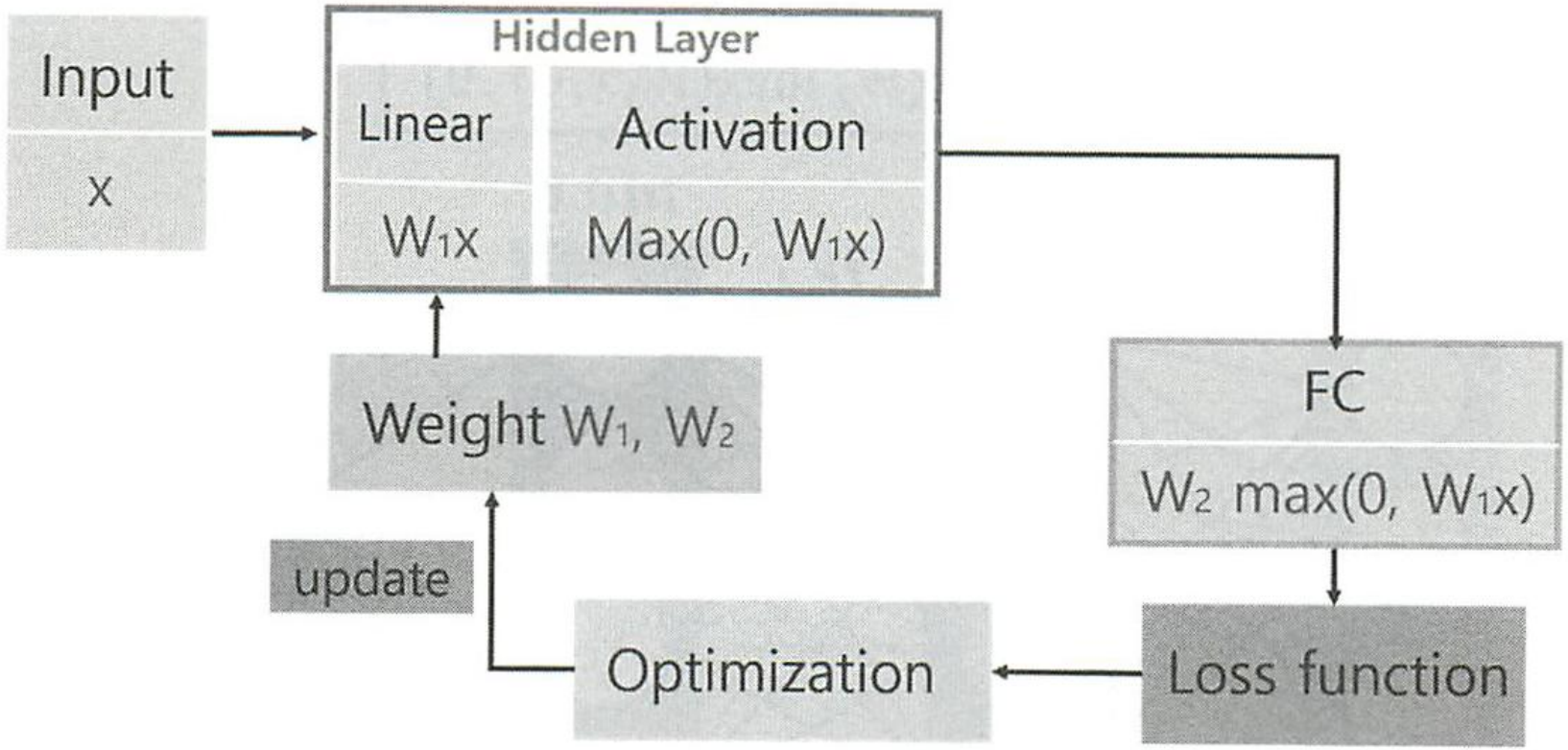
신경망 모델

신경망모델의 과정

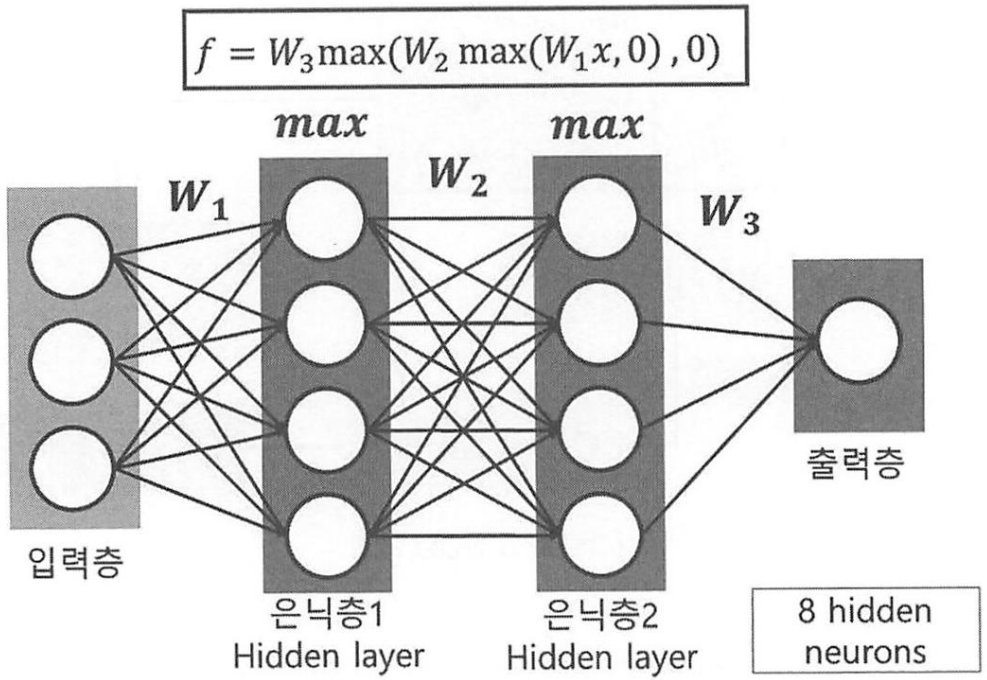
- 신경망의 층을 구분해주는 것은 비선형함수인 활성화 함수



3-Layer 신경망

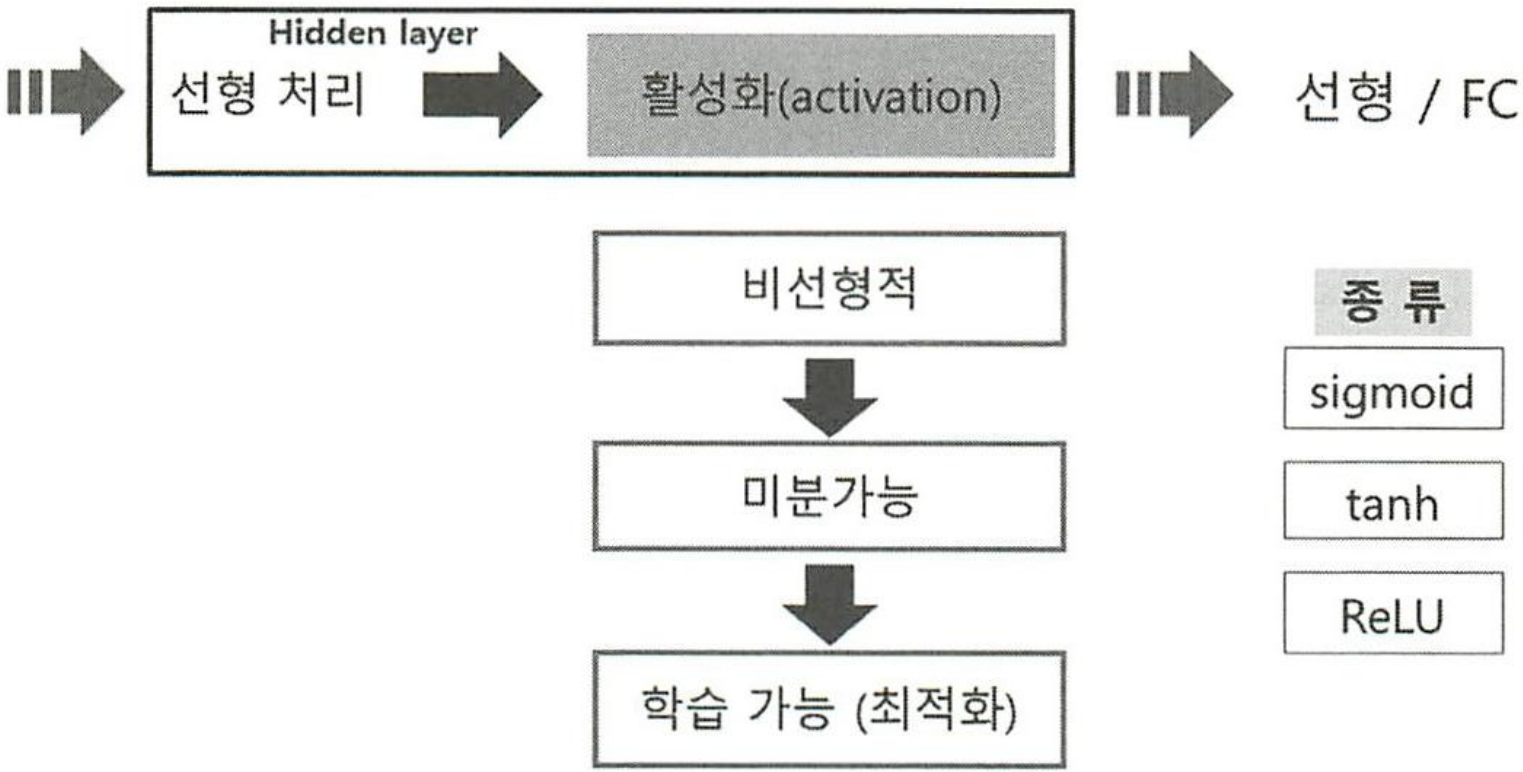
- 입력 데이터 x에 첫 번째 행렬을 곱하면 W_1x가 되고 여기에 활성화 함수 max(x,0)를 적용
- 같은 방식으로 마지막 층까지 계산하면 아래와 같은 식이 된다.

$$x \rightarrow W_1x \rightarrow max(x, 0) \rightarrow W_2x \rightarrow max(x, 0) \rightarrow W_3x$$



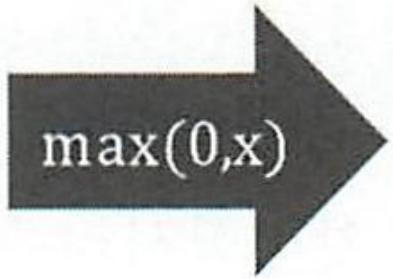
활성화 함수 - 특징

- 활성화 함수는 선형 모델에서 계산된 결과값을 활성화시켜주는 역할을 담당.



