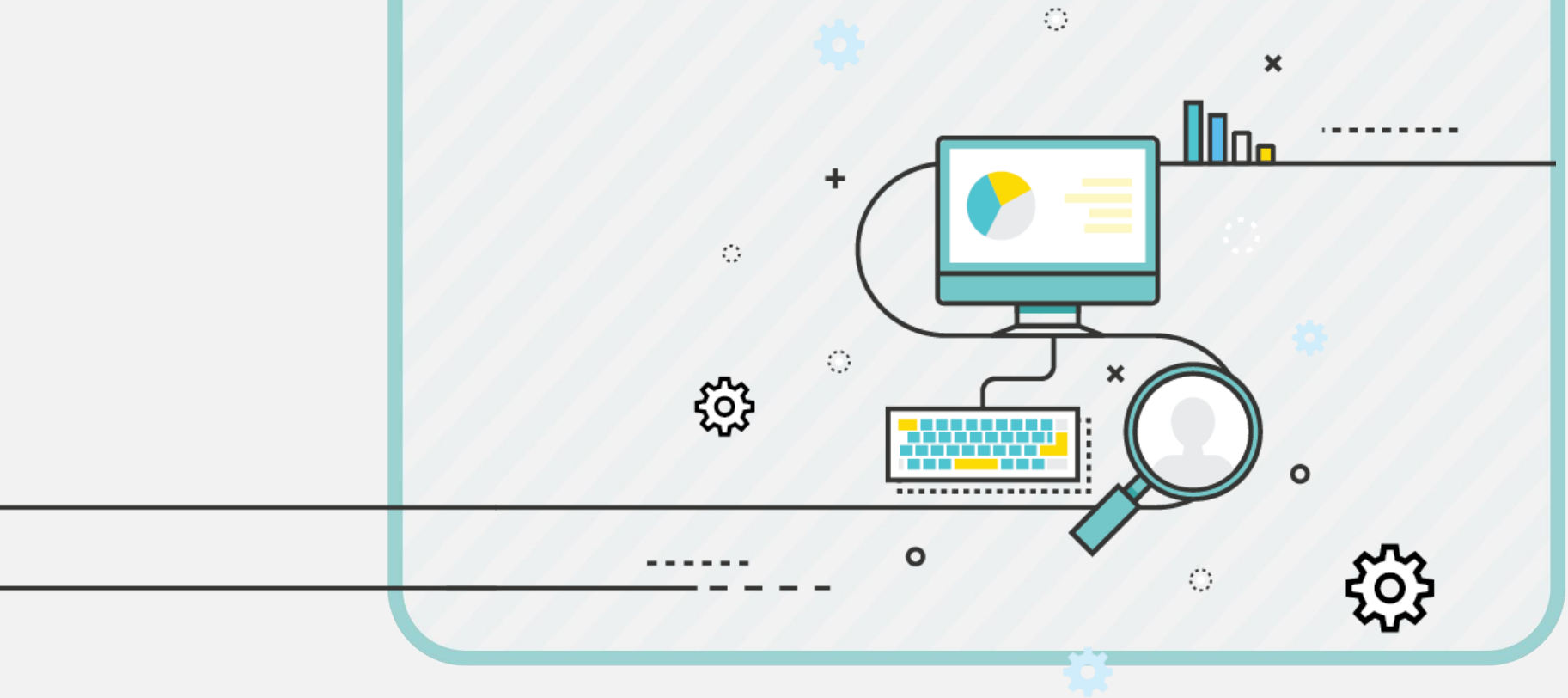
# 신경망이란

01 신경망의 기본 개념
02 케라스 개발 과정



# 세번째 만나볼 내용?

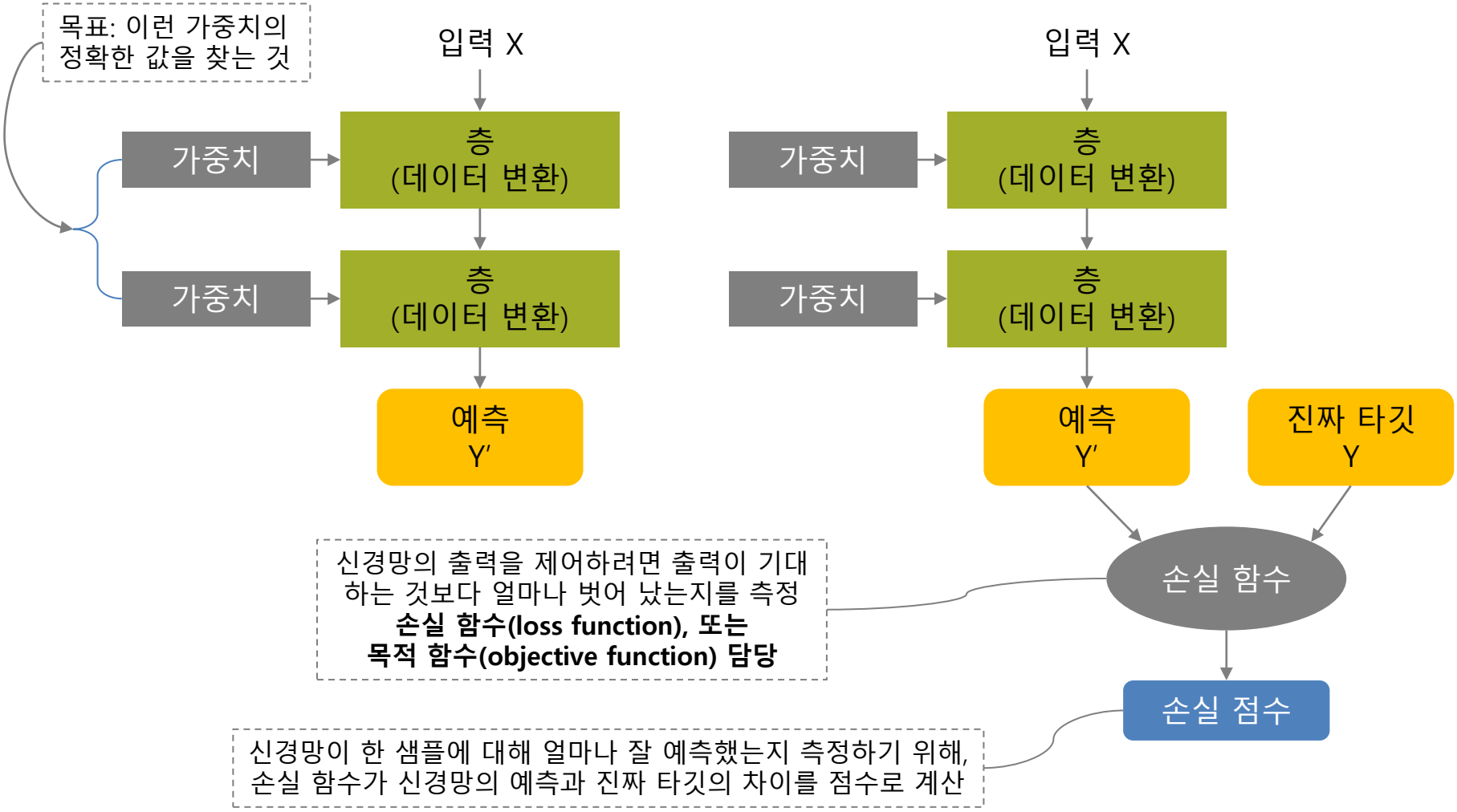
- 텐서플로우를 사용해보고, 신경망의 기본 개념에 대해 알아보자
  - 경사하강법과 역전파는 신경망을 다루기 위한 필수 개념이므로 많은 글을 참고하여 정확히 이해해야 함

