cùng số lượng tham số.

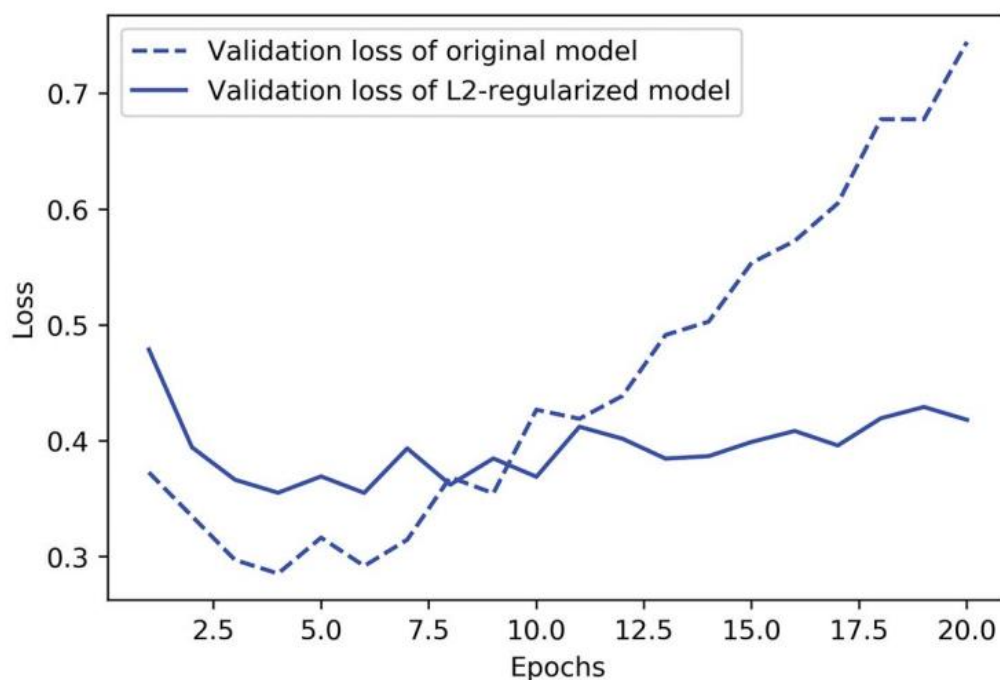


Figure 5.19 Effect of L2 weight regularization on validation loss

#### 4.4.3 Thêm dropout

Dropout là một trong những kỹ thuật điều chuẩn hiệu quả nhất và được sử dụng phổ biến nhất cho mạng nơ-ron; nó được phát triển bởi Geoff Hinton và các học trò của ông tại Đại học Toronto. Dropout, khi được áp dụng vào một tầng, sẽ ngẫu nhiên loại bỏ (đặt về 0) một số đặc trưng đầu ra của tầng đó trong quá trình huấn luyện.

Giả sử một tầng nhất định thông thường sẽ trả về một vector đầu ra [0.2, 0.5, 1.3, 0.8, 1.1] cho một mẫu đầu vào trong quá trình huấn luyện. Sau khi áp dụng dropout, vector này sẽ có một số giá trị bằng 0 được phân bố ngẫu nhiên, ví dụ: [0, 0.5, 1.3, 0, 1.1].

Tỷ lệ dropout là tỷ lệ phần trăm của các đặc trưng bị đặt về 0; thường nằm trong khoảng từ 0.2 đến 0.5.

Trong quá trình kiểm tra (test time), không có đơn vị nào bị dropout; thay vào đó, các giá trị đầu ra của tầng được giảm xuống theo một hệ số bằng với tỷ lệ dropout, nhằm cân bằng với việc có nhiều đơn vị hoạt động hơn so với khi huấn luyện.



