Deep Learning:

Keras 프로그래밍

Deep learning 프로그래밍

- 기능 구현
  - Dense network, CNN(Convolutional neural network), RNN(Recurrent neural network) 등 다양한 형태의 신경망을 구현해야 함
  - 순방향 계산, 역방향 확산 등의 절차를 구현해야 함
  - 데이터를 이용한 훈련과 검증 등을 수행
- 실행 하드웨어에 맞추어 코드를 수정
  - 성능 향상을 위해 CPU, GPU(Graphical Processing Unit), TPU(Tensor Processing Unit) 등 실행 환경에 맞도록 코드를 수정

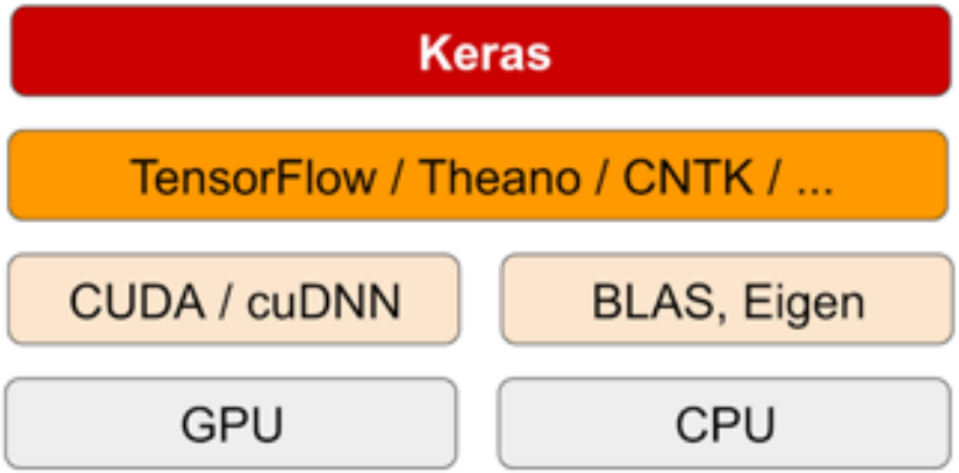
어려움

- 구현을 위한 패키지 활용
  - 정형화된 신경망 프로그래밍을 패키지 형태로 구현해 왔음
  - 자신의 수요에 적합한 패키지를 선택하여 사용

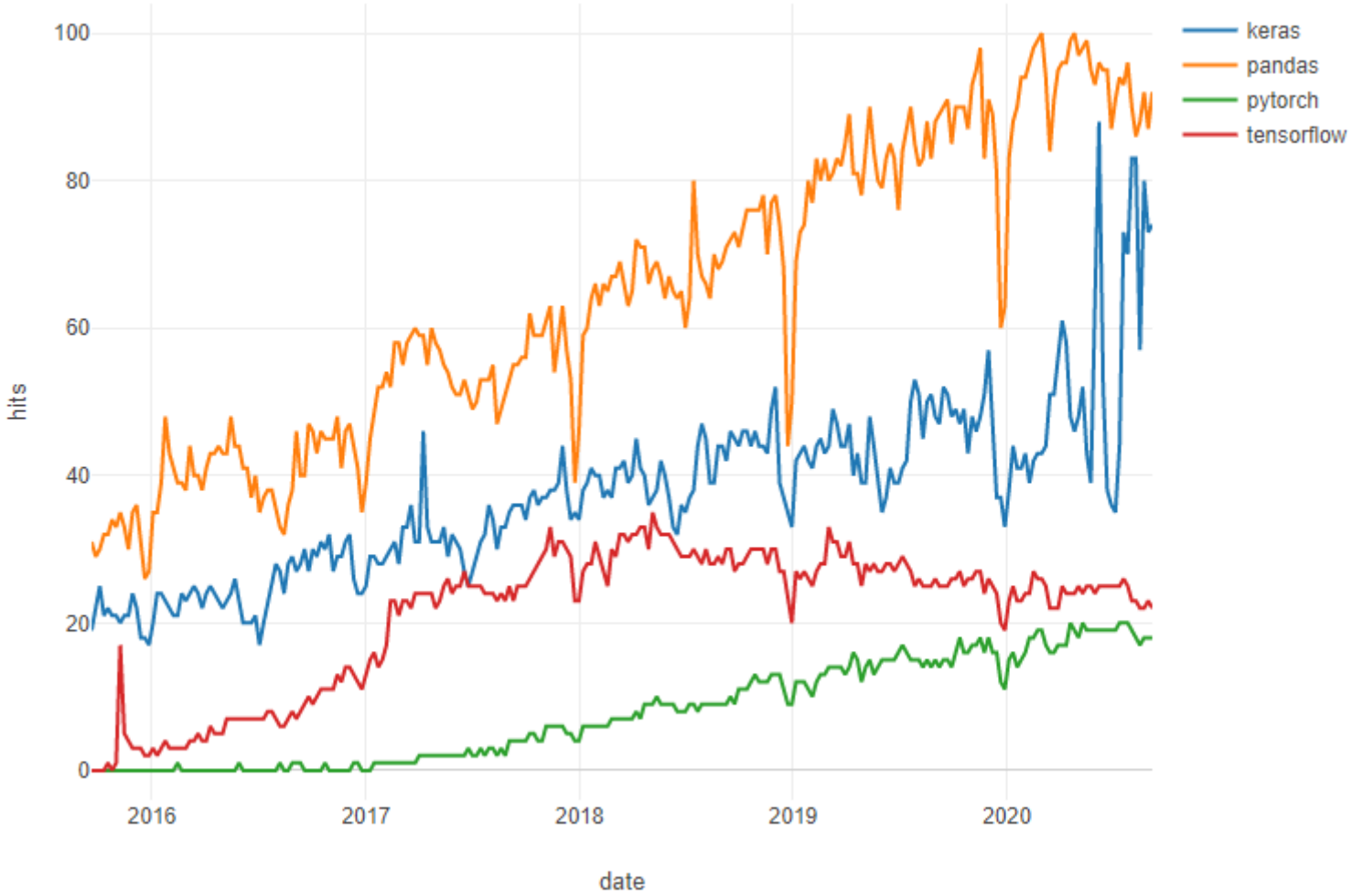
쉬움

TensorFlow와 Keras

- 기계학습 프로그래밍 지원을 위해 Google에서 TensorFlow를 개발
  - 2015년 부터 패키지를 제공하고 있음
  - 다양한 하드웨어를 지원하는 실행환경과 라이브러리를 제공
  - 기능은 우수하지만 사용하기가 다소 어려움
- **쉬움:** 보다 쉬운 프로그램 개발을 지원하는 Keras를 제공하고 있음



Machine Learning 패키지 선호도



- pytorch는 Facebook에서 개발되었고, Tensorflow는 Google에서 개발
- pytorch와 Tensorflow 중 하나를 선택할 수 있음

Deep Learning 패키지 설치: Tensorflow와 keras

- Anaconda prompt 도구를 이용하여 Tensorflow 설치

  > pip install tensorflow

- 설치 확인: Anaconda prompt에서 ipython을 실행

  > ipython

  ...

