

19강. DNN(Deep Neural Network)

- 네트워크 정의, 컴파일(compile), 학습(fit)
- 신경망의 행렬식 표현

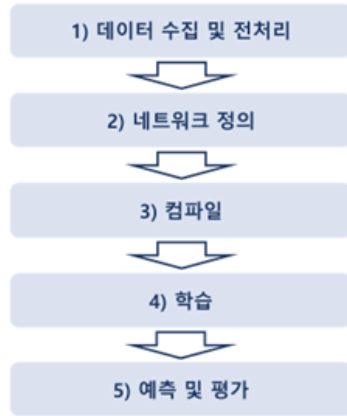
■ 딥러닝의 주요 모델



더블클릭 또는 Enter 키를 눌러 수정

DNN(Deep Neural Network)

신경망의 학습 절차



프로젝트 #1

주변 도시의 정보를 활용하여
특정 도시 주택 가격
예측하는 모델 만들기

DNN(Deep Neural Network)

프로젝트 #1



총 1000건의 데이터 수집 7개의 독립변수 종속변수

| 별墅발생률 | 방 개수 | 미세먼지 농도 | 나이 | 고속도로 접근 용이도 | 재산세율 | 학원 개수 | 배출 | 주택가격 |
|---------|------|---------|------|-------------|------|-------|----|-------|
| 0.00632 | 6.57 | 0.538 | 65.2 | 1 | 296 | 4.98 | | 24000 |
| 0.02731 | 6.42 | 0.469 | 78.8 | 2 | 242 | 9.14 | | 21600 |
| 1.23247 | 6.14 | 0.538 | 91.7 | 4 | 307 | 13.83 | | 19600 |

· 특성 선택, 추출, 제거 등의 특성 공학

· 이상치 및 결측치 처리, 정규화 등의 전처리

· 독립변수와 종속변수 구분

DNN(Deep Neural Network)

프로젝트 #1



총 1000건의 데이터 수집 7개의 독립변수 종속변수

| 범죄발생률 | 방 개수 | 미세먼지 농도 | 나이 | 고속도로 접근 용이도 | 재산세율 | 하위 계층 비율 | 주택가격 |
|---------|------|---------|------|-------------|------|----------|-------|
| 0.00632 | 6.57 | 0.538 | 65.2 | 1 | 296 | 4.98 | 24000 |
| 0.02731 | 6.42 | 0.469 | 78.8 | 2 | 242 | 9.14 | 21600 |
| 1.23247 | 6.14 | 0.538 | 91.7 | 4 | 307 | 13.83 | 19600 |

입력층, 은닉층, 출력층 구조에 대한 정의

Tensorflow의 Sequential(), add() 함수 등을 활용

· 입력층 구조 : 7개의 input

· 은닉층 구조

: 은닉층 수와 각 층의 노드 수 결정

→ 하이퍼파라미터로서 사용자가 설정

: 활성화함수 → 일반적인 모델이므로 RELU 적용

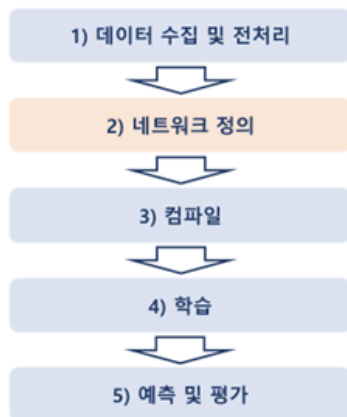
: 기타 Layer 추가 → Dropout, BN Layer 등의 추가

· 출력층 구조 : 1개의 Output

: 활성화함수 → 회귀 모델이므로 **항등함수** 적용

DNN(Deep Neural Network)

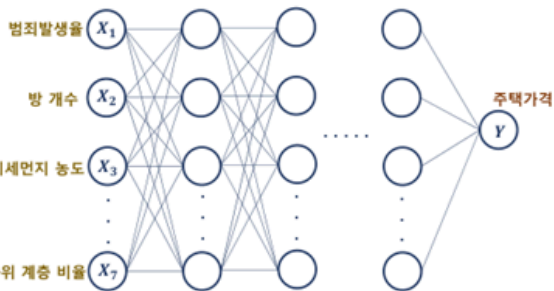
프로젝트 #1



총 1000건의 데이터 수집 7개의 독립변수 종속변수

| 범죄발생률 | 방 개수 | 미세먼지 농도 | 나이 | 고속도로 접근 용이도 | 재산세율 | 하위 계층 비율 | 주택가격 |
|---------|------|---------|------|-------------|------|----------|-------|
| 0.00632 | 6.57 | 0.538 | 65.2 | 1 | 296 | 4.98 | 24000 |
| 0.02731 | 6.42 | 0.469 | 78.8 | 2 | 242 | 9.14 | 21600 |
| 1.23247 | 6.14 | 0.538 | 91.7 | 4 | 307 | 13.83 | 19600 |

입력층, 은닉층, 출력층 구조에 대한 정의



DNN(Deep Neural Network)