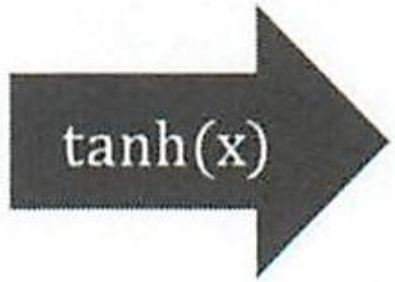
활성화

-1	3	2
0	2	-3
-1	-4	1



0	3	2
0	2	0
0	0	4

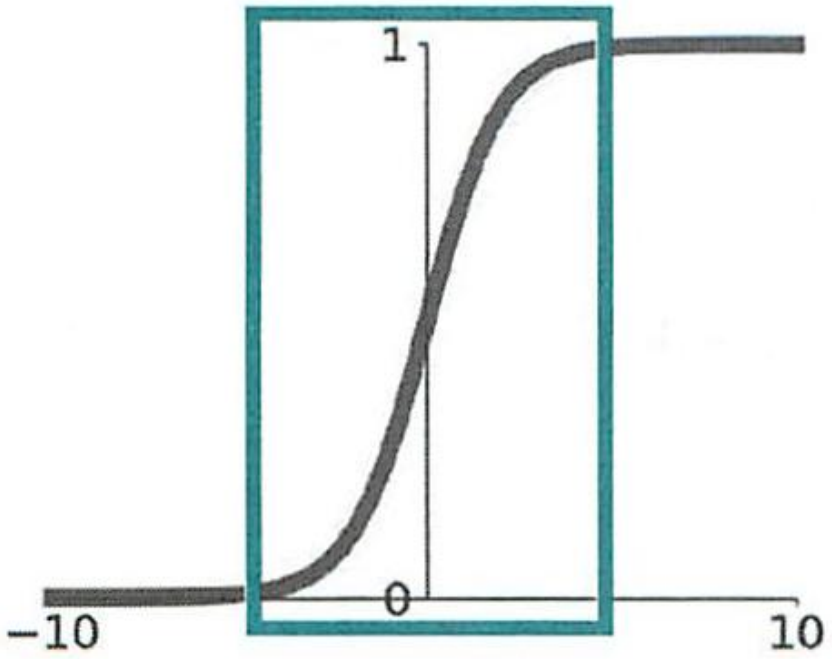
-1	3	2
0	2	-3
-1	-4	1



-0.762	0.995	0.964
0	0.964	-0.995
-0.762	-0.999	0.762

활성화 함수 - Sigmoid

- 자연어 처리등의 특별한 경우를 제외하고 잘 사용되지 않음 (컴퓨터 자원을 많이 소모함)

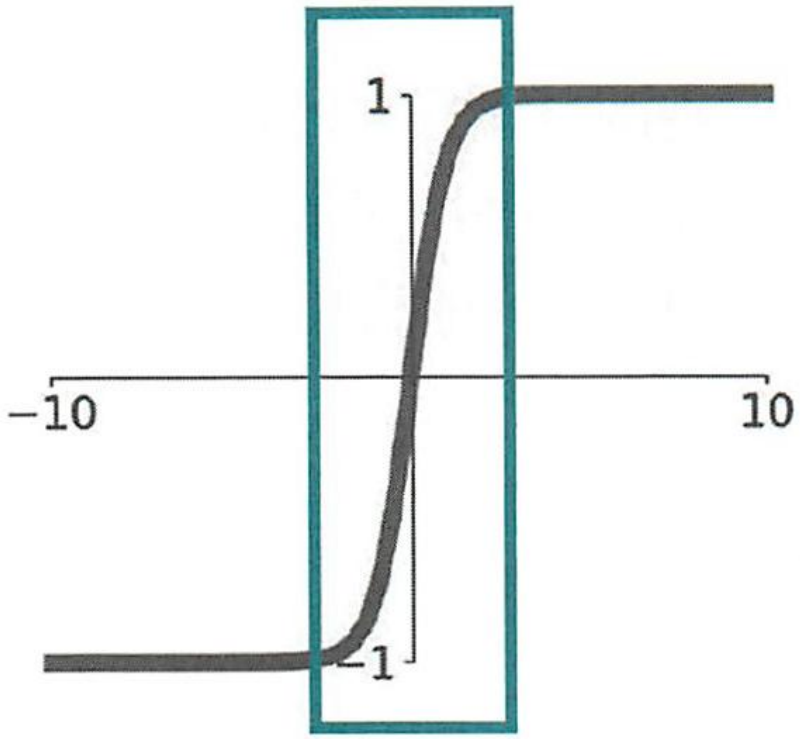


$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

input	output
$(-\infty, \infty)$	$\longrightarrow$ $(0, 1)$

활성화 함수 - tanh

- 원점 대칭 특징으로 인하여 sigmoid보다 나아졌지만, 여전히 학습이 되지 않는 구간이 많다.



