یادگیری عمیق

درس ۹

مروری بر شبکه‌های عصبی مصنوعی (۱)

An Overview on Artificial Neural Networks (1)

کاظم فولادی قلعه

دانشکده مهندسی، دانشکدگان فارابی

دانشگاه تهران

http://courses.fouladi.ir/deep

یادگیری عمیق

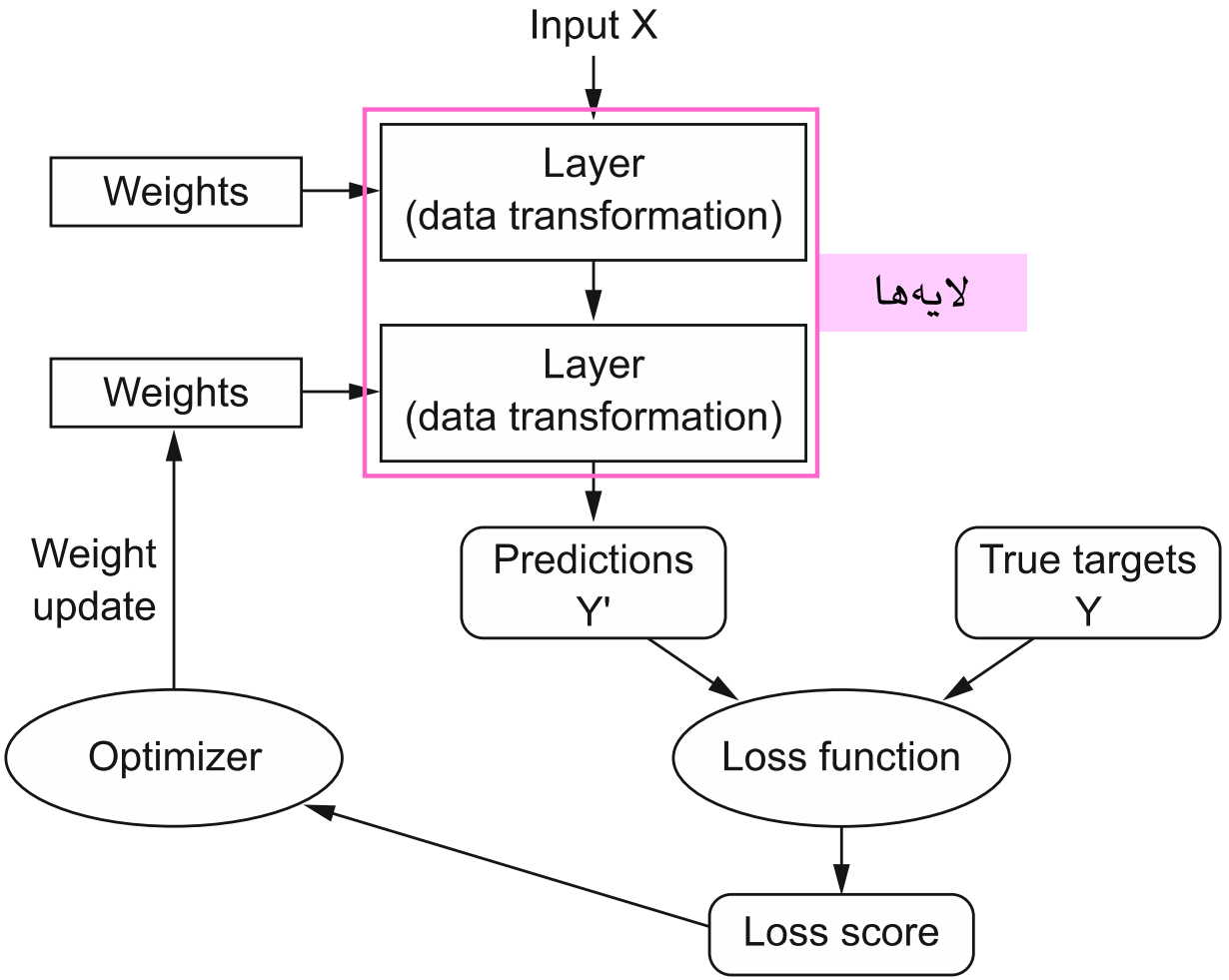
مروری بر شبکه‌های عصبی مصنوعی

۱

آناتومی یک شبکه‌ی عصبی

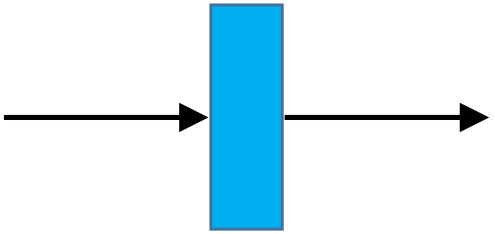
آناتومی یک شبکه‌ی عصبی

ANATOMY OF A NEURAL NETWORK



آناتومی یک شبکه‌ی عصبی

لایه‌ها: بلوک‌های سازنده‌ی یادگیری عمیق



لایه
Layer

یک ماژول پردازش داده که یک یا چند تانسور را به‌عنوان ورودی دریافت می‌کند
و یک یا چند تانسور را به‌عنوان خروجی تحویل می‌دهد.

بدون حالت / بدون حافظه
Stateless / Memory-less

برخی لایه‌ها بدون حالت هستند.

دارای حالت / حافظه‌دار
Stateful / Memory-ful

بیشتر لایه‌ها دارای حالت هستند:
وزن‌های لایه: یک یا چند تانسور
که با الگوریتم SGD یادگرفته شده‌اند.

- لایه‌ی Dense: پردازش داده‌های برداری ساده
- لایه‌ی LSTM/بازگشتی: پردازش داده‌های دنباله‌ای
- لایه‌ی Conv2D: پردازش داده‌های تصویری

آناتومی یک شبکه‌ی عصبی

لایه‌ها: بلوک‌های سازنده‌ی یادگیری عمیق

یک مدل، از کنار هم قرار دادن لایه‌های سازگار برای ایجاد خطوط لوله‌ی تبدیل‌های داده‌ی مفید تشکیل می‌شود.

سازگاری لایه‌ای *Layer Compatibility*	هر لایه فقط تانسورهایی با یک شکل خاص را به‌عنوان ورودی دریافت می‌کند و تانسورهایی با یک شکل خاص را به‌عنوان خروجی تحویل می‌دهد.

مثال:

```
from tensorflow.keras import layers
layer = layers.Dense(32, input_shape=(784,))
```

یک لایه: تانسورهای دو-بعدی (محور صفر: هر عددی، محور یک: ۷۸۴ بعد) را می‌پذیرد.
خروجی این لایه، یک تانسور است که محور یک آن ۳۲ بعد دارد.
⇐ این لایه فقط به لایه‌ای می‌تواند متصل شود که بردارهای ۳۲ بعدی را به‌عنوان ورودی دریافت می‌کند.

```
from tensorflow.keras import models
from tensorflow.keras import layers
model = models.Sequential([
    layers.Dense(32, input_shape=(784,)),
    layers.Dense(32)])
```

وقتی از Keras استفاده می‌کنیم، نگران سازگاری نیستیم.
لایه‌هایی که به مدل اضافه می‌شوند، به‌صورت پویا ساخته می‌شوند تا با شکل لایه‌ی وارد شده به آن مطابقت پیدا کنند.
در مثال فوق، به لایه‌ی دوم آرگومان شکل ورودی داده نشده است،
در عوض، شکل ورودی آن مطابق شکل خروجی لایه‌ی قبل از آن استنتاج می‌شود.

آناتومی یک شبکه‌ی عصبی

مدل‌ها: شبکه‌ای از لایه‌ها

مدل
Model

یک مدل یادگیری عمیق:
یک گراف جهت‌دار بدون دور از لایه‌ها

توپولوژی‌ها

پشته‌ی خطی از لایه‌ها
Linear Stack of Layers

متداول‌ترین نمونه از مدل‌ها:
نگاشت یک ورودی به یک خروجی

شبکه‌های دو–شاخه
Two-Branch Networks

شبکه‌های چند–سر
Multihead Networks

بلوک‌های اینسپشن
Inception Blocks

توپولوژی یک شبکه،
یک **فضای فرضیه** تعریف می‌کند

با انتخاب یک توپولوژی شبکه، فضای امکان‌ها برای نگاشت داده‌های ورودی به خروجی محدود می‌شود.
آنچه برای آن جستجو می‌شود، تانسورهای **وزن** است.

انتخاب یک معماری درست برای شبکه، بیشتر یک هنر است تا یک علم.

آناتومی یک شبکه‌ی عصبی

توابع اتلاف و بهینه‌سازها: کلیدهای پیکربندی فرآیند یادگیری

LOSS FUNCTIONS AND OPTIMIZERS: KEYS TO CONFIGURING THE LEARNING PROCESS

تابع اتلاف
Loss Function

تابع اتلاف (تابع هدف): کمیتی است که در طول آموزش باید می‌نیمم شود؛
تابع اتلاف، معیار موفقیت وظیفه‌ای که در دست است را بازنمایی می‌کند.

انتخاب تابع هدف مناسب برای هر مسئله، فوق‌العاده مهم است.

شبکه از هر مسیر میانبری که بتواند برای می‌نیمم‌سازی اتلاف اقدام می‌کند.
⇐ اگر تابع هدف کاملاً با موفقیت در وظیفه همبستگی نداشته باشد، شبکه در نهایت به کارهای ناخواسته دست می‌زند.

یک شبکه‌ی عصبی که چند خروجی دارد، می‌تواند توابع اتلاف چندگانه داشته باشد (برای هر خروجی یک تابع اتلاف)،
اما فرآیند کاهش گرادیانی باید بر اساس یک مقدار اتلاف اسکالر واحد باشد.
⇐ برای شبکه‌های دارای چند اتلاف، همه‌ی اتلاف‌ها در یک کمیت اسکالر واحد ترکیب می‌شوند (از طریق میانگین‌گیری).

بهینه‌ساز
Optimizer

بهینه‌ساز: تعیین می‌کند که شبکه چگونه باید براساس تابع اتلاف به‌هنگام شود.
بهینه‌ساز، یک گونه از الگوریتم *SGD* را پیاده‌سازی می‌کند.

