초보자 딥러닝 입문하기

- keras를 이용한 딥러닝을 이용한 손글씨 인식

학습 내용

- MNIST 데이터 셋을 활용하여 간단한 딥러닝 모델을 구현해 본다.

라이브러리 불러오기

- 대표적인 딥러닝 구현 라이브러리
  - Tensorflow, Keras, Pytorch

환경

- tensorflow 2.6.0
- keras 2.6.0
- python 3.8.8

In [3]:

```
import tensorflow as tf
import keras
import sys
```

In [4]:

```
print(tf.__version__)
print(keras.__version__)
print(sys.version)
```

```
2.6.0
2.6.0
3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
```

In [5]:

```
### 라이브러리 확인
import numpy        # 선형대수 관련(배열 생성 관련)
import matplotlib   # 시각화
```

케라스(Keras) 의미 이해

- ONEIROS(Open-ended Neuro-Electronic Intelligent Robot Operating System) 프로젝트 일부
- 오네이로스의 의미는 꿈을 의인화 시킨 신이다.

케라스의(Keras) 주요 특징 4가지

- 개발 및 유지 보수 : 프랑소와 쏠레(Francois Chollet) - 구글에 있음.(tf 2.0)
- 모듈화(Modularity) : 모듈은 독립적이고 최소한의 제약사항으로 연결
- 최소주의(Minimalism) : 각 모듈은 짧고 간결하다.
- 쉬운 확장성 : 새로운 클래스나 함수로 모듈을 아주 쉽게 추가
- 파이썬 기반 : 파이썬 코드로 모델들이 정의

케라스(Keras) 딥러닝 모델 만들기 절차 이해

- 가. 데이터 셋 생성 및 데이터 형태 맞추기
  - 데이터 준비(훈련셋, 검증셋, 시험셋 등)
  - 딥러닝 모델의 학습 및 평가를 위한 데이터 형태 맞추기(포맷 변환)
- 나. 모델 만들기
  - 모델(Sequential)을 생성 후, 레이어를 추가하여 구성
  - 복잡한 모델을 구성시에 Keras API를 사용
- 다. 모델 학습과정 설정하기(학습에 대한 설정 - compile())
  - 학습에 대한 설정, 손실 함수 및 최적화 방법 정의
- 라. 모델 학습(모델을 훈련셋(train)으로 학습 - fit() 함수)
- 마. 학습과정 살펴보기(훈련셋, 검증셋의 손실 및 정확도 측정)
- 바. 모델 평가(evaluate()) - 준비된 시험셋으로 학습 모델 평가
- 사. 모델 사용하기(predict())

코드 이해

- 십진수 숫자를 0,1로 이루어진 것으로 변환
  - from keras.utils import np_utils
- 데이터 셋(손글씨)
  - from keras.datasets import mnist
- 모델 만들기
  - from keras.models import Sequential
- 딥러닝 층 구성
  - (Dense-층세부내용 Activation:활성화함수)
  - from keras.layers import Dense, Activation

In [6]:

```
# 사용할 패키지 불러오기
from keras.utils import np_utils
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [21]:

```
# 데이터셋 생성하기
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [22]:

```
import matplotlib.pyplot as plt
```