$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

input	output
$(-\infty, \infty)$	$(-1, 1)$

미분 >0

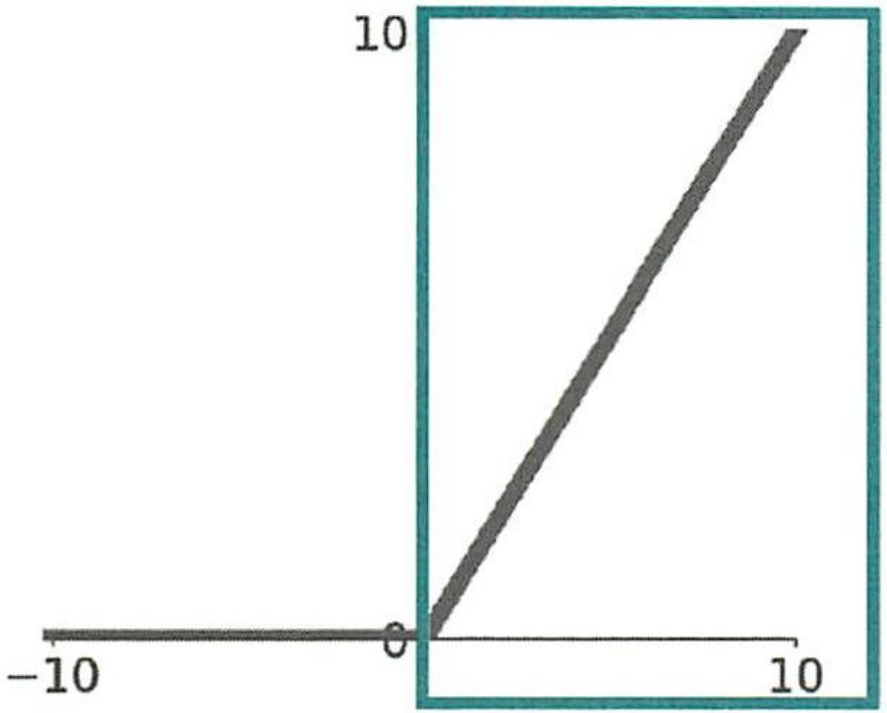
$(-a, a)$ ⟶ positive

outside ⟶ 0

활성화 함수 - ReLU

- 성능이 우수한 활성화 함수
- 음수인 경우는 학습이 이루어 지지 않음
- Leaky ReLU, ELU, Selu, mish, Swish

$$\mathrm{ReLU}(x) = \max(x, 0) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



input → **output**

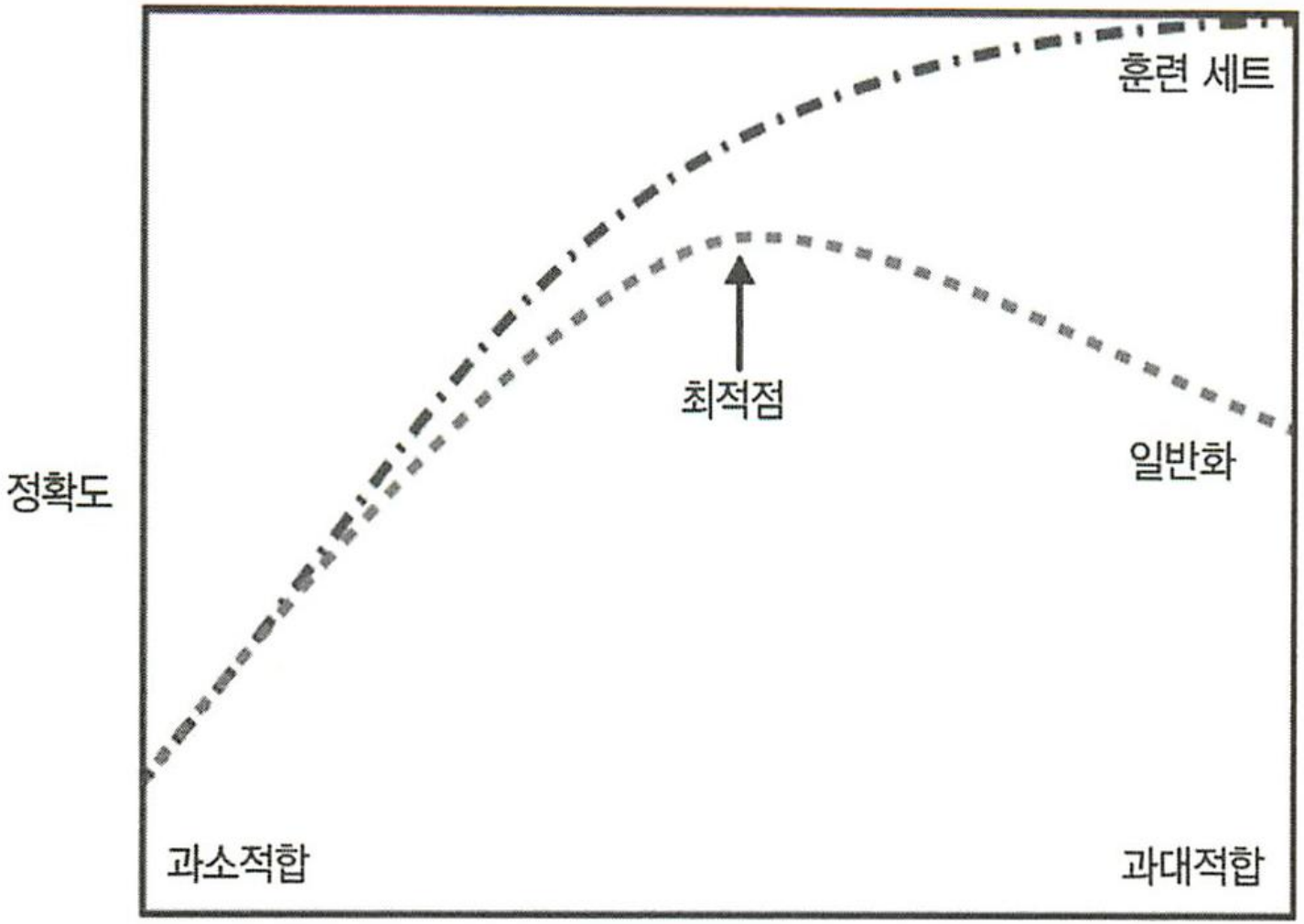
$(-\infty, \infty) \longrightarrow (0, \infty)$

