یادگیری عمیق

جلسه ۱۱

مبانی یادگیری ماشینی (۱)

Fundamentals of Machine Learning (1)

کاظم فولادی قلعه

دانشکده مهندسی، دانشکدگان فارابی

دانشگاه تهران

http://courses.fouladi.ir/deep

یادگیری عمیق

مبانی یادگیری ماشینی

۱

یادگیری ماشینی

صورت‌های یادگیری ماشینی

FORMS OF MACHINE LEARNING

صورت‌های یادگیری	
یادگیری استقرائی *Inductive Learning*	**یادگیری استنباطی** *Deductive Learning*
یادگیری جزء به کل	**یادگیری کل به جزء**
یادگیری یک تابع یا قاعده‌ی عمومی (درست/نادرست) از روی جفت‌های خاص ورودی ـ خروجی / مثال‌ها	**یادگیری تحلیلی** *Analytical Learning*
	حرکت از یک قاعده‌ی عمومی شناخته‌شده به قاعده‌ی جدیدی که منطقاً استلزام می‌شود. (مفید است، زیرا امکان پردازش کارآمدتر را فراهم می‌کند.)

صورت‌های یادگیری ماشینی

یادگیری استقرائی

یادگیری استقرائی
Inductive Learning

یادگیری جزء به کل

یادگیری یک تابع یا قاعده‌ی عمومی (درست/نادرست)
از روی جفت‌های خاص ورودی ـ خروجی / مثال‌ها

یادگیری بانظارت *Supervised*	یادگیری بی‌نظارت *Unsupervised*	یادگیری تقویتی *Reinforcement*	یادگیری نیمه‌نظارتی *Semisupervised*	یادگیری خودنظارتی *Self-Supervised*
یادگیری نگاشت ورودی به خروجی با دیدن مثال‌های برچسب‌دار	یادگیری الگوهای درون ورودی بدون وجود فیدبک صریح (مثل clustering)	یادگیری از روی یک سری تقویت‌ها (پاداش‌ها و جریمه‌ها)	یادگیری با وجود تعداد کمی مثال برچسب‌دار و مجموعه‌ی بزرگی از داده‌های بی‌برچسب	یادگیری بانظارت بدون استفاده از برچسب‌های تهیه شده توسط انسان / برچسب‌ها از روی داده‌های ورودی تولید می‌شوند.

تقسیم‌بندی بر اساس **نوع فیدبک** موجود برای یادگیری

سه نوع اصلی یادگیری

شاخه‌های یادگیری ماشینی

یادگیری بانظارت

یادگیری بانظارت، رایج‌ترین شکل یادگیری ماشینی است:
یادگیری نگاشت داده‌های ورودی به تارگت‌های معلوم
بر اساس مجموعه‌ای از مثال‌های داده شده

عموماً، تقریباً تمامی کاربردهای درخشان یادگیری عمیق، از نوع یادگیری بانظارت هستند.

مانند:
بازشناسی نوری حروف، بازشناسی گفتار، طبقه‌بندی تصاویر، ترجمه‌ی زبان

شاخه‌های یادگیری ماشینی

یادگیری بانظارت

SUPERVISED LEARNING

یادگیری بانظارت
Supervised Learning

یک **مجموعه‌ی آموزشی** از N جفت ورودی–خروجی نمونه داده شده است:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

که در آن هر y_j به‌وسیله‌ی یک تابع مجهول f تولید شده است:

$$y = f(x)$$

یک تابع h را کشف کنید که تابع واقعی f را تقریب بزند.

شاخه‌های یادگیری ماشینی

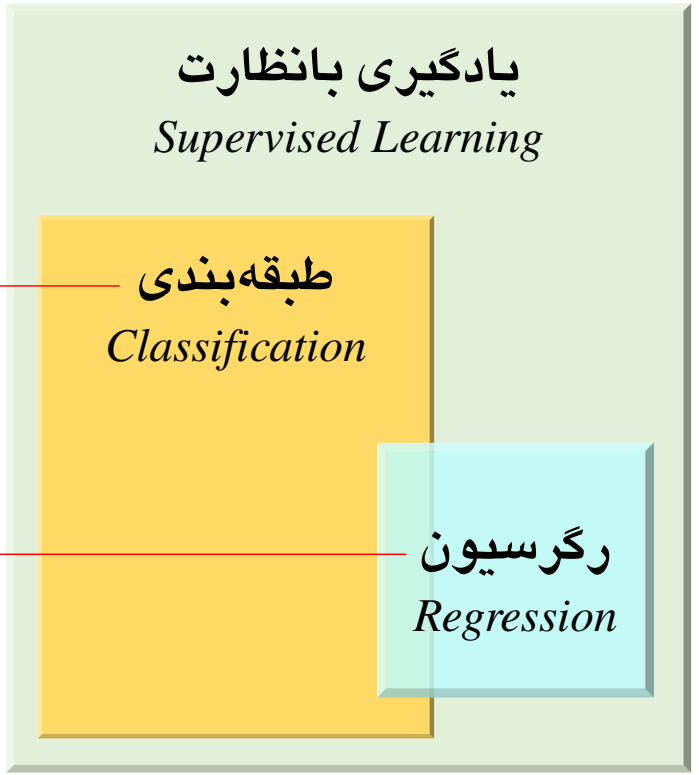
یادگیری بانظارت : مفاهیم کلیدی

یادگیری بانظارت *Supervised Learning*	
مجموعه‌ی آموزشی *Training Set*	مجموعه‌ی جفت‌های ورودی–خروجی معلوم برای آموزش یادگیرنده
مجموعه‌ی آزمایشی *Test Set*	مجموعه‌ی جفت‌های ورودی–خروجی معلوم برای آزمایش یادگیرنده
هدف / تارگت *Target*	تابع f که باید یاد گرفته شود
فرضیه *Hypothesis*	تابع مجهول h که باید تقریب مناسبی برای f باشد
فضای فرضیه *Hypothesis Space*	مجموعه‌ی همه‌ی توابع منتخب h برای تقریب f

$\mathcal{H}$

شاخه‌های یادگیری ماشینی

دو دسته‌ی مهم از مسائل یادگیری بانظارت : طبقه‌بندی و رگرسیون



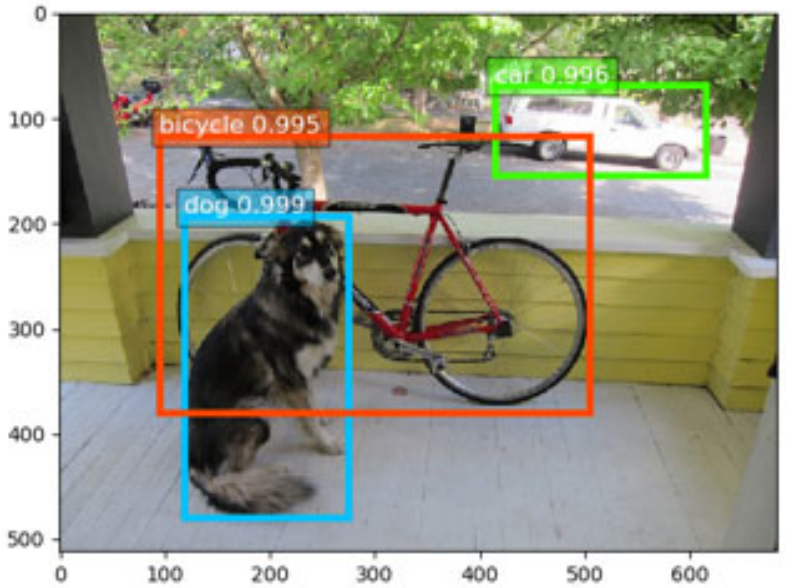
وقتی خروجی y از یک مجموعه‌ی متناهی مشخص انتخاب شود.

وقتی خروجی y یک عدد باشد (یافتن امید شرطی یا متوسط y)

شاخه‌های یادگیری ماشینی

نمونه‌هایی از کاربردهای یادگیری بانظارت

تولید دنباله *Sequence Generation*	**پیش‌بینی یک شرح برای توصیف یک تصویر داده شده** تولید دنباله اغلب می‌تواند به‌صورت یک سری از مسائل طبقه‌بندی فرمول‌بندی شود.
پیش‌بینی درخت نحو *Syntax Tree Prediction*	**پیش‌بینی تجزیه‌ی یک جمله‌ی داده شده به درخت نحو آن**
آشکارسازی اشیا *Object Detection*	**رسم یک چهارگوش دور اشیای خاص داخل یک تصویر داده شده** آشکارسازی اشیا می‌تواند به‌صورت یک مسئله‌ی طبقه‌بندی بیان شود (چهارگوش‌های کاندیدای متفاوت، طبقه‌بندی محتوای هر یک) یا مسئله‌ی طبقه‌بندی و رگرسیون توأم (رگرسیون برداری برای تعیین مختصات گوشه‌ها)
بخش‌بندی تصویر *Image Segmentation*	**رسم یک ماسک در سطح پیکسل بر روی یک شیئ خاص در یک تصویر داده شده**



آشکارسازی اشیا
Object Detection

بخش‌بندی تصویر
Image Segmentation

شاخه‌های یادگیری ماشینی

یادگیری بدون نظارت

UNSUPERVISED LEARNING

در یادگیری بدون نظارت،
یادگیری حاوی یافتن تبدیل‌های مفید داده‌های ورودی بدون کمک گرفتن از هرگونه تارگت است.

اهداف: بصری‌سازی داده‌ها، فشرده‌سازی داده‌ها، نویززدایی از داده‌ها، درک بهتر از همبستگی موجود در داده‌ها

یادگیری بی‌نظارت، معمولاً یک گام لازم برای فهم بهتر یک مجموعه داده
پیش از تلاش برای حل یک مسئله‌ی یادگیری با نظارت است.

کاهش ابعاد (dimensionality reduction) و **خوشه‌بندی** (clustering)
دو مسئله‌ی معروف در یادگیری بی‌نظارت هستند.

شاخه‌های یادگیری ماشینی

یادگیری خود نظارتی

SELF-SUPERVISED LEARNING

یادگیری خود نظارتی،
نوع خاصی از یادگیری با نظارت است که در آن از برچسب‌های تولید شده توسط انسان استفاده نمی‌شود.

در این نوع از یادگیری، برچسب وجود دارد،
اما برچسب‌ها از روی داده‌های ورودی و معمولاً با استفاده از یک الگوریتم هیوریستیک تولید می‌شوند.

برای مثال: خودکدگذارها (autoencoders). نمونه‌ی معروفی از یادگیری خودنظارتی هستند
که در آنها تارگت‌های تولید شده همان ورودی‌ها بدون هیچ تغییری هستند.

پیش‌بینی فریم بعدی در یک ویدئو با داشتن فریم‌های قبلی،
پیش‌بینی کلمه‌ی بعدی در یک متن با داشتن کلمات قبلی
یادگیری با نظارت زمانی (temporally supervised learning): داده‌های نظارت از آینده‌ی داده‌های ورودی می‌آیند.

