یادگیری عمیق

درس ۹

مروری بر شبکه‌های عصبی مصنوعی (۱)

An Overview on Artificial Neural Networks (1)

کاظم فولادی قلعه

دانشکده مهندسی، پردیس فارابی

دانشگاه تهران

http://courses.fouladi.ir/deep

یادگیری عمیق

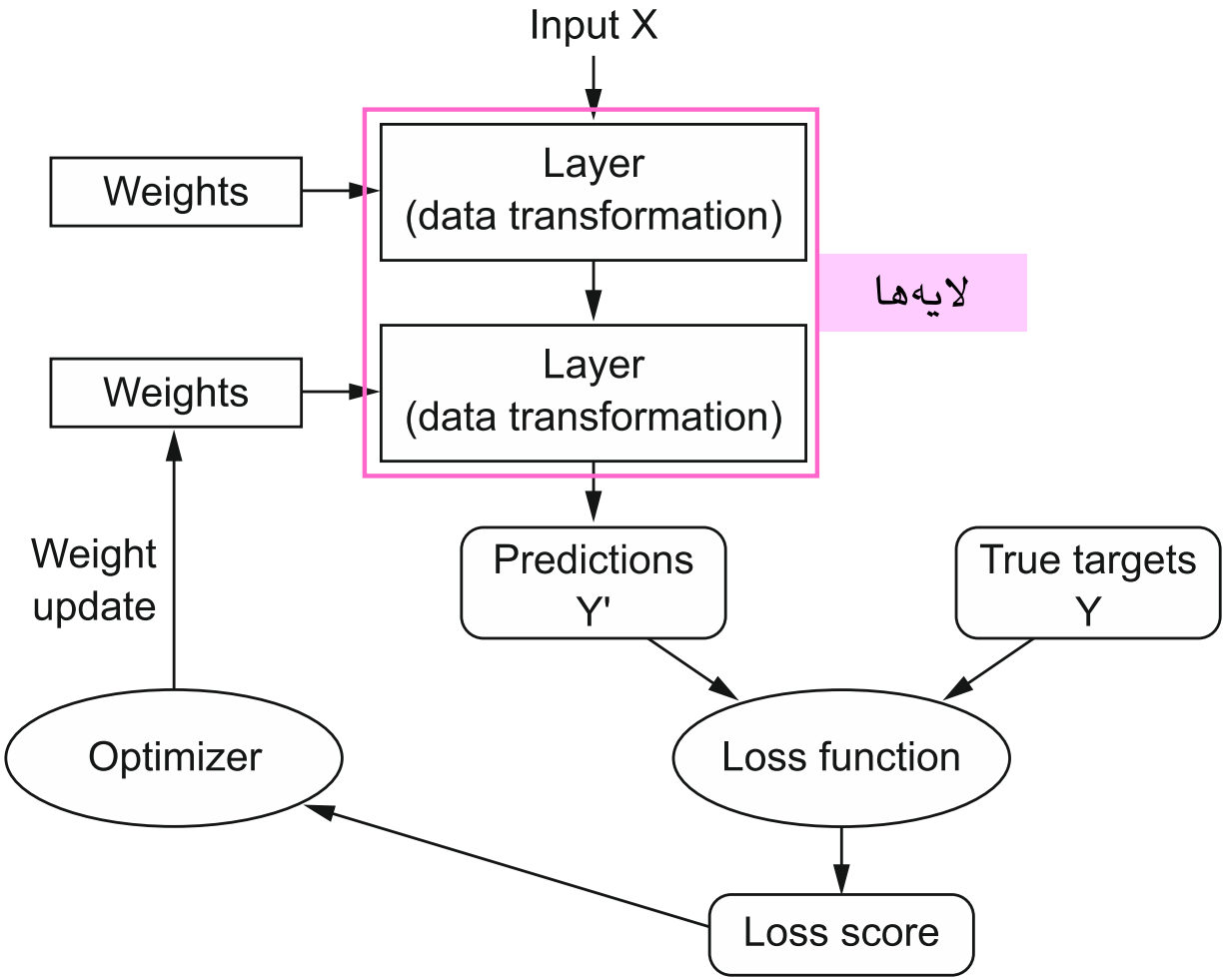
مروری بر شبکه‌های عصبی مصنوعی

۱

آناتومی یک شبکه‌ی عصبی

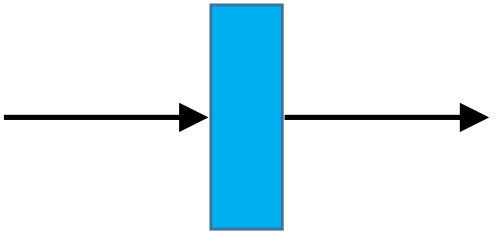
آناتومی یک شبکه‌ی عصبی

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 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 keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(32, input_shape=(784,)))
model.add(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.

