기계학습 활용 (13주차)

2019. 12. 6.

Prof. Seung Ho Lee

KOREATECH
한국기술교육대학교

강의 주제 : 딥러닝 활용(최적 모델 선택)

- 이론
  - 이미지 인식

- 실습
  - 이미지 인식 (MNIST 손 글씨 인식하기)
    - ✓ 기본 프레임 구축 소스코드
    - ✓ 최적 모델 저장 소스코드 추가
    - ✓ 그래프 표시 소스코드 추가

지난 강의 요약

- 과적합 문제 이해
  - 딥러닝 모델 학습에 사용했던 데이터를 모델 평가에 그대로 사용하면 성능은 높지만 신뢰성이 떨어지는 문제(과적합) 발생
  - 따라서 주어진 데이터 셋을 서로 중복되지 않은 학습용과 테스트용으로 분리하여 딥러닝 모델 학습에 활용
  - 테스트 셋을 최대한 확보하기 위한 K겹 교차 검증 방법 소개

출처 : 모두의 딥러닝(조태호)

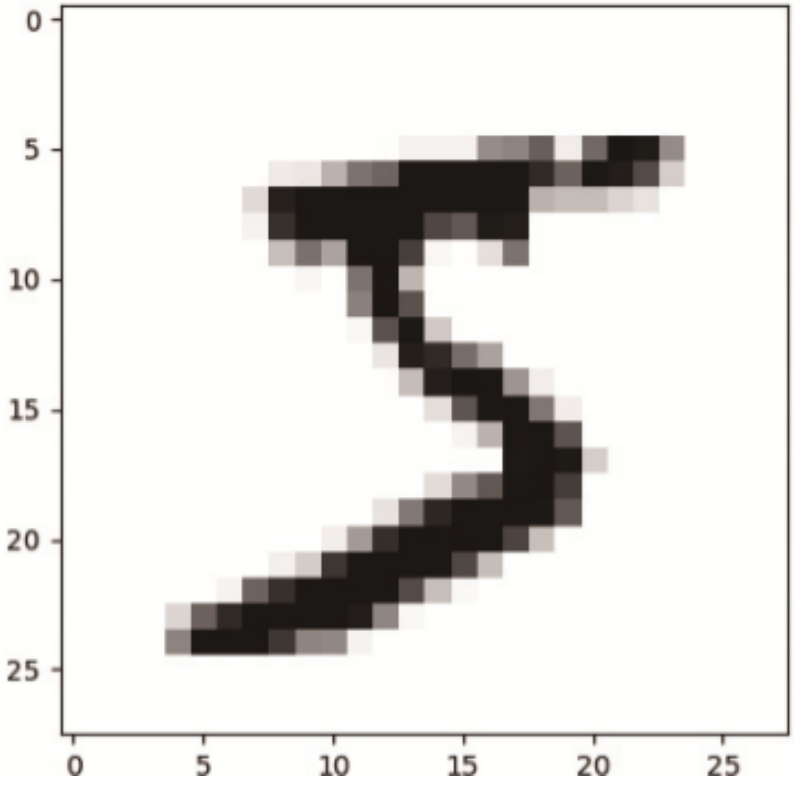
오늘 강의 요약