프로젝트 #1



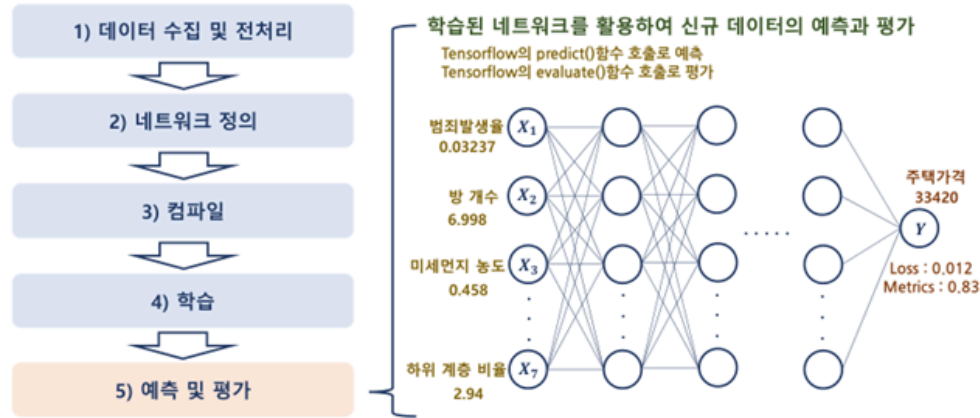
DNN(Deep Neural Network)

프로젝트 #1

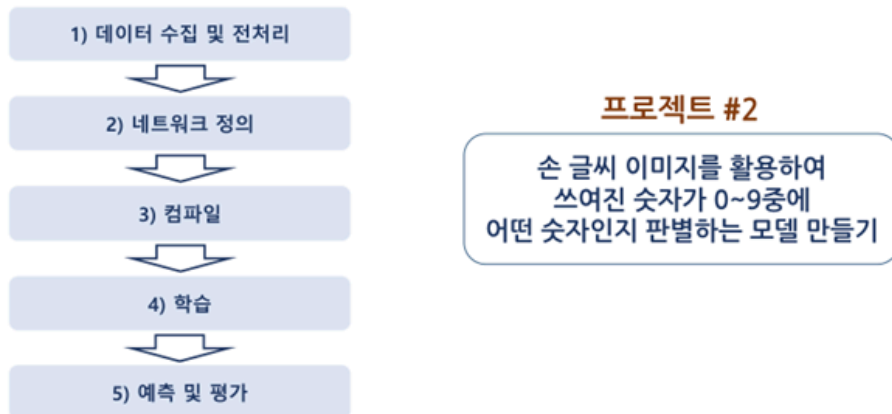


DNN(Deep Neural Network)

프로젝트 #1



DNN(Deep Neural Network)



DNN(Deep Neural Network)

프로젝트 #2



총 60,000건의 데이터 수집

28 X 28의 GrayScale 이미지

0 ~ 9까지 10개 클래스



※ 출처 : Mnist 손글씨 Dataset

· 이미지 데이터의 특성 파악

· Ground Truth에 대한 One-Hot Encoding

0 → [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
 1 → [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
 2 → [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
 ⋮
 9 → [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

- Ground Truth(정답 데이터)의 의미

- 현실 세계에서 수집된 정확하고 신뢰할 수 있는 데이터.
- 모델이나 알고리즘의 출력이 얼마나 정확한지 비교할 수 있는 기준값.
- 사람이 직접 라벨링하거나, 실측/실험 등을 통해 얻은 "정답" 역할.

DNN(Deep Neural Network)

프로젝트 #2

총 60,000건의 데이터 수집
28 X 28의 GrayScale 이미지 0 ~ 9까지 10개 클래스



입력층, 은닉층, 출력층 구조에 대한 정의

Tensorflow의 Sequential(), add() 함수 등을 활용

· 입력층 구조 : 28X28(784)개의 Input

· 은닉층 구조

: 은닉층 수와 각 층의 노드 수 결정

→ 하이퍼파라미터로서 사용자가 설정

: 활성화함수 → 일반적인 모델이므로 **ReLU** 적용

: 기타 Layer 추가 → Dropout, BN Layer 등의 추가

· 출력층 구조 : 10개의 Output

: 활성화함수 → 다중 분류 모델이므로 **소프트맥스** 적용

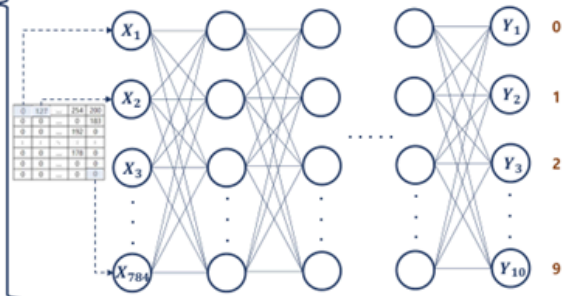
DNN(Deep Neural Network)

프로젝트 #2

총 60,000건의 데이터 수집
28 X 28의 GrayScale 이미지 0 ~ 9까지 10개 클래스

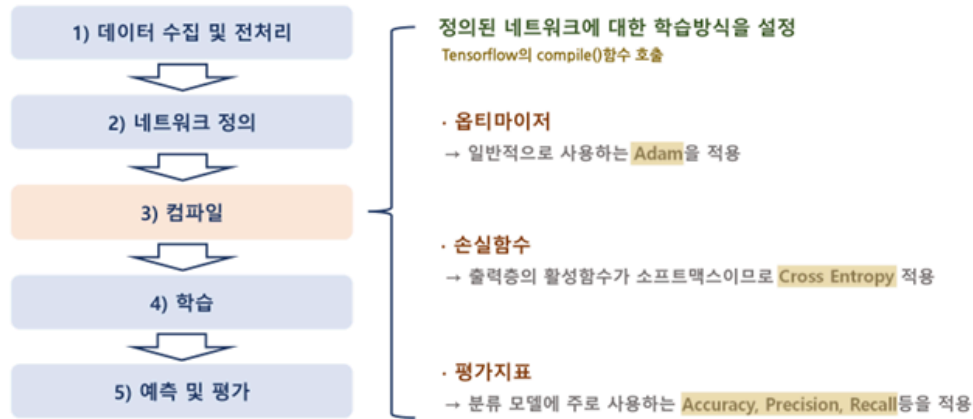


입력층, 은닉층, 출력층 구조에 대한 정의



DNN(Deep Neural Network)

