**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN 1**

****

**BÁO CÁO THỰC TẬP CƠ SỞ**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn** | **PGS.TS TRẦN ĐÌNH QUẾ** |
| **Nhóm sinh viên thực hiện** |  |
| **B22DCCN014** | **ĐỖ TUẤN ANH** |
| **B22DCCN901** | **TRẦN ĐỨC VIỆT** |
| **B22DCCN072** | **THÀO A BẢY** |
| **B22DCCN407** | **ĐINH QUANG HƯNG** |
| **Nhóm bài tập** | **02** |
| **Nhóm** | **30** |

***Hà Nội – 2025***

**ĐẾ TÀI**

**TÌM HIỂU VỀ:**

* **CÁC NGUYÊN TẮC CƠ BẢN CỦA HỌC MÁY.**
* **QUY TRÌNH LÀM VIỆC CHUNG CỦA HỌC MÁY.**
* **LÀM VIỆC VỚI KERAS.**

**NỘI DUNG**

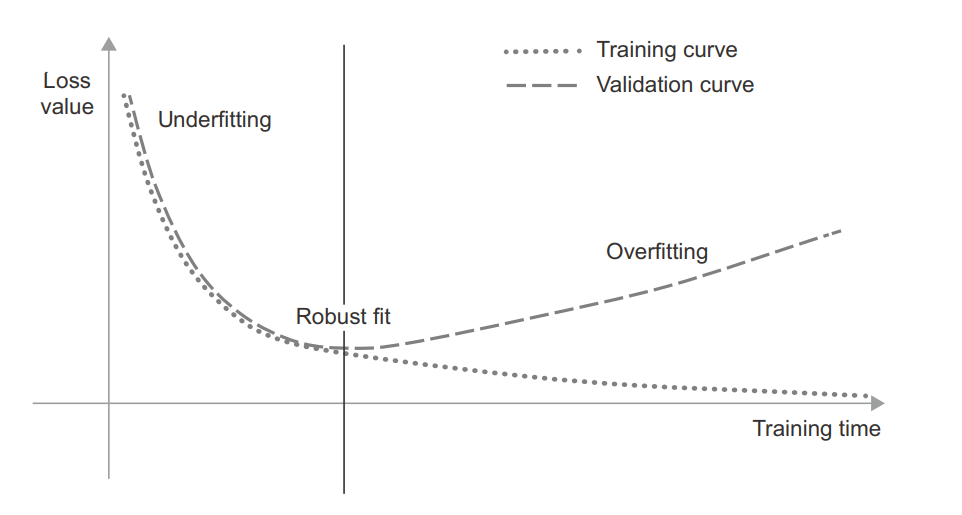
**PHẦN 1: CÁC NGUYÊN TẮC CƠ BẢN CỦA HỌC MÁY**

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. ***Hiện tượng thiếu khớp (Underfitting) và quá khớp(Overfitting)***



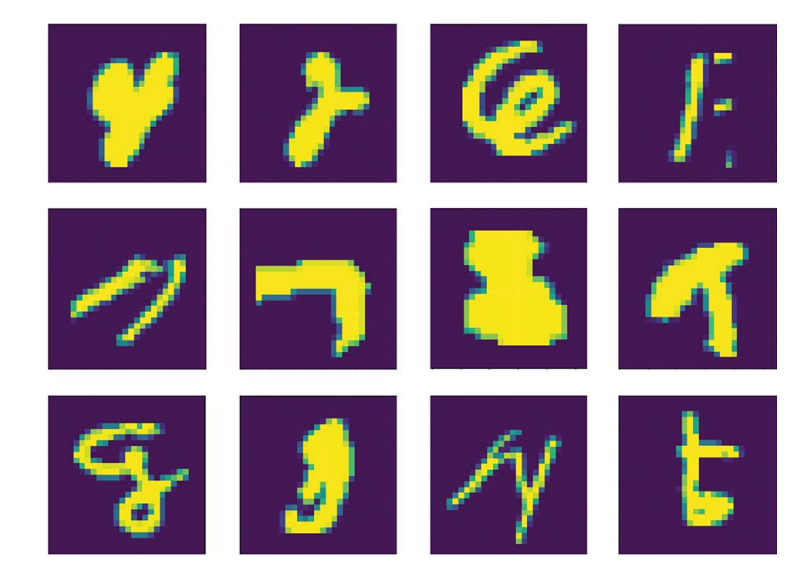
Đối với các mô hình bạn đã thấy trong chương trước, hiệu suất trên dữ liệu kiểm tra giữ lại (held-out validation data) ban đầu sẽ cải thiện khi quá trình huấn luyện diễn ra và sau đó không thể tránh khỏi việc đạt đỉnh trước khi suy giảm. Mô hình này là một quy luật phổ biến. Bạn sẽ thấy điều này với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

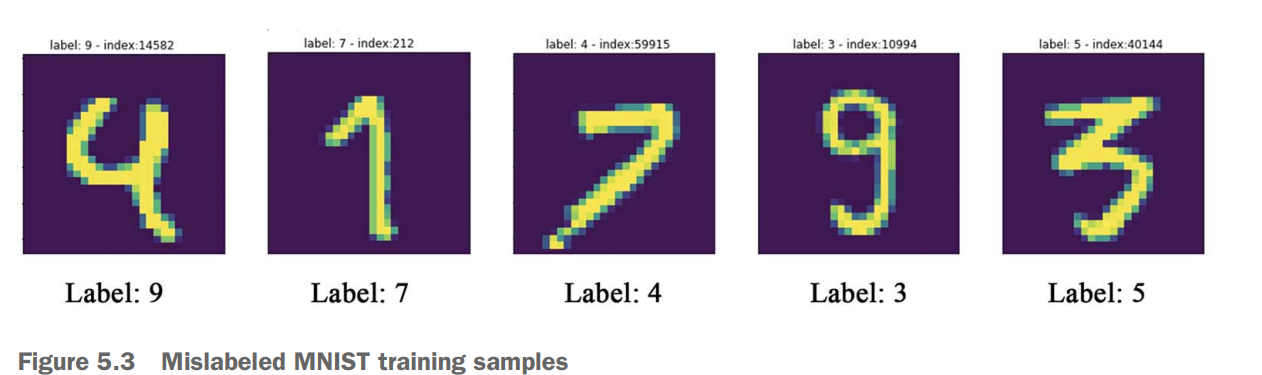
Nhưng sau một số lần lặp nhất định trên dữ liệu huấn luyện, quá trình tổng quát hóa ngừng cải thiện, các chỉ số xác thực bị chững lại và sau đó bắt đầu suy giảm: mô hình bắt đầu overfit (quá khớp). Nghĩa là nó bắt đầu học các mẫu chỉ xuất hiện trong dữ liệu huấn luyện nhưng lại không có ý nghĩa hoặc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



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.

Nhiễu 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 nhiễu, nó có thể nhầm lẫn nhiễu với các mẫu thực sự.

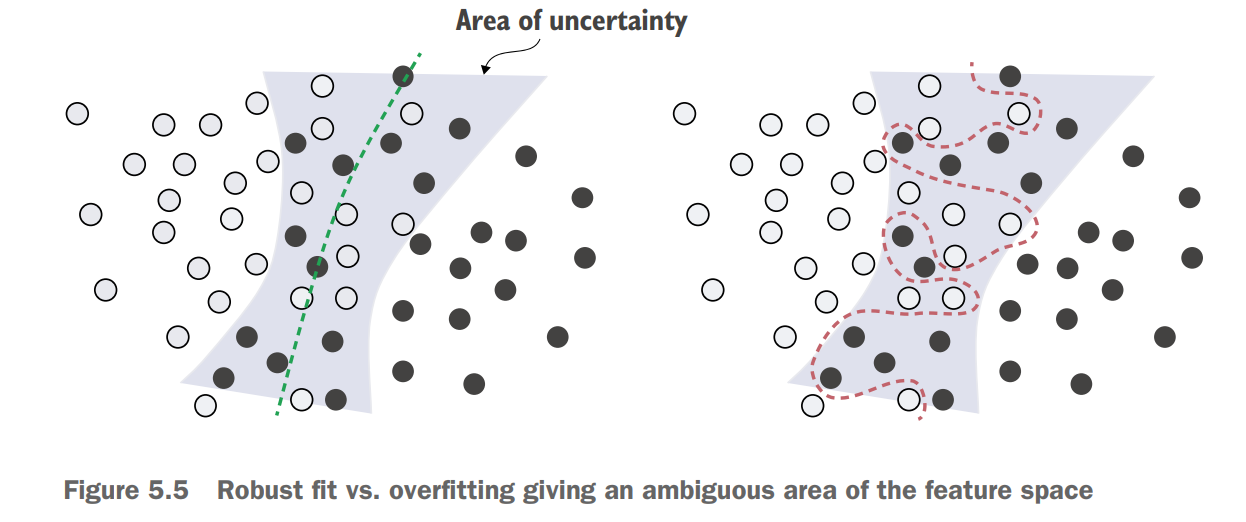
*1.1.2 Các đặc trưng mơ hồ*

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 bạn 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. Bạn 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.



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, bạn 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, bạn 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 ngụy 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 bạn 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

Hãy 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")

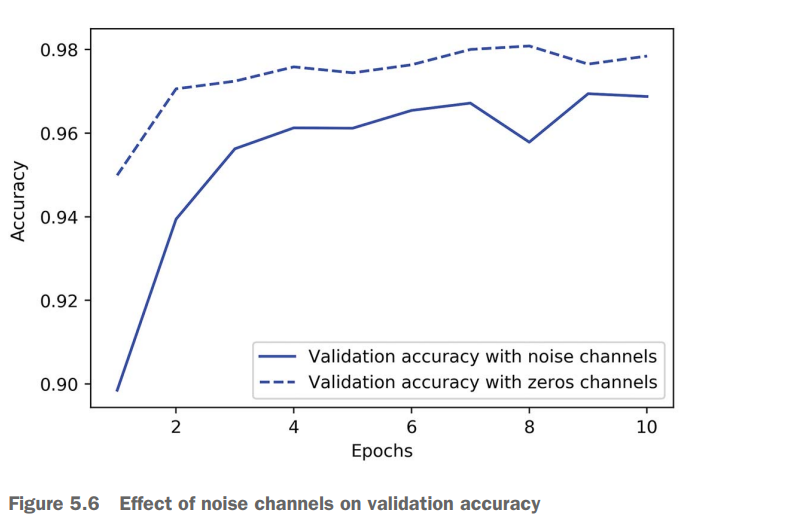
plt.ylabel("Accuracy")

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 ngụy tạo. Càng thêm nhiều đặc trưng nhiễu, độ chính xác của mô hình càng suy giảm.



Do các đặc trưng nhiễu 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.

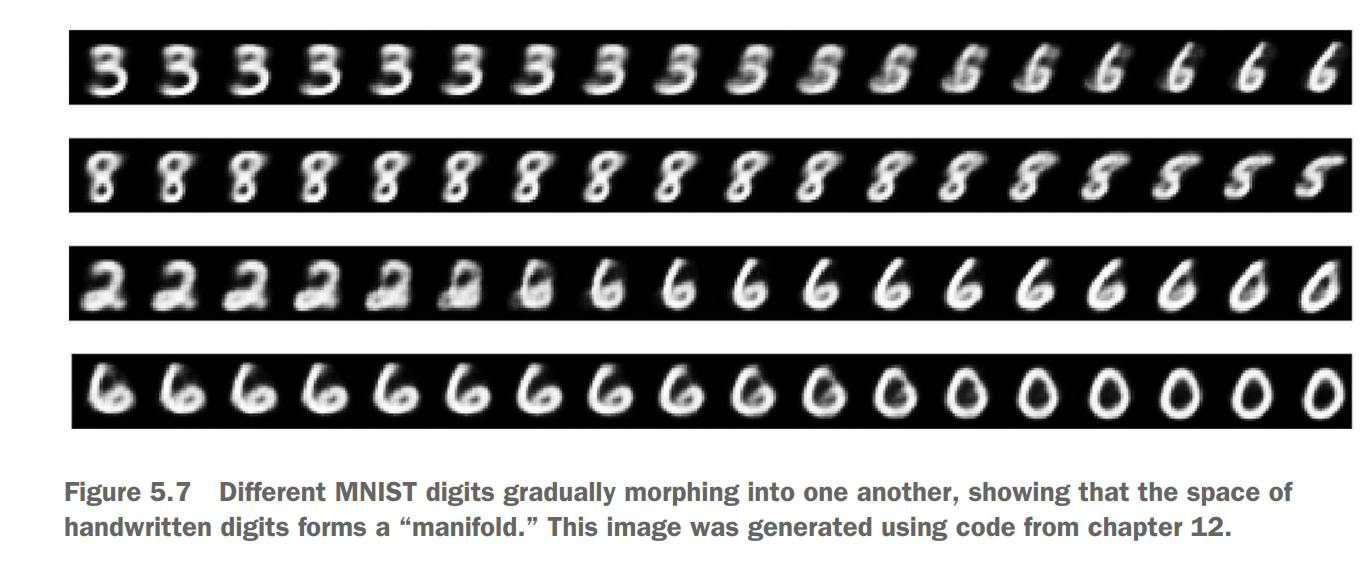
* 1. ***Bản chất của khái quát hóa trong học sâu***

*1.2.1 Giả thuyết đa tạp*

Đầu vào của một bộ phân loại MNIST (trước khi tiền xử lý) là một mảng 28 × 28 gồm các số nguyên từ 0 đến 255. Tổng số giá trị đầu vào có thể có do đó là 256^764—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 cha chứa tất cả các mảng uint8 28 × 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 bạn 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 bạn 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 × 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:



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).

