یادگیری عمیق

جلسه ۷

پیش‌نیازهای یادگیری عمیق (۱)

Preliminaries of Deep Learning (1)

کاظم فولادی قلعه

دانشکده مهندسی، دانشکدگان فارابی

دانشگاه تهران

http://courses.fouladi.ir/deep

پیش‌نیازهای یادگیری عمیق (۱)

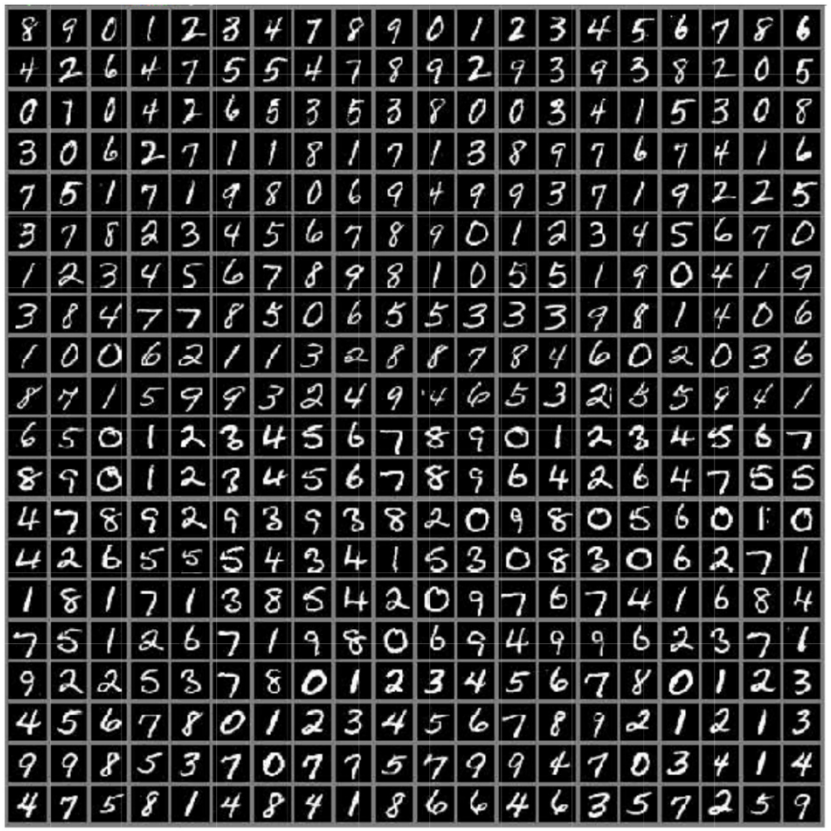
۱

مقدمات

بازشناسی ارقام دست‌نویس

HANDWRITTEN DIGIT RECOGNITION

The problem we're trying to solve here is to classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 through 9).



MNIST Dataset

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.

مفاهیم مورد استفاده در مسائل طبقه‌بندی

CONCEPTS USED IN CLASSIFICATION PROBLEMS

دسته *Category*	**طبقه** *Class*	**کلاس** *Class*	مفهوم یک دسته در مسئله‌ی طبقه‌بندی
نمونه *Sample*	**نقطه داده‌ای** *Data Point*	مفهوم یک نقطه‌ی داده در مسئله‌ی طبقه‌بندی	
برچسب *Label*	طبقه‌ای که یک نمونه‌ی خاص به آن نسبت داده شده است		

بازشناسی ارقام دست‌نویس

HANDWRITTEN DIGIT RECOGNITION

MNIST sample digits

The MNIST dataset comes preloaded in Keras,
in the form of a set of four `Numpy` arrays.

بازشناسی ارقام دست‌نویس

بار کردن مجموعه داده

Loading the MNIST dataset in Keras

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
```

```
>>> train_images.shape
(60000, 28, 28)
>>> len(train_labels)
60000
>>> train_labels
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
>>> test_images.shape
(10000, 28, 28)
>>> len(test_labels)
10000
>>> test_labels
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

بازشناسی ارقام دست‌نویس

تعریف معماری شبکه

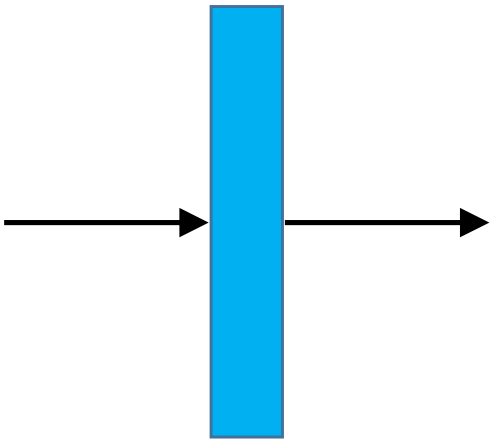
The network architecture

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

لایه

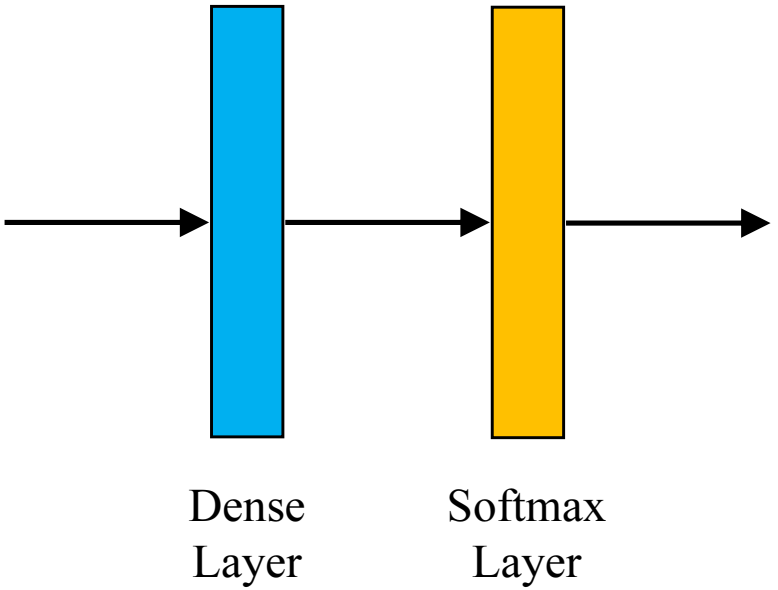
LAYER

لایه *Layer*	یک ماژول پردازش داده که می‌توان آن را به‌عنوان یک فیلتر (صافی) تصور کرد.



بازشناسی ارقام دست‌نویس

شبکه‌ی عصبی دو لایه



اجزای لازم برای آموزش شبکه‌ی عصبی

سایر اجزای لازم برای آموزش شبکه
Other Components for Training Network

تابع اتلاف *Loss Function*	برای اینکه شبکه قادر شود کارآیی خود را بر روی مجموعه‌ی آموزشی اندازه‌گیری کند و از طریق آن بتواند خودش را در مسیر درست قرار دهد.
بهینه‌ساز *Optimizer*	مکانیسمی که از طریق آن شبکه خودش را بر اساس داده‌هایی که می‌بیند و تابع اتلاف به‌هنگام‌سازی کند.
متریک نظارت *Monitoring Metrics*	معیاری برای نظارت بر شبکه در حین آموزش و آزمایش (مانند دقت – *Accuracy*: نسبت تصاویری که درست طبقه‌بندی شدند)

بازشناسی ارقام دست‌نویس

گام کامپایل کردن شبکه‌ی عصبی

The compilation step

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

بازشناسی ارقام دست‌نویس

آماده‌سازی داده‌ها و برچسب آنها

Preparing the image data

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255
```

Preparing the labels

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

بازشناسی ارقام دست‌نویس

آموزش شبکه

```
>>> model.fit(train_images, train_labels, epochs=5, batch_size=128)
Epoch 1/5
60000/60000 [==============================] - 5s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [=====================>.....] - ETA: 1s - loss: 0.1035 - acc: 0.9692
```

```
>>> test_digits = test_images[0:10]
>>> predictions = model.predict(test_digits)
>>> predictions[0]
array([1.0726176e-10, 1.6918376e-10, 6.1314843e-08, 8.4106023e-06,
2.9967067e-11, 3.0331331e-09, 8.3651971e-14, 9.9999106e-01,
2.6657624e-08, 3.8127661e-07], dtype=float32)
```

Each number of index `i` in that array corresponds to the probability that digit image `test_digits[0]` belongs to class `i`.

This first test digit has the highest probability score (0.99999106, almost 1) at index 7, so according to our model, it must be a 7:

```
>>> predictions[0].argmax()
7
>>> predictions[0][7]
0.99999106
```

We can check that the test label agrees:

```
>>> test_labels[0]
7
```

On average, how good is our model at classifying such never-before-seen digits? Let's check by computing average accuracy over the entire test set.

بازشناسی ارقام دست‌نویس

آموزش شبکه

Evaluating the model on new data

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"test_acc: {test_acc}")
test_acc: 0.9785
```

overfitting

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

یادگیری عمیق

پیش‌نیازهای یادگیری عمیق (۱)

۲

تانسورها

تانسور

TENSOR

به‌طور کلی، همه‌ی سیستم‌های یادگیری ماشینی موجود
از تانسورها به‌عنوان ساختمان‌داده‌ی پایه‌ی خود استفاده می‌کنند.