توابع اتلاف متداول

COMMON LOSS FUNCTIONS

binary crossentropy	**طبقه‌بندی دو–طبقه‌ای** *Two-Class Classification*
categorical crossentropy	**طبقه‌بندی چند–طبقه‌ای** *Many-Class Classification*
mean-squared error	**رگرسیون** *Regression*
connectionist temporal classification (CTC)	**یادگیری دنباله** *Sequence-Learning*

تنها زمانی که بخواهیم بر روی یک مسئله‌ی پژوهشی واقعاً جدید کار کنیم،
باید تابع هدف (تابع اتلاف) اختصاصی طراحی کنیم.

یادگیری عمیق

مروری بر شبکه‌های عصبی مصنوعی

۲

آشنایی با Keras

آشنایی با کراس

INTRODUCTION TO KERAS

Keras is a deep-learning framework for Python
that provides a convenient way to define and train almost any kind of deep-learning model.

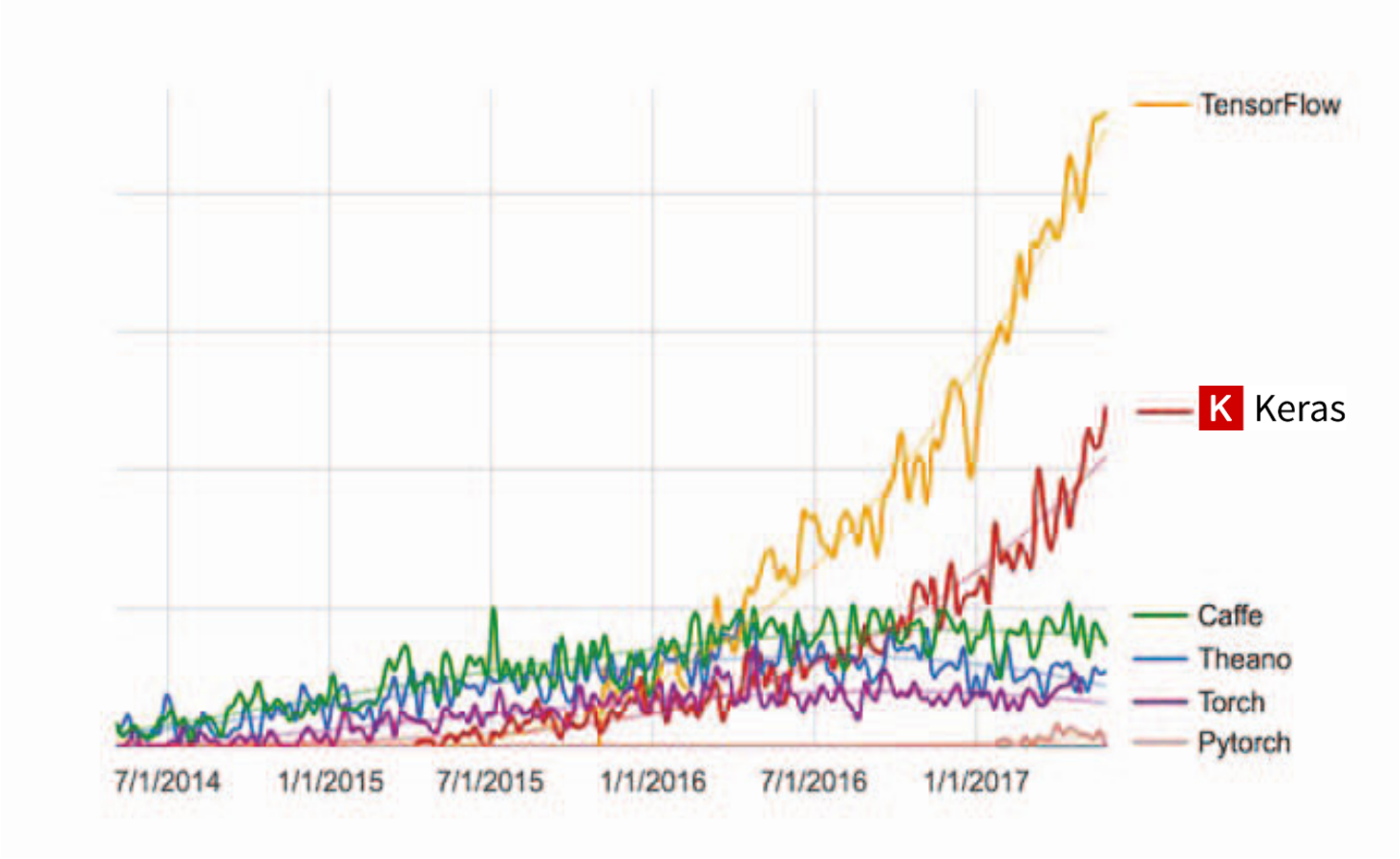
Keras was initially developed for researchers, with the aim of enabling fast experimentation.

Keras has the following key features:

- It allows the same code to run seamlessly on CPU or GPU.
- It has a user-friendly API that makes it easy to quickly prototype deep-learning models.
- It has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, and so on. This means Keras is appropriate for building essentially any deep-learning model, from a generative adversarial network to a neural Turing machine.

آشنایی با کراس

INTRODUCTION TO KERAS



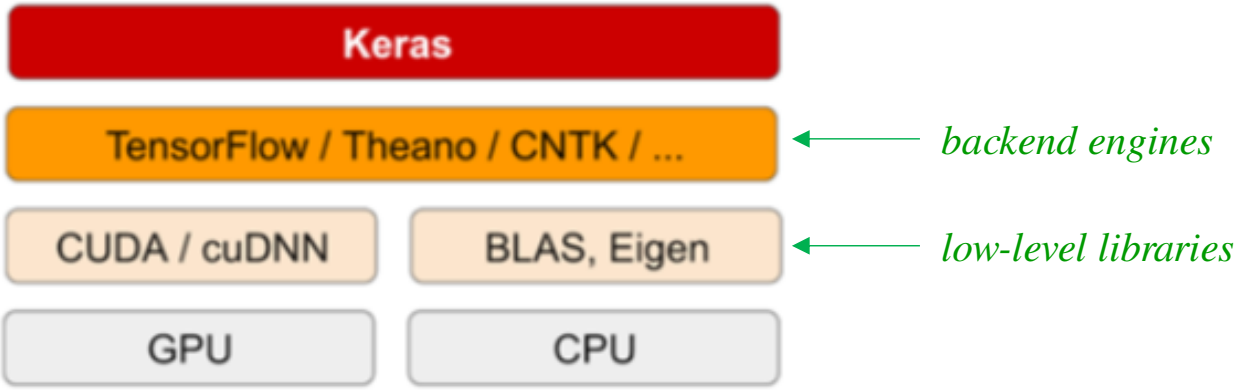
Google web search interest for different deep-learning frameworks over time

آشنایی با کراس

کراس به‌عنوان یک کتابخانه در سطح مدل

INTRODUCTION TO KERAS (-2020)

Keras is a model-level library, providing high-level building blocks for developing deep-learning models.



- **Theano** (`http://deeplearning.net/software/theano`) is developed by the MILA lab at Université de Montréal.
- **TensorFlow** (`http:// www.tensorflow.org`) is developed by Google.
- **CNTK** (`https://github.com/Microsoft/CNTK`) is developed by Microsoft.

Any piece of code that you write with **Keras** can be run with any of these backends without having to change anything in the code.

آشنایی با کراس

کراس به‌عنوان یک کتابخانه در سطح مدل

INTRODUCTION TO KERAS (2021+)

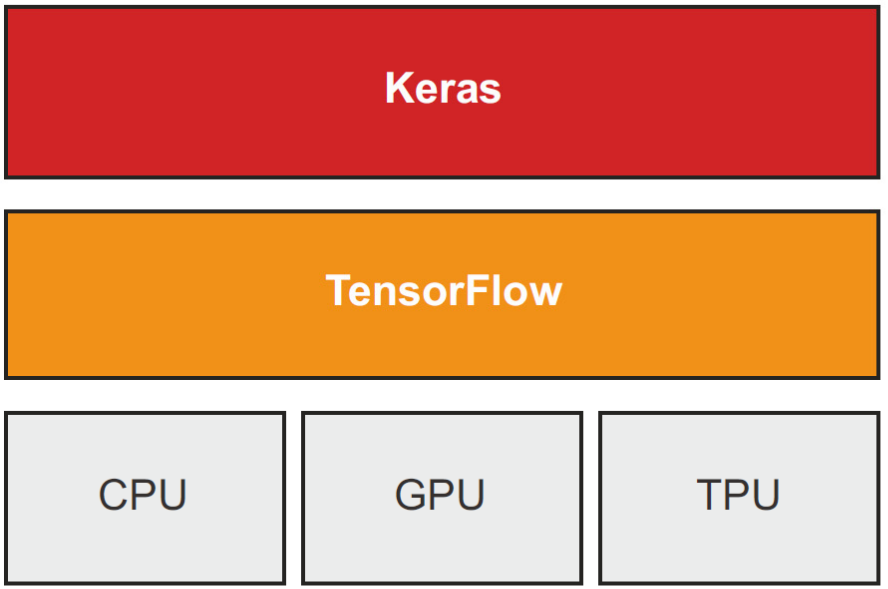


Figure 3.1 Keras and TensorFlow: TensorFlow is a low-level tensor computing platform, and Keras is a high-level deep learning API

آشنایی با کراس

توسعه با کراس

DEVELOPING WITH KERAS

The typical Keras workflow:

1) Define your training data: input tensors and target tensors.
2) Define a network of layers (or model) that maps your inputs to your targets.
3) Configure the learning process by choosing a loss function, an optimizer, and some metrics to monitor.
4) Iterate on your training data by calling the fit() method of your model.

There are two ways to define a model:

A. Using the `Sequential` class
   (only for linear stacks of layers, which is the most common network architecture by far)
B. Using the functional API
   (for directed acyclic graphs of layers, which lets you build completely arbitrary architectures).
C. Model subclassing
   (a low-level option where you write everything yourself from scratch. This is ideal if you want full control over every little thing. However, you won't get access to many built-in Keras features, and you will be more at risk of making mistakes).

