CH2. Mathematical Building Blocks of Deep-learning

Il-Youp Kwak

Chung-Ang University

Mathematical Building Blocs of Deep-learning

A first look on neural network basic example & construction Tensorflow keras

Data representations for neural network

The gear of neural networks: tensor operations

The engine of neural networks: gradient-based optimization

Looking back at our first example

A first look on neural network

Figure 2.1 MNIST sample digits

Mnist data analysis example

hand written digit data set

Hand written digit (28,28 pixcels) for 10 categories (0-9)

60,000 training set and 10,000 test set

* There is keras module in Tensorflow. (TF)
- 케라스는 텐서플로라 해도 무방 오픈소스, 텐서플로에서도 케라스 사용
- In keras module, there is some widely popular used dataset. (MNIST is one of that)

Mnist data

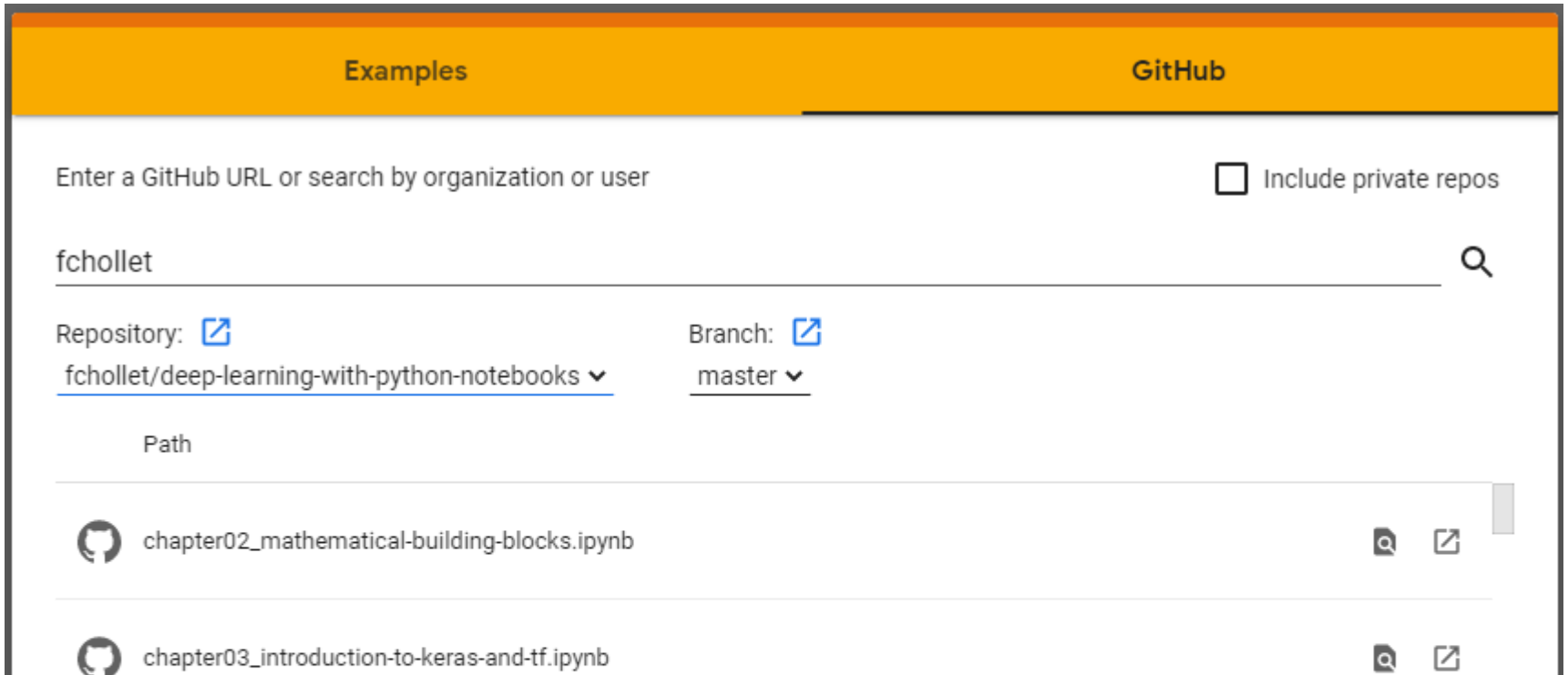
already in tensorflow module

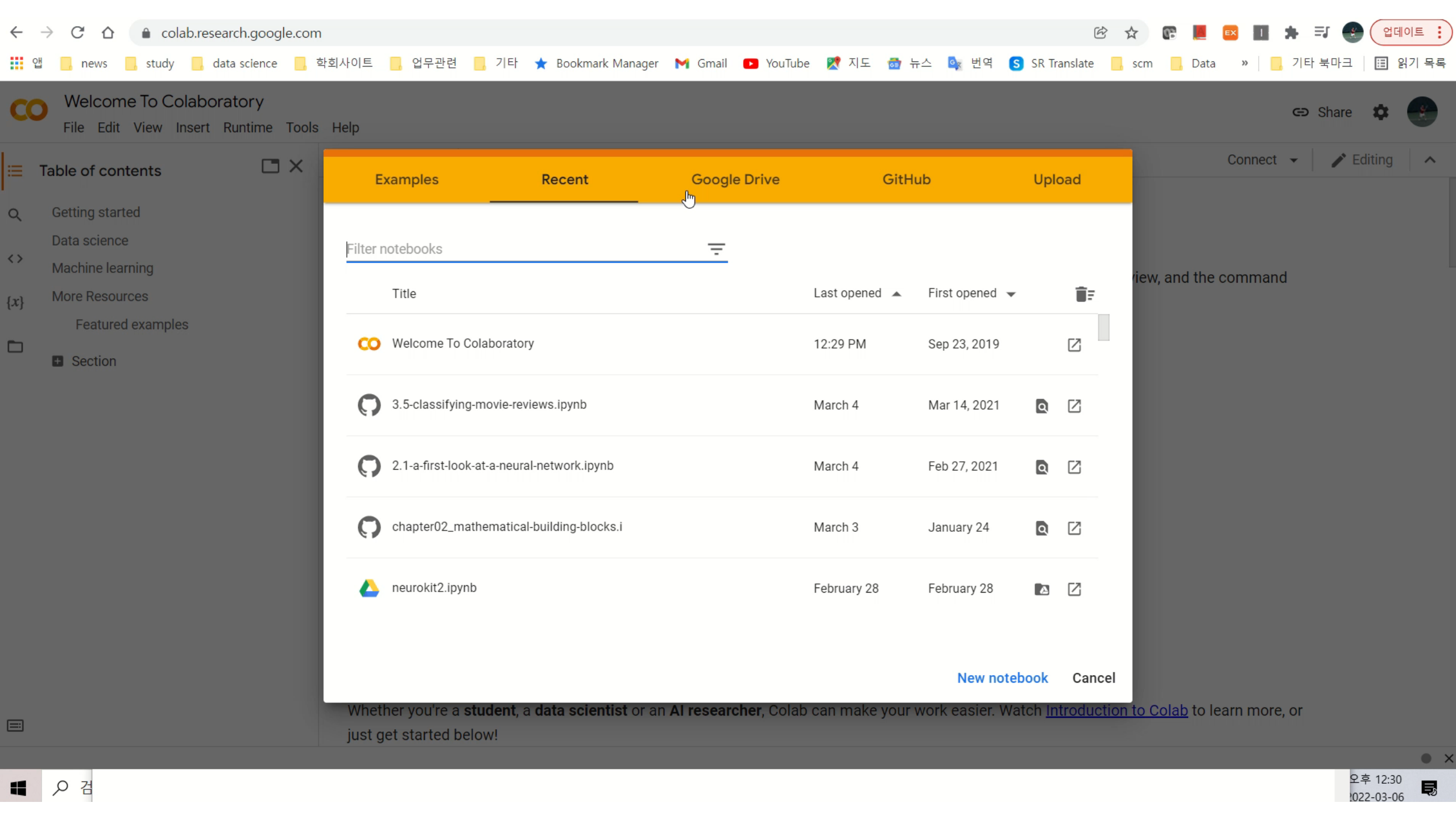
Listing 2.1 Loading the MNIST dataset in Keras

