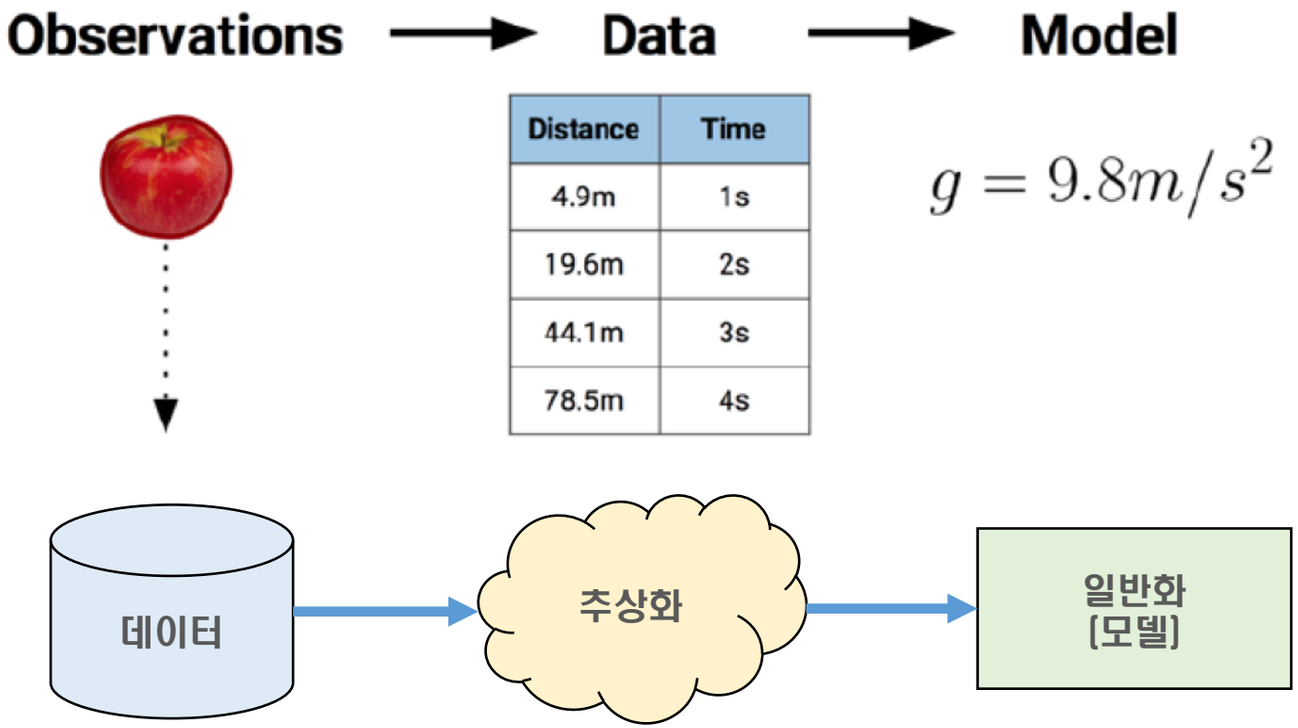
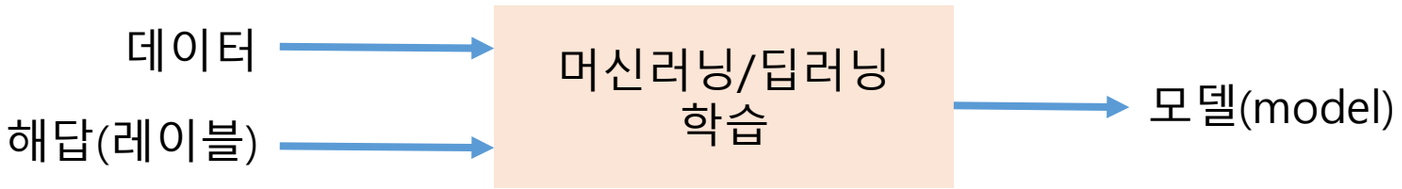
인공지능(AI)

04주차.딥러닝 학습 및 학습사이클에 대한 이해

비유 내용

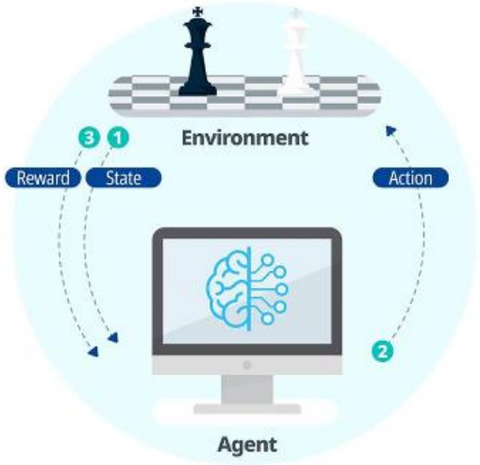
모델(model)은 관측값(observations)을 축적하여 데이터(data)를 구축하고, 데이터를 학습하여 일반화된 규칙 생성



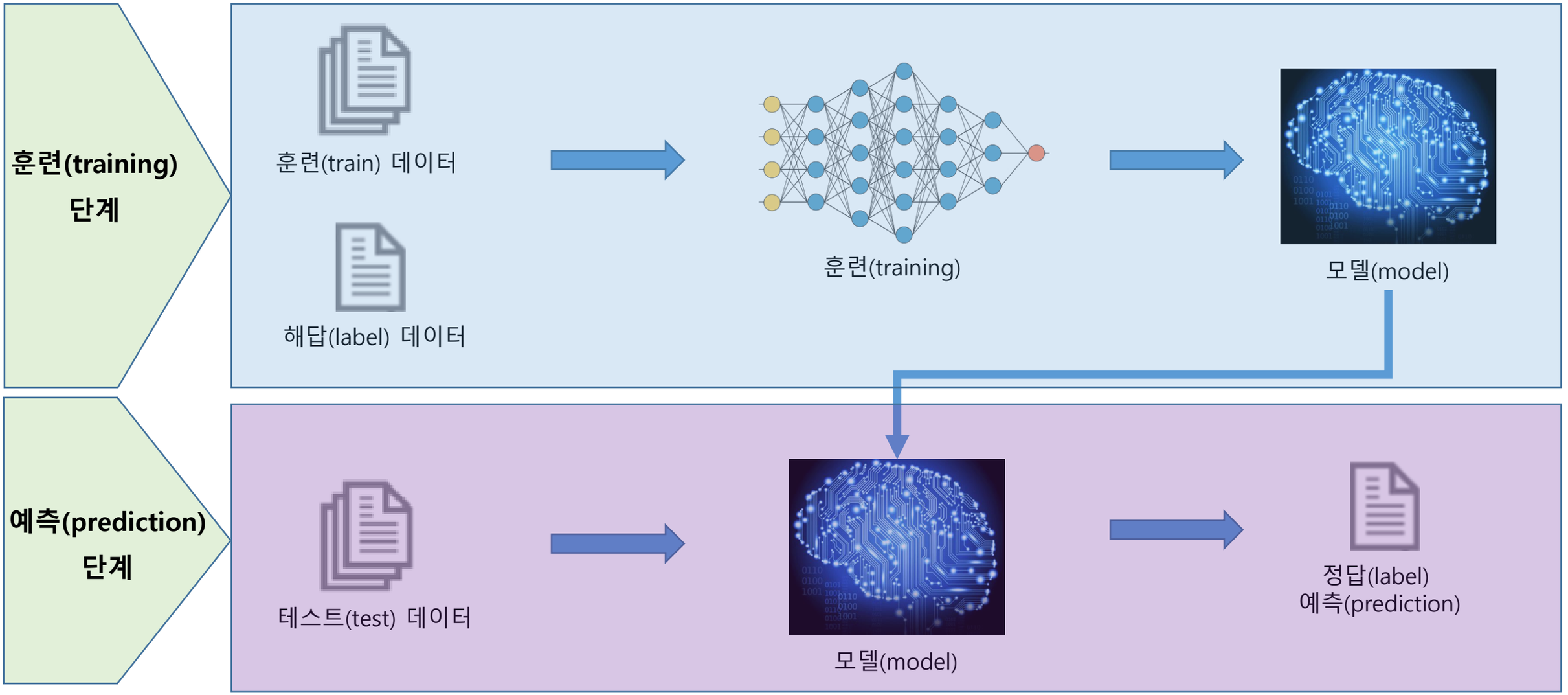


데이터 학습 과정에서 정답(레이블) 유무에 따라 지도학습과 비지도 학습으로 나눌 수 있으며,
행동심리학에 기반하여 상태와 행동에 따른 보상을 통한 학습 방법인 강화학습 등이 있음

