機器學習研究應用
Study for Machine Learning and Its Applications

The Basics of Neural Networks

孫士韋
Shih-Wei Sun
swsun@newmedia.tnua.edu.tw

Outline

- Data representations for neural networks
- The engine of neural networks
- Looking back at our first example

Data Representations for Neural Networks

Data Representations for Neural Networks

- Tensors: container for numbers
  - Google's TensorFlow named after
- Scalars (0D tensors)
  - Only one number
  - Float32 or float 64
  - 0 axes (ndim: 0)
  - Number of axes: rank

Code:

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

Result:

```
12
0
```

Vectors (1D Tensor)

- Vector: an array of numbers
- Exactly one axis

Code:

1D tensor: 1 bracket

```
x = np.array([12, 3, 6, 14, 7])
print(x)
print(x.ndim)
```

- The vector
  - Has 5 entries
    - 5-dimensional vector
  - Only one axis

1D tensor (axis)

Result:

[12 3 6 14 7]

5 entries

```
[12  3  6 14  7]
1
```

(12, 3, 6, 14, 7)

- Dimension: number of entries, a specific axis

Matrices (2D Tensor)

- Matrix: an array of vectors
- Two axes

Code:

2D tensor: 2 brackets

The same part of code

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

Result:

```
[[ 5 78  2 34  0]
 [ 6 79  3 35  1]
 [ 7 80  4 36  2]]
2
```

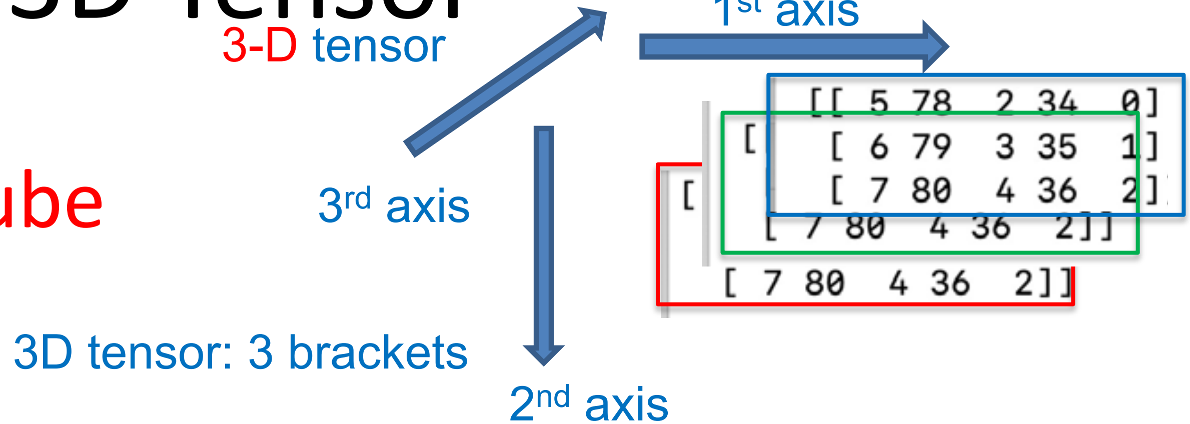
1st axis

$$\begin{bmatrix} 5 & 78 & 2 & 34 & 0 \\ 6 & 79 & 3 & 35 & 1 \\ 7 & 80 & 4 & 36 & 2 \end{bmatrix}$$

2nd axis

2-D tensor

3D Tensor



- Many arrays: cube
- Three axes

Code:

```python
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]]])
print(x)
print(x.ndim)
```

Result:

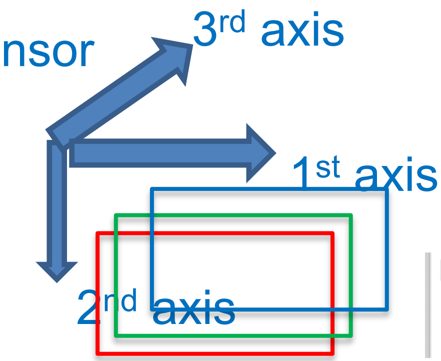
```
[[[ 5 78  2 34  0]
  [ 6 79  3 35  1]
  [ 7 80  4 36  2]]

 [[ 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]]]
3
```