프로젝트 #2



DNN(Deep Neural Network)

프로젝트 #2

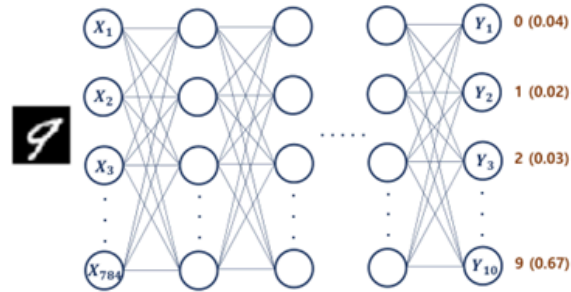


DNN(Deep Neural Network)

프로젝트 #2



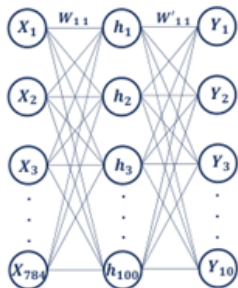
학습된 네트워크를 활용하여 신규 데이터의 예측과 평가
 Tensorflow의 predict() 함수 호출로 예측
 Tensorflow의 evaluate() 함수 호출로 평가



DNN(Deep Neural Network)

프로젝트 #2

은닉층의 깊이 : 1
 은닉층의 뉴런 개수 : 100개



$$h_1 = W_{11}X_1 + W_{12}X_2 + \dots + W_{1783}X_{783} + W_{1784}X_{784} + \beta_1$$

$$h_2 = W_{21}X_1 + W_{22}X_2 + \dots + W_{2783}X_{783} + W_{2784}X_{784} + \beta_2$$

$$\vdots$$

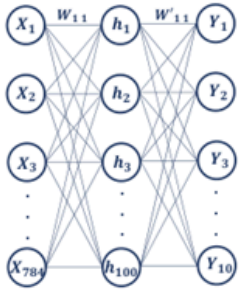
$$h_{100} = W_{1001}X_1 + W_{1002}X_2 + \dots + W_{100783}X_{783} + W_{100784}X_{784} + \beta_{100}$$



DNN(Deep Neural Network)

프로젝트 #2

은닉층의 깊이: 1
은닉층의 뉴런 개수: 100개



$$h_1 = W_{11}X_1 + W_{12}X_2 + \dots + W_{1783}X_{783} + W_{1784}X_{784} + \beta_1$$

$$h_2 = W_{21}X_1 + W_{22}X_2 + \dots + W_{2783}X_{783} + W_{2784}X_{784} + \beta_2$$

$$\vdots$$

$$h_{100} = W_{1001}X_1 + W_{1002}X_2 + \dots + W_{100783}X_{783} + W_{100784}X_{784} + \beta_{100}$$

행렬식 계산

입력층 은닉층

$$\begin{pmatrix} h_1 \\ \vdots \\ h_{100} \end{pmatrix} = \begin{pmatrix} W_{11} & \dots & W_{1784} \\ \vdots & \ddots & \vdots \\ W_{1001} & \dots & W_{100784} \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_{783} \\ X_{784} \end{pmatrix} + \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_{100} \end{pmatrix}$$

은닉층 출력층

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_{10} \end{pmatrix} = \begin{pmatrix} W'_{11} & \dots & W'_{1100} \\ \vdots & \ddots & \vdots \\ W'_{101} & \dots & W'_{10100} \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_{99} \\ h_{100} \end{pmatrix} + \begin{pmatrix} \beta'_1 \\ \vdots \\ \beta'_{10} \end{pmatrix}$$

딥러닝의 주요 모델

DNN(Deep Neural Network)

다수의 은닉층으로 구성된 기본적인 신경망



이미지 데이터에서
각 픽셀의 인접한 관계에 대한 설명 불가

CNN(Convolutional Neural Network)

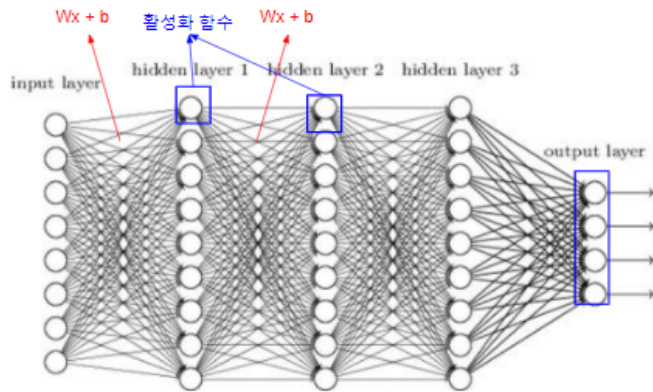
영상 처리에 활용되는 합성곱을 이용한 신경망

RNN(Recurrent Neural Network)