- 케라스의 개발 과정
  - 매우 간단하지만 강력한 결과를 얻을 수 있음

- 이 장에서 배울 내용
  - 케라스 개발 과정
    - 학습 데이터 정의 → 모델 구성(model, compile) → 학습(fit) → 평가(evaluate, predict)

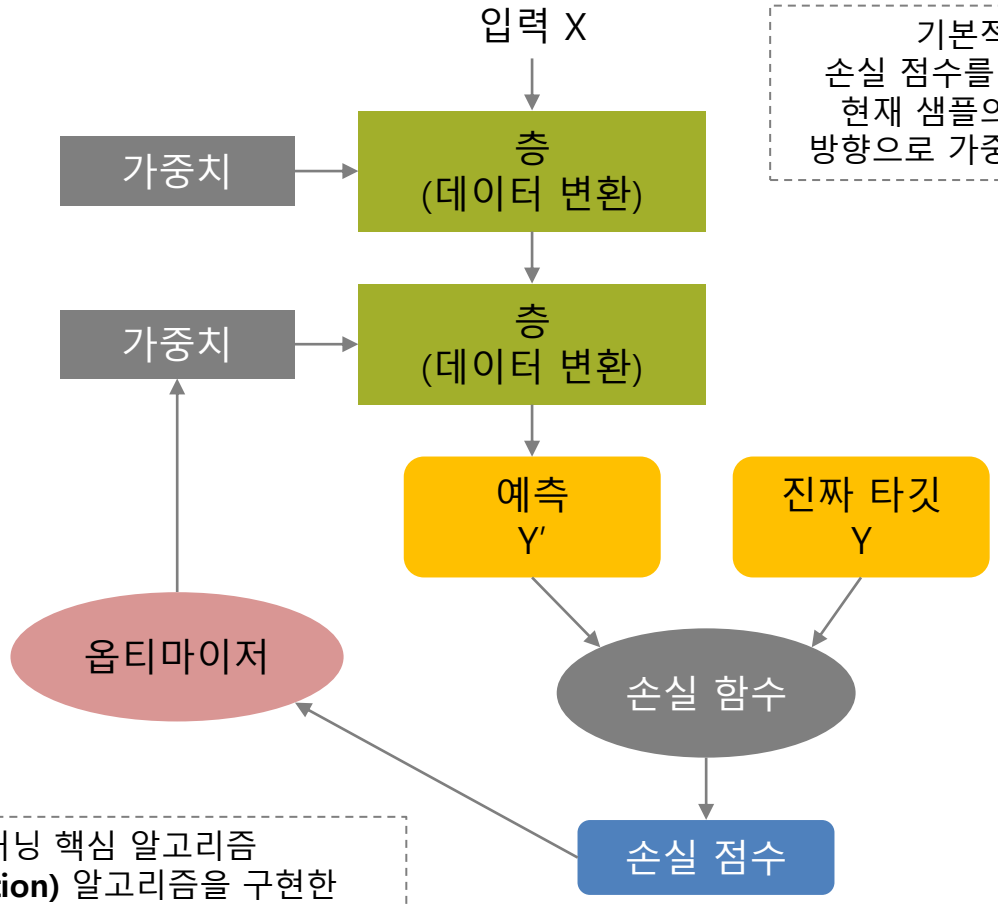
# 신경망의 작동 원리(1/2)

- 신경망은 가중치를 파라미터로 가진다.
- 손실 함수가 신경망의 출력 품질을 측정.



# 신경망의 작동 원리(2/2)

- 손실 점수를 피드백 신호로 사용하여 가중치 조절




기본적인 딥러닝 방식은
손실 점수를 피드백 신호로 사용하여
현재 샘플의 손실 점수가 감소되는
방향으로 가중치를 조금씩 수정하는 것

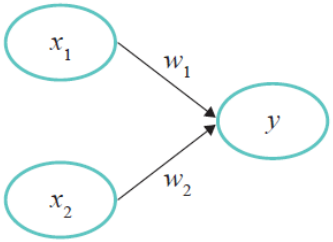
수정 과정은 딥러닝 핵심 알고리즘
**역전파(Backpropagation)** 알고리즘을 구현한
**옵티마이저(optimizer) 담당**