  In [1]: **import** tensorflow **as** tf

  In [2]: tf.__version__

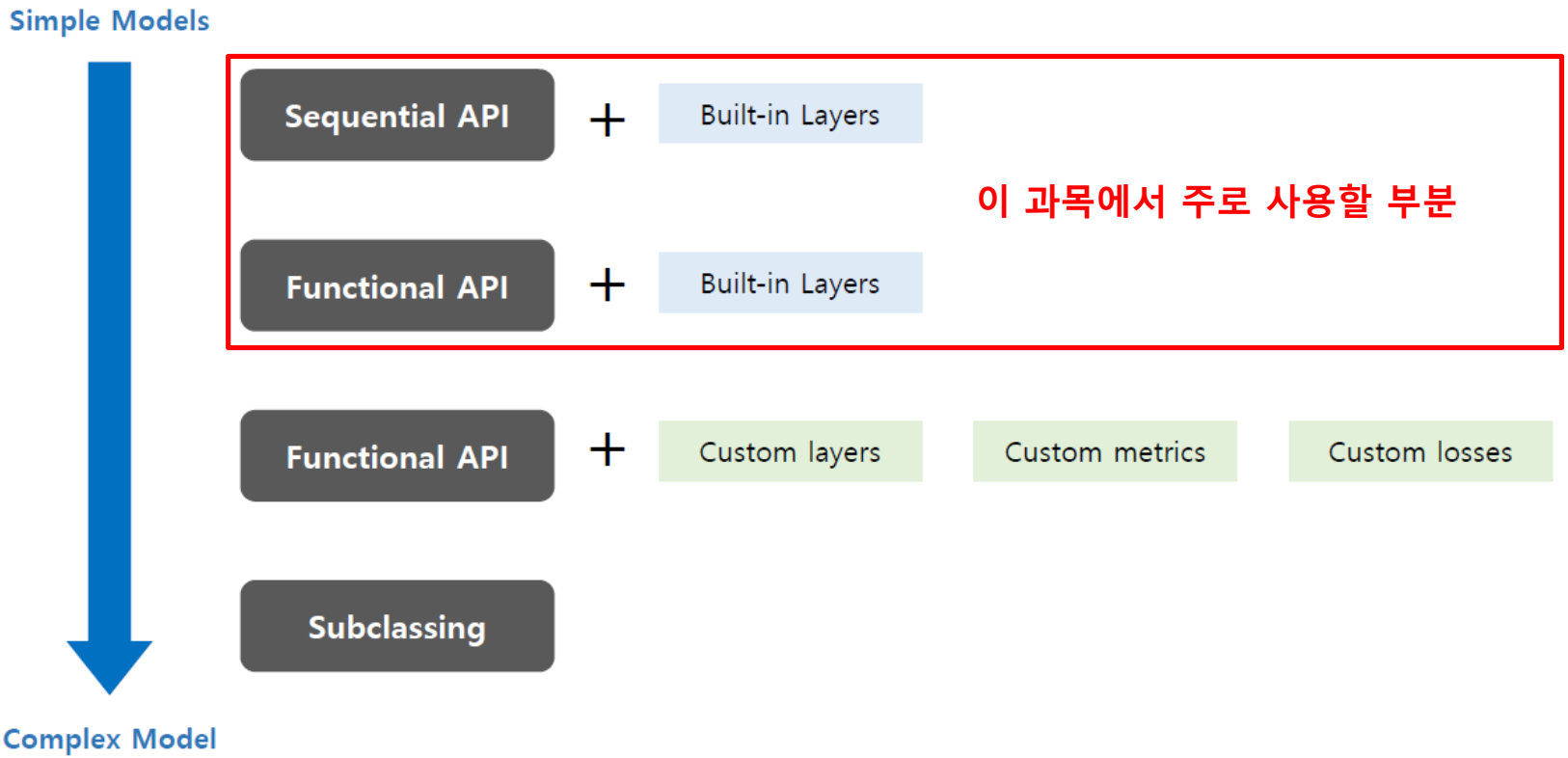
  Out[2]: '2.3.0'

- keras도 마찬가지로 설치

  > pip install keras

Keras 프로그래밍

- 짧은 프로그래밍을 통해 프로그램을 구현할 수 있음
- 단일 입출력 신경망으로 구현되는 시스템은 Sequential API로 구현할 수 있음

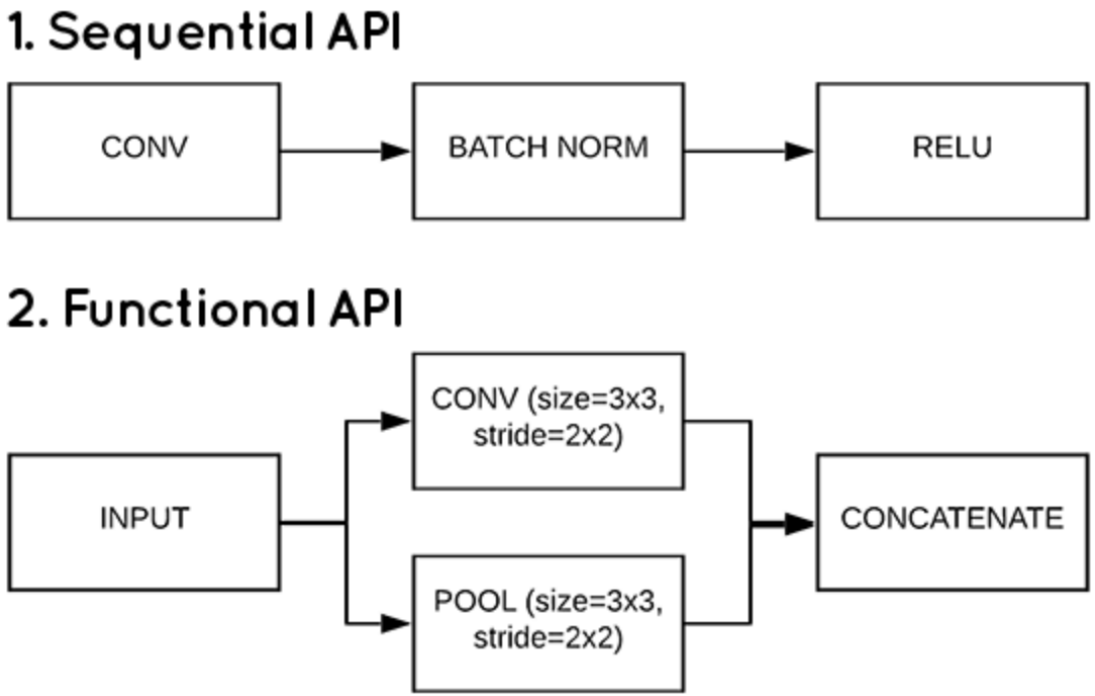


Keras 설명 자료

- 케라스 사이트: https://keras.io
  - 설명 자료와 Code Examples 들이 있음
- 케라스 도서
  - ***Deep Learning with Python***, François Chollet
  - https://www.manning.com/books/deep-learning-with-python-second-edition
  - 번역서: **케라스 창시자에게 배우는 딥러닝**
- 교재에서의 케라스 설명
  - 8.6절의 '케라스 훑어보기 ' 에 개략적인 설명이 있음