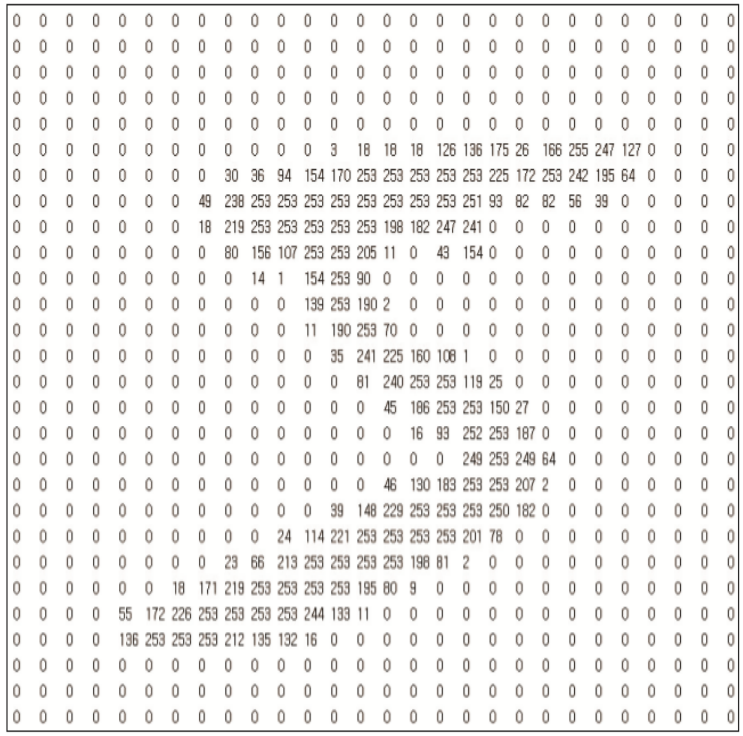
- 목표
  - 이미지 인식에 적합한 딥러닝 기본 프레임을 구현
  - 딥러닝 기본 프레임을 학습할 때 과적합을 방지하기 위한 최적 모델을 자동으로 찾고 저장하는 기능 구현
  - 학습의 반복 횟수(에포크)에 따른 딥러닝 모델의 변화 추이를 그래프로 확인

출처 : 모두의 딥러닝(조태호)

딥러닝과 이미지 인식

- 컴퓨터가 이미지를 보는 법
  - 디지털화 된 숫자(0~255의 값을 가짐)

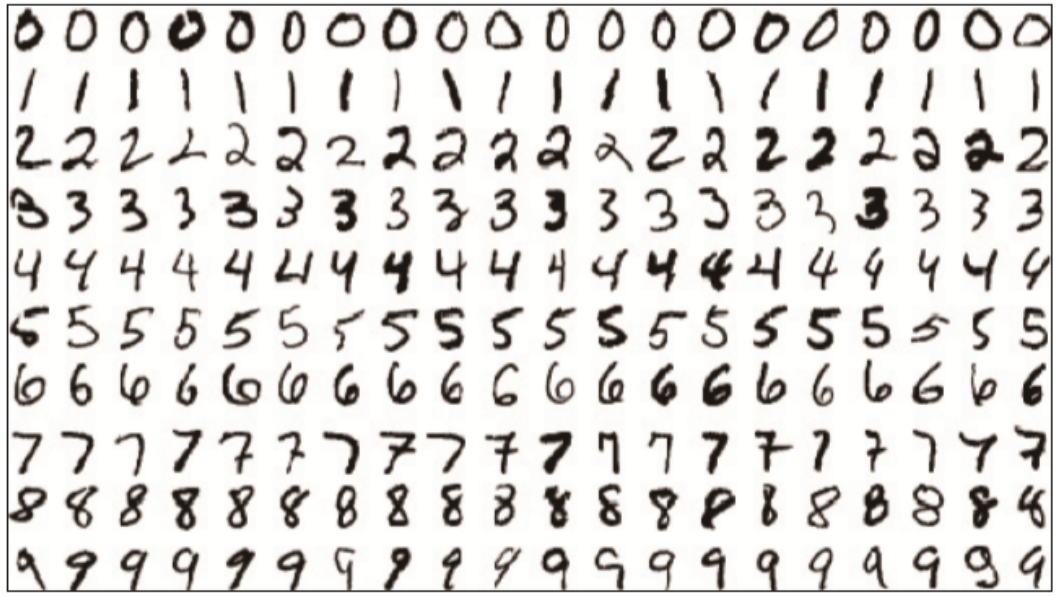




출처 : 모두의 딥러닝(조태호)

딥러닝과 이미지 인식

- MNIST 데이터 셋
  - 미국 국립표준기술원NIST에서 고등학생과 인구조사국 직원 등이 쓴 손 글씨를 이용해 만든 데이터로 구성됨
  - 총 70,000개의 샘플
    - ✓ 학습용 60,000개
    - ✓ 테스트용 10,000개
  - 클래스는 10개 (0~9)