تانسور *Tensor*	آرایه‌ی چندبعدی: حاوی اعداد

بُعد *Dimension*	**محور** *Axis*	در ادبیات تانسورها، به بُعد، معمولاً «محور» گفته می‌شود.

تانسور صفر–بعدی: **اسکالر**
تانسور یک–بعدی: **بردار**
تانسور دو–بعدی: **ماتریس**

تانسورها

اسکالرها (تانسورهای صفر-بعدی)

SCALARS (0D TENSORS)

تانسور صفر-بعدی *0D Tensor*	تانسوری که فقط حاوی یک عدد باشد: اسکالر، تانسور اسکالر، تانسور صفر-بعدی

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

تانسور اسکالر، دارای صفر محور است (`ndim == 0`)

رتبه *Rank*	تعداد محورهای یک تانسور

تانسورها

بردارها (تانسورهای یک-بعدی)

VECTORS (1D TENSORS)

تانسور یک-بعدی *1D Tensor*	آرایه‌ای از اعداد: بردار

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

مثال فوق، یک بردار ۴ بعدی (4-dimensional vector) است.

تانسور یک-بعدی، دارای یک محور است (`ndim == 1`)

تانسورها

ماتریس‌ها (تانسورهای دو–بعدی)

MATRICES (2D TENSORS)

تانسور دو–بعدی *2D Tensor*	آرایه‌ای از بردارها: ماتریس

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

ماتریس، یک جدول از اعداد است.

تانسور دو–بعدی، ماتریس، دارای دو محور است (`ndim == 2`)
که به آنها سطر (row) و ستون (column) می‌گویند.

تانسورها

تانسورهای سه–بعدی و تانسورهای ابعاد بالاتر

3D TENSORS AND HIGHER-DIMENSIONAL TENSORS

```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

تانسورها

خصیصه‌های کلیدی

KEY ATTRIBUTES

خصیصه‌های کلیدی تانسورها
Key Attributes of Tensors

تعداد محورها (رتبه) *Number of Axes (Rank)*	`ndim`
شکل *Shape*	یک چندتایی مرتب که تعداد ابعاد در امتداد هر محور تانسور را بیان می‌کند. `shape` `eg. (3,5) (3,3,5) (5,) ()`
نوع داده‌ای *Data Type*	نوع داده‌های موجود در تانسور `dtype` `eg. uint8 float32 float64`

تانسور از نوع char به‌ندرت استفاده می‌شود.
تانسور رشته‌ای در NumPy وجود ندارد (به‌دلیل ملاحظات تخصیص حافظه).

تانسورها

خصیصه‌های کلیدی: مثال

KEY ATTRIBUTES

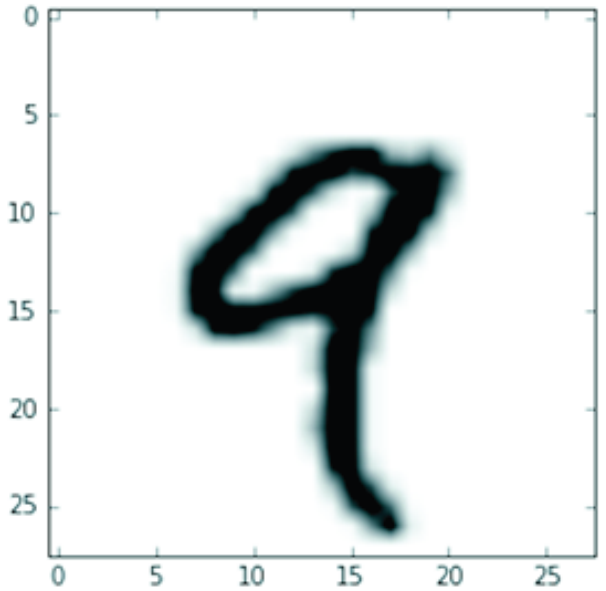
```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
>>> print(train_images.ndim)
3
>>> print(train_images.shape)
(60000, 28, 28)
>>> print(train_images.dtype)
uint8
```

تانسورها

نمایش تصویر

Displaying the fourth digit

```
digit = train_images[4]
import matplotlib.pyplot as plt
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```



تانسورها

دست‌کاری تانسورها در Numpy

MANIPULATING TENSORS IN NUMPY

برش تانسور
Tensor Slicing

انتخاب عناصر خاص در یک تانسور

```
>>> my_slice = train_images[10:100]
>>> print(my_slice.shape)
(90, 28, 28)
```

```
>>> my_slice = train_images[10:100, :, :]
>>> my_slice.shape
(90, 28, 28)
>>> my_slice = train_images[10:100, 0:28, 0:28]
>>> my_slice.shape
(90, 28, 28)
```

Equivalent to the previous example

Also equivalent to the previous example

```
my_slice = train_images[:, 14:, 14:]
```

```
my_slice = train_images[:, 7:-7, 7:-7]
```

تانسورها

مفهوم دسته‌های داده

THE NOTION OF DATA BATCHES

در مجموعه داده‌های مورد استفاده در یادگیری عمیق،
نخستین محور تانسور**محور نمونه‌ها (samples axis)** است.

دسته‌ی داده *Data Batch*	مجموعه‌ای از داده‌ها که برای پردازش به‌طور همزمان به شبکه داده می‌شود.

مدل‌های یادگیری عمیق، کل مجموعه داده را به‌صورت یک‌باره و همزمان پردازش نمی‌کنند؛
بلکه داده‌ها را به دسته‌های کوچکی (small batches) تقسیم می‌کنند.

```
# one batch of our MNIST digits, with batch size of 128 (index: 0..127)
batch = train_images[:128]
```

```
# the second batch of our MNIST digits, with batch size of 128
batch = train_images[128:256]
```

```
# the n-th batch of our MNIST digits, with batch size of 128
batch = train_images[128 * n:128 * (n + 1)]
```

به محور اول تانسور، **محور دسته** (batch axis) یا **بعد دسته** (batch dimension) می‌گوییم.

بازنمایی داده‌ها برای شبکه‌های عصبی

مثال‌های دنیای واقعی از تانسورهای داده‌ای

REAL-WORLD EXAMPLES OF DATA TENSORS

داده‌های برداری *Vector Data*	تانسورهای دوبعدی به شکل: (samples, features)
داده‌های سری‌های زمانی/ **داده‌های دنباله‌ای** *Timeseries Data / Sequence Data*	تانسورهای سه‌بعدی به شکل (samples, timesteps, features)
تصویرها *Images*	تانسورهای چهاربعدی به شکل‌های (samples, height, width, channels) (samples, channels, height, width)
ویدئو *Video*	تانسورهای پنج‌بعدی به شکل‌های (samples, frames, height, width, channels) (samples, frames, channels, height, width)