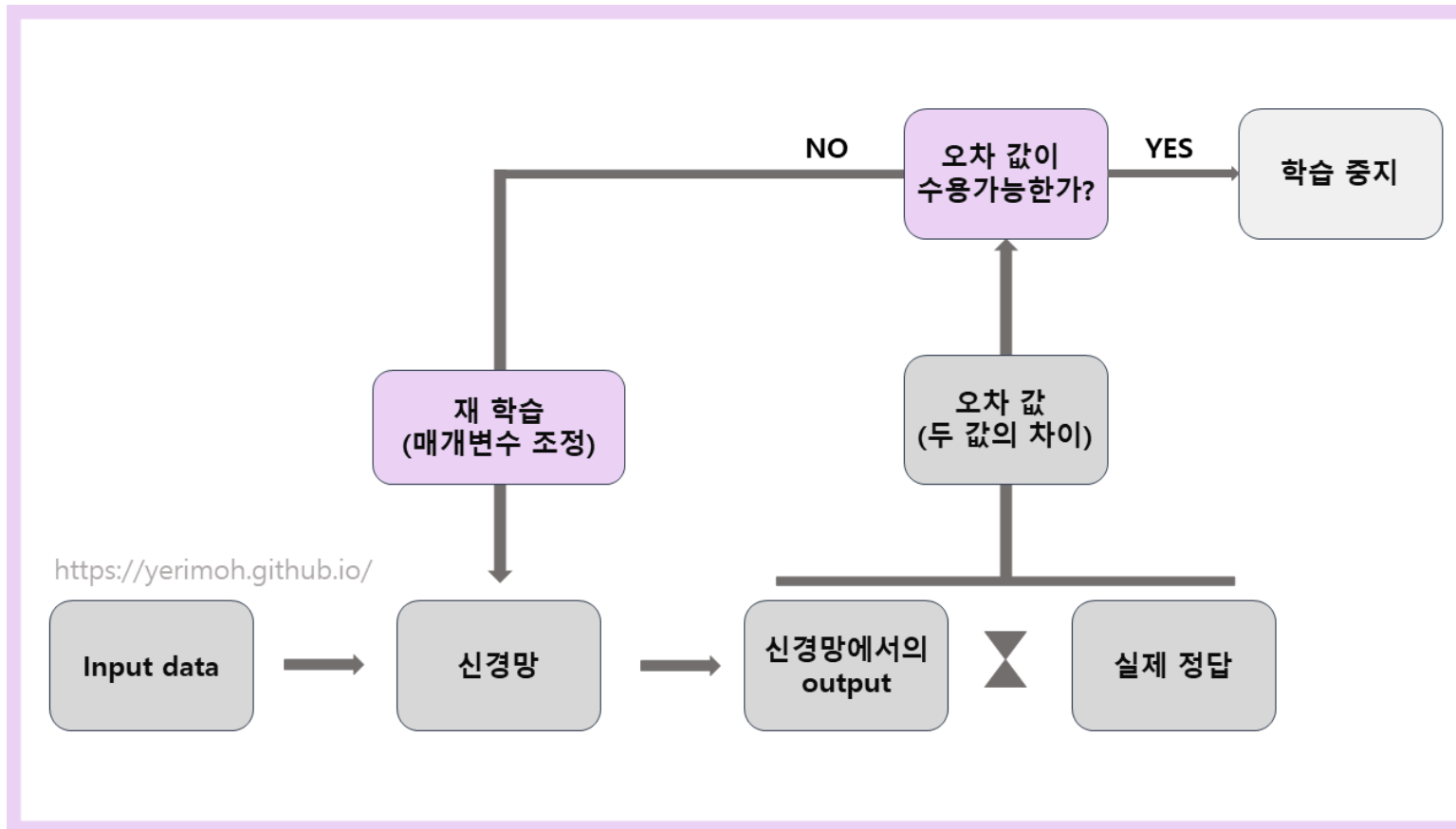
Auto Encoder

▽ DNN(Deep Neural Network)이란?

- 입력층(input layer)과 출력층(output layer) 사이에 여러 개의 은닉층(hidden layer)들로 이뤄진 인공신경망(Artificial Neural Network, ANN)
 - input에 대해서 output까지 Layer를 거치는데 input Layer 다음에 W 와 b 가 있어서 $Wx+b$ 를 거치게 됨
 - hidden layer1에서는 활성화 함수가 있음
 - 그 후 다시 $Wx + b$ 를 거치고 활성화 함수를 거친 후, output layer가 만약 분류였다면 마지막에는 softmax를 activation function(활성화 함수)을 활용함



▽ 딥러닝 학습과정



✓ DNN 실습

✓ 다양한 작업에 대해 데이터 흐름 프로그래밍을 위한 오픈소스 소프트웨어 라이브러리 import

- 텐서플로는 파이썬 객체를 텐서(Tensor)로 변환해서 행렬 연산을 수행

```
1 import tensorflow as tf
```

✓ 데이터 읽기

• Tensorflow에서 MNIST를 제공

- MNIST는 인공지능 연구의 권위자 LeCun교수가 만든 데이터 셋이고 현재 딥러닝을 공부할 때 반드시 거쳐야할 Hello, World 같은 존재
- MNIST는 60,000개의 트레이닝 셋과 10,000개의 테스트 셋으로 이루어져 있고 이중 트레이닝 셋을 학습데이터로 사용하고 테스트 셋을 신경망을 검증하는 데에 사용



```
1 # data download
2 # load_data() 함수는 MNIST 데이터를 학습용(train) 과 테스트용(test) 으로 자동 다운로드 및 로딩
3 (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

⏮ Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 ————— 2s 0us/step

✓ train : 60000장, test : 10000장, 데이터가 28*28 크기로 다운로드 된것을 확인