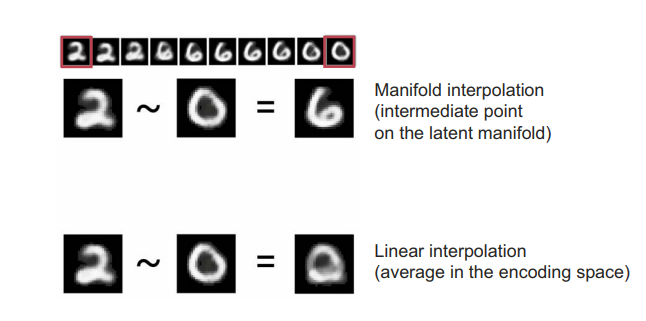
Bên trong một đa tạp như vậy, luôn có thể nội suy giữa hai đầu vào, tức là biến đổi một đầu vào thành một đầu vào khác thông qua một đường dẫn liên tục mà tất cả các điểm đều nằm trên đa tạp.

*1.2.2 Nội suy là một nguồn gốc của tổng quát hóa*

Nếu bạn làm việc với các điểm dữ liệu có thể được nội suy, bạn có thể bắt đầu hiểu được ý nghĩa của những điểm mà bạn chưa từng thấy trước đây bằng cách liên hệ chúng với các điểm khác nằm gần trên mặt đa tạp. Nói cách khác, bạn có thể hiểu toàn bộ không gian chỉ bằng cách sử dụng một mẫu của không gian đó. Bạn 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 bạn hiểu những thứ rất gần với những gì bạn đã thấy trước đây:



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. Bạn không cần phải được huấn luyện trước với vô số ví dụ về mọi tình huống mà bạn sẽ gặp phải trong đời. Mỗi ngày của bạn đều khác với bất kỳ ngày nào bạn 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. Bạn 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.

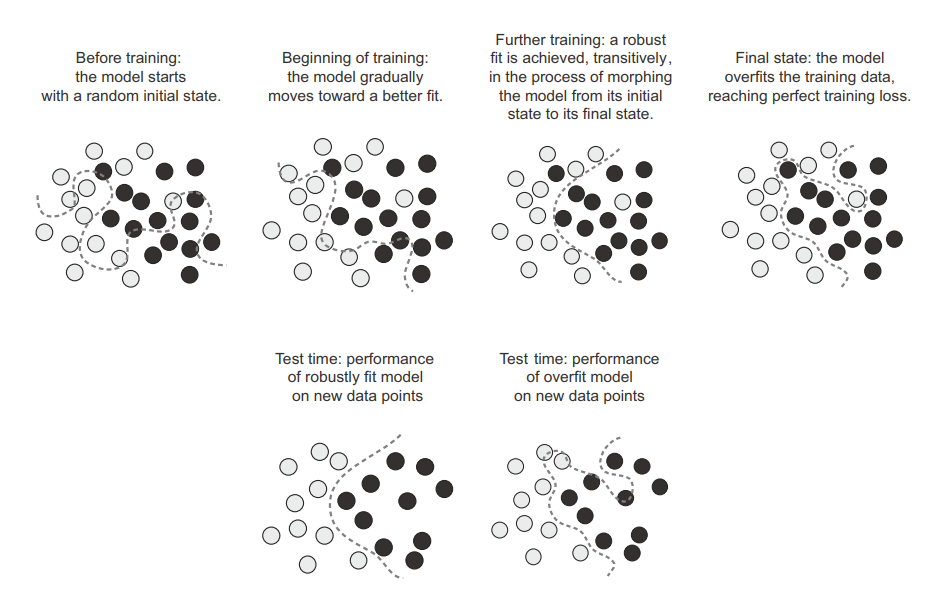
Con người có khả năng tổng quát hóa cực độ, điều này được hỗ trợ bởi các cơ chế nhận thức khác ngoài nội suy: sự trừu tượng, các mô hình biểu tượng của thế giới, lập luận, logic, tri thức thông thường, các tiên nghiệm bẩm sinh về thế giới—những gì chúng ta thường gọi là lý trí, trái ngược với trực giác 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

*1.2.3Tại 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 bạn 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 bạn 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). Điều này đặc biệt đúng với các mô hình xử lý ảnh (được thảo luận trong chương 8 và 9) và mô hình xử lý chuỗi (chương 10). 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.

1. **Đánh giá mô hình học máy**

Bạn chỉ có thể kiểm soát những gì bạn quan sát được. Vì mục tiêu của bạn 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. Bạn 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). Bạn 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, bạn 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, bạn có thể triển khai mô hình vào môi trường sản xuất.

Bạn có thể tự hỏi: Tại sao không chỉ chia thành hai tập: tập huấn luyện và tập đánh giá? Bạn 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). Bạn 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 bạn 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 bạn 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 bạn 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 bạn 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, bạn 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 bạn quan tâm là hiệu suất trên dữ liệu hoàn toàn mới, không phải trên tập kiểm tra. Vì vậy, bạn 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).

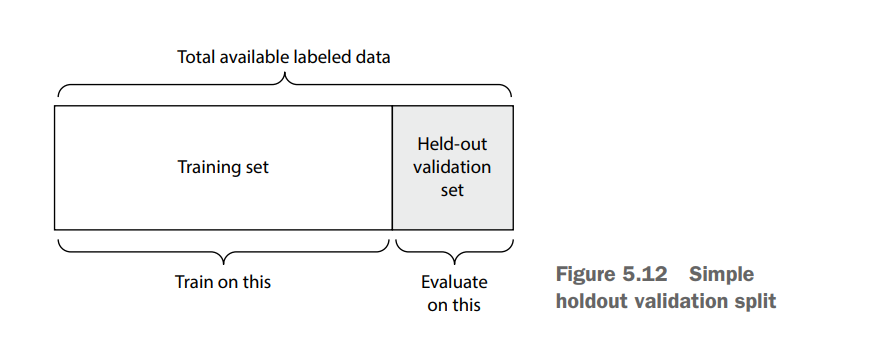
Mô hình của bạn không nên có bất kỳ thông tin nào về tập đánh giá, dù là gián tiếp. Nếu có bất kỳ yếu tố nào của mô hình được điều chỉnh dựa trên hiệu suất trên tập đánh giá, thì phép đo về khả năng tổng quát hóa của mô hình sẽ bị sai lệch.

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**

Bạn 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, bạn không nên điều chỉnh mô hình dựa trên tập kiểm tra. Vì vậy, bạn 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 5.12**.



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, ...)

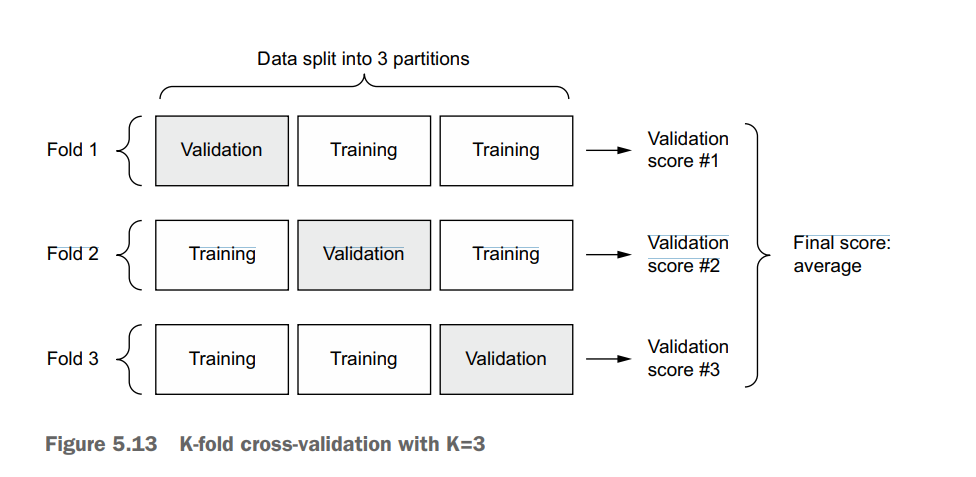
K-Fold Cross-Validation: Với phương pháp này, bạn chia dữ liệu thành K phần có kích thước bằng nhau. Đối với mỗi phần i:

+ 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.



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, ...)

1. **Cải thiện độ khớp của mô hình**

Vì bạn không biết trước ranh giới nằm ở đâu, bạn 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, bạn sẽ tập trung vào việc cải thiện khả năng tổng quát hóa bằng cách chống lại overfitting.

Có ba vấn đề phổ biến mà bạn 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: bạn không thể vượt qua ngưỡng cơ sở hợp lý mà bạn đã đặt ra.

Độ mất mát trong huấn luyện và kiểm tra đều giảm theo thời gian, và bạn vượt qua ngưỡng cơ sở, nhưng bạn 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 bạn có thể huấn luyện một mô hình với dữ liệu ngẫu nhiên. Ngay cả khi không có gì về bài toán của bạn có ý nghĩa, bạn 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 bạn 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, bạn sẽ tìm được cấu hình phù hợp để bắt đầu quá trình huấn luyện thành công.

1. **Cải thiện khái quát hóa**

Khi mô hình của bạn 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**

Bạn đã biết rằng khả năng tổng quát hóa trong deep learning đến từ cấu trúc tiềm ẩn của dữ liệu. Nếu dữ liệu của bạn có thể được nội suy mượt mà giữa các mẫu, bạn có thể huấn luyện một mô hình học sâu có khả năng tổng quát hóa tốt. Nhưng nếu bài toán của bạn quá nhiễu 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 bạn 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:

Bạn 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:

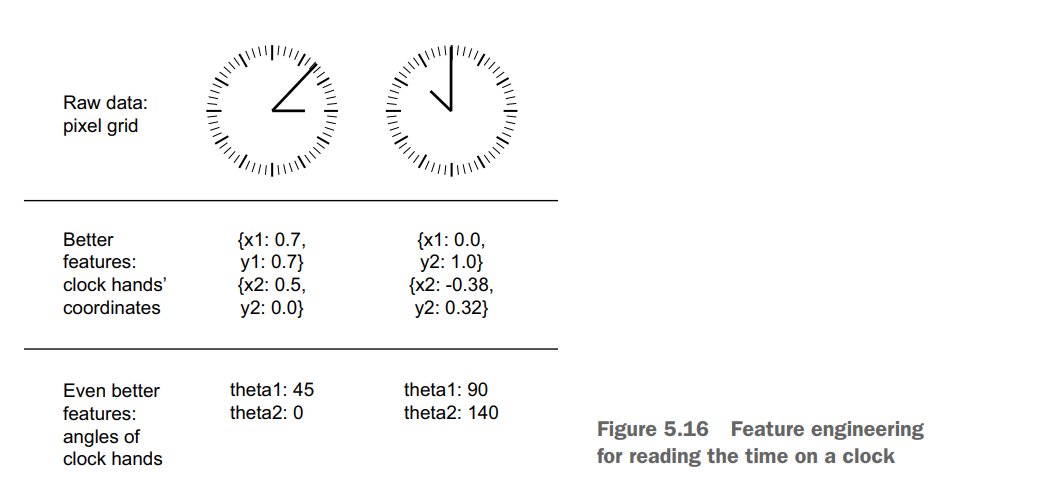
Đây là một bước quan trọng, sẽ được đề cập chi tiết hơn trong chương tiếp theo.