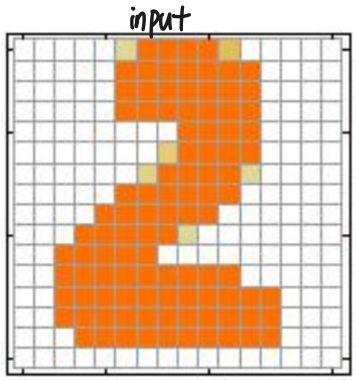
```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

{ calling importing } library, we load mnist data.

Try these codes from https://colab.research.google.com/github/




colab.research.google.com
업데이트
앱 news study data science 학회사이트 업무관련 기타 Bookmark Manager Gmail YouTube 지도 뉴스 번역 SR Translate scm Data 기타 북마크 읽기 목록
Welcome To Colaboratory
File Edit View Insert Runtime Tools Help
Share
Connect
Editing
Table of contents
Getting started
Data science
Machine learning
More Resources
Featured examples
Section
Examples
Recent
Google Drive
GitHub
Upload
Filter notebooks
Title
Last opened
First opened
Welcome To Colaboratory 12:29 PM Sep 23, 2019
3.5-classifying-movie-reviews.ipynb March 4 Mar 14, 2021
2.1-a-first-look-at-a-neural-network.ipynb March 4 Feb 27, 2021
chapter02_mathematical-building-blocks.i March 3 January 24
neurokit2.ipynb February 28 February 28
New notebook
Cancel
Whether you're a student, a data scientist or an AI researcher, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!
오후 12:30