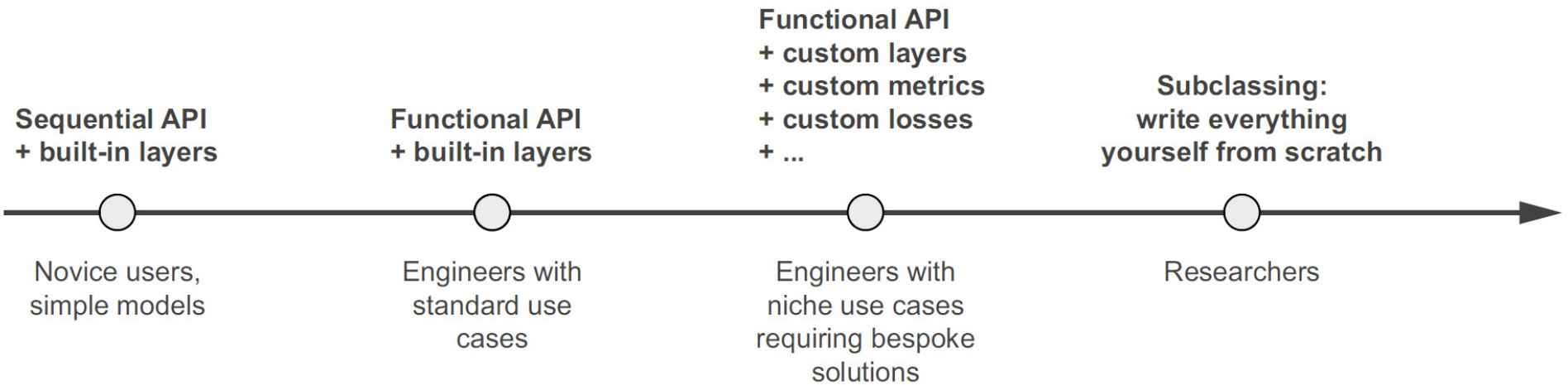


Figure 7.1 Progressive disclosure of complexity for model building

آشنایی با کراس

توسعه با کراس: مثال (تعریف مدل)

DEVELOPING WITH KERAS

Using the `Sequential` class

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
   layers.Dense(64, activation="relu"),
   layers.Dense(10, activation="softmax")
])
```

Using the functional API

```
inputs = keras.Input(shape=(3,), name="my_input")
features = layers.Dense(64, activation="relu")(inputs)
outputs = layers.Dense(10, activation="softmax")(features)
model = keras.Model(inputs=inputs, outputs=outputs)
```

With the functional API, you're manipulating the data tensors that the model processes and applying layers to this tensor as if they were functions.

آشنایی با کراس

توسعه با کراس: مثال (آموزش مدل)

DEVELOPING WITH KERAS

The learning process is configured in the compilation step,
where you specify the optimizer and loss function(s) that the model should use,
as well as the metrics you want to monitor during training:

```
from keras import optimizers
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='mse',
              metrics=['accuracy'])
```

Finally, the learning process consists of passing `Numpy` arrays of input data
(and the corresponding target data) to the model via the fit() method, similar to what you
would do in Scikit-Learn and several other machine-learning libraries:

```
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

یادگیری عمیق

مروری بر شبکه‌های عصبی مصنوعی

۳

راه‌اندازی یک ایستگاه کاری یادگیری عمیق

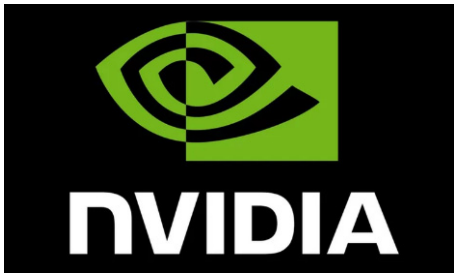
راه‌اندازی یک ایستگاه کاری یادگیری عمیق

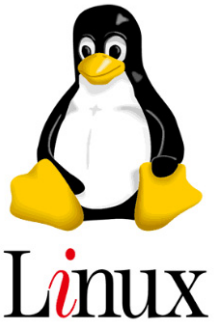
SETTING UP A DEEP-LEARNING WORKSTATION

It's highly recommended, although not strictly necessary,
that you run deep-learning code on a modern NVIDIA GPU.

If you don't want to install a GPU on your machine, you can alternatively consider running your experiments on an AWS EC2 GPU instance or on Google Cloud Platform.

Whether you're running locally or in the cloud, it's better to be using a Linux workstation.





راه‌اندازی یک ایستگاه کاری یادگیری عمیق

دفترچه‌های ژوپیتر: روش مرجح برای اجرای آزمایش‌های یادگیری عمیق

JUPYTER NOTEBOOKS: THE PREFERRED WAY TO RUN DEEP-LEARNING EXPERIMENTS

Jupyter notebooks are a great way to run deep-learning experiments.
A notebook is a file generated by the **Jupyter Notebook app** (`https://jupyter.org`),
which you can edit in your browser.

It mixes the ability to execute Python code with
rich text-editing capabilities for annotating what you're doing.

A notebook also allows you to break up long experiments into smaller pieces
that can be executed independently,
which makes development interactive and means
you don't have to rerun all of your previous code if something goes wrong late in an experiment.

This is a companion notebook for the book Deep Learning with Python, Second Edition. For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

The mathematical building blocks of neural networks

A first look at a neural network

Loading the MNIST dataset in Keras

In [0]:
```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

In [0]:
```
train_images.shape
```

In [0]:
```
len(train_labels)
```

In [0]:
```
train_labels
```

In [0]:
```
test_images.shape
```

In [0]:
```
len(test_labels)
```

In [0]:
```
test_labels
```

The network architecture

In [0]:
```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

https://nbviewer.org/github/fchollet/deep-learning-with-python-notebooks/blob/master/chapter02_mathematical-building-blocks.ipynb

راه‌اندازی یک ایستگاه کاری یادگیری عمیق

اجرای کراس: دو گزینه

GETTING KERAS RUNNING: TWO OPTIONS

To get started in practice, we recommend one of the following two options:

1) Use the official **EC2** Deep Learning AMI (`https://aws.amazon.com/amazonai/amis`), and run Keras experiments as Jupyter notebooks on EC2.
Do this if you don't already have a GPU on your local machine.

2) Install everything from scratch on a **local Linux workstation**.
You can then run either local Jupyter notebooks or a regular Python codebase.
Do this if you already have a high-end NVIDIA GPU.

راه‌اندازی یک ایستگاه کاری یادگیری عمیق

اجرای کارهای یادگیری عمیق بر روی ابر: مزایا و معایب

Running Deep-learning Jobs in The Cloud: Pros and Cons

If you don't already have a GPU that you can use for deep learning
(a recent, high-end NVIDIA GPU),
then running deep-learning experiments in the cloud is a simple, low-cost way
for you to get started without having to buy any additional hardware.

If you're using Jupyter notebooks,
the experience of running in the cloud is no different from running locally.

As of mid-2017,
the cloud offering that makes it easiest to get started with deep learning is definitely **AWS EC2**.
But if you're a heavy user of deep learning, this setup isn't sustainable in the long term
—or even for more than a few weeks. EC2 instances are **expensive**.

راه‌اندازی یک ایستگاه کاری یادگیری عمیق

بهترین GPU برای یادگیری عمیق چیست؟

What is the Best GPU for Deep Learning? (2025)

GPU Type	VRAM Capacity	Memory Bandwidth	Memory Standard	Best For
Entry-level (e.g., RTX 3060, RTX 4060)	8GB – 12GB	~200-300 GB/s	GDDR6	Small models, image classification, hobby projects
Mid-Range (e.g., RTX 3090, RTX 4090)	24GB	~1,000 GB/s	GDDR6X	Large datasets, deep neural networks, transformers
High-end AI GPUs (e.g., Nvidia A100, H100, AMD MI300X)	40GB – 80GB	~1,600+ GB/s	HBM2	Large language models (LLMs), AI research, enterprise-level ML
Super High-end GPUs (e.g., Nvidia H100, AMD Instinct MI300X)	80GB – 256GB	~2,000+ GB/s	HBM3	Large-scale AI training, supercomputing, research on massive datasets