- 머신러닝 (Machine Learning)
  - 지도 학습 (Supervised Learning)
    - 정답(레이블) 있는 데이터로 학습
    - 분류 (Classification)
      - 스팸 메일 분류
      - 필기체 숫자 인식
    - 회귀 (Regression)
      - 주가 예측
      - 시장규모 예측
  - 비지도 학습 (Unsupervised Learning)
    - 정답(레이블) 없는 데이터로 학습
    - 클러스터링 (Clustering)
      - 타겟 고객 군집화
      - 언어별 군집화
  - 강화 학습 (Reinforcement Learning)
    - 보상과 패널티를 이용한 학습
    - 의사결정을 위한 최적의 Action 선택



딥러닝 학습과 예측은 모델 구축을 위한 training 단계와 구축된 모델을 이용한 prediction 단계로 구분

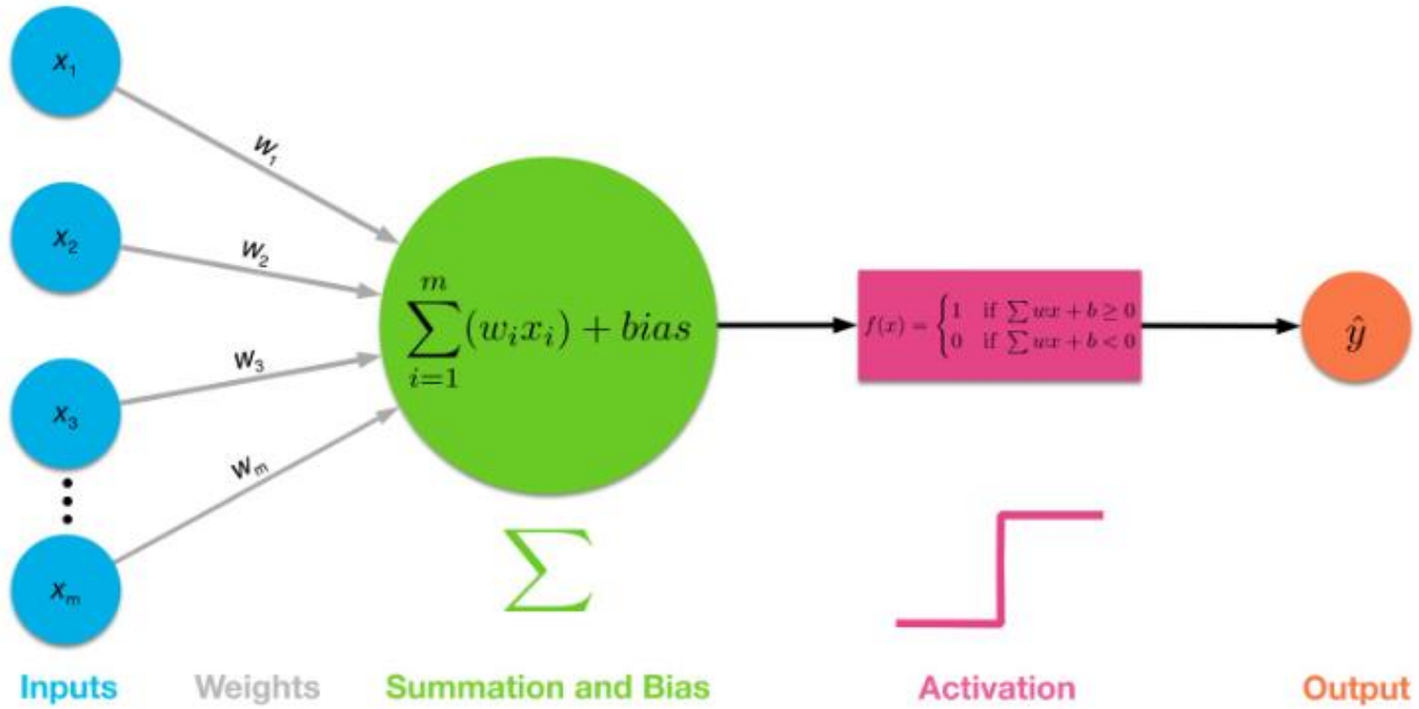


가중치(weight) : 입력신호의 강도를 표현($W_1, W_2, \cdots, W_n$)

입력신호의 총합(summation) : 각 입력신호에 가중치를 곱하여 합한 값($W_1 \cdot x_1 + W_2 \cdot x_2 + \cdots + W_n \cdot x_n = \sum W_i \cdot x_i$)

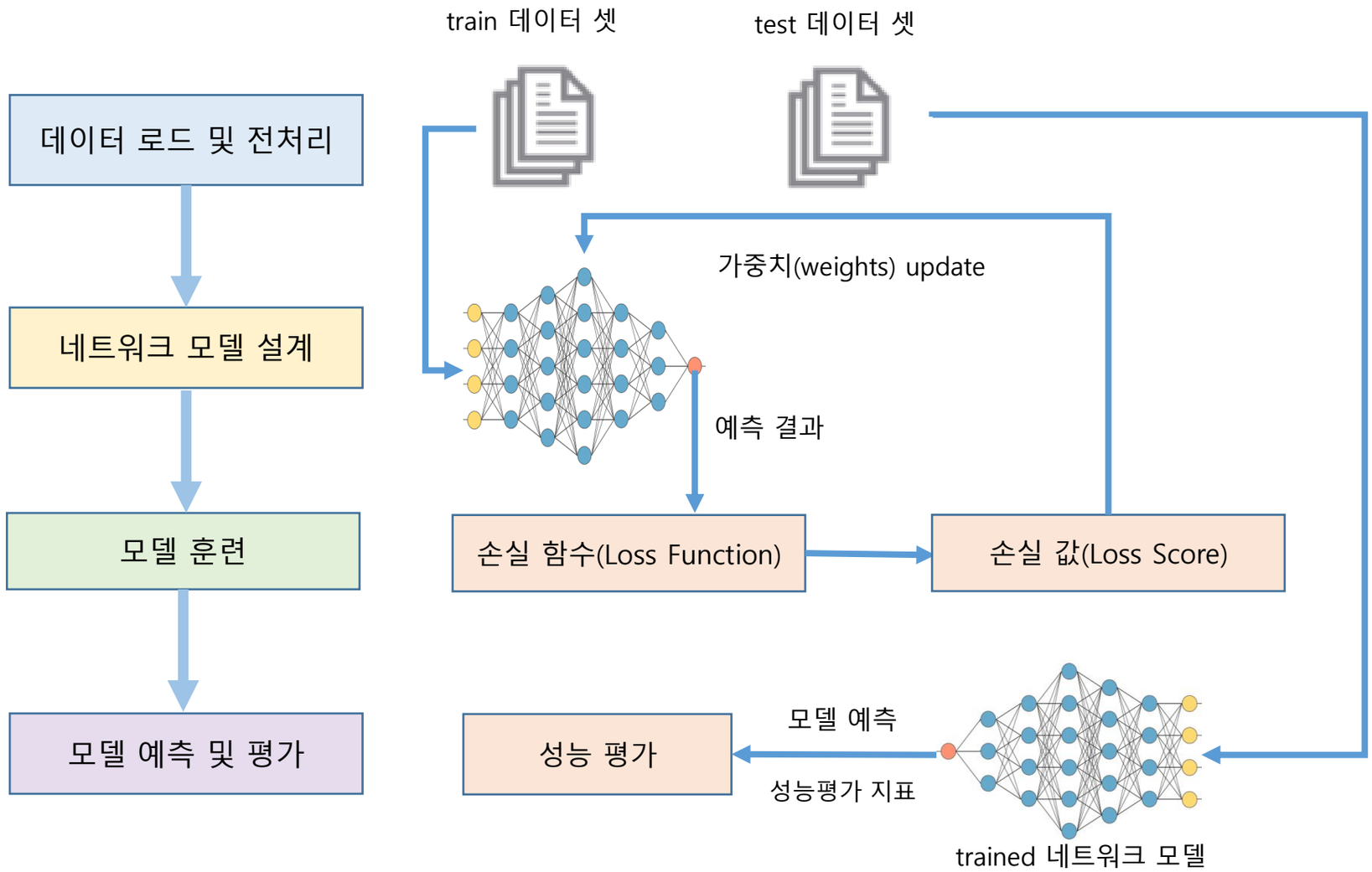
활성화 함수(activation function) : 신호의 총합을 출력신호로 변환