Lựa chọn đặc trưng (feature selection):

Nếu bạn 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 bạn 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 bạn 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 (xem hình 5.16). 

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

Nhưng nếu bạn đã hiểu vấn đề ở mức độ cao (bạn hiểu cách con người đọc giờ trên mặt đồng hồ), bạn 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.

Bạn 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 bạn sẽ trở thành góc theta của từng kim đồng hồ. Ở bước này, đặc trưng của bạn 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 bạn 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 bạn 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 bạn 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 bạn 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 bạn 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 bạn 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. Bạn 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, bạn có thể lưu trạng thái mô hình sau mỗi epoch, và khi tìm ra epoch tốt nhất, bạn 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. Bạn sẽ học cách sử dụng callbacks trong chương 7.

**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. Bạn chỉ có thể đạt được khả năng tổng quát hóa nếu bạn 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*

Bạn 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 bạn cần có đủ tham số để không bị underfit: mô hình của bạn 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. Bạn 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",

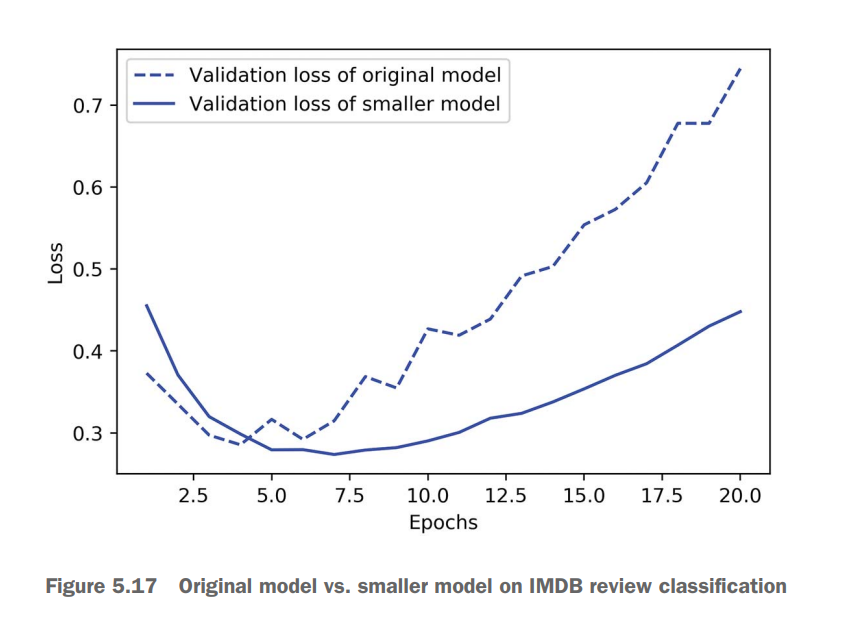
loss="binary\_crossentropy",

metrics=["accuracy"])

history\_smaller\_model = model.fit(

train\_data, train\_labels,

epochs=20, batch\_size=512, validation\_split=0.4)



Như bạn 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. Bạn 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).

Listing 5.12 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",

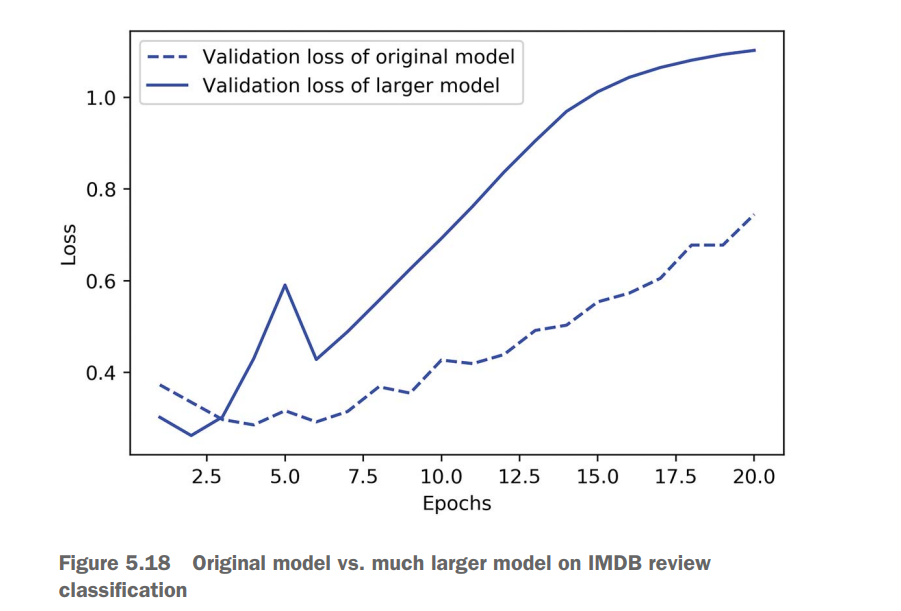
loss="binary\_crossentropy",

metrics=["accuracy"])

history\_larger\_model = model.fit(

train\_data, train\_labels,

epochs=20, batch\_size=512, validation\_split=0.4)



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 nhiễu 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*

Bạn 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 bạn 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 bạn 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",

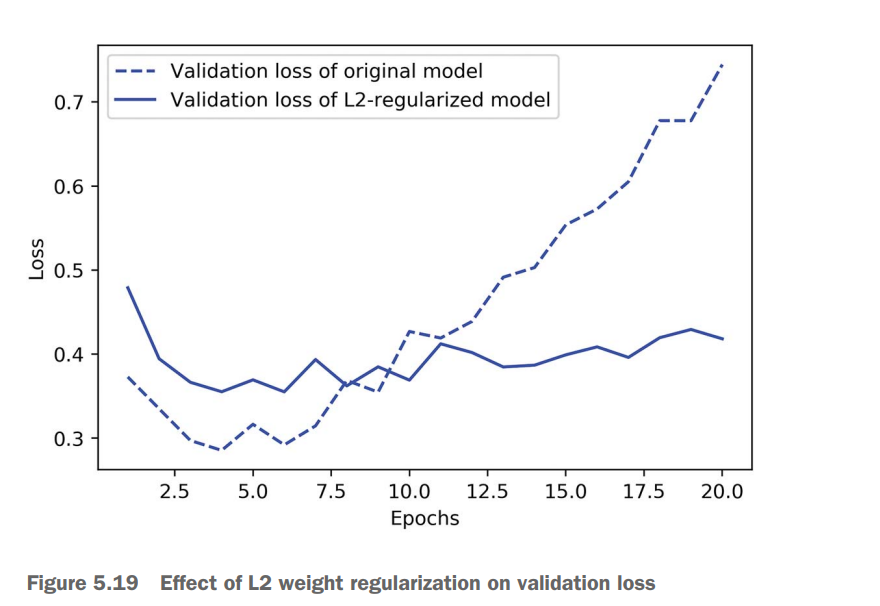
loss="binary\_crossentropy",

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 \* 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 5.19 minh họa tác động của hình phạt L2 regularization. Như bạn 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.



*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

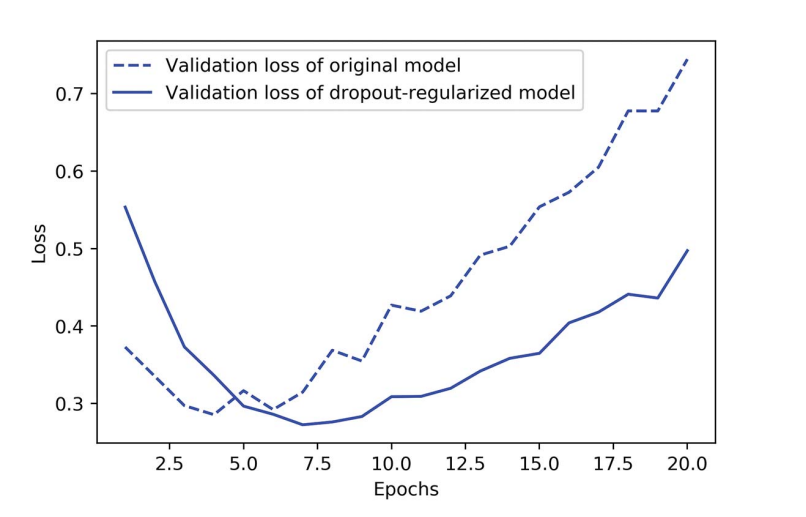
Kỹ thuật này có vẻ kỳ lạ và tùy ý. Tại sao nó lại giúp giảm overfitting?

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 nhiễu 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 nhiễu.

Trong Keras, bạn 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

**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 bạn 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. Bạn 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 bạn 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 bạn chuyển sang **cải thiện tổng quát hóa của mô hình** thông qua việc điều chỉnh mô hình. Bạn 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.

**PHẦN 2: QUY TRÌNH LÀM VIỆC CHUNG CỦA HỌC MÁY**

1. **Define the task (Xác định bài toán)**

Phần 1 tập trung vào bước đầu tiên trong quy trình Machine Learning: hiểu rõ bài toán và chuẩn bị dữ liệu. Đây là một bước cực kỳ quan trọng vì nếu bạn định nghĩa bài toán sai ngay từ đầu, tất cả các bước sau đó (thu thập dữ liệu, xây dựng mô hình, đánh giá kết quả) có thể đi sai hướng.

* 1. **Frame the Problem (Định khung bài toán)**
* Đây là bước giúp xác định mục tiêu, loại dữ liệu cần thiết, phương pháp tiếp cận và ràng buộc của bài toán. Cần trả lời được các câu hỏi sau:
* Dữ liệu đầu vào là gì? Bạn đang cố dự đoán điều gì?
* Bài toán của bạn thuộc loại Machine Learning nào?
* Phân loại nhị phân (Binary classification)
* Phân loại đa lớp (Multiclass classification)
* Hồi quy (Regression)
* Phân cụm (Clustering)
* Phát sinh dữ liệu (Generative Modeling)
* Các giải pháp hiện tại như thế nào?
* Có ràng buộc đặc biệt nào không?
* Ví dụ: Bài toán phát hiện gian lận thẻ tín dụng

Bước 1: Xác định bài toán

* Mục tiêu: Dự đoán giao dịch có phải là gian lận hay không
* Loại bài toán: Phân loại nhị phân (Binary classification).
* Dữ liệu đầu vào: Các thông tin giao dịch như số tiền, vị trí, loại thẻ, thời gian giao dịch.
* Nhãn đầu ra: 1 nếu là gian lận, 0 nếu không gian lận.

Bước 2: Đánh giá phương pháp hiện tại

* Hiện tại, ngân hàng sử dụng hệ thống quy tắc cứng (ví dụ: chặn giao dịch nếu số tiền vượt quá một ngưỡng nhất định).
* Nhược điểm: Hệ thống này có thể bỏ sót giao dịch gian lận tinh vi và chặn nhầm giao dịch hợp lệ.

Bước 3: Các ràng buộc cần xem xét

* Hệ thống phải hoạt động nhanh để không làm gián đoạn trải nghiệm thanh toán của khách hàng.
* Mô hình cần có độ chính xác cao vì sai sót có thể gây thiệt hại lớn cho ngân hàng.
  1. **Collect a Dataset (Thu thập tập dữ liệu)**
     1. Các phương pháp thu thập dữ liệu
* Dữ liệu có sẵn (Existing datasets)
* Các nguồn dữ liệu mở như Kaggle, UCI Machine Learning Repository, Google Dataset Search.
* Các tập dữ liệu có sẵn trong thư viện Keras.datasets, như MNIST (chữ số viết tay), CIFAR-10 (hình ảnh), IMDB (bình luận phim).
* Thu thập dữ liệu từ thực tế (Collecting your own data)
* Ví dụ: Một công ty sản xuất bánh quy có thể lắp đặt camera trên băng chuyền để chụp ảnh
* bánh lỗi, sau đó sử dụng nhân công để dán nhãn
* Ví dụ: Hệ thống gợi ý nhạc của Spotify dựa vào dữ liệu người dùng:
* Đầu vào (Input): Danh sách các bài hát mà người dùng thích.
* Đầu ra (Target): Gợi ý bài hát mới dựa trên sở thích
  + 1. Chất lượng dữ liệu quan trọng hơn thuật toán

Ví dụ minh họa dự đoán giá nhà: Nếu bạn đang xây dựng mô hình dự đoán giá nhà:

* Dữ liệu kém chất lượng: Chỉ có diện tích và giá nhà → Mô hình không hiệu quả.
* Dữ liệu tốt: Có thêm số phòng ngủ, vị trí, tuổi nhà, tình trạng bảo trì → Mô hình tốt hơn.
  + 1. Chú ý đến Bias trong dữ liệu

Ví dụ lịch sử: Bầu cử Mỹ năm 1948

* Báo Chicago Tribune dự đoán sai kết quả bầu cử vì dùng khảo sát qua điện thoại.
* Năm 1948, chỉ người giàu có điện thoại, nên kết quả khảo sát bị lệch.
* Hệ thống AI cũng có thể mắc lỗi này nếu dữ liệu không đại diện.