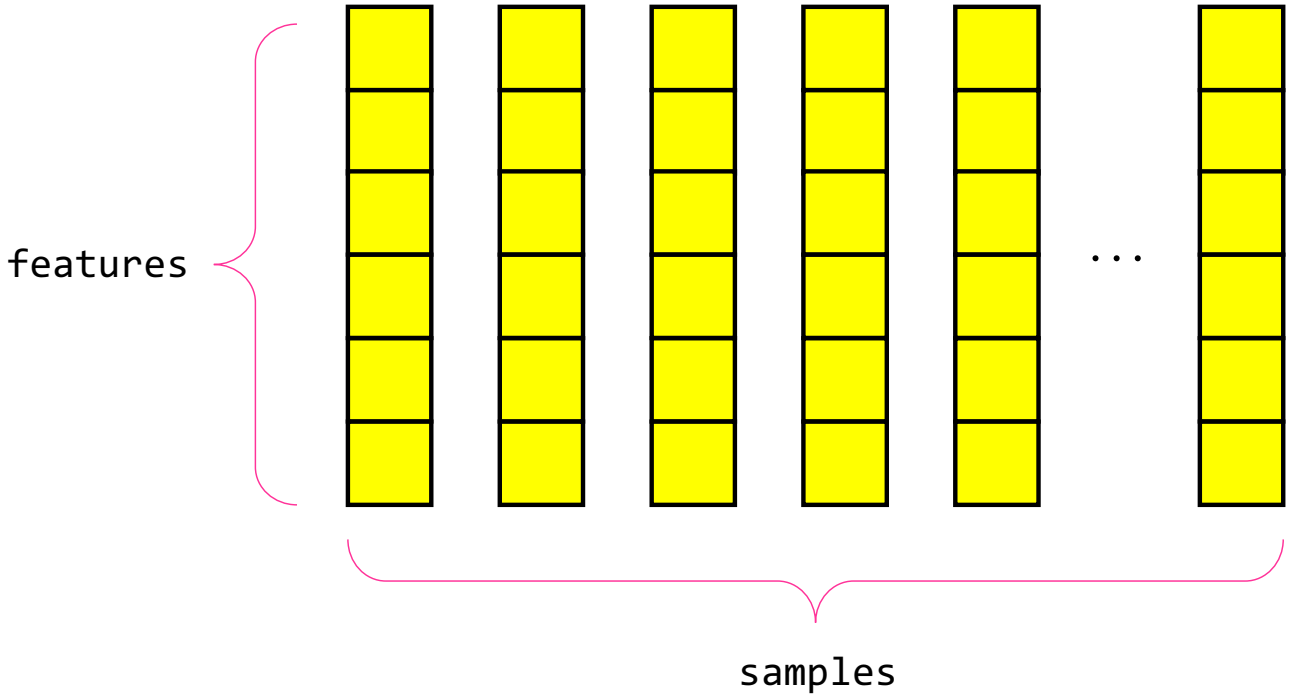
بازنمایی داده‌ها برای شبکه‌های عصبی

داده‌های برداری

VECTOR DATA

داده‌های برداری
Vector Data

تانسورهای دوبعدی به شکل:
`(samples, features)`



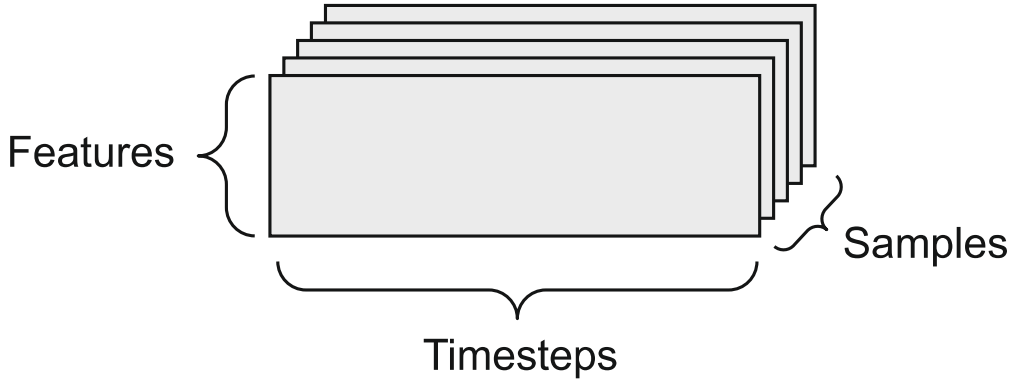
بازنمایی داده‌ها برای شبکه‌های عصبی

داده‌های سری زمانی یا داده‌های دنباله‌ای

TIMESERIES DATA OR SEQUENCE DATA

داده‌های سری‌های زمانی / داده‌های دنباله‌ای
Timeseries Data / Sequence Data

تانسورهای سه‌بعدی به شکل
`(samples, timesteps, features)`



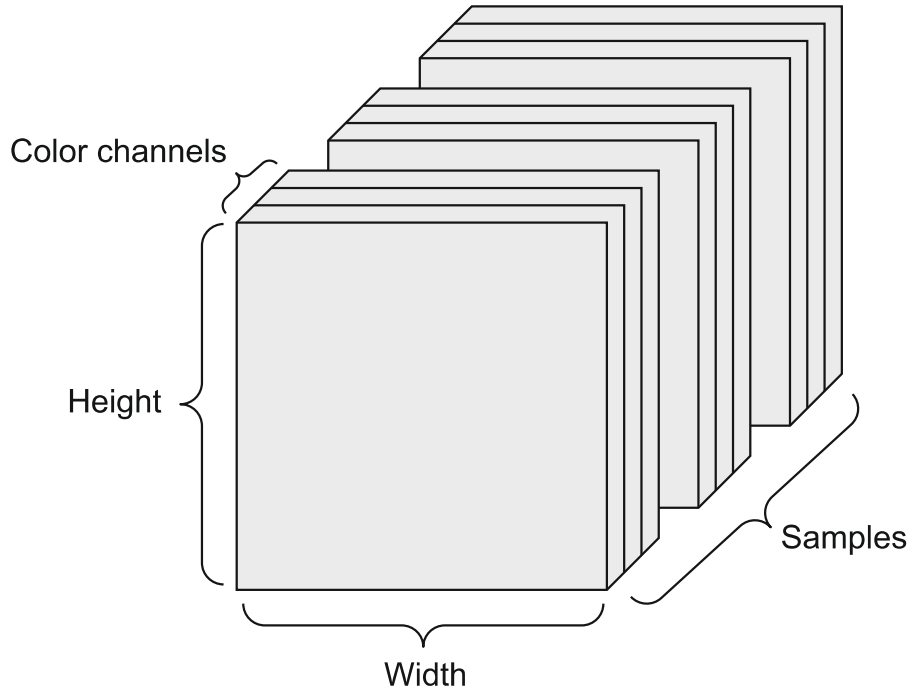
طبق قرارداد، زمان همیشه محور دوم (اندیس 1) است.

بازنمایی داده‌ها برای شبکه‌های عصبی

داده‌های تصویر

IMAGE DATA

تصویرها *Images*	تانسورهای چهاربعدی به شکل‌های (samples, height, width, channels) (samples, channels, height, width)



A 4D image data tensor (channels-first convention) [Theano]

بازنمایی داده‌ها برای شبکه‌های عصبی

داده‌های ویدئو

VIDEO DATA

ویدئو *Video*	تانسورهای پنج‌بعدی به شکل‌های (samples, frames, height, width, channels) (samples, frames, channels, height, width)

A video can be understood as a sequence of frames, each frame being a color image.

عملیات تانسوری

چرخ‌دنده‌های شبکه‌های عصبی

THE GEARS OF NEURAL NETWORKS: TENSOR OPERATIONS

همان‌گونه که هر برنامه‌ی کامپیوتری نهایتاً به مجموعه‌ی کوچکی از عملیات دودویی
بر روی ورودی‌های دودویی
(مانند AND، OR، NOR، ...)
کاهش می‌یابد،
همه‌ی تبدیل‌هایی که توسط شبکه‌های عصبی عمیق یادگرفته می‌شوند،
می‌توانند به تعدادی عملیات تانسوری که
بر روی تانسورهایی از داده‌های عددی اعمال می‌شود،
کاهش یابند.

```
output = relu(dot(W, input) + b)
```

عملیات تانسوری

عملیات عنصر به عنصر

ELEMENT-WISE OPERATIONS

عملیات عنصر به عنصر
Element-wise operations

عملیاتی که به‌طور مستقل بر روی هر درایه از تانسور اعمال می‌شود.
(⇐ قابلیت بالا برای پیاده‌سازی‌های موازی انبوه = پیاده‌سازی برداری‌شده)

```
def naive_relu(x):
    assert len(x.shape) == 2
    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] = max(x[i, j], 0)
    return x
```

x is a 2D Numpy tensor.