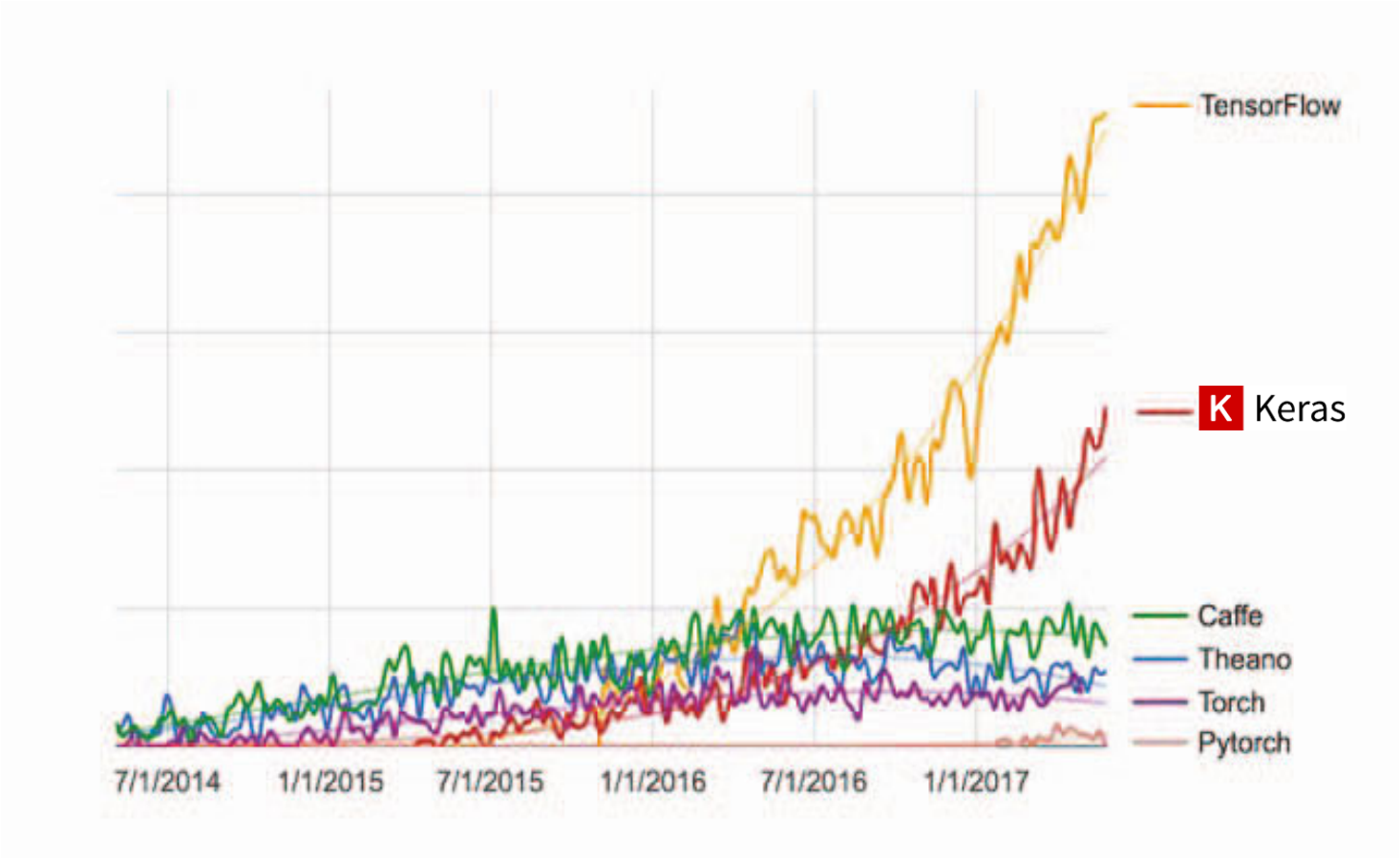
Keras was initially developed for researchers, with the aim of enabling fast experimentation.