딥러닝 학습은 최적의 가중치($W_1, W_2, \cdots, W_n$) 값들을 찾는 과정

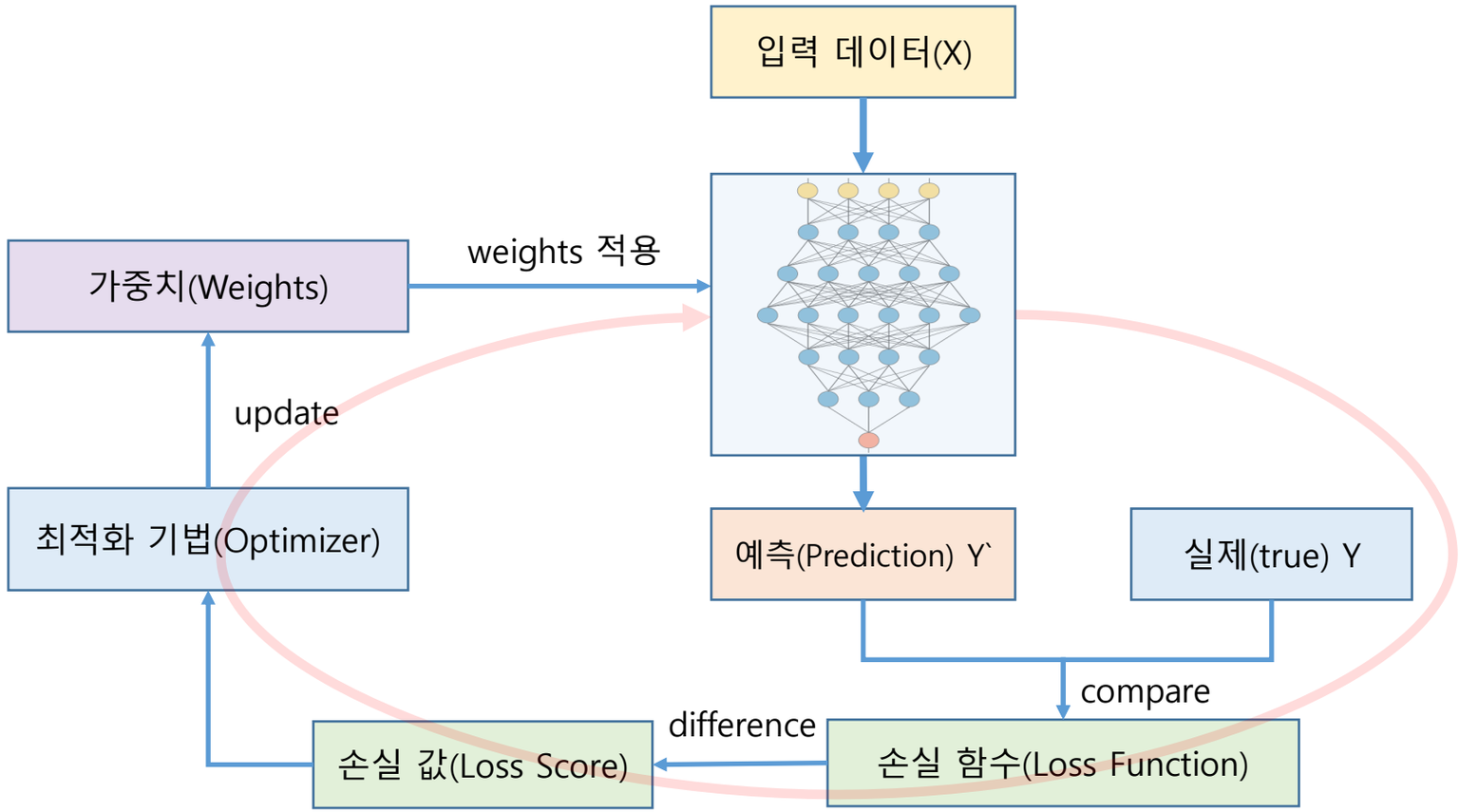


출처 : https://galaxydatatech.com/2018/06/25/multi-layer-perceptron-model/

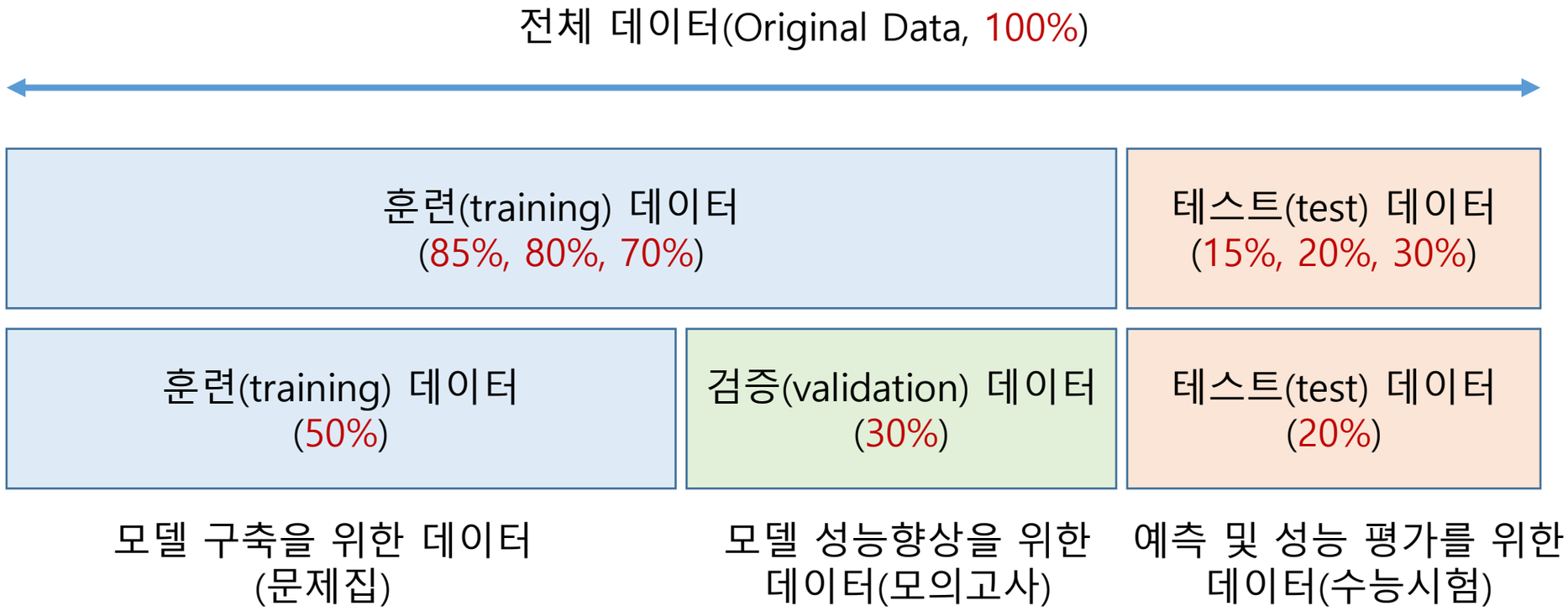
딥러닝 학습 절차는 데이터 로드 및 전처리, 네트워크 모델 설계, 모델 훈련, 모델 예측 및 평가 절차에 따라 진행



입력된 데이터(X)를 대해 뉴럴 네트워크를 통해 정답 후보 값인 Y'을 예측하여 실제 정답인 Y와 비교하여 차이 계산
계산된 차이 만큼의 손실 값(loss score)을 활용하여 최적화 기법으로 새롭게 업데이트할 가중치(weights) 계산
가중치를 업데이트하여 입력 데이터 X에 대한 새로운 정답 후보 값 Y' 예측