شاخه‌های یادگیری ماشینی

یادگیری تقویتی

REINFORCEMENT LEARNING

یادگیری تقویتی،
عامل اطلاعاتی در مورد محیط خود دریافت می‌کند و
یاد می‌گیرد بهترین کنش‌هایی را انتخاب کند که پاداش او را حداکثر می‌سازد.

برای مثال:
یک شبکه‌ی عصبی که به صفحه‌ی یک بازی ویدئویی نگاه می‌کند و کنش‌های بازی را در خروجی تحویل می‌دهد،
به‌گونه‌ای که امتیاز آن حداکثر شود، می‌تواند با یادگیری تقویتی آموزش داده شود.

در حال حاضر یادگیری تقویتی بیشتر یک زمینه‌ی پژوهشی است و خارج از زمینه‌ی بازی‌ها
(مانند یادگیری بازی‌های آتاری [Google DeepMind] یا بازی Go)
موفقیت عملی چشمگیری نداشته است.

انتظار می‌رود موفقیت یادگیری تقویتی در حوزه‌ی گسترده‌ای از کاربردهای دنیای واقعی دیده شود، مانند:
خودروهای خودران، رباتیک، مدیریت منابع، تربیت و ...

یادگیری عمیق

مبانی یادگیری ماشینی

۲

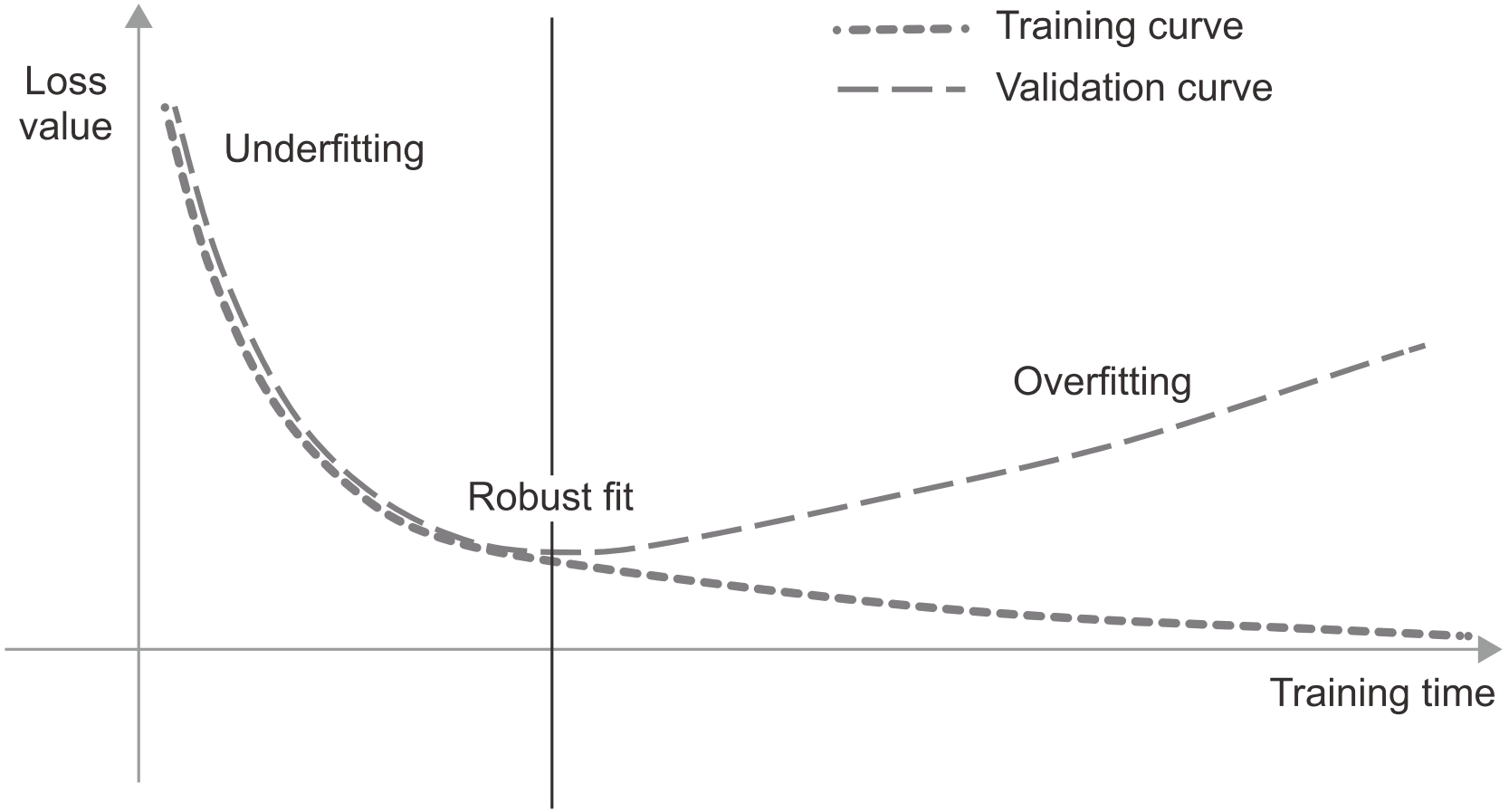
تعمیم: هدف یادگیری ماشینی

تعمیم: هدف یادگیری ماشینی

GENERALIZATION: THE GOAL OF MACHINE-LEARNING

هدف در یادگیری ماشینی، دستیابی به مدل‌هایی است که **قابلیت تعمیم (generalization)** داشته باشند
[یعنی بتوانند بر روی داده‌هایی که هرگز آنها را ندیده‌اند، به‌خوبی عمل کنند]
و **بیش‌برازش (overfitting)** مانع اصلی در قبال این موضوع است.

برازش: نقطه میان بیش‌برازش و کم‌برازش



وقوع بیش‌برازش در شرایط داده‌های آموزشی نویزی

NOISY TRAINING DATA

وقتی داده‌های آموزشی، نویزی باشند / حاوی عدم اطمینان باشند / یا حاوی ویژگی‌های اندکی باشند، وقوع **بیش‌برازش** محتمل است.