2022-03-06



we define this model

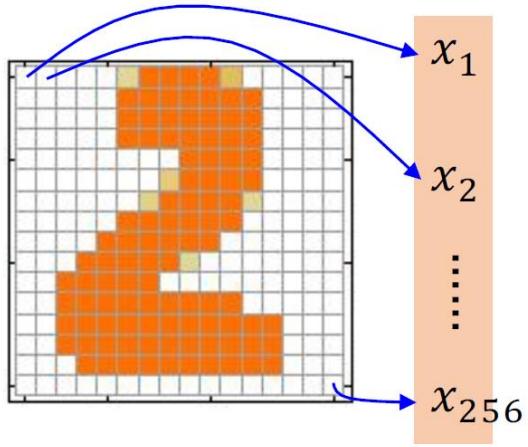
Machine

classifies the result.

"2"

output vector should be length '10'. (0~9)

input



machine (model)

output

	scores	
$\hat{y}_1$	0.09	is "1"
$\hat{y}_2$	0.71	is "2"
$\hat{y}_{10}$	0.03	is "0"

largest → classifies input as "2"

the image is "2"

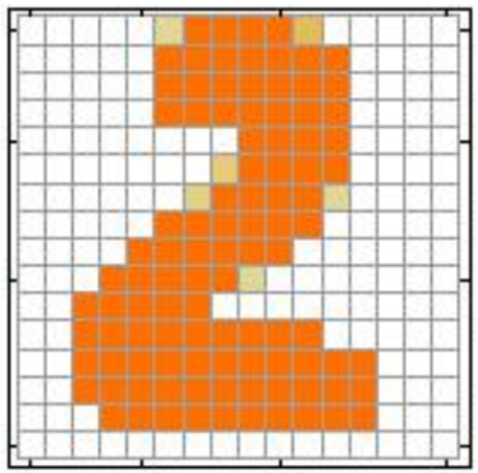
$16 \times 16 = 256$

ink → 1

no ink → 0

into input vector

Each dimension represents the confidence of a digit.



input

x_1

x_2

$\vdots$

x_{256}

Machine

output

$\hat{y}_1$

$\hat{y}_2$

$\vdots$

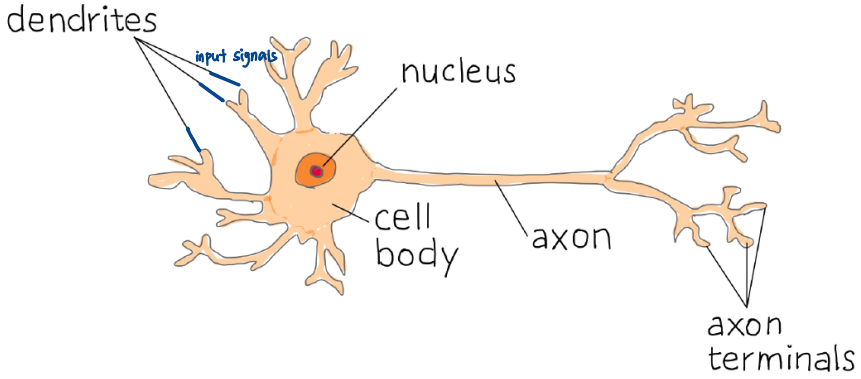
$\hat{y}_{10}$

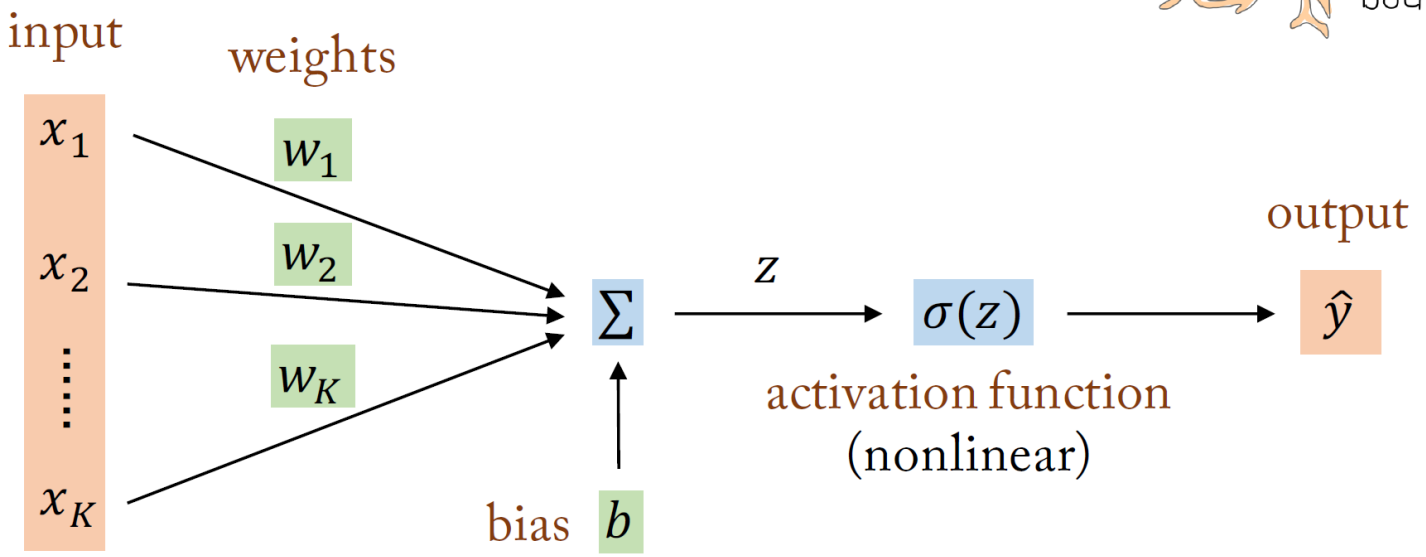
$f : R^{256} \rightarrow R^{10}$

In Deep Learning, the function f is represented by a Neural Network.

Perceptron (neuron) [Frank Rosenblatt, 1957]

- The simplest structure of DL is one-layer model.





$$f : \mathbb{R}^K \to \mathbb{R}$$

$= \sigma(Wx+b)$ (P×K, K×1, P×1; P×1 vector)

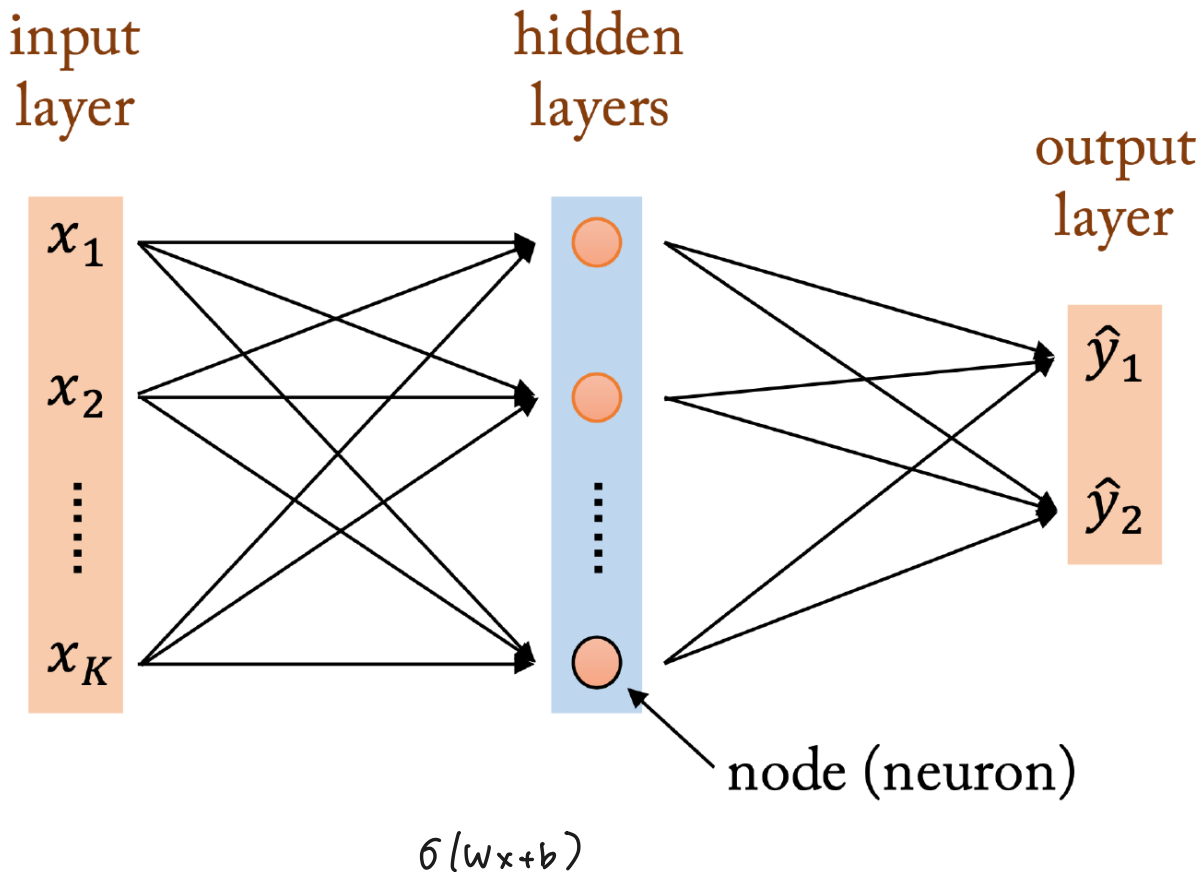
similar to dense layer.