https://cloudzy.com/blog/best-gpu-for-machine-learning/

Best GPUs for Machine Learning in 2025

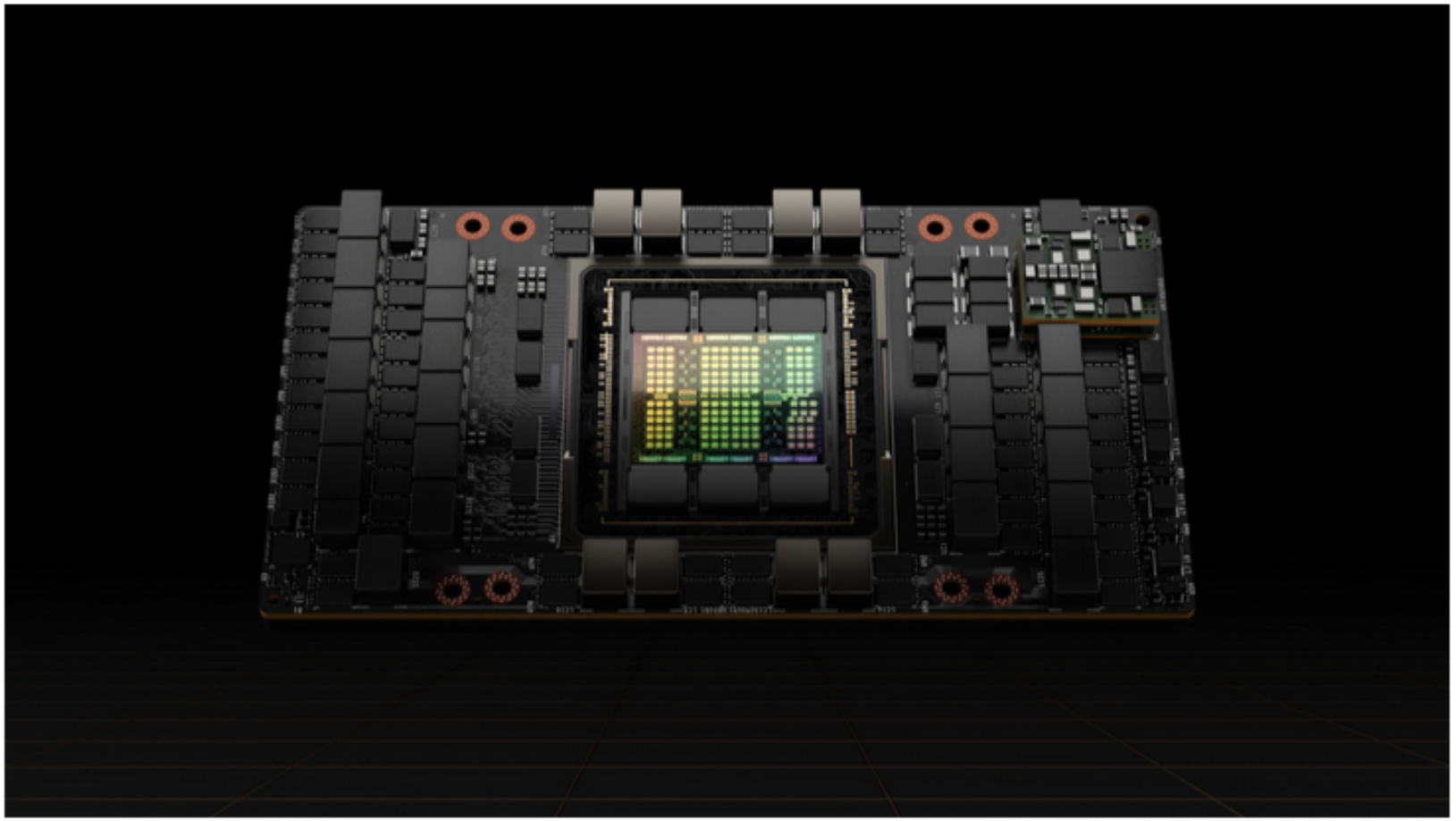
Now that you have a good idea of what the best GPUs for machine learning should have, here's our list of the best GPUs ranked by tops, memory bandwidth, VRAM, etc.

GPU	VRAM	Memory Bandwidth	Memory Standard	TFLOPS	Floating Point Precision	Compatibility
NVIDIA H100 NVL	188 GB	7.8 TB/s	HBM3	3,958	FP64, FP32, FP16	CUDA, TensorFlow
NVIDIA A100 Tensor Core	80 GB	2 TB/s	HBM2	1,979	FP64, FP32, FP16	CUDA, TensorFlow, PyTorch
NVIDIA RTX 4090	24 GB	1.008 TB/s	GDDR6X	82.6	FP32, FP16	CUDA, TensorFlow
NVIDIA RTX A6000 Tensor Core	48 GB	768 GB/s	GDDR6	40	FP64, FP32, FP16	CUDA, TensorFlow, PyTorch
NVIDIA GeForce RTX 4070	12 GB	504 GB/s	GDDR6X	35.6	FP32, FP16	CUDA, TensorFlow
NVIDIA RTX 3090 Ti	24 GB	1.008 TB/s	GDDR6X	40	FP64, FP32, FP16	CUDA, TensorFlow, PyTorch
AMD Radeon Instinct MI300	128 GB	1.6 TB/s	HBM3	60	FP64, FP32, FP16	ROCm, TensorFlow

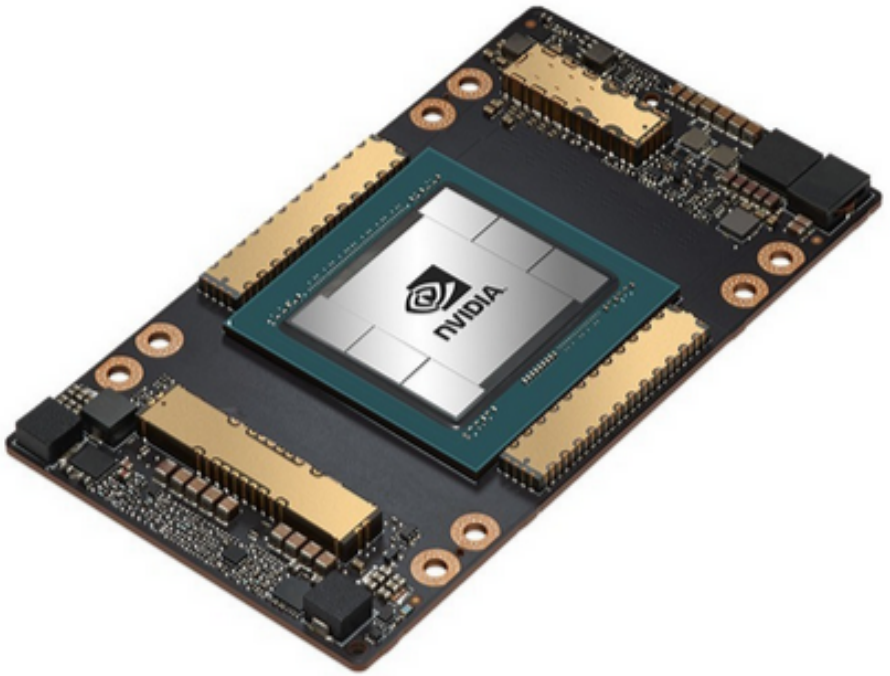
https://cloudzy.com/blog/best-gpu-for-machine-learning/


Titan X
VS
GTX 980
VS
Tesla M40
VS
Tesla K80
TITAN
GEFORCE GTX
GTX 980
NVIDIA
TESLA

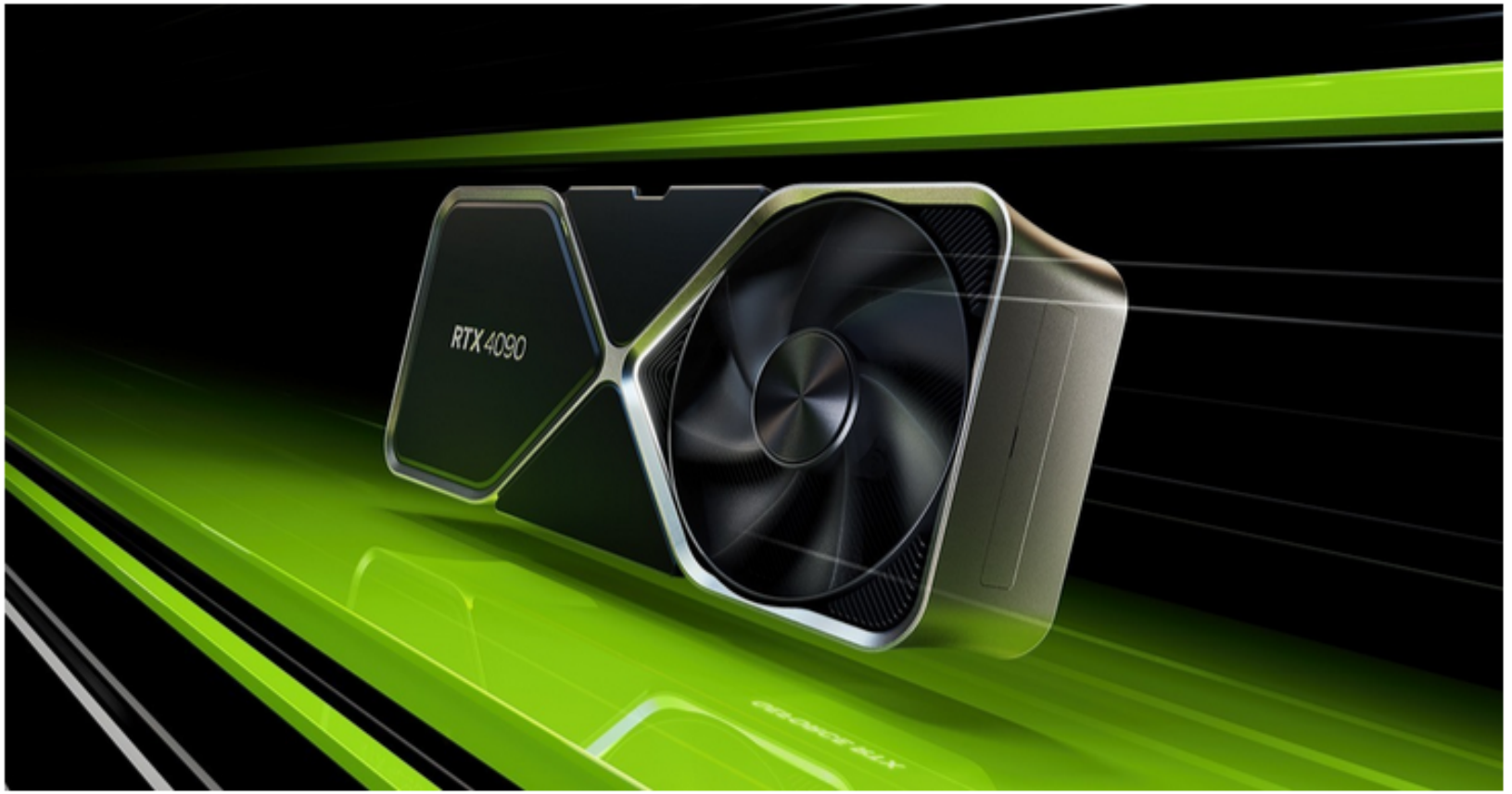
NVIDIA H100 NVL



NVIDIA A100 Tensor Core GPU



NVIDIA RTX 4090



یادگیری عمیق

مروری بر شبکه‌های عصبی مصنوعی

۴

طبقه‌بندی دودویی

مثال: طبقه‌بندی
نقدهای فیلم

طبقه‌بندی نقدهای فیلم

یک مثال برای طبقه‌بندی دودویی

CLASSIFYING MOVIE REVIEWS: A BINARY CLASSIFICATION EXAMPLE

طبقه‌بندی دو–طبقه‌ای یا طبقه‌بندی دودویی،
یکی از گسترده‌ترین انواع مسائل یادگیری ماشینی است.