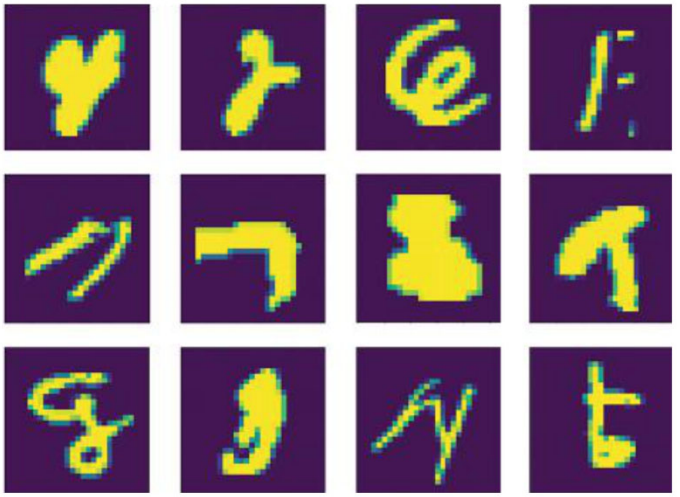


Figure 5.2 Some pretty weird MNIST training samples

Figure 5.3 Mislabeled MNIST training samples

برازش مقاوم با تشخیص بیرون‌افتاده‌ها

OUTLIERS DETECTION

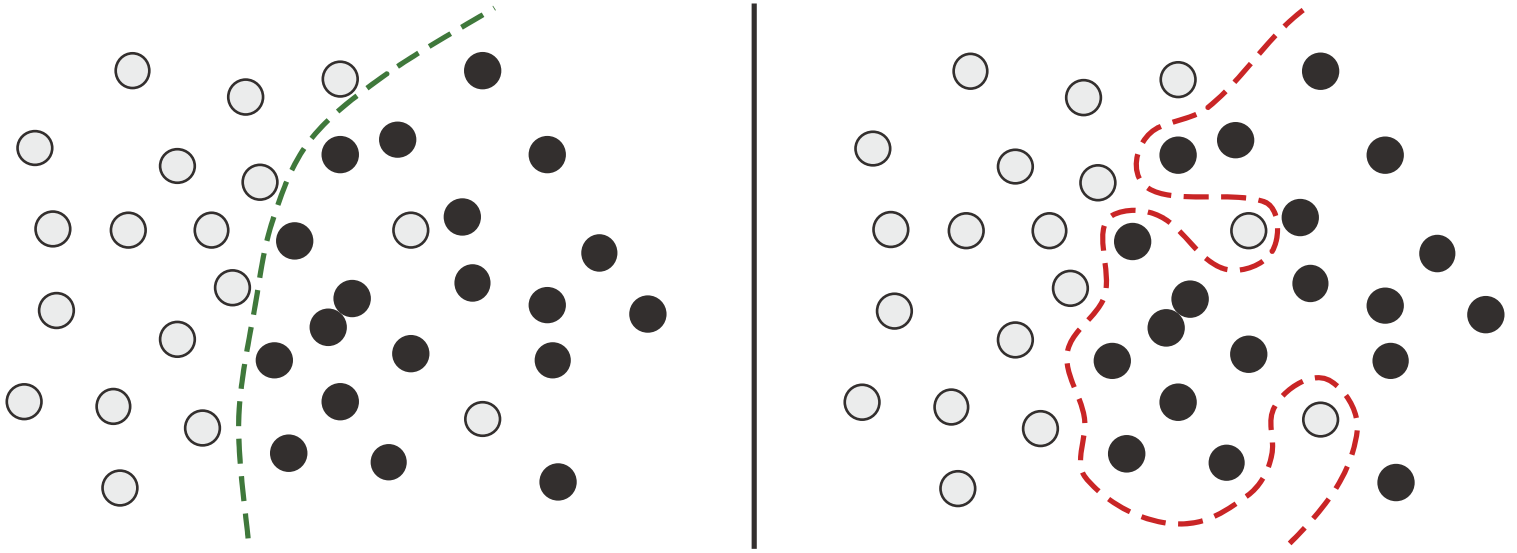


Figure 5.4 Dealing with outliers: robust fit vs. overfitting

برازش مقاوم با تشخیص ناحیه‌های مبهم فضای ویژگی

AMBIGUOUS REGIONS OF THE FEATURE SPACE

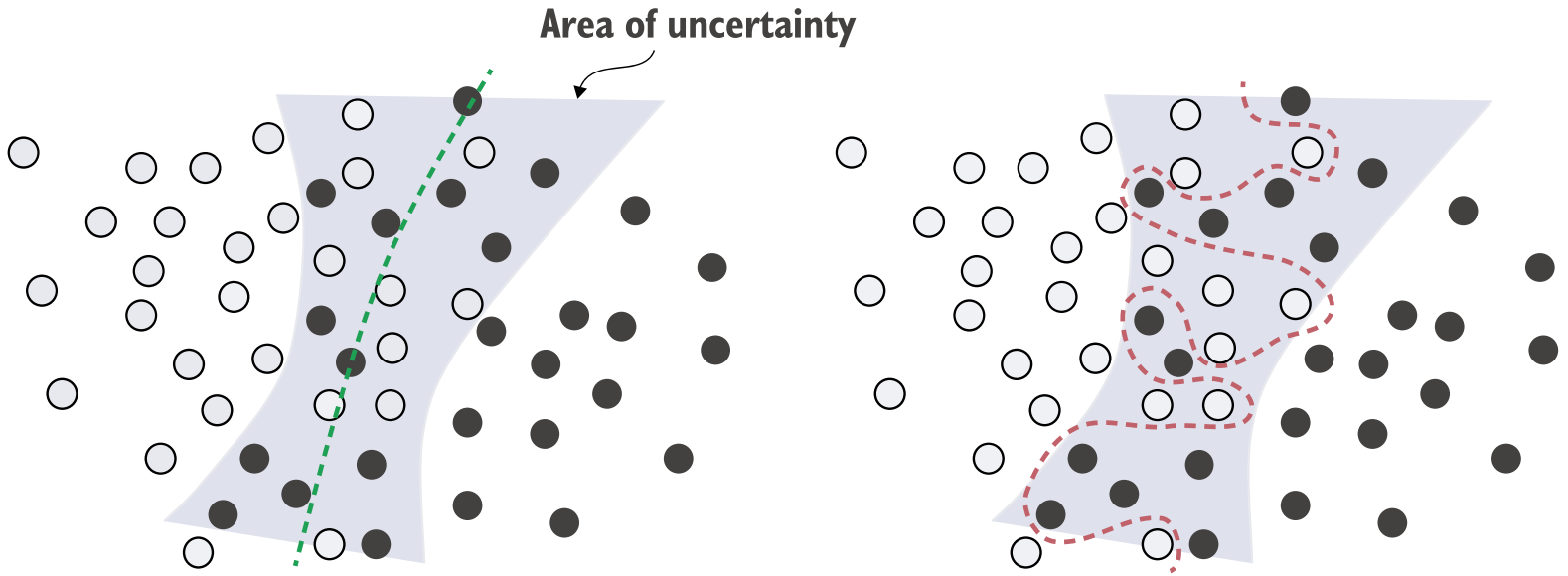


Figure 5.5 Robust fit vs. overfitting giving an ambiguous area of the feature space

Listing 5.1 Adding white noise channels or all-zeros channels to MNIST

```
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

train_images_with_noise_channels = np.concatenate(
    [train_images, np.random.random((len(train_images), 784))], axis=1)

train_images_with_zeros_channels = np.concatenate(
    [train_images, np.zeros((len(train_images), 784))], axis=1)
```

Listing 5.2 Training the same model on MNIST data with noise channels or all-zero channels

```
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

model = get_model()
history_noise = model.fit(
    train_images_with_noise_channels, train_labels,
    epochs=10,
    batch_size=128,
    validation_split=0.2)

model = get_model()
history_zeros = model.fit(
    train_images_with_zeros_channels, train_labels,
    epochs=10,
    batch_size=128,
    validation_split=0.2)
```

Listing 5.3 Plotting a validation accuracy comparison

```
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()
```

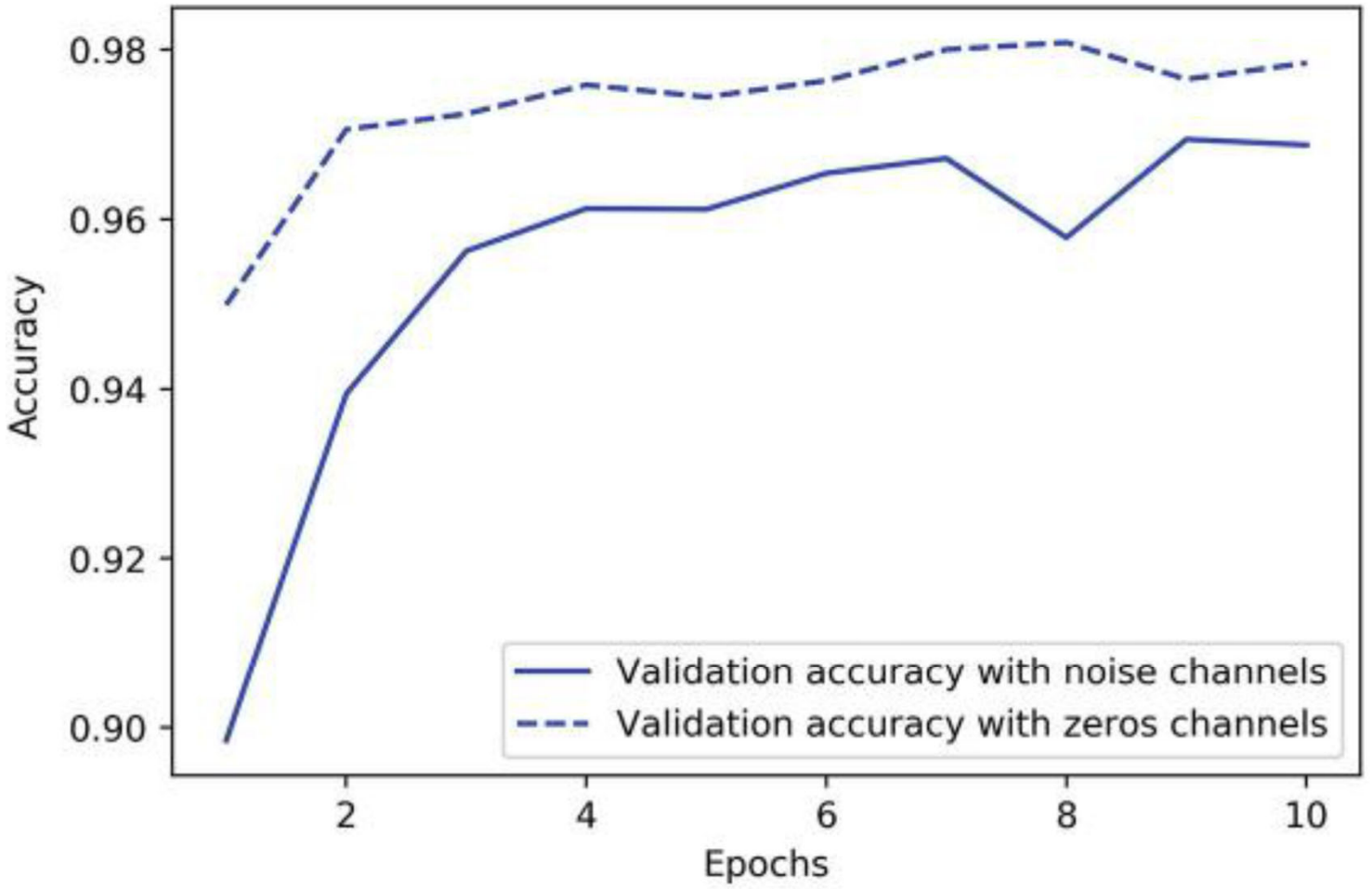


Figure 5.6 Effect of noise channels on validation accuracy

ماهیت تعمیم در یادگیری عمیق

فرض منیفولد: درون‌یابی به‌عنوان منبع تعمیم در یادگیری عمیق

THE NATURE OF GENERALIZATION IN DEEP LEARNING

The manifold hypothesis implies that

- ✓ Machine learning models only have to fit relatively simple, low-dimensional, highly structured subspaces within their potential input space (*latent manifolds*).
- ✓ Within one of these manifolds, it's always possible to *interpolate* between two inputs, that is to say, morph one into another via a continuous path along which all points fall on the manifold.

The ability to interpolate between samples is the **key** to understanding generalization in deep learning.

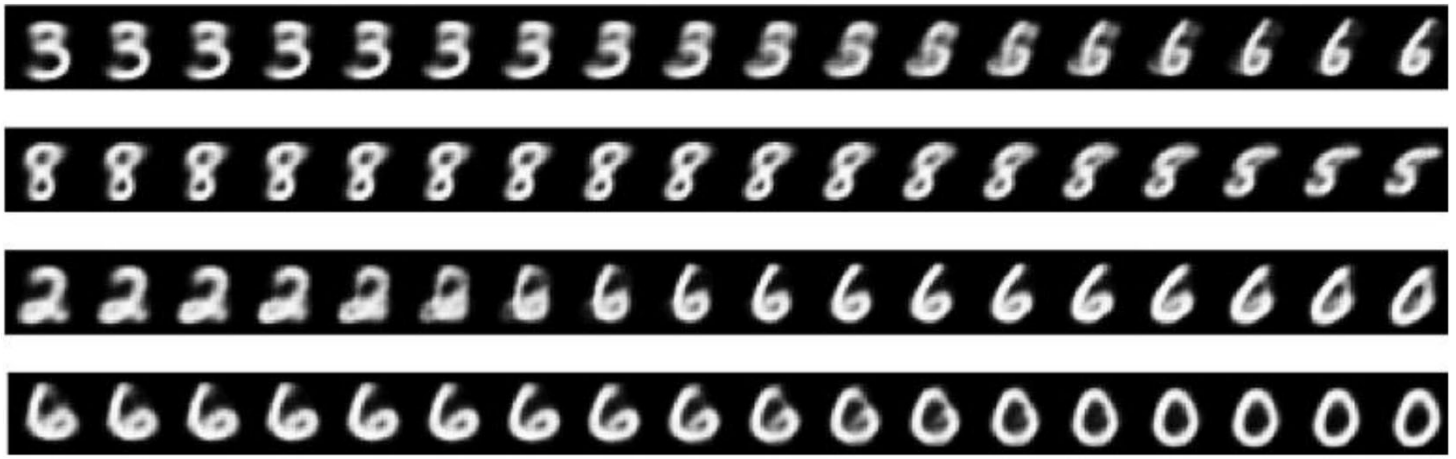


Figure 5.7 Different MNIST digits gradually morphing into one another, showing that the space of handwritten digits forms a "manifold." This image was generated using code from chapter 12.