Avoid overwriting the input tensor.

```
def naive_add(x, y):
    assert len(x.shape) == 2
    assert x.shape == y.shape
    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] += y[i, j]
    return x
```

x and y are 2D Numpy tensors.

Avoid overwriting the input tensor.

عملیات تانسوری

عملیات عنصر به عنصر

ELEMENT-WISE OPERATIONS

عملیات عنصر به عنصر
Element-wise operations

عملیاتی که به‌طور مستقل بر روی هر درایه از تانسور اعمال می‌شود.
(⇐ قابلیت بالا برای پیاده‌سازی‌های موازی انبوه = پیاده‌سازی برداری‌شده)

```
import numpy as np

z = x + y                    ⟵ Element-wise addition

z = np.maximum(z, 0.)        ⟵ Element-wise relu
```

عملیات تانسوری

پخش

BROADCASTING

وقتی شکل دو تانسور با هم متفاوت باشد، چگونه می‌توانیم آنها را با هم جمع کنیم؟

هرگاه ممکن باشد و هیچ‌گونه ابهامی وجود نداشته باشد،
تانسور کوچکتر پخش می‌شود (broadcasted) تا با شکل تانسور بزرگتر مطابقت پیدا کند.

عمل پخش از دو مرحله تشکیل می‌شود:

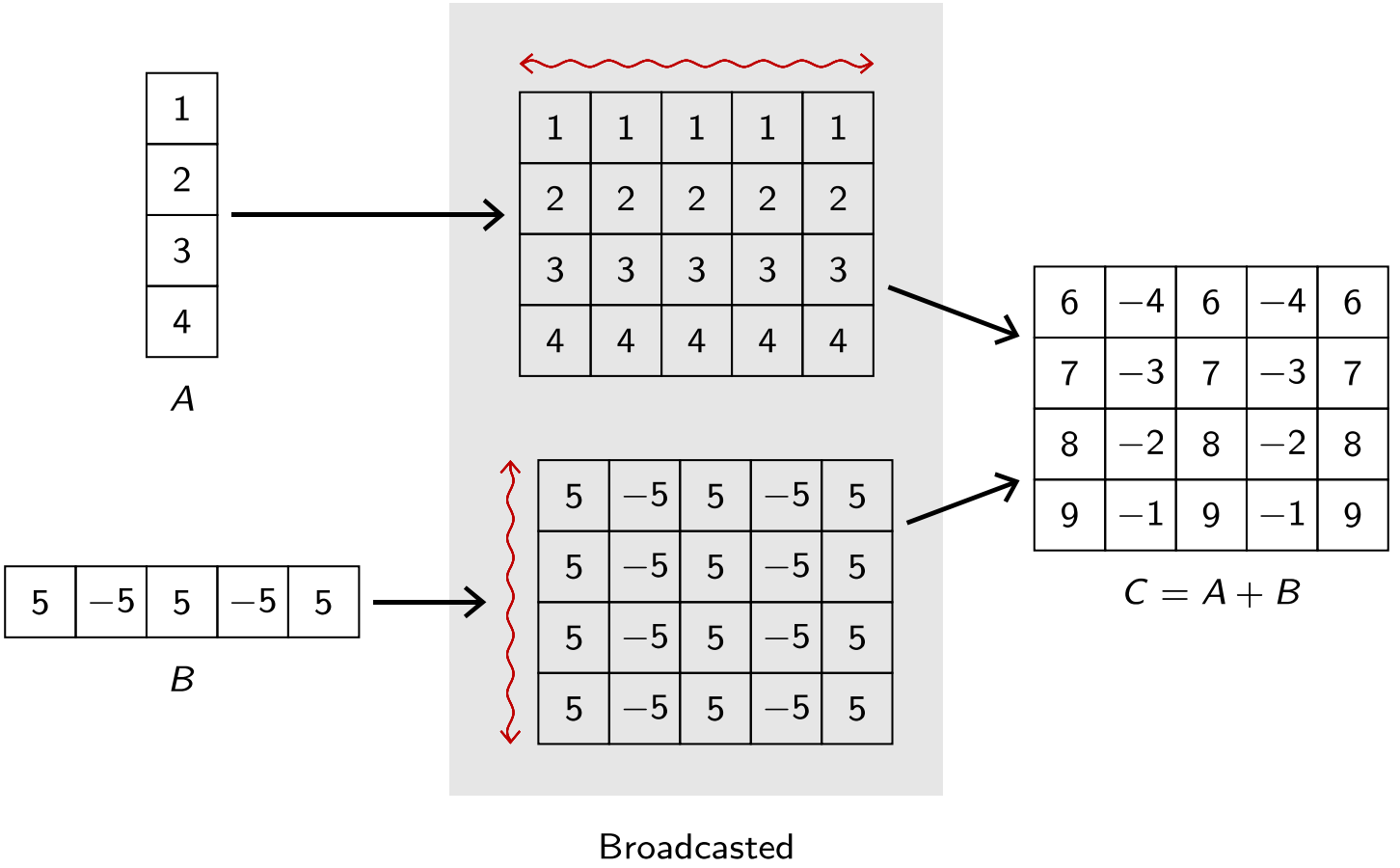
۱. محورهایی (محورهای پخش: *broadcast axes*) به تانسور کوچکتر اضافه می‌شود تا با `ndim` تانسور بزرگتر مطابقت پیدا کند.
۲. تانسور کوچکتر در راستای این محورهای جدید تکرار می‌شود تا با شکل کل تانسور بزرگتر مطابقت پیدا کند.

Ex. Consider `X` with shape (32, 10) and `y` with shape (10,).
First, we add an empty first axis to `y`, whose shape becomes (1, 10).
Then, we repeat `y` 32 times alongside this new axis,
so that we end up with a tensor `Y` with shape (32, 10), where `Y[i,:] == y` for `i` in range(0, 32).
At this point, we can proceed to add `X` and `Y`, because they have the same shape.

عملیات تانسوری

پخش: مثال

BROADCASTING



عملیات تانسوری

پخش: پیاده‌سازی

BROADCASTING

در پیاده‌سازی، تانسور جدیدی در حافظه با اندازه‌ی بزرگ‌تر ساخته نمی‌شود،
این کار در سطح الگوریتمی اتفاق می‌افتد.

یک نمونه پیاده‌سازی خام عمل پخش به‌صورت زیر است:

```
def naive_add_matrix_and_vector(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 1
    assert x.shape[1] == y.shape[0]

    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] += y[j]
    return x
```

x is a 2D Numpy tensor.

y is a Numpy vector.

Avoid overwriting the input tensor.

عملیات تانسوری

پخش: مثال

BROADCASTING

With broadcasting, you can generally apply two-tensor element-wise operations if one tensor has shape `(a, b, … n, n + 1, … m)` and the other has shape `(n, n + 1, … m)`. The broadcasting will then automatically happen for axes `a` through `n - 1`.

```
import numpy as np

x = np.random.random((64, 3, 32, 10))
y = np.random.random((32, 10))

z = np.maximum(x, y)
```

x is a random tensor with shape (64, 3, 32, 10).

y is a random tensor with shape (32, 10).

The output z has shape (64, 3, 32, 10) like x.

عملیات تانسوری

ضرب نقطه‌ای تانسوری – ضرب تانسوری

TENSOR DOT

ضرب تانسوری (بر خلاف ضرب عنصر به عنصر)، درایه‌های موجود در تانسورهای ورودی را ترکیب می‌کند.

```
import numpy as np
z = np.dot(x, y)
```

با نمادگذاری ریاضی:

```
z = x . y
```

عملیات تانسوری

ضرب نقطه‌ای تانسوری – ضرب تانسوری: ضرب نقطه‌ای دو بردار

TENSOR DOT

```
def naive_vector_dot(x, y):
    assert len(x.shape) == 1
    assert len(y.shape) == 1
    assert x.shape[0] == y.shape[0]
    z = 0.
    for i in range(x.shape[0]):
        z += x[i] * y[i]
    return z
```

x and y are Numpy vectors.

عملیات تانسوری

ضرب نقطه‌ای تانسوری – ضرب تانسوری: ضرب نقطه‌ای ماتریس و بردار

TENSOR DOT

```
import numpy as np

def naive_matrix_vector_dot(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 1
    assert x.shape[1] == y.shape[0]

    z = np.zeros(x.shape[0])
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            z[i] += x[i, j] * y[j]
    return z
```

x is a Numpy matrix.

y is a Numpy vector.