$$\hat{y} = \sigma(z) = \sigma(w_1x_1 + w_2x_2 + \cdots + w_Kx_K + b)$$

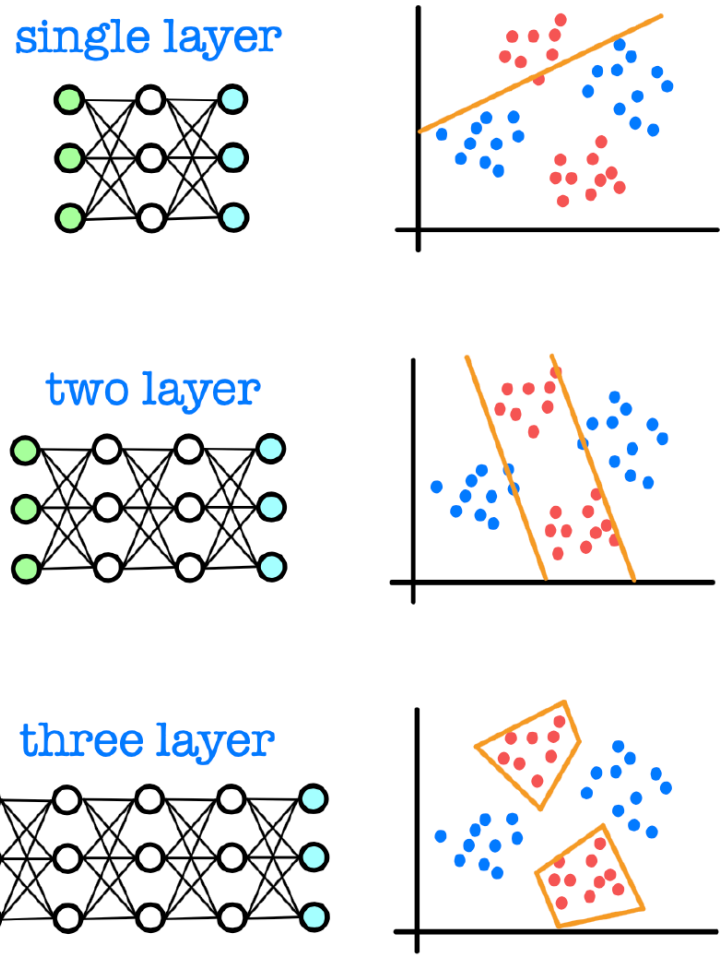
nonlinear — weighted sum (linear)

Multilayer Perceptron (MLP)

- Applying perceptron model layer by layer (= multi layer)



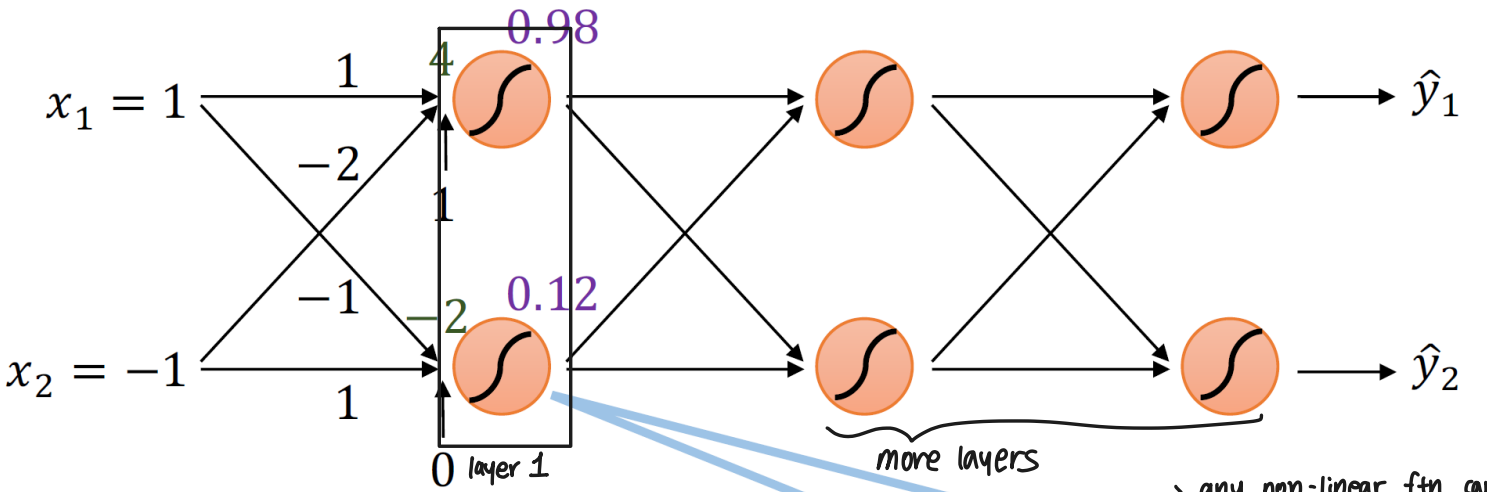
$\sigma(Wx+b)$



Forward pass computation 순전파

$$\sigma(w_1 x_1 + w_2 x_2 + \cdots + w_K x_K + b)$$

$$0.98 = \sigma(1 \cdot 1 + (-1) \cdot (-2) + 1)$$



activation ftn

$$\sigma\left(\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right)$$

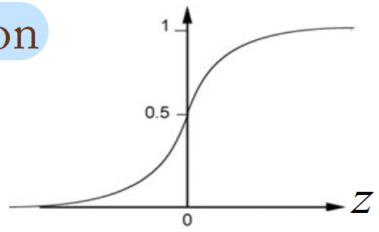
$$= \sigma\left(\begin{bmatrix} 4 \\ -2 \end{bmatrix}\right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

W, b trainable parameters.
After setting up DL model,
we should optimize weights (w) & bias (b)
to make model better.

any non-linear ftn can be used in activation ftn.
eg) relu, sigmoid, etc.

eg)
sigmoid function

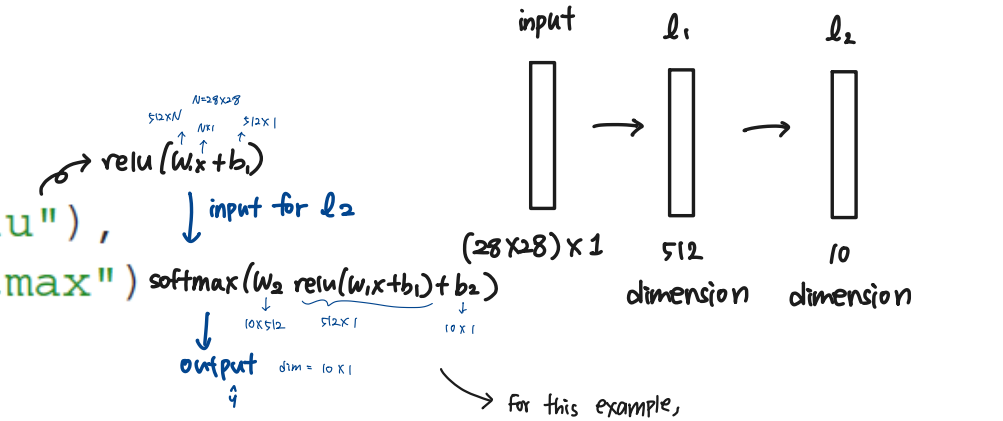
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



activation function

Listing 2.2 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")
])
```