머신러닝/딥러닝 학습데이터는 훈련(training) 데이터, 검증(validation) 데이터 및 테스트(test)로 분할(split)하여 사용

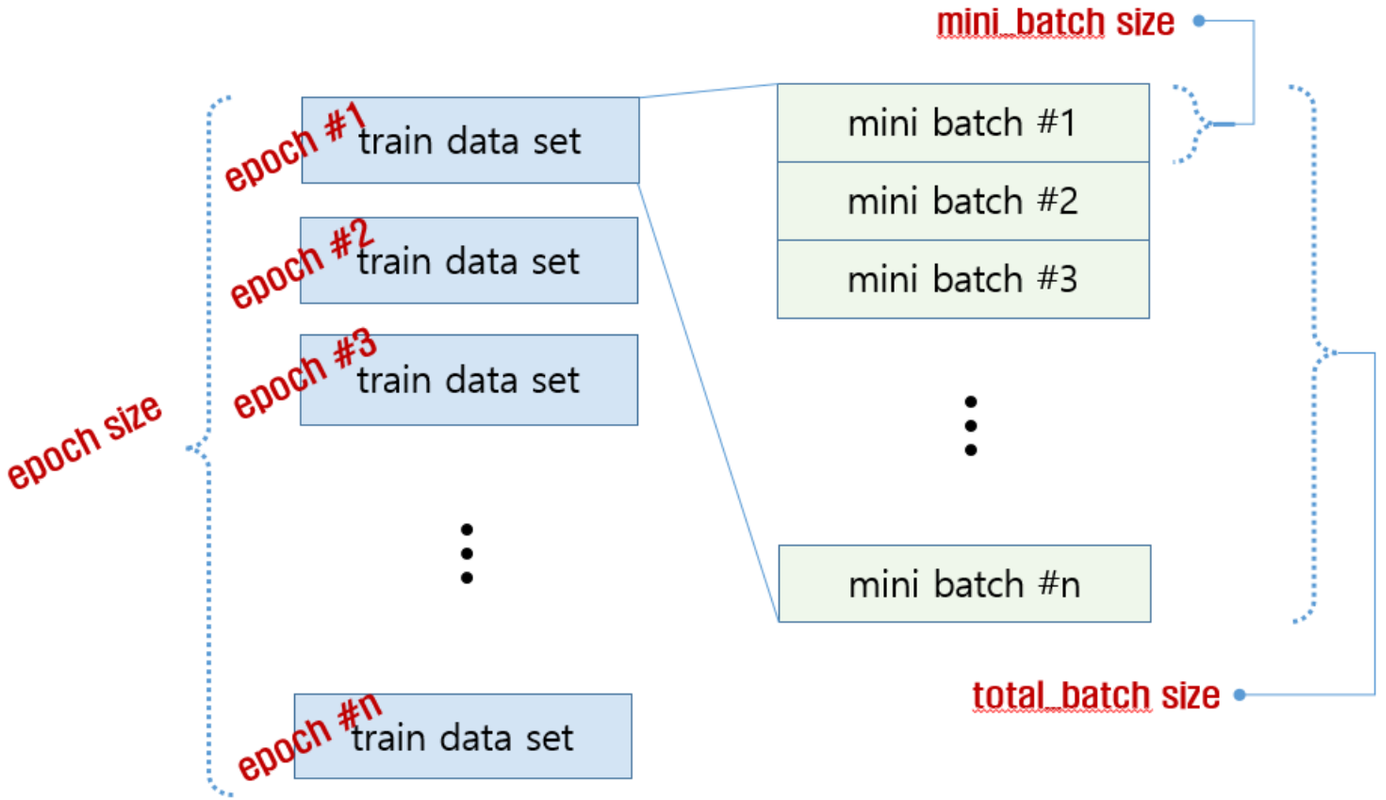


딥러닝 반복 학습은 동일한 train 데이터 셋을 mini batch 크기 만큼 분할하여 학습

mini batch size는 가중치(weights) 한 번 업데이트하는 주기

epoch는 train data set 입력 데이터로 모두 한번 사용한 주기

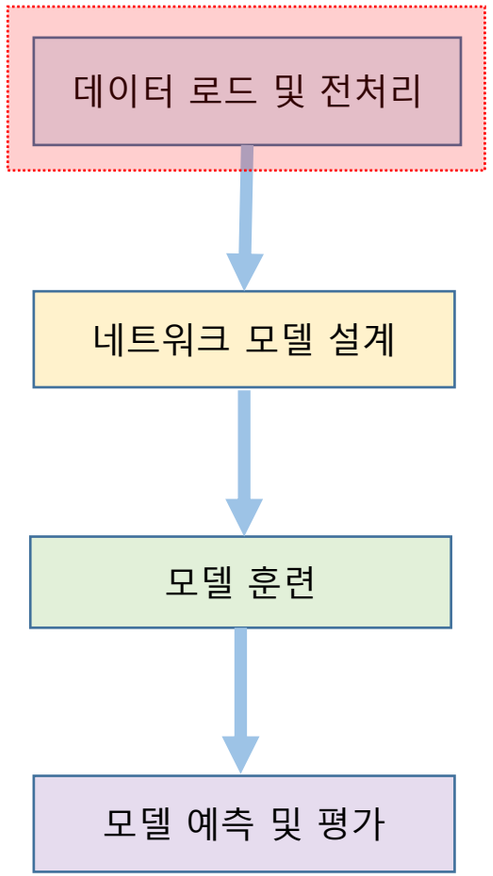
전체 epoch size는 동일한 train data set으로 반복 학습한 수



keras₩datasets디렉토리에서 mnist 모듈을 import하고 load_data 함수를 이용해서 데이터를 로드

training 데이터 셋은 train_images과 train_labels의 pair로 구성

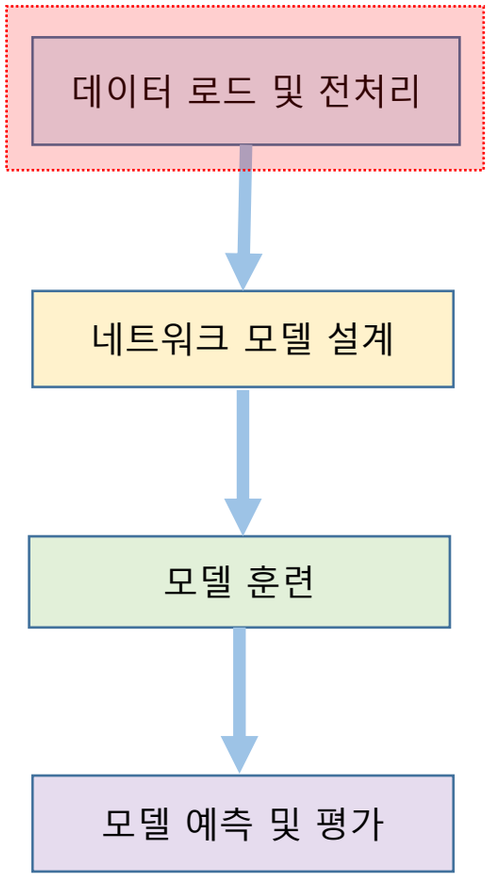
test 데이터 셋은 test_images과 test_labels의 pair로 구성



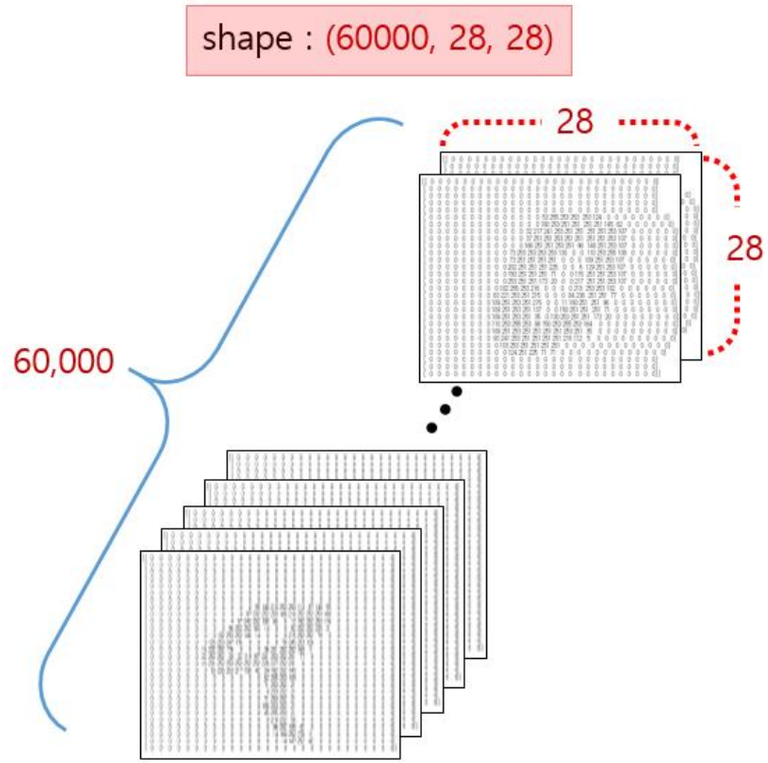

```
In [2]: from keras.datasets import mnist
        Using TensorFlow backend.

In [3]: (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

In [4]: train_images.shape
Out[4]: (60000, 28, 28)

In [5]: test_images.shape
Out[5]: (10000, 28, 28)

In [6]: train_labels.shape
Out[6]: (60000,)

In [7]: test_labels.shape
Out[7]: (10000,)
```

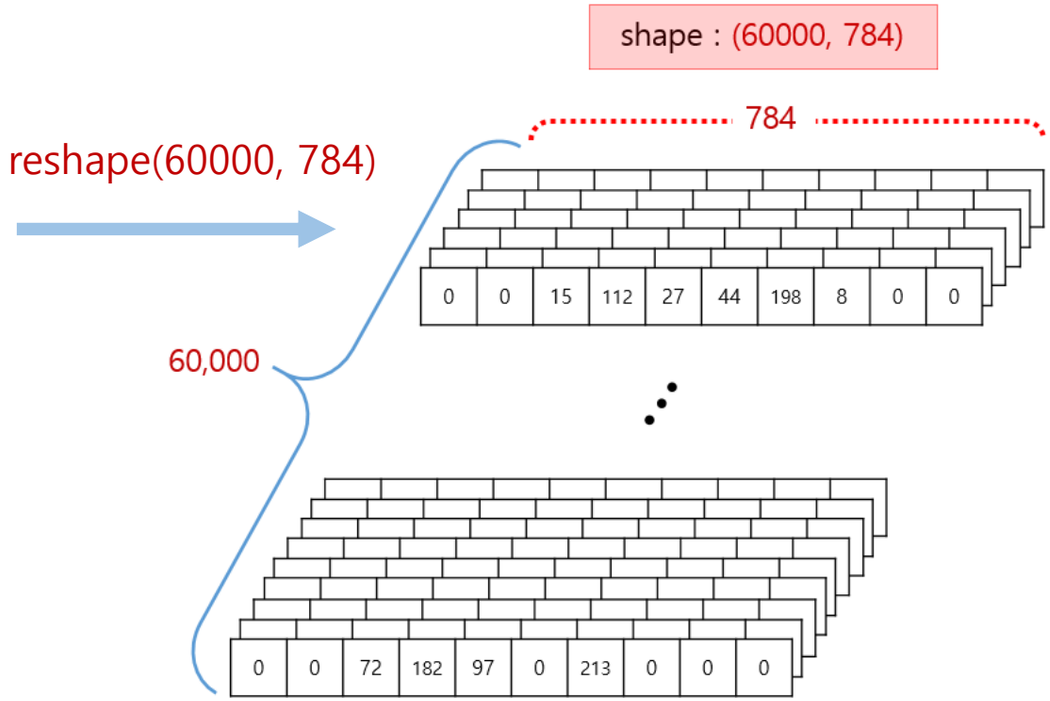
완전 연결(fully connected) 네트워크에 데이터를 입력하기 위해 이미지 데이터의 shape 변경