The first dimension of x must be the same as the 0th dimension of y!

This operation returns a vector of 0s with the same shape as y.

```
def naive_matrix_vector_dot(x, y):
    z = np.zeros(x.shape[0])
    for i in range(x.shape[0]):
        z[i] = naive_vector_dot(x[i, :], y)
    return z
```

عملیات تانسوری

ضرب نقطه‌ای تانسوری – ضرب تانسوری: ضرب نقطه‌ای دو تانسور با ابعاد بزرگ‌تر از یک

TENSOR DOT

متداول‌ترین حالت، ضرب نقطه‌ای دو ماتریس است:

```
def naive_matrix_dot(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 2
    assert x.shape[1] == y.shape[0]

    z = np.zeros((x.shape[0], y.shape[1]))
    for i in range(x.shape[0]):
        for j in range(y.shape[1]):
            row_x = x[i, :]
            column_y = y[:, j]
            z[i, j] = naive_vector_dot(row_x, column_y)
    return z
```

x and y are Numpy matrices.

The first dimension of x must be the same as the 0th dimension of y!

This operation returns a matrix of 0s with a specific shape.

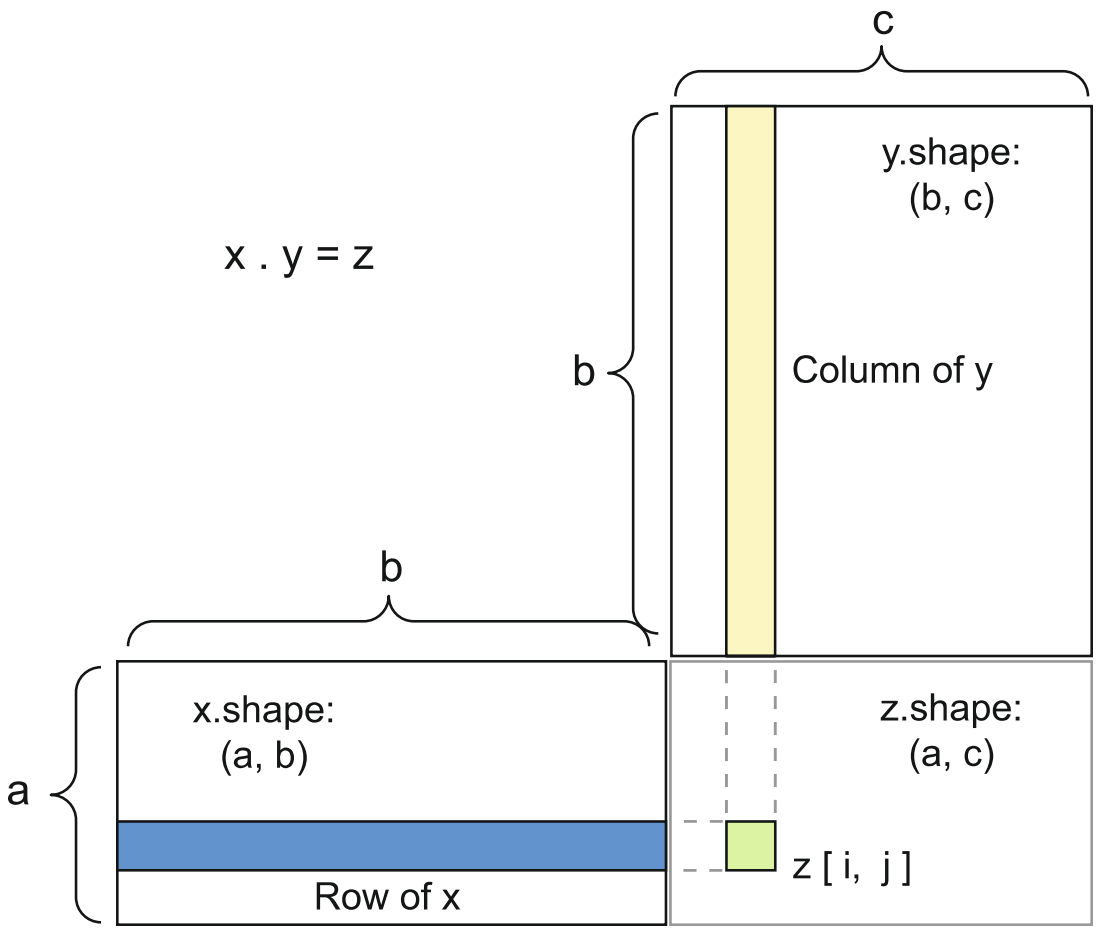
Iterates over the rows of x …

… and over the columns of y.

عملیات تانسوری

ضرب نقطه‌ای تانسوری – ضرب تانسوری

TENSOR DOT



عملیات تانسوری

تغییر شکل تانسور

TENSOR RESHAPING

تغییر شکل تانسور
Tensor Reshaping

تغییر شکل یک تانسور، به‌معنی بازآرایی سطرها و ستون‌های آن برای تطابق با شکل مقصد است.

```
>>> x = np.array([[0., 1.],
[2., 3.],
[4., 5.]])
>>> print(x.shape)
(3, 2)

>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
[ 1.],
[ 2.],
[ 3.],
[ 4.],
[ 5.]])
>>> x = x.reshape((2, 3))
>>> x
array([[ 0., 1., 2.],
[ 3., 4., 5.]])
```

عملیات تانسوری

تغییر شکل تانسور: (حالت خاص: ترانهاده کردن)

TENSOR RESHAPING

ترانهاده کردن *Transposition*	ترانهاده کرده یک ماتریس، به‌معنی تعویض سطرها با ستون‌های آن ماتریس است.

```
>>> x = np.zeros((300, 20))
>>> x = np.transpose(x)
>>> print(x.shape)
(20, 300)
```

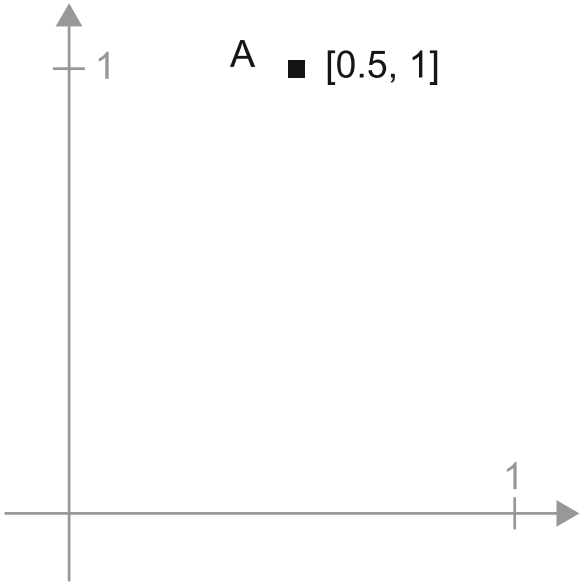
Creates an all-zeros matrix of shape (300, 20)

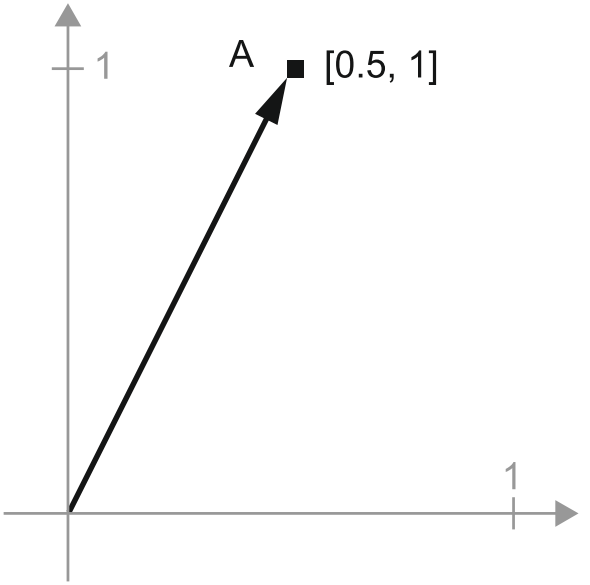
عملیات تانسوری

تعبیر هندسی عملیات تانسوری

GEOMETRIC INTERPRETATION OF TENSOR OPERATIONS

هر عنصر یک تانسور n–بعدی، در قالب یک نقطه در فضای n–بعدی قابل نمایش است.