Keras 호출 구조

- **Sequential API:** 신경망 layer(Dense, CNN, RNN 등)들이 단일 입-출력으로 이루어진 구조
- **Functional API:** Layer들의 입-출력이 여러 개로 구성될 수 있음



Sequential API 프로그래밍 구조

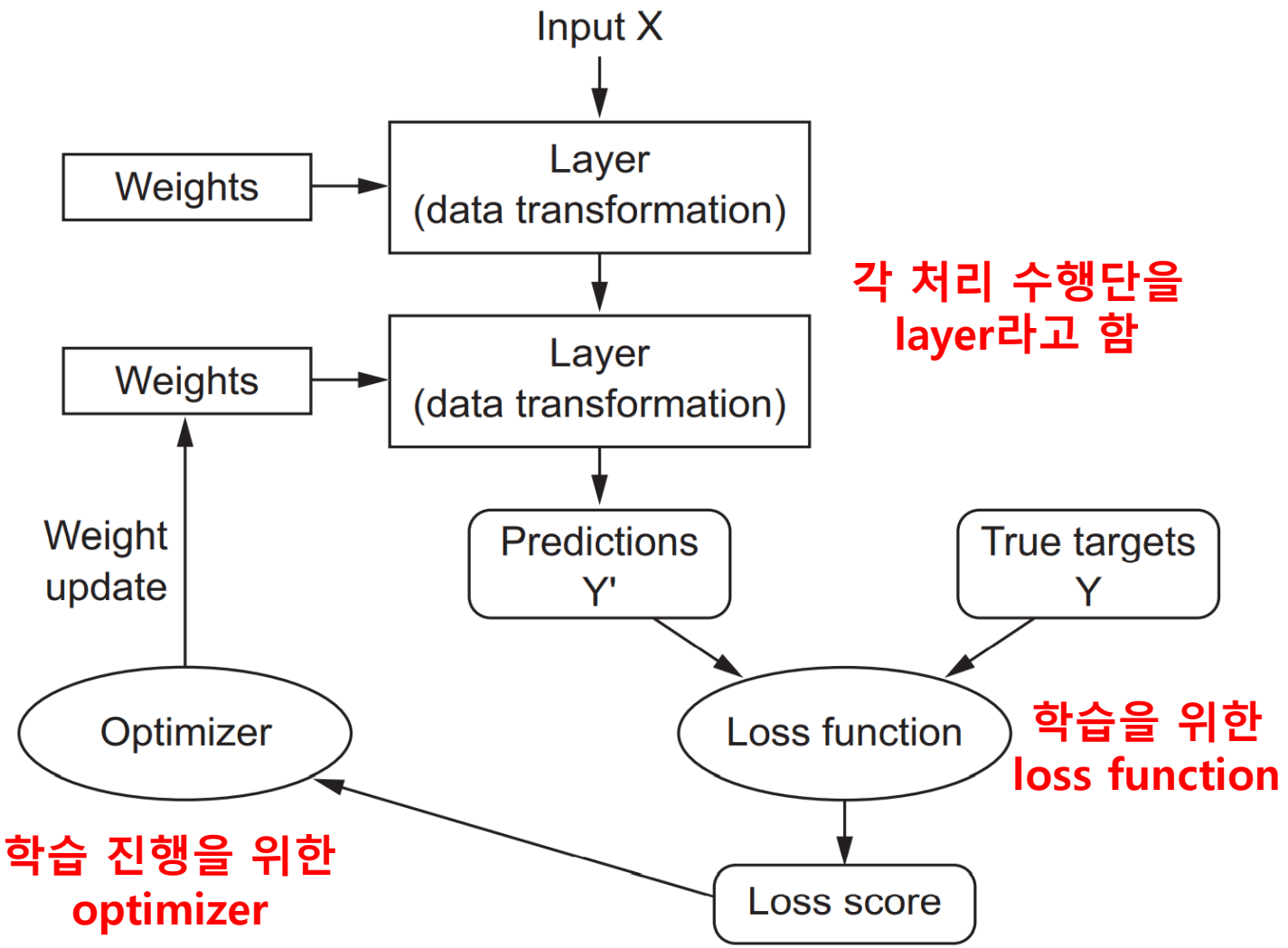
- 신경망의 각 layer를 차례로 지정: 구조(Dense, CNN, RNN 등), 출력수, 활성화 함수 등
- 훈련 설정(compile): 비용 함수에 대한 optimizer, loss 함수 등을 지정
- 훈련 실행(fit): 입력 데이터를 신경망에 입력하여 훈련을 실행
- 성능 평가(evaluate): 테스트 데이터를 이용하여 성능을 평가

```
from keras import models
from keras import layers

model = models.Sequential()  # Sequential API를 사용

model.add(...) # 레이어 추가
model.add(…) # 레이어 추가
model.add(…) # 레이어 추가
```