```
1 print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

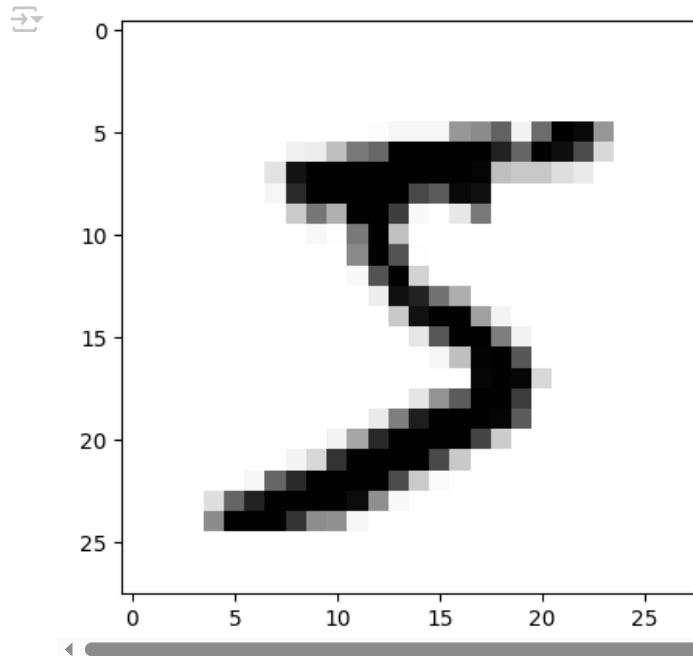
⏮ (60000, 28, 28) (60000,) (10000, 28, 28) (10000,)

```
1 print("학습셋 이미지 수 : %d 개" % (x_train.shape[0]))
2 print("테스트셋 이미지 수 : %d 개" % (x_test.shape[0]))
```

⇒ 학습셋 이미지 수 : 60000 개
테스트셋 이미지 수 : 10000 개

▼ x_train 데이터의 첫번째 원소 확인하기 - 이미지 출력


```
1 import sys # Python의 표준 라이브러리인 sys 모듈을 불러오는 구문 - 이 모듈은 시스템과 관련된 다양한 기능을 제공
2
3 # 이미지 형태 출력을 위한 pyplot 모듈 import
4 import matplotlib.pyplot as plt
5
6 # 위 60000개 데이터 배열에서 0번째 원소를 흑백으로 출력
7 # cmap은 "color map"의 줄임말로, 데이터를 시각화할 때 숫자 값들을 색상으로 표현해주는 역할
8 # cmap='Greys'의 의미 - 'Greys'는 회색조(그레이스케일) 컬러맵
9 plt.imshow(x_train[0], cmap='Greys')
10 plt.show()
```



 ndarray (28, 28) [show data](#)


▼ x_train 데이터의 첫번째 원소 확인하기 - 픽셀값 출력

```
1 # 0번째 원소의 모든 데이터를 출력
2 for x in x_train[0]:
3     for i in x:
4         sys.stdout.write('%d ' % i) #변수 i를 포맷팅된 문자열 형태로 줄 바꿈 없이 출력할 때 사용
5     sys.stdout.write('\n')
```



```
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 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 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 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 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 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 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 49 238 253 253 253 253 253 253 253 251 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 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 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 0 139 253 190 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 11 190 253 70 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 81 240 253 253 119 25 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 45 186 253 253 150 27 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 16 93 252 253 187 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 249 253 249 64 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 46 130 183 253 253 207 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 39 148 229 253 253 253 250 182 0 0 0 0 0 0
0 0 0 0 0 0 0 24 114 221 253 253 253 253 201 78 0 0 0 0 0
0 0 0 0 0 0 0 23 66 213 253 253 253 253 198 81 2 0 0 0 0
0 0 0 0 0 0 18 171 219 253 253 253 253 195 80 9 0 0 0
0 0 0 0 55 172 226 253 253 253 253 244 133 11 0 0 0
0 0 0 0 136 253 253 253 212 135 132 16 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
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 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

▼ 데이터 정규화

- 여기서 x_train, x_test에 255를 나누는 이유는 사진을 읽어오면 컬러 기준으로 (R, G, B)로 최대 (255, 255, 255) 형태로 가져오는데 여기에 255를 나눠 0~1의 값을 가지게 만들어 데이터의 분산의 정도를 바꾸는, 즉 데이터 정규화 과정을 거치게 하기 위해서 임

```
1 x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
1 #실제 들어있는 값 확인 (x_train, y_train)  
2 x_train[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.      , 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.      , 0.      ,
0.      , 0.      , 0.      , 0.      , 0.      ,
0.      , 0.      , 0.      , 0.      , 0.      ,
0.      , 0.      , 0.      , 0.      , 0.      ,
0.      , 0.      , 0.      , ]]

```

✓ 이미지 '5'에 대한 목표값(레이블) 숫자 '5'를 출력