train_images의 shape을 (60000, 28, 28) → (60000, 784) 로 변경



train_images = train_images.reshape(60000, 784)

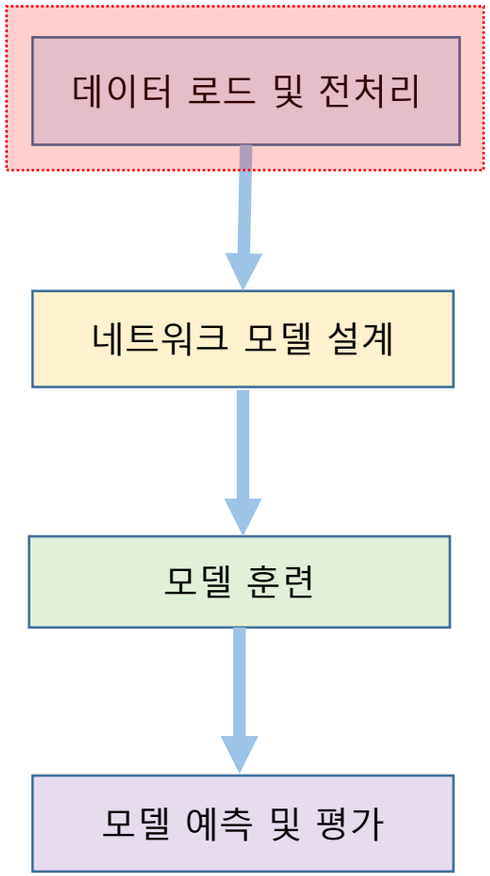


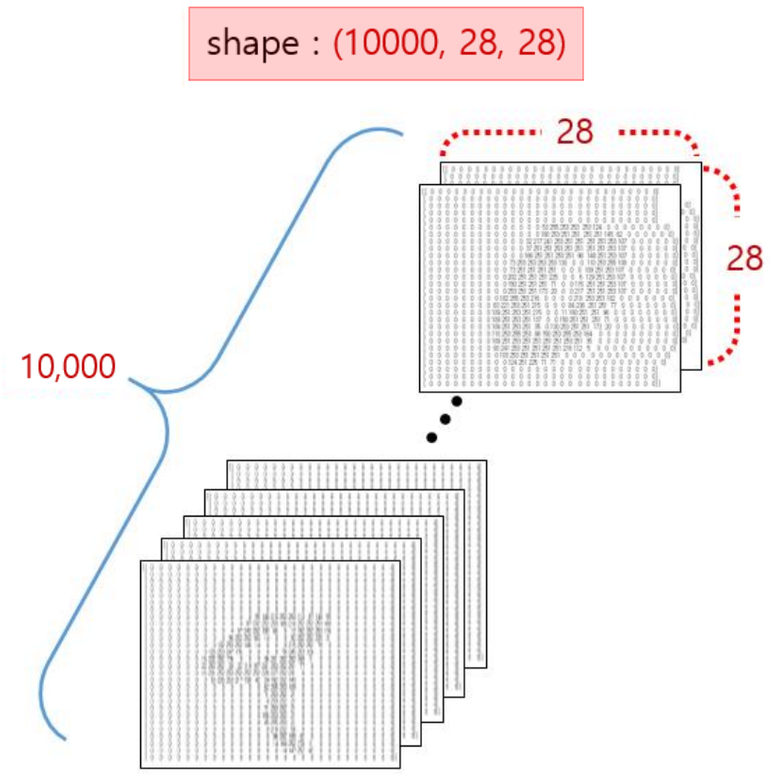


완전 연결(fully connected) 네트워크 구성을 위해 shape 변경

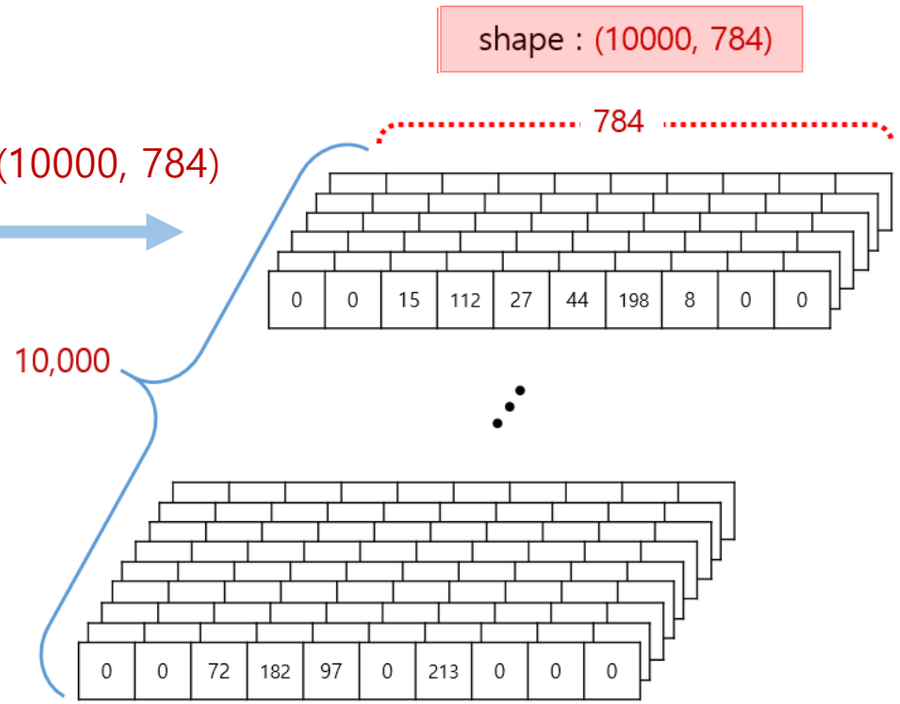
test_images의 shape을 (10000, 28, 28) → (10000, 784) 로 변경

test_images = test_images.reshape(60000, 784)

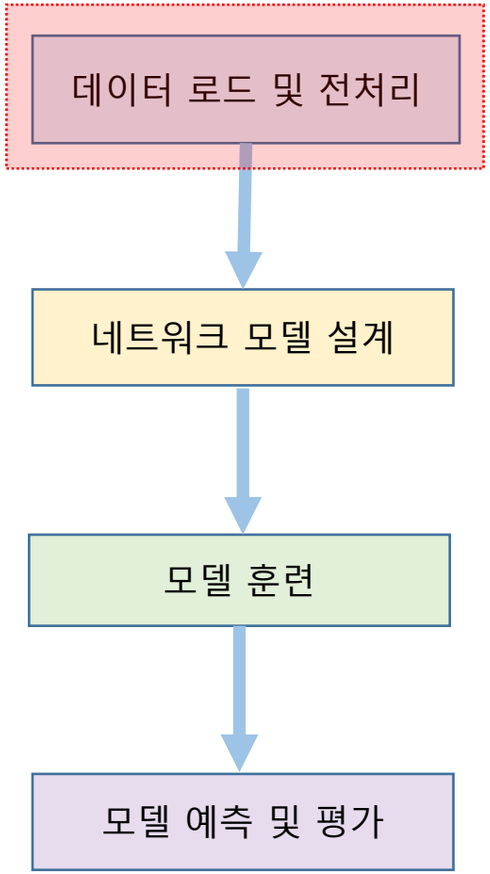




reshape(10000, 784)