미분 ≥ 0

$x > 0 \longrightarrow 1$

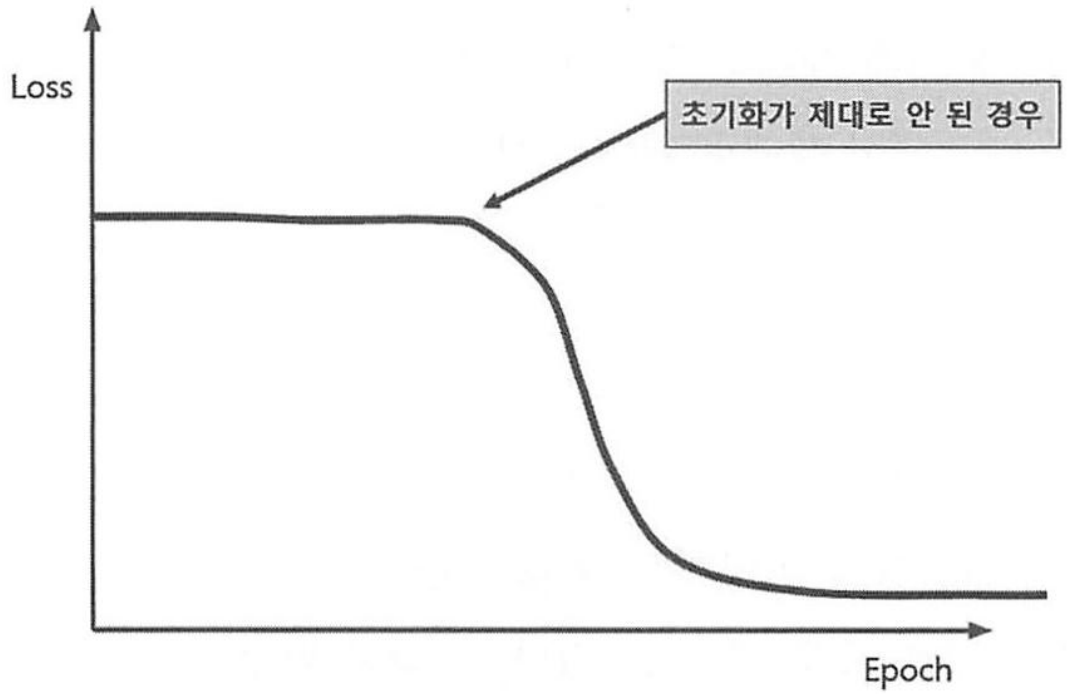
$x < 0 \longrightarrow 0$

학습분석 - 과적합

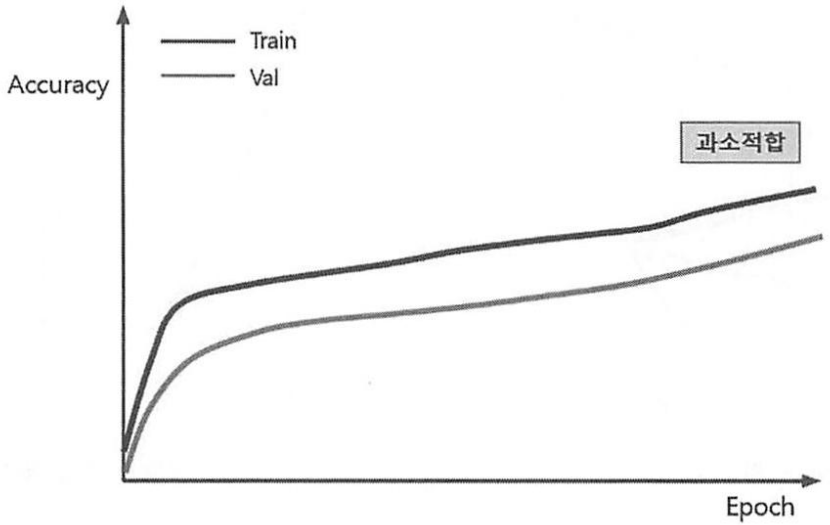


학습분석 – 학습곡선

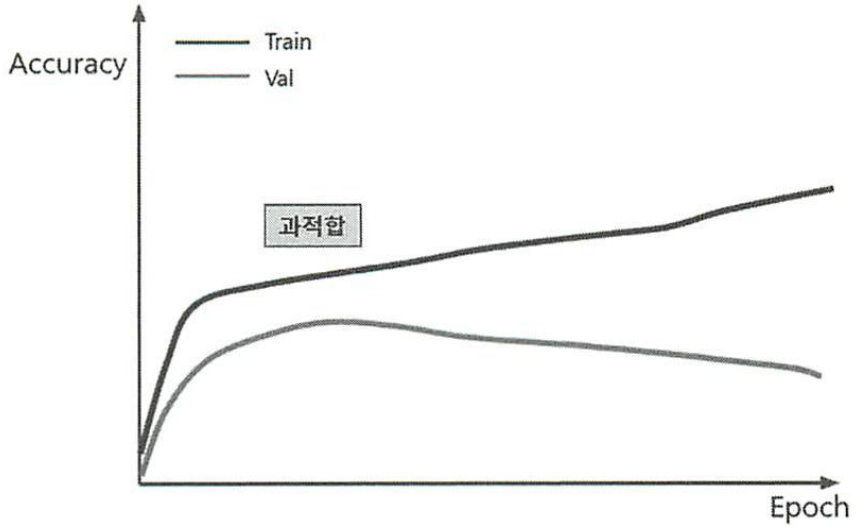
- 모델 가중치의 초기값 설정이 잘못된 경우, 가중치 설정을 다시 할 필요가 있음.