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



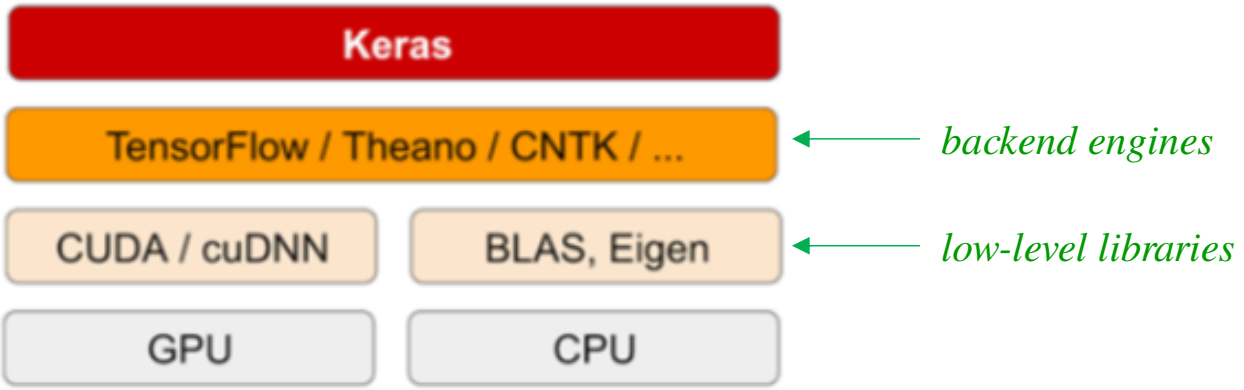
Google web search interest for different deep-learning frameworks over time

آشنایی با کراس

کراس به‌عنوان یک کتابخانه در سطح مدل

INTRODUCTION TO KERAS

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

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).

آشنایی با کراس

توسعه با کراس: مثال (تعریف مدل)

DEVELOPING WITH KERAS

Using the `Sequential` class

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))
```

Using the functional API

```
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation='relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = models.Model(inputs=input_tensor, outputs=output_tensor)
```

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

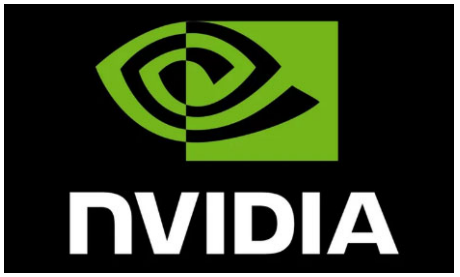
راه‌اندازی یک ایستگاه کاری یادگیری عمیق

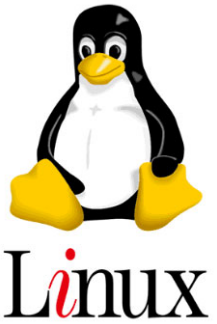
SETTING UP A DEEP-LEARNING WORKSTATION

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.

jupyter nbviewer

JUPYTER FAQ

deep-learning-with-python-notebooks / 2.1-a-first-look-at-a-neural-network.ipynb

In [1]:
```
import keras
keras.__version__
```

```
Using TensorFlow backend.
```

Out[1]: '2.0.8'

A first look at a neural network

This notebook contains the code samples found in Chapter 2, Section 1 of Deep Learning with Python. Note that the original text features far more content, in particular further explanations and figures: in this notebook, you will only find source code and related comments.

We will now take a look at a first concrete example of a neural network, which makes use of the Python library Keras to learn to classify hand-written digits. Unless you already have experience with Keras or similar libraries, you will not understand everything about this first example right away. You probably haven't even installed Keras yet. Don't worry, that is perfectly fine. In the next chapter, we will review each element in our example and explain them in detail. So don't worry if some steps seem arbitrary or look like magic to you! We've got to start somewhere.

The problem we are trying to solve here is to classify grayscale images of handwritten digits (28 pixels by 28 pixels), into their 10 categories (0 to 9). The dataset we will use is the MNIST dataset, a classic dataset in the machine learning community, which has been around for almost as long as the field itself and has been very intensively studied. It's a set of 60,000 training images, plus 10,000 test images, assembled by the National Institute of Standards and Technology (the NIST in MNIST) in the 1980s. You can think of "solving" MNIST as the "Hello World" of deep learning -- it's what you do to verify that your algorithms are working as expected. As you become a machine learning practitioner, you will see MNIST come up over and over again, in scientific papers, blog posts, and so on.

The MNIST dataset comes pre-loaded in Keras, in the form of a set of four Numpy arrays:

In [2]:
```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

`train_images` and `train_labels` form the "training set", the data that the model will learn from. The model will then be tested on the "test set", `test_images`

https://nbviewer.jupyter.org/github/fchollet/deep-learning-with-python-notebooks/blob/master/2.1-a-first-look-at-a-neural-network.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?