Key Attributes

- A tensor is defined by:
  - Number of axes (rank)
    - ndim
  - Shape
    - How many dimensions the tensor has
      - Along each axis
  - Data type
    - dtype
    - float32, float 64, uint8,…




```
[[[ 5 78  2 34  0]
[[[ 5 78  2 34  0]
[[[ 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]]
```

3x5

Shape: (3,5)

```
[[[ 5 78  2 34  0]
[[[ 5 78  2 34  0]
[[[ 5 78  2 34  0]
  [ 6 79  3 35  1]
  [ 7 80  4 36  2]]
```

3x3x5

Shape: (3,3,5)

3 slices of 3x5 arrays

Result:

```
print(x.shape)
```

```
(3, 3, 5)
```

Try to add this line: end of the codes

Like a toast: different slices

Sesame, peanuts, raisin

An Example of Key Attributes

- Load the data from MNIST
- Display the
  - number of axes: the tensor of train_image, ndim
  - shape of the tensor
  - datatype

Result:

```
Using TensorFlow backend.
3
(60000, 28, 28)
uint8
```

3D tensor

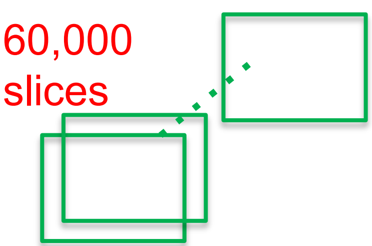
60,000 matrices of 28x28 integers (8-bit)

Code:

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

print(train_images.ndim)
print(train_images.shape)
print(train_images.dtype)
```

60,000 slices



28x28

Each matrix:
grayscale image, 0~255

Displaying the fourth digit

- Display the 4th digit in this 3D tensor
- Using the library Matplotlib
  - Standard scientific Python suite

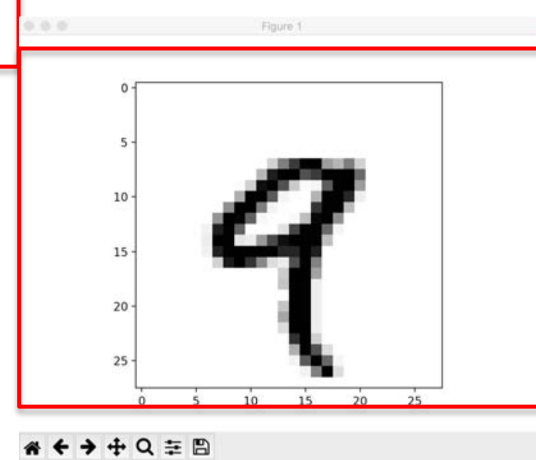
60,000 matrices of 28x28 integers (8-bit)

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

Code:

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

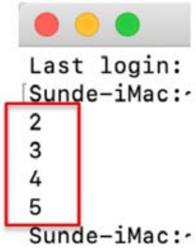
Result:

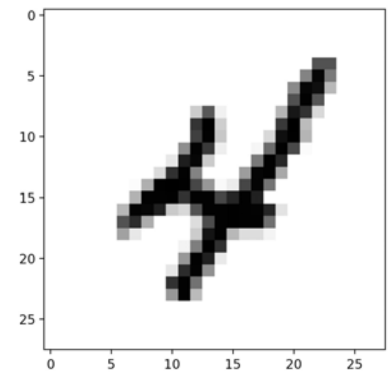


Practice 2

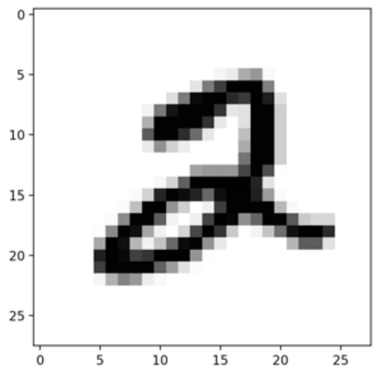
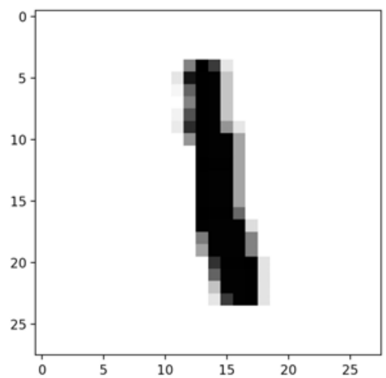
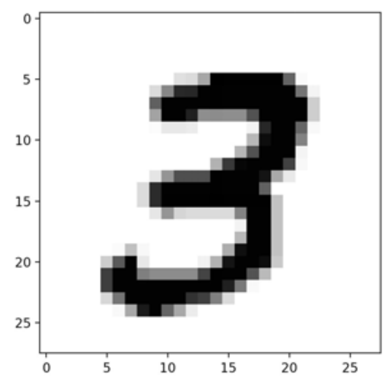
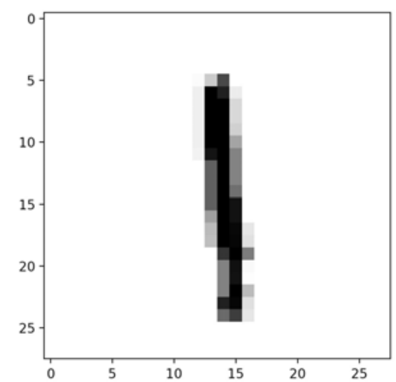
- Based on the above code
- Use a for in range loop
  - Hint:

```
for x in range(2, 6):
    print(x)
```





- Display the 5th to the 10th digits
  - of the MNIST dataset

The gears of neural networks

Manipulating Tensors

- Select a specific digit
  - Alongside the first axis: train_images[i]
- Select specific elements in a tensor
  - Tensor slicing
- Select digits #10 to #100 (#100 not included)

Code:

```
my_slice = train_images[10:100]
print(my_slice.shape)
```

Result:

```
Using TensorFlow backend.
(90, 28, 28)
```

90 slices of 28x28 matrices

Examples of Tensors Slicing

- Equivalent to the previous example

Code:

```
my_slice = train_images[10:100]
print(my_slice.shape)

my_slice = train_images[10:100, :, :]
print(my_slice.shape)

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

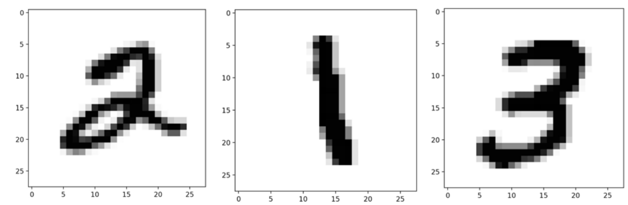
Colon (:) is equivalent to selecting the entire axis

Result:

```
(90, 28, 28)
(90, 28, 28)
(90, 28, 28)
```

The notation of data batches

- The first axis (axis 0, index from 0)
  - Sample axis => batch axis
  - in MNIST, sample: images of digits
- Deep learning models
  - Don't process an entire dataset at once
- Break the data into small batches (a small group of data)
- Batch example: batch size of 128



Keras, deep-learning libraries

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

Code:

```
batch = train_images[:128]
print(batch.shape)
batch = train_images[128:256]
print(batch.shape)
```

Result:

First batch

```
(128, 28, 28)
(128, 28, 28)
```

next batch

More batches…

Full code

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

print(train_images.ndim)
print(train_images.shape)
print(train_images.dtype)

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

my_slice = train_images[10:100]
print(my_slice.shape)

my_slice = train_images[10:100, :, :]
print(my_slice.shape)