```

1 # np.uint8(5)는 NumPy 라이브러리에서 8비트 부호 없는 정수(unsigned 8-bit integer) 자료형을 사용하여 값 5를 나타내는 객체를 생성하는 명령
2 # 즉, 5라는 정수를 8비트 범위 내에서 저장할 수 있도록 변환하는 것!!!
3 y_train[0]

```

⇒ np.uint8(5)

✓ 모델 만들기 - Sequential Model(연속적인 입력으로부터 연속적인 출력을 생성하는 모델)

- [tf.keras.models.Sequential](#)

- Sequential한 모델을 하나 만들고 layer를 쌓음(함수를 통해 뉴럴네트워크 레이어를 생성할 수 있으며 이 함수에 포함된 레이어들로 모델을 구성해줌)

- [tf.keras.layers.Flatten\(input_shape=\(28, 28\)\)](#)

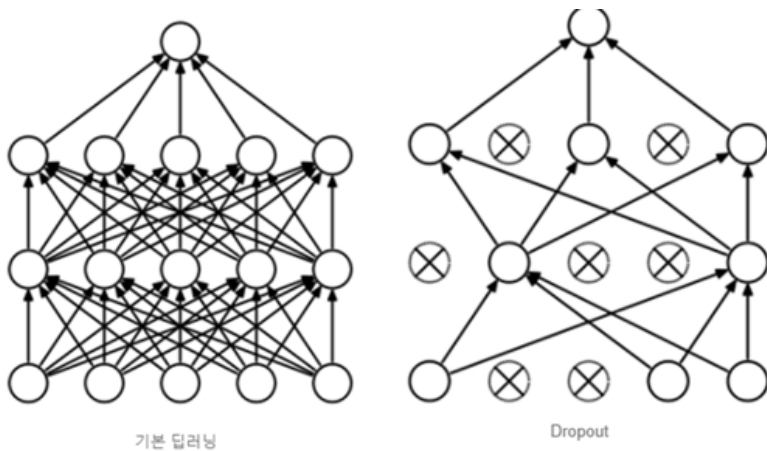
- 입력 레이어를 평평하게 만들어 주는 함수(다차원 배열을 일차원 배열로 만들어 줌, 28 x 28 배열을 입력으로 받아 1차원 배열로 변환해줌)

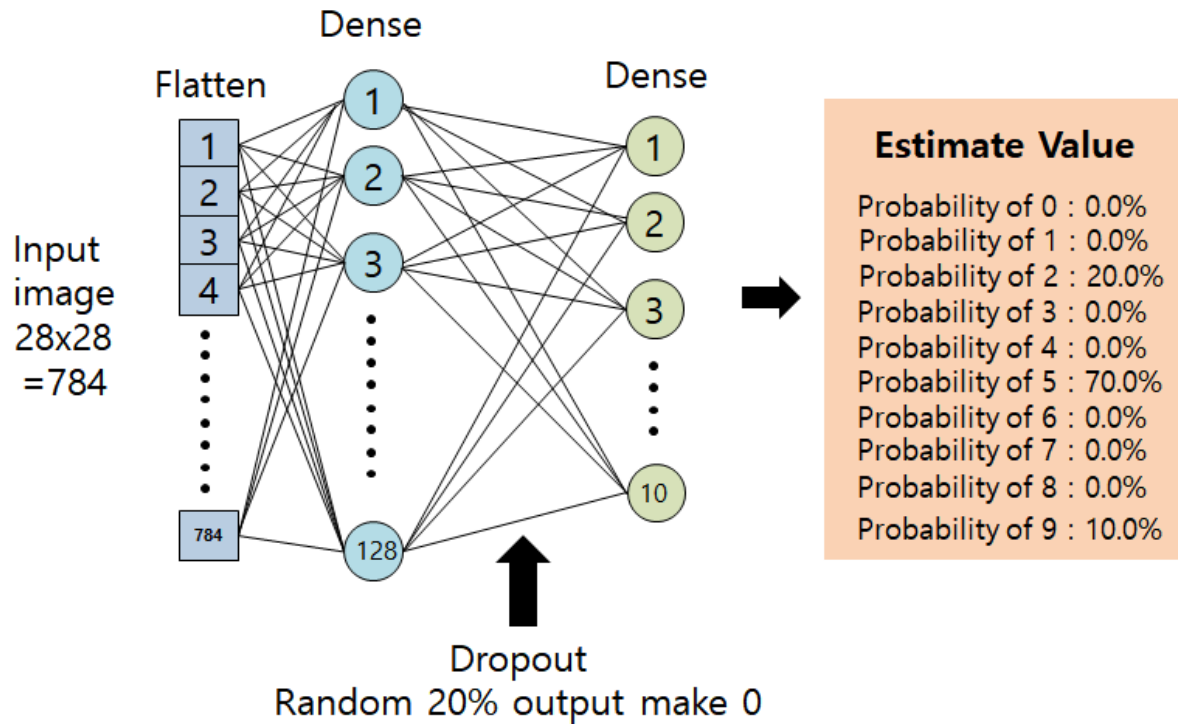
- [tf.keras.layers.Dense\(128, activation='relu'\)](#)

- 히든 레이어의 노드개수는 128개, 활성화 함수는 relu임(이 레이어에서 입력은 가중치 값과 곱해지고 더해지며 학습이 진행되며 이 가중치 값이 수정되어 지며 정답을 찾아감)
- [tf.keras.layers.Dropout\(0.2\)](#)
 - Dropout은 일부 노드를 끊어줌
 - 일부 노드를 끊어주면 Overfitting될 확률이 낮아짐
 - Dropout도 epochs 돌때마다 비활성화되는 노드들이 바뀌기 때문에 Dropout을 적용하게 되면 머신러닝에서 나왔던 Overfitting 과적합 문제가 많이 줄어듬
 - 오버피팅 방지한 것으로 이전 레이어의 출력을 20% 정도를 0으로 만들
- [tf.keras.layers.Dense\(10, activation='softmax'\)](#)
 - 숫자는 0부터 9까지 총 10개의 라벨값을 가지므로 모델의 출력은 10개임(활성화 함수는 softmax로 다중 클래스 분류 문제의 출력층에서 주로 쓰이는 모델임)

```
1 model = tf.keras.models.Sequential([
2     tf.keras.layers.Flatten(input_shape=(28,28)),
3     tf.keras.layers.Dense(128, activation='relu'),
4     tf.keras.layers.Dropout(0.2),
5     tf.keras.layers.Dense(10, activation='softmax')
6 ])
```

⚠ /usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential mode `super().__init__(**kwargs)`





✓ 모델의 파라미터 개수 확인

- `model.summary()` 메서드 호출을 통해 모델을 요약해서 layer마다 shape와 같은 정보들을 볼 수 있음

- flatten layer - 추출된 주요 특징을 전결합층에 전달하기 위해 1차원 자료로 바꿔주는 layer ==> $28 * 28 = 784$
- dense layer - 다층 퍼셉트론 신경망에서 사용되는 레이어로 입력과 출력을 모두 연결해줌
- 첫번째 dense 입력 개수 = $(784) * W(128) + b(128) = 100480$
 - shape 하나당 b가 하나 있기 때문에 128을 더해줌
- 두번째 dense 입력 개수 = $(128) * W(10) + b(10) = 1290$

- 전체 파라미터의 수 = $100480 + 1290 = 101770$