손글씨 데이터 시각화

- y_train : 손글씨 그림의 숫자(0~9) 정보
- x_train : 손글씨 그림의 픽셀정보(28x28) 총 784개의 픽셀정보

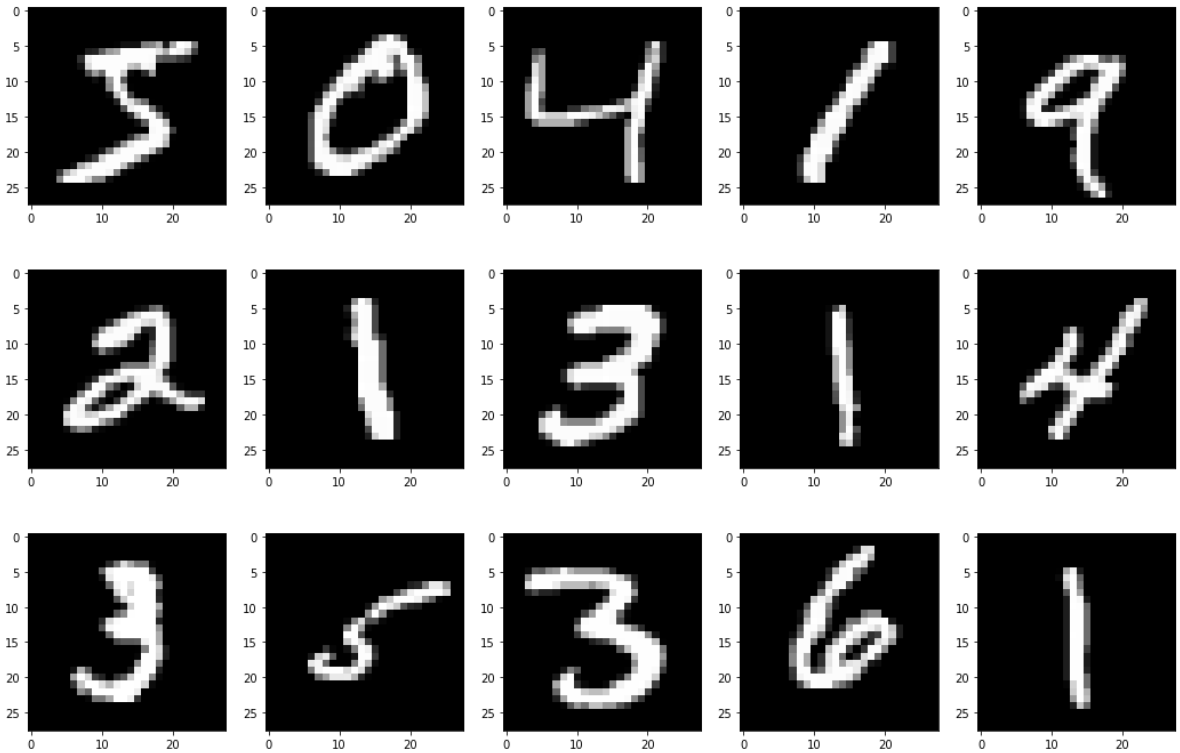
In [23]:

```
figure,axes = plt.subplots(nrows=3, ncols=5)  # 3행 5열의 구조
figure.set_size_inches(18,12)  # 전체 크기

plt.gray()
print("label={}".format(y_train[0:15]))   # x데이터 0~14개 가져오기

col = 0
for row in range(0,3):
    col = row * 5
    axes[row][0].imshow(X_train[col])  # 0,5,10의 값을 갖는 위치 값 이미지 표시
    axes[row][1].imshow(X_train[col+1])# 1,6,11의 값을 갖는 위치 값 이미지 표시
    axes[row][2].imshow(X_train[col+2])# 2,7,12의 값을 갖는 위치 값 이미지 표시
    axes[row][3].imshow(X_train[col+3])# 3,8,13의 값을 갖는 위치 값 이미지 표시
    axes[row][4].imshow(X_train[col+4])# 4,9,114의 값을 갖는 위치 값 이미지 표시
```

```
label=[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1]
```



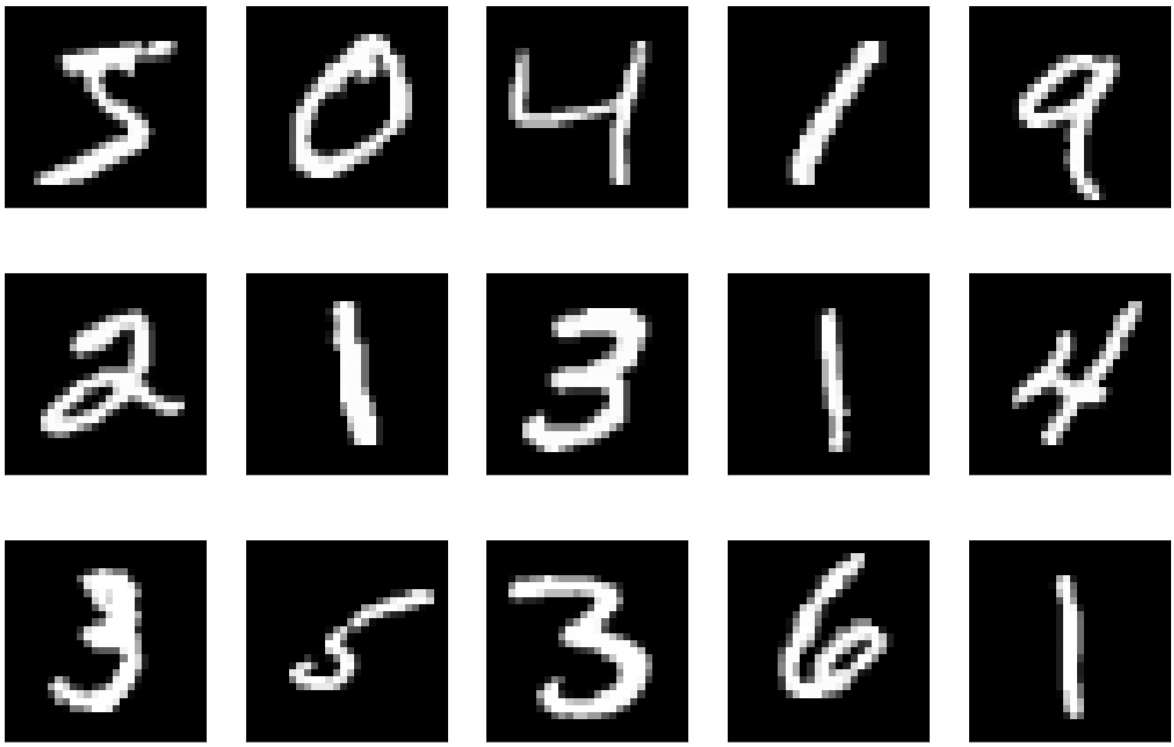
In [24]:

```
fig, axes = plt.subplots(3, 5, figsize=(18,12),
                         subplot_kw= {'xticks':(), 'yticks':() })

print("label={}".format(y_train[0:15]))   # x데이터 0~14개 가져오기

for image, ax in zip( X_train, axes.ravel() ):
    ax.imshow(image) # 이미지 표시
```

```
label=[5 0 4 1 9 2 1 3 1 4 3 5 3 6 1]
```