در این مثال:
هدف، یادگیری طبقه‌بندی نقدهای فیلم در قالب **مثبت** یا **منفی** بر اساس محتوای متنی نقدهاست.

طبقه‌بندی نقدهای فیلم

مجموعه داده‌ی IMDB

THE IMDB DATASET

IMDB Dataset

a set of 50,000 highly polarized reviews from the Internet Movie Database.
They're split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting of 50% negative and 50% positive reviews.

طبقه‌بندی نقدهای فیلم

خواندن مجموعه داده‌ی IMDB

هر **نقد** دنباله‌ای از کلمات است که در این مجموعه (کراس) پیش‌پردازش شده است
و به دنباله‌ای از اعداد صحیح تبدیل شده است.
[هر عدد صحیح به‌جای یک کلمه‌ی مشخص در لغت‌نامه قرار گرفته است.]

Loading the IMDB dataset in Keras

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

آرگومان num_words=10000 به معنی این است که ۱۰۰۰۰ کلمه‌ی فراوان‌تر در داده‌های آموزشی را نگه می‌داریم.
کلمات نادرتر کنار گذاشته می‌شوند ⇐ می‌توانیم با بردارهایی با اندازه‌ی ثابت کار کنیم.

متغیرهای train_data و test_data لیست‌هایی از نقدها هستند: هر نقد لیستی از اندیس کلمات است.
train_labels و test_labels لیست‌هایی از 0ها (برای نقد منفی) و 1ها (برای نقد مثبت) است.

```
>>> train_data[0]
[1, 14, 22, 16, ... 178, 32]
>>> train_labels[0]
1
```

طبقه‌بندی نقدهای فیلم

استخراج متن نقدها

Listing 4.2 Decoding reviews back to text

```
word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

word_index is a dictionary mapping words to an integer index.

Reverses it, mapping integer indices to words

Decodes the review. Note that the indices are offset by 3 because 0, 1, and 2 are reserved indices for "padding," "start of sequence," and "unknown."

طبقه‌بندی نقدهای فیلم

آماده‌سازی داده‌ها

PREPARING THE DATA

لیستی از اعداد صحیح را نمی‌توان وارد یک شبکه‌ی عصبی کرد.
لیست‌ها باید تبدیل به تانسورها شوند.

راه‌های تبدیل لیست به تانسور

پدگذاری
Padding

لیست‌ها را پدگذاری می‌کنیم تا همگی آنها دارای طول یکسان شوند،
سپس آنها را به یک تانسور صحیح با شکل `(samples, word_indices)` تبدیل می‌کنیم.
سپس از لایه‌ای که بتواند روی این تانسورهای صحیح کار کند (مانند لایه‌ی `Embedding`)،
به‌عنوان لایه‌ی اول شبکه استفاده می‌کنیم.

کدگذاری «تک–داغ»
One-Hot Encoding

لیست‌ها را به بردارهایی از 0ها و 1 ها تبدیل می‌کنیم (کدگذاری تک–داغ):
[همه‌جا صفر، غیر از اندیس کلمات حاضر در لیست]
سپس از لایه‌ای که بتواند روی این داده‌های برداری اعشاری کار کند (مانند لایه‌ی `Dense`)،
به‌عنوان لایه‌ی اول شبکه استفاده می‌کنیم.

This would mean, for instance, turning the sequence [3, 5] into a 10,000-dimensional vector that would be all 0s except for indices 3 and 5, which would be 1s.

طبقه‌بندی نقدهای فیلم

آماده‌سازی داده‌ها: کد

PREPARING THE DATA

Listing 4.3 Encoding the integer sequences via multi-hot encoding

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 1.
    return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Creates an all-zero matrix of shape (len(sequences), dimension)

Sets specific indices of results[i] to 1s

Vectorized training data

Vectorized test data

```
>>> x_train[0]
array([ 0., 1., 1., ..., 0., 0., 0.])
```

Vectorizing labels

```
y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")
```

طبقه‌بندی نقدهای فیلم

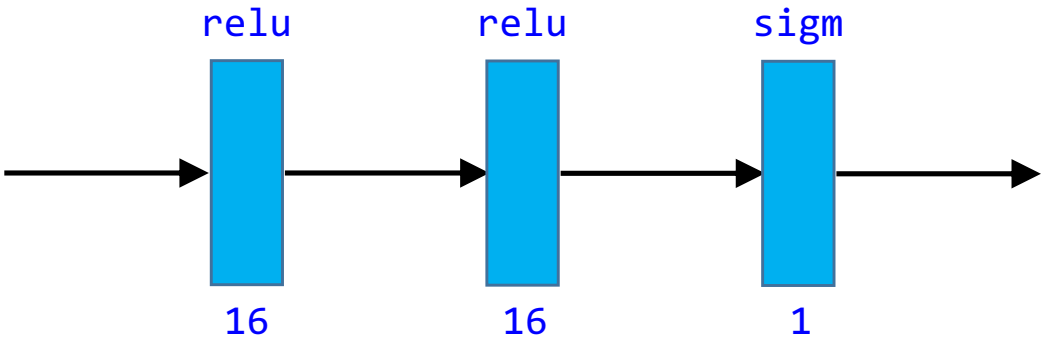
ساخت شبکه

BUILDING THE NETWORK

شبکه را به‌صورت یک پشته‌ی ساده از لایه‌های تماماً متصل (Dense) با توابع فعالیت `relu` می‌سازیم.

دو تصمیم کلیدی برای معماری پشته‌ی لایه‌های *Dense*	۱) تعداد لایه‌های مورد استفاده
	۲) تعداد واحدهای پنهان برای هر لایه

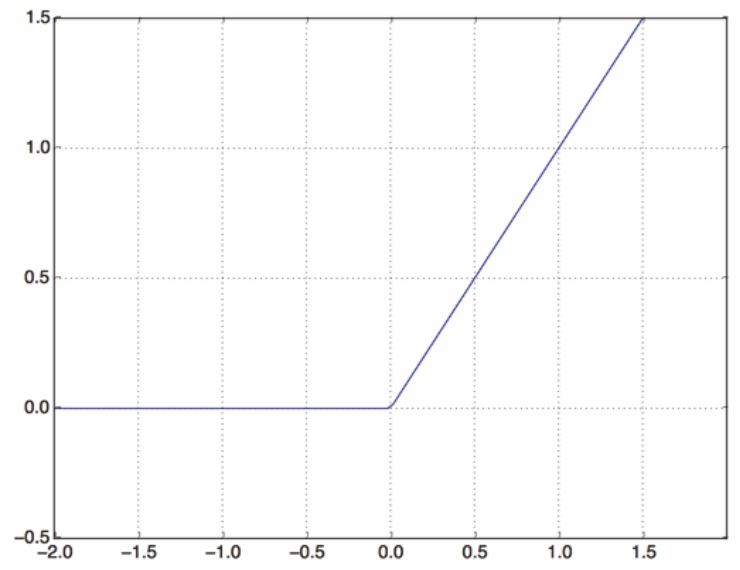
هر چه تعداد واحدهای پنهان بیشتر باشد (فضای بازنمایی با ابعاد بالاتر)،
به شبکه اجازه می‌دهد بازنمایی‌های پیچیده‌تری را یاد بگیرد،
اما شبکه از نظر محاسباتی گران‌تر می‌شود
و ممکن است منجر به یادگیری الگوهای ناخواسته شود.



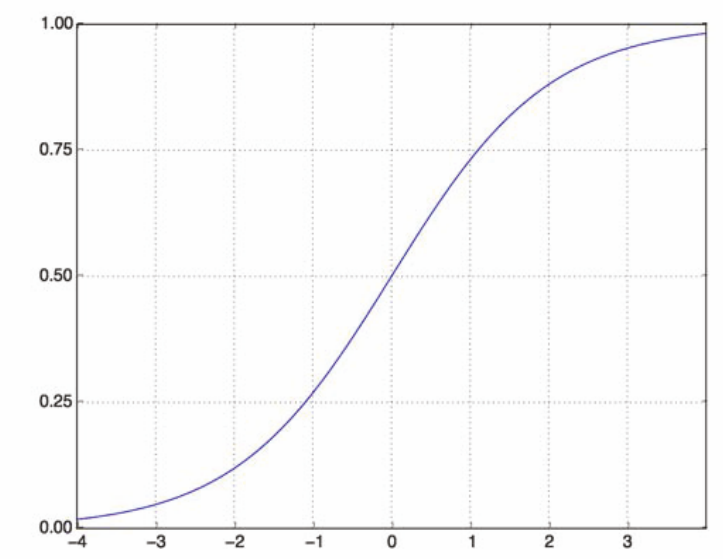
طبقه‌بندی نقدهای فیلم

توابع فعالیت

ACTIVATION FUNCTIONS



ReLU



Sigmoid

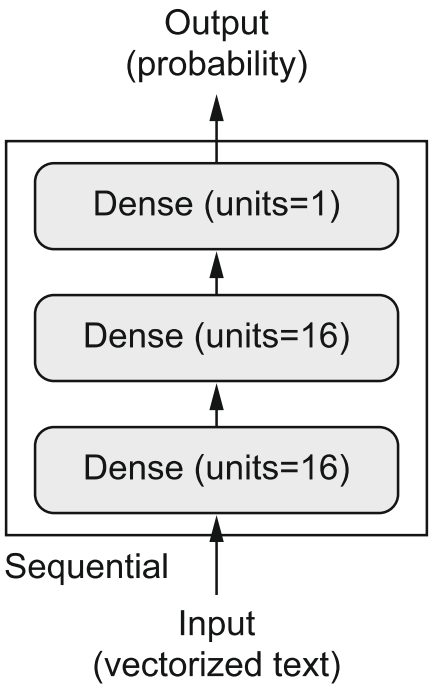
لایه‌ی آخر از تابع فعالیت سیگموئید استفاده می‌کند
تا یک مقدار احتمال را بیان کند.

a score between 0 and 1, indicating how likely the sample is to have the target “1”:
how likely the review is to be positive

طبقه‌بندی نقدهای فیلم

معماری شبکه

THE NETWORK ARCHITECTURE



Listing 4.4 Model definition

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

model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

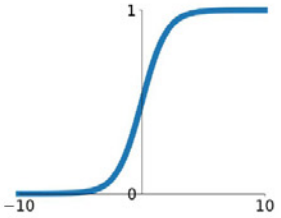
نقش تابع فعالیت غیرخطی

توابع فعالیت غیرخطی، امکان یادگیری تبدیل‌های غیرخطی از یک لایه به لایه‌ی دیگر را فراهم می‌کنند.

اگر توابع فعالیت همه‌ی لایه‌ها خطی باشد،
با افزایش تعداد لایه‌ها فضای فرضیه‌ها تغییری پیدا نمی‌کند
و شبکه هنوز صرفاً قادر به یافتن نگاشت‌های خطی خواهد بود.

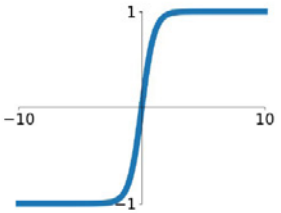
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



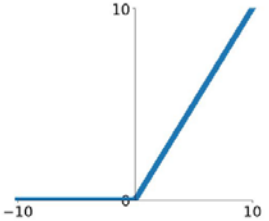
tanh

$$\tanh(x)$$



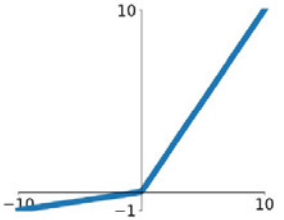
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

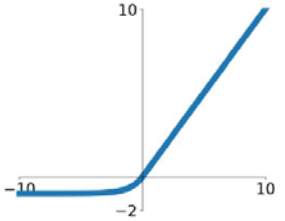


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



طبقه‌بندی نقدهای فیلم

انتخاب تابع اتلاف

CHOOSING LOSS FUNCTION

با یک مسئله‌ی طبقه‌بندی دودویی مواجه هستیم و خروجی شبکه یک احتمال است
⇐ بهترین انتخاب برای تابع اتلاف `binary_crossentropy loss` است.

آنتروپی متقابل
Cross-Entropy

آنتروپی متقابل، کمیتی از حوزه‌ی نظریه‌ی اطلاعات است
که فاصله بین توزیع‌های احتمال (توزیع تارگت‌ها و توزیع پیش‌بینی‌ها) را می‌سنجد.

از دیگر توابع اتلاف مانند `mean_squared_error` نیز می‌توان استفاده کرد،
اما آنتروپی متقابل معمولاً بهترین گزینه برای زمانی است که با مدل‌هایی با خروجی احتمالاتی سر و کار داریم.

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

طبقه‌بندی نقدهای فیلم

کامپایل کردن مدل

COMPILING THE MODEL

Listing 4.5 Compiling the model