Listing 5.4 Fitting an MNIST model with randomly shuffled labels

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

random_train_labels = train_labels[:]
np.random.shuffle(random_train_labels)

model = keras.Sequential([
    layers.Dense(512, activation="relu"),
                     layers.Dense(10, activation="softmax")
                 ])
                 model.compile(optimizer="rmsprop",
                               loss="sparse_categorical_crossentropy",
                               metrics=["accuracy"])
                 model.fit(train_images, random_train_labels,
                           epochs=100,
                           batch_size=128,
                           validation_split=0.2)
```

ماهیت تعمیم در یادگیری عمیق

فرض منیفولد: درون‌یابی به‌عنوان منبع تعمیم در یادگیری عمیق

THE NATURE OF GENERALIZATION IN DEEP LEARNING

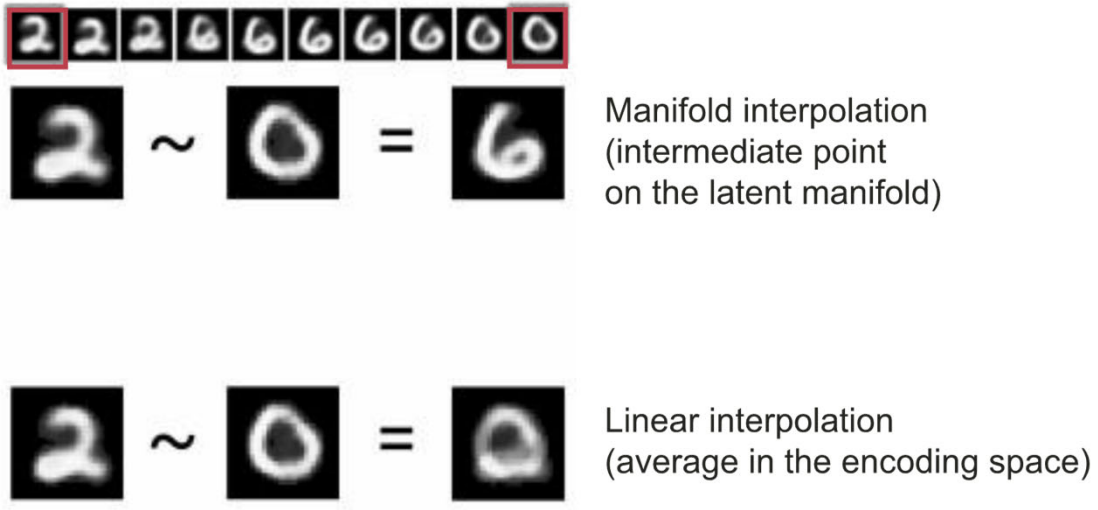


Figure 5.8 Difference between linear interpolation and interpolation on the latent manifold. Every point on the latent manifold of digits is a valid digit, but the average of two digits usually isn't.

چرا یادگیری عمیق کار می‌کند؟

WHY DEEP LEARNING WORKS

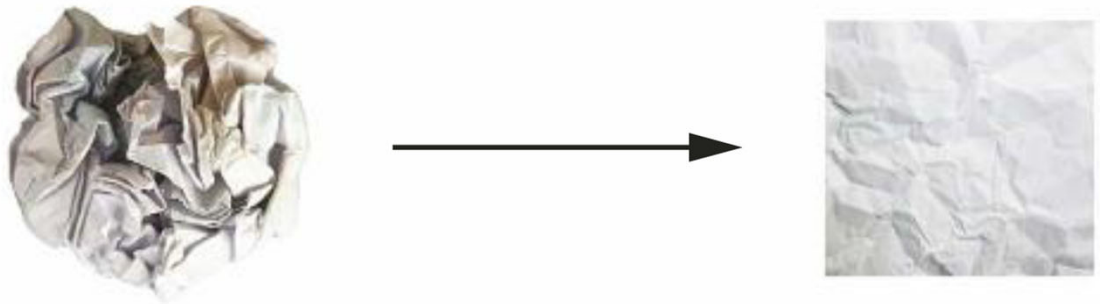


Figure 5.9 Uncrumpling a complicated manifold of data

چرا یادگیری عمیق کار می‌کند؟

WHY DEEP LEARNING WORKS

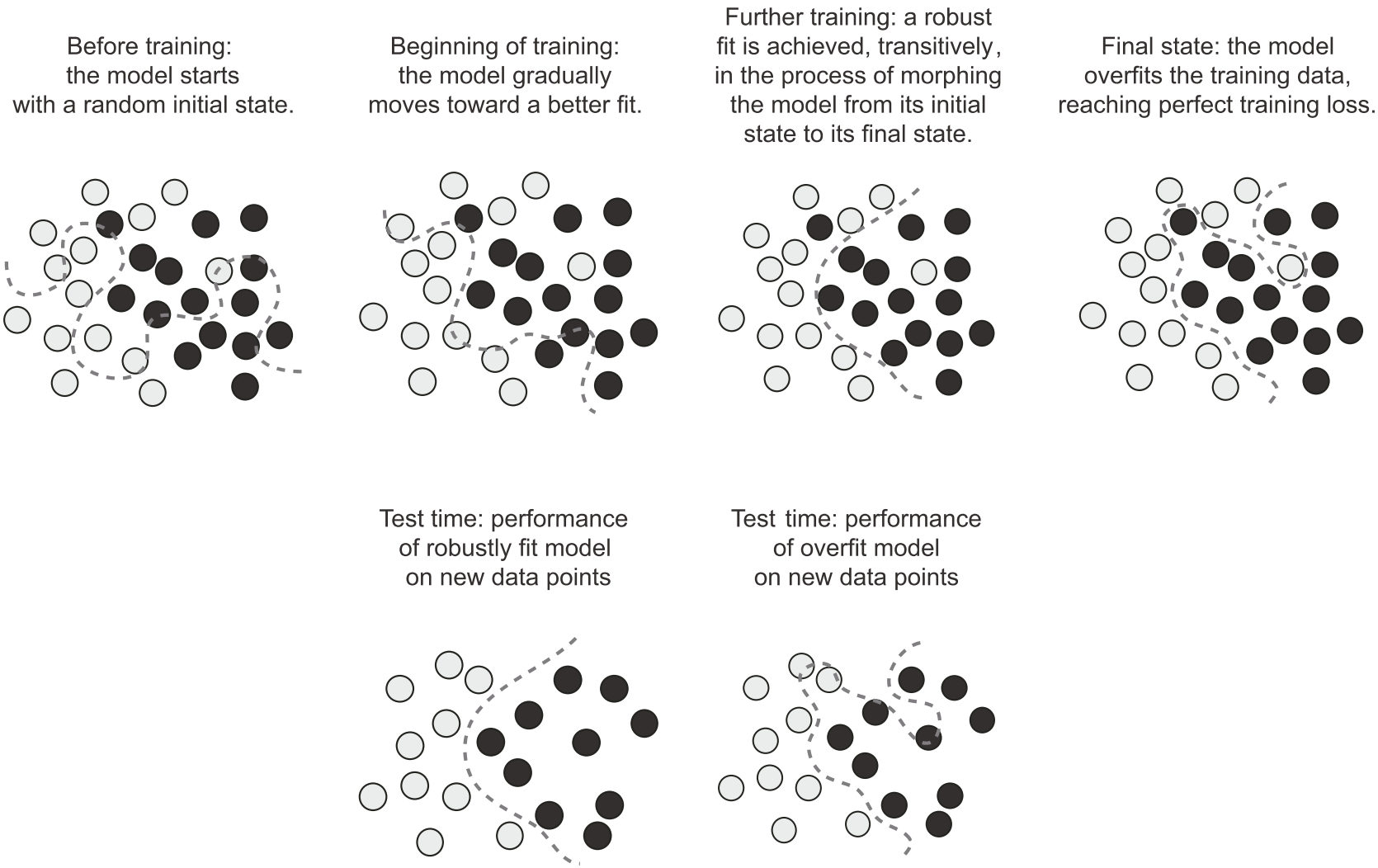


Figure 5.10 Going from a random model to an overfit model, and achieving a robust fit as an intermediate state

چرا یادگیری عمیق کار می‌کند؟

WHY DEEP LEARNING WORKS

Besides the trivial fact that they have sufficient representational power, there are a few properties of deep learning models that make them particularly well-suited to learning latent manifolds:

- Deep learning models implement a smooth, continuous mapping from their inputs to their outputs. It has to be smooth and continuous because it must be differentiable, by necessity (you couldn't do gradient descent otherwise).
  This smoothness helps approximate latent manifolds, which follow the same properties.

- Deep learning models tend to be structured in a way that mirrors the "shape" of the information in their training data (*via architecture priors*). This is particularly the case for image-processing models (discussed in chapters 8 and 9) and sequence-processing models (chapter 10). More generally, deep neural networks structure their learned representations in a hierarchical and modular way, which echoes the way natural data is organized.

چرا یادگیری عمیق کار می‌کند؟

WHY DEEP LEARNING WORKS

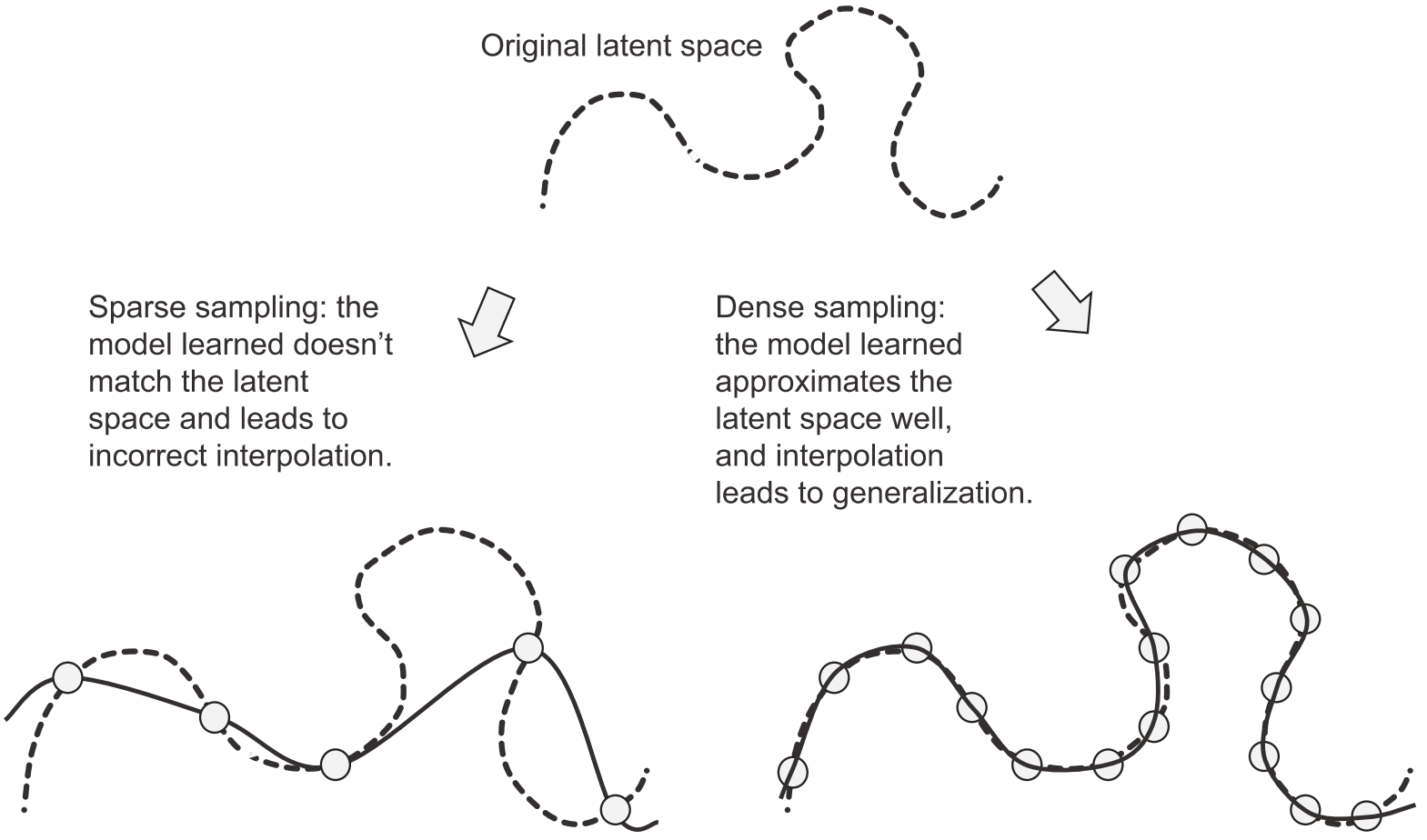


Figure 5.11 A dense sampling of the input space is necessary in order to learn a model capable of accurate generalization.

یادگیری عمیق

مبانی یادگیری ماشینی

۳

ارزیابی مدل‌های یادگیری ماشینی

ارزیابی مدل‌های یادگیری ماشینی

EVALUATING MACHINE-LEARNING MODELS

هدف در یادگیری ماشینی، دستیابی به مدل‌هایی است که **قابلیت تعمیم (generalization)** داشته باشند
[یعنی بتوانند بر روی داده‌هایی که هرگز آنها را ندیده‌اند، به‌خوبی عمل کنند]
و **بیش‌برازش (overfitting)** مانع اصلی در قبال این موضوع است.

از آنجا که ما تنها می‌توانیم چیزی را کنترل کنیم که بتوانیم آن را مشاهده کنیم،
ضروری است که قادر باشیم قدرت تعمیم یک مدل را به‌طور مطمئن اندازه‌گیری کنیم.

⇓

چگونگی اندازه‌گیری تعمیم ⇐ چگونگی ارزیابی یک مدل یادگیری ماشین

ارزیابی مدل‌های یادگیری ماشینی

مجموعه‌های آموزشی، اعتبارسنجی، آزمایشی

TRAINING, VALIDATION, AND TEST SETS

مجموعه‌ی داده‌ها را به سه زیرمجموعه تقسیم می‌کنیم:

فرآیند آموزش روی **داده‌های آموزشی** انجام می‌شود.
مدل بر اساس **داده‌های اعتبارسنجی** ارزیابی می‌شود.
پس از آماده شدن مدل، برای آخرین بار روی **داده‌های آزمایشی** مورد آزمون و بررسی قرار می‌گیرد.

ارزیابی مدل‌های یادگیری ماشینی

چرا فقط از دو مجموعه‌ی آموزشی و آزمایشی استفاده نکنیم؟

چرا فقط از دو مجموعه‌ی آموزشی و آزمایشی استفاده نکنیم؟

زیرا:
توسعه‌ی یک مدل همیشه حاوی **تنظیم پیکربندی** آن است،
(مانند: انتخاب تعداد لایه‌ها، اندازه‌ی لایه‌ها، ... [هایپرپارامترها: hyperparameters])

تنظیم این پارامترها با استفاده از یک سیگنال فیدبک کارآیی شبکه روی داده‌های اعتبارسنجی انجام می‌شود.
خود این تنظیم نوعی یادگیری است که با استفاده از داده‌های اعتبارسنجی انجام شده است.

نشت اطلاعاتی
Information Leaks

هرگاه هایپرپارامتری از مدل را بر اساس کارآیی آن روی مجموعه‌ی اعتبارسنجی تنظیم می‌کنیم، مقداری اطلاعات در مورد داده‌های اعتبارسنجی به مدل نشت می‌کند.

بنابراین باید از یک مجموعه داده‌ی کاملاً متفاوت که قبلاً هرگز مشاهده نشده است برای آزمایش مدل استفاده کنیم:
مجموعه‌ی آزمایشی.
مدل نباید به هیچ گونه اطلاعاتی در مورد این مجموعه‌ی آزمایشی دسترسی داشته باشد (حتی به‌طور غیرمستقیم).

ارزیابی مدل‌های یادگیری ماشینی

روش‌های اعتبارسنجی

تقسیم داده‌ها به مجموعه‌های آموزشی، اعتبارسنجی، آزمایشی به‌نظر سرراست می‌رسد،
اما روش‌های پیشرفته‌ای وجود دارد که زمانی که داده‌های کمی موجود است مفید واقع می‌شوند.

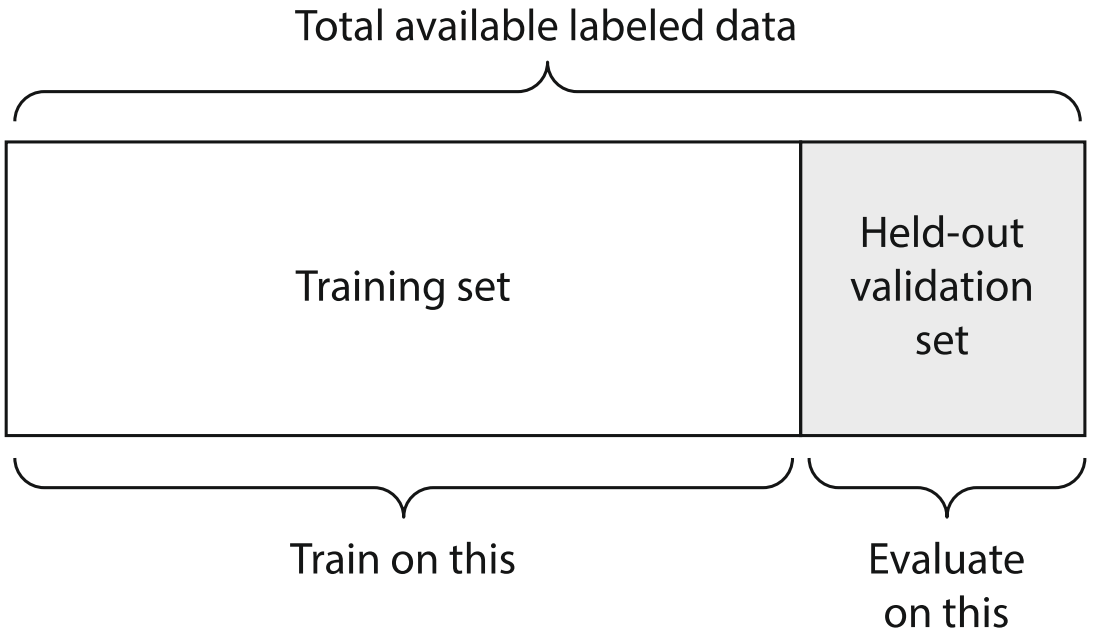
- **اعتبارسنجی ساده نگه‌داشت–برون‌گذاشت**
  Simple Hold-Out Validation
- **اعتبارسنجی *K-fold***
  K-fold Validation
- **اعتبارسنجی *K-fold* تکراری با بُر زدن**
  Iterated K-fold Validation with Shuffling

ارزیابی مدل‌های یادگیری ماشینی

اعتبارسنجی ساده نگه‌داشت–برون‌گذاشت

Simple Hold-Out Validation

بخشی از داده‌های آموزشی را به‌منظور اعتبارسنجی جدا می‌کنیم.



این ساده‌ترین پروتکل ارزیابی است و از یک نقیصه رنج می‌برد:
اگر داده‌های اندکی موجود باشد، آن‌گاه مجموعه‌های اعتبارسنجی و آزمایشی ممکن است حاوی تعداد نمونه‌های بسیار کمی برای نمایندگی آماری داده‌ها باشند.

ارزیابی مدل‌های یادگیری ماشینی

اعتبارسنجی ساده نگه‌داشت–برون‌گذاشت

Listing 5.5 Holdout validation (note that labels are omitted for simplicity)