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

batch = train_images[:128]
print(batch.shape)
batch = train_images[128:256]
print(batch.shape)
```

Real-world examples of data tensors

- Vector data
  - 2D tensor (samples, features)
- Timeseries or sequence data
  - 3D tensor (samples, timesteps, feature)
- Images
  - 4D tensors (samples, channels, height, width)
- Video
  - 5D tensors (samples, frames, channels, height, width)

Figure 2.3 A rank-3 timeseries data tensor

3D tensor

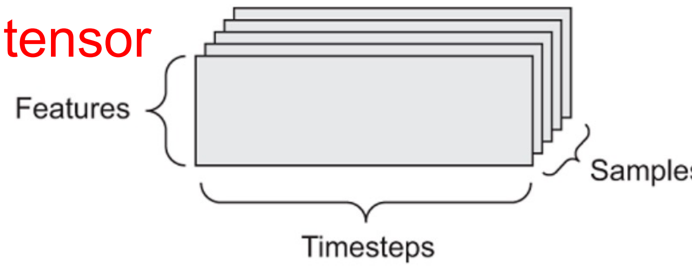
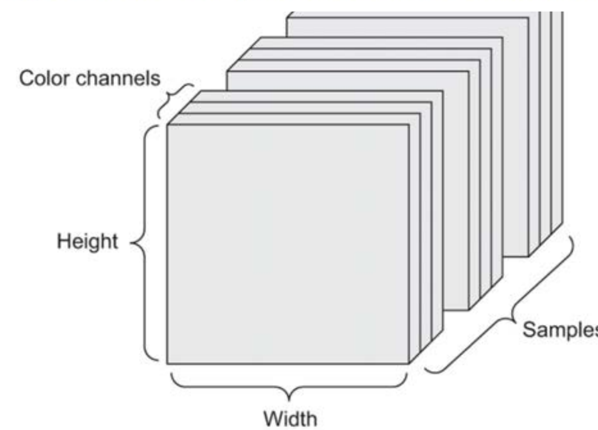


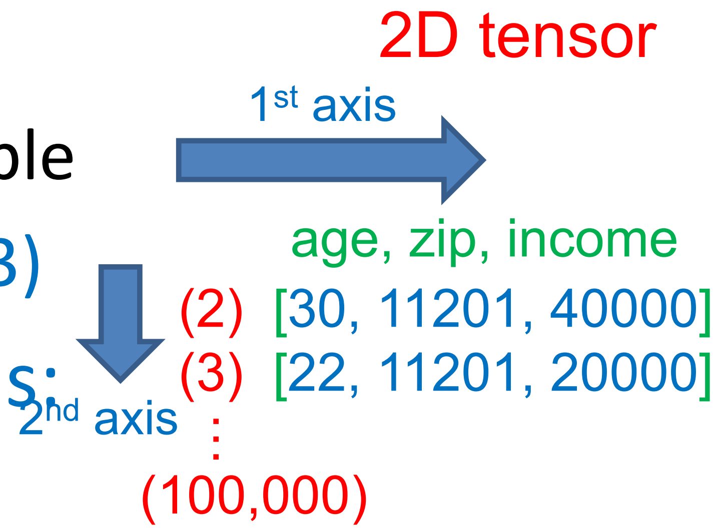
Figure 2.4 A rank-4 image data tensor (channels-first convention)

4D tensor



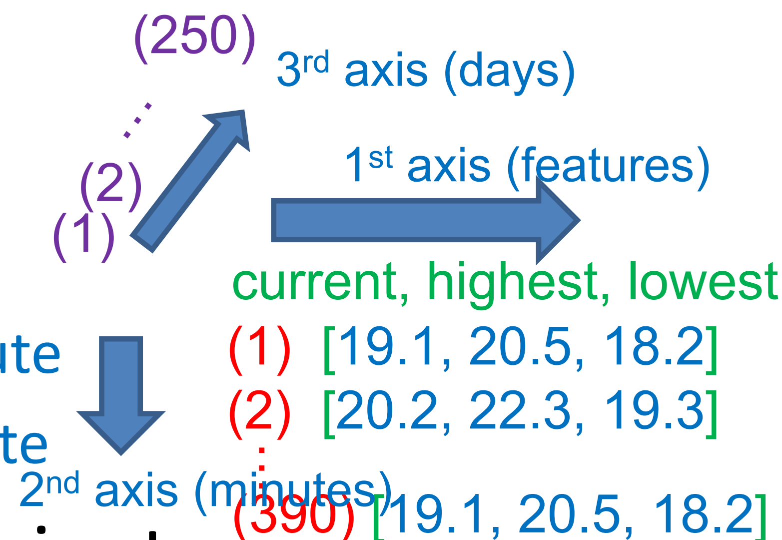
Examples of a 2D tensor

- Actuarial dataset of people
  - Person’s age
  - ZIP code
  - Income
- Each person characterized as a vector of
  - 3 values (feature)
  - Entire dataset of 100,000 people
  - 2D tensor of shape: (100000, 3)
- Example of #2 and #3 persons:



Timeseries data or sequence data, 3D Tensor (audio, sensor data)

- Stock price dataset
  - For every minute
    - Current price of the stock
    - Highest price in the past minute
    - Lowest price in the past minute
- Entire day of trading: 390 minutes
  - 2D tensor of shape (390, 3)
- 250 trading days:
  - (250, 390, 3)
  - Each sample: one day's worth of data



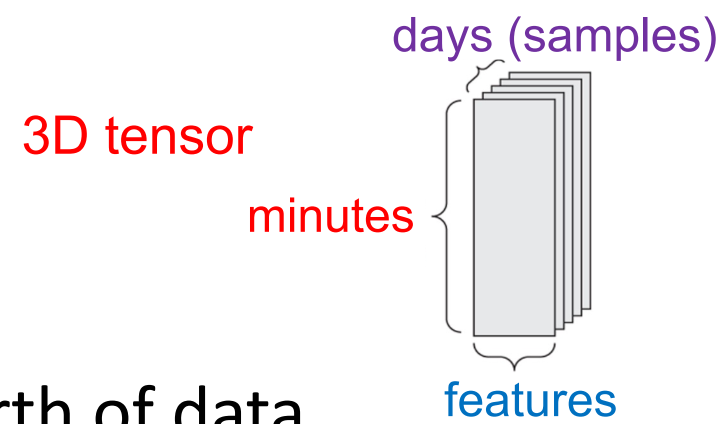
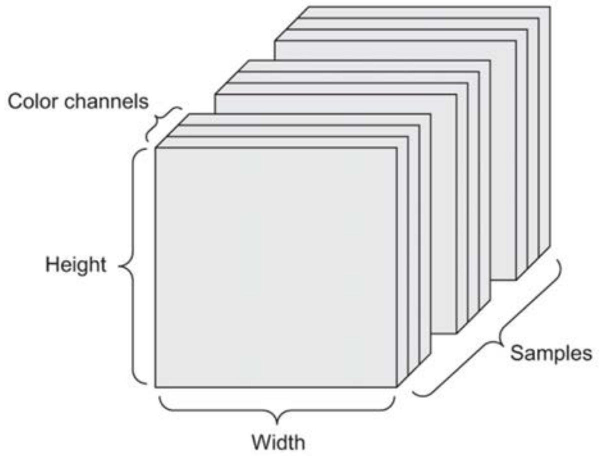


Image data



- Images
  - Height
  - Width
  - Color depth
    - Gray scale image have only single color channel
- Example of an 256x256 color image
  - with a batch size of 128
  - (128, 256, 256, 3): (samples, height, width, color_depth)
    - Channels-last in Tensorflow
    - Channel-first in Theano: (samples, color_depth , height, width)
    - Both conventions supported: Keras

Tensor reshaping

- Preprocessing the digits data
  - Before feeding it into the neural network