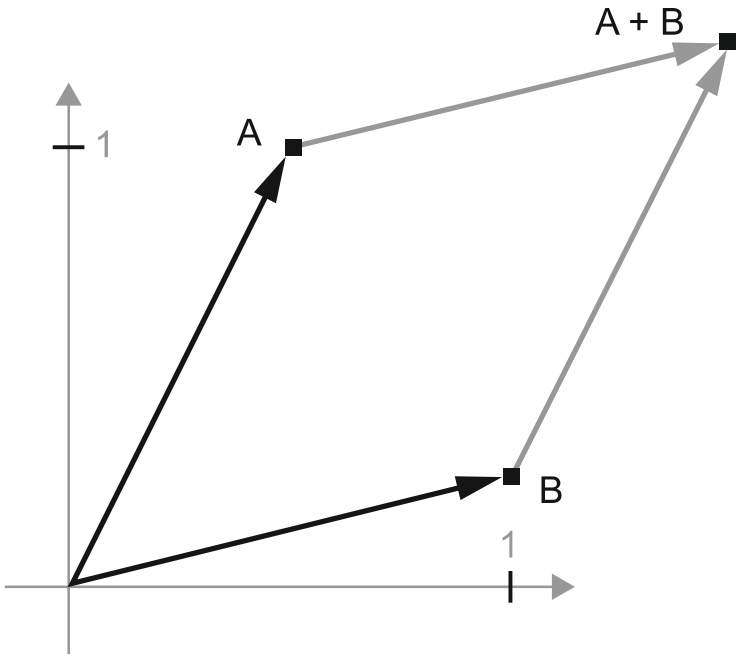


عملیات تانسوری

تعبیر هندسی عملیات تانسوری

GEOMETRIC INTERPRETATION OF TENSOR OPERATIONS

جمع دو بردار:



عملیات تانسوری

تعبیر هندسی عملیات تانسوری

GEOMETRIC INTERPRETATION OF TENSOR OPERATIONS

جابه‌جایی به‌عنوان جمع برداری

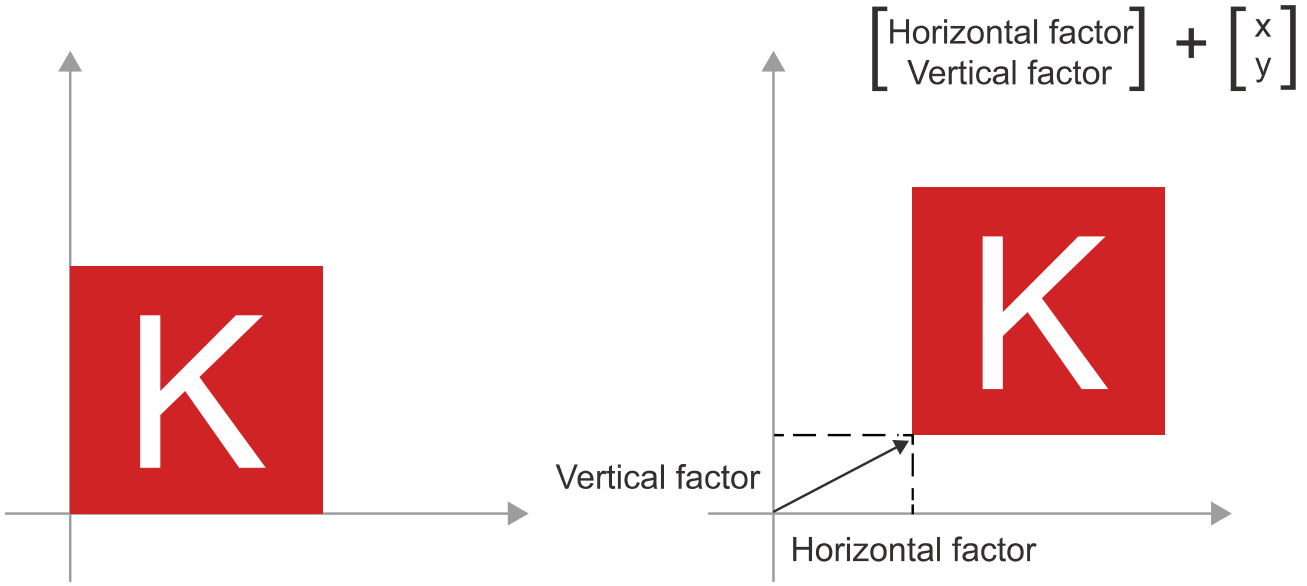


Figure 2.9 2D translation as a vector addition

In general, elementary geometric operations such as translation, rotation, scaling, skewing, and so on can be expressed as tensor operations. Here are a few examples:

- *Translation*: As you just saw, adding a vector to a point will move the point by a fixed amount in a fixed direction. Applied to a set of points (such as a 2D object), this is called a "translation" (see figure 2.9).

عملیات تانسوری

تعبیر هندسی عملیات تانسوری

GEOMETRIC INTERPRETATION OF TENSOR OPERATIONS

دوران دوبعدی (پادساعت‌گرد) به‌عنوان ضرب نقطه‌ای

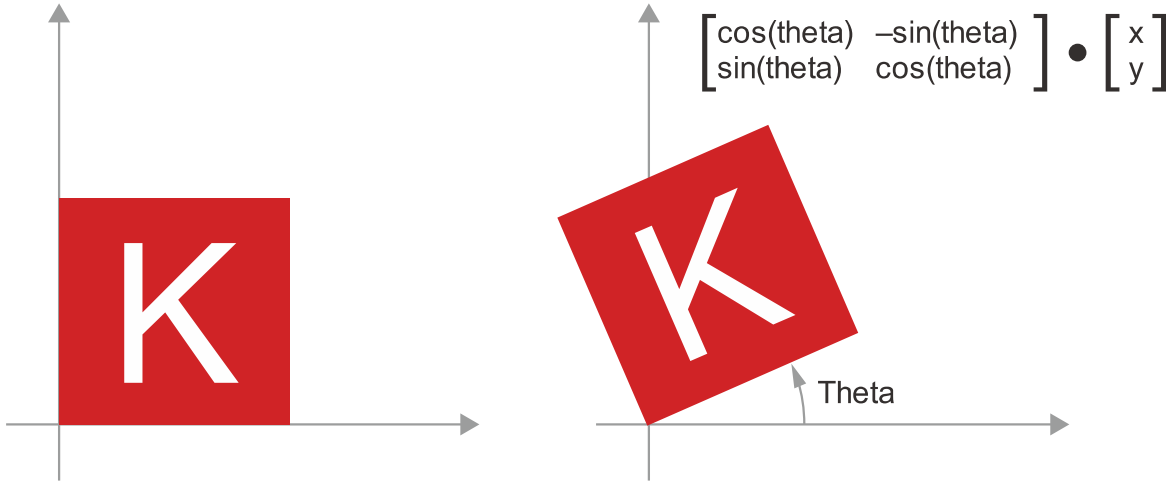


Figure 2.10 2D rotation (counterclockwise) as a dot product

- *Rotation*: A counterclockwise rotation of a 2D vector by an angle theta (see figure 2.10) can be achieved via a dot product with a 2 × 2 matrix `R = [[cos(theta), -sin(theta)], [sin(theta), cos(theta)]]`.