딥러닝은 데이터 분포가 넓은 경우 오버피팅(over fitting)으로 인해 훈련 단계에서는 정확도가 높지만 예측 단계에서는 정확도가 낮아질 수 있으므로 데이터 스케일링을 통해 데이터의 분포를 좁게함
Training 이미지 데이터와 test 이미지 데이터는 0 ~ 255 사이의 값을 가지는데, 0 ~ 1 사이의 값을 가지도록 변경




```
train_images = train_images.astype('float32')/255
test_images = test_images.astype('float32')/255
```

```
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   3,  18,  18,  18,
126, 136, 175,  26, 166, 255, 247, 127,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170, 253,
253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253,
253, 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  18, 219, 253,
253, 253, 253, 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 80, 156, 107, 253, 253, 205,  11,   0,  43, 154,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,  14,   1, 154, 253,  90,   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, 139, 253, 190,   2,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190, 253,  70,
  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,  35,
241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

0 ~ 255 사이의 데이터

astype('float32')/255

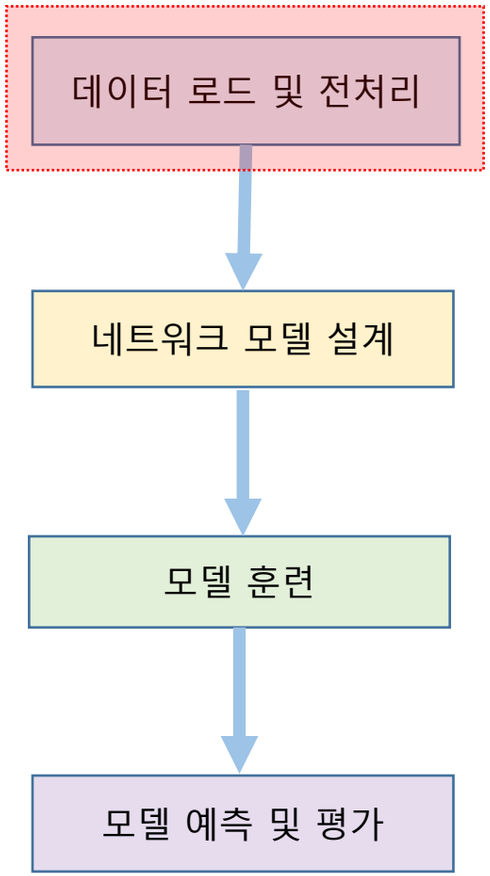
```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.01176471, 0.07058824, 0.07058824,
0.07058824, 0.49411765, 0.53333336, 0.6862745 , 0.10196079,
0.6509804 , 1.        , 0.96862745, 0.49803922, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.11764706, 0.14117648, 0.36862746, 0.6039216 ,
0.6666667 , 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.88235295, 0.6745098 , 0.99215686, 0.9490196 ,
0.7647059 , 0.2509804 , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.19215687, 0.93333334,
0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.9843137 , 0.3647059 ,
0.32156864, 0.32156864, 0.21960784, 0.15294118, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

0 ~ 1 사이의 데이터

예측 결과는 0 ~ 9일 확률로 출력되며, 최종 결과는 one-hot 벡터이므로

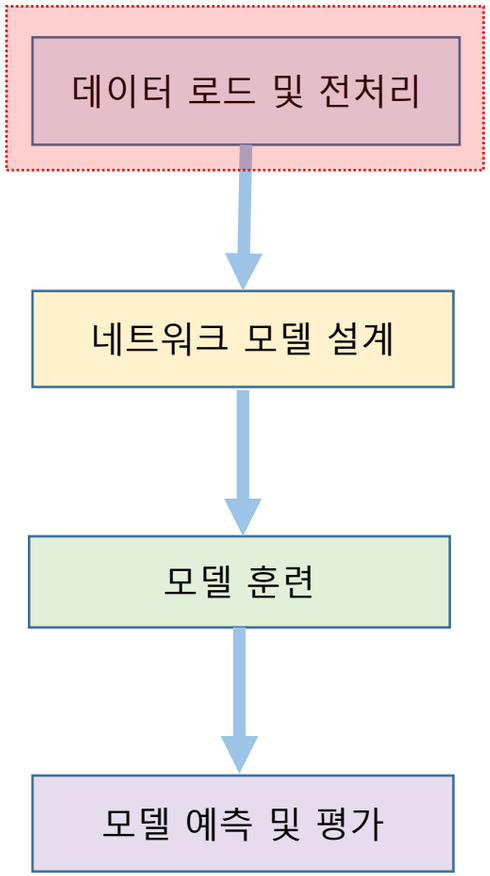
train_labels와 test_labels에 대해 one-hot 인코딩(encoding) 수행하여 비교 가능하도록 변환

train_labels의 shape은 (60000,) → (60000, 10), test_labes의 shape은 (10000,) → (10000, 10)



keras.utils의 to_categorical 함수를 import 하고

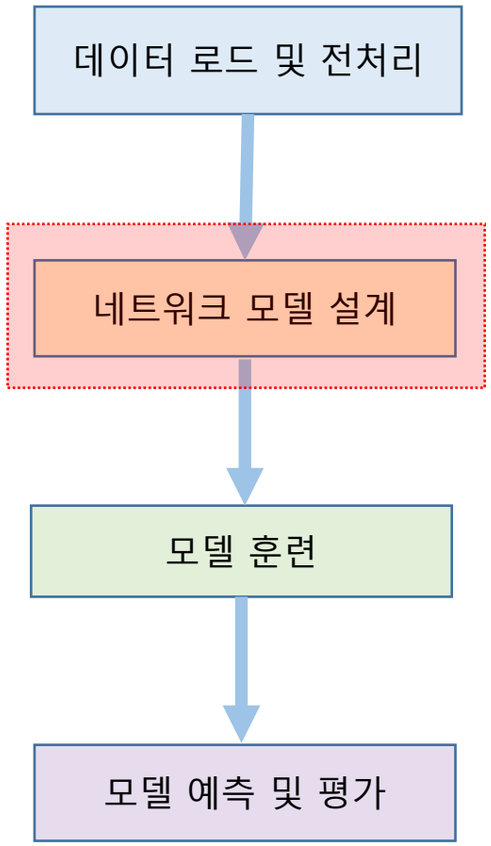
to_categorical 함수를 이용하여 one-hot 인코딩 수행

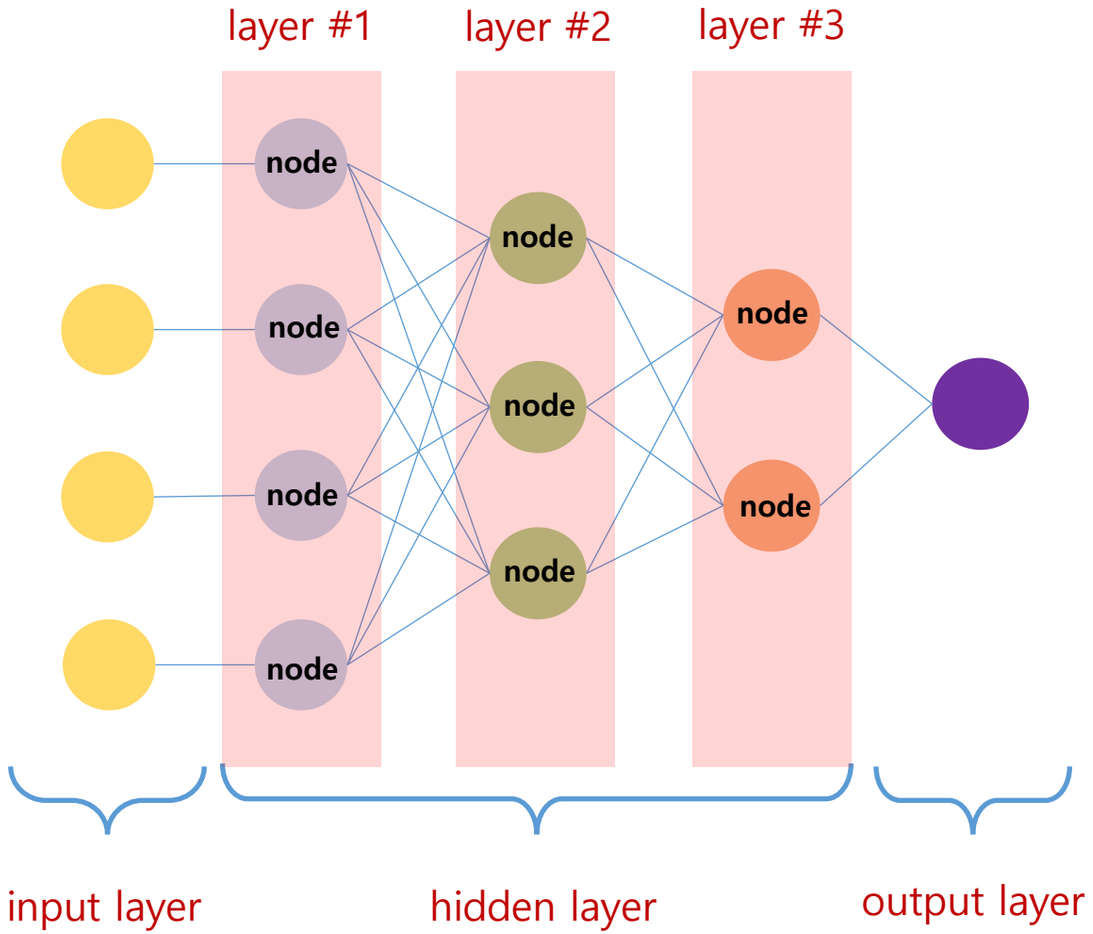



```
In [14]: train_labels[0]
Out[14]: 5

In [19]: train_labels.shape
Out[19]: (60000,)

In [15]: from keras.utils import to_categorical

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

In [16]: train_labels[0]
Out[16]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)

In [17]: train_labels.shape
Out[17]: (60000, 10)
```