✅ Giải pháp: Đảm bảo dữ liệu thu thập phản ánh đúng thực tế bằng cách lấy mẫu đa dạng

* + 1. Concept Drift – Khi dữ liệu thay đổi theo thời gian

Ví dụ về Concept Drift

* Mô hình phát hiện gian lận thẻ tín dụng: Kiểu gian lận thay đổi theo thời gian, nên phải liên tục cập nhật dữ liệu.
* Hệ thống gợi ý nhạc: Người dùng thích nhạc khác nhau theo từng năm → Cần huấn luyện lại mô hình định kỳ.

✅ Giải pháp: Luôn thu thập dữ liệu mới và kiểm tra xem mô hình có cần cập nhật không.

* + 1. Cách chú thích dữ liệu (Data Annotation)

Ví dụ minh họa: Nhận diện giống chó

* Bài toán: Xác định giống chó trong ảnh.
* Nếu chỉ cần phân loại chó vs mèo, ai cũng có thể gán nhãn.
* Nhưng nếu cần phân biệt 100 giống chó, bạn cần chuyên gia.
  1. **Hiểu dữ liệu của bạn (Understand Your Data)**
     1. Tại sao cần hiểu dữ liệu?

• Xác định xem dữ liệu có có đủ thông tin để giải quyết bài toán không.

• Kiểm tra xem dữ liệu có mất cân bằng hay bị thiếu giá trị không.

• Phát hiện lỗi hoặc rò rỉ dữ liệu (data leakage), tức là có cột nào tiết lộ quá nhiều thông tin về kết quả dự đoán.

* + 1. Các bước quan trọng để hiểu dữ liệu
* Xem xét dữ liệu mẫu
* Nếu dữ liệu là ảnh hoặc văn bản, hãy xem xét một số mẫu trực tiếp.

Ví dụ về bài toán phân loại cảm xúc bình luận phim (Sentiment Analysis)

* Dữ liệu: Văn bản bình luận phim.
* Hành động: Đọc một số bình luận và kiểm tra nhãn cảm xúc có chính xác không.
* Nếu dữ liệu là số, hãy vẽ biểu đồ để kiểm tra phân bố giá trị.

Ví dụ về dự đoán giá nhà

* Dữ liệu: Diện tích, số phòng ngủ, giá nhà.
* Hành động: Vẽ biểu đồ histogram để xem giá trị có phân bố đồng đều không.

*import pandas as pd*

*import matplotlib.pyplot as plt*

*# Tạo dữ liệu giả định*

*data = pd.DataFrame({*

*"Diện tích (m2)": [50, 60, 75, 100, 120, 150, 180, 200, 220, 250],*

*"Giá nhà (tỷ VNĐ)": [1.2, 1.5, 2.0, 2.8, 3.5, 4.2, 5.0, 5.8, 6.5, 7.0]*

*})*

*# Vẽ biểu đồ histogram của giá nhà*

*plt.hist(data["Giá nhà (tỷ VNĐ)"], bins=5, edgecolor="black")*

*plt.xlabel("Giá nhà (tỷ VNĐ)")*

*plt.ylabel("Số lượng nhà")*

*plt.title("Phân bố giá nhà")*

*plt.show()*

* Kiểm tra dữ liệu bị thiếu

Nếu dữ liệu có cột bị thiếu nhiều giá trị, bạn phải quyết định:

* Loại bỏ cột/ dòng có quá nhiều dữ liệu bị thiếu.
* Điền giá trị thay thế (ví dụ: dùng giá trị trung bình).
* Kiểm tra sự mất cân bằng trong dữ liệu

Nếu dữ liệu huấn luyện có sự chênh lệch lớn giữa các lớp, mô hình có thể bị thiên vị.

Ví dụ về phân loại bệnh nhân mắc bệnh tim

* 95% mẫu là bệnh nhân khỏe mạnh.
* 5% mẫu là bệnh nhân mắc bệnh tim.
* Mô hình có thể học cách luôn dự đoán "khỏe mạnh" để đạt độ chính xác cao nhưng thực chất không có giá trị.
* Kiểm tra rò rỉ dữ liệu (Data Leakage)

Rò rỉ dữ liệu xảy ra khi một cột nào đó trong dữ liệu cung cấp thông tin trực tiếp về nhãn dự đoán, khiến mô hình đạt độ chính xác cao một cách giả tạo.

* Ví dụ sai lầm trong bài toán dự đoán bệnh ung thư
* Nếu dữ liệu đầu vào có cột “Bệnh nhân đã được điều trị ung thư chưa?”, thì mô hình có thể dễ dàng dự đoán bệnh nhân có ung thư hay không.
* Nhưng cột này không tồn tại trong thực tế khi triển khai mô hình!
* Giải pháp: Loại bỏ các cột có thể dẫn đến rò rỉ dữ liệu trước khi huấn luyện.

*from sklearn.datasets import load\_boston*

*import pandas as pd*

*import seaborn as sns*

*import matplotlib.pyplot as plt*

*# Tải dữ liệu*

*boston = load\_boston()*

*df = pd.DataFrame(boston.data, columns=boston.feature\_names)*

*df["PRICE"] = boston.target*

*# Kiểm tra dữ liệu sơ bộ*

*print(df.head())*

*print(df.info())*

*# Xem 5 dòng đầu tiên*

*# Kiểm tra thông tin dữ liệu*

*print(df.describe()) # Kiểm tra thống kê cơ bản*

*# Kiểm tra dữ liệu bị thiếu*

*print(df.isnull().sum()) # Xem có cột nào bị thiếu không*

*df.fillna(df.mean(), inplace=True) # Điền giá trị bị thiếu bằng trung bình*

*# Kiểm tra phân bố giá nhà*

*sns.histplot(df["PRICE"], bins=30, kde=True)*

*plt.xlabel("Giá nhà (nghìn USD)")*

*plt.ylabel("Số lượng nhà")*

*plt.title("Phân phối giá nhà ở Boston")*

*plt.show()*

*# Kiểm tra sự mất cân bằng*

*sns.boxplot(x=df["PRICE"])*

*plt.title("Boxplot kiểm tra giá trị ngoại lai")*

*plt.show()*

*# Kiểm tra rò rỉ dữ liệu (tương quan giữa các biến)*

*corr\_matrix = df.corr()*

*sns.heatmap(corr\_matrix, annot=True, cmap="coolwarm")*

*plt.title("Biểu đồ tương quan giữa các biến")*

*plt.show()*

*# Kiểm tra cột nào ảnh hưởng mạnh nhất đến giá nhà*

*print(df.corr()["PRICE"].sort\_values(ascending=False))*

* 1. **Chọn thước đo thành công**
     1. Tại sao cần chọn thước đo thành công?
* Xác định rõ mục tiêu: Bạn cần biết chính xác mình đang cố gắng tối ưu điều gì.
* So sánh mô hình khác nhau: Khi có nhiều mô hình, bạn phải có cách đánh giá cái nào tốt hơn.
* Tránh mô hình "tốt nhưng vô dụng": Một mô hình có độ chính xác cao nhưng không phục vụ đúng mục tiêu thực tế có thể vô ích.

Ví dụ:

* Nếu bạn đang xây dựng một hệ thống phát hiện gian lận thẻ tín dụng, một mô hình có độ chính xác cao nhưng bỏ sót nhiều gian lận sẽ không thực tế.
* Trong hệ thống gợi ý sản phẩm, bạn không chỉ quan tâm đến độ chính xác, mà còn quan tâm đến tỷ lệ giữ chân khách hàng.
  + 1. Các loại thước đo thành công phổ biến
* Bài toán phân loại (Classification)

Nếu bài toán của bạn yêu cầu phân loại (ví dụ: phân loại email là spam hay không), có một số thước đo phổ biến:



Ví dụ thực tế:

Trong bài toán phát hiện ung thư, Precision cao giúp tránh chẩn đoán sai người khỏe mạnh là mắc ung thư, trong khi Recall cao đảm bảo không bỏ sót bệnh nhân ung thư

*from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score*

*# Giả sử y\_true là nhãn thực tế và y\_pred là nhãn mô hình dự đoán*

*y\_true = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]*

*y\_pred = [1, 0, 1, 0, 0, 1, 1, 0, 1, 0]*

*print("Accuracy:", accuracy\_score(y\_true, y\_pred))*

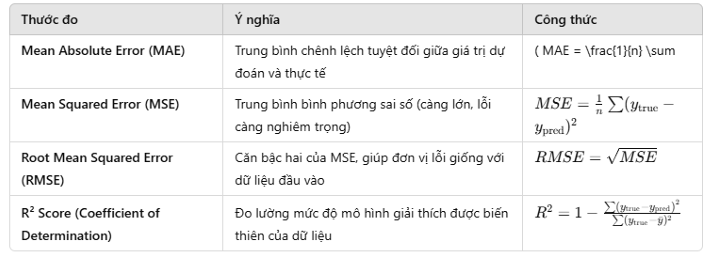
*print("Precision:", precision\_score(y\_true, y\_pred))*

*print("Recall:", recall\_score(y\_true, y\_pred))*

*print("F1 Score:", f1\_score(y\_true, y\_pred))*

* Bài toán hồi quy (Regression)

Nếu bài toán của bạn yêu cầu dự đoán một giá trị liên tục (ví dụ: giá nhà), bạn có thể dụng các thước đo sau:



* Ví dụ thực tế:

Nếu bạn dự đoán giá nhà, MAE giúp biết sai số trung bình là bao nhiêu, còn R² Score cho biết mô hình có giải thích được sự thay đổi của dữ liệu không

*From sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score*

*Giả sử y\_true là giá trị thực tế, y\_pred là giá trị mô hình dự đoán*

*y\_true = [100, 200, 300, 400, 500] y\_pred = [110, 100, 200, 410, 400]*

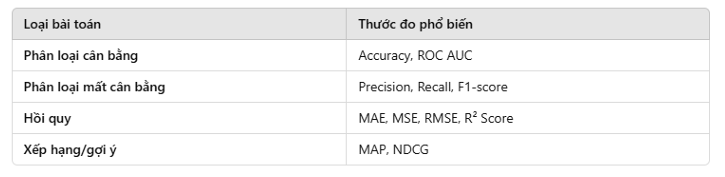
*print("MAE:", mean\_absolute\_error(y\_true, y\_pred))*

*print("MSE:", mean\_squared\_error(y\_true, y\_pred))*

*print("RMSE:", mean\_squared\_error(y\_true, y\_pred, squared=False))*

*print("R² Score:", r2\_score(y\_true, y\_pred))*

* + 1. Chọn thước đo thành công phù hợp



1. **Develop a model (Phát triển mô hình)**

Phần 2 tập trung vào quy trình phát triển mô hình, bao gồm các bước từ tiền xử lý dữ liệu đến tối ưu hóa mô hình để đạt hiệu suất tốt nhất. Đây là giai đoạn quan trọng để đảm bảo mô hình có thể học tốt từ dữ liệu mà không bị quá khớp (overfitting) hoặc chưa khớp (underfitting).

* 1. **Chuẩn bị dữ liệu (Prepare the Data)**
     1. Tại sao cần có baseline?