학습분석 – 학습곡선



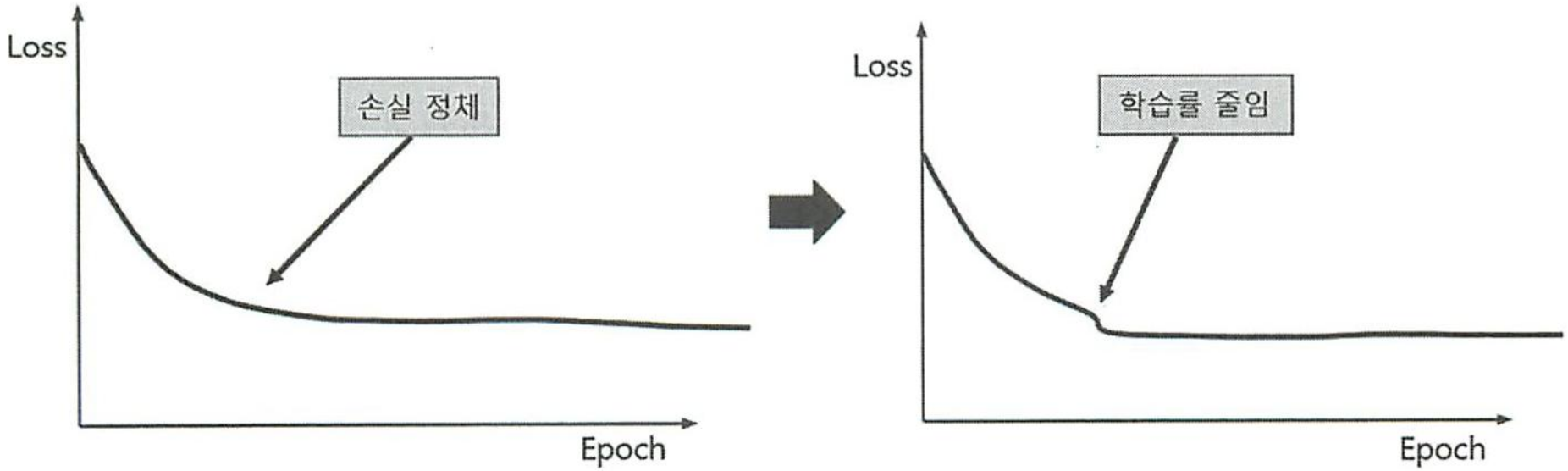
모델을 더 복잡하게 해야함.



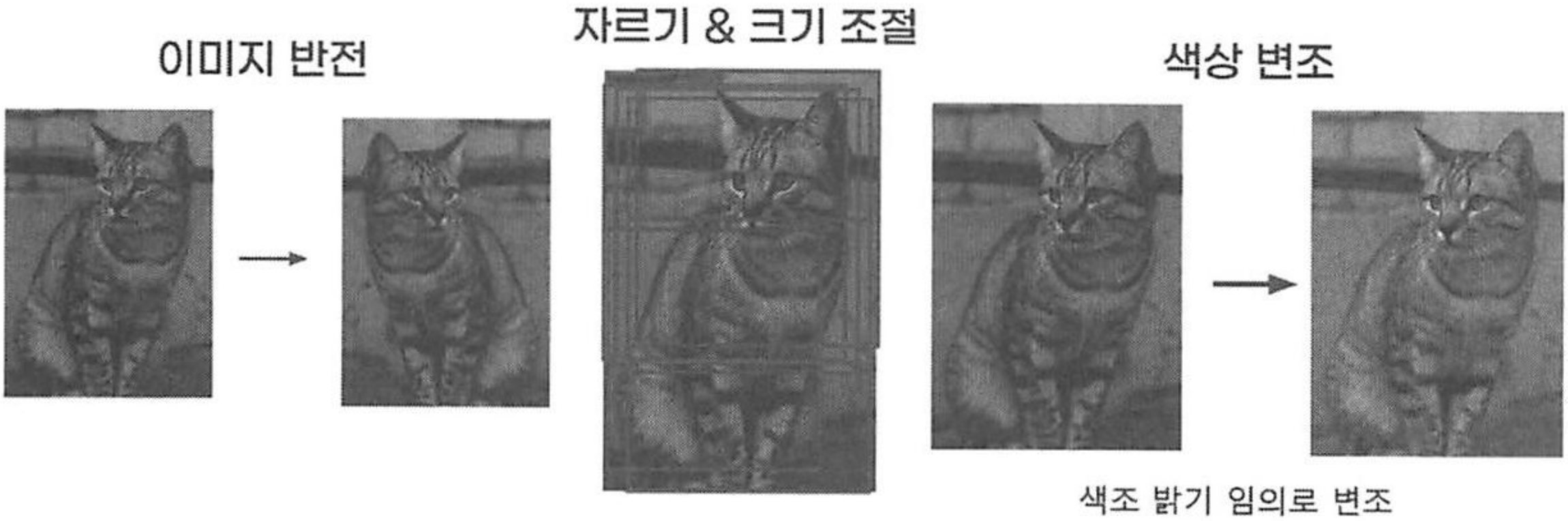
모델을 더 단순하게 해야함.