Core building block of NN is layers

Layers extract *representations*

This step define our model f(x; weights)

Table of contents


```
[8] train_images.shape
```

```
(60000, 28, 28)
```

```
[9] len(train_labels)
```

```
60000
```

```
train_labels
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
[11] test_images.shape
```

```
(10000, 28, 28)
```

```
[12] len(test_labels)
```

```
10000
```

```
[13] test_labels
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

The network architecture

step 1) define our model $f_\theta(x)=y$

step 2) define criteria → loss function
How the network measure its performance.

What's next?

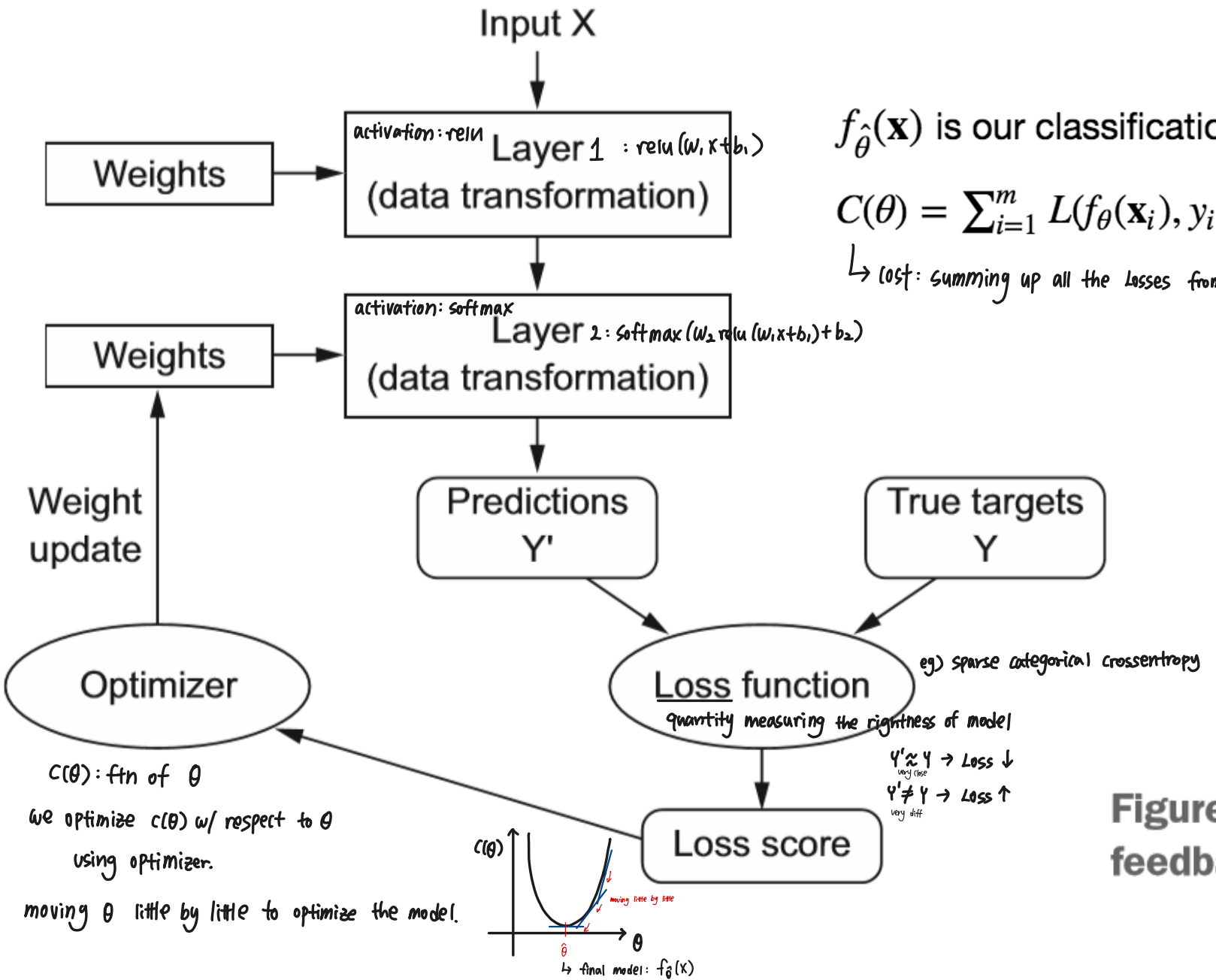
Loss function: How the network measure its performance

Optimizer: Mechanism through which the network will update itself based on the data it sees and its loss function

Metrics to monitor during training and testing: Accuracy

Listing 2.3 The compilation step

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



$f_{\hat{\theta}}(\mathbf{x})$ is our classification model with estimated parameters $\hat{\theta}$

$$C(\theta) = \sum_{i=1}^{m} L(f_\theta(\mathbf{x}_i), y_i)$$

↳ cost: summing up all the losses from i to m.
(all indicies. / # training data set.)

Figure 1.9 The loss score is used as a feedback signal to adjust the weights.

chapter02_mathematical-building-blocks.i

File Edit View Insert Runtime Tools Help Cannot save changes

Share

Table of contents

+ Code + Text Copy to Drive

RAM Disk

Editing

[14] 3s
```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

The compilation step

+ Code + Text

0s
```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

[ ]

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
```

"Fitting" the model

[ ]
```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

0s completed at 1:03 PM

Rescale values from [0,255] to [0,1]

Listing 2.4 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
```

Fitting the model

Listing 2.5 "Fitting" the model

```
>>> 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
```

using training data 5 times

every iteration of the training how many mini batch sample that we are using?

Make Predictions

Listing 2.6 Using the model to make predictions

```
>>> 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)
```

0 1 2 3 4 5 6 7 8 9

the result would be "7".

Evaluating the model

Listing 2.7 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
```

Table of contents

3s

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

The compilation step

[22] 0s

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

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
```

"Fitting" the model

```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

CH2. Mathematical Building Blocks of Deep-learning (2)

Il-Youp Kwak

Chung-Ang University

Data representations for neural networks

Tensor is a container for data

0D tensors (Scalars)

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

1D tensors (Vectors)

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

2D tensors (Matrices)

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

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
```

Key attributes for *tensors*

Number of axes (rank): can be checked by .ndim **for numpy objects**

0D, 1D, 2D

rank

Shape: tuple of integers that describes how many dimensions the tensor has along each axis. Can be checked by .shape **for numpy objects**

Data type (dtype**): type of the data in tensor. Ex:** float32, uint8, float64, **and so on.**

Key attributes for *tensors*

```
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
```

Displaying Digits

```
digit = train_images[4]

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

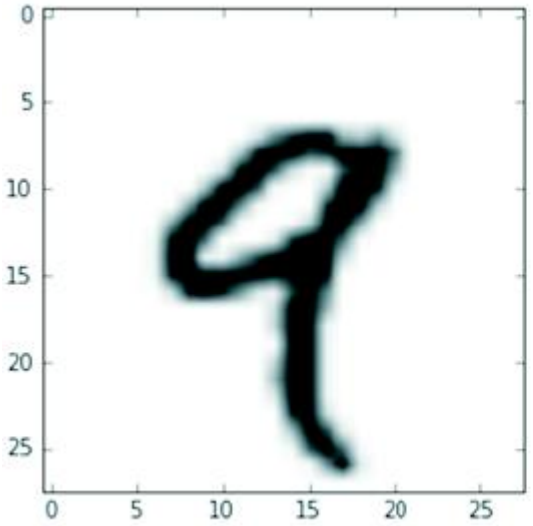


Figure 2.2 The fourth sample in our dataset

chapter02_mathematical-building-blocks.i

File Edit View Insert Runtime Tools Help Cannot save changes

Share

Table of contents

+ Code + Text Copy to Drive

RAM Disk Editing

```
x.ndim
```

Key attributes

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

```
[ ] train_images.ndim
```

```
[ ] train_images.shape
```

```
[ ] train_images.dtype
```

Displaying the fourth digit

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

```
[ ] train_labels[4]
```

Manipulating tensors in NumPy

1s completed at 1:16 PM

Manipulating tensors in Numpy

Tensor slicing

```
>>> 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)
```

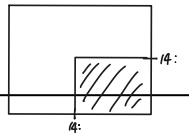
||

|| same

Manipulating tensors in Numpy

Select 14×14 pixels in bottom-right corner of all images

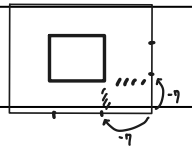
```
my_slice = train_images[:, 14:, 14:]
```



Crop 14 × 14 pixels centered in the middle

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

from backward




```
plt.imshow(my_slice[4], cmap=plt.cm.binary)
plt.show()
```

```
[47] train_labels[4]
```

```
9
```

Manipulating tensors in NumPy

```
my_sli Loading... images[10:100]
my_slice.shape
```

```
[ ] my_slice = train_images[10:100, :, :]
my_slice.shape
```

```
[ ] my_slice = train_images[10:100, 0:28, 0:28]
my_slice.shape
```

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

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

The notion of data batches