Dữ liệu thực tế thường không ở dạng mà mô hình Machine Learning có thể sử dụng ngay. Bạn cần phải:

* Chuyển đổi dữ liệu về dạng số để mô hình có thể đọc hiểu (Vectorization).
* Chuẩn hóa dữ liệu để tránh giá trị quá lớn hoặc quá nhỏ gây ảnh hưởng đến quá trình học (Normalization).
* Xử lý dữ liệu bị thiếu để tránh gây sai lệch trong quá trình huấn luyện.
  + 1. Các bước quan trọng trong chuẩn bị dữ liệu
* Vectorization – Chuyển đổi dữ liệu về dạng tensor
* Vấn đề: Machine Learning không thể hiểu văn bản, hình ảnh, hay âm t hanh trực tiếp mà cần chuyển đổi chúng về dạng số.

✅ Ví dụ với dữ liệu văn bản:

* Nếu bạn có một câu: "Tôi yêu học máy" → Cần chuyển nó thành chuỗi số hoặc biểu diễn vector.
* Một cách phổ biến là one-hot encoding hoặc embedding vector.
* Cách thực hiện vectorization cho văn bản bằng Python:

*from tensorflow.keras.preprocessing.text import Tokenizer*

*texts = ["Tôi yêu học máy", "Học máy rất thú vị"]*

*tokenizer = Tokenizer(num\_words=100) tokenizer.fit\_on\_texts(texts)*

*sequences = tokenizer.texts\_to\_sequences(texts)*

*print(sequences)*

* *Cách thực hiện vectorization cho hình ảnh:*

*from tensorflow.keras.datasets import mnist*

*# Tải dữ liệu MNIST (ảnh chữ số viết tay)*

*(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data() # Chuyển ảnh thành dạng tensor*

*train\_images = train\_images.reshape((60000, 28, 28, 1))*

*train\_images = train\_images.astype("float32") / 255 # Chuẩn hóa giá trị về 0 1*

* Normalization – Chuẩn hóa dữ liệu
* Vấn đề: Nếu dữ liệu có giá trị quá lớn (ví dụ: 10,000) và quá nhỏ (ví dụ:

0.1), mô hình có thể khó học do gradient cập nhật không ổn định .

* Giải pháp: Biến đổi tất cả các giá trị về một phạm vi chuẩn như [0, 1] hoặc có giá trị trung bình 0, độ lệch chuẩn 1.
* Ví dụ chuẩn hóa dữ liệu giá nhà:

*import numpy as np*

*# Giả sử x là tập dữ liệu đầu vào*

*x = np.array([[100, 200], [150, 300], [200, 400]])*

*# Chuẩn hóa về trung bình 0, độ lệch chuẩn 1*

*x -= x.mean(axis=0)*

*x /= x.std(axis=0)*

*print(x)*

* Handling Missing Values – Xử lý dữ liệu bị thiếu
* Vấn đề: Nếu dữ liệu có giá trị bị thiếu (NaN), mô hình sẽ không thể học tốt.
* Giải pháp:

1. Loại bỏ dòng/cột bị thiếu (nếu số lượng bị thiếu quá nhiều).

2. Điền giá trị thay thế (trung bình, trung vị, hoặc một giá trị đặc biệt).

* Ví dụ kiểm tra và xử lý dữ liệu bị thiếu:

*import pandas as pd*

*# Giả sử có dữ liệu giá nhà với một số giá trị bị thiếu*

*data = pd.DataFrame({"Diện tích": [50, 60, np.nan, 100], "Giá nhà": [1.2,*

*np.nan, 2.0, 2.8]})*

*# Kiểm tra dữ liệu bị thiếu*

*print(data.isnull().sum())*

*# Điền giá trị bị thiếu bằng trung bình của cột*

*data.fillna(data.mean(), inplace=True)*

*print(data)*

* 1. **Chọn phương thức đánh giá mô hình (Choose an Evaluation Protocol)**
     1. Tại sao cần chọn phương thức đánh giá?
* Giúp đo lường khả năng tổng quát hóa (generalization) của mô hình.
* Phát hiện vấn đề quá khớp (overfitting) hoặc chưa khớp (underfitting).
* Đảm bảo cách đánh giá công bằng khi so sánh giữa các mô hình khác nhau.

Ví dụ: Nếu bạn đang xây dựng mô hình nhận diện chữ số viết tay, bạn cần một cách đánh giá khách quan để biết mô hình hoạt động tốt đến đâu trên dữ liệu mới.

* + 1. Các phương thức đánh giá phổ biến
* Giữ lại một tập kiểm tra cố định (Holdout Validation)
* Cách làm:

Chia dữ liệu thành ba tập:

* Training set (tập huấn luyện) – dùng để huấn luyện mô hình.
* Validation set (tập xác thực) – dùng để điều chỉnh tham số và chọn mô hình tốt nhất.
* Test set (tập kiểm tra) – chỉ dùng một lần cuối cùng để đánh giá mô hình.
* Khi nào nên dùng?

Khi có nhiều dữ liệu, bạn có thể tách riêng một phần để làm validation và test.

* Ví dụ với Python:

*from sklearn.model\_selection import train\_test\_split*

*import numpy as np*

*# Tạo dữ liệu mẫu*

*X = np.random.rand(1000, 10) # 1000 mẫu, mỗi mẫu có 10*

*đặc trưng*

*y = np.random.randint(0, 2, 1000) # Nhãn nhị phân (0 hoặc*

*1)*

*# Chia thành training (70%), validation (15%), test (15%)*

*X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y,*

*test\_size=0.3, random\_state=42)*

*X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp,*

*y\_temp, test\_size=0.5, random\_state=42)*

*print("Training set size:", len(X\_train))*

*print("Validation set size:", len(X\_val))*

*print("Test set size:", len(X\_test))*

* K-fold Cross-Validation
* Cách làm:
* Chia dữ liệu thành K phần bằng nhau (folds).
* Lặp lại K lần: mỗi lần chọn một phần làm validation set, phần còn lại làm training set.
* Cuối cùng, lấy trung bình kết quả của K lần để có đánh giá chính xác hơn.
* Khi nào nên dùng?

Khi dữ liệu ít, vì phương pháp này giúp tận dụng tối đa dữ liệu.

* Ví dụ với Python (K=5):

*from sklearn.model\_selection import KFold*

*kf = KFold(n\_splits=5, shuffle=True, random\_state=42)*

*for train\_index, val\_index in kf.split(X):*

*print("Train:", train\_index, "Validation:", val\_index)*

* Iterated K-fold Cross-Validation
* Cách làm:
* Giống K-fold, nhưng lặp lại nhiều lần với cách chia dữ liệu khác

nhau

* Cuối cùng, lấy trung bình kết quả của tất cả các lần lặp.
* Khi nào nên dùng?

Khi dữ liệu cực kỳ ít, giúp đánh giá mô hình chính xác hơn.

* Nhược điểm:

Tốn rất nhiều thời gian vì phải huấn luyện mô hình nhiều lần hơn cả

K-fold.

* 1. **Vượt qua một baseline (Beat a Baseline)**

Sau khi chọn phương thức đánh giá mô hình, bước tiếp theo là xây dựng một mô hình đơn giản có thể vượt qua một phương pháp cơ bản (baseline). Baseline giúp bạn xác định liệu mô hình học máy có thực sự mang lại giá trị hay không.

* + 1. Tại sao cần có baseline?
* Đảm bảo rằng mô hình đang sử dụng thông tin trong dữ liệu một cách hợp lý.
* Nếu mô hình không thể vượt qua baseline, có thể bạn cần kiểm tra lại dữ liệu hoặc cách tiếp cận.
* Giúp so sánh các mô hình một cách khách quan.

Ví dụ:

* Trong bài toán phân loại email spam, baseline đơn giản có thể là luôn dự đoán "không phải spam".
* Trong bài toán dự đoán giá nhà, baseline có thể là dự đoán trung bình của toàn bộ tập dữ liệu.
  + 1. Các bước để đánh bại baseline
* Feature Engineering – Tạo và chọn đặc trưng phù hợp
* Vấn đề: Không phải tất cả các cột trong dữ liệu đều hữu ích. Bạn cần chọn và tạo đặc trưng mới để cải thiện hiệu suất mô hình.
* Ví dụ với bài toán dự đoán giá nhà:

Dữ liệu gốc: Diện tích nhà, số phòng ngủ, năm xây dựng. Đặc trưng mới:

* Giá nhà trung bình theo khu vực (có thể rất hữu ích).
* Tỷ lệ diện tích trên số phòng ngủ (có thể phản ánh mức độ cao cấp của ngôi nhà).
* Cách thực hiện Feature Selection với Python:

*from sklearn.feature\_selection import SelectKBest, f\_regression*

*import pandas as pd*

*# Giả sử có dữ liệu về giá nhà*

*data = pd.DataFrame({*

*"Diện tích": [50, 100, 150, 200, 250],*

*"Số phòng": [2, 3, 3, 4, 4],*

*"Giá nhà": [1.2, 2.5, 3.8, 4.5, 5.5]*

*})*

*X = data[["Diện tích", "Số phòng"]]*

*y = data["Giá nhà"]*

*# Chọn 1 đặc trưng tốt nhất*

*selector = SelectKBest(score\_func=f\_regression, k=1)*

*X\_new = selector.fit\_transform(X, y)*

*print("Đặc trưng quan trọng nhất:", X.columns[selector.get\_support()])*

* Chọn kiến trúc mô hình phù hợp

Ví dụ với mô hình hồi quy tuyến tính để dự đoán giá nhà:

*from sklearn.linear\_model import LinearRegression*

*model = LinearRegression()*

*model.fit(X, y)*

*# Dự đoán giá nhà*

*predictions = model.predict(X)*

*print(predictions)*

* Chọn cấu hình huấn luyện hợp lý

Vấn đề: Chọn hàm mất mát, batch size, learning rate sao cho tối ưu.

Ví dụ chọn hàm mất mát cho bài toán phân loại dùng TensorFlow/Keras:

*import tensorflow as tf*

*from tensorflow.keras import layers*

*# Xây dựng mô hình*

*model = tf.keras.Sequential([*

*layers.Dense(16, activation="relu"),*

*layers.Dense(1, activation="sigmoid")*

*])*

*# Cấu hình mô hình với hàm mất mát crossentropy*

*model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])*

* 1. **Mở rộng mô hình - Phát triển một mô hình quá khớp (Scale Up: Develop a Model That Overfits)**
     1. Tại sao cần phát triển một mô hình quá khớp?
* Xác định giới hạn của mô hình: Nếu mô hình không thể quá khớp, có thể nó chưa đủ mạnh để học dữ liệu.
* Tìm ranh giới giữa underfitting và overfitting: Mô hình lý tưởng là đứng ngay tại ranh giới này.
* Cải thiện khả năng tổng quát hóa sau này: Một khi đã quá khớp, ta có thể điều chỉnh mô hình để làm nó tổng quát hơn.

Ví dụ:

* Một mô hình Logistic Regression có thể đạt kết quả tốt trên bộ dữ liệu MNIST nhưng không thể học các mối quan hệ phức tạp như CNN.
* Nếu mô hình của bạn không bao giờ đạt điểm quá cao trên tập huấn luyện, có thể nó chưa đủ lớn hoặc chưa được huấn luyện đủ lâu.
  + 1. Cách làm mô hình quá khớp
* Thêm nhiều lớp (Add layers)

Ví dụ với mạng nơ-ron nhiều lớp (DNN):

*from tensorflow import keras*

*from tensorflow.keras import layers*

*model = keras.Sequential([*

*layers.Dense(256, activation="relu"),*

*layers.Dense(256, activation="relu"),*

*layers.Dense(256, activation="relu"),*

*layers.Dense(1, activation="sigmoid")*

*])*

*model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])*

* Tăng số lượng neuron trong mỗi lớp (Make layers bigger)

model = keras.Sequential([

layers.Dense(1024, activation="relu"),

layers.Dense(1024, activation="relu"),

layers.Dense(1, activation="sigmoid")

])

* Huấn luyện lâu hơn (Train for more epochs)

history = model.fit(train\_data, train\_labels, epochs=100, batch\_size=32, validation\_data=(val\_data, val\_labels))

* + 1. Cách phát hiện overfitting

Khi mô hình quá khớp, bạn sẽ thấy loss trên tập huấn luyện giảm mạnh, trong khi loss trên tập validation bắt đầu tăng lên.

import matplotlib.pyplot as plt

plt.plot(history.history["loss"], label="Training loss")

plt.plot(history.history["val\_loss"], label="Validation loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

* 1. **Điều chỉnh mô hình và tối ưu siêu tham số (Regularize and Tune Your Model)**
     1. Tại sao cần regularization và tuning?
* Giảm overfitting: Mô hình phức tạp có thể nhớ dữ liệu huấn luyện quá kỹ, dẫn đến tổng quát kém.
* Cải thiện hiệu suất trên dữ liệu thực tế: Giúp mô hình hoạt động tốt trên dữ liệu chưa thấy trước.
* Tìm kiếm cấu hình tốt nhất: Tối ưu siêu tham số giúp mô hình chạy hiệu quả hơn.