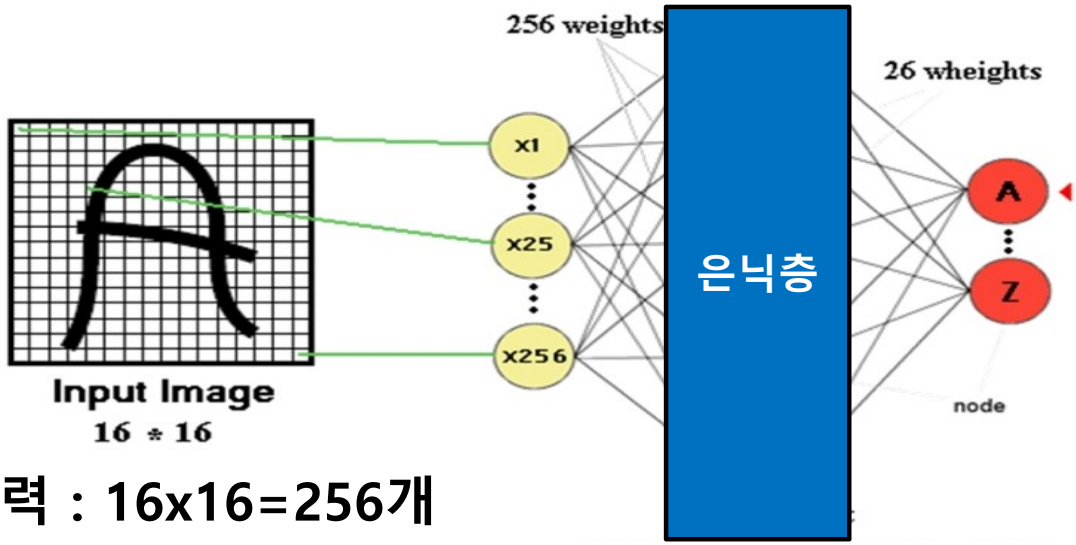


출처 : 모두의 딥러닝(조태호)

딥러닝과 이미지 인식

▪ 신경망의 구조

- **입력층 :** 픽셀개수(=사용할 속성개수)에 따라 자동 결정
- **은닉층 :** 개발자가 자유롭게 설계 가능
- **출력층 :** 클래스의 개수에 따라 자동 결정



입력 : 16x16=256개

출력 : 26개(=알파벳 개수)의 확률값(0~1)

A : 0.9
B : 0.001
C : 0.002
D : 0.002
...
Z : 0.001

딥러닝과 이미지 인식

- 신경망의 구조 (예 : 손 글씨 인식용 기본 프레임)
  - **입력층** : 노드 개수 = 픽셀 개수 (28x28=784)
  - **은닉층** : 한 층 (512개의 노드로 구성)
  - **출력층** : 노드 개수 = 클래스 개수 (0~9)
    - ✓ 다중 분류이므로 소프트맥스 함수 사용

출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 기본 프레임 구축

- 라이브러리 및 데이터 불러오기
  - ✓ 소스코드를 이용하여 케라스에서 제공하는 MNIST 데이터 셋 다운로드

출처 : 모두의 딥러닝(조태호)

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense

import numpy
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)

# MNIST 데이터 셋 불러오기
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

MNIST 손 글씨 인식 : 기본 프레임 구축

- MNIST 데이터 셋의 손 글씨 이미지 확인

X_test - NumPy array

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	84	185	159	151	60	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	222	254	254	254	254	241	198	198	198	198	198	198	198	198	170	52	0	0	0	0	0	0
9	0	0	0	0	0	0	67	114	72	114	163	227	254	225	254	254	254	250	229	254	254	140	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	17	66	14	67	67	67	59	21	236	254	106	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	83	253	209	18	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	233	255	83	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	129	254	238	44	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	249	254	62	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	133	254	187	5	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	205	248	58	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	126	254	182	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	75	251	240	57	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	19	221	254	166	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	3	203	254	219	35	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	38	254	254	77	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	31	224	254	115	1	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	133	254	254	52	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	61	242	254	254	52	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	121	254	254	219	40	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	121	254	207	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Format Resize Background color