# 신경망

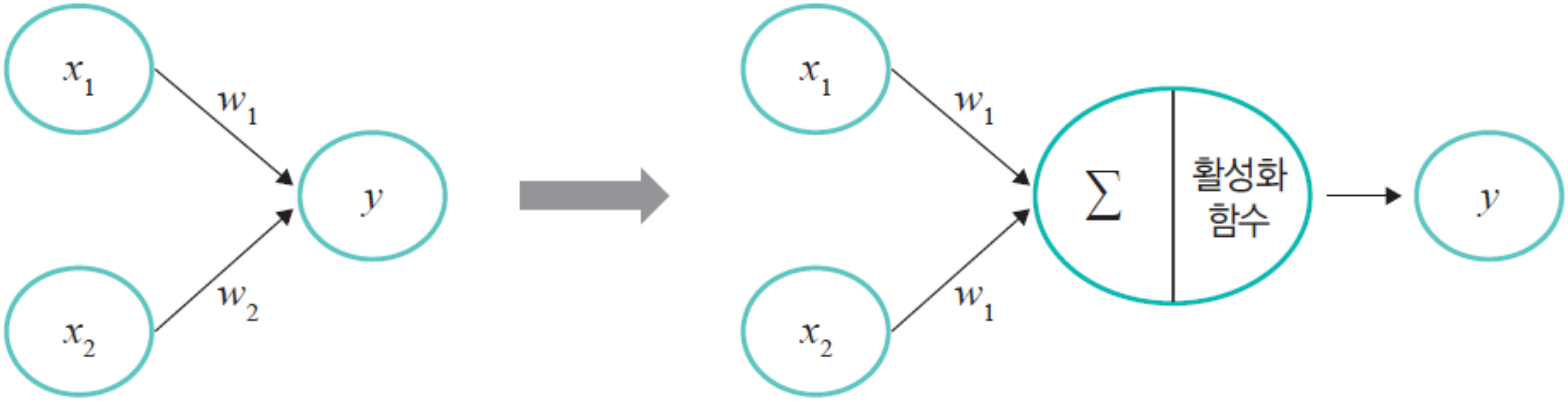
- 퍼셉트론(Perceptron)
  - **여러 개의 신호를 입력으로 받아 하나의 값을 출력**
  - x는 입력, y는 출력, w는 가중치
  - x와 가중치 w를 곱한 값을 모두 더하여 하나의 값(y)로 만들어냄
  - 이때, 임곗값(threshold)과 비교하여 크면 1, 그렇지 않으면 0을 출력

    → **활성화 함수(Activation Function)**

    → 위에서 사용한 것은 계단 함수(Step Function)




[그림 3-1] 퍼셉트론




[그림 3-2] 퍼셉트론과 퍼셉트론의 기본 단위

# 신경망: OR 게이트 문제

[함께 해봐요] OR 게이트 구현해보기 perceptron.ipynb

```
01 import tensorflow as tf
02 tf.random.set_seed(777)  # 시드를 설정합니다.
03
04 import numpy as np
05 from tensorflow.keras.models import Sequential
06 from tensorflow.keras.layers import Dense
07 from tensorflow.keras.optimizers import SGD
08 from tensorflow.keras.losses import mse
09
10 # 데이터 준비하기
11 x = np.array([[0, 0], [1, 0], [0, 1], [1, 1]])
12 y = np.array([[0], [1], [1], [1]])
13
14 # 모델 구성하기
15 model = Sequential()
16 # 단층 퍼셉트론을 구성합니다.
17 model.add(Dense(1, input_shape = (2, ), activation = 'linear'))
18
19 # 모델 준비하기
20 model.compile(optimizer = SGD(),
21               loss = mse,
22               metrics = ['acc'])  # list 형태로 평가지표를 전달합니다.
23
24 # 학습시키기
25 model.fit(x, y, e
```

```
Epoch 194/500
4/4 [==============================] - 0s 2ms/sample - loss: 0.1164 - acc: 1.0000
... 생략 ...
```

| x1 | x2 | y |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

- tf.random.set_seed(777)
  - **실험의 재생산성**
- Dense층
  - 퍼셉트론 생성
  - 밀집층, 다층 퍼셉트론, 완전 연결층 등
- Dense(1, input_shape = (2, ))
  - 두 개의 특성을 가지는 1차원 데이터를 입력으로 받고, 한 개의 출력을 가지는 Dense층
  - '1'은 퍼셉트론의 개수 또는 **은닉 유닛(hidden unit)**이라고 표현
- AND, NAND 게이트도 해결해보세요!

# 신경망: AND, XOR 게이트 문제

- AND 게이트
  - **둘 다 참일 때만, 참을 결과로 출력하는 연산**
  - 모든 입력값이 1일 때만 1을 출력

| Input 1 | Input 2 | AND |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

```
x = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([[1], [0], [0], [0]])
```

```
loss: 0.0666 - acc: 1.0000

loss: 0.0666 - acc: 1.0000
```

학습 성공

- XOR 게이트
  - **둘 중 참 하나일 때, 참을 결과로 출력하는 연산**
  - 입력값이 같지 않으면 1을 출력

| Input 1 | Input 2 | XOR |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

```
x = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([[0], [1], [1], [0]])
```

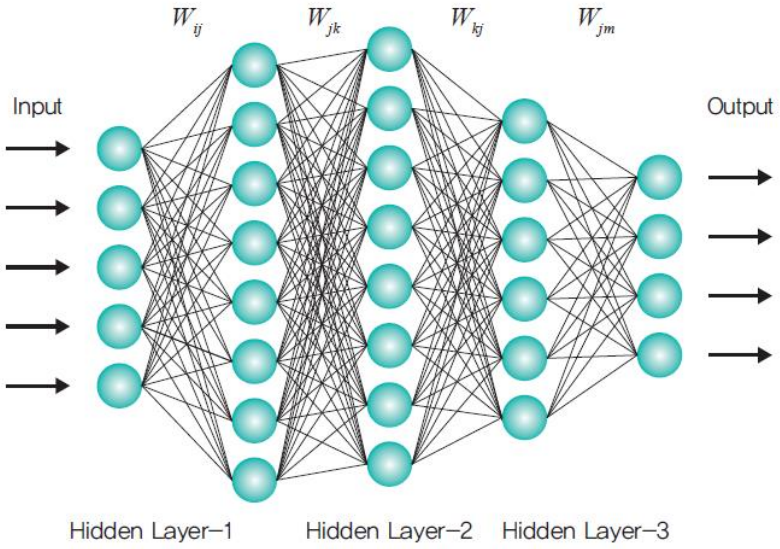
```
loss: 0.2500 - acc: 0.5000

loss: 0.2500 - acc: 0.5000
```