```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

طبقه‌بندی نقدهای فیلم

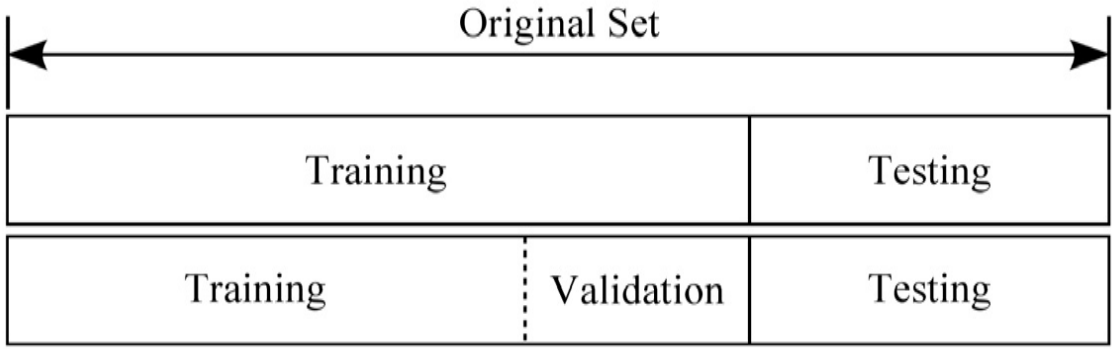
اعتبارسنجی روی‌کرد

VALIDATING THE APPROACH

برای نظارت بر دقت مدل «بر روی داده‌هایی که هرگز تاکنون دیده نشده‌اند» در حین آموزش،
باید یک **مجموعه‌ی اعتبارسنجی** ایجاد کنیم.
برای این منظور، ۱۰,۰۰۰ نمونه از داده‌های آموزشی اصلی جدا می‌کنیم.

Listing 4.6 Setting aside a validation set

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```



طبقه‌بندی نقدهای فیلم

آموزش مدل

TRAINING THE MODEL

حال باید مدل را آموزش بدهیم: ۲۰ اپک (epoch) = ۲۰ تکرار بر روی همه‌ی نمونه‌ها در داده‌های آموزشی
با ریزدسته‌های دارای ۵۱۲ نمونه.
همزمان، بر روی مقادیر **اتلاف** و **دقت** بر روی ۱۰,۰۰۰ داده‌ی اعتبارسنجی جدا شده نظارت می‌کنیم.

Listing 4.7 Training your model

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

در انتهای هر اپک، مکث اندکی وجود دارد: برای محاسبه‌ی اتلاف و دقت بر روی ۱۰,۰۰۰ نمونه‌ی اعتبارسنجی

طبقه‌بندی نقدهای فیلم

تاریخچه‌ی مدل

MODEL HISTORY

The call to `model.fit()` returns a `History` object.
This object has a member `history`,
which is a dictionary containing data about everything that happened during training.

```
>>> history_dict = history.history
>>> history_dict.keys()
[u"accuracy", u"loss", u"val_accuracy", u"val_loss"]
```

The dictionary contains four entries:
one per metric that was being monitored during training and during validation.

طبقه‌بندی نقدهای فیلم

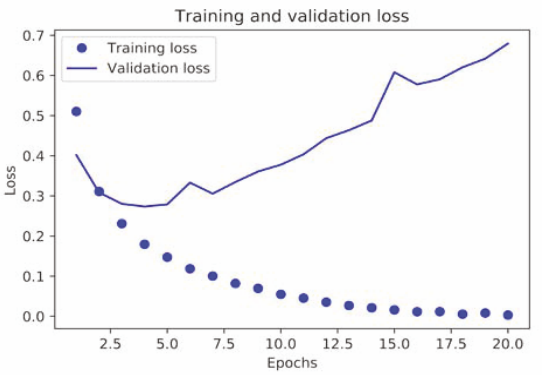
رسم نمودار اتلاف

Listing 4.8 Plotting the training and validation loss

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

"bo" is for "blue dot."

"b" is for "solid blue line."



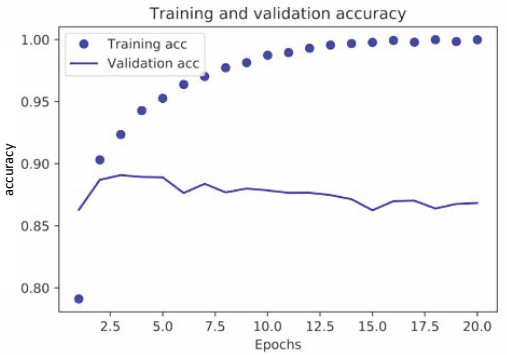
طبقه‌بندی نقدهای فیلم

رسم نمودار دقت

Listing 4.9 Plotting the training and validation accuracy

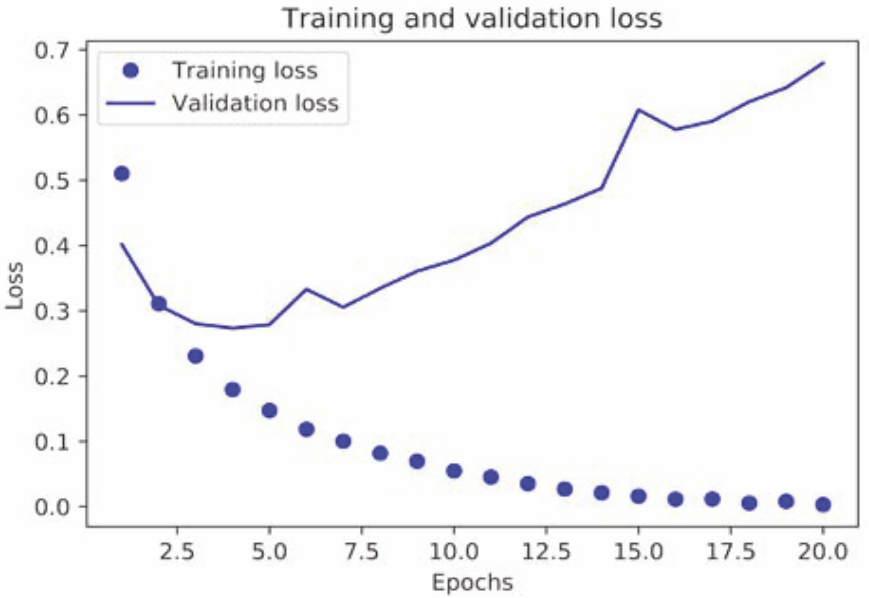
```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

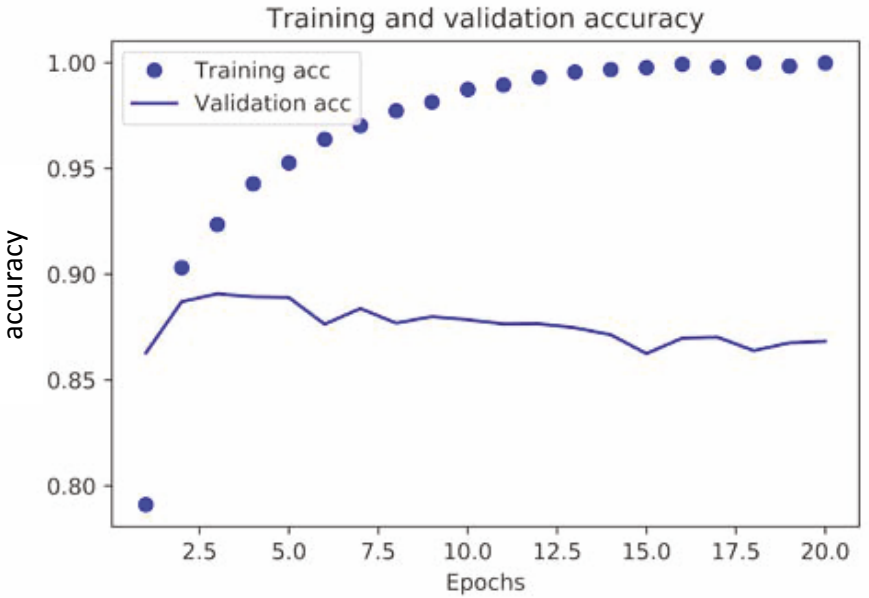
Clears the figure



طبقه‌بندی نقدهای فیلم

تحلیل نمودارهای اتلاف و دقت





اتلاف روی داده‌های آموزشی با هر اپک کاهش می‌یابد.
دقت بر روی داده‌های آموزشی بر روی هر اپک افزایش می‌یابد.

اما برای داده‌های اعتبارسنجی این‌گونه نیست: در اپک چهارم به قله می‌رسد. ⇐ بیش‌برازش

بیش‌برازش *Overfitting*	دقت مدل روی داده‌های آموزشی بیشتر از دقت مدل روی داده‌های اعتبارسنجی شده است.

طبقه‌بندی نقدهای فیلم

جلوگیری از بیش‌برازش

در این مورد، برای جلوگیری از بیش‌برازش، می‌توانیم آموزش را پس از ۴ اپک متوقف کنیم.

Retraining a model from scratch