Sequential API 프로그램 사례

```
from keras import models
from keras import layers

# Sequential API를 사용
network = models.Sequential()

# Layer들을 설정
network.add(layers.Dense(64, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))

# 훈련 설정
network.compile(optimizer='rmsprop',
        loss='categorical_crossentropy',
        metrics=['accuracy'])

# 모델 훈련
network.fit(train_data, train_labels, epochs=10, batch_size=32)

# 모델 평가
network.evaluate(test_data, test_labels)
```

신경망 일반 구조


Input X
Weights
Layer
(data transformation)
Weights
Layer
(data transformation)
Predictions
Y'
True targets
Y
Weight
update
Optimizer
Loss function
Loss score
각 처리 수행단을
layer라고 함
학습을 위한
loss function
학습 진행을 위한
optimizer

Layer 추가 방법

- 만들어진 network에 layer를 추가할 때는 add 함수를 사용
- layer의 형태를 지정함: Dense, SimpleRNN, Conv2D 등
- 활성화 함수를 지정함
- 출력 단자의 숫자를 지정함
- 이와 같이 지정하면 문장 순서에 의해 layer들이 생성됨

출력 단자 숫자

```
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'))
```

레이어의 유형

활성화 함수

Layer 유형

- 참고 keras.layers
  - **class Conv2D:** 2D convolution layer (e.g. spatial convolution over images).
  - **class Dense:** Just your regular densely-connected NN layer.

  - class Flatten: Flattens the input. Does not affect the batch size.
  - class Reshape: Reshapes an output to a certain shape.
  - class InputLayer: Layer to be used as an entry point into a Network (a graph of layers).

  - class MaxPool2D: Max pooling operation for spatial data.
  - class AveragePooling2D: Average pooling operation for spatial data.
  - class GlobalAveragePooling2D: Global average pooling operation for spatial data.

  - class BatchNormalization: Normalize and scale inputs or activations. (Ioffe and Szegedy, 2014).
  - class Dropout: Applies Dropout to the input.

  - class Embedding: Turns positive integers (indexes) into dense vectors of fixed size.
  - **class SimpleRNN:** Fully-connected RNN where the output is to be fed back to input.
  - **class LSTM:** Long Short-Term Memory layer - Hochreiter 1997.
  - class GRU: Gated Recurrent Unit - Cho et al. 2014Sigmoid $\sigma(u)$

훈련 방법

- Layer들이 지정되면 compile 함수를 통해 훈련 방법을 지정

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

compile 파라미터

- keras.optimizers
  - class SGD: Stochastic gradient descent and momentum optimizer.
  - class Adagrad: Optimizer that implements the Adagrad algorithm.
  - class RMSprop: Optimizer that implements the RMSprop algorithm.
  - class Adam: Optimizer that implements the Adam algorithm.

- keras.losses
  - class MeanSquaredError: Computes the mean of squares of errors between labels and predictions.
  - class MeanAbsoluteError: Computes the mean of absolute difference between labels and predictions.
  - class BinaryCrossentropy: Computes the cross-entropy loss between true labels and predicted labels.
  - class CategoricalCrossentropy: Computes the crossentropy loss between the labels and predictions.
  - class SparseCategoricalCrossentropy: Computes the crossentropy loss between the labels and predictions.

compile 파라미터

- keras.metrics
  - class Accuracy: Calculates how often predictions matches labels.
  - class MeanAbsoluteError: Computes the mean absolute error between the labels and predictions.
  - class MeanSquaredError: Computes the mean squared error between y_true and y_pred.