Hãy xem xét một ma trận NumPy chứa đầu ra của một tầng, `layer_output`, có kích thước (`batch_size`, `features`). Trong quá trình huấn luyện, ta sẽ ngẫu nhiên đặt về 0 một phần giá trị trong ma trận:

```
layer_output *= np.random.randint(0, high=2, size=layer_output.shape)
```

Trong quá trình kiểm tra, ta giảm giá trị đầu ra theo tỷ lệ dropout. Ở đây, ta nhân với 0.5 (vì trước đó ta đã dropout 50% đơn vị):

```
layer_output *= 0.5
```

Lưu ý rằng quá trình này cũng có thể được triển khai bằng cách thực hiện cả hai thao tác ngay trong lúc huấn luyện và giữ nguyên đầu ra trong quá trình kiểm tra. Đây cũng là cách triển khai phổ biến trong thực tế:

```
layer_output *= np.random.randint(0, high=2, size=layer_output.shape)  
layer_output /= 0.5
```

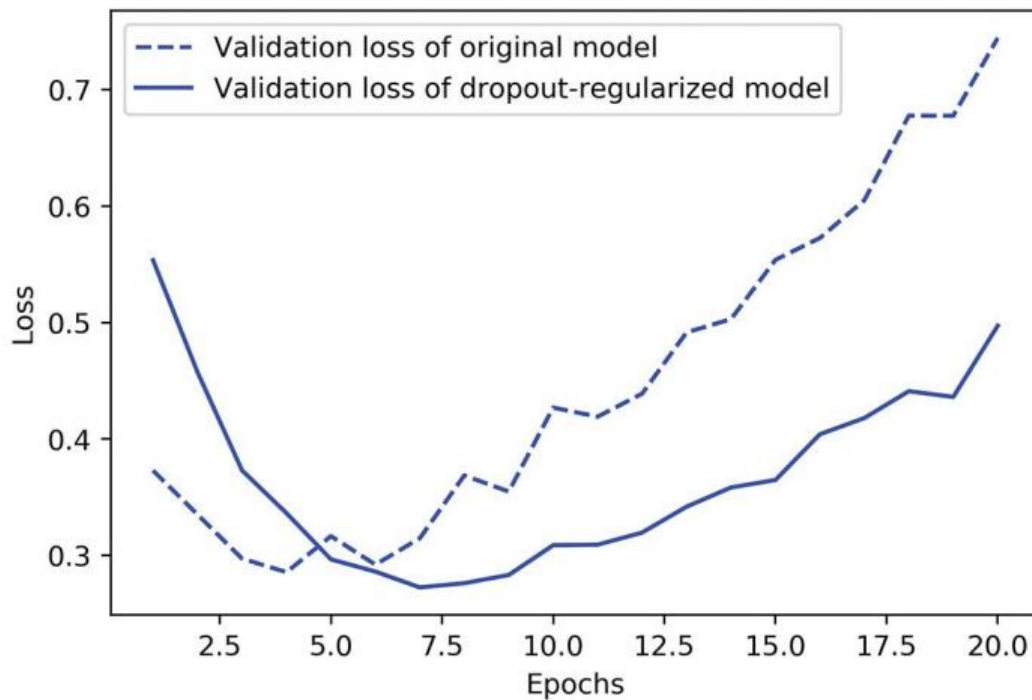
Tại sao nó lại giúp giảm overfitting?

Hinton cho biết ông lấy cảm hứng từ một số nguồn, trong đó có một cơ chế chống gian lận được sử dụng bởi các ngân hàng. Theo lời của ông:

"Tôi đến ngân hàng của mình. Các nhân viên giao dịch liên tục thay đổi, và tôi hỏi một người trong số họ lý do tại sao. Anh ta nói rằng anh không biết, nhưng họ thường xuyên bị luân chuyển. Tôi nghĩ điều này có thể là để ngăn chặn sự thông đồng giữa các nhân viên nhằm gian lận ngân hàng. Điều này khiến tôi nhận ra rằng nếu loại bỏ ngẫu nhiên một tập hợp khác nhau của các nơ-ron cho mỗi mẫu đầu vào, thì sẽ ngăn chặn được sự 'thông đồng' giữa chúng, từ đó giảm overfitting."