```
# look at the training dat
print(train_images.shape)
```

```
Using TensorFlow backend.
(60000, 28, 28)
```

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

- Reshaping: rearranging its rows and columns
  - To match a target shape: (28, 28) 2D to (28*28) 1D
- Same total number
  - As the initial tensor

```
print(train_images.shape)
```

```
(60000, 784)
```

Example of Tensor Reshaping

- A simple example

Code:

```
import numpy as np

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

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

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

Result:

```
(3, 2)
[[0. 1.]
 [2. 3.]
 [4. 5.]]
(6, 1)
[[0.]
 [1.]
 [2.]
 [3.]
 [4.]
 [5.]]
(2, 3)
[[0. 1. 2.]
 [3. 4. 5.]]
```

Practice 3

- Run the above code
  - For tensor reshaping

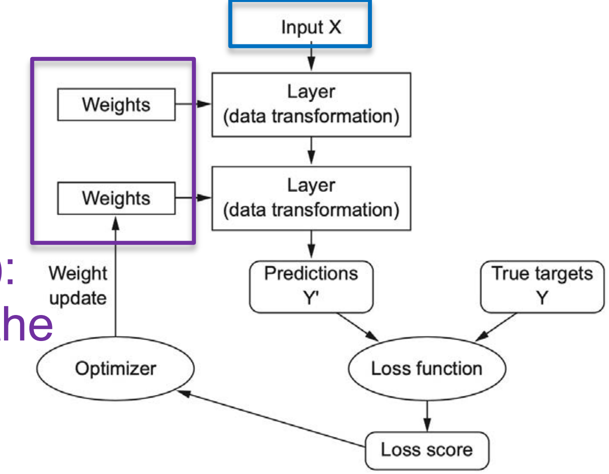
Result:

```
(3, 2)
[[0. 1.]
 [2. 3.]
 [4. 5.]]
(6, 1)
[[0.]
 [1.]
 [2.]
 [3.]
 [4.]
 [5.]]
(2, 3)
[[0. 1. 2.]
 [3. 4. 5.]]
```