```
num_validation_samples = 10000
np.random.shuffle(data)
validation_data = data[:num_validation_samples]
training_data = data[num_validation_samples:]
model = get_model()
model.fit(training_data, ...)
validation_score = model.evaluate(validation_data, ...)

...

model = get_model()
model.fit(np.concatenate([training_data,
                          validation_data]), ...)
test_score = model.evaluate(test_data, ...)
```

Shuffling the data is usually appropriate.

Defines the validation set

Defines the training set

Trains a model on the training data, and evaluates it on the validation data

At this point you can tune your model, retrain it, evaluate it, tune it again.

Once you've tuned your hyperparameters, it's common to train your final model from scratch on all non-test data available.

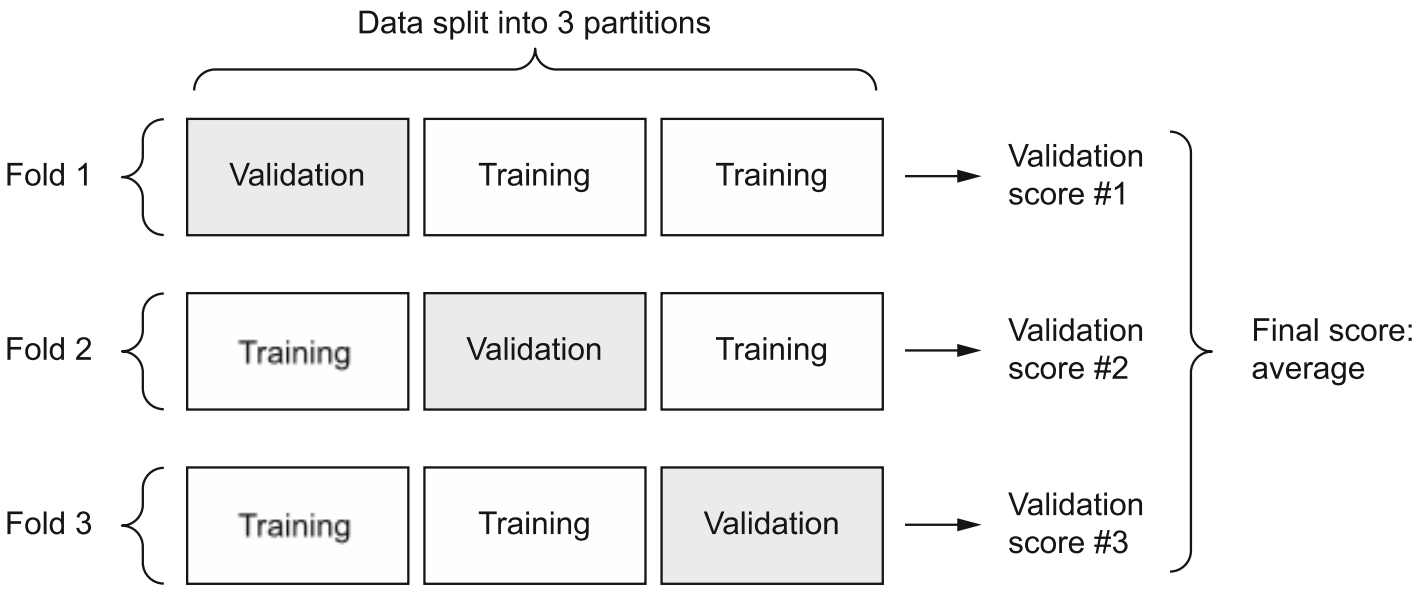
ارزیابی مدل‌های یادگیری ماشینی

اعتبارسنجی K-fold

K-FOLD VALIDATION

داده‌های موجود را به K سلول با اندازه‌ی مساوی افراز می‌کنیم (معمولاً $K = 4,5$)،
برای هر سلول i، یک مدل را بر روی $K-1$ سلول باقیمانده آموزش می‌دهیم
و آن را روی سلول i ارزیابی می‌کنیم.

امتیاز نهایی اعتبارسنجی مدل = متوسط K امتیاز اعتبارسنجی حاصل.



این روش زمانی کمک کننده است که کارآیی مدل ما واریانس بالایی بر اساس تقسیم آموزش–آزمایش داشته باشد.
این روش مانند روش ساده، ما را از به‌کارگیری یک مجموعه‌ی اعتبارسنجی مجزا برای کالیبراسیون مدل معاف نمی‌کند.

ارزیابی مدل‌های یادگیری ماشینی

اعتبارسنجی K-fold

Listing 5.6 K-fold cross-validation (note that labels are omitted for simplicity)