Ví dụ:

* Nếu bạn có một mô hình dự đoán giá nhà quá khớp, việc thêm dropout hoặc L2 regularization có thể giúp cải thiện tổng quát hóa.
* Nếu mô hình chạy chậm, giảm số lượng neuron hoặc thử nghiệm với learning rate khác nhau có thể tăng tốc mà không làm giảm chất lượng.
  + 1. Các phương pháp regularization phổ biến
* Dropout – Ngăn chặn mạng nơ-ron ghi nhớ dữ liệu

*from tensorflow.keras import layers, models*

*model = models.Sequential([*

*layers.Dense(256, activation="relu"),*

*layers.Dropout(0.5), # 50% số neuron sẽ bị vô hiệu hóa mỗi lần huấn luyện*

*layers.Dense(128, activation="relu"),*

*layers.Dense(1, activation="sigmoid")*

*])*

* L1 & L2 Regularization – Hạn chế giá trị trọng số quá lớn
* L1 Regularization: Tạo ra trọng số thưa thớt (nhiều giá trị gần 0), giúp chọn ra các đặc trưng quan trọng nhất.
* L2 Regularization (Ridge Regression): Hạn chế giá trị trọng số lớn, giúp tránh việc mô hình quá phụ thuộc vào một số feature nhất định.

Ví dụ với L2 Regularization trong Keras:

*from tensorflow.keras.regularizers import l2*

*model = models.Sequential([*

*layers.Dense(256, activation="relu", kernel\_regularizer=l2(0.01)), # Áp dụng L2 regularization*

*layers.Dense(128, activation="relu"),*

*layers.Dense(1, activation="sigmoid")*

*])*

* Batch Normalization – Ổn định quá trình học

Cách hoạt động: Chuẩn hóa đầu ra của mỗi lớp về giá trị trung bình 0 và độ lệch chuẩn 1, giúp mô hình hội tụ nhanh hơn và ổn định hơn.

Ví dụ với Batch Normalization:

*model = models.Sequential([*

*layers.Dense(256, activation="relu"),*

*layers.BatchNormalization(), # Thêm bước chuẩn hóa*

*layers.Dense(128, activation="relu"),*

*layers.Dense(1, activation="sigmoid")*

*])*

* + 1. Tối ưu siêu tham số (Hyperparameter Tuning)
* Sau khi đã áp dụng regularization, bước tiếp theo là tìm kiếm các tham số tối ưu như:
* Số lớp và số neuron mỗi lớp
* Hàm kích hoạt
* Batch size, learning rate
* Hệ số dropout, regularization (L1, L2)
* Cách thử nghiệm thủ công bằng Grid Search hoặc Random Search:

*from sklearn.model\_selection import GridSearchCV*

*from sklearn.ensemble import RandomForestClassifier*

*# Định nghĩa tập siêu tham số để thử nghiệm*

*param\_grid = {*

*"n\_estimators": [50, 100, 200],*

*"max\_depth": [10, 20, 30],*

*"min\_samples\_split": [2, 5, 10]*

*}*

*# Sử dụng GridSearchCV để tìm kiếm giá trị tối ưu*

*grid\_search = GridSearchCV(RandomForestClassifier(), param\_grid, cv=3)*

*grid\_search.fit(X\_train, y\_train)*

*print("Best parameters:", grid\_search.best\_params\_)*

* Sử dụng AutoML với KerasTuner (tự động tìm kiếm siêu tham số tốt nhất):

*import keras\_tuner as kt*

*def build\_model(hp):*

*model = models.Sequential()*

*model.add(layers.Dense(hp.Int("units", 32, 512, step=32), activation="relu"))*

*model.add(layers.Dense(1, activation="sigmoid"))*

*model.compile(optimizer="adam", loss="binary\_crossentropy", metrics=["accuracy"])*

*return model*

*tuner = kt.RandomSearch(build\_model, objective="val\_accuracy", max\_trials=10)*

*tuner.search(X\_train, y\_train, epochs=10, validation\_data=(X\_val, y\_val))*

*best\_hps = tuner.get\_best\_hyperparameters(num\_trials=1)[0]*

*print("Best units:", best\_hps.get("units"))*

1. **Deploy the model (Triển khai mô hình)**

Phần 3 tập trung vào giai đoạn sau khi mô hình đã hoàn thành, bao gồm việc triển khai vào môi trường thực tế và duy trì hiệu suất lâu dài. Đây là một bước quan trọng vì một mô hình dù huấn luyện tốt đến đâu cũng có thể trở nên lỗi thời nếu không được giám sát và cập nhật thường xuyên.

* 1. **Giải thích mô hình với các bên liên quan và đặt kỳ vọng (Explain Your Work to Stakeholders and Set Expectations)** 
     1. Tại sao cần giải thích mô hình cho stakeholders?
* Xây dựng niềm tin: Nếu những người ra quyết định không hiểu mô hình, họ sẽ ngần ngại sử dụng nó.
* Tránh kỳ vọng không thực tế: Nhiều người nghĩ AI có thể làm được mọi thứ, nhưng thực tế không phải vậy.
* Xác định cách sử dụng mô hình đúng cách: Một số sai lầm phổ biến có thể làm mô hình bị lạm dụng hoặc hiểu sai.

Ví dụ:

* Một hệ thống phát hiện gian lận không thể đảm bảo 100% chính xác, vì vậy cần định rõ tỷ lệ sai sót (false positives, false negatives) để tránh gây hiểu nhầm.
* Một mô hình phân loại ảnh không thực sự "hiểu" hình ảnh như con người, nó chỉ tìm ra các mẫu trong dữ liệu.
  + 1. Cách giải thích mô hình cho stakeholders
* Giải thích mô hình bằng ngôn ngữ dễ hiểu
* Vấn đề: Các nhà quản lý, khách hàng, hoặc đội kinh doanh thường không có kiến thức sâu về AI.
* Giải pháp: Tránh dùng thuật ngữ kỹ thuật phức tạp, thay vào đó:
* So sánh với ví dụ thực tế: “Mô hình của chúng ta giống như một người học
* cách nhận diện chữ viết tay dựa trên hàng ngàn ví dụ trước đó.”
* Dùng đồ thị, hình ảnh để minh họa.
* Trình bày bằng các câu hỏi đơn giản:
* Mô hình hoạt động như thế nào?
* Nó có thể mắc lỗi gì?
* Nó sẽ mang lại lợi ích gì cho doanh nghiệp?
* Ví dụ trình bày trực quan bằng Python:

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*# Giả sử có dữ liệu về tỷ lệ sai sót của mô hình*

*false\_negatives = 5 # 5% bỏ sót gian lận*

*false\_positives = 2.5 # 2.5% giao dịch hợp lệ bị cờ gian lận*

*labels = ["Phát hiện đúng gian lận", "Bỏ sót gian lận", "Sai cờ gian lận"]*

*sizes = [92.5, false\_negatives, false\_positives]*

*plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)*

*plt.title("Tỷ lệ phát hiện của mô hình AI")*

*plt.show()*

* Giới thiệu các tình huống mô hình có thể thất bại
* Vấn đề: Stakeholders có thể kỳ vọng AI luôn đúng.
* Giải pháp: Cung cấp các ví dụ về lỗi của mô hình để giúp họ hiểu giới hạn của nó.
* Nếu là mô hình phát hiện gian lận, hãy chỉ ra một số giao dịch bị gắn nhãn sai.
* Nếu là mô hình chatbot, hãy cho thấy các câu trả lời kỳ lạ mà AI có thể tạo ra.
* Ví dụ trong Python:

*import random*

*mistakes = ["Dự đoán sai email spam là hợp lệ", "Nhận diện sai chữ viết tay",*

*"Phát hiện nhầm giao dịch bình thường là gian lận"]*

*print("Ví dụ về lỗi của mô hình:", random.choice(mistakes))*

* Định hình kỳ vọng bằng các chỉ số rõ ràng
* Vấn đề: Nếu bạn nói “mô hình đạt độ chính xác 98%”, nhiều người sẽ hiểu nhầm là nó luôn đúng.
* Giải pháp: Sử dụng cách diễn đạt dễ hiểu hơn, ví dụ:
* “Mô hình có tỷ lệ sai sót 5% trên các giao dịch gian lận” thay vì “độ chính xác 95%”.
* “Trung bình mỗi ngày có 200 giao dịch bị kiểm tra thủ công” thay vì “mô hình hoạt động tốt
  + 1. Định nghĩa tham số triển khai quan trọng
* Vấn đề: Quyết định ngưỡng cắt (threshold) ảnh hưởng lớn đến hiệu suất mô hình.
* Giải pháp: Thảo luận với stakeholders về các trade-offs:
* Nếu ngưỡng phát hiện gian lận quá cao, có thể bỏ sót nhiều giao dịch gian lận (false negatives cao).
* Nếu ngưỡng quá thấp, nhiều giao dịch hợp lệ sẽ bị chặn sai (false positives cao).
* Ví dụ mô phỏng ảnh hưởng của threshold:

*`import numpy as np*

*thresholds = np.linspace(0.1, 0.9, 9)*

*false\_negatives = [90, 70, 50, 30, 20, 10, 5, 2, 1]*

*false\_positives = [1, 5, 10, 20, 30, 50, 70, 90, 95]*

*plt.plot(thresholds, false\_negatives, label="False Negatives")*

*plt.plot(thresholds, false\_positives, label="False Positives")*

*plt.xlabel("Ngưỡng phát hiện (Threshold)")*

*plt.ylabel("Số lượng lỗi")*

*plt.legend()*

*plt.show()*

* 1. **Triển khai mô hình (Ship an Inference Model)** 
     1. Những thách thức khi triển khai mô hình
* Môi trường sản xuất có thể không hỗ trợ Python: Nếu ứng dụng chạy trên trình duyệt hoặc thiết bị di động, bạn có thể cần xuất mô hình sang TensorFlow.js hoặc TensorFlow Lite.
* Tối ưu hóa tốc độ và bộ nhớ: Mô hình chỉ thực hiện dự đoán (inference) chứ không cần huấn luyện, do đó có thể tối ưu để giảm kích thước và tăng tốc độ.
* Cách triển khai phù hợp với từng loại ứng dụng: Có nhiều cách để triển khai mô hình, tùy thuộc vào yêu cầu của hệ thống.
  + 1. Các phương thức triển khai phổ biến
* Triển khai dưới dạng REST API
* Cách hoạt động:
* Mô hình được lưu trên một máy chủ hoặc cloud.
* Các ứng dụng khác gửi yêu cầu (request) và nhận kết quả dự đoán thông
* qua API.
* Khi nào nên dùng?
* Khi ứng dụng có kết nối internet ổn định.
* Khi không có yêu cầu thời gian thực khắt khe (độ trễ khoảng 500ms vẫn chấp nhận được).
* Khi dữ liệu không quá nhạy cảm, vì nó phải được gửi lên server.
* Ví dụ triển khai REST API bằng Flask:

*from flask import Flask, request, jsonify*

*import tensorflow as tf*

*app = Flask(\_\_name\_\_)*

*model = tf.keras.models.load\_model("model.h5") # Tải mô hình*

*@app.route("/predict", methods=["POST"])*

*def predict():*

*data = request.get\_json(force=True)*

*prediction = model.predict([data["input"]]).tolist()*

*return jsonify({"prediction": prediction})*

*if \_\_name\_\_ == "\_\_main\_\_":*

*app.run(debug=True)*

* Triển khai mô hình trực tiếp trên thiết bị (Edge AI)
* Cách hoạt động:
* Mô hình được lưu và chạy trực tiếp trên điện thoại, máy ảnh thông minh, thiết bị IoT, v.v.
* Sử dụng TensorFlow Lite hoặc ONNX để tối ưu mô hình.
* Khi nào nên dùng?
* Khi yêu cầu thời gian thực và không có mạng internet.
* Khi dữ liệu nhạy cảm và không thể gửi lên cloud.
* Ví dụ chuyển mô hình Keras sang TensorFlow Lite:

*mport tensorflow as tf*

*model = tf.keras.models.load\_model("model.h5")*

*converter = tf.lite.TFLiteConverter.from\_keras\_model(model)*

*tflite\_model = converter.convert()*

*with open("model.tflite", "wb") as f:*

*f.write(tflite\_model)*

* Triển khai mô hình trong trình duyệt với TensorFlow.js
* Cách hoạt động:
* Mô hình chạy trực tiếp trên trình duyệt, không cần server.
* Giúp giảm chi phí server và tăng tốc phản hồi.
* Khi nào nên dùng?
* Khi muốn tận dụng GPU của người dùng để chạy mô hình.
* Khi dữ liệu cần giữ bí mật trên máy khách (VD: phát hiện spam trên chat app).
* Khi muốn ứng dụng hoạt động offline.
* Ví dụ chuyển mô hình sang TensorFlow.js:

*mport tensorflowjs as tfjs*

*model = tf.keras.models.load\_model("model.h5")*

*tfjs.converters.save\_keras\_model(model, "tfjs\_model")*

* + 1. Tối ưu hóa mô hình khi triển khai

Để mô hình chạy nhanh hơn và sử dụng ít tài nguyên hơn, bạn có thể:

* Weight Pruning: Loại bỏ các trọng số không quan trọng để giảm kích thước mô hình
* Weight Quantization: Chuyển từ float32 sang int8 để giảm dung lượng.
* Batching Requests: Gửi nhiều yêu cầu cùng lúc để tận dụng GPU.
* Ví dụ giảm kích thước mô hình với TensorFlow Model Optimization:

*import tensorflow\_model\_optimization as tfmot*

*prune\_low\_magnitude = tfmot.sparsity.keras.prune\_low\_magnitude*

*model = prune\_low\_magnitude(model)*

*converter = tf.lite.TFLiteConverter.from\_keras\_model(model)*

*tflite\_model = converter.convert()*

* 1. **Giám sát mô hình sau triển khai (Monitor Your Model in the Wild)** 
     1. Tại sao cần giám sát mô hình sau triển khai?
* Phát hiện lỗi hoặc sai lệch dữ liệu (data drift): Dữ liệu trong thực tế có thể khác với dữ liệu huấn luyện.
* Đảm bảo hiệu suất mô hình không giảm dần theo thời gian.
* Theo dõi tác động của mô hình đến hệ thống hoặc doanh nghiệp (ví dụ: tỷ lệ click vào quảng cáo có tăng không?).
* Phát hiện và sửa lỗi sớm trước khi gây ảnh hưởng nghiêm trọng.

Ví dụ:

* Một mô hình đề xuất nhạc có thể hoạt động tốt vào ngày đầu tiên, nhưng nếu sở thích người dùng thay đổi mà mô hình không cập nhật, nó sẽ trở nên lỗi thời.
* Một mô hình phát hiện gian lận thẻ tín dụng có thể bị các kẻ gian lận thích nghi và tìm cách vượt qua, làm giảm độ chính xác.
  + 1. Cách giám sát mô hình hiệu quả
* Sử dụng A/B Testing để đo lường tác động
* Cách hoạt động:
* Chia người dùng thành 2 nhóm:
* Nhóm A: Vẫn sử dụng mô hình cũ.
* Nhóm B: Sử dụng mô hình mới.
* So sánh kết quả giữa hai nhóm để xem mô hình mới có thực sự tốt hơn không.
* Ví dụ với hệ thống quảng cáo:
* Nếu mô hình mới dự đoán tỷ lệ click quảng cáo (CTR) tốt hơn, bạn sẽ thấy CTR của nhóm B tăng cao hơn nhóm A.
* Nếu không có sự khác biệt hoặc nhóm B có CTR thấp hơn, mô hình mới có thể không tốt như kỳ vọng.
* Lợi ích: Giúp xác định mô hình có thực sự hiệu quả hay không trước khi áp dụng rộng rãi.
* Kiểm tra thủ công các dự đoán của mô hình
* Cách hoạt động:
* Chọn một tập nhỏ dữ liệu thực tế.
* Yêu cầu chuyên gia kiểm tra và đánh giá xem mô hình có đưa ra dự đoán chính xác không.
* So sánh kết quả với nhãn thực tế để đo lường hiệu suất.
* Ví dụ với hệ thống tìm kiếm hình ảnh:

Nếu mô hình đề xuất sai hình ảnh không liên quan, bạn cần kiểm tra

lại thuật toán.

* Lợi ích: Giúp phát hiện các sai sót khó nhận ra chỉ bằng số liệu.
* Theo dõi chỉ số quan trọng theo thời gian
* Cách hoạt động:
* Lưu trữ dữ liệu dự đoán và đánh giá hiệu suất theo thời gian.
* Nếu accuracy, precision, recall giảm dần, mô hình có thể bắt đầu lỗi thời.
* Ví dụ theo dõi bằng Python:

*import matplotlib.pyplot as plt*

*# Giả sử có dữ liệu hiệu suất theo thời gian*

*dates = ["Tuần 1", "Tuần 2", "Tuần 3", "Tuần 4"]*

*accuracy = [0.90, 0.89, 0.85, 0.80] # Accuracy giảm theo thời gian*

*plt.plot(dates, accuracy, marker='o', linestyle='-')*

*plt.xlabel("Thời gian")*

*plt.ylabel("Accuracy")*

*plt.title("Giám sát hiệu suất mô hình")*

*plt.show()*

* Thu thập phản hồi từ người dùng
* Cách hoạt động:
* Hỏi người dùng về trải nghiệm của họ với mô hình.
* Nếu mô hình lọc nội dung xấu (spam, ngôn từ độc hại), bạn có thể cho phép người dùng báo cáo sai sót.
* Ví dụ với chatbot AI:

Nếu nhiều người báo cáo chatbot trả lời sai, có thể cần cập nhật dữ liệu

huấn luyện.

* Lợi ích: Phát hiện lỗi sớm mà không cần đợi dữ liệu lớn.
  1. **Bảo trì mô hình (Maintain Your Model)** 
     1. Tại sao phải bảo trì mô hình?
* Dữ liệu thay đổi theo thời gian: Ví dụ, sở thích âm nhạc của người dùng thay đổi theo xu hướng, hoặc gian lận thẻ tín dụng xuất hiện các thủ thuật mới.
* Mô hình có thể mất hiệu suất: Một mô hình phân loại email spam có thể hoạt động tốt hôm nay nhưng sẽ kém hiệu quả sau vài tháng do email spam thay đổi nội dung.
* Dữ liệu mới có thể cải thiện mô hình: Nếu bạn thu thập thêm dữ liệu và huấn luyện lại, mô hình có thể hoạt động tốt hơn.

Ví dụ:

* Hệ thống gợi ý nhạc có thể cần cập nhật hàng tuần.
* Hệ thống phát hiện gian lận thẻ tín dụng có thể cần cập nhật hàng ngày.
* Công cụ tìm kiếm hình ảnh có thể hoạt động tốt trong vài năm trước khi cần cải tiến.
  + 1. Cách bảo trì mô hình hiệu quả
* Giám sát dữ liệu đầu vào
* Vấn đề: Nếu dữ liệu thực tế thay đổi so với dữ liệu huấn luyện, mô hình có thể mất hiệu quả.
* Giải pháp:
* Theo dõi đặc trưng quan trọng: Có đặc trưng nào mới xuất hiện không?
* Xác định dữ liệu có bị lệch không: Phân phối của dữ liệu có thay đổi đáng kể không?
* Ví dụ kiểm tra dữ liệu bằng Python:

*import numpy as np*

*import matplotlib.pyplot as plt*

*# Giả sử có dữ liệu huấn luyện cũ và mới*

*old\_data = np.random.normal(50, 10, 1000)*

*new\_data = np.random.normal(55, 12, 1000)*

*plt.hist(old\_data, alpha=0.5, label="Dữ liệu cũ")*

*plt.hist(new\_data, alpha=0.5, label="Dữ liệu mới")*

*plt.legend()*

*plt.title("So sánh phân phối dữ liệu cũ và mới")*

*plt.show()*

* Thu thập và gắn nhãn dữ liệu mới
* Vấn đề: Mô hình cần dữ liệu mới để cập nhật.
* Giải pháp:
* Tạo pipeline tự động thu thập dữ liệu mới.
* Chú ý đến các mẫu khó: Ví dụ, trong hệ thống phát hiện gian lận, hãy lưu lại các giao dịch mà mô hình không chắc chắn.
* Ví dụ tạo pipeline thu thập dữ liệu:

*def collect\_new\_data():*

*# Giả sử lấy dữ liệu từ API hoặc hệ thống ghi nhận*

*new\_samples = get\_new\_transactions()*

*return new\_samples*

* Huấn luyện lại mô hình định kỳ
* Vấn đề: Khi dữ liệu thay đổi, mô hình cần được cập nhật.
* Giải pháp:
* Huấn luyện lại theo lịch trình (hàng tháng, hàng quý, v.v.).
* Sử dụng kỹ thuật tiếp tục học (continual learning) thay vì đào tạo lại từ đầu.
* Ví dụ huấn luyện lại mô hình với dữ liệu mới:

*new\_data, new\_labels = collect\_new\_data()*

*model.fit(new\_data, new\_labels, epochs=5)*

* Kiểm tra hiệu suất sau khi cập nhật
* Vấn đề: Mô hình mới có thể không phải lúc nào cũng tốt hơn mô hình cũ.
* Giải pháp:
* Chạy A/B Testing: So sánh mô hình mới với mô hình cũ trên một tập dữ liệu kiểm tra.
* Theo dõi chỉ số hiệu suất: Nếu accuracy giảm, có thể cần điều chỉnh mô hình.
* Ví dụ kiểm tra mô hình mới trước khi triển khai:

*old\_predictions = old\_model.predict(test\_data)*

*new\_predictions = new\_model.predict(test\_data)*

*# So sánh độ chính xác*

*old\_accuracy = np.mean(old\_predictions == test\_labels)*

*new\_accuracy = np.mean(new\_predictions == test\_labels)*

*print(f"Old Accuracy: {old\_accuracy:.2f}, New Accuracy: {new\_accuracy:.2f}")*

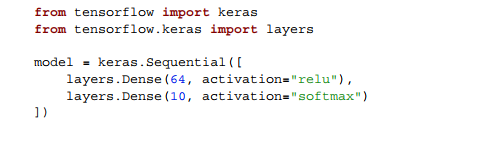
**PHẦN 3: LÀM VIỆC VỚI KERAS**

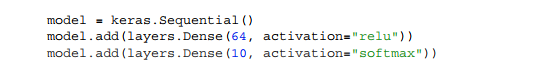
***1.Có ba API để xây dựng mô hình trong Keras***

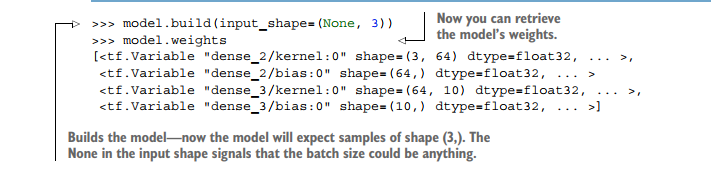
* **Mô hình Sequential**: Đây là API dễ tiếp cận nhất—về cơ bản, nó giống như một danh sách trong Python. Do đó, nó bị giới hạn trong việc xây dựng các mô hình theo dạng **xếp chồng các lớp tuần tự** một cách đơn giản.
* **Functional API**: Tập trung vào các kiến trúc mô hình có dạng **đồ thị**. Đây là một điểm cân bằng giữa **tính dễ sử dụng** và **tính linh hoạt**, và cũng là API phổ biến nhất để xây dựng mô hình.
* **Model subclassing**: Một tùy chọn **cấp thấp**, nơi bạn phải **tự viết mọi thứ từ đầu**. Điều này lý tưởng nếu bạn muốn **kiểm soát hoàn toàn** mọi khía cạnh của mô hình. Tuy nhiên, bạn sẽ **không thể tận dụng nhiều tính năng tích hợp sẵn của Keras** và cũng có nguy cơ **gặp lỗi cao hơn**.