```
[ ] batch = train_images[:128]
```

```
[ ] batch = train_images[128:256]
```

The notion of data batches

The 1st axis of data tensors in deep-learning will be sample axis 60,000

Deep-learning models don't process an entire dataset at once Process little by little (using batches) Like mini batch gradient descent algorithm

```
# 1st batch
batch = train_images[:128]
# 2nd batch
batch = train_images[128:256]
# nth batch
batch = train_images[128 * n:128 * (n + 1)]
```

Real-world examples of data tensors

Vector data – 2D tensor of shape (samples, features)

Timeseries data or sequence data – 3D tensors of shape (samples, timesteps, features)

(cf. MNIST : 3D ∵ black-white image ⇒ it means 1 channel)

Images – 4D tensors of shape (samples, height, width, channels) **or** (samples, channels, height, width)

represents colors

sequence of images

it means 'frame'

Video – 5D tensors of shape (samples, frames, height, width, channels) **or** (samples, frames, channels, height, width)

Vector Data

Vector data – 2D tensor of shape (samples, features)

Example:

Dataset of people, where we consider each person's *age*, *ZIP code*, **and** *income*.
Each person can be characterized as a vector of 3 values
Entire dataset of 100,000 people can be stored in a 2D tensor of shape (100000, 3).

Timeseries data or sequence data

Timeseries data or sequence data – 3D tensors of shape (samples, timesteps, features)

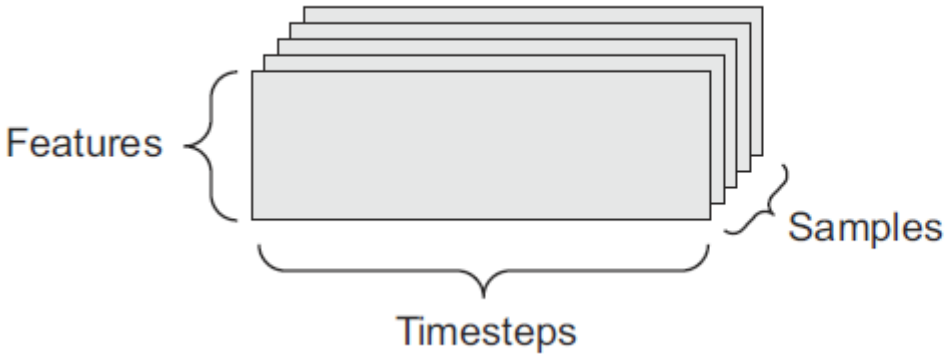


Figure 2.3 A 3D timeseries data tensor

Dataset of stock prices. Every minute, we store the *current price*, *the highest* **and** *the lowest price* **in the past minute. Encoded as a 2D tensor of shape (390, 3) for 390 minutes in a trading day**
250 days' data stored in 3D tensor of shape (250, 390, 3).

Image data

Images – 4D tensors of shape (samples, height, width, channels) **or** (samples, channels, height, width)

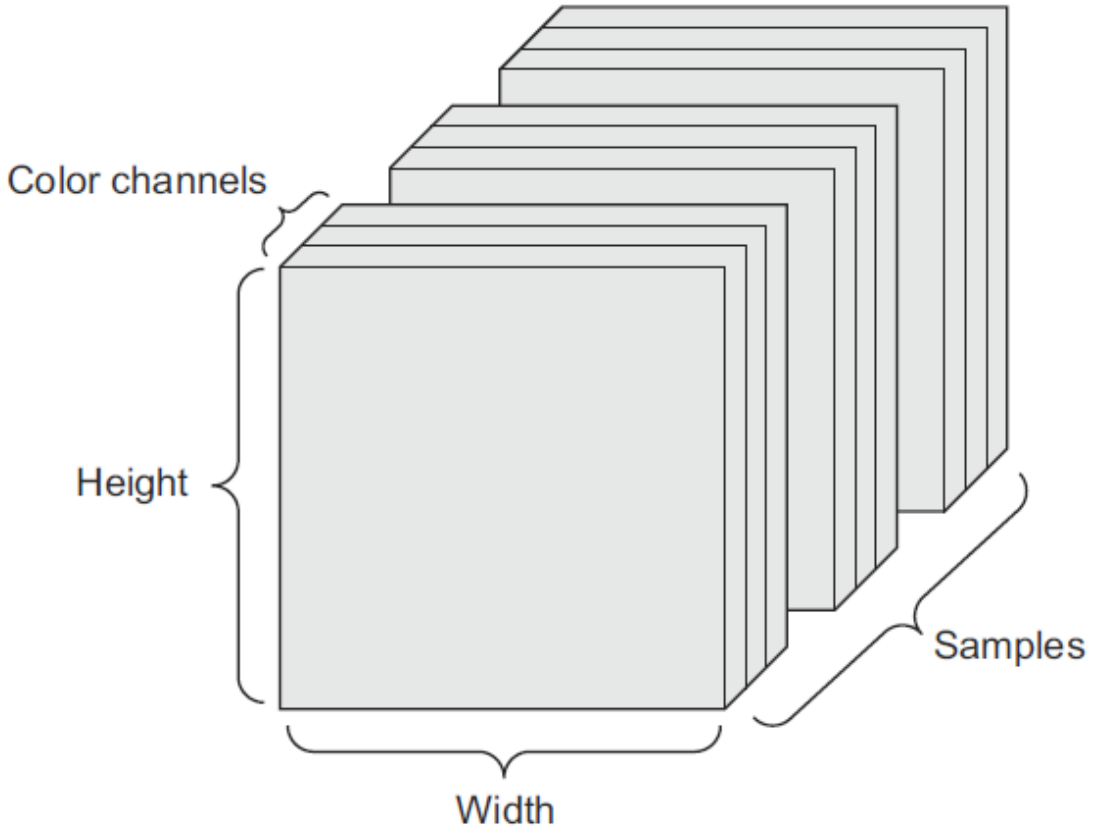


Figure 2.4 A 4D image data tensor (channels-first convention)

Video data

Video – 5D tensors of shape (samples, frames, height, width, channels) **or** (samples, frames, channels, height, width)

Example:

A 60-second, 144 × 256 YouTube video clip sampled at 4 frames per second would have 240 frames. A batch of four such video clips would be stored in a tensor of shape (4, 240, 144, 256, 3)

The gears of neural networks: tensor operations

Tensor operations **applied to tensors of numeric data.**

Example:

keras.layers.Dense(512, activation='relu')

Unpack this

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

relu: max(x, 0) — dot: dot product — +: addition

Three tensor operations:

dot product **(dot) between input tensor and a tensor W**

addition **(+) between the resulting 2D tensor and vector b**

relu operation: **relu(x) is max(x, 0).**

Element-wise operations

Ex) relu and addition

eg) (1,2,3) + (0,0,1) = (1,2,4)
Calculate elementwisely!

Naïve implementation of an element-wise relu operation:

x should be matrix form.