```
k = 3
num_validation_samples = len(data) // k
np.random.shuffle(data)
validation_scores = []
for fold in range(k):
    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):])
    model = get_model()
    model.fit(training_data, ...)
    validation_score = model.evaluate(validation_data, ...)
    validation_scores.append(validation_score)
validation_score = np.average(validation_scores)
model = get_model()
model.fit(data, ...)
test_score = model.evaluate(test_data, ...)
```

Selects the validation-data partition

Creates a brand-new instance of the model (untrained)

Validation score: average of the validation scores of the k folds

Trains the final model on all non-test data available

Uses the remainder of the data as training data. Note that the + operator represents list concatenation, not summation.

ارزیابی مدل‌های یادگیری ماشینی

اعتبارسنجی K-fold تکراری با بُر زدن

ITERATED K-FOLD VALIDATION WITH SHUFFLING

استفاده از اعتبارسنجی K-fold به‌تعداد P مرتبه
و بُر زدن (shuffling) داده‌ها در هر مرتبه پیش از تقسیم آن به K قسمت.

امتیاز نهایی = متوسط امتیازهای کسب شده در هر مرتبه از اجرای اعتبارسنجی K-fold

در پایان تعداد $P \times K$ مدل تحت آموزش و ارزیابی قرار گرفته است،
که می‌تواند بسیار پرهزینه باشد.

این روش مختص موقعیت‌هایی است داده‌های اندکی در اختیار داریم
و نیاز داریم مدل با حداکثر دقت ممکن ارزیابی شود.

ارزیابی مدل‌های یادگیری ماشینی

نکات مهم

نمایندگی داده‌ها
Data Representativeness

هم مجموعه‌ی آموزشی و هم مجموعه‌ی آزمایشی باید نماینده‌ی همه‌ی داده‌های موجود باشند (از همه‌ی کلاس‌ها به تعداد کافی در هر دو موجود باشد).

⇐ لازم است پیش از تقسیم داده‌ها بین مجموعه‌های آموزش و آزمایش آنها را به‌صورت تصادفی بر بزنیم (randomly shuffle).

پیکان زمان
The Arrow of Time

اگر هدف حل یک مسئله‌ی پیش‌بینی آینده بر اساس گذشته است، نباید پیش از تقسیم داده‌ها آنها را بر بزنیم و ترتیب آنها تصادفاً تغییر دهیم (این کار موجب نشت زمانی (temporal leak) می‌شود و باعث می‌شود مدل بر روی داده‌هایی از آینده آموزش ببیند).

در چنین مواردی باید مطمئن بود که همه‌ی داده‌های آزمایشی، از نظر زمانی پس از داده‌های آموزشی قرار داشته باشند.

افزونگی در داده‌ها
Redundancy in the Data

اگر برخی نقاط داده‌ای چند بار ظاهر شده باشند، بر زدن آنها و تقسیم آنها به مجموعه‌های آموزشی و اعتبارسنجی، موجب بروز افزونگی بین این دو مجموعه می‌شود. در نتیجه، تست روی بخشی از داده‌های آموزش انجام می‌شود (بدترین کار ممکن).

باید مطمئن باشیم که مجموعه‌های آموزشی و اعتبارسنجی مجزا باشند.

یادگیری عمیق

مبانی یادگیری ماشینی

۴

بهبود برازش مدل

بهبود برازش مدل

IMPROVING MODEL FIT

5.3 Improving model fit

To achieve the perfect fit, you must first overfit. Since you don't know in advance where the boundary lies, you must cross it to find it. Thus, your initial goal as you start working on a problem is to achieve a model that shows some generalization power and that is able to overfit. Once you have such a model, you'll focus on refining generalization by fighting overfitting.

There are three common problems you'll encounter at this stage:

- Training doesn't get started: your training loss doesn't go down over time.
- Training gets started just fine, but your model doesn't meaningfully generalize: you can't beat the common-sense baseline you set.
- Training and validation loss both go down over time, and you can beat your baseline, but you don't seem to be able to overfit, which indicates you're still underfitting.

Let's see how you can address these issues to achieve the first big milestone of a machine learning project: getting a model that has some generalization power (it can beat a trivial baseline) and that is able to overfit.

5.3.1 Tuning key gradient descent parameters

Sometimes training doesn't get started, or it stalls too early. Your loss is stuck. This is *always* something you can overcome: remember that you can fit a model to random data. Even if nothing about your problem makes sense, you should *still* be able to train something—if only by memorizing the training data.

When this happens, it's always a problem with the configuration of the gradient descent process: your choice of optimizer, the distribution of initial values in the weights of your model, your learning rate, or your batch size. All these parameters are interdependent, and as such it is usually sufficient to tune the learning rate and the batch size while keeping the rest of the parameters constant.

Let's look at a concrete example: let's train the MNIST model from chapter 2 with an inappropriately large learning rate of value 1.

Listing 5.7 Training an MNIST model with an incorrectly high learning rate

```
(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)
```

The model quickly reaches a training and validation accuracy in the 30%–40% range, but cannot get past that. Let's try to lower the learning rate to a more reasonable value of `1e-2`.

Listing 5.8 The same model with a more appropriate learning rate

```
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)
```

The model is now able to train.

If you find yourself in a similar situation, try

- Lowering or increasing the learning rate. A learning rate that is too high may lead to updates that vastly overshoot a proper fit, like in the preceding example, and a learning rate that is too low may make training so slow that it appears to stall.
- Increasing the batch size. A batch with more samples will lead to gradients that are more informative and less noisy (lower variance).

You will, eventually, find a configuration that gets training started.

5.3.2 Leveraging better architecture priors

You have a model that fits, but for some reason your validation metrics aren't improving at all. They remain no better than what a random classifier would achieve: your model trains but doesn't generalize. What's going on?

This is perhaps the worst machine learning situation you can find yourself in. It indicates that *something is fundamentally wrong with your approach*, and it may not be easy to tell what. Here are some tips.

First, it may be that the input data you're using simply doesn't contain sufficient information to predict your targets: the problem as formulated is not solvable. This is what happened earlier when we tried to fit an MNIST model where the labels were shuffled: the model would train just fine, but validation accuracy would stay stuck at 10%, because it was plainly impossible to generalize with such a dataset.

It may also be that the kind of model you're using is not suited for the problem at hand. For instance, in chapter 10, you'll see an example of a timeseries prediction problem where a densely connected architecture isn't able to beat a trivial baseline, whereas a more appropriate recurrent architecture does manage to generalize well. Using a model that makes the right assumptions about the problem is essential to achieve generalization: you should leverage the right architecture priors.

In the following chapters, you'll learn about the best architectures to use for a variety of data modalities—images, text, timeseries, and so on. In general, you should always make sure to read up on architecture best practices for the kind of task you're attacking—chances are you're not the first person to attempt it.

5.3.3 Increasing model capacity

If you manage to get to a model that fits, where validation metrics are going down, and that seems to achieve at least some level of generalization power, congratulations: you're almost there. Next, you need to get your model to start overfitting.

Consider the following small model—a simple logistic regression—trained on MNIST pixels.

Listing 5.9 A simple logistic regression on MNIST

```
model = keras.Sequential([layers.Dense(10, activation="softmax")])
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
history_small_model = model.fit(
    train_images, train_labels,
    epochs=20,
    batch_size=128,
    validation_split=0.2)
```

You get loss curves that look like figure 5.14:

```
import matplotlib.pyplot as plt
val_loss = history_small_model.history["val_loss"]
epochs = range(1, 21)
plt.plot(epochs, val_loss, "b--",
         label="Validation loss")
plt.title("Effect of insufficient model capacity on validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

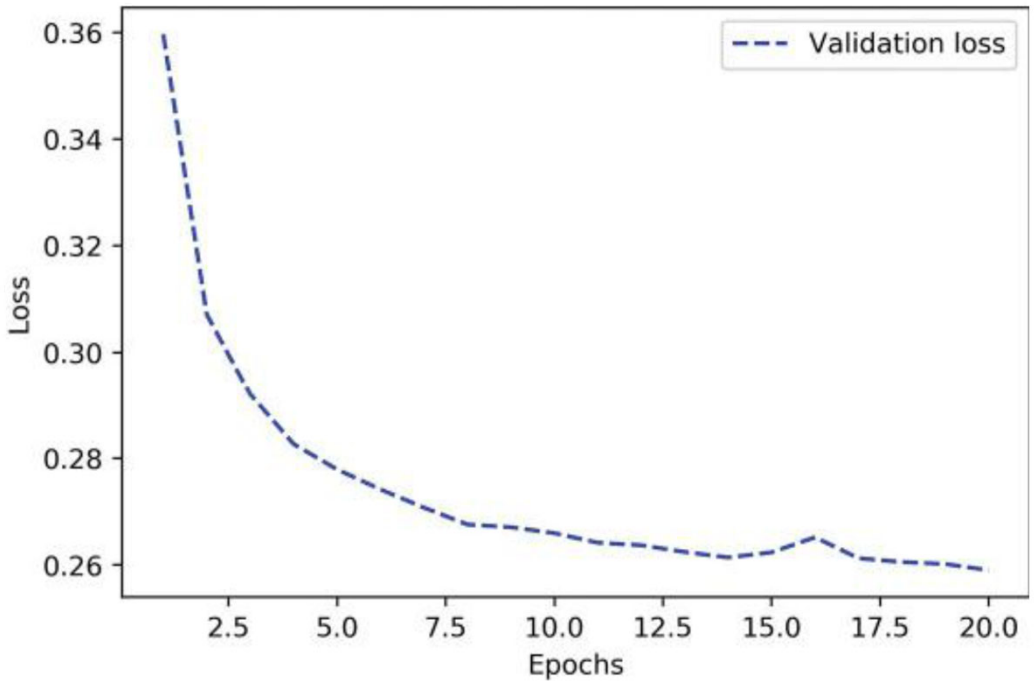


Figure 5.14 Effect of insufficient model capacity on loss curves

Validation metrics seem to stall, or to improve very slowly, instead of peaking and reversing course. The validation loss goes to 0.26 and just stays there. You can fit, but you can't clearly overfit, even after many iterations over the training data. You're likely to encounter similar curves often in your career.

Remember that it should always be possible to overfit. Much like the problem where the training loss doesn't go down, this is an issue that can always be solved. If you can't seem to be able to overfit, it's likely a problem with the *representational power* of your model: you're going to need a bigger model, one with more *capacity*, that is to say, one able to store more information. You can increase representational power by adding more layers, using bigger layers (layers with more parameters), or using kinds of layers that are more appropriate for the problem at hand (better architecture priors).