عملیات تانسوری

تعبیر هندسی عملیات تانسوری

GEOMETRIC INTERPRETATION OF TENSOR OPERATIONS

تغییر مقیاس دوبعدی به‌عنوان ضرب نقطه‌ای

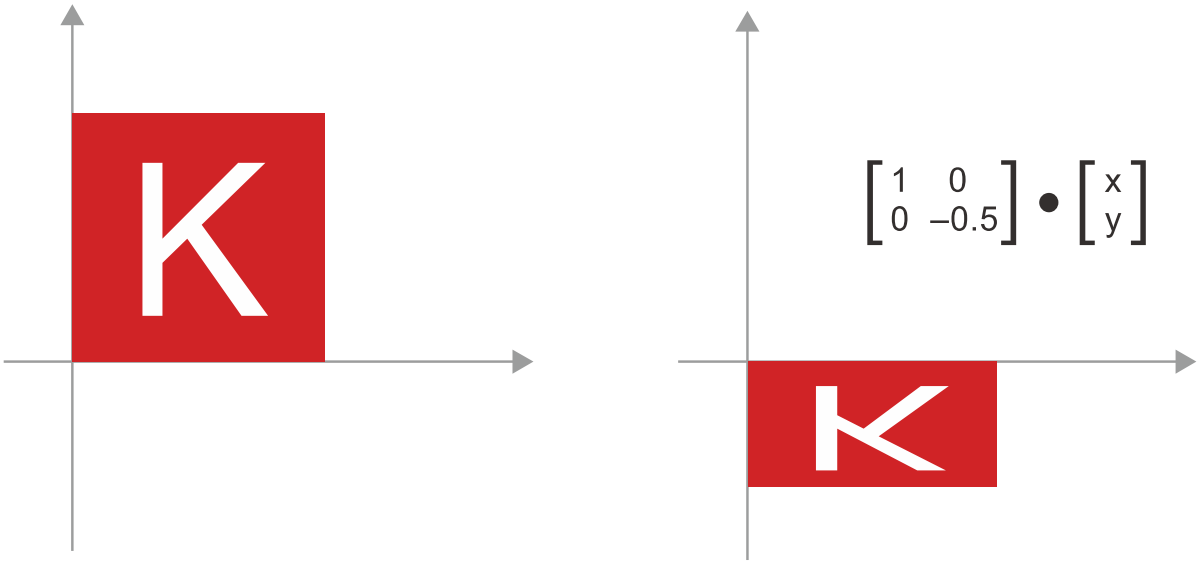


Figure 2.11
2D scaling as a dot product

- *Scaling*: A vertical and horizontal scaling of the image (see figure 2.11) can be achieved via a dot product with a 2 × 2 matrix `S = [[horizontal_factor, 0], [0, vertical_factor]]` (note that such a matrix is called a "diagonal matrix," because it only has non-zero coefficients in its "diagonal," going from the top left to the bottom right).

عملیات تانسوری

تعبیر هندسی عملیات تانسوری

GEOMETRIC INTERPRETATION OF TENSOR OPERATIONS

تبدیل آفین در صفحه

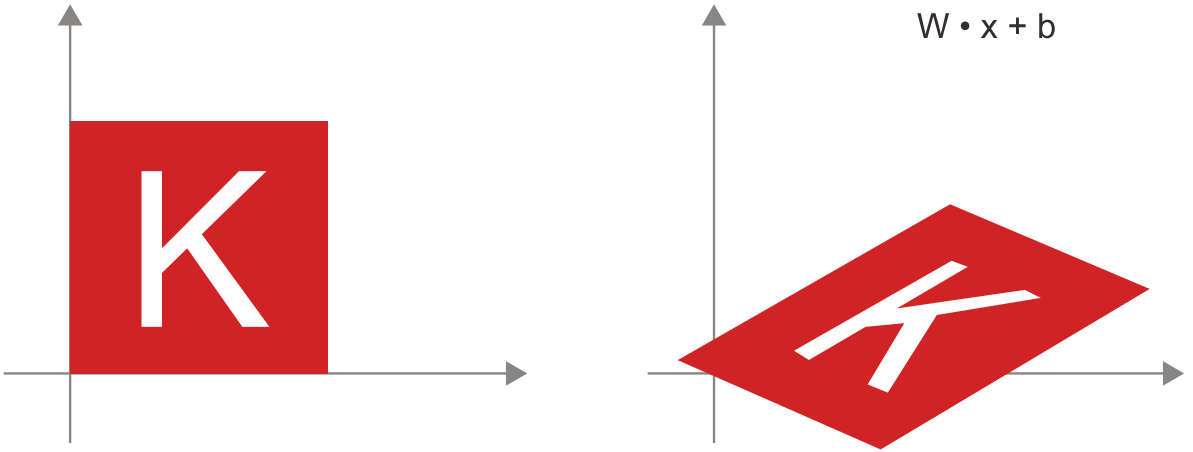


Figure 2.12 Affine transform in the plane

- *Linear transform*: A dot product with an arbitrary matrix implements a linear transform. Note that *scaling* and *rotation*, listed previously, are by definition linear transforms.
- *Affine transform*: An affine transform (see figure 2.12) is the combination of a linear transform (achieved via a dot product with some matrix) and a translation (achieved via a vector addition). As you have probably recognized, that's exactly the `y = W x + b` computation implemented by the `Dense` layer! A `Dense` layer without an activation function is an affine layer.

عملیات تانسوری

تعبیر هندسی عملیات تانسوری

GEOMETRIC INTERPRETATION OF TENSOR OPERATIONS

تبدیل آفین که به‌دنبال آن تابع فعالیت relu آمده است

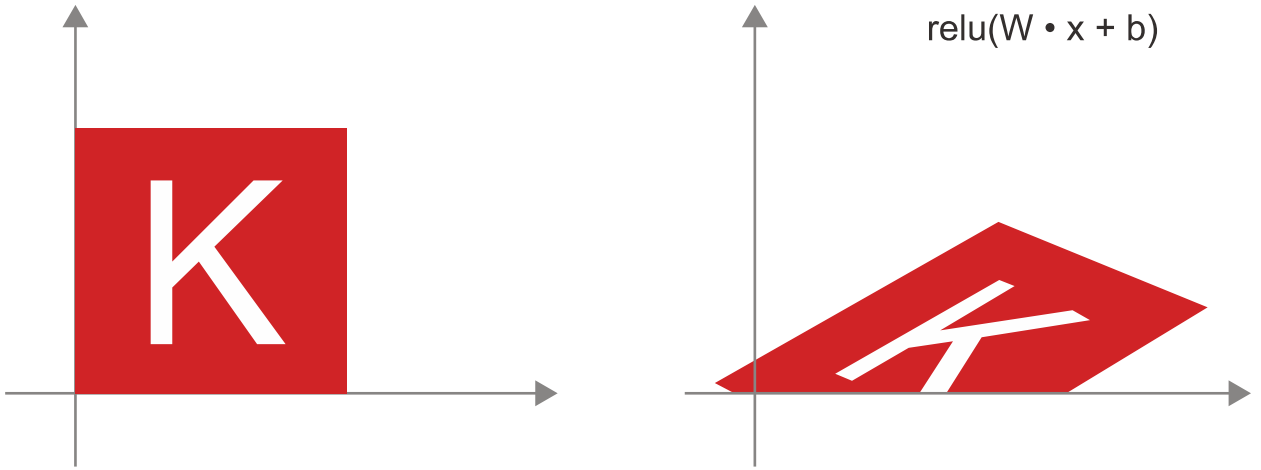


Figure 2.13 Affine transform followed by `relu` activation

- `Dense` *layer with* `relu` *activation*: An important observation about affine transforms is that if you apply many of them repeatedly, you still end up with an affine transform (so you could just have applied that one affine transform in the first place). Let's try it with two: `affine2(affine1(x)) = W2 • (W1 • x + b1) + b2 = (W2 • W1) • x + (W2 • b1 + b2)`. That's an affine transform where the linear part is the matrix `W2 • W1` and the translation part is the vector `W2 • b1 + b2`.

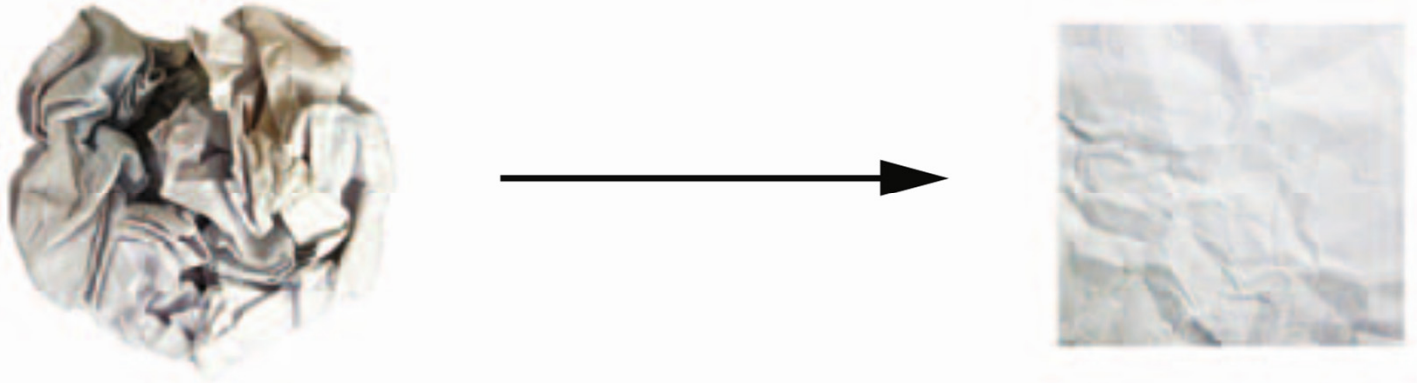
عملیات تانسوری

تعبیر هندسی یادگیری عمیق

A GEOMETRIC INTERPRETATION OF DEEP LEARNING

شبکه‌های عصبی از دنباله‌ای از عملیات تانسوری تشکیل شده‌اند و
هر عملیات تانسوری دقیقاً یک تبدیل هندسی بر روی داده‌های ورودی آن است.