```
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"])
    model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
>>> results
[0.2929924130630493, 0.8832799999999995]
```

The first number, 0.29, is the test loss, and the second number, 0.88, is the test accuracy.

This fairly naive approach achieves an accuracy of 88%.
With state-of-the-art approaches, you should be able to get close to 95%.

طبقه‌بندی نقدهای فیلم

استفاده از یک شبکه‌ی آموزش دیده برای تولید پیش‌بینی‌ها بر روی داده‌های جدید

USING A TRAINED NETWORK TO GENERATE PREDICTIONS ON NEW DATA

پس از آموزش دیدن شبکه، می‌توانیم از آن در یک موقعیت عملی استفاده کنیم:
می‌توانیم احتمال اینکه یک نقد مثبت یا منفی باشد را محاسبه کنیم:

```
>>> model.predict(x_test)
array([[ 0.98006207]
       [ 0.99758697]
       [ 0.99975556]
       ...,
       [ 0.82167041]
       [ 0.02885115]
       [ 0.65371346]], dtype=float32)
```

As you can see,
the network is confident for some samples (0.99 or more, or 0.01 or less)
but less confident for others (0.6, 0.4).

طبقه‌بندی نقدهای فیلم

آزمایش‌های بیشتر

FURTHER EXPERIMENTS

The following experiments will help convince you that the architecture choices you've made are all fairly reasonable, although there's still room for improvement:

- You used two hidden layers.
  Try using one or three hidden layers,
  and see how doing so affects validation and test accuracy.
- Try using layers with more hidden units or fewer hidden units:
  32 units, 64 units, and so on.
- Try using the **mse** loss function instead of **binary_crossentropy**.
- Try using the **tanh** activation (an activation that was popular in the early days of neural networks) instead of relu.

طبقه‌بندی نقدهای فیلم

جمع‌بندی

WRAPPING UP

Here's what you should take away from this example:

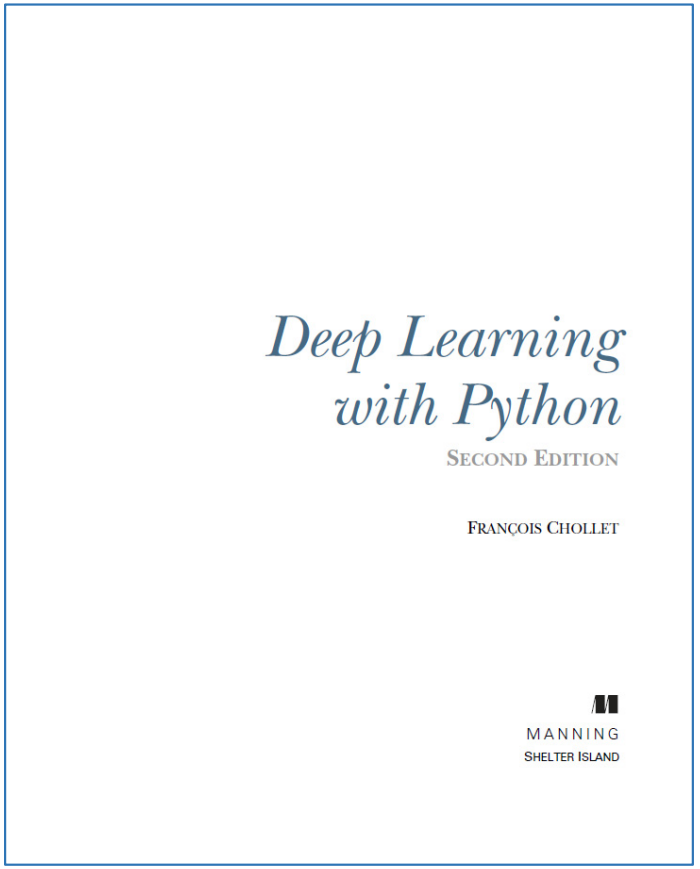
- You usually need to do quite a bit of preprocessing on your raw data in order to be able to feed it—as tensors—into a neural network. Sequences of words can be encoded as binary vectors, but there are other encoding options, too.
- Stacks of `Dense` layers with `relu` activations can solve a wide range of problems (including sentiment classification), and you'll likely use them frequently.
- In a binary classification problem (two output classes), your network should end with a `Dense` layer with one unit and a `sigmoid` activation:
  the output of your network should be a scalar between 0 and 1, encoding a probability.
- With such a scalar sigmoid output on a binary classification problem,
  the loss function you should use is `binary_crossentropy`.
- The `rmsprop` optimizer is generally a good enough choice, whatever your problem.
  That's one less thing for you to worry about.
- As they get better on their training data, neural networks eventually start overfitting and end up obtaining increasingly worse results on data they've never seen before.