Let's try training a bigger model, one with two intermediate layers with 96 units each:

```
model = keras.Sequential([
    layers.Dense(96, activation="relu"),
    layers.Dense(96, activation="relu"),
    layers.Dense(10, activation="softmax"),
])
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
history_large_model = model.fit(
    train_images, train_labels,
    epochs=20,
    batch_size=128,
    validation_split=0.2)
```

The validation curve now looks exactly like it should: the model fits fast and starts overfitting after 8 epochs (see figure 5.15).

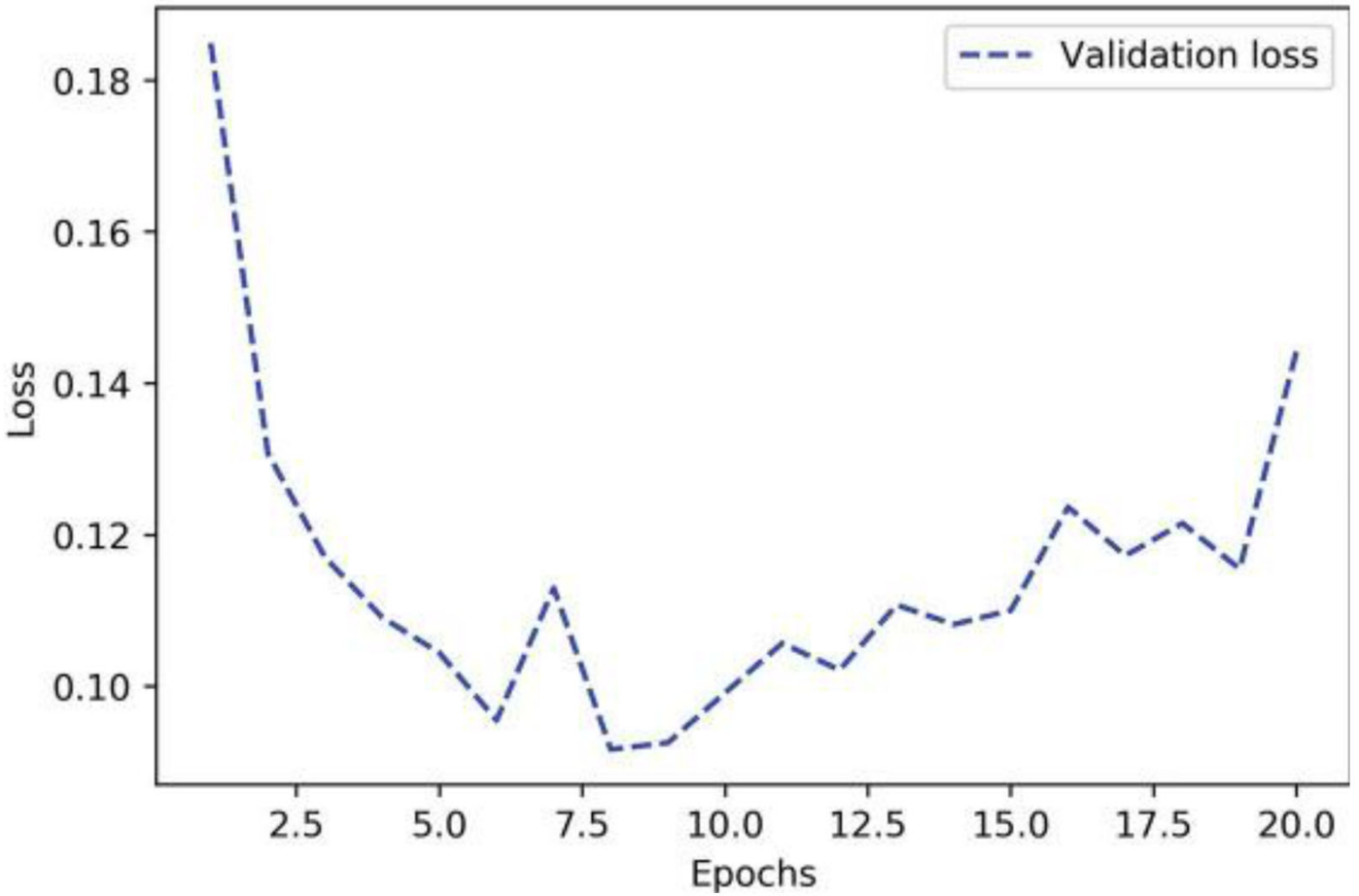


Figure 5.15 Validation loss for a model with appropriate capacity

یادگیری عمیق

مبانی یادگیری ماشینی

۵

پیش‌پردازش داده‌ها، مهندسی ویژگی‌ها، یادگیری ویژگی‌ها

پیش‌پردازش داده‌ها، مهندسی ویژگی‌ها، یادگیری ویژگی‌ها

DATA PREPROCESSING, FEATURE ENGINEERING, AND FEATURE LEARNING

چگونه باید داده‌های ورودی و تارگت‌ها را برای خوراندن به شبکه‌های عصبی آماده کنیم؟

بسیاری از تکنیک‌های **پیش‌پردازش داده‌ها** و **مهندسی ویژگی‌ها**، مختص به یک حوزه‌ی خاص هستند
(مانند: داده‌های متنی یا داده‌های تصویری، ...)

در اینجا، پایه‌های مشترک بین تمامی حوزه‌های داده‌ای بررسی می‌شوند.

پیش‌پردازش داده‌ها برای شبکه‌های عصبی

DATA PREPROCESSING FOR NEURAL NETWORKS

هدف از پیش‌پردازش داده‌ها، مناسب‌تر سازی داده‌های موجود برای شبکه‌های عصبی است.

- **برداری‌سازی**
  Vectorization
- **نرمال‌سازی**
  Normalization
- **برخورد با مقادیر گم‌شده**
  Handling Missing Values
- **استخراج ویژگی**
  Feature Extraction

پیش‌پردازش داده‌ها برای شبکه‌های عصبی

برداری‌سازی

VECTORIZATION

برداری‌سازی
Vectorization

تمام ورودی‌ها و تارگت‌ها در شبکه‌های عصبی باید به
تانسورهایی از داده‌های ممیز شناور (یا در موارد خاص اعداد صحیح)
تبدیل شوند.

مانند: کدگذاری تک–داغ برای کلمات و ...

پیش‌پردازش داده‌ها برای شبکه‌های عصبی

نرمال‌سازی

NORMALIZATION

نرمال‌سازی
Normalization

به‌طور کلی، خوراندن داده‌هایی که مقادیر نسبتاً بزرگی دارند یا داده‌های ناهمگن به یک شبکه‌ی عصبی، کار چندان ایمنی نیست.
(انجام این کار می‌تواند موجب تحریک به‌روزرسانی گرادیانی بزرگی شود که از همگرا شدن شبکه جلوگیری می‌کند.)

ویژگی‌های ضروری داده‌ها برای یادگیری ساده تر توسط شبکه‌های عصبی

- **مقادیر کوچک** (small values):
  اکثر داده‌ها باید در بازه‌ی [0,1] باشند.
- **مقادیر همگن** (homogenous):
  همه‌ی ویژگی‌ها باید مقادیری در بازه‌های تقریباً مشابه داشته باشند.

روش‌های زیر برای نرمال‌سازی قوی‌تر (استانداردسازی) در عمل متداول است، اما همیشه ضروری نیست:

- **نرمال‌سازی هر ویژگی به‌طور مستقل** تا میانگین آن صفر شود.
- **نرمال‌سازی هر ویژگی به‌طور مستقل** تا انحراف استاندارد آن یک شود.

```
x -= x.mean(axis=0)
x /= x.std(axis=0)
```

نرمال‌سازی داده‌ها

تکنیک‌ها

DATA NORMALIZATION: TECHNIQUES

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

بازمقیاس‌دهی (نرمال‌سازی *Min-Max*)
Rescaling (Min-Max Normalization)

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

نرمال‌سازی با میانگین
Mean Normalization

$$x' = \frac{x - \text{average}(x)}{\text{std}(x)}$$

استاندارد سازی
Standardization

پیش‌پردازش داده‌ها برای شبکه‌های عصبی

برخورد با مقادیر گم‌شده

HANDLING MISSING VALUES

برخورد با مقادیر گم‌شده
Handling Missing Values

در برخی موارد ممکن است مقادیر برخی از ویژگی‌ها در برخی از نمونه‌ها موجود نباشد
(مقدار گم‌شده).

در شبکه‌های عصبی، روش ایمن این است که
مقدار داده‌های ورودی ناموجود را برابر با **صفر** قرار دهیم.
به‌شرط آنکه **صفر** یک مقدار معنادار نباشد.

شبکه هنگام مواجهه با داده‌ای که مقدار آن برابر با صفر است،
یاد می‌گیرد که صفر به‌معنی داده‌ی ناموجود است و شروع به نادیده گرفتن آنها می‌کند.

نکته: اگر انتظار می‌رود مقادیر گم‌شده در داده‌های آزمایشی وجود داشته باشد،
اما شبکه با داده‌هایی فاقد هرگونه مقدار گم‌شده آموزش دیده باشد،
آن‌گاه این شبکه روش نادیده گرفتن مقادیر گم‌شده را نخواهد آموخت!
در این موارد باید به‌صورت مصنوعی نمونه‌های آموزشی دارای درایه‌های گم‌شده را تولید کنیم:
چند نمونه‌ی آموزشی را چند بار کپی می‌کنیم و چند ویژگی که در برخی داده‌های آزمایشی وجود ندارد را حذف می‌کنیم.

پیش‌پردازش داده‌ها برای شبکه‌های عصبی

مهندسی ویژگی‌ها

FEATURE ENGINEERING

استخراج ویژگی
Feature Extraction

در مهندسی ویژگی‌ها،
از دانایی موجود در مورد داده‌ها و الگوریتم یادگیری ماشینی مورد نظر (در اینجا شبکه‌های عصبی) استفاده می‌شود و تبدیل‌هایی بر روی داده‌ها پیش از ورود به مدل از طریق کدنویسی اعمال می‌شود تا موجب بهبود عملکرد الگوریتم شود.

در بسیاری از موارد، منطقی نیست که توقع داشته باشیم یک مدل یادگیری ماشینی قادر باشد از **داده‌های کاملاً دلخواه** یاد بگیرد.

داده‌ها باید به‌شیوه‌ای به مدل ارائه شوند که کار مدل را ساده‌تر کنند.

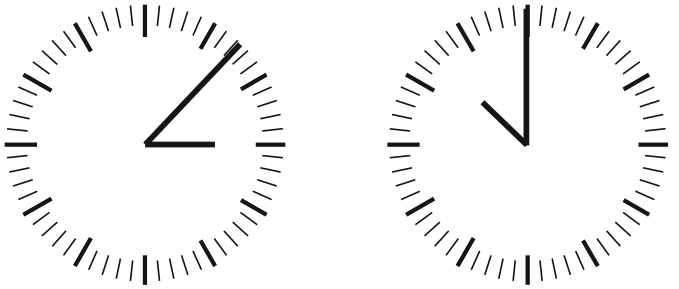
پیش‌پردازش داده‌ها برای شبکه‌های عصبی

مهندسی ویژگی: مثال (خواندن زمان از روی یک ساعت)

فرض می‌کنیم می‌خواهیم مدلی توسعه بدهیم که
تصویر یک ساعت را به‌عنوان **ورودی** دریافت کند و زمان روز را به‌عنوان **خروجی** تحویل بدهد.