As of mid-2017,
the NVIDIA TITAN Xp had recommend as the best card on the market for deep learning.
For lower budgets, you may want to consider the GTX 1060.


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

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 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

```
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

Encoding the integer sequences into a binary matrix

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    # Creates an all-zero matrix of shape (len(sequences), dimension)
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    # Sets specific indices of results[i] to 1s
    return results
x_train = vectorize_sequences(train_data) # Vectorized training data
x_test = vectorize_sequences(test_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

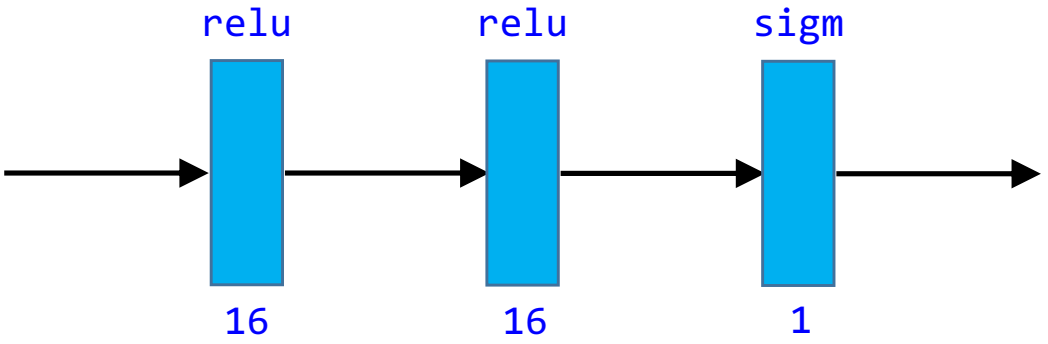
ساخت شبکه

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

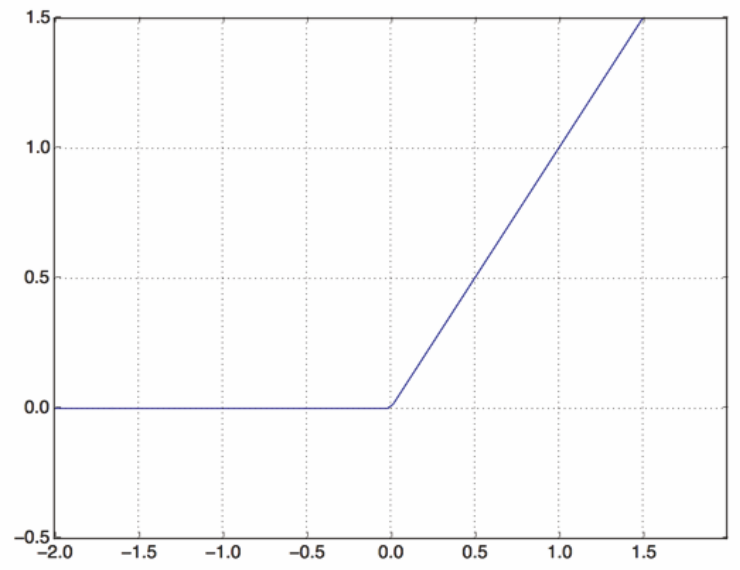
هر چه تعداد واحدهای پنهان بیشتر باشد (فضای بازنمایی با ابعاد بالاتر)،
به شبکه اجازه می‌دهد بازنمایی‌های پیچیده‌تری را یاد بگیرد،
اما شبکه از نظر محاسباتی گران‌تر می‌شود
و ممکن است منجر به یادگیری الگوهای ناخواسته شود.



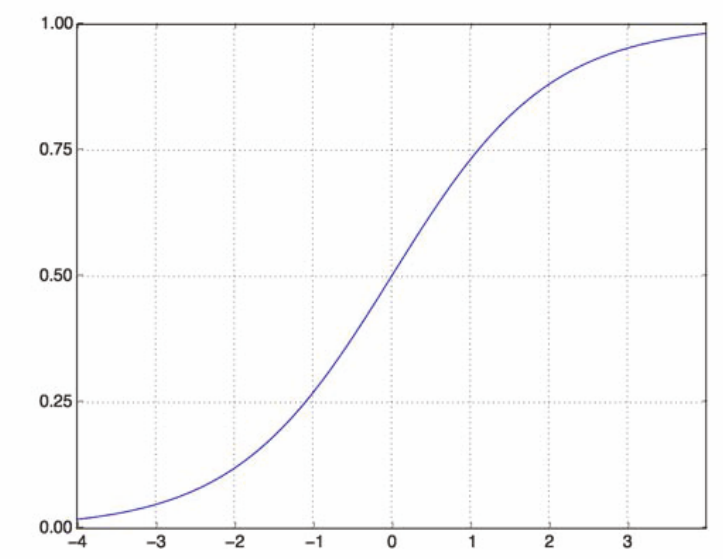
طبقه‌بندی نقدهای فیلم

توابع فعالیت

ACTIVATION FUNCTIONS



ReLU



Sigmoid

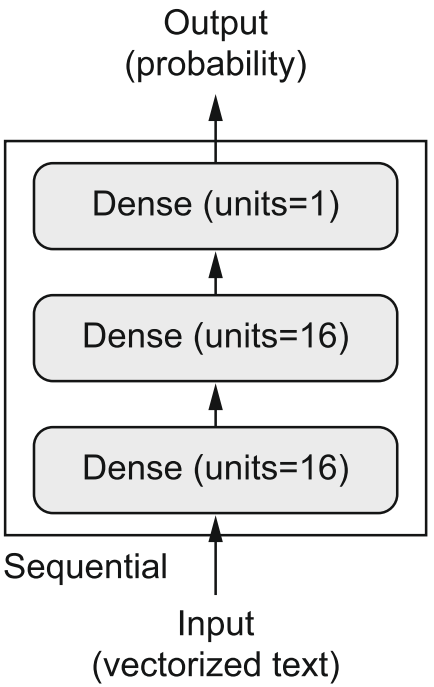
لایه‌ی آخر از تابع فعالیت سیگموئید استفاده می‌کند
تا یک مقدار احتمال را بیان کند.

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



The model definition

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

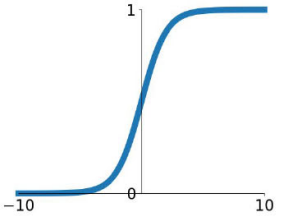
نقش تابع فعالیت غیرخطی

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.

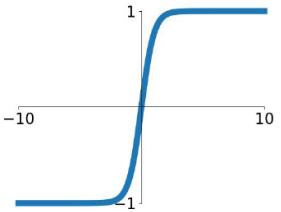
Sigmoid

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



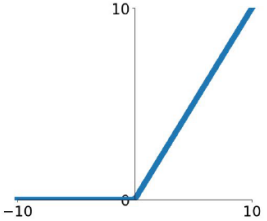
tanh

$$\tanh(x)$$



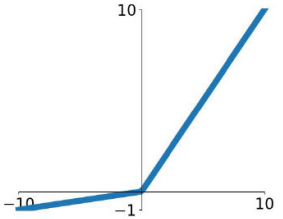
ReLU

$$\max(0, x)$$



Leaky ReLU

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

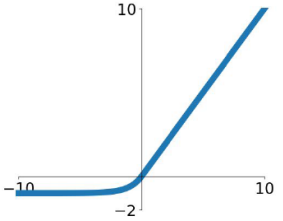


Maxout

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

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

Compiling the model

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

Configuring the optimizer

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

Using custom losses and metrics