```
1 model.summary()
```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|-------------------|--------------|---------|
| flatten (Flatten) | (None, 784) | 0 |
| dense (Dense) | (None, 128) | 100,480 |
| dropout (Dropout) | (None, 128) | 0 |
| dense_1 (Dense) | (None, 10) | 1,290 |

Total params: 101,770 (397.54 KB)

Trainable params: 101,770 (397.54 KB)

✓ 모델을 컴파일(model.compile) - 손실, 메트릭 및 최적화 프로그램 지정하기

- 모델을 학습할 때 어떤 방식을 사용할지 정해주는 함수

- 손실함수 loss = 'sparse_categorical_crossentropy'

- cross entropy인데 분류를 여러 개로 할 때 사용

- 혼동행렬 metrics=['accuracy']

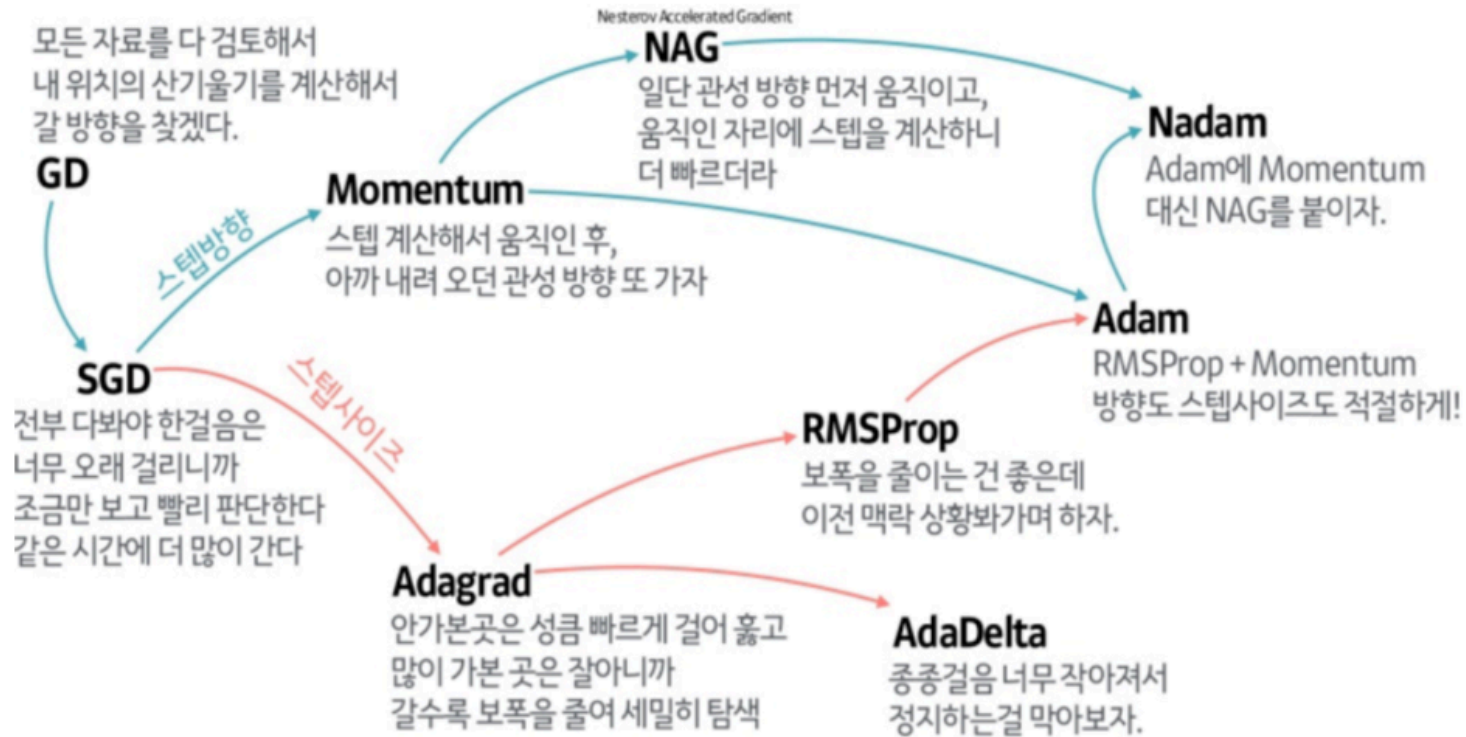
- 평가 지표를 설정
- 정확도를 보겠다는 것(다중클래스분류 문제에서 평가기준을 'accuracy'로 지정했을 경우 내부적으로 categorical_accuracy() 함수를 이용하여 정확도가 계산)
- 학습 중 모델의 성능을 평가할 때 사용
- 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 점수 등이 있음

- Optimizer(최적화)

- 최소값을 찾아가는 것(이를 수행하는 알고리즘이 최적화 알고리즘) [링크 텍스트](#)
- 학습속도를 빠르고 안정적이게하는 것을 목표로 함

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

Optimizer 종류