اگر پیکسل‌های خام تصویر را به‌عنوان ورودی در نظر بگیریم، یا یک مسئله‌ی دشوار یادگیری ماشینی سروکار خواهیم داشت.
(نیاز به CNN با منابع محاسباتی بالا و داده‌های آموزشی بسیار زیاد)

Raw data: pixel grid



اگر این مسئله را در سطح بالا درک کنیم (روش خواندن زمان از روی ساعت توسط انسان‌ها) می‌توانیم ویژگی‌های ورودی بهتری برای الگوریتم یادگیری بیابیم (مثل: مختصات نوک عقربه‌ها)

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

حتی می‌توانیم از مختصات قطبی استفاده کنیم که در این‌صورت ویژگی‌های لازم، زاویه‌ی هر عقربه است. در اینجا که به‌حدی ساده می‌شود که دیگر حتی نیازی به یادگیری ماشینی نیست!

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

پیش‌پردازش داده‌ها برای شبکه‌های عصبی

مهندسی ویژگی: ضرورت

اساس مهندسی ویژگی‌ها: ساده‌ترسازی یک مسئله از طریق بیان آن به‌صورتی ساده‌تر

پیش از یادگیری عمیق، مهندسی ویژگی‌ها بسیار حیاتی بود،
زیرا مدل‌های کم‌عمق فضاهای فرضیه‌ی به‌اندازه‌ی کافی غنی ندارند تا ویژگی‌های مفید را خودشان یادبگیرند؛
خوشبختانه، یادگیری عمیق مدرن نیاز به مهندسی را تا حد بسیاری حذف کرده است:
زیرا این مدل‌ها قادر به استخراج خودکار ویژگی‌های سودمند از داده‌های خام هستند.

اما این به‌آن معنا نیست که در استفاده از شبکه‌های عصبی عمیق به مهندسی ویژگی‌ها نیازی نداریم.

دلایل نیاز به مهندسی ویژگی‌ها در یادگیری عمیق

- **امکان حل مسائل به‌صورتی بهتر با استفاده از منابع کمتر با ویژگی‌های خوب**
- **امکان حل مسائل با حجم داده‌ی بسیار کمتر**

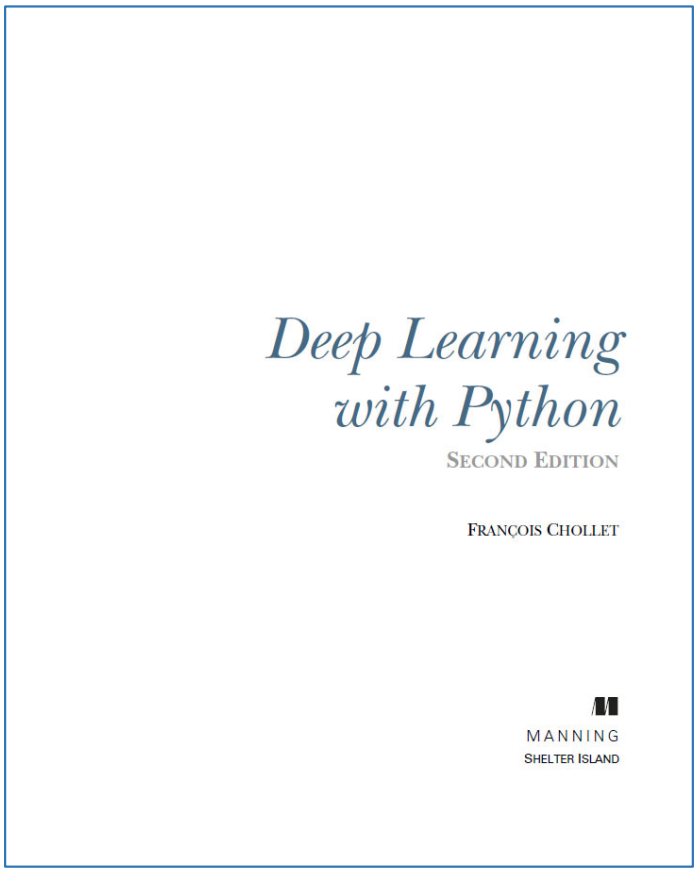
یادگیری عمیق

مبانی یادگیری ماشینی

۶

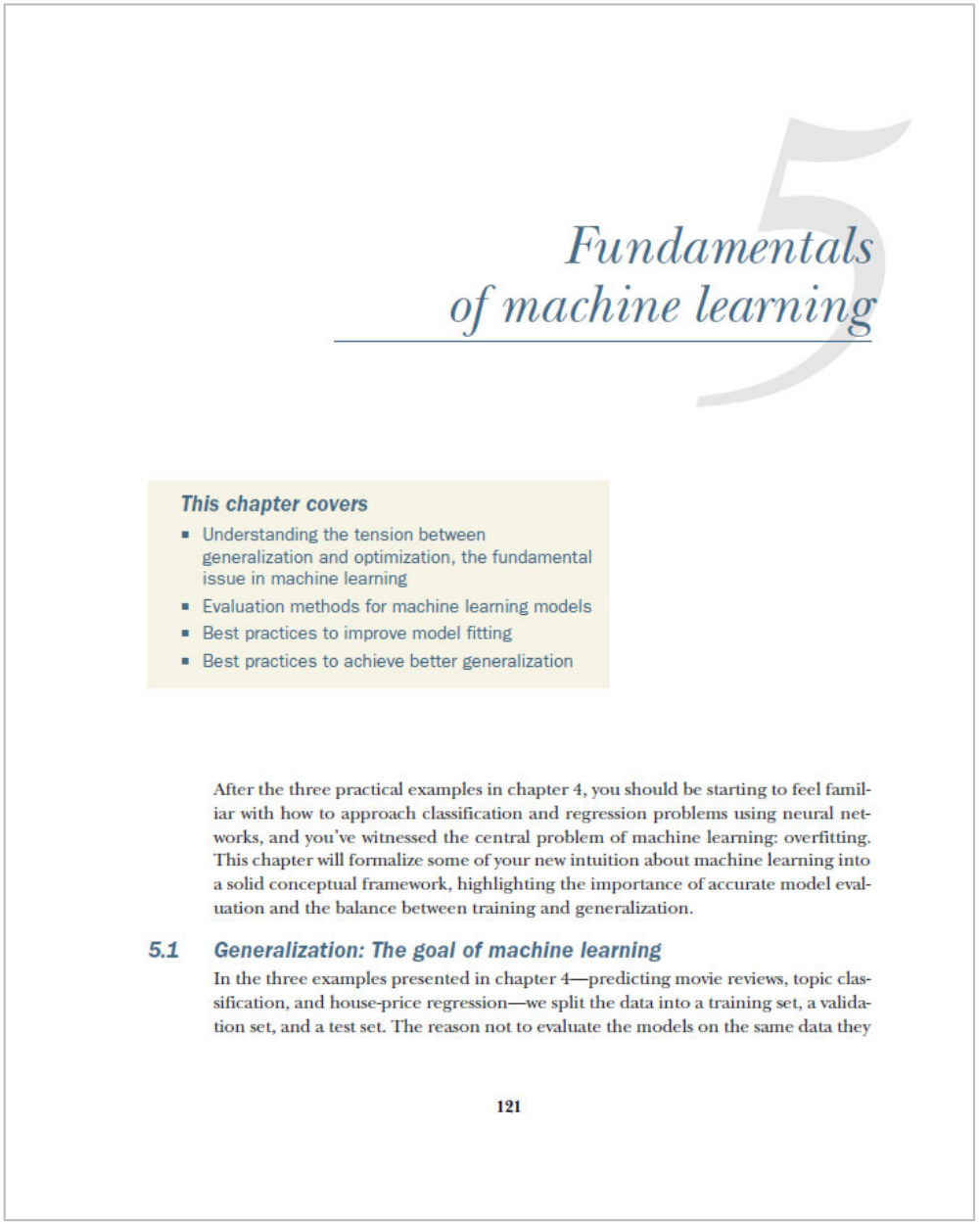
منابع

منبع اصلی



François Chollet,
Deep Learning with Python,
Second Edition, Manning Publications, 2021.

Chapter 5



5

Fundamentals of machine learning

This chapter covers

- Understanding the tension between generalization and optimization, the fundamental issue in machine learning
- Evaluation methods for machine learning models
- Best practices to improve model fitting
- Best practices to achieve better generalization

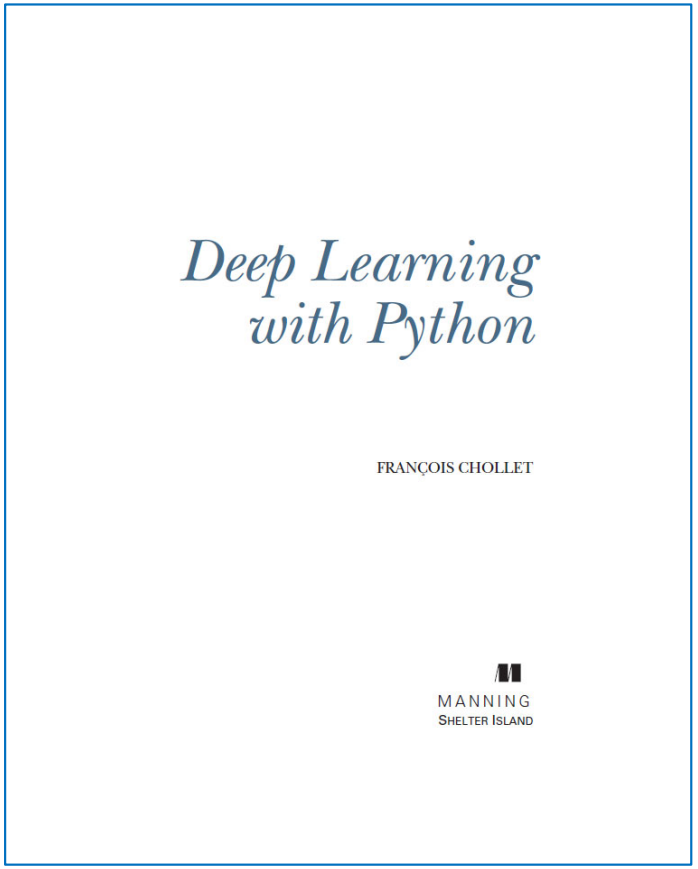
After the three practical examples in chapter 4, you should be starting to feel familiar with how to approach classification and regression problems using neural networks, and you've witnessed the central problem of machine learning: overfitting. This chapter will formalize some of your new intuition about machine learning into a solid conceptual framework, highlighting the importance of accurate model evaluation and the balance between training and generalization.

5.1 Generalization: The goal of machine learning

In the three examples presented in chapter 4—predicting movie reviews, topic classification, and house-price regression—we split the data into a training set, a validation set, and a test set. The reason not to evaluate the models on the same data they

121

منبع اصلی



François Chollet,
Deep Learning with Python,
Manning Publications, 2018.

Chapter 4