```
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

طبقه‌بندی نقدهای فیلم

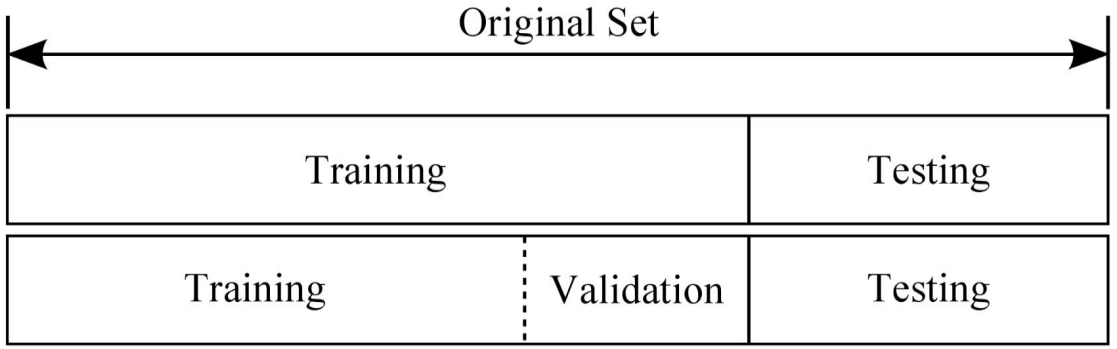
اعتبارسنجی روی‌کرد

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.

