

2. 신경망 기본 구성 요 소

2.1. 신경망 모델 기초 훈련법

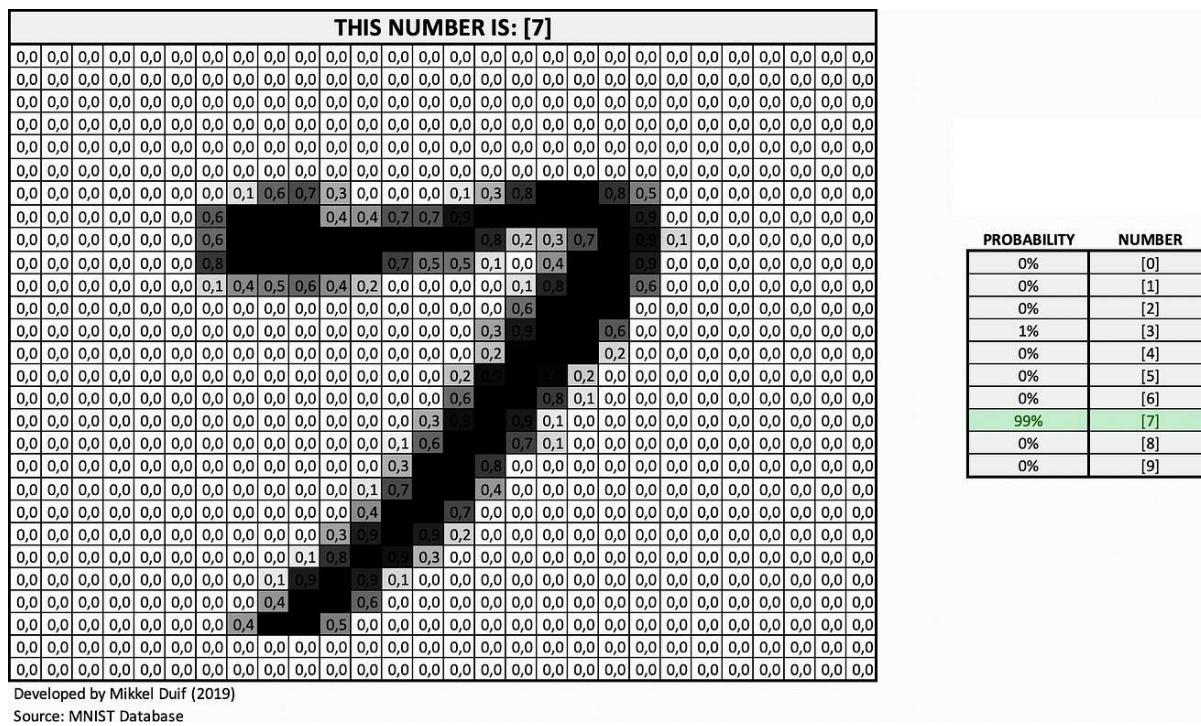
케라스 라이브러리를 이용하여 MNIST 손글씨 데이터셋을 대상으로 분류를 학습하는 신경망 모델을 구성, 훈련, 활용하는 방법을 소개

2.1. 훈련셋 준비: MNIST 데이터셋

```
from tensorflow.keras.datasets import mnist  
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

- 손글씨 숫자 인식 용도 데이터셋. 28x28 픽셀 크기의 사진 70,000개의 샘플로 구성
라벨: 0부터 9까지 10개의 클래스 중 하나
- 훈련셋: 샘플 60,000개 (모델 훈련용)
 - `train_images`
 - `train_labels`
- 테스트셋: 샘플 10,000개 (훈련된 모델 성능 테스트용)
 - `test_images`
 - `test_labels`

하나의 샘플



샘플, 타깃, 라벨, 예측값, 클래스

- 샘플
- 타깃과 라벨
- 예측값
- 클래스(범주)

2.1.2. 신경망 모델 지정

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

2.1.3. 신경망 모델 컴파일

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

2.1.4. 데이터 전처리

- 머신러닝 모델에 따라 입력값이 적절한 형식을 갖춰야 함
- 앞서 두 개의 `Dense` 층과 `Sequential` 클래스로 지정된 모델의 입력값은 1차원 어레이 형식을 갖춰야 함.