The engine of neural networks

Tensor Operations

neural network

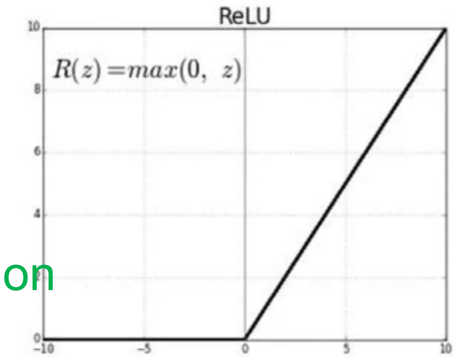


W (weights): learned by the network

- In our Initial example:
  - keras.layers.Dense(512, activation='relu')
  - The layer: function
    - Input: 2D tensor, output: 2D tensor
  - The function is like this:

relu: activation function

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



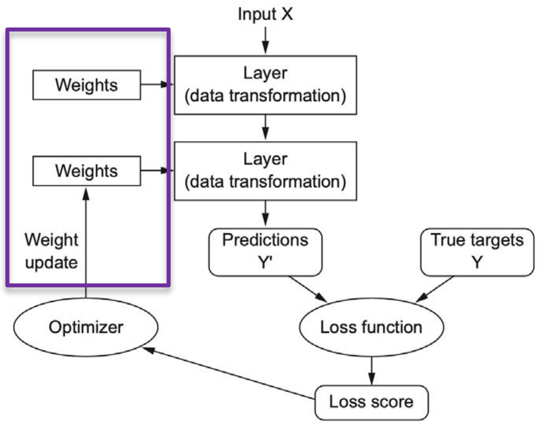
- Three tensor operations

W: weights / (kernel)
b: trainable parameters / (bias)

  - a dot product (dot) between the input, W (weight)
  - addition (+): resulting 2D tensor and a vector b (bias)
  - relu operation: relu(x) is max(x, 0)

Weight matrix W

- function: output = relu(dot(W, input) + b)
- Weight matrix W :
  - filled with small random values (random init.)
  - No reason to expect the function
    - Useful representations, just the starting point!
- Gradual adjustment: training in machine learning!
- Gradual adjust the weights W
  - Based on a feedback signal