학습분석 – 학습곡선

- 학습 곡선이 진행 중에 정체되었을 경우, 학습률을 낮춰보면 학습이 더 원활히 진행되기도 한다.

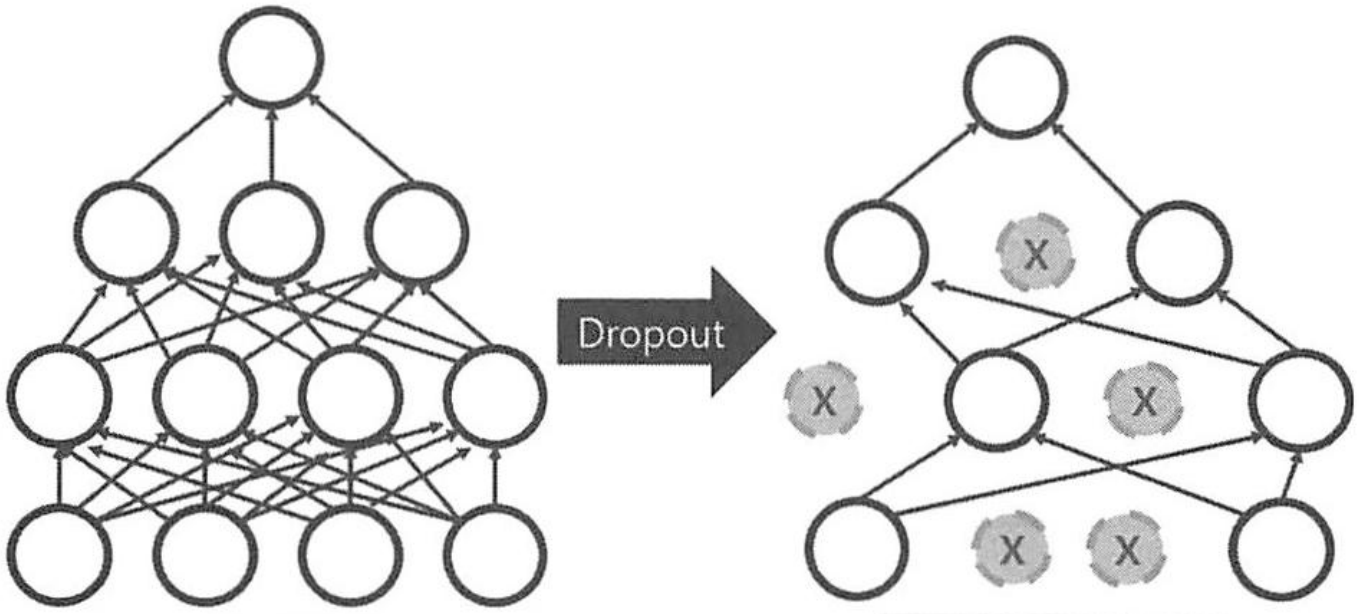


데이터 증강 – Data Augmentation

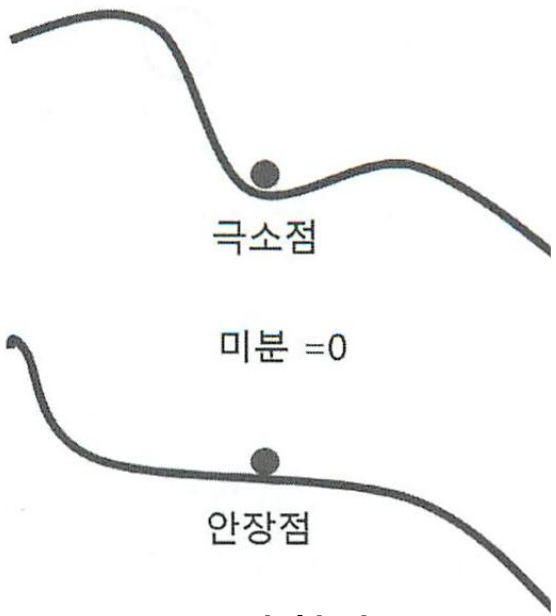


Drop Out

- 신경망의 연결을 일정 부분 제거하여 모델을 단순화하여 과적합을 해결할 수 있음.