```
def naive_relu(x):
    assert len(x.shape) == 2, 'x.ndim() must 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
```

chapter02_mathematical-building-blocks.i

File Edit View Insert Runtime Tools Help Cannot save changes

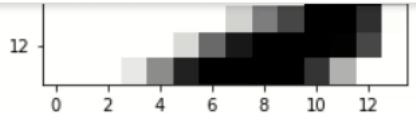
Share

Table of contents

+ Code + Text Copy to Drive

RAM Disk

Editing



The notion of data batches

```
[ ] batch = train_images[:128]
```

```
[ ] batch = train_images[128:256]
```

```
[ ] n = 3
    batch = train_images[128 * n:128 * (n + 1)]
```

Real-world examples of data tensors

Vector data

Timeseries data or sequence data

Image data

0s completed at 1:36 PM

Element-wise operations

Ex) relu and addition

Naïve implementation of addition:

```
def naive_add(x, y):
    assert len(x.shape) == 2
    assert x.shape == y.shape

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

input으로 넣은 x가 그대로 살아남게

and original values changing

Element-wise operations

Element-wise relu:

```
import numpy as np
z = np.maximum(z, 0.)
```

Element-wise add:

```
z = x + y
```

Broadcasting

What happens with addition when the shapes of the two tensors being added differ?

Broadcasting consists of two steps:

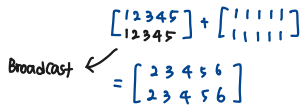
eg) X & Y
dims are diff
X + Y
X.ndim > Y.ndim ⇒ Y Broadcasting

1. **Axes (called broadcast axes) are added to the smaller tensor to match the ndim of the larger tensor.**
2. **The smaller tensor is repeated alongside these new axes to match the full shape of the larger tensor.**

Broadcasting

Example:

```
import numpy as np
x = np.array([1,2,3,4,5])
y = np.array([[1,1,1,1,1], [1,1,1,1,1]])
```




```
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.

Broadcasting

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.

[73] x1

0s

```
array([[1, 0, 3],
       [2, 3, 0]])
```

```
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
```

```
import time

x = np.random.random((20, 100))
y = np.random.random((20, 100))

t0 = time.time()
for _ in range(1000):
    z = x + y
    z = np.maximum(z, 0.)
print("Took: {0:.2f} s".format(time.time() - t0))
```

```
t0 = time.time()
for _ in range(1000):
```

Tensor dot

Dot product in math

$(a,b,c) \cdot (d,e,f) = ad + be + cf$

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

```
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.

$x_1y_1 + x_2y_2 + x_3y_3$

Matrix vector dot

```
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
```

(W x
axb bx1

x is a NumPy matrix. (W)

y is a NumPy vector. (x)

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.

Matrix vector dot

```
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.

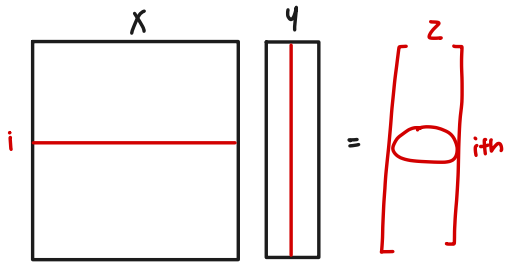
y is a NumPy vector.

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.

Could also use the previous codes

```
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
```



Dot product for 2D tensors

```
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!

Iterates over the rows of x . . .

. . . and over the columns of y.

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

The gears of neural networks: Tensor operations

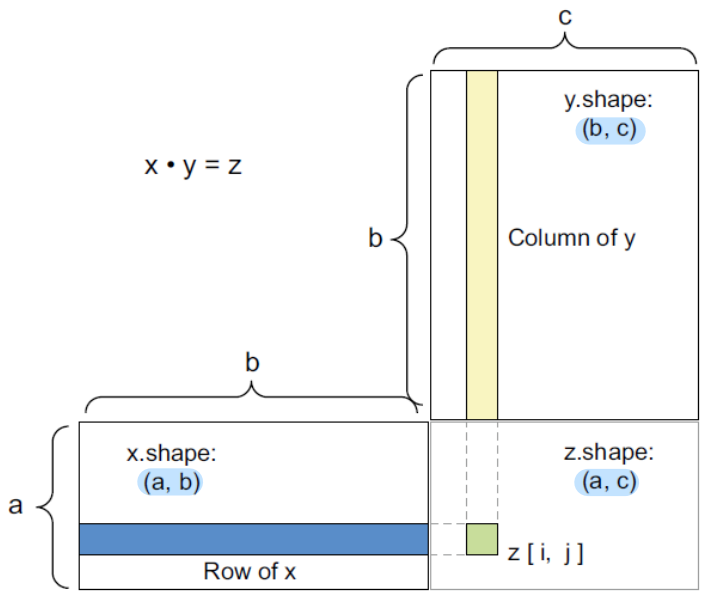


Figure 2.5 Matrix dot-product box diagram

Tensor dot

More generally, you can take the dot product between higher-dimensional tensors, following the same rules for shape compatibility as outlined earlier for the 2D case:

$a \times b \cdot b \times c \Rightarrow a \times c$

4D 1D

(a, b, c, d) . (d,) -> (a, b, c)

(a, b, c, d) . (d, e) -> (a, b, c, e)

Tensor reshaping

we preprocessed the digits data before feeding it into our network:

```
train_images = train_images.reshape((60000, 28 * 28))
```

Try it:

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

x = x.reshape((6, 1))

x = x.reshape((2, 3))
```

```
 1.87539025, 1.41756671, 1.11338379, 0.96418242, 1.21331544],
[1.31749392, 0.94878296, 1.91253185, 0.38539614, 0.40926378,
 1.57273443, 1.31628261, 1.66346262, 1.51518779, 0.81068861]])
```

```
import numpy as np
x = np.random.random((64, 3, 32, 10))
y = np.random.random((32, 10))
z = np.maximum(x, y)
```

Tensor product

```
[ ]  x = np.random.random((32,))
     y = np.random.random((32,))
     z = np.dot(x, y)
```