딥러닝은 여러 은닉층(hidden layer)을 가지는 모델

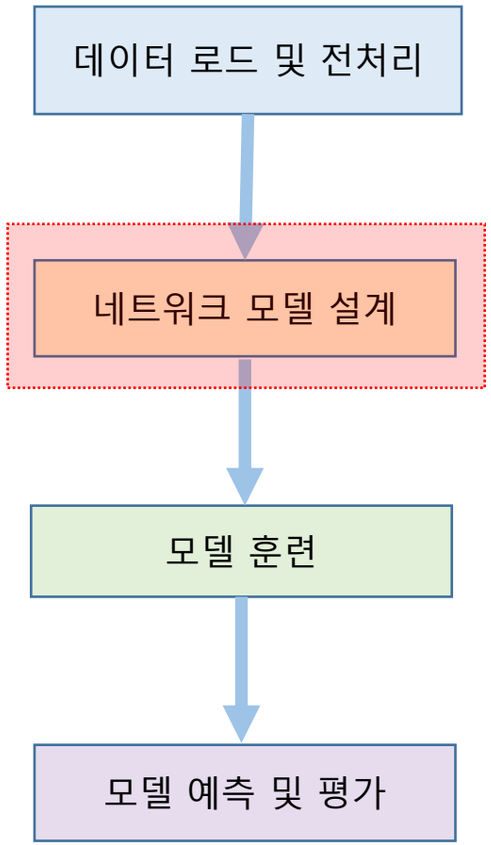
하나의 은닉층은 여러 노드로 구성





keras의 layers 라이브러리를 이용하여 model에 Dense(fully connected) layer 추가

layer 추가가 완료되면 model에 대한 컴파일(compile) 수행




```
from keras import models
from keras import layers
```

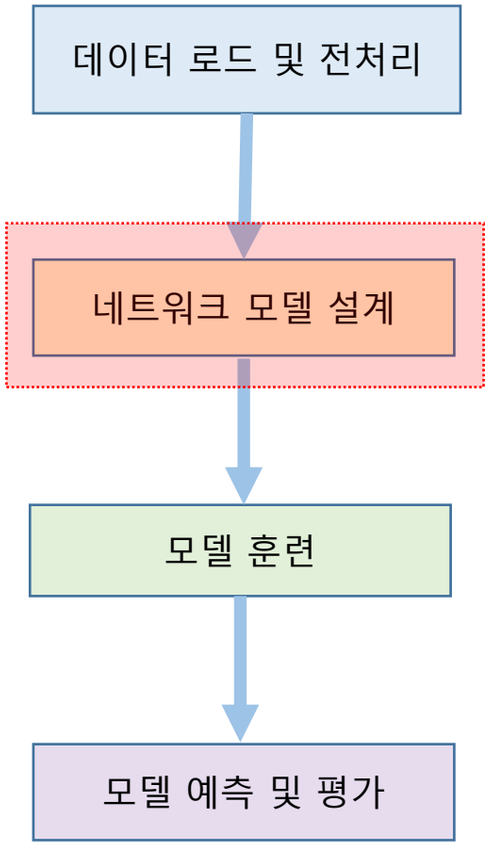
```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
```

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

model.summary()함수를 이용해서 설계된 모델의 레이아웃 출력

첫번째 layer는 512개의 노드를 가지는 Dense(fully Connected) layer이며,

두번째 layer는 10개의 노드를 가지는 Dense(fully Connected) layer

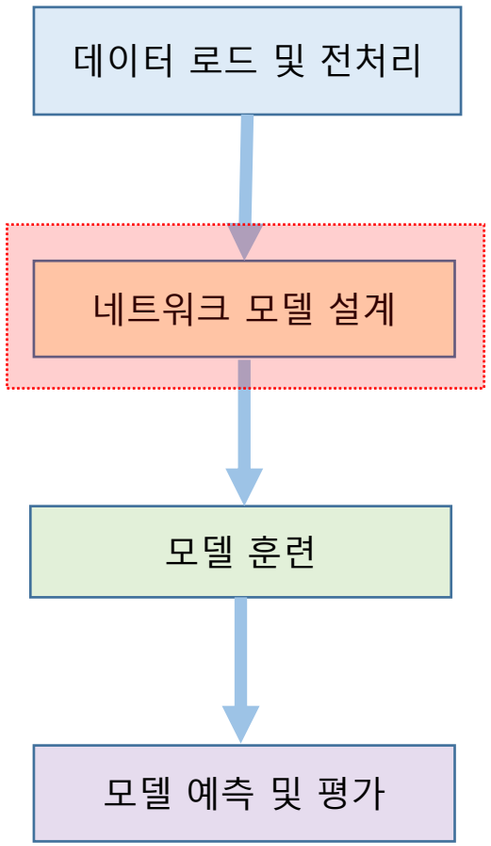


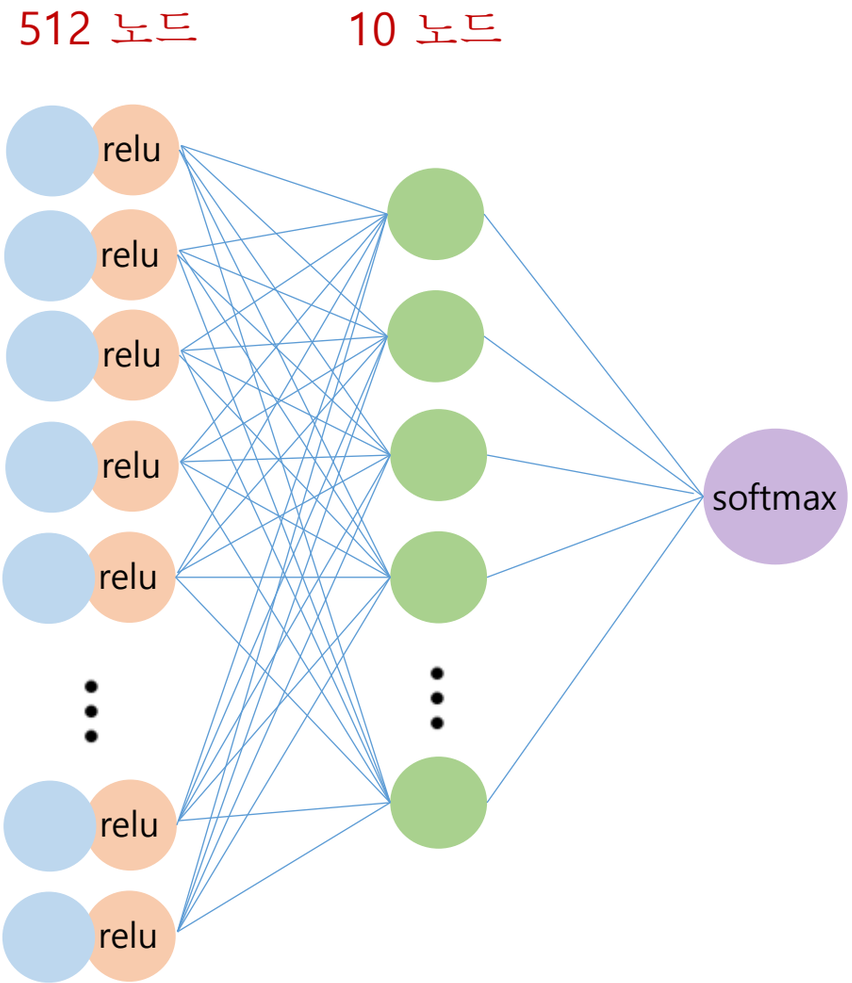

```
model.summary()
```

```
_________________________________________________________________
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
_________________________________________________________________
```