학습 실패

# 다층 퍼셉트론

- 전과 같은 문제를 해결한 것이 **다층 퍼셉트론(Multi-Layer Perceptron)**
  - 그림의 선이 전부 가중치에 해당함
  - 실제로 사용한 퍼셉트론은 굉장히 많음
    → 연산 비용이 큼
    → **벡터화(Vectorization)**을 이용




[그림 3-4] 다층 퍼센트론

m : 데이터의 개수
n : 데이터 특성의 개수
k : 층의 유닛 수

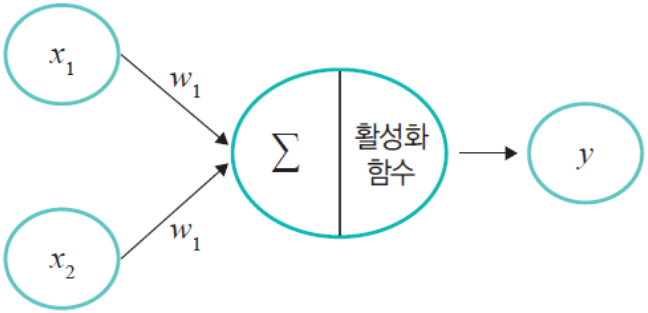
$$\begin{bmatrix} X_{11} & \cdots & X_{1n} \\ \vdots & \ddots & \vdots \\ X_{m1} & \cdots & X_{mn} \end{bmatrix} \cdot \begin{bmatrix} W_{11} & \cdots & W_{1k} \\ \vdots & \ddots & \vdots \\ W_{n1} & \cdots & W_{nk} \end{bmatrix} = \begin{bmatrix} Z_{11} \end{bmatrix}$$

(m, n) (n, k) (m, k)

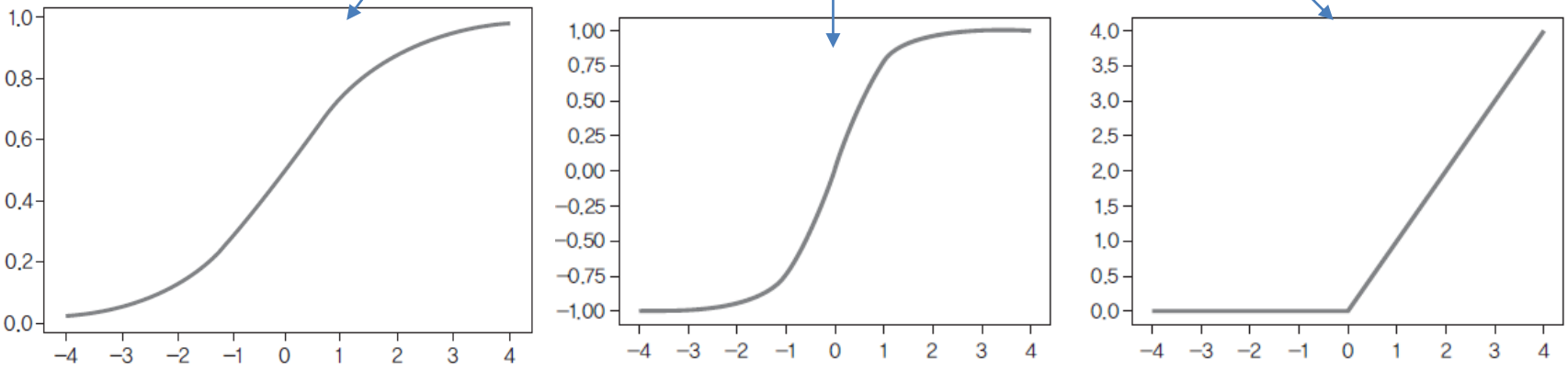
[그림 3-5] 벡터의 내적

# 활성화 함수

- XOR 게이트 문제에서 ReLU 활성화 함수를 사용
  - 비선형 활성화 함수
  - **선형 활성화 함수를 쓰면, f(f(f(x))) → f(x)와 동일**
  - 층을 쌓는 의미가 없어져... → 비선형 활성화 함수 사용으로 해결

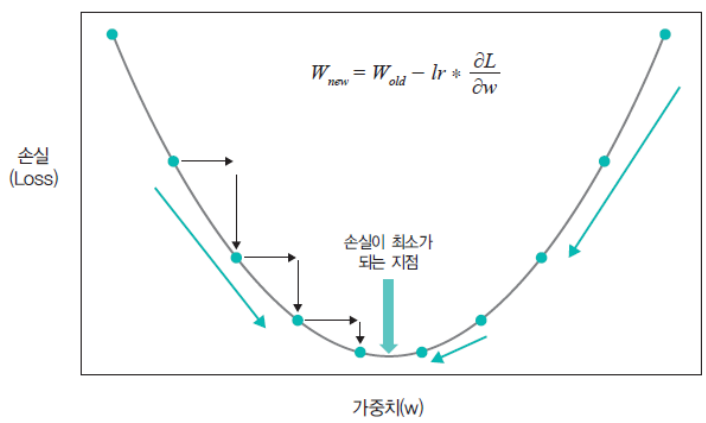



- 대표적으로 사용되는 시그모이드(Sigmoid), 하이퍼볼릭 탄젠트(tanh), ReLU 활성화 함수
  - 그 외에 PReLU, ELU, Leaky-ReLU, swish 등이 존재

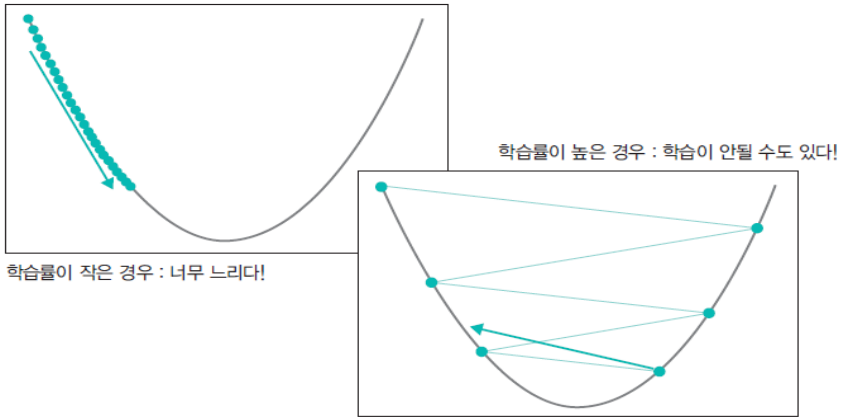


# 경사하강법

- Gradient Descent Algorithm (경사하강법)
  - 특정 함수에서의 미분을 통해 얻은 기울기를 활용하여 최적의 값을 찾아가는 방법