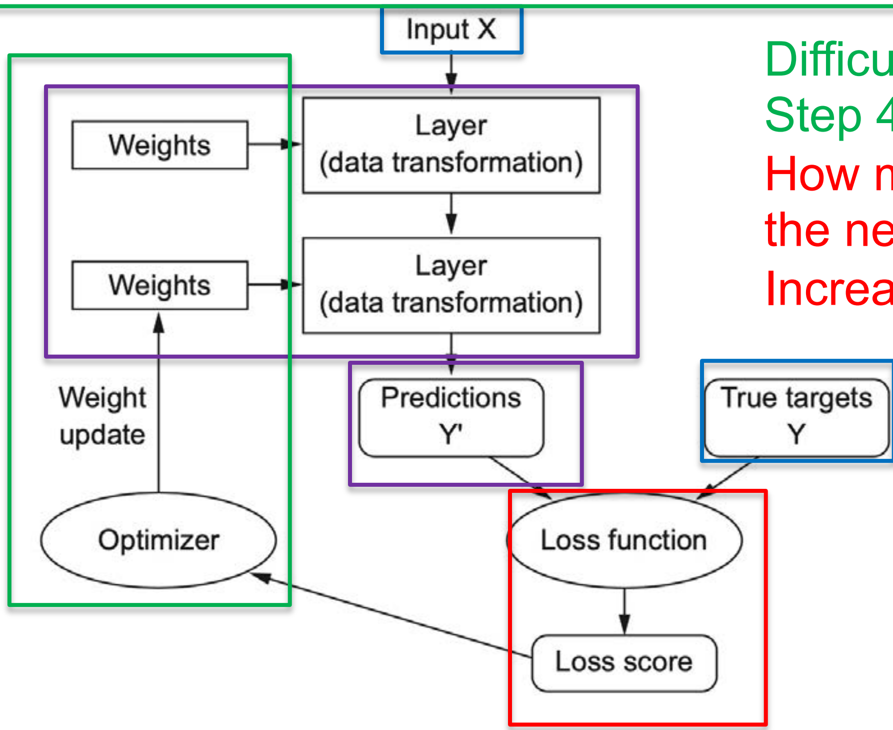
Training loop

1 Draw a batch of training samples x and corresponding targets y.

2 Run the network on x (a step called the *forward pass*) to obtain predictions y_pred.

3 Compute the loss of the network on the batch, a measure of the mismatch between y_pred and y.

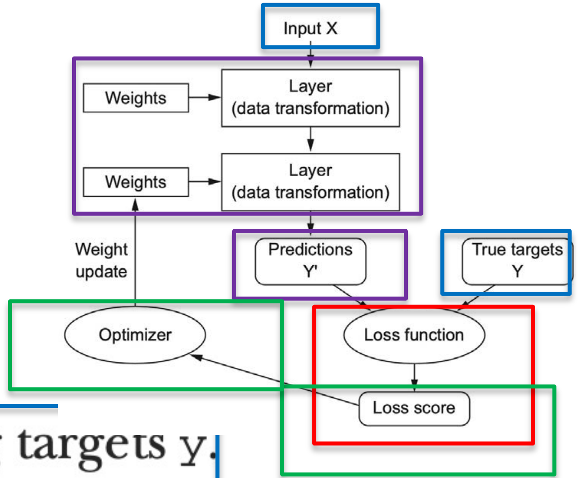
4 Update all weights of the network in a way that slightly reduces the loss on this batch.



Difficult part:
Step 4
How much to update the network's weight?
Increase or decrease?

- **Finally:**
  - End up a network
    - With low loss
  - Low mismatch
    - y_pred and y
  - Network learned

Optimization



- Stochastic gradient decent (SGD)

1 Draw a batch of training samples x and corresponding targets y.

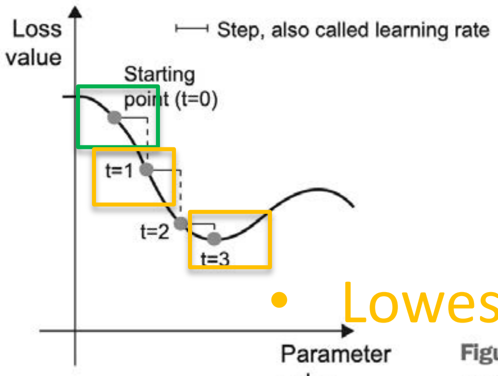
2 Run the network on x to obtain predictions y_pred.

3 Compute the loss of the network on the batch, a measure of the mismatch between y_pred and y.

4 Compute the gradient of the loss with regard to the network's parameters (a *backward pass*).

5 Move the parameters a little in the opposite direction from the gradient—for example `W -= step * gradient`—thus reducing the loss on the batch a bit.

- Move the parameters
  - until Gradient = 0
  - (Derivative = 0)