Axis: 0 Shape: (10000, 28, 28) Index: 0 Slicing: [0, :, :]

Axis: 0 Shape: (10000, 28, 28) Index: 0 Slicing: [0, :, :]

MNIST 손 글씨 인식 : 기본 프레임 구축

- MNIST 데이터 셋의 손 글씨 이미지 확인

X_test - NumPy array

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
5	0	0	0	0	0	0	0	0	169	253	253	253	213	142	176	253	253	122	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	52	250	253	210	32	12	0	6	206	253	140	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	77	251	210	25	0	0	0	122	248	253	65	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	31	18	0	0	0	0	209	253	253	65	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	117	247	253	198	10	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	76	247	253	231	63	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	128	253	253	144	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	176	246	253	159	12	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	25	234	253	233	35	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	198	253	253	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	78	248	253	189	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	19	200	253	253	141	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	134	253	253	173	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	248	253	253	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	248	253	253	43	20	20	20	20	5	0	5	20	20	37	150	150	150	147	10	0
20	0	0	0	0	0	0	0	0	248	253	253	253	253	253	253	253	168	143	166	253	253	253	253	253	253	253	123	0
21	0	0	0	0	0	0	0	0	174	253	253	253	253	253	253	253	253	253	253	253	249	247	247	169	117	117	57	0
22	0	0	0	0	0	0	0	0	0	118	123	123	123	166	253	253	253	155	123	123	41	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Format | Resize | Background color

Axis: 0 Shape: (10000, 28, 28) Index: 1 Slicing: [1, :, :]

Axis: 0 Shape: (10000, 28, 28) Index: 1 Slicing: [1, :, :]

MNIST 손 글씨 인식 : 기본 프레임 구축

- 주어진 28x28픽셀의 2차원 배열을 1차원 배열(벡터)로 수정
  - ✓ 이를 위해 reshape(총 샘플 수, 속성 개수) 함수 사용
  - ✓ 학습용 총 샘플 수는 X_train.shape[0] 이용
  - ✓ 테스트용 총 샘플 수는 X_test.shape[0] 이용

```
In [6]: X_train.shape[0]
Out[6]: 60000

In [7]: X_test.shape[0]
Out[7]: 10000
```

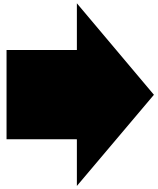
```
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
# 학습용 속성 데이터
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255
# 테스트용 속성 데이터
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 기본 프레임 구축

- 2차원 배열을 1차원 배열(벡터)로 변환한 결과

Name	Type	Size
X_test	uint8	(10000, 28, 28)
X_train	uint8	(60000, 28, 28)



Name	Type	Size
X_test	float32	(10000, 784)
X_train	float32	(60000, 784)

출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 기본 프레임 구축

- 픽셀값의 범위 수정
  - ✓ 손 글씨 이미지의 픽셀값 범위 : 0(흰색) ~ 255(검정)
  - ✓ 케라스는 데이터를 0에서 1 사이의 값으로 적용할 때 최적의 성능을 보임
  - ✓ 따라서 픽셀값을 최대값 255로 나누어 0~1사이의 값을 갖도록 수정함
  - ✓ 255로 나누기 전에 astype() 함수를 이용하여 정수를 실수로 변환해줌

```
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
# 학습용 속성 데이터
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255
# 테스트용 속성 데이터
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 기본 프레임 구축