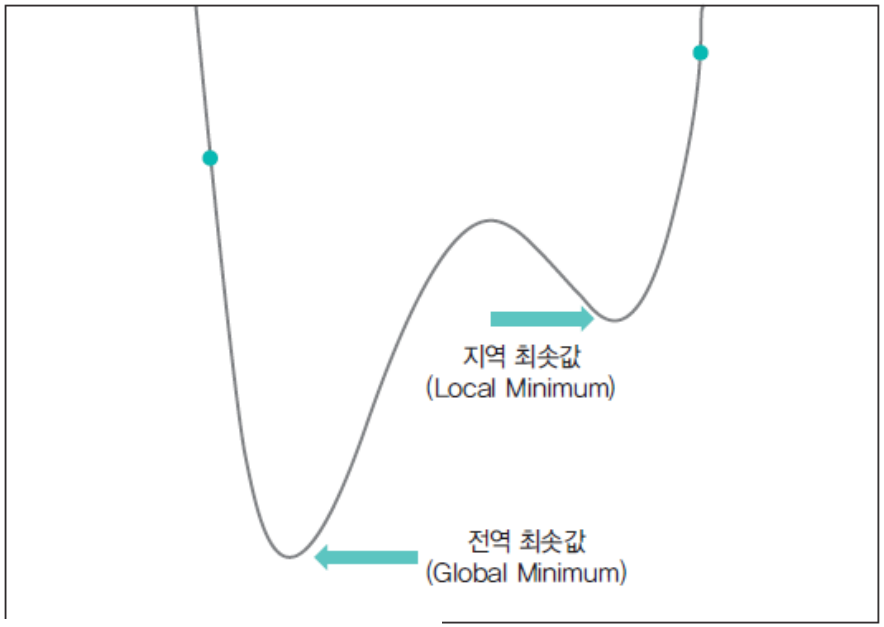

[그림 3-8] 경사하강법($y=x^2$)

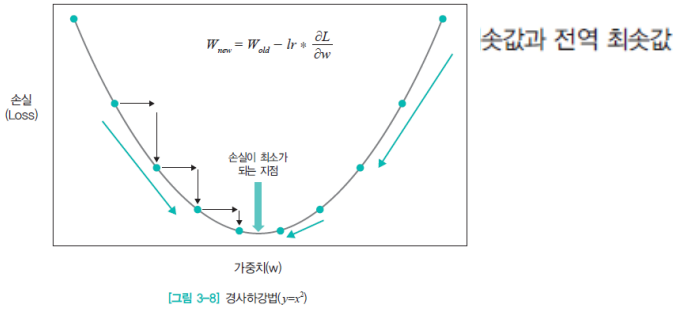



[그림 3-10] 학습에 영향을 미치는 학습률의 크기

  - 그림에서 사용하고 있는 **학습률(lr)**은 성능, 학습 속도에 중요한 영향을 끼치는 **하이퍼파라미터**
    - 학습률이 너무 높으면 학습이 되지 않을 수 있음
    - 그렇다고 너무 낮으면, 최적값에 도달하기 전에 학습이 종료
    - 주로 1e-3(0.001)을 기본값으로 사용
  - 위 함수는 어느 지점에서 출발해도 경사를 따라가다 보면 최적값(손실이 최소가 되는 지점)에 도달함
  - 하지만 우리가 만날 함수 공간은?

# 경사하강법



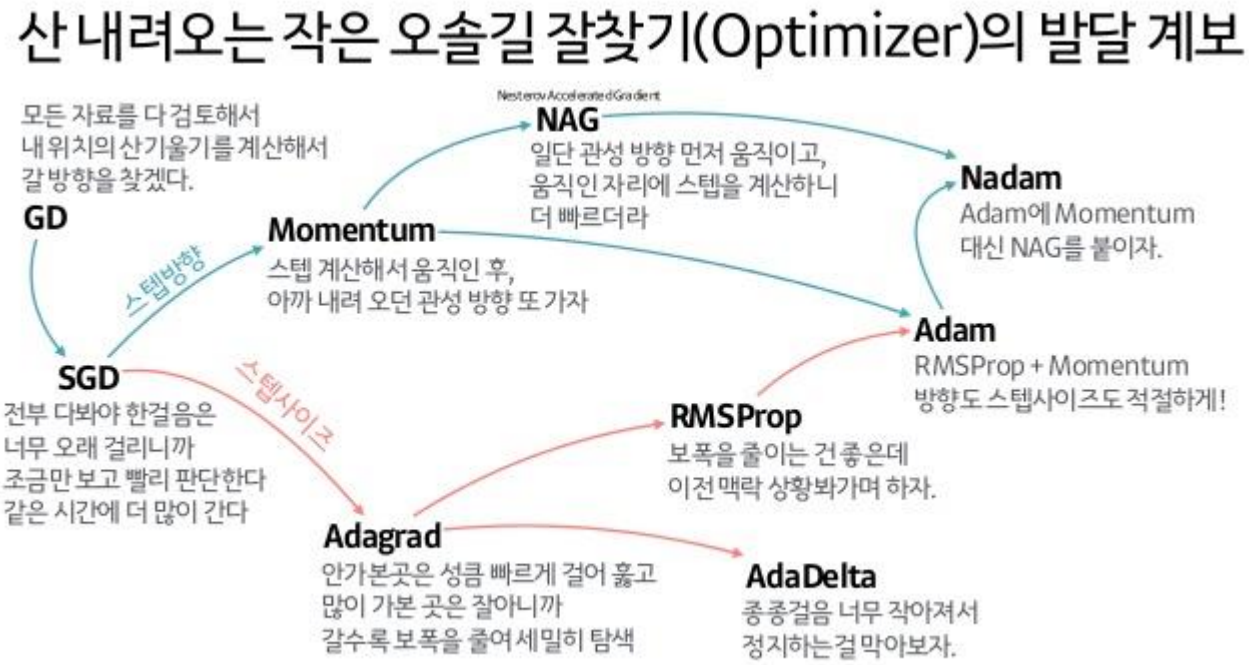

[그림 3-8] 경사하강법($y=x^2$)