  Be sure to always monitor performance on data that is outside of the training set.

یادگیری عمیق

مروری بر شبکه‌های عصبی مصنوعی

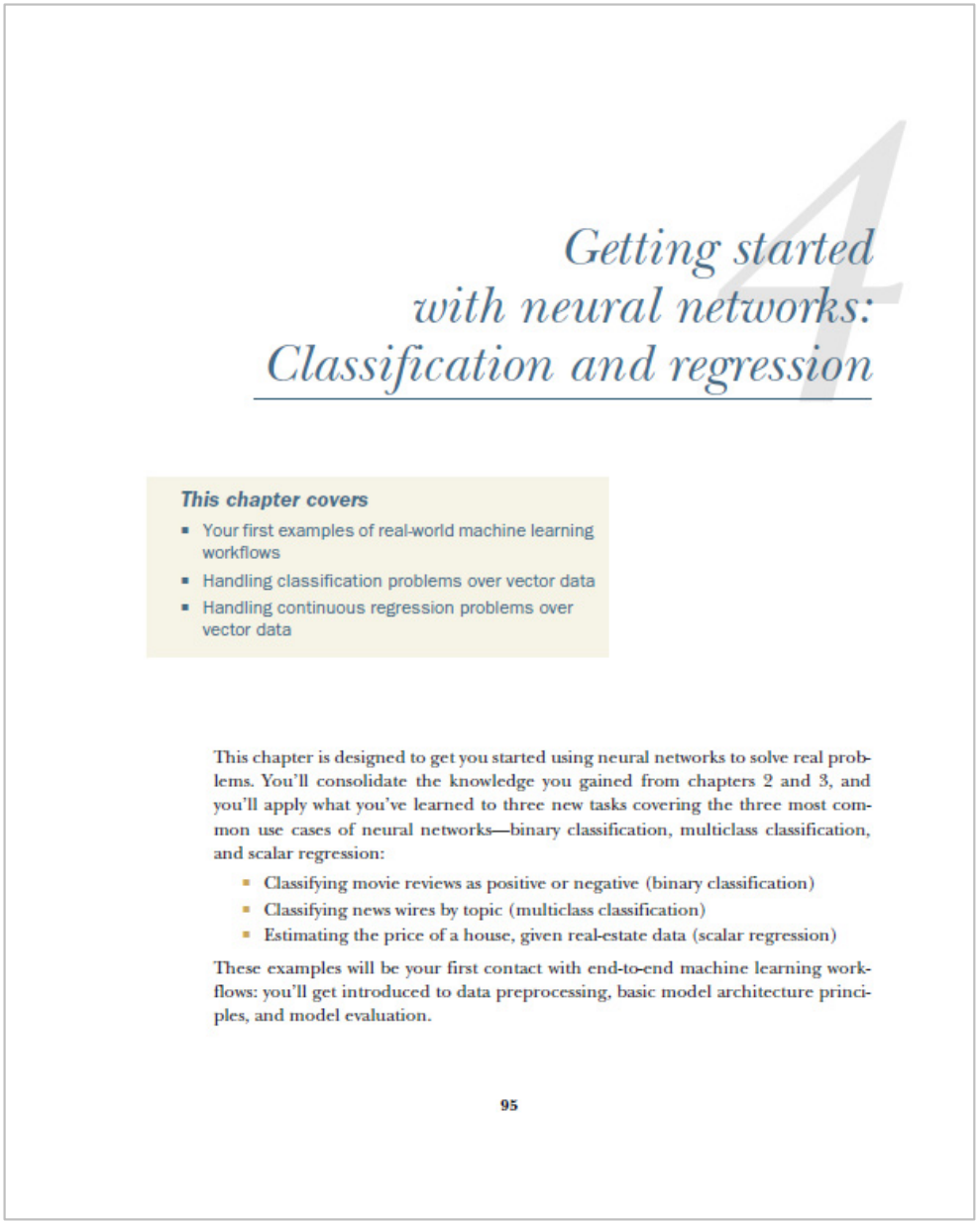
منابع

منبع اصلی



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

Chapter 4



4 Getting started with neural networks: Classification and regression

This chapter covers

- Your first examples of real-world machine learning workflows
- Handling classification problems over vector data
- Handling continuous regression problems over vector data

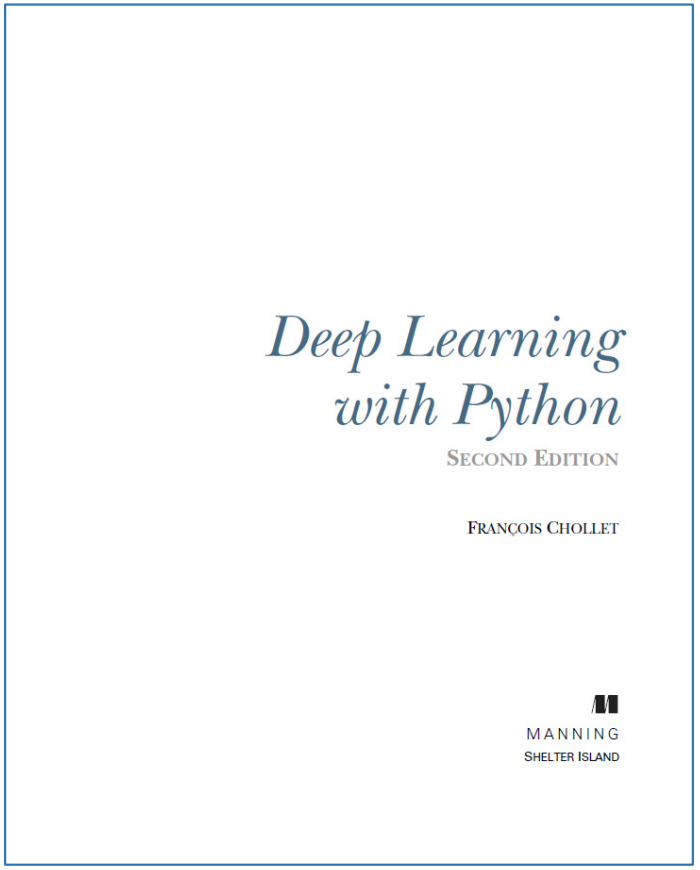
This chapter is designed to get you started using neural networks to solve real problems. You'll consolidate the knowledge you gained from chapters 2 and 3, and you'll apply what you've learned to three new tasks covering the three most common use cases of neural networks—binary classification, multiclass classification, and scalar regression:

- Classifying movie reviews as positive or negative (binary classification)
- Classifying news wires by topic (multiclass classification)
- Estimating the price of a house, given real-estate data (scalar regression)

These examples will be your first contact with end-to-end machine learning workflows: you'll get introduced to data preprocessing, basic model architecture principles, and model evaluation.

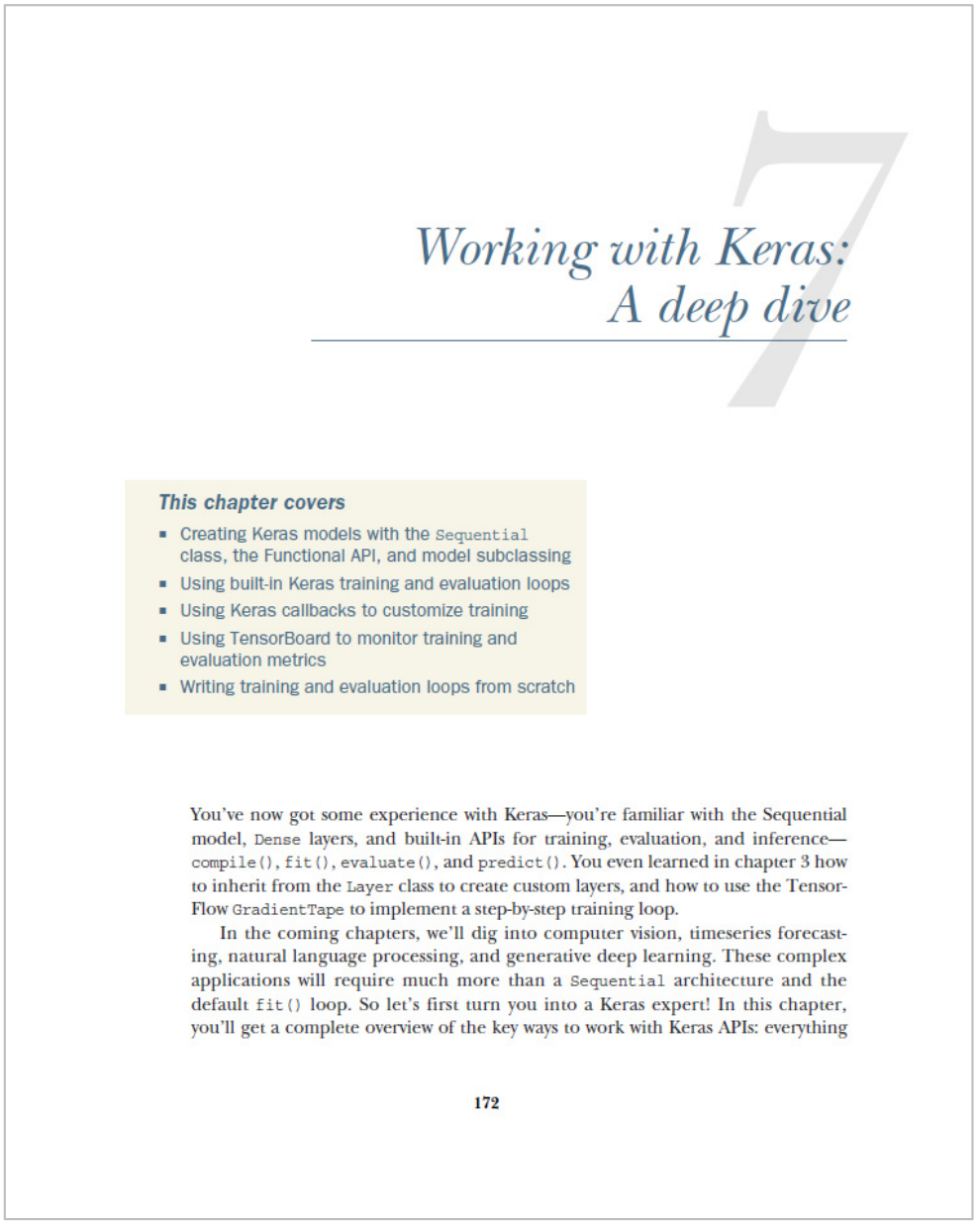
95

منبع اصلی



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

Chapter 7



7

Working with Keras: A deep dive

This chapter covers

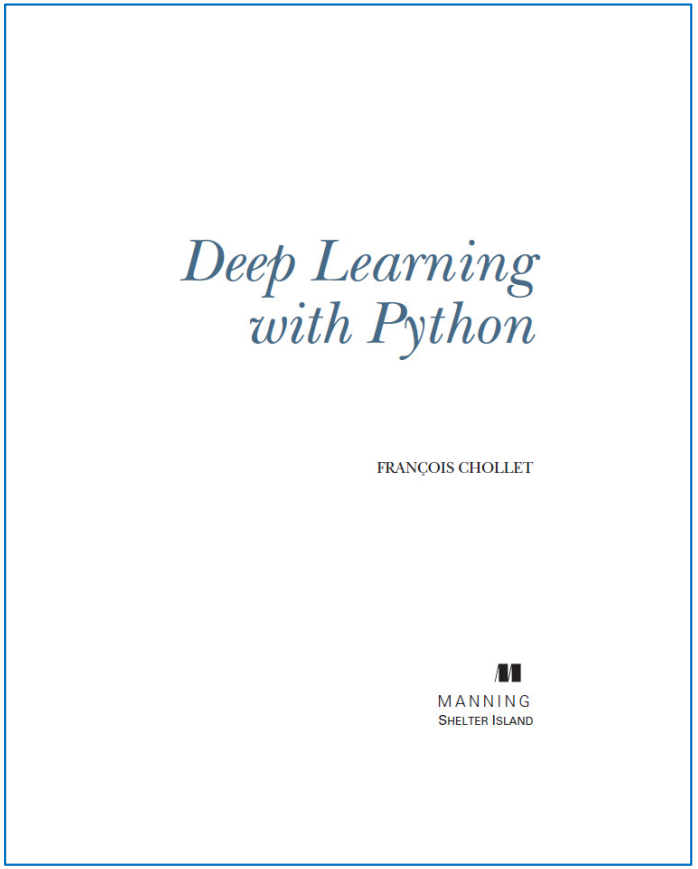
- Creating Keras models with the `Sequential` class, the Functional API, and model subclassing
- Using built-in Keras training and evaluation loops
- Using Keras callbacks to customize training
- Using TensorBoard to monitor training and evaluation metrics
- Writing training and evaluation loops from scratch

You've now got some experience with Keras—you're familiar with the Sequential model, `Dense` layers, and built-in APIs for training, evaluation, and inference—`compile()`, `fit()`, `evaluate()`, and `predict()`. You even learned in chapter 3 how to inherit from the `Layer` class to create custom layers, and how to use the TensorFlow `GradientTape` to implement a step-by-step training loop.

In the coming chapters, we'll dig into computer vision, timeseries forecasting, natural language processing, and generative deep learning. These complex applications will require much more than a `Sequential` architecture and the default `fit()` loop. So let's first turn you into a Keras expert! In this chapter, you'll get a complete overview of the key ways to work with Keras APIs: everything

172

منبع اصلی



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

Chapter 3