데이터 처리

- x_train, x_test (60000, 28, 28) -> (60000, 784)
- y_train, y_test 숫자(0~9)를 2진 벡터형태로 변경(0 0 0 0 0 1 0 0 0 0 0)

In [25]:

```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

모델 만들기

- 첫번째 add =>
  - Dense() : 층의 세부 내용 지정
  - input_dim : 입력층의 뉴런 개수, units : 입력층 이후의 은닉층의 뉴런 개수
  - activation : 활성화 함수 지정
- 두번째 add =>
  - units = 10 : 손글씨의 예측 값이 0~9사이의 값이므로 10개
  - 일반적으로 마지막 층(출력층)이 범주형 여러개 예측일 경우 softmax 함수를 사용

In [26]:

```
# 2. 모델 구성하기
model = Sequential()
model.add(Dense(units=64, input_dim=28*28, activation='relu'))
model.add(Dense(units=10, activation='softmax'))
```

모델의 세부 설정

- 비용함수(loss) : categorical_crossentropy(범주형(다항분류)의 경우)
- 최적화함수(optimizer) : sgd(Stochastic Gradient Descent) 알고리즘 이용
- 최종 평가(metrics) : 정확도로 측정(손글씨를 정답을 맞춰는지 아닌지)

In [27]:

```
# 3. 모델 학습과정 설정하기
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

모델 학습시키기

In [28]:

```
# 4. 모델 학습시키기
hist = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 13.8581 - accuracy:
0.1291 - val_loss: 2.3012 - val_accuracy: 0.1136
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step - loss: 2.3017 - accuracy:
0.1128 - val_loss: 2.3008 - val_accuracy: 0.1137
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 2.2879 - accuracy:
0.1243 - val_loss: 2.3014 - val_accuracy: 0.1135
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 2.3014 - accuracy:
0.1124 - val_loss: 2.3011 - val_accuracy: 0.1135
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 2.3013 - accuracy:
0.1124 - val_loss: 2.3011 - val_accuracy: 0.1135
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 2.3013 - accuracy:
0.1124 - val_loss: 2.3010 - val_accuracy: 0.1135
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 2.3013 - accuracy:
0.1124 - val_loss: 2.3010 - val_accuracy: 0.1135
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 2.3013 - accuracy:
0.1124 - val_loss: 2.3010 - val_accuracy: 0.1135
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 2.3013 - accuracy:
0.1124 - val_loss: 2.3010 - val_accuracy: 0.1135
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 2.3013 - accuracy:
0.1124 - val_loss: 2.3010 - val_accuracy: 0.1135
```

학습을 통해 얻어진 값 살펴보기

- epoch 10번이므로 값이 각각 10개씩

In [29]:

```
# 5. 학습과정 살펴보기
print('## training loss and acc ##')
print(hist.history['loss'])        # 'val_loss' : 평가셋 손실값
print(hist.history['accuracy'])    # 'val_acc'  : 평가셋 정확도
```

```
## training loss and acc ##
[13.858094215393066, 2.3017001152038574, 2.287893295288086, 2.30135178565979, 2.3012
807369232178, 2.3012688159942627, 2.3012614250183105, 2.30126690864563, 2.3012826442
718506, 2.3012659549713135]
[0.12908333539962769, 0.11275000125169754, 0.12433333694934845, 0.11236666887998581,
0.11236666887998581, 0.11236666887998581, 0.11236666887998581, 0.11236666887998581,
0.11236666887998581, 0.11236666887998581]
```

모델 평가하기

- 테스트 데이터 셋(x_test, y_test) 을 이용
- 첫번째 값이 : loss 손실값.
- 두번째 값이 : 정확도

In [31]:

```
loss_and_metrics = model.evaluate(X_test, y_test, batch_size=32)
print('## evaluation loss and_metrics ##')
print(loss_and_metrics)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 2.3010 - accuracy: 0.
1135
## evaluation loss and_metrics ##
[2.301030397415161, 0.11349999904632568]
```

실습 과제

- 가. 픽셀값을 0~255를 0~1사이로 변경해 보기
- 나. epoch=10 -> 20를 변경하면 해 보자. 어떤 현상이 발생되는가?
- 다. batch_size=32 -> 16를 변경한 이후에 해 보자.

참고 동영상 : https://www.youtube.com/watch?v=aircAruvnKk (https://www.youtube.com/watch?v=aircAruvnKk)