512개의 노드는 활성화(Activation) 함수로 relu를 사용하며,

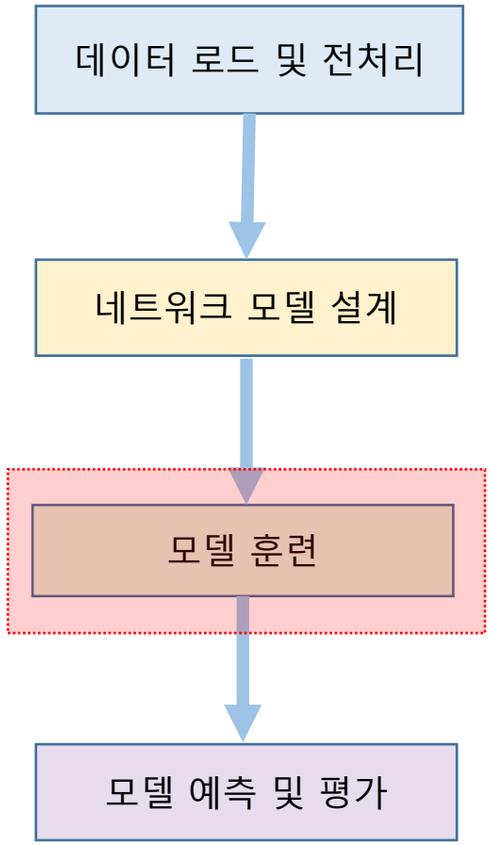
10개의 노드는 활성화 함수로 softmax를 사용하며 모델 네트워크 구성





epoch는 5(60000 * 5), batch_size는 64(이미지 64장 마다 가중치 업데이트)으로 학습 진행

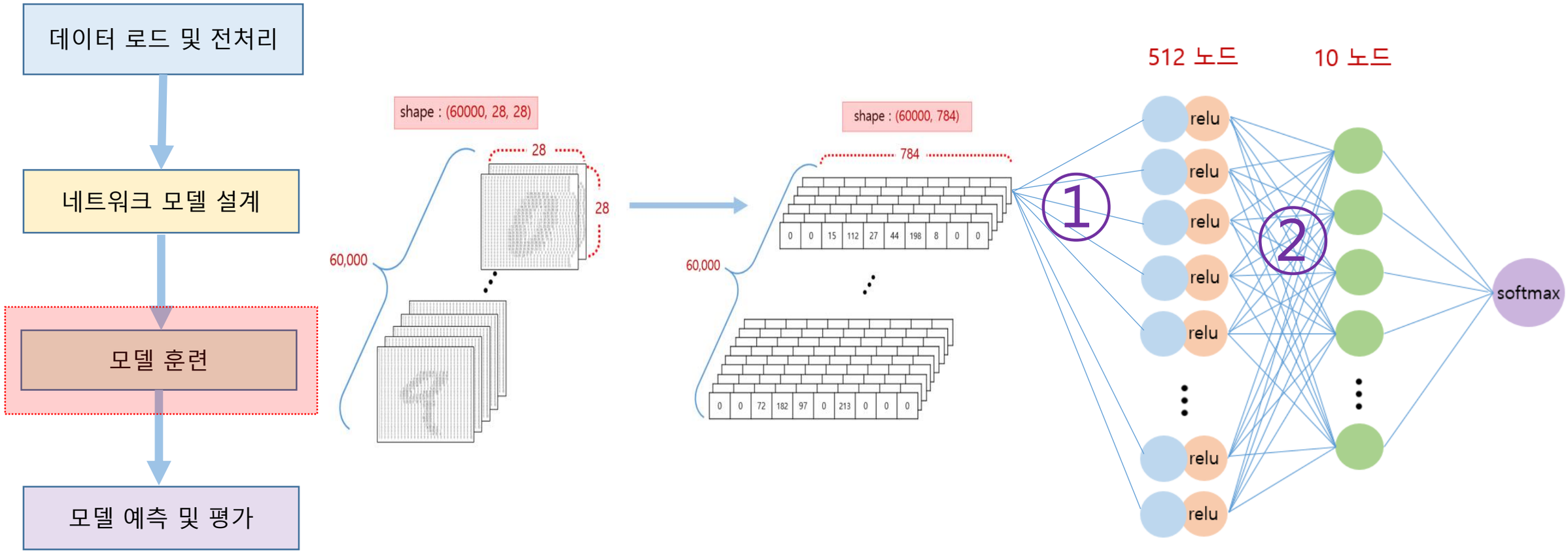
training 데이터 셋을 이용한 학습 결과는 정확도는 98.95%




```
: model.fit(train_images, train_labels, epochs=5, batch_size=64)

Epoch 1/5
60000/60000 [==============================] - 4s 73us/step - loss: 0.2227 - acc: 0.9353
Epoch 2/5
60000/60000 [==============================] - 2s 40us/step - loss: 0.0923 - acc: 0.9728
Epoch 3/5
60000/60000 [==============================] - 2s 40us/step - loss: 0.0625 - acc: 0.9809
Epoch 4/5
60000/60000 [==============================] - 2s 40us/step - loss: 0.0468 - acc: 0.9863
Epoch 5/5
60000/60000 [==============================] - 2s 39us/step - loss: 0.0357 - acc: 0.9895

: <keras.callbacks.History at 0x27a28bdd2e8>
```

①의 학습할 가중치 parameter의 갯수는 401920 = [784(이미지) +1(bias)] X 512(입력 노드 수)

②의 학습할 가중치 parameter의 갯수는 5130 = [512(이전 출력 노드 수)] +1(bias) X 10(입력 노드 수)



test 데이터 셋과 model.evaluate() 함수를 이용해서 모델의 예측 및 평가

모델의 예측 정확도는 97.95%

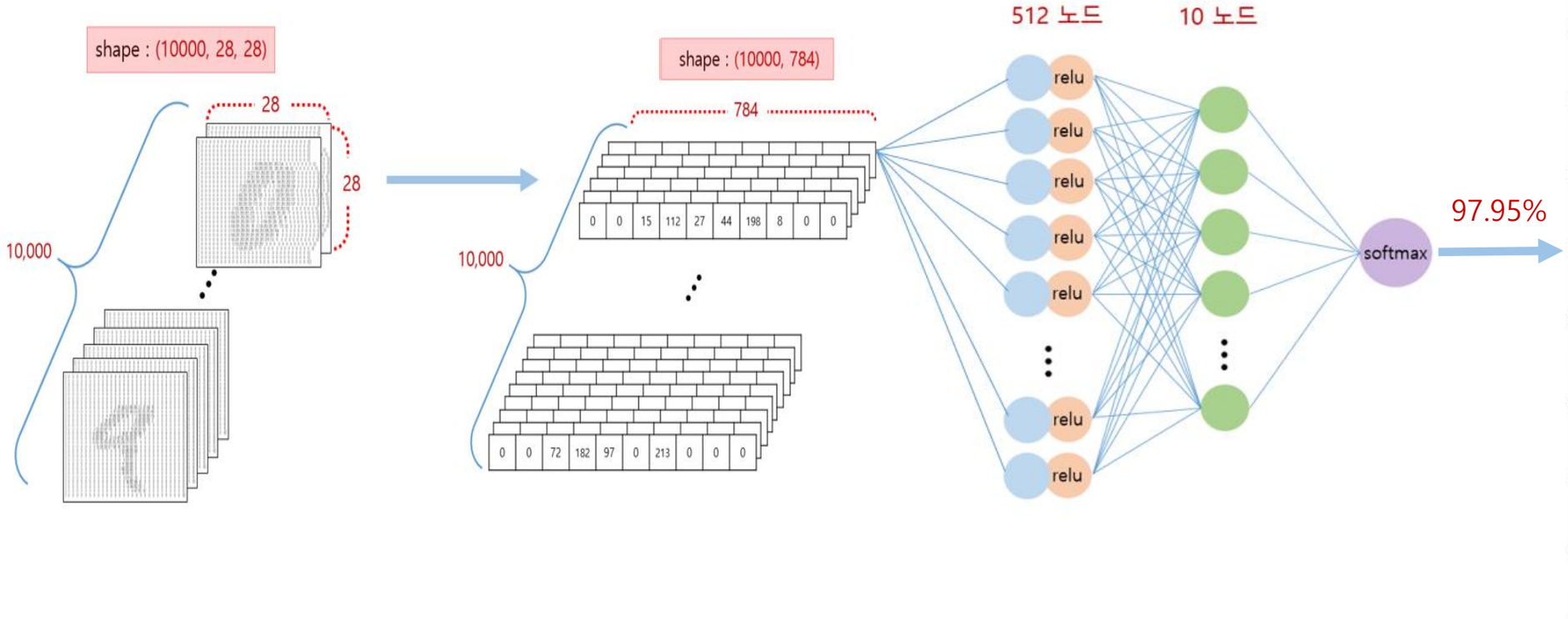
```
: test_loss, test_acc = model.evaluate(test_images, test_labels)
10000/10000 [==============================] - 0s 30us/step

: print('test_acc:', test_acc)
test_acc: 0.9795
```

데이터 로드 및 전처리

네트워크 모델 설계

모델 훈련

모델 예측 및 평가



네트워크 모델 설계

In [45]:
```
from keras import models
from keras import layers
```

In [46]:
```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
```

In [47]:
```
model.compile(optimizer
              loss='cat
              metrics=[
```

train_mnist.ipynb 실습

In [48]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

모델 훈련(training)

In [49]:
```
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```