**1.1 Mô hình tuần tự (The sequential model)**

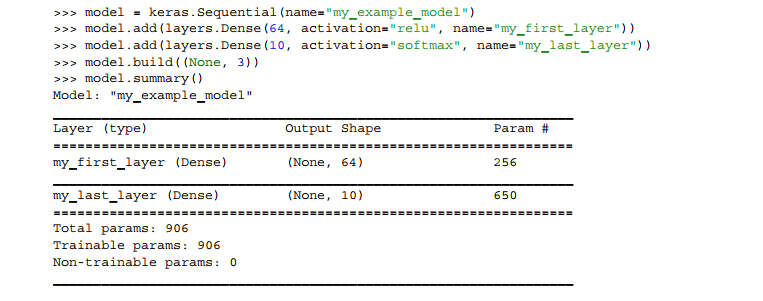
* ***The squential class***



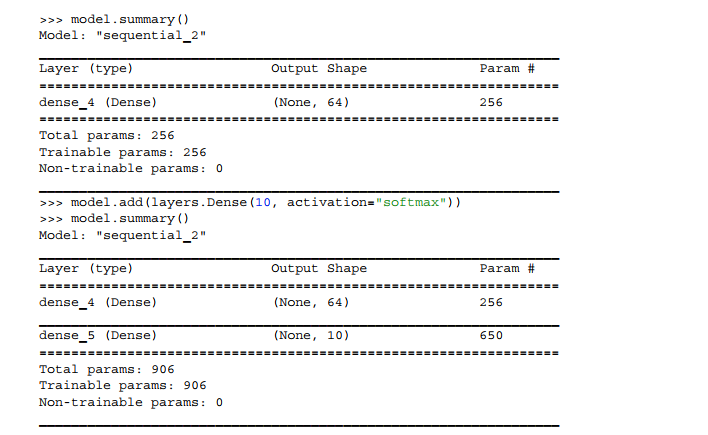
* **Incrementally building a Sequential model**Listing 7.2 Incrementally building a Sequential model  
  Listi
* **Calling a model for the first time to build it**



* **Naming models and layers with the name argument**



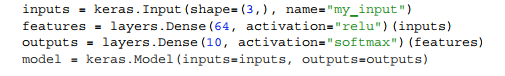
* **Specifying the input shape of your model in advance**



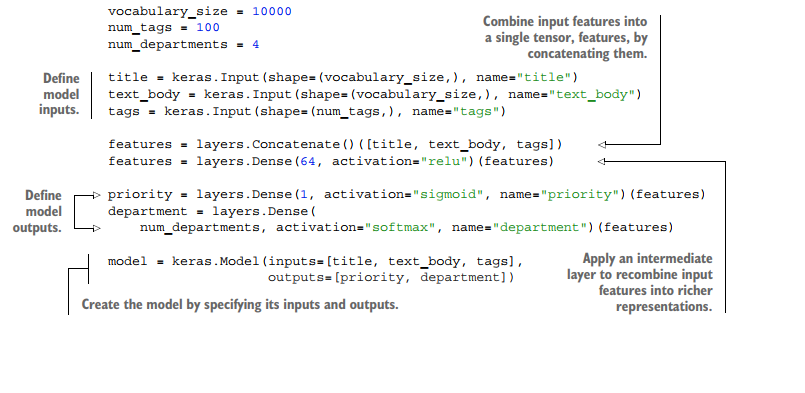
**1.2. The Funtional API**

**Functional API trong Keras là một cách linh hoạt để xây dựng mô hình mạng nơ-ron bằng cách kết nối các lớp (layers) theo cấu trúc dạng đồ thị thay vì chỉ đơn giản là xếp chồng lớp lên nhau như Sequential API.**

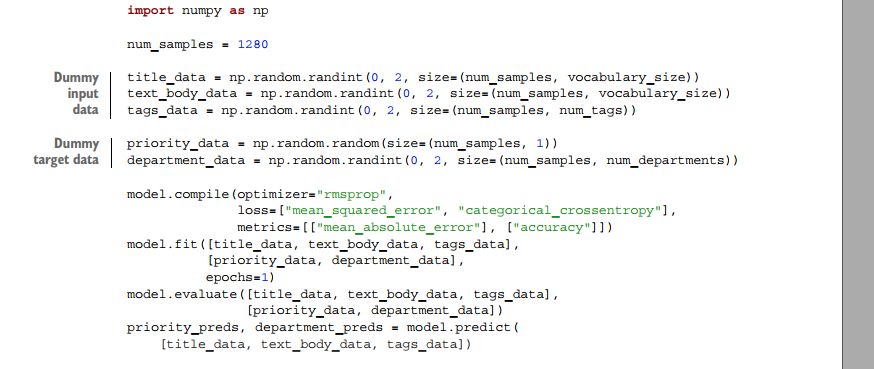
* **A simple Functional model with two Dense layers**

****

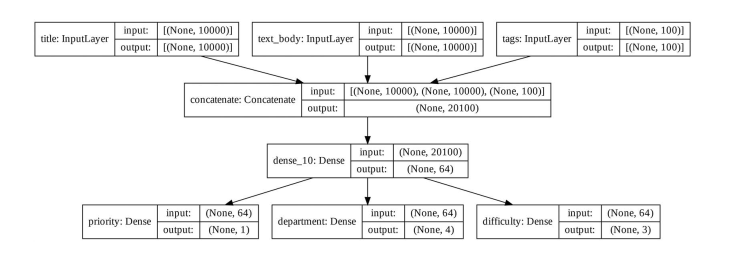
* **A multi-input, multi-output Functional model**

****

* **Training a model by providing litst of input and target arrays**

****

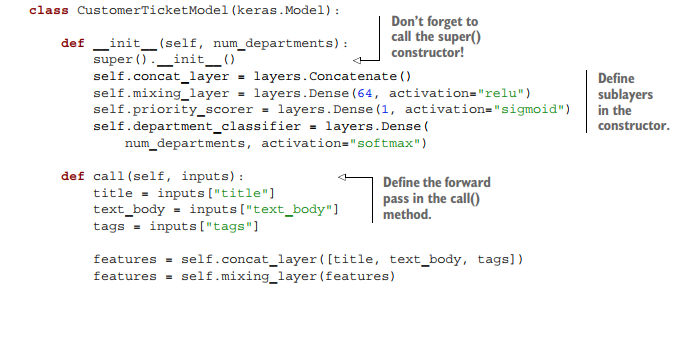
* **Creating a new model by reusing intermediate layer outputs**



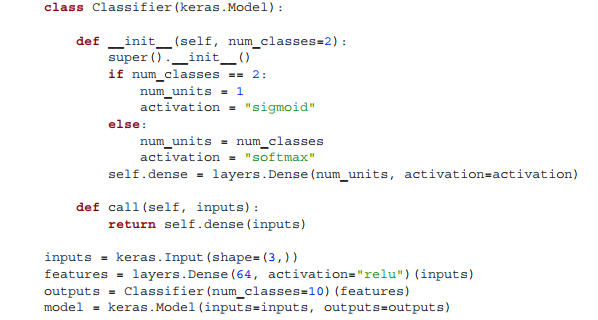
* 1. **Subclassing the Model class**

**Model subclassing là một cách tùy chỉnh mô hình trong Keras bằng cách kế thừa (**subclassing**) lớp** Model **và tự định nghĩa các thành phần của mạng nơ-ron. Phương pháp này cho phép bạn kiểm soát chi tiết từng bước trong quá trình xây dựng mô hình, nhưng cũng yêu cầu viết nhiều mã hơn so với Sequential API và Functional API.**

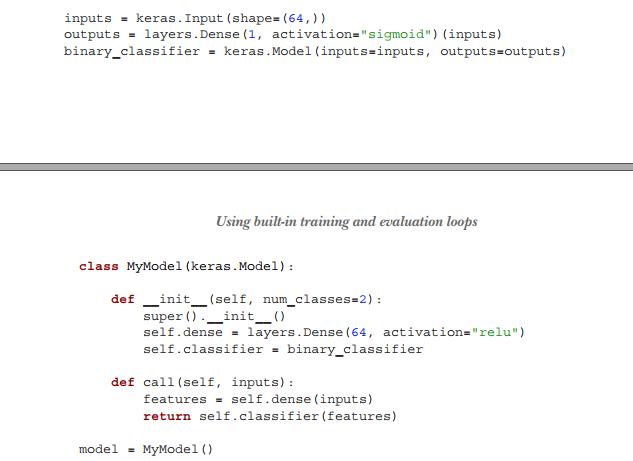
* **A simple subclassed model**

****

* **Creating a Functional model that includes a subclassed model**

****

* **Creating a subclassed model that includes a Functional model**

****

**2.** **Using built-in training and evaluation loops**

***Nguyên tắc tiết lộ dần sự phức tạp—cung cấp một loạt quy trình làm việc từ cực kỳ đơn giản đến linh hoạt tùy ý, từng bước một—cũng áp dụng cho quá trình huấn luyện mô hình. Keras cung cấp cho bạn nhiều cách khác nhau để huấn luyện mô hình. Bạn có thể chỉ cần gọi fit() trên dữ liệu của mình, hoặc nếu muốn nâng cao hơn, có thể tự viết một thuật toán huấn luyện mới từ đầu.***

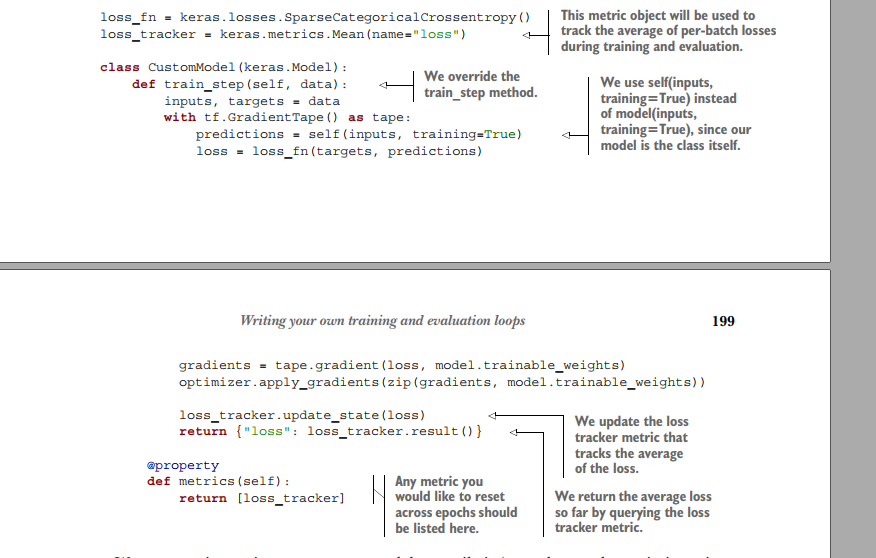
33.17 Listing 7.17 The standard workflow: **compile()**, **fit()**, **evaluate()**, **predict()**Listing 7.17 The standard workflow: **compile()**, **fit()**, **evaluate()**, **predict()**T Listing 7.17 The standard workflow: **compile()**, **fit()**, **evaluate()**, **predict()**he standard workflow: **compile()**, **fit()**, **evaluate()**, **predict()  
3.Using the callbacks argument in the fit() method**

****

### 4. Writing your own callbacks

### 

**5.Implementing a custom training step to use with fit()**

****

**Summary(Tóm tắt lại)**

Keras cung cấp một loạt các quy trình làm việc khác nhau, dựa trên nguyên tắc **tiết lộ dần sự phức tạp**. Tất cả các quy trình này có thể hoạt động cùng nhau một cách mượt mà.

* Bạn có thể xây dựng mô hình thông qua **lớp Sequential**, **API Functional**, hoặc **kế thừa lớp Model**. Hầu hết thời gian, bạn sẽ sử dụng **API Functional**.
* Cách đơn giản nhất để huấn luyện và đánh giá một mô hình là sử dụng các phương thức mặc định **fit()** và **evaluate()**.
* **Keras callbacks** cung cấp một cách đơn giản để theo dõi mô hình trong quá trình gọi **fit()** và tự động thực hiện các hành động dựa trên trạng thái của mô hình.
* Bạn cũng có thể kiểm soát hoàn toàn hoạt động của **fit()** bằng cách ghi đè phương thức **train\_step()**.
* Ngoài **fit()**, bạn còn có thể tự viết toàn bộ vòng lặp huấn luyện từ đầu. Điều này đặc biệt hữu ích cho các nhà nghiên cứu khi triển khai các thuật toán huấn luyện mới.

**-HẾT-**