솟값과 전역 최솟값

- **경사하강법은 항상 최적값을 반환한다는 보장을 할 수 없음**
  - 왼쪽 점에서 시작할 경우
    - 전역 최솟값 ← Good!!
  - 오른쪽 점에서 시작할 경우
    - 지역 최솟값 ← Bad!!
- **가중치 초기화 문제**
  - weight initialization
  - 왼쪽 점? 오른쪽 점?
  - 특별한 경우가 아닌 이상, 케라스가 제공하는 기본값을 사용해도 무방
  - Glorot(Xavier), he, Lecun 초기화
- 배치 단위를 사용하여 진행
  - **확률적 경사 하강법**
  - SGD; Stochastic Gradient Descent

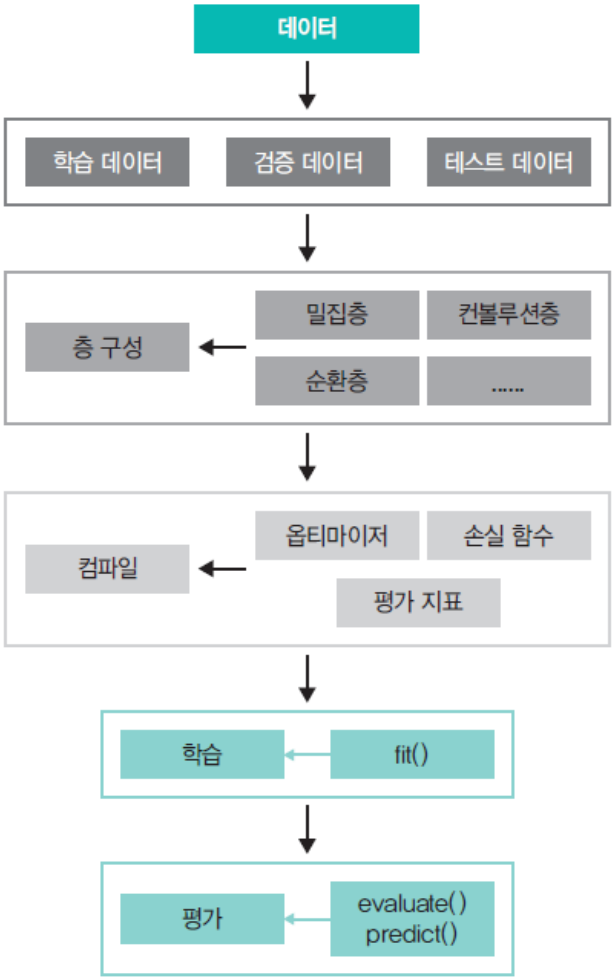
# 신경망 학습 최적화 Optimization

- 하이퍼파라미터를 어떻게 설정하느냐에 따라서 학습의 결과가 천차만별로 달라짐
- 신경망 모델의 학습과 그 결과에 따른 손실함수의 값을 최소화하는 방향으로 하이퍼파라미터의 값을 찾는 것이 최적화의 목표
- 최적의 가중치 값을 구하기 위해서 앞에서는 미분을 통해 기울기를 구하여 가중치 값을 갱신하는 방법인 확률적 경사하강법(Stochastic Gradient Descent; SGD) 방법
- 확률적 경사하강법 이외에도 다양한 최적화 기법을 통해 최적의 하이퍼파라미터 조합 찾을 수 있음



# 케라스에서의 개발 과정

1. 학습 데이터를 정의합니다.
2. 데이터에 적합한 모델을 정의합니다.
3. 손실 함수, 옵티마이저, 평가지표를 선택하여 학습 과정을 설정합니다.
4. 모델을 학습시킵니다.
5. 모델을 평가합니다.




[그림 3-12] 케라스에서의 개발 과정

# 케라스에서의 개발 과정

- 데이터 준비
  - 학습, 검증, 테스트 데이터로 분리

- 모델 구성
  - Sequential(), Functional API(7장) 방법

예시: Sequential()을 사용한 모델 구성

```
01 model = Sequential()
02 model.add(Dense(32, input_shape = (2, ), activation = 'relu'))
03 model.add(Dense(1, activation = 'sigmoid'))
```

  - 항상 모델의 첫 번째 층은 데이터의 형태(위 코드에서 input_shape 인자)를 전달해주어야 함
  - 두 번째 층부터는 자동으로 이전 층의 출력 형태가 입력 형태로 지정

# 케라스에서의 개발 과정

- compile() 함수를 통한 학습 과정 설정

예시: model.compile()

```
# 평균 제곱 오차 회귀 문제
model.compile(optimizer = RMSprop(),
              loss = 'mse',
              metrics = [ ])

# 이항 분류 문제
model.compile(optimizer = RMSprop(),
              loss ='binary_crossentropy',
              metrics = ['acc'])

# 다항 분류 문제
model.compile(optimizer = RMSprop(),
              loss ='categorical_crossentropy',
              metrics = ['acc'])
```

  - 옵티마이저(optimizer): 최적화 방법을 설정, SGD(), RMSProp(), Adam(), NAdam() 등
    - 'sgd', 'rmsprop', 'adam'과 같이 문자열로 지정하여 사용 가능
    - tf.keras.optimizers
  - 손실 함수(loss function): 학습 과정에서 최적화시켜야 할 손실 함수를 설정
    - mse(mean_squared_error), binary_crossentropy, categorical_crossentropy
    - tf.keras.losses
  - 평가 지표: 학습 과정을 모니터링하기 위해 설정
    - tf.keras.metrics

# 케라스에서의 개발 과정

- fit() 함수를 통한 모델 학습

예시: model.fit()

```
01  model.fit(data, label, epochs = 100)
02
03  model.fit(data, label, epochs = 100, validation_data = (val_data, val_label))
```