```
[ ]  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
```

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

Geometric interpretation of tensor operations

Vector as an arrow

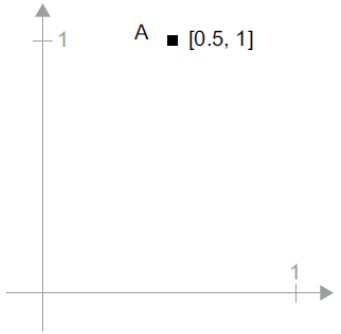


Figure 2.6 A point in a 2D space

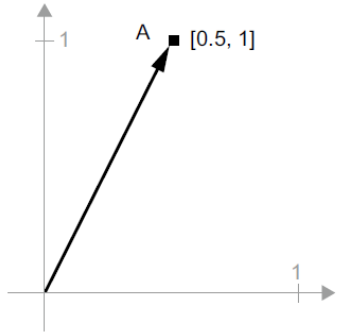


Figure 2.7 A point in a 2D space pictured as an arrow

Sum of two vectors

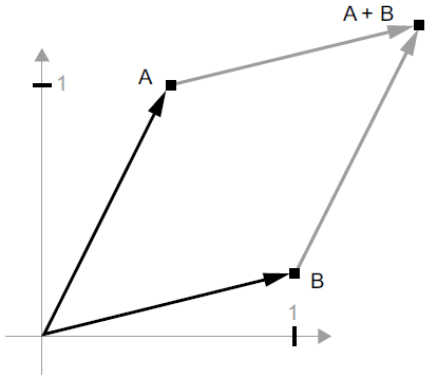


Figure 2.8 Geometric interpretation of the sum of two vectors

Translation as vector addition

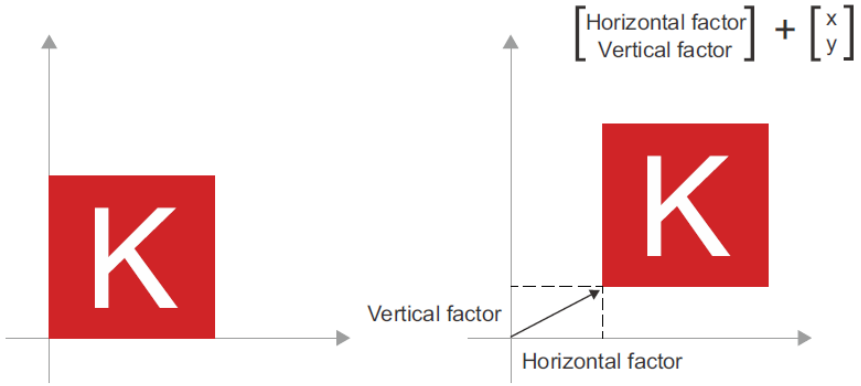


Figure 2.9 2D translation as a vector addition

2D rotation

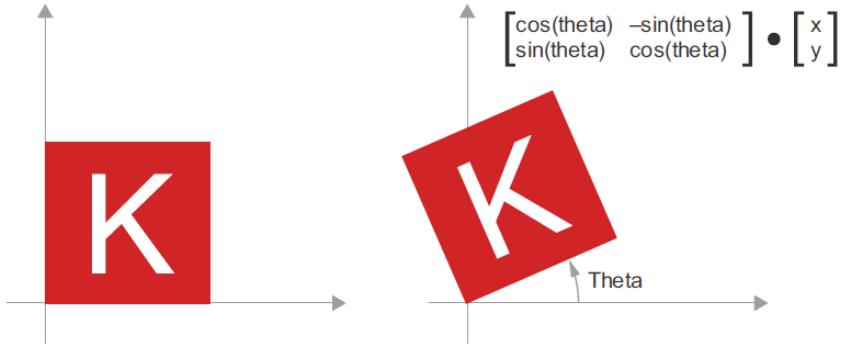


Figure 2.10 2D rotation (counterclockwise) as a dot product

Geometric interpretation of tensor operations

2D scaling

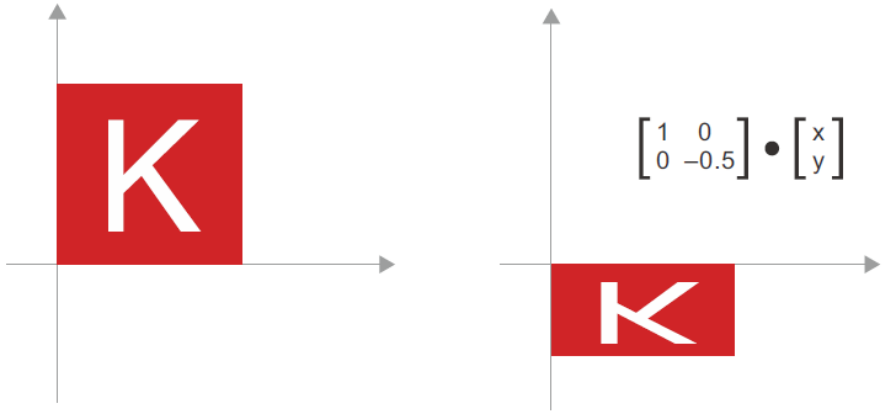


Figure 2.11 2D scaling as a dot product

Affine transform

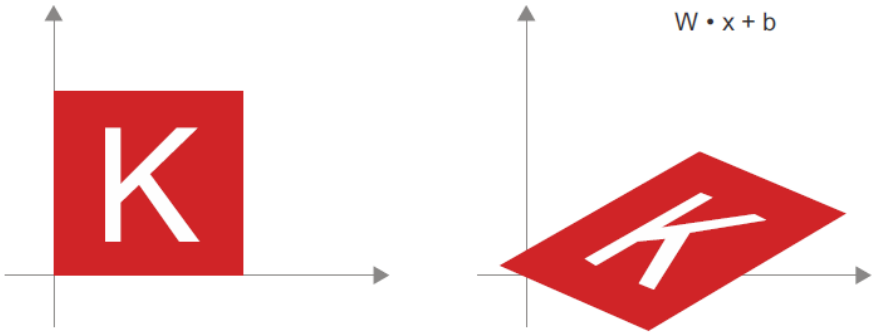


Figure 2.12 Affine transform in the plane

Affine transform followed by relu

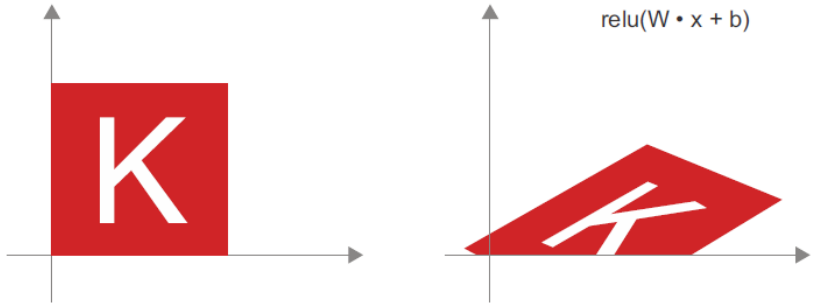


Figure 2.13 Affine transform followed by `relu` activation

Geometric interpretation of deep learning

Neural networks consist entirely of chains of tensor operations

These tensor operations are just simple geometric transformations of the input data

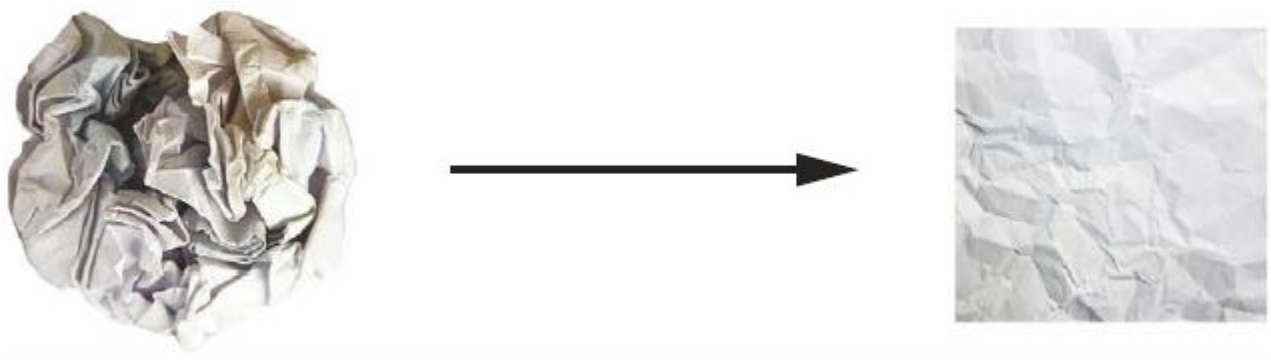


Figure 2.14 Uncrumpling a complicated manifold of data

Thank you! ☺