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.

Training the Model

```
model.compile(optimizer='rmsprop’,
              loss='binary_crossentropy’,
              metrics=['acc'])
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'acc', u'loss', u'val_acc', u'val_loss']
```

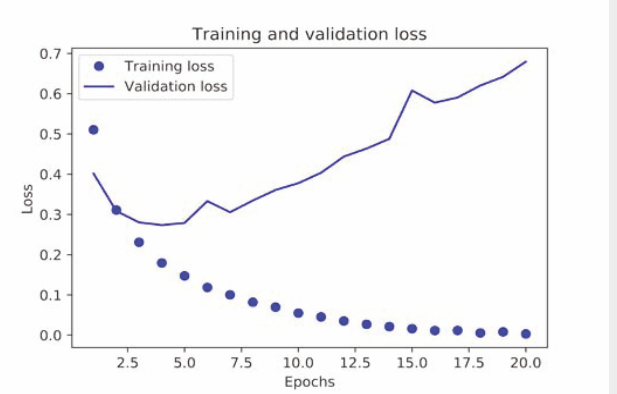
The dictionary contains four entries:
one per metric that was being monitored during training and during validation.

طبقه‌بندی نقدهای فیلم

رسم نمودار اتلاف

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(acc) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss’)
                              # 'bo' is for "blue dot".
plt.plot(epochs, val_loss_values, 'b', label='Validation loss’)
                              # 'b' is for "solid blue line".
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

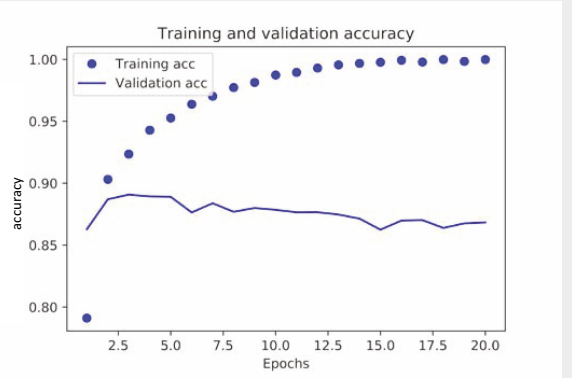


طبقه‌بندی نقدهای فیلم

رسم نمودار دقت

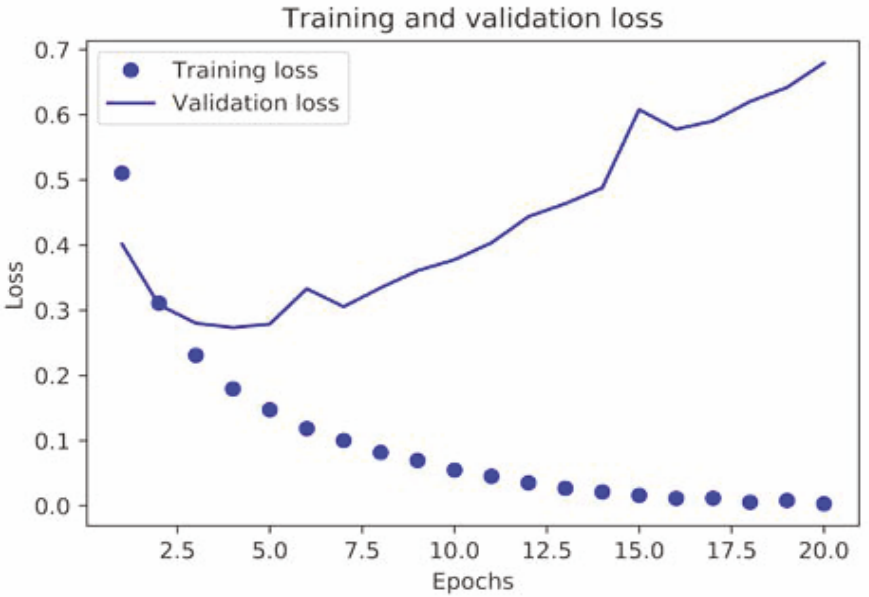
Plotting the training and validation accuracy