모델 훈련

- 입력 데이터를 이용하여 생성된 네트워을 훈련시킴

```
network.fit(train_data, train_labels, epochs=10, batch_size=32)
```

  - batch_size: 배치 크기 (default 32)
  - epochs: 총 epoch 수 (epoch는 training set을 한번 실행하는 단위)

성능 평가

- 테스트 데이터를 이용하여 훈련된 네트워 성능을 평가

```
test_loss, test_acc = network.evaluate(test_data, test_labels)
```

사례 1: 필기체 숫자 인식

- 미국의 MNIST에서 구축한 28x28 크기의 숫자들을 인식하는 프로그램을 구성: 이 프로그램은 "***Deep learning with Python***" 책에 있음
- 영상 데이터는 keras에서 제공되는데, train과 test 용으로 구분되어 있음

- **데이터 읽기**

```
from keras.datasets import mnist

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

데이터 확인

- 훈련 데이터 확인

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

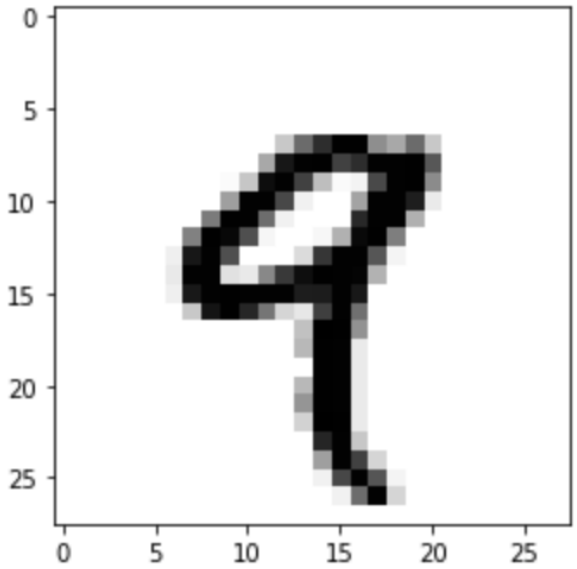
- 테스트 데이터 확인

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

데이터 영상 보기

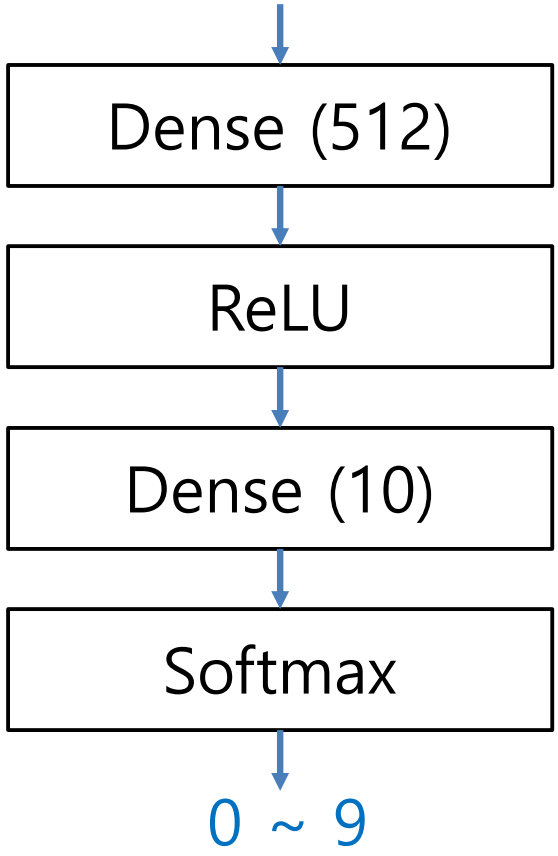
- 영상 보기

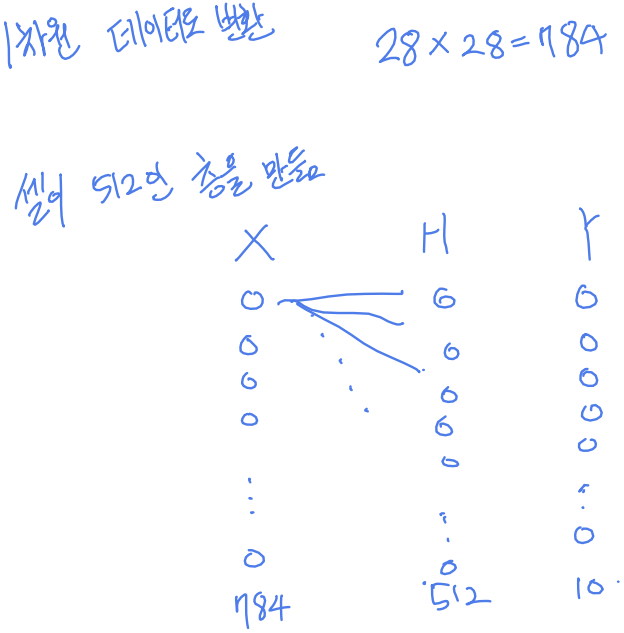
```
digit = train_images[4]