  - 에폭(epochs): 전체 학습 데이터를 몇 회 반복할지 결정
  - 배치 크기(batch_size): 전달한 배치 크기만큼 학습 데이터를 나누어 학습을 진행
  - 검증 데이터(validation_data): 모델 성능을 모니터링하기 위해 사용

- 평가 진행
  - evaluate(), predict()

예시: model.evaluate(), model.predict()

```
01  model.evaluate(data, label)
```

```
[0.21061016619205475, 1.0]
```

손실과 평가지표

```
01  result = model.predict(data)
02  print(result)
```

```
array([[0.48656905],
       [0.5464304 ],
       [0.552116  ],
       [0.4465039 ]], dtype=float32)
```

# 정리해봅시다

1. 머신러닝 프로세스는 간략하게 **[문제 정의 및 데이터 준비하기 → 학습하기 → 추론 및 평가]**로 나눌 수 있습니다.
2. **[문제 정의 및 데이터 준비하기]**는 명확한 문제 정의와 데이터 전처리가 매우 중요합니다.
3. **[학습하기]**는 본격적으로 모델을 선택하고, 학습시키는 단계입니다. 하이퍼파라미터 실험 환경 등을 고려하여 학습시간을 효율적으로 활용할 수 있도록 해야 합니다.
4. **[추론 및 평가]**는 올바른 지표를 통해 모델의 성능을 신뢰할 수 있어야 합니다. 주어진 상황에 맞는 지표를 선택하는 것은 매우 어렵고 중요합니다.
5. **구글 데이터셋 검색과 캐글**은 데이터셋을 탐색하고 수집할 최적의 장소입니다. 그 외에도 공공 데이터 포털, AI Hub가 있습니다.
6. 문제는 **공유와 소통**를 통해 더 빠르게 해결될 수 있습니다. 국내에 이를 위한 다양한 커뮤니티가 존재한다는 점을 잊지마세요.

# 정리해봅시다

1. 신경망은 **퍼셉트론 알고리즘**에서부터 출발합니다. **다층 퍼셉트론**을 사용하면 XOR 게이트 문제를 해결할 수 있습니다.
2. 케라스 모델의 **첫 번째 층은 항상 입력 데이터의 형태를 전달**해주어야 합니다.
3. 대표적으로 손실 함수에는 **['mse', 'binary_crossentropy', 'categorical_crossentropy']**, 옵티마이저에는 **['sgd', 'rmsprop', 'adam']**이 있으며, 문자열로 지정하여 사용할 수 있습니다

# 신경망 실습하기

01 신경망 모델 만들어보기
당뇨병 발병 유무 예측
폐암 수술 환자의 생존유무 예측

# 네번째 만나볼 내용?

- 배운 내용을 활용해서 다음 2가지 문제 해결을 다뤄봅니다.
  - 당뇨병 발병 유무 예측
  - 폐암 수술 환자의 생존유무 예측

- 이번 장에서는 위처럼 여러 가지 데이터셋을 다뤄볼 것입니다. 이러한 데이터셋은 다양한 구조의 신경망을 실험해볼때도 자주 사용되니 적극적으로 활용하기를 바랍니다.

# 다층 퍼셉트론 모델 만들어보기 - 당뇨병 발병 유무를 예측

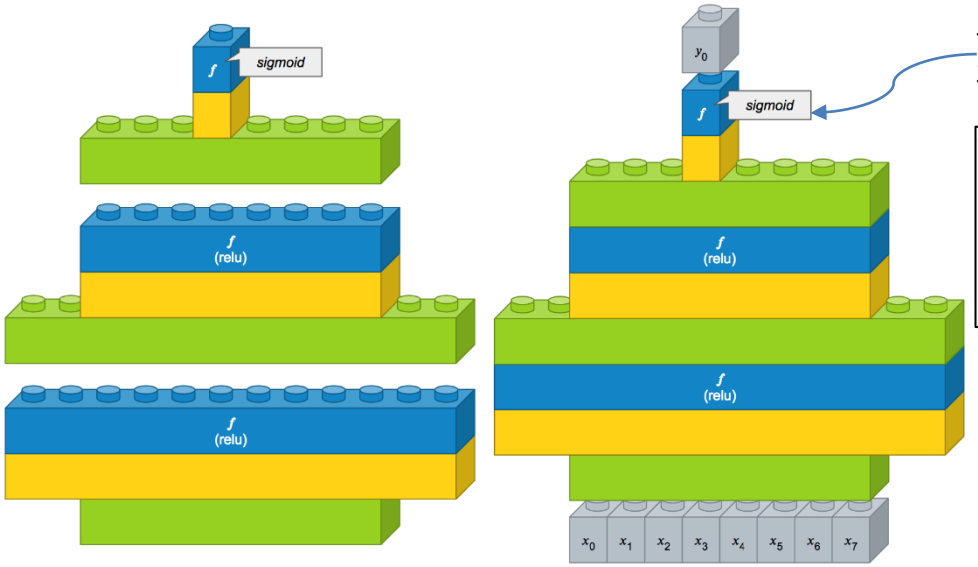
- **문제 정의하기**
  - 이진 분류 예제에 적합한 데이터셋은 8개 변수와 당뇨병 발병 유무가 기록된 '피마족 인디언 당뇨병 발병 데이터셋'
  - 8개 변수를 독립변수로 보고 **당뇨병 발병 유무를 예측하는 이진 분류 문제로 정의(**결과로 1이면 당뇨병, 0이면 아님)
- **데이터 준비하기**
- https://www.kaggle.com/uciml/pima-indians-diabetes-database
- 8가지 속성(1번~8번)과 결과(9번)의 상세 내용은 다음과 같습니다.
  - 임신 횟수
  - 경구 포도당 내성 검사에서 2시간 동안의 혈장 포도당 농도
  - 이완기 혈압 (mm Hg)
  - 삼두근 피부 두겹 두께 (mm)
  - 2 시간 혈청 인슐린 (mu U/ml)
  - 체질량 지수
  - 당뇨 직계 가족력
  - 나이 (세)
  - 5년 이내 당뇨병이 발병 여부

```
[14]  x_train = dataset[:700,0:8]
      y_train = dataset[:700,8]
      x_test  = dataset[700:,0:8]
      y_test  = dataset[700:,8]
```