```

1 from IPython.display import HTML
2 from base64 import b64encode
3 mp4 = open('/content/drive/MyDrive/머신러닝&딥러닝/딥러닝-19-Deep Neural Networks (DNN)/optimizer.mp4', 'rb').read()
4 data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
5 HTML("""
6 <video width=500 controls>
7     <source src="%s" type="video/mp4">
8 </video>
9 """) % data_url)

```



1.00

0:20 / 0:20

혼동 행렬(Confusion Matrix)란 무엇일까요??

- 혼동 행렬 또는 오차 행렬이라 불리는 Confusion Matrix는 분류 모델의 평가지표로 많이 쓰입니다.
- 혼동 행렬은 각 데이터의 실제 클래스와 예측된 클래스의 개수를 조건에 맞게 행렬 성분에 채워 넣은 것입니다.

| | | 예 측 클 래 스 | |
|-----------------------|----------|------------------------|------------------------|
| | | Positive | Negative |
| 실 제 클 래 스 | Positive | TP (True Positive) | FN (False Negative) |
| | Negative | FP (False Positive) | TN (True Negative) |

✓ 학습

- `fit()` 함수를 호출할 때 추가적으로 넣을 수 있는 게 많음(다음과 같은 매개변수를 받음)

- `x`: 모델의 입력 데이터를 나타내는 Numpy 배열 또는 Numpy 배열의 리스트
- `y`: 모델의 정답 데이터를 나타내는 Numpy 배열 또는 Numpy 배열의 리스트
- `batch_size`: 한 번에 처리되는 샘플의 수를 나타내는 정수 값. 기본값은 32입니다.
- `epochs`: 모델이 학습할 총 횟수를 나타내는 정수 값(기본값은 1임)
- `verbose`: 학습 과정을 어떻게 출력할 것인지를 결정하는 값. 0, 1, 2 중 하나의 값을 가질 수 있으며, 0일 경우 출력이 없고, 1일 경우 진행 막대(progress bar)가 표시되고, 2일 경우 에포크마다 한 줄씩 출력됨(기본값은 1임)
- `validation_data`: 검증용 데이터를 나타내는 튜플. (x_val, y_val) 형태로 입력하며, 기본값은 None임
- `callbacks`: 훈련 중에 호출되는 콜백 함수의 리스트를 지정하는 인수(콜백(callback) 함수는 훈련 중간에 모델의 상태를 확인하거나, 모델의 가중치를 저장하는 등의 역할을 수행할 수 있음)
- `shuffle`: 샘플을 학습할 때마다 데이터를 무작위로 섞을지 여부를 결정하는 값. 기본값은 True임

```
1 model.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 ————— 8s 3ms/step - accuracy: 0.8652 - loss: 0.4698
Epoch 2/10
1875/1875 ————— 7s 2ms/step - accuracy: 0.9554 - loss: 0.1539
Epoch 3/10
1875/1875 ————— 5s 3ms/step - accuracy: 0.9666 - loss: 0.1121
Epoch 4/10
1875/1875 ————— 5s 2ms/step - accuracy: 0.9727 - loss: 0.0886
Epoch 5/10
1875/1875 ————— 4s 2ms/step - accuracy: 0.9761 - loss: 0.0750
Epoch 6/10
1875/1875 ————— 6s 3ms/step - accuracy: 0.9792 - loss: 0.0646
Epoch 7/10
1875/1875 ————— 4s 2ms/step - accuracy: 0.9815 - loss: 0.0568
Epoch 8/10
1875/1875 ————— 5s 3ms/step - accuracy: 0.9837 - loss: 0.0499
Epoch 9/10
1875/1875 ————— 6s 3ms/step - accuracy: 0.9859 - loss: 0.0442
Epoch 10/10
1875/1875 ————— 5s 3ms/step - accuracy: 0.9870 - loss: 0.0401
<keras.src.callbacks.history.History at 0x7fae30f79750>
```

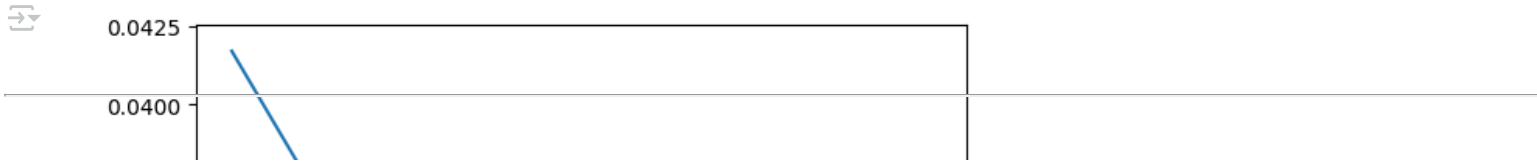
▽ 평가

- test 데이터가 따로 있기 때문에 test 데이터를 넣어서 몇 점이 나오는지 출력
- verbose - Keras의 Model이라는 패키지에서 제공하는 fit() 함수의 argument 중에 verbose가 있음
 - verbose - Integer 0, 1, or 2(보통 0 은 출력하지 않고, 1은 자세히, 2는 함축적인 정보만 출력하는 형태로 되어 있음).
- loss - 예측값과 실제값이 차이나는 정도를 나타내는 지표(작을 수록 좋음)

```
1 score = model.evaluate(x_test, y_test, verbose=1)
2 print('정답률 = ', score[1], 'loss=', score[0])
3 history = model.fit(x_train, y_train, epochs=10, verbose=0)
```

```
⇒ 313/313 _____ 1s 3ms/step - accuracy: 0.9793 - loss: 0.0738
정답률 = 0.9817000031471252 loss= 0.05985850840806961
```

```
1 import matplotlib.pyplot as plt
2
3 plt.plot(history.history['loss'])
4 plt.xlabel('epoch')
5 plt.ylabel('loss')
6 plt.show()
```

```
1 plt.plot(history.history['accuracy'])  
2 plt.xlabel('epoch')  
3 plt.ylabel('accuracy')  
4 plt.show()
```