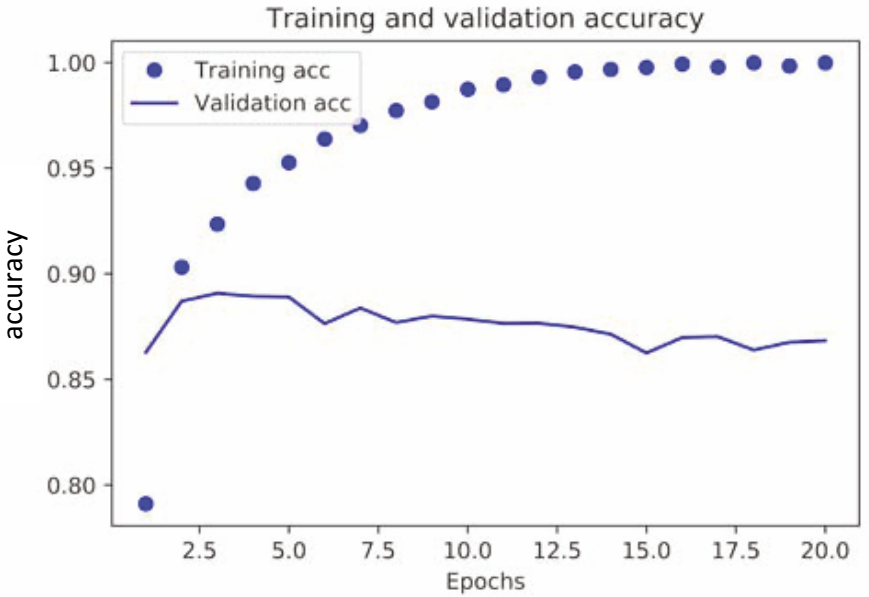
```
plt.clf()    # clear the figure


acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc’]


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('Loss')
plt.legend()
plt.show()
```



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 ۳ اپک متوقف کنیم.