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



숫자 인식 시스템 구성

- 512개와 10개의 뉴런을 가지는 은닉층과 출력층을 구성





신경망 프로그래밍

- Sequential API로 구현할 수 있음

```
from keras import models
from keras import layers

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

X 와 Y 데이터를 신경망에 연결해야함

영상 변환

- 영상은 28x28 크기의 정수(0~255) 밝기를 가지고 있는데, 신경망에서는 784x1 크기로 0~1 사이의 밝기 영상을 이용하므로 데이터 형태를 변환

```
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
```

영상 레이블 변환

- 영상 레이블을 카테고리 형으로 변환

```
from keras.utils import to_categorical

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

훈련 과정

- compile과 fit을 수행

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

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

전체 프로그램

```
# Loading the MNIST dataset
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

from keras import models
from keras import layers
# Network architecture
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
# The compilation step
network.compile(optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy'])
# Preparing 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)
# Fit the model to training data
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

신경망 파라미터

- 신경망을 정의한 다음 network.summary() 함수를 호출하면 신경망에서 설정된 파라미터(미지수) 개수를 볼 수 있음

```
Model: "sequential_1"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
dense_2 (Dense) (None, 512) 401920
_________________________________________________________________
dense_3 (Dense) (None, 10) 5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
```

dense_2: 28x28x512+512 = 401,920
dense_3: (512+1)x10 = 5,130

실행 결과

- 훈련 데이터에 대해 98% 정도의 인식률을 보임

```
Epoch 1/5
469/469 [==============================] - 2s 4ms/step - loss: 0.2570 -
accuracy: 0.9265
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.1045 -
accuracy: 0.9690
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0696 -
accuracy: 0.9788
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0502 -
accuracy: 0.9850
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0391 -
accuracy: 0.9883
```

사례 2:
IMDB 영화평 긍정/부정 분류

IMDB 데이터

- Internet Movie Database에는 일반인들의 영화평 50,000개가 저장되어 있음
- 영화평은 텍스트로 구성되는데 긍정 또는 부정의 태그가 붙어있음
- 여기서는 영화평 텍스트로부터 평가가 긍정인지 부정인지 추정하는 것이 과제임
- 이 프로그램과 설명은 ***Deep Learning with Python*** 3장에 있음

IMDB 데이터 읽기

- IMDB 데이터는 keras에서 읽을 수 있음
- 아래 프로그램에서는 텍스트에서 사용하는 단어수를 10,000개로 제한하고 있음

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) =
        imdb.load_data( num_words=10000)
```

- keras에서는 전체 데이터에서 단어 빈도수를 조사하고 상위 10,000개의 단어를 사용해서 각 리뷰를 숫자 리스트로 표시
- 숫자 리스트 데이터로는 무슨 내용인지 알 수 없음

```
>>> train_data[0]
[1, 14, 22, 16, ... 178, 32]

>>> train_labels[0]
1       # 1은 이 평가가 긍정적임을 의미
```

IMDB 텍스트 보기

- 리뷰 내용을 보려면 get_word_index 함수를 이용하여 처리

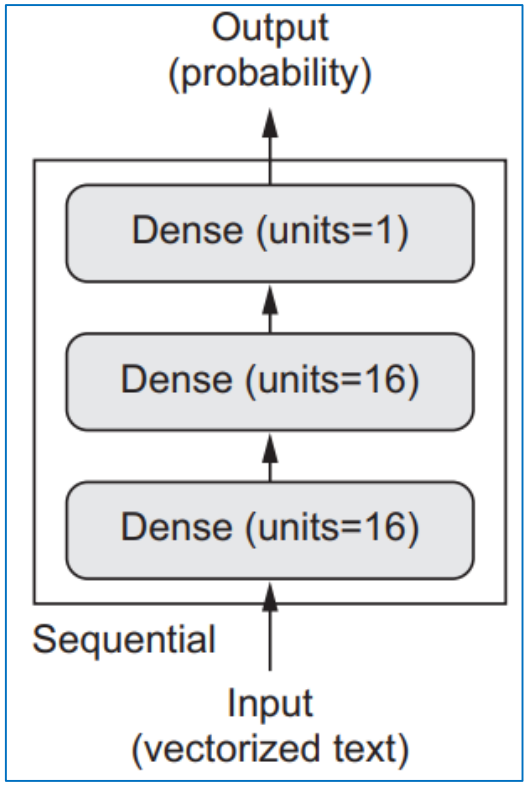
```
word_index = imdb.get_word_index()
reverse_word_index = dict(
        [(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join(
        [reverse_word_index.get(i - 3, '?') for i in train_data[0]])
```