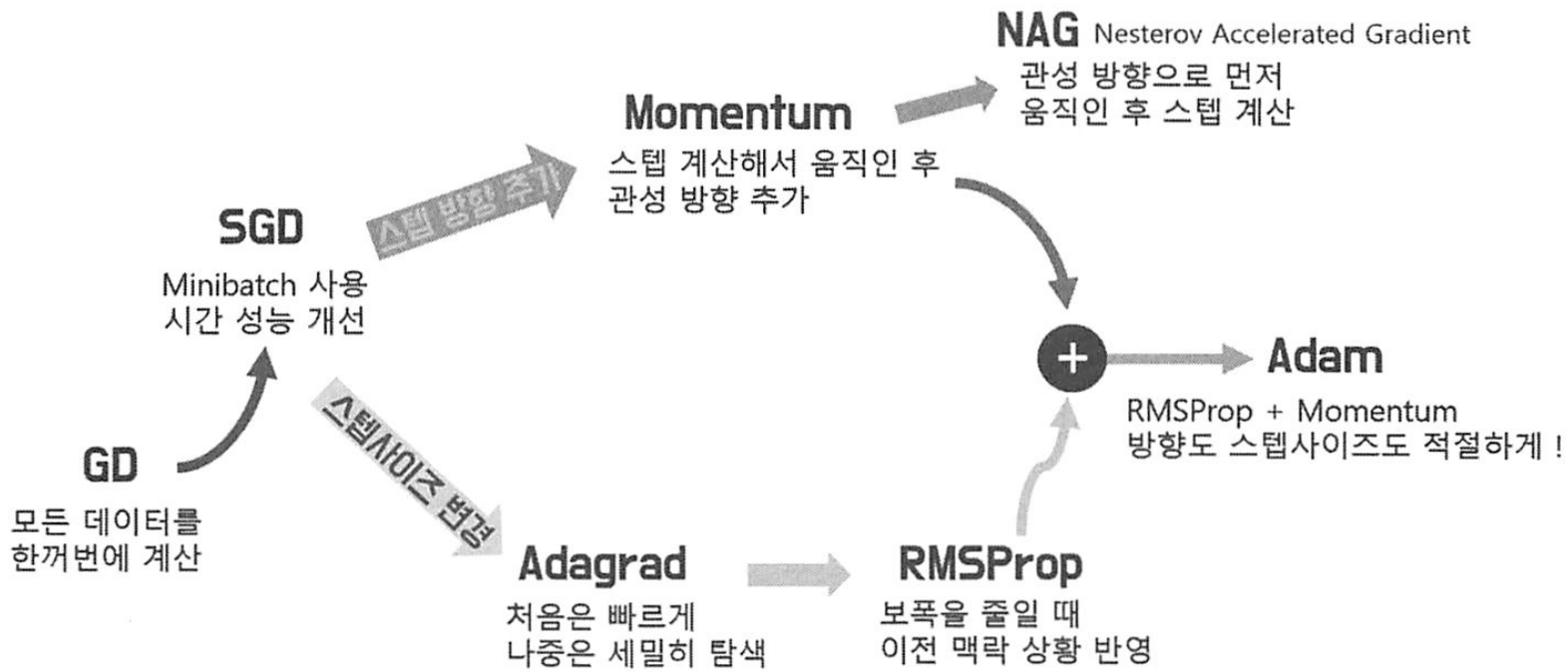


최적화기법



SGD의 한계

SGD의 개선



Mnist 실습

1 기본 라이브러리 불러오기

```
import numpy as np
import pandas as pd
```

2 데이터셋 불러오기

```
from tensorflow.keras.datasets.mnist import load_data
(train_x, train_y), (test_x, test_y) = load_data()
```

2 – 1 데이터 확인하기

```
train_x.shape, train_y.shape    # train 데이터 크기 확인

test_x.shape, test_y.shape    # test 데이터 크기 확인
```

Mnist 실습

2 – 2 이미지 확인하기

```
from PIL import Image
img = train_x[0]

import matplotlib.pyplot as plt
img1 = Image.fromarray(img, mode = 'L')
plt.imshow(img1)

train_y[0]    # 첫번째 데이터 확인
```

Mnist 실습

3 데이터 전처리

3 – 1 입력 형태 변환: 3차원 → 2차원

데이터를 2차원 형태로 변환: 입력 데이터가 선형모델에서는 벡터 형태

```
train_x1 = train_x.reshape(60000, -1)
test_x1 = test_x.reshape(10000, -1)
```

3 – 2 데이터 값의 크기 조절 : 0~1 사이 값으로 변환

```
train_x2 = train_x1/255
test_x2 = test_x1/255
```

Mnist 실습

4 모델 설정

4 – 1 모델 설정에 필요한 라이브러리 호출

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

4 – 2 모델명을 md로 설정해 선형층(Dense) 쌓기

```
md = Sequential()
md.add(Dense(128, activation = 'relu', input_shape = (28*28, )))
md.add(Dense(64, activation = 'relu'))
md.add(Dense(10, activation = 'softmax')) # 선형분류이므로 마지막 층은 softmax로 활성화
md.summary() # 모델을 요약해 출력
```

Mnist 실습

5 모델 학습 진행

5 – 1 모델 compile: 손실 함수, 최적화 함수, 측정 함수 설정

```
md.compile(loss = 'sparse _ categorical _ crossentropy', optimizer = 'sgd',
metrics = 'acc')
```

5 – 2 모델 학습: 학습 횟수, batch_size, 검증용 데이터 설정

```
hist = md.fit(train _ x2, train _ y, epochs = 30, batch _ size = 64, validation _
split = 0.2)
```

Mnist 실습

학습결과 분석 : 학습 곡선 그리기

```
plt.figure(figsize=(10,8))
plt.plot(epoch, acc, 'b', label='Training accuracy')
plt.plot(epoch, val_acc, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Mnist 실습

6 테스트용 데이터 평가

```
md.evaluate(test_x2, test_y)
```

7 가중치 저장

```
weight = md.get_weights()
weight
```

Mnist 실습

Model Loss 시각화

```
plt.plot(hist.history['loss'], label='loss')
plt.plot(hist.history['val_loss'], label='val_loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```

Mnist 실습

모델 재설정

```
## 모델 재설정
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
md = Sequential()
md.add(Dense(128, activation = 'relu', input_shape = (28*28, )))
md.add(Dense(64, activation = 'relu'))
md.add(Dense(32, activation = 'relu'))
md.add(Dense(10, activation = 'softmax'))
md.summary()

# 0.9740999937057495
```

이렇게 모델을 재설정하면 성능이 별로 향상되지 않는다.

```
## Dropout 적용 모델
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
md = Sequential()
md.add(Dense(128, activation = 'relu', input_shape = (28*28, )))
md.add(Dense(64, activation = 'relu'))
md.add(Dropout(0.5))
md.add(Dense(10, activation = 'softmax'))
md.summary()

# 0.9785000085830688
```