

# Lecture 1. Mathematical Building Blocks of Neural Networks

## 1.1 MNIST data



- 딥러닝의 "Hello world"
- 흑백 손글씨 숫자 이미지( $28 \times 28$  픽셀)를 0-9까지 10개의 범주로 구분해놓은 데이터셋
- 6만개 training set, 1 만개 test set으로 구성

```
In [1]: import keras
        from keras.datasets import mnist

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

        Using TensorFlow backend.

        Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz (https://s3.amazonaws.com/img-datasets/mnist.npz)
        11493376/11490434 [=====] - 2s 0us/step
```

### Training set

```
In [2]: train_images.shape
```

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

```
In [3]: len(train_labels)
```

```
60000
```

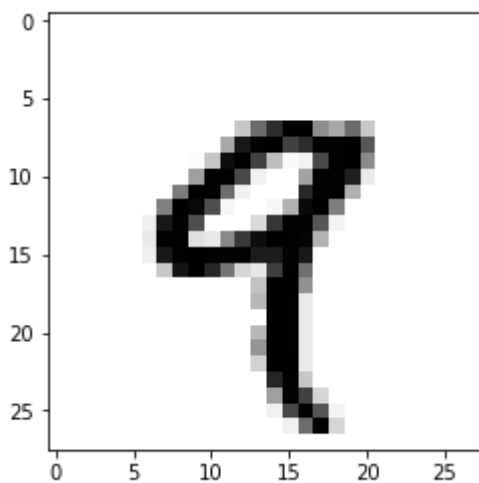
```
In [4]: train_labels
```

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

```
In [6]: digit
```

[illegible]

```
In [5]: digit = train_images[4]
import matplotlib.pyplot as plt
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```



Test set

```
In [7]: test_images.shape
```

(10000, 28, 28)

```
In [8]: len(test_labels)
```

10000

```
In [9]: test_labels
```

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

---

## Workflow

### (1) Network architecture

```
In [10]: from keras import models
         from keras import layers

         network = models.Sequential()
         network.add(layers.Dense(512, activation='relu', input_shape=(28 *
         network.add(layers.Dense(10, activation='softmax'))
```

```
In [11]: network.summary()
```

```
-----
Layer (type)                 Output Shape          Param #
-----
dense_1 (Dense)              (None, 512)           401920
-----
dense_2 (Dense)              (None, 10)            5130
-----
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
-----
```

- Fully connected layer 두 개가 연결된 모형을 구축
- 마지막 layer는 10개의 확률 점수가 들어있는 배열을 반환하는 softmax activation을 사용하여 0-9의 숫자 클래스에 들어갈 확률을 예측

### (2) Compilation

```
In [12]: network.compile(optimizer='rmsprop',
                        loss='categorical_crossentropy',
                        metrics=['accuracy'])
```

- loss function : 모형의 성능을 측정하는 방법으로 네트워크가 옳은 방향으로 학습될 수 있도록 만들어줌
- optimizer : input data를 기반으로 loss function을 최소화 시키도록 weight를 업데이트하는 메커니즘
- metrics : training/test 과정을 모니터링할 지표

### (3) Preparing the image data and the labels

```
In [13]: 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
```

```
In [14]: train_images.shape
```

```
(60000, 784)
```

- 데이터를 네트워크에 맞는 크기로 바꿈
- 모든 값을 0과 1사이의 값으로 변환

```
In [15]: train_labels
```

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

```
In [16]: from keras.utils import to_categorical
```

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
In [18]: train_labels.shape
```

```
(60000, 10)
```

- 레이블을 범주형으로 인코딩(one-hot encoding)

#### (4) Training the model

```
In [28]: network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
60000/60000 [=====] - 2s 28us/step - loss: 0.2583 - acc: 0.
9253
Epoch 2/5
60000/60000 [=====] - 2s 27us/step - loss: 0.1030 - acc: 0.
9691
Epoch 3/5
60000/60000 [=====] - 2s 28us/step - loss: 0.0681 - acc: 0.
9794
Epoch 4/5
60000/60000 [=====] - 2s 28us/step - loss: 0.0494 - acc: 0.
9853
Epoch 5/5
60000/60000 [=====] - 2s 28us/step - loss: 0.0378 - acc: 0.
9885
```

```
<keras.callbacks.History at 0x7f654806be48>
```

- 훈련하는 동안 loss와 accuracy가 출력됨
- 60000개의 데이터셋을 총 5번 사용하는 만큼(epoch=5) 훈련을 지속

### (5) Prediction for test set

```
In [29]: test_loss, test_acc = network.evaluate(test_images, test_labels)
```

```
10000/10000 [=====] - 0s 47us/step
```

```
In [30]: print('test_acc:', test_acc)
```

```
test_acc: 0.9795
```

- Test set에 대한 accuracy가 0.9799로 training set에 대한 accuracy보다 약간 낮음 (overfitting)

## 1.2 Data representations for neural networks

- Tensor: 데이터를 위한 컨테이너
- 행렬의 일반화된 형태
- 차원을 axis라고 부름

### Scalar (0D tensor)

- 하나의 숫자만 담고 있는 tensor

```
In [19]: import numpy as np
x=np.array(12)
x
```

```
array(12)
```

```
In [20]: x.ndim
```

```
0
```

### Vectors (1D tensor)

- 하나의 axis를 가지는 벡터

```
In [21]: x=np.array([12,3,6,14,7])
x.ndim
```

```
1
```

- x 는 5-dimensional vector (5D tensor가 아님)

### Matrices (2D tensors)

- 2개의 axis가 있음: row, column

```
In [24]: x = np.array([[5, 78, 2, 34, 0],
                        [6, 79, 3, 35, 1],
                        [7, 80, 4, 36, 2]])

x.ndim

2
```

```
In [26]: x.shape
```