```
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
```

2.1.5. 모델 훈련

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

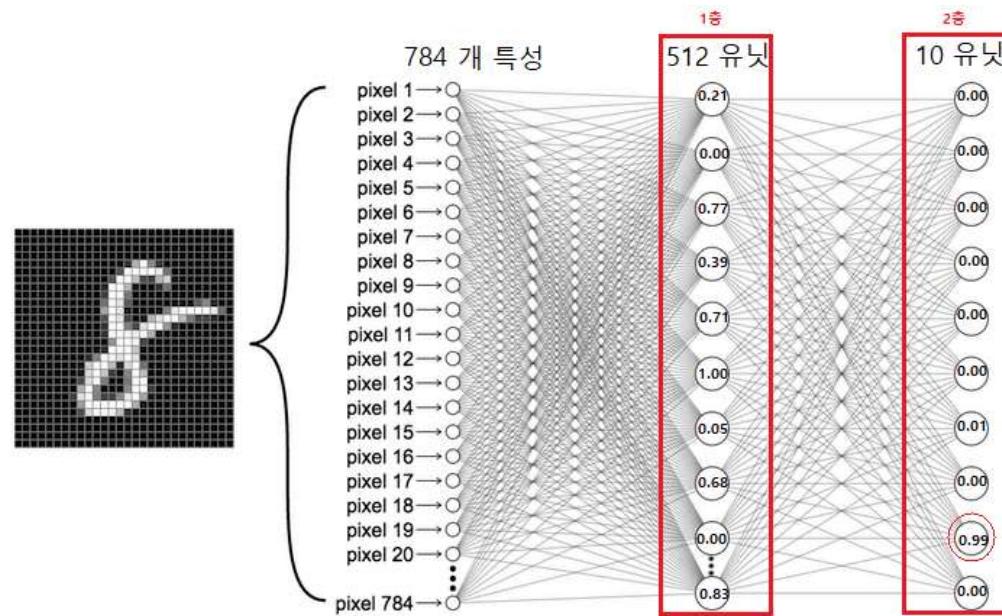
- 첫째 인자: 훈련 데이터셋
- 둘째 인자: 훈련 라벨셋
- epochs: 에포크. 전체 훈련 세트 대상 반복 훈련 횟수.
- batch_size: 배치 크기. 배치 크기만큼의 훈련 데이터셋으로 훈련할 때마다 가중치 업데이트.

모델의 훈련 과정

- 에포크가 끝날 때마다 평균 손실값과 평균 정확도 출력

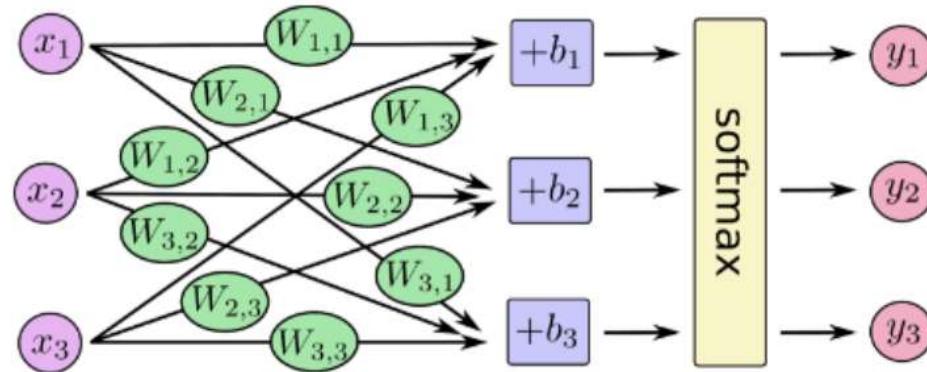
```
Epoch 1/5
469/469 [=====] - 5s 4ms/step - loss: 0.2551 - accuracy: 0.9263
Epoch 2/5
469/469 [=====] - 2s 4ms/step - loss: 0.1044 - accuracy: 0.9693
Epoch 3/5
469/469 [=====] - 2s 3ms/step - loss: 0.0683 - accuracy: 0.9793
Epoch 4/5
469/469 [=====] - 2s 4ms/step - loss: 0.0504 - accuracy: 0.9847
Epoch 5/5
469/469 [=====] - 2s 3ms/step - loss: 0.0378 - accuracy: 0.9885
```

모델 예측값 계산 과정



가중치 행렬과 출력값 계산 상세

- 층과 층 사이에서 이뤄지는 데이터 변환
- 유닛과 유닛 사이의 모든 선에 가중치 관여



행렬 연산

- 가중치 행렬과 입력값의 행렬 연산

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_1 + W_{1,3}x_1 + b_1 \\ W_{2,1}x_2 + W_{2,2}x_2 + W_{2,3}x_2 + b_2 \\ W_{3,1}x_3 + W_{3,2}x_3 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

2.2. 신경망 모델 훈련의 핵심 요소

배치 크기, 스텝, 에포크

- **배치 크기**: 훈련셋의 샘플을 일정 개수로 묶기 위해 지정된 양의 정수
- **스텝**_{step}: 하나의 배치(묶음)에 대해 훈련하는 과정
 - 스텝이 끝날 때마다 사용된 배치 묶음에 대한 손실값 계산 후 가중치 조정
- **에포크**_{epoch}: 훈련셋 전체의 모든 스텝을 한 번 실행하는 과정
- MNIST 데이터셋 예제
 - 배치 크기(batch_size)가 128이기에 총 6만개의 훈련 샘플을 128개씩 묶음
 - 따라서 $469(60,000/128 = 468.75)$ 개의 배치 생성
 - 하나의 에포크 동안 총 469번의 스텝이 실행

2.2.1. 훈련 스텝과 에포크