Retraining a model from scratch

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(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]
```

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:

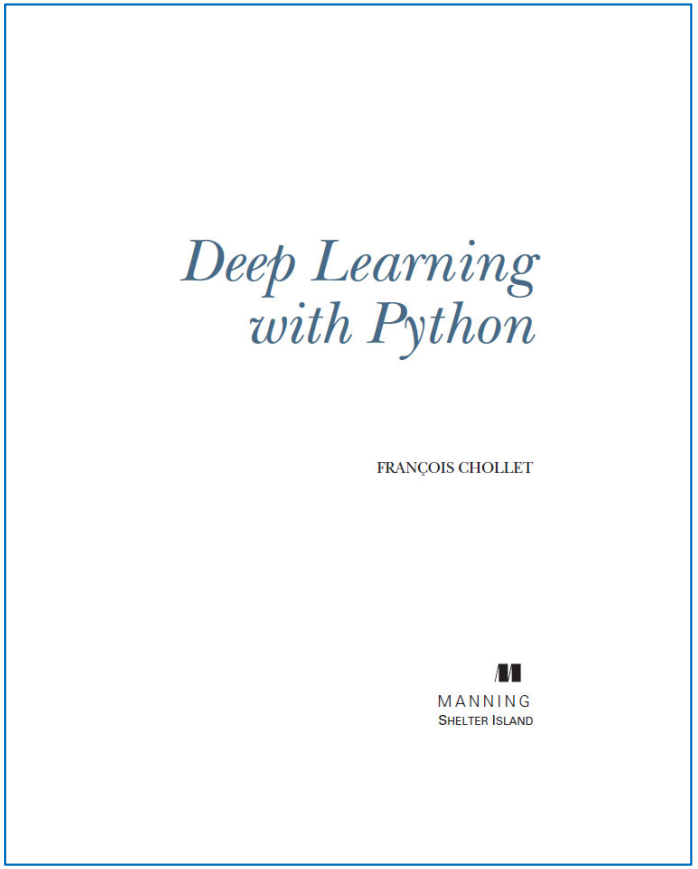
- You usually need to do quite a bit of preprocessing on your raw data in order to be able to feed it—as tensors—into a neural network. Sequences of words can be encoded as binary vectors, but there are other encoding options, too.
- Stacks of `Dense` layers with `relu` activations can solve a wide range 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

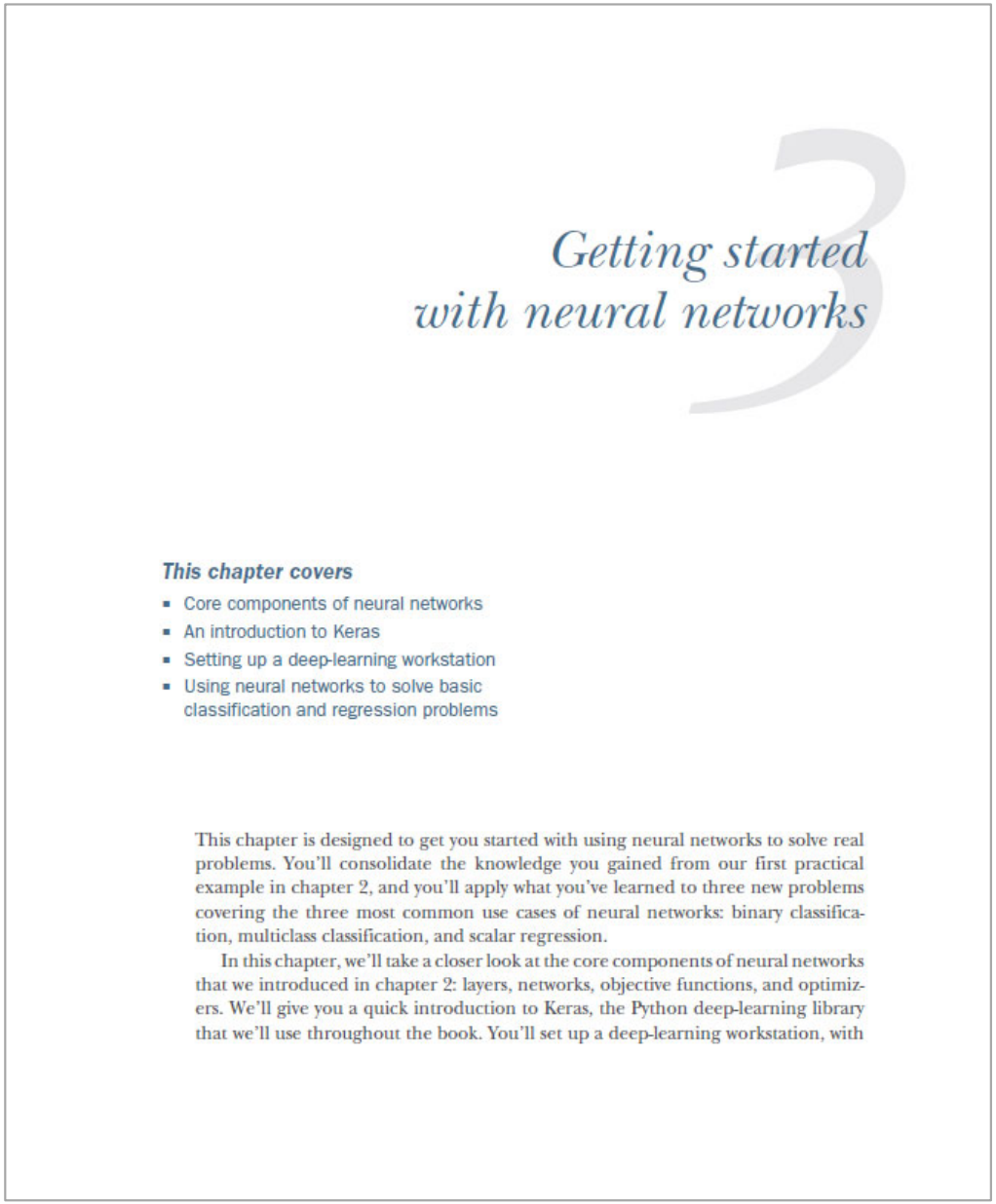
منابع

منبع اصلی



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

Chapter 3



3 Getting started with neural networks

This chapter covers

- Core components of neural networks
- An introduction to Keras
- Setting up a deep-learning workstation
- Using neural networks to solve basic classification and regression problems

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

In this chapter, we'll take a closer look at the core components of neural networks that we introduced in chapter 2: layers, networks, objective functions, and optimizers. We'll give you a quick introduction to Keras, the Python deep-learning library that we'll use throughout the book. You'll set up a deep-learning workstation, with