(3, 5)

- 1st axis: row
- 2nd axis: column
- 1st row = [5, 78, 2, 34, 0]
- 1st column = [5, 6, 7]

### 3D tensors and higher-dimensional tensors

- 행렬들을 모아 하나의 array로 만들면 직육면체의 3D tensor가 만들어짐

```
In [27]: 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
```

```
In [28]: x.shape
```

(3, 3, 5)

### Key attributes

- rank(axis의 개수)
  - 3D tensor의 rank는 3
  - 2D tensor의 rank는 2
  - `ndim` 속성에 저장
- shape
  - Tensor의 각 axis를 따라 얼마나 많은 차원이 있는지를 나타낸 tuple
  - 위의 2D tensor의 shape은 (3,5)
  - 위의 3D tensor의 shape은 (3,3,5)
- data type
  - Tensor에 포함된 데이터의 타입
  - float32, uint8, float64 등

- dtype 속성에 저장

```
In [29]: (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
In [30]: train_images.shape
```

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

```
In [31]: train_images.dtype
```

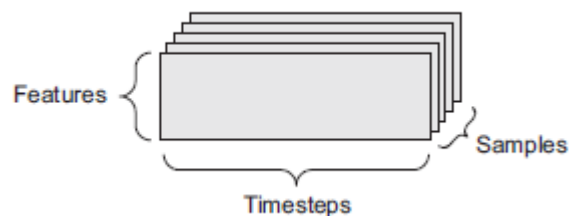
```
dtype('uint8')
```

### TO DO: MNIST 데이터

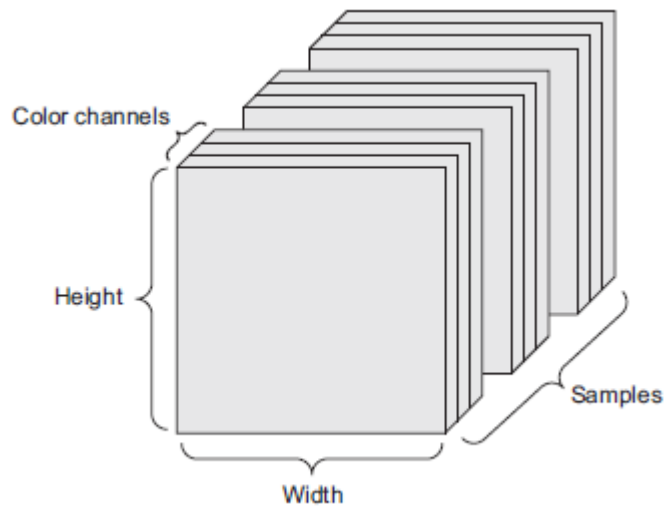
1. train\_images의 rank는?
2. train\_images의 shape은?
3. train\_images의 data type은?

### Examples of tensor data

- Vector data—2D tensors of shape (samples, features)
  - 사람의 나이, 우편번호, 소득으로 구성된 인구통계 데이터. 10만명이 포함된 데이터 셋이라면 (100000,3)
  - 공통 단어 2만 개로 이루어진 텍스트 문서 500개가 있다. 하나의 문서를 각 단어가 등장한 횟수로 표현된 벡터로 나타낸다면 (500,20000)
- Timeseries data or sequence data—3D tensors of shape (samples, timesteps, features)



- 관례적으로 시간은 axis=1인 축으로 표현
  - 주식가격 데이터셋: 1분 마다 3개의 feature(현재 주식가격, 지난 1분 동안 최고 가격, 최저 가격)를 저장. 1분마다 3-dimensional vector 생성. 하루의 거래는 (390,3) 크기의 2D tensor로 저장. 250일치 데이터는 (250,390,3) 크기의 3D tensor로 저장
  - 트윗 데이터셋: 각 트윗은 128개의 알파벳/문자로 구성된 280개의 문자 시퀀스. 하나의 트윗은 (280,128) 크기의 2D tensor. 100만개의 트윗이 포함된 데이터는 (1000000,280,128) 크기의 3D tensor
- Images—4D tensors of shape (samples, height, width, channels)



- 하나의 이미지는 전형적으로 높이, 너비, 컬러채널의 3-dimensional data.  $256 \times 256$  픽셀의 이미지라면 (256, 256, 3) 크기의 3D tensor
- 128개의 이미지가 포함된 batch는 (128, 256, 256, 3) 크기의 4D tensor로 표현
- 만일 흑백 이미지라면 (128, 256, 256, 1)
- Video—5D tensors of shape (samples, frames, height, width, channels)
  - 비디오 데이터는 이미지의 연속
  - 60초 짜리  $144 \times 256$  유튜브 비디오 클립을 초당 4프레임으로 샘플링하면 240 프레임을 생성. 하나의 비디오 클립은 (240, 144, 256, 3)
  - 비디오 클립을 4개 포함한 batch는 (4, 240, 144, 256, 3) 크기의 5D tensor

\_TO DO: \_

영화 리뷰 25000개가 있다. 각 리뷰는 200개의 단어로 이루어져 있다.

- 가장 자주 등장하는 10000개의 단어에 대해 각 리뷰에 해당 단어의 포함 유무를 0/1로 나타낸다. 이 데이터의 shape은 무엇인가?
- 가장 자주 등장하는 10000개에 대해 각 단어를 one-hot vector로 표현하여 하나의 문장을 matrix로 표현한다. 이 데이터의 shape은 무엇인가?

## References

- [Deep Learning with Python, François Chollet, \(https://www.manning.com/books/deep-learning-with-python\)](https://www.manning.com/books/deep-learning-with-python)