- 원-핫 인코딩 방식 적용
  - ✓ 숫자 5 이미지의 정답 클래스는 [5]라고 저장되어 있음
  - ✓ 이것을 신경망 출력층에 맞게 [0,0,0,0,0,1,0,0,0,0] 로 바꿔야 함
  - ✓ 이를 가능하게 하는 함수가 np_utils.to_categorical()

```
Y_train = np_utils.to_categorical(Y_train, 10) # 학습용 정답 클래스 데이터
Y_test = np_utils.to_categorical(Y_test, 10) # 테스트용 정답 클래스 데이터
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 기본 프레임 구축

- 딥러닝 구조 설정
  - ✓ 784개의 속성, 10개의 클래스를 고려하여 모델 구조 설계
  - ✓ 활성화 함수 : 은닉층에서 렐루 함수, 출력층에서 소프트맥스 함수 사용

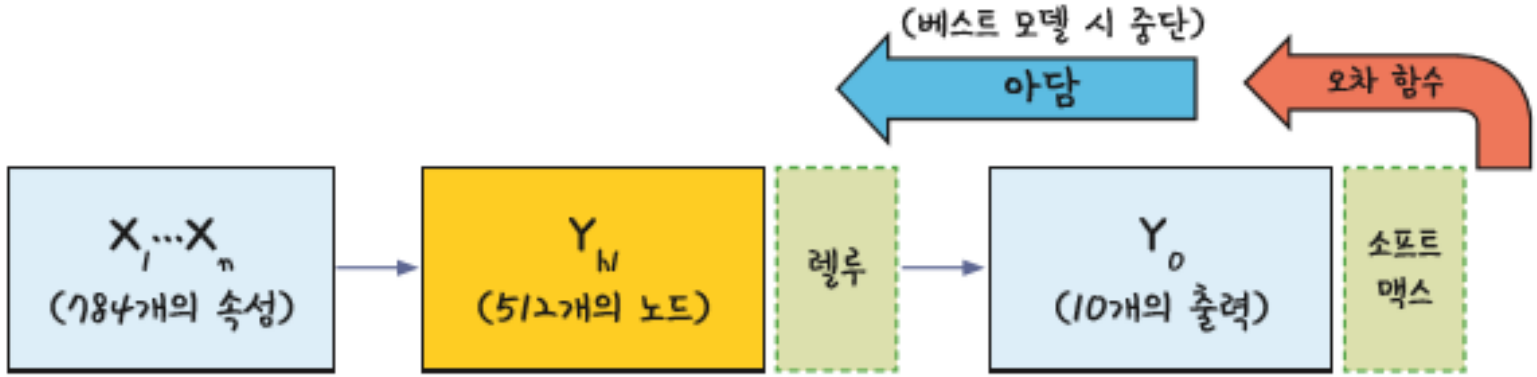
```
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 기본 프레임 구축

- 오차 함수는 mean_squared_error 사용
- 경사하강법 최적화 함수는 adam 사용

```
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['accuracy'])
```




```
model.fit(X_train, Y_train, epochs=10, batch_size=500) # 학습
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test)[1])) # 평가
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 기본 프레임 구축

- 기본 프레임 구축 소스코드

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense

import numpy
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)

# MNIST 데이터 셋 불러오기
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 기본 프레임 구축

```
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255 # 학습용 속성 데이터
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255 # 테스트용 속성 데이터

Y_train = np_utils.to_categorical(Y_train, 10) # 학습용 정답 클래스 데이터
Y_test = np_utils.to_categorical(Y_test, 10) # 학습용 정답 클래스 데이터

model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X_train, Y_train, epochs=10, batch_size=500) # 학습

print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test)[1])) # 평가
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 최적 모델 저장

- 모델의 성능을 에포크 마다 계산하고 최적 모델 찾기
  - ✓ 현재 디렉터리 안에 model이란 이름의 폴더를 모델 파일 저장 폴더로 지정
  - ✓ model 폴더가 없으면 자동으로 생성시킴

```
import os
from keras.callbacks import ModelCheckpoint,EarlyStopping

MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_acc:.5f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_acc',
verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_acc', patience=2)
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 최적 모델 저장

- 모델의 성능을 에포크 마다 계산하고 최적 모델 찾기
  - ✓ 모델 파일에 대한 파일명과 파일형식(.hdf5) 설정
  - ✓ 예를 들어, 100번째 에포크를 실행하고 난 결과 정확도가 0.93770이라면, 파일명은 100-0.93770.hdf5이 됨