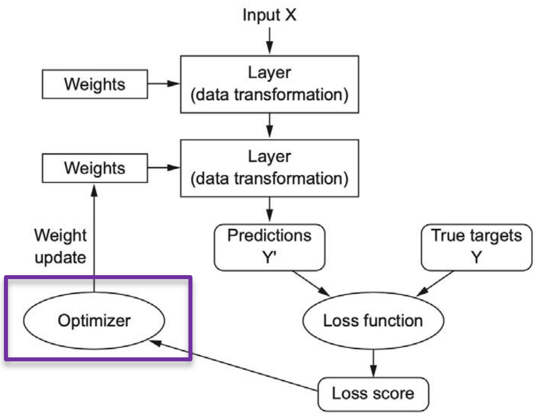


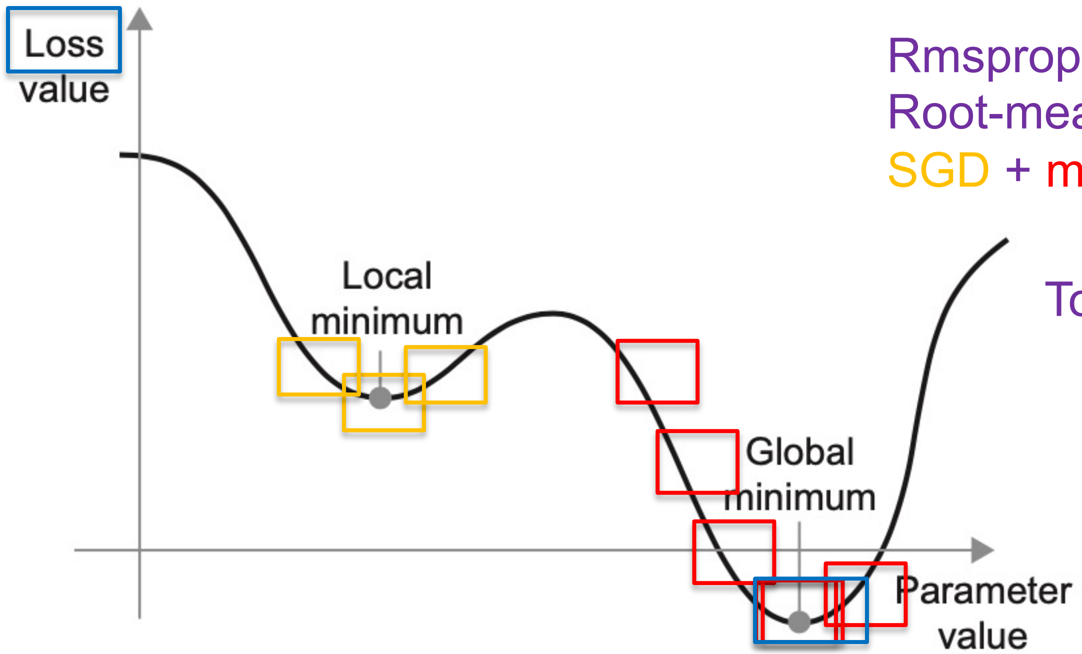
- Lowest value: loss

Figu
curv

Local minimum vs. a global minimum

- Stuck in a local minimum
- Need a momentum
  - Moving a ball by a velocity





Rmsprop (optimizer):
Root-mean-square propagation algorithm
SGD + momentum

To: Get the smallest LOSS

Figure 2.13 A local minimum and a global minimum

Looking back at our first example

Looking back at our first example

- Input data

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

- Network

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

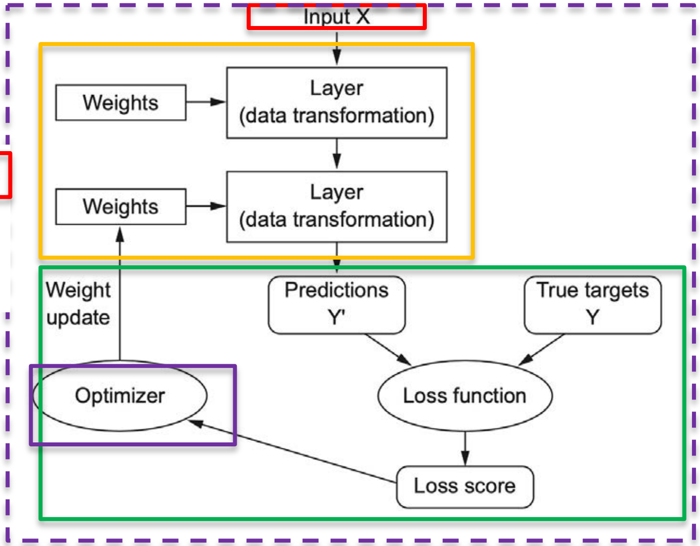
- Network-compilation

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

rmsprop: optimizer

- Training

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



Loss and optimizer: you need to define
- Before feeding data into a network

loss: quantity attempt to minimize

optimizer: gradient of the loss,
Uses to update parameters

Epoch: a full training loop
Epochs: number of training loops

How many times for staying in a training loop?

Textbook Reading

- *Deep Learning with Python*
  - Ch 2, the mathematical building blocks of neural networks
    - p.31 - p. 62