- train_data[0]의 내용은 다음과 같음

```
? this film was just brilliant casting location scenery story
direction everyone's really suited the part they played and
you could just imagine being there robert ? is an amazing
actor and now the same being director ? father came from the
same scottish island as myself so i loved the fact there was
a real connection with this film the witty remarks throughout
the film were great it was just brilliant so much that i
bought the film as soon as it was released for ? …….
```

신경망 구조

- 텍스트 데이터에서 평가의 긍정/부정 여부를 판단하기 위해 다음과 같은 신경망 구조를 이용함
- 텍스트 입력은 10,000 크기의 Document-term matrix로 표현
- 1단: 10,000 x 16 Dense (ReLU)
  2단: 16 x 16 Dense (ReLU)
  출력단: 16 x 1 (Sigmoid)
- 기본 개념은 어떤 단어들로 리뷰했는지를 보고 긍정/부정 여부를 판단하는 것임



입력 데이터 변환

- 각 영화평을 10,000 크기의 Document-term matrix로 표현
- 다음과 같은 vectorize_sequences 함수를 이용하여 DTM을 구축

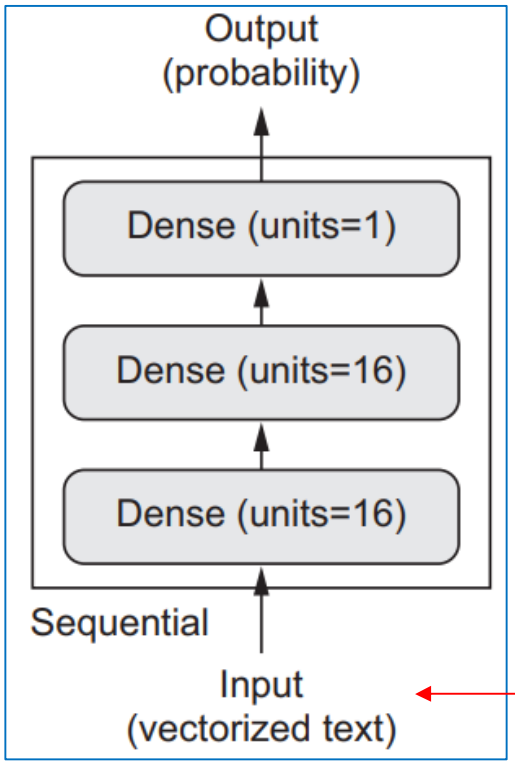
```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

- train_data에는 영화평이 25,000개가 있으므로 x_train은 25,000x10,000 크기의 행렬이 됨
- x_train의 각 원소에는 그 review에서 해당 단어의 사용여부에 따라 1 또는 0의 값이 저장됨

```
>>> train_data[0]
[1, 14, 22, 16, ... 178, 32]       # 리스트의 크기는 텍스트의 단어 숫자
>>> x_train[0]
array([ 0., 1., 1., ..., 0., 0., 0.]) # 배열의 크기는 10,000으로 고정
```

신경망 구조

```
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'))
```



크기: 10,000x1

신경망 훈련 구조

```
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)
```

신경망 파라미터 숫자

```
Model: "sequential_2"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
dense_4 (Dense) (None, 16) 160016
_________________________________________________________________
dense_5 (Dense) (None, 16) 272
_________________________________________________________________
dense_6 (Dense) (None, 1) 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
```

dense_4: 16x10,000+16 = 160,016
dense_5: 16x16+16 = 272
dense_6: 16+1 = 17

전체 프로그램

```
from keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) =
          imdb.load_data( num_words=10000)

import numpy as np
# 입력 텍스트 vectorization
def vectorize_sequences(sequences, dimension=10000):
     results = np.zeros((len(sequences), dimension))
     for i, sequence in enumerate(sequences):
          results[i, sequence] = 1.
     return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

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'))

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)
```

과제 #3 Due: 11/1

제출 방법: 프로그램과 결과를 e-class에 제출

1. 이 자료의 MNIST 숫자 인식 프로그램을 수행하여 최종 결과를 얻음. 이 자료의 29쪽에 있는 결과와 유사하게 10번까지의 epoch 결과를 제출. 정확도가 이 자료에 있는 숫자와 다르게 나타나는 이유를 설명.

자료는 5번
과제는 10번

2. IMDB 영화평을 분류하는 사례 2를 수행시키고 결과를 얻음. epoch을 6번 진행하고 정확도와 오차가 어떻게 나타나는지 결과를 제출.