```
import os
from keras.callbacks import ModelCheckpoint,EarlyStopping

MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_acc:.5f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_acc',
verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_acc', patience=2)
```

소수점 다섯자리까지 표시

출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 최적 모델 저장

- 모델의 성능을 에포크 마다 계산하고 최적 모델 찾기
  - ✓ 모니터링 할 변수 : 테스트 셋 정확도(val_acc)
  - ✓ verbose=1 : 에포크 별 학습 상태 출력(0으로 설정하면 일부 출력 안 함)
  - ✓ save_best_only=True를 설정하여 현재 시점에서 테스트 셋 정확도가 가장 높은 최적 모델만 저장

```
import os
from keras.callbacks import ModelCheckpoint,EarlyStopping

MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_acc:.5f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_acc',
verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_acc', patience=2)
```

MNIST 손 글씨 인식 : 최적 모델 저장

verbose=1 설정 ^|

```
Epoch 11/30
60000/60000 [==============================] - 1s 10us/step - loss: 0.0014
- acc: 0.9934 - val_loss: 0.0030 - val_acc: 0.9804
Epoch 00011: val_acc improved from 0.98000 to 0.98040, saving model
to ./model/11-0.98040.hdf5
```

verbose=0 설정 ^|

```
Epoch 11/30
60000/60000 [==============================] - 1s 10us/step - loss: 0.0014
- acc: 0.9934 - val_loss: 0.0030 - val_acc: 0.9804
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 최적 모델 저장

- 모델의 성능을 에포크 마다 계산하고 최적 모델 찾기
  - ✓ 모니터링 할 변수 : 테스트 셋 정확도(val_acc)
  - ✓ early_stopping_callback() : 학습 진행 중에 테스트 셋 정확도가 더 이상 높아지지 않으면 학습을 중단하는 함수
  - ✓ patience=2 : 2회 연속으로 모델의 정확도 개선이 없는 경우 학습 자동 중단

```
import os
from keras.callbacks import ModelCheckpoint,EarlyStopping

MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_acc:.5f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_acc',
verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_acc', patience=2)
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 최적 모델 저장

- 최종 선택된 모델로 정확도를 측정하여 그 결과를 출력

```
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=30,
batch_size=500, verbose=1, callbacks=[early_stopping_callback,checkpointer])

print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test) [1]))
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 : 그래프 표시

- 실행 결과를 그래프로 표현하려면 아래 소스코드 추가

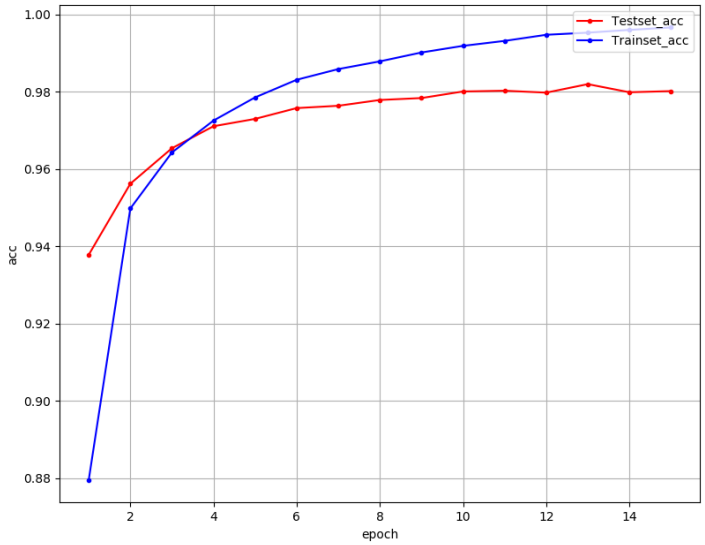
```
import matplotlib.pyplot as plt

# 테스트 셋의 정확도
y_vacc = history.history['val_acc']

# 학습 셋의 정확도
y_acc = history.history['acc']

# 그래프로 표현
x_len = numpy.arange(len(y_vacc))
plt.plot(x_len+1, y_vacc, marker='.', c="red", label='Testset_acc') # 테스트 셋 정확도는 빨갛게 표시
plt.plot(x_len+1, y_acc, marker='.', c="blue", label='Trainset_acc') # 학습 셋 정확도는 파랗게 표시

# 그래프에 격자(그리드)를 표시하고 가로축/세로축 이름 표시
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```



출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 최종 소스코드

- 손 글씨 인식 최종 소스코드
  - ✓ 기본 프레임 + 최적 모델 저장 + 그래프 표시

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint,EarlyStopping

import matplotlib.pyplot as plt
import os
import numpy
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.set_random_seed(seed)

# MNIST 데이터 불러오기
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 최종 소스코드