- **데이터셋 생성하기**
  - 입력(속성값 8개)와 출력(판정결과 1개) 변수 학습 데이터: 700건, 테스트 데이터: 68건 → 총 데이터 768건

# 다층 퍼셉트론 모델 만들어보기 - 당뇨병 발병 유무를 예측

- **모델 구성하기**
  - Dense 레이어만을 사용하여 다층 퍼셉트론 모델을 구성데이터 준비하기
  - 속성이 8개이기 때문에 입력 뉴런을 8개이고, 이진 분류이기 때문에 0~1사이의 값을 나타내는 출력 뉴런이 1개




출력 레이어만 0과 1사이로 값이 출력될 수 있도록 활성화 함수를 'sigmoid'로 사용

```
[15] model = Sequential()
     model.add(Dense(12, input_dim=8, activation='relu'))
     model.add(Dense(8, activation='relu'))
     model.add(Dense(1, activation='sigmoid'))
```

- **모델 학습과정 설정하기**
  - 모델을 정의했다면 모델을 손실함수와 최적화 알고리즘(하이퍼파라미터) 설정
  - loss : 이진 클래스 문제이므로 'binary_crossentropy'으로 지정
  - optimizer : 효율적인 경사 하강법 알고리즘 중 하나인 'adam'을 사용
  - metrics : 평가 척도를 나타내며 분류 문제에서는 일반적으로 'accuracy'으로 지정

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

# 다층 퍼셉트론 모델 만들어보기 - 당뇨병 발병 유무를 예측

- **모델 학습시키기**
  - 모델을 학습시키기 위해서 fit() 함수를 사용
  - 첫번째 인자 : 입력 변수; 8개의 속성 값을 담고 있는 X를 입력
  - 두번째 인자 : 출력 변수; 라벨값(정답) 결과 값을 담고 있는 Y를 입력
  - epochs : 전체 훈련 데이터셋에 대해 학습 반복 횟수를 지정; 1500번을 반복적으로 학습
  - batch_size : 가중치를 업데이트할 배치 크기를 의미하며, 64개로 지정 (700데이터 나눠서 학습)

```
[18] model.fit(x_train, y_train, epochs=1500, batch_size=64)

     700/700 [==============================] - 0s 27us/step - loss: 0.4454 - accuracy: 0.7914
     Epoch 727/1500
     700/700 [==============================] - 0s 25us/step - loss: 0.4442 - accuracy: 0.7914
     Epoch 728/1500
     700/700 [==============================] - 0s 29us/step - loss: 0.4479 - accuracy: 0.7986
```

- **모델 평가하기**
  - 테스트 셋으로 학습한 모델을 평가

```
[20] scores = model.evaluate(x_test, y_test)
     print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))

     68/68 [==============================] - 0s 165us/step
     accuracy: 80.88%
```

# 다층 퍼셉트론 모델 만들어보기 - 폐암 수술 환자의 생존유무 예측하기

- **문제 정의하기**
  - 폴란드의 브로츠와프 의과대학에서 2013년 공개한 폐암 수술 환자의 수술 전 진단 데이터와 수술 후 생존 결과를 기록한 실제 의료 기록 데이터
  - 이진 분류 예제에 적합한 데이터셋은 17개 변수(종양의 유형, 폐활량, 호흡곤란 여부,기침, 흡연, 천식여부 등의 17가지 환자 상태), 18번째는 수술 후 생존 결과로 1이면 생존, 0이면 사망 폐암환자 수술 예측 데이터 셋
  - 17개 변수를 독립변수로 보고 생존 유무를 예측하는 이진 분류 문제로 정의
- **데이터 준비하기**
- https://www.kaggle.com/sojinoh/thoraricsurgery-dataset
- **데이터셋 생성하기**
  - 입력(속성값 17개)와 출력(판정결과 1개) 변수 학습 데이터: 420건, 테스트 데이터: 50건 → 총 데이터 470건
- **모델 구성하기**
  - Dense 레이어만을 사용하여 다층 퍼셉트론 모델을 구성데이터 준비하기
  - 속성이 17개이기 때문에 입력 뉴런을 17개이고, 퍼셉트론의 개수 30, 활성화 함수 relu ,
    두번째 Dense 레이어 뉴련 퍼셉트론의 개수 20, 활성화 함수 relu ,
    세번째 Dense 레이어 뉴련 퍼셉트론의 개수 10, 활성화 함수 relu ,
    마지막 출력 Dense 레이어 이진 분류이기 때문에 0~1사이의 값을 나타내는 출력 뉴런이 1개 활성화 함수 sigmoid

# 다층 퍼셉트론 모델 만들어보기 - 폐암 수술 환자의 생존유무 예측하기

- **모델 학습과정 설정하기**
  - 모델을 정의했다면 모델을 손실함수와 최적화 알고리즘(하이퍼파라미터) 설정
  - loss : 이진 클래스 문제이므로 'binary_crossentropy'으로 지정
  - optimizer : 효율적인 경사 하강법 알고리즘 중 하나인 'adam'을 사용
  - metrics : 평가 척도를 나타내며 분류 문제에서는 일반적으로 'accuracy'으로 지정

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

- **모델 학습시키기** epochs=100, batch_size=64

```
[ ]  history= model.fit(x_train, y_train, epochs=100, batch_size=64)

     Epoch 1/100
     7/7 [==============================] - 0s 2ms/step - loss: 42.0801 - accuracy: 0.1429
     Epoch 2/100
     7/7 [==============================] - 0s 2ms/step - loss: 26.7980 - accuracy: 0.1667
     Epoch 3/100
     7/7 [==============================] - 0s 3ms/step - loss: 13.3884 - accuracy: 0.2405
     Epoch 4/100
     7/7 [==============================] - 0s 2ms/step - loss: 2.6262 - accuracy: 0.5524
```

- **모델 평가하기**

```
[ ]  scores = model.evaluate(x_test, y_test)
     print("%s: %.2f%%" %(model.metrics_names[1], scores[1]*100))

     2/2 [==============================] - 0s 1ms/step - loss: 0.5632 - accuracy: 0.7800
     accuracy: 78.00%
```

# Thank you for your attention