```
Epoch 1/5
469/469 [=====] - 5s 4ms/step - loss: 0.2551 - accuracy: 0.9263
Epoch 2/5
469/469 [=====] - 2s 4ms/step - loss: 0.1044 - accuracy: 0.9693
Epoch 3/5
469/469 [=====] - 2s 3ms/step - loss: 0.0683 - accuracy: 0.9793
Epoch 4/5
469/469 [=====] - 2s 4ms/step - loss: 0.0504 - accuracy: 0.9847
Epoch 5/5
469/469 [=====] - 2s 3ms/step - loss: 0.0378 - accuracy: 0.9885
```

배치 크기

- 훈련셋의 샘플을 일정 개수로 묶기 위해 지정된 양의 정수
- 예제:
 - 배치 크기: 128
 - MNIST 손글씨 이미지를 분류하는 모델은 128x784 모양의 2차원 어레이를 입력값으로 사용

훈련 스텝과 경사하강법

- 배치 단위로 입력 데이터를 변환함
- 예제: 마지막 층에서의 변환

```
softmax(W X + b)
```

- 훈련 스텝: 배치 단위로 예측 결과의 오차를 이용하여 가중치 행렬 W 와 편향 벡터 b 조정
- 경사하강법 활용됨

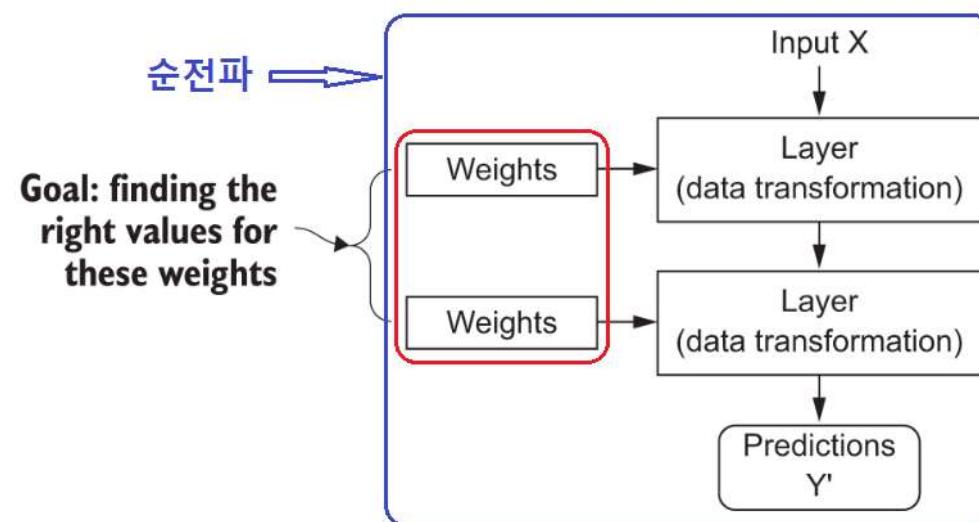
에포크

- 전체 훈련셋을 지정된 크기의 배치 묶음으로 나눈 후 각각의 배치에 대한 모든 스텝을 한 번 완료하는 과정
- 입력 데이터를 배치 단위로 여러 층을 통과시키면서 변환시키는 과정을 최적의 예측값을 만들 때까지 여러 에포크를 거치면서 진행
- MNIST 데이터셋 예제
 - 배치 크기(`batch_size`)가 128이기에 총 6만개의 훈련 샘플을 128개씩 묶음
 - 따라서 $469(60,000/128 = 468.75)$ 개의 배치 생성
 - 하나의 에포크 동안 총 469번의 스텝이 실행

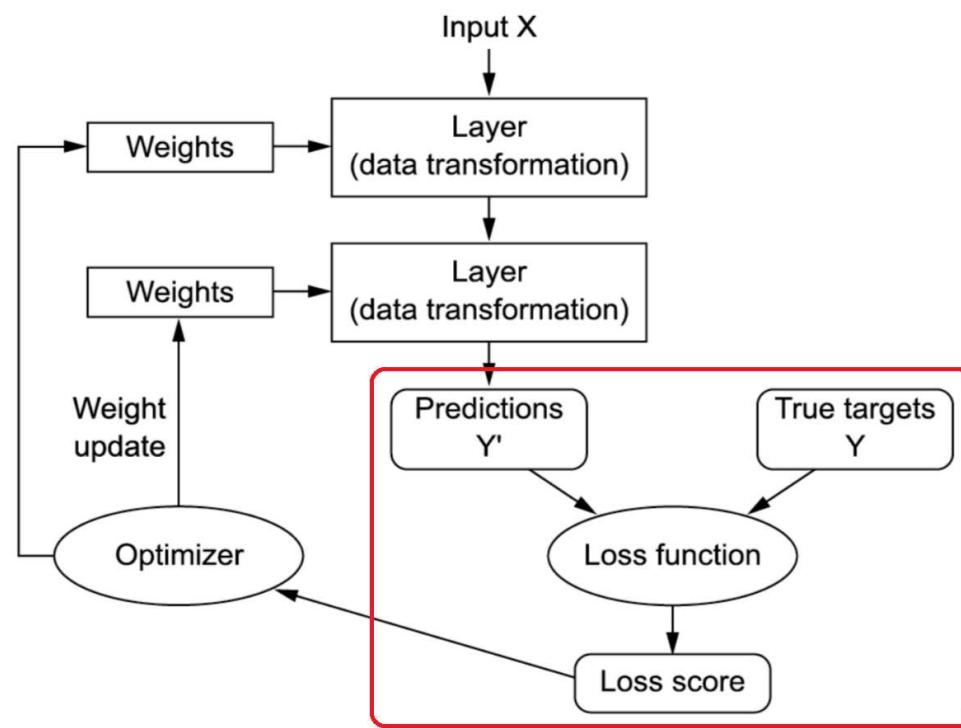
2.2.2. 훈련 스텝의 핵심 요소

- 가중치
- 순전파
- 손실 함수
- 역전파
- 경사하강법
- 옵티마이저
- 훈련 루프

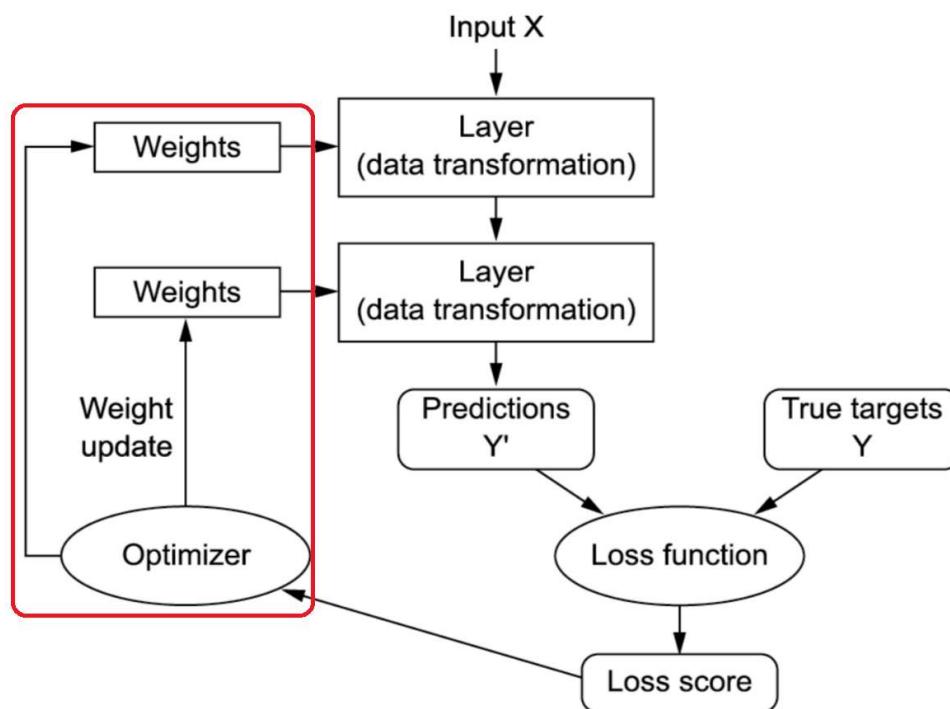
가중치와 순전파



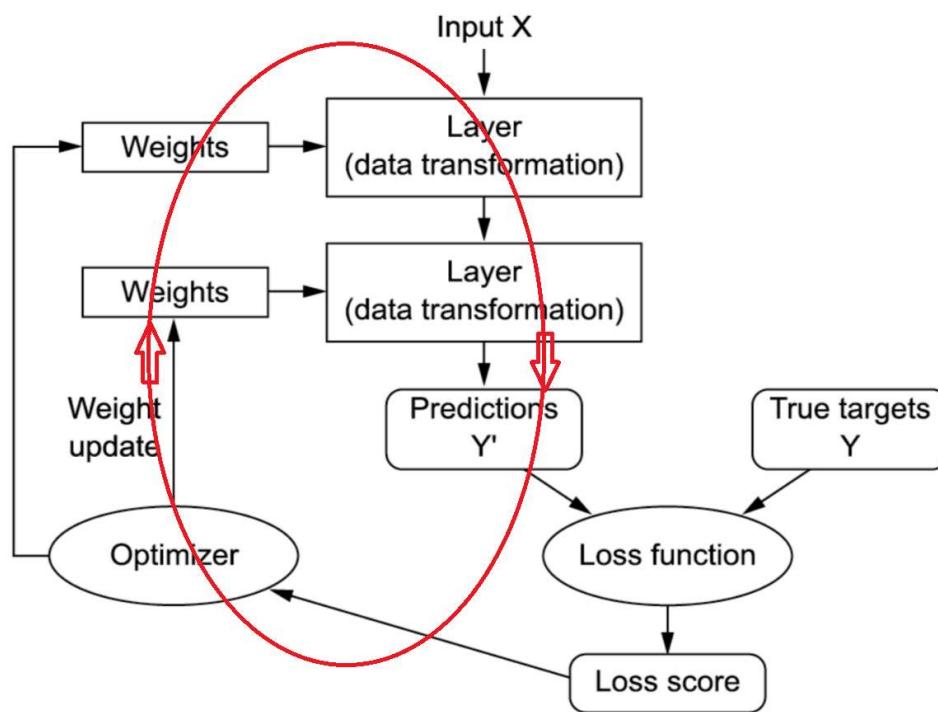
손실 함수



역전파, 경사하강법, 옵티마이저



훈련 루프



2.3. 학습된 모델의 활용과 평가

2.3.1. 모델 활용

```
test_digits = test_images[0:10]
predictions = model.predict(test_digits)
```

```
>>> predictions[0]
array([5.6115879e-10, 6.5201892e-11, 3.8620074e-06, 2.0421362e-04,
       2.3715735e-13, 1.0822280e-08, 3.6126845e-15, 9.9979085e-01,
       2.0998414e-08, 1.0214288e-06], dtype=float32)

>>> predictions[0].argmax()
7

>>> predictions[0][7]
0.99999106

>>> test_labels[0]
7
```

2.3.2. 모델 성능 평가

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
313/313 [=====] - 1s 3ms/step - loss: 0.0635 -
accuracy: 0.9811

>>> print(f"test_acc: {test_acc}")
test_acc: 0.9811000227928162
```