Ý tưởng cốt lõi là việc đưa nhiều vào giá trị đầu ra của một tầng có thể phá vỡ những mẫu hình tình cờ không có ý nghĩa (mà Hinton gọi là "sự thông đồng"), vốn sẽ được mô hình ghi nhớ nếu không có nhiều.

Trong Keras, chúng ta có thể thêm dropout vào mô hình thông qua lớp Dropout, lớp này sẽ được áp dụng cho đầu ra của tầng ngay trước nó. Hãy thêm hai tầng Dropout vào mô hình IMDB để xem chúng có hiệu quả thế nào trong việc giảm overfitting.



```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_dropout = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

Tóm lại, đây là những cách phổ biến nhất để tối đa hóa khả năng tổng quát hóa và ngăn chặn overfitting trong mạng nơ-ron:

- + Thu thập thêm dữ liệu huấn luyện hoặc cải thiện chất lượng dữ liệu huấn luyện.
- + Phát triển các đặc trưng (features) tốt hơn.
- + Giảm dung lượng (capacity) của mô hình.
- + Thêm regularization trọng số (weight regularization) cho các mô hình nhỏ hơn.
- + Thêm dropout.

## Tóm tắt chương 5

Mục đích của một mô hình học máy là tổng quát hóa: thực hiện chính xác trên các đầu vào chưa từng thấy trước đó. Điều này khó hơn so với tưởng tượng.

Một mạng nơ-ron sâu đạt được tổng quát hóa bằng cách học một mô hình tham số có thể nội suy thành công giữa các mẫu huấn luyện—một mô hình như vậy có thể được coi là đã học được “mặt đa tạp tiềm ẩn” của dữ liệu huấn luyện. Đây là lý do tại sao các mô hình học sâu chỉ có thể hiểu các đầu vào rất gần với những gì chúng đã thấy trong quá trình huấn luyện.

Vấn đề cốt lõi trong học máy là sự căng thẳng giữa tối ưu hóa và tổng quát hóa: để đạt được tổng quát hóa, trước tiên chúng ta phải đạt được sự khớp tốt với dữ liệu huấn luyện, nhưng cải thiện sự khớp này sẽ dần làm giảm tổng quát hóa. Mọi thực hành tốt nhất trong học sâu đều liên quan đến việc quản lý sự cân bằng này.

Khả năng tổng quát hóa của các mô hình học sâu đến từ việc chúng học cách xấp xỉ mặt đa tạp tiềm ẩn của dữ liệu, do đó có thể hiểu các đầu vào mới thông qua nội suy.

Việc đánh giá chính xác khả năng tổng quát hóa của mô hình trong quá trình phát triển là rất quan trọng. Chúng ta có thể sử dụng nhiều phương pháp đánh giá khác nhau, từ phương pháp chia tập giữ lại đơn giản đến kiểm định chéo K-fold và kiểm định chéo K-fold lặp với xáo trộn. Hãy luôn giữ một tập kiểm tra hoàn toàn tách biệt để đánh giá mô hình cuối cùng, vì thông tin từ tập xác thực có thể đã rò rỉ vào mô hình.

Khi bắt đầu làm việc với một mô hình, mục tiêu đầu tiên của chúng ta là đạt được một mô hình có một số khả năng tổng quát hóa và có thể overfit. Các thực hành tốt nhất để đạt được điều này bao gồm tinh chỉnh tốc độ học và kích thước batch, tận dụng các tiên nghiệm kiến trúc tốt hơn, tăng dung lượng mô hình hoặc đơn giản là huấn luyện lâu hơn.

Khi mô hình bắt đầu overfitting, mục tiêu của chúng ta chuyển sang cải thiện tổng quát hóa của mô hình thông qua việc điều chỉnh mô hình. Chúng ta có thể giảm dung lượng mô hình, thêm dropout hoặc điều chuẩn trọng số, và sử dụng early stopping. Và tất nhiên, một tập dữ liệu lớn hơn hoặc tốt hơn luôn là cách số một để giúp một mô hình tổng quát hóa tốt hơn.