دو کاغذ رنگی (دو کلاس) را روی هم قرار می‌دهیم و آنها را مچاله می‌کنیم تا یک توپ ساخته شود (داده‌های ورودی).
شبکه‌ی عصبی باید تبدیلی را پیاده‌سازی کند که این توپ از حالت مچالگی باز شود.



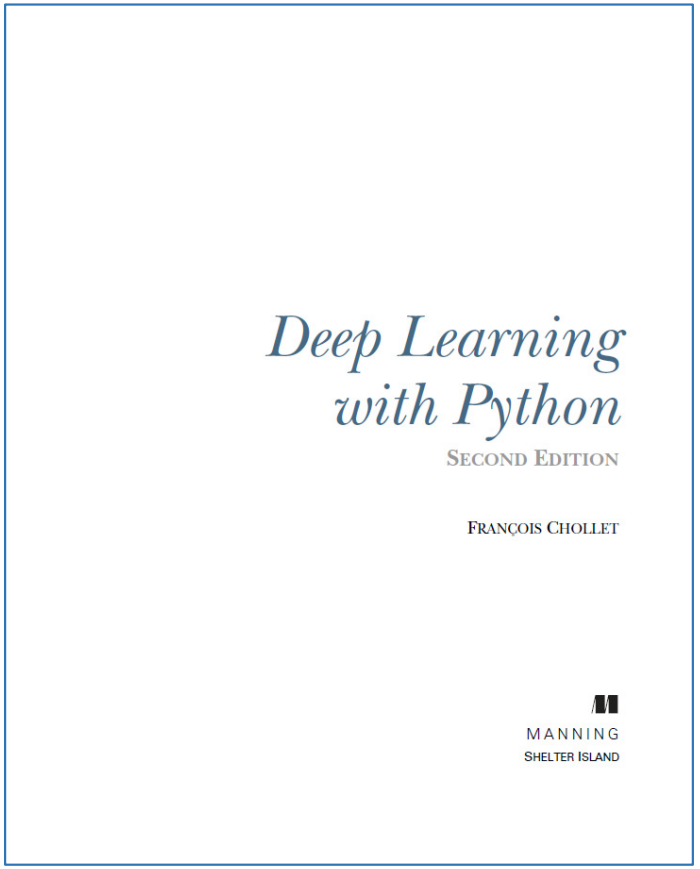
روی‌کرد یادگیری عمیق، تجزیه‌ی گام به گام یک تبدیل هندسی پیچیده
به یک زنجیره‌ی طولانی از تبدیل‌های ابتدایی است.
هر لایه در یک شبکه‌ی عمیق یک تبدیل اعمال می‌کند که داده‌ها را کمی باز می‌کند
و یک پشته‌ی عمیق از لایه‌ها یک فرآیند بازشدگی بسیار پیچیده را امکان‌پذیر می‌کند.

پیش‌نیازهای یادگیری عمیق (۱)

۳

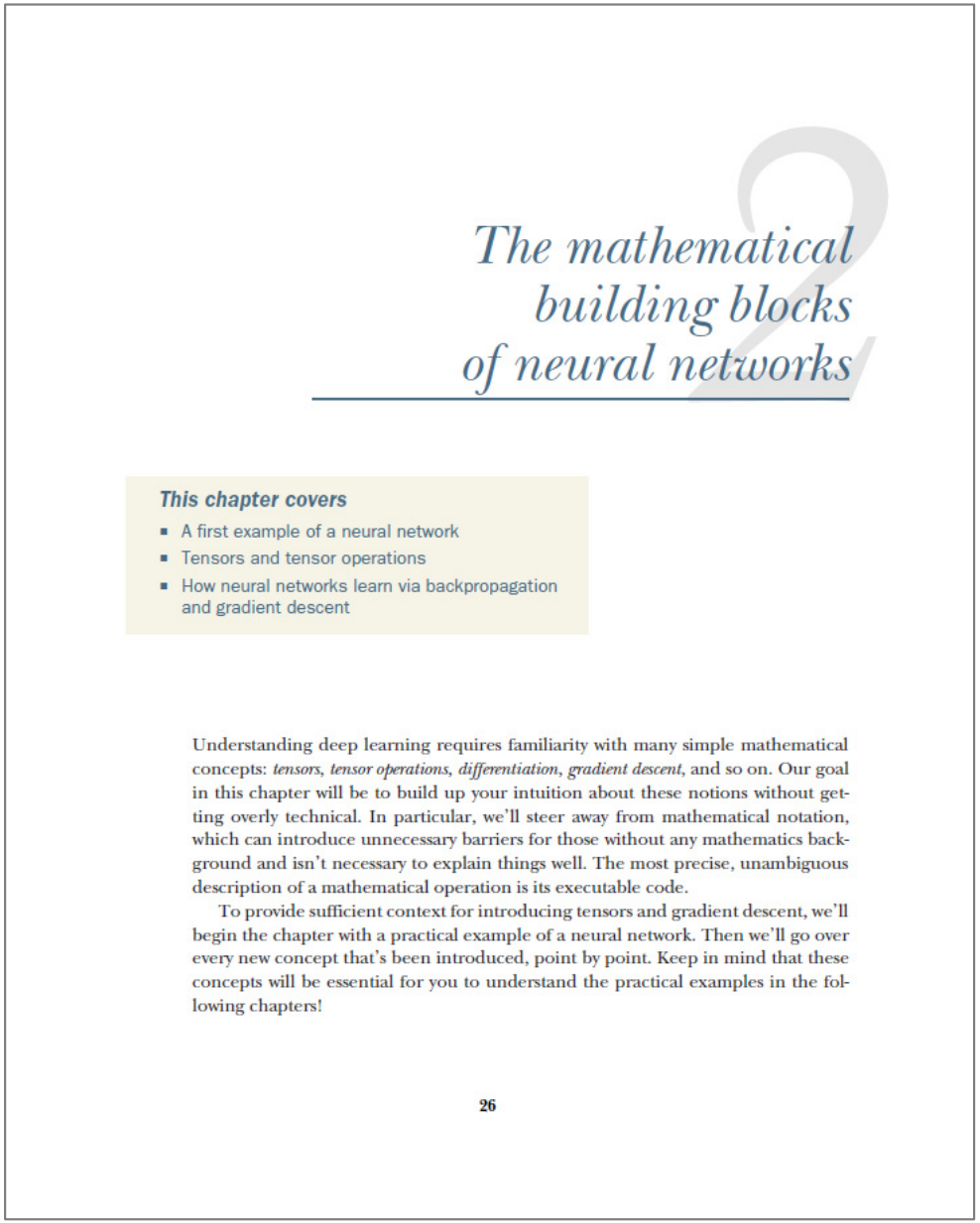
منابع

منبع اصلی



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

Chapter 2



2

The mathematical building blocks of neural networks

This chapter covers

- A first example of a neural network
- Tensors and tensor operations
- How neural networks learn via backpropagation and gradient descent

Understanding deep learning requires familiarity with many simple mathematical concepts: *tensors*, *tensor operations*, *differentiation*, *gradient descent*, and so on. Our goal in this chapter will be to build up your intuition about these notions without getting overly technical. In particular, we'll steer away from mathematical notation, which can introduce unnecessary barriers for those without any mathematics background and isn't necessary to explain things well. The most precise, unambiguous description of a mathematical operation is its executable code.

To provide sufficient context for introducing tensors and gradient descent, we'll begin the chapter with a practical example of a neural network. Then we'll go over every new concept that's been introduced, point by point. Keep in mind that these concepts will be essential for you to understand the practical examples in the following chapters!

26