```
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255

Y_train = np_utils.to_categorical(Y_train, 10)
Y_test = np_utils.to_categorical(Y_test, 10)

# 모델 프레임 설정
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))

# 모델 실행 환경 설정
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['accuracy'])
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 최종 소스코드

```
# 모델 최적화 설정
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_acc:.5f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_acc', verbose=1,
save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_acc', patience=2)

# 모델의 실행
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=30,
batch_size=500, verbose=1, callbacks=[early_stopping_callback,checkpointer])

# 테스트 정확도 출력
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test) [1]))
```


출처 : 모두의 딥러닝(조태호)

MNIST 손 글씨 인식 최종 소스코드

```
# 테스트셋의 정확도
y_vacc = history.history['val_acc']

# 학습셋의 정확도
y_acc = history.history['acc']

# 그래프로 표현
x_len = numpy.arange(len(y_vacc))
plt.plot(x_len+1, y_vacc, marker='.', c="red", label='Testset_acc')
plt.plot(x_len+1, y_acc, marker='.', c="blue", label='Trainset_acc')

# 그래프에 격자(그리드)를 표시하고 가로축/세로축 이름 표시
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```


출처 : 모두의 딥러닝(조태호)

결과 해석

- 실행결과

```
Epoch 00001: val_acc improved from -inf to 0.93780, saving model to ./model/01-0.93780.hdf5
Epoch 00002: val_acc improved from 0.93780 to 0.95620, saving model to ./model/02-0.95620.hdf5
Epoch 00003: val_acc improved from 0.95620 to 0.96540, saving model to ./model/03-0.96540.hdf5
Epoch 00004: val_acc improved from 0.96540 to 0.97110, saving model to ./model/04-0.97110.hdf5
Epoch 00005: val_acc improved from 0.97110 to 0.97300, saving model to ./model/05-0.97300.hdf5
Epoch 00006: val_acc improved from 0.97300 to 0.97580, saving model to ./model/06-0.97580.hdf5
Epoch 00007: val_acc improved from 0.97580 to 0.97640, saving model to ./model/07-0.97640.hdf5
Epoch 00008: val_acc improved from 0.97640 to 0.97790, saving model to ./model/08-0.97790.hdf5
Epoch 00009: val_acc improved from 0.97790 to 0.97840, saving model to ./model/09-0.97840.hdf5
Epoch 00010: val_acc improved from 0.97840 to 0.98010, saving model to ./model/10-0.98010.hdf5
Epoch 00011: val_acc improved from 0.98010 to 0.98030, saving model to ./model/11-0.98030.hdf5
Epoch 00012: val_acc did not improve from 0.98030
Epoch 00013: val_acc improved from 0.98030 to 0.98200, saving model to ./model/13-0.98200.hdf5
Epoch 00014: val_acc did not improve from 0.98200
Epoch 00015: val_acc did not improve from 0.98200
10000/10000 [==============================] - 1s 64us/step

 Test Accuracy: 0.9802
```

두 번 연속으로 정확도 개선에 실패하자 학습이 자동으로 중단됨

출처 : 모두의 딥러닝(조태호)

결과 해석

- 실행결과
  - ✓ 1~15번째 에포크까지 학습을 반복 수행한 뒤 중단됨
  - ✓ 13번째 에포크일 때의 모델이 최적이다!