

## Chapter 34. Beginning of DeepLearning

---



# Tensorflow 란?

Tensorflow란?

## 텐서플로우 소개



- 머신러닝을 위한 오픈소스 플랫폼 - 딥러닝 프레임워크
- 구글이 주도적으로 개발 - 구글 코랩에는 기본 장착
- 최근 2.x 버전이 발표
- Keras라고 하는 고수준 API를 병합

Tensorflow란?

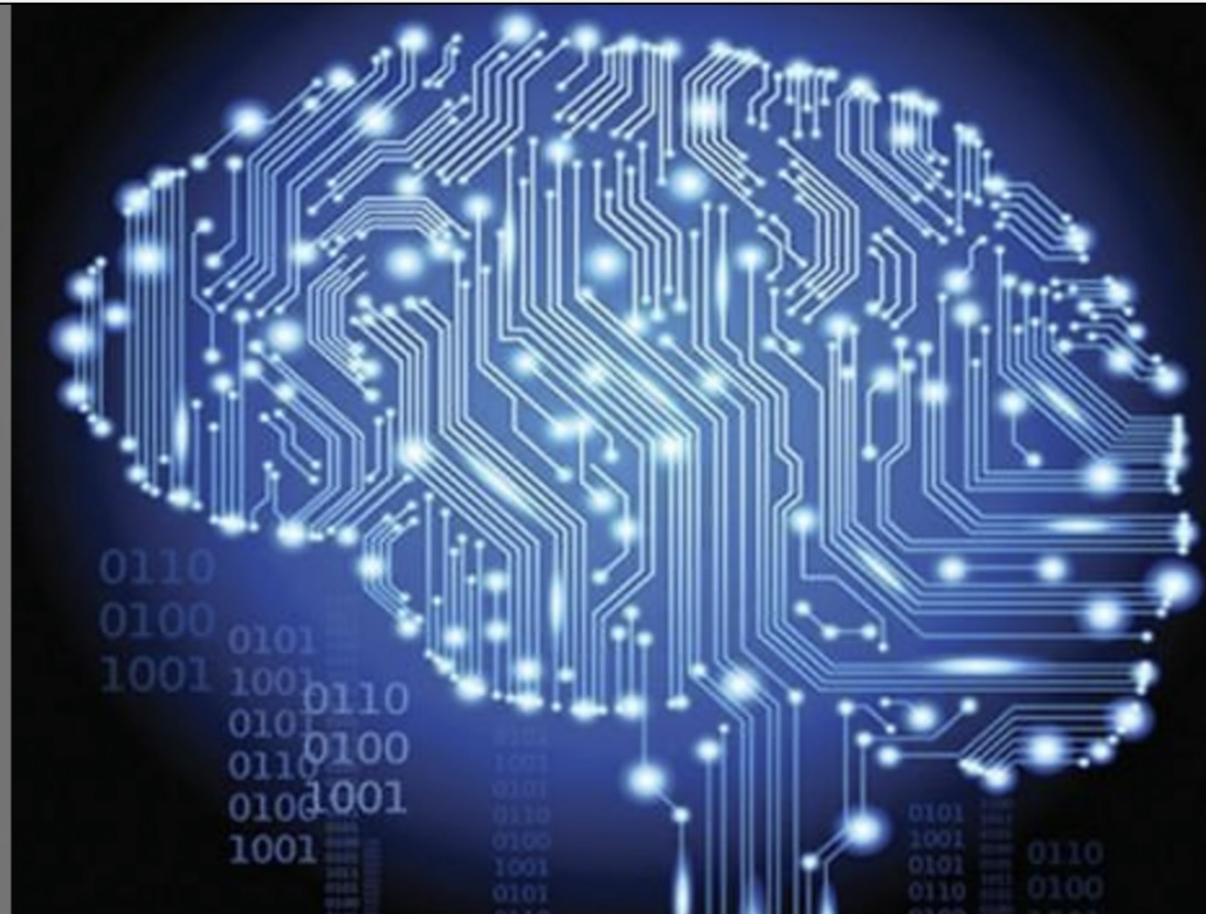
## 텐서플로우?

- Tensor : 벡터나 행렬을 의미
- Graph : 텐서가 흐르는 경로(혹은 공간)
- Tensor Flow : 텐서가 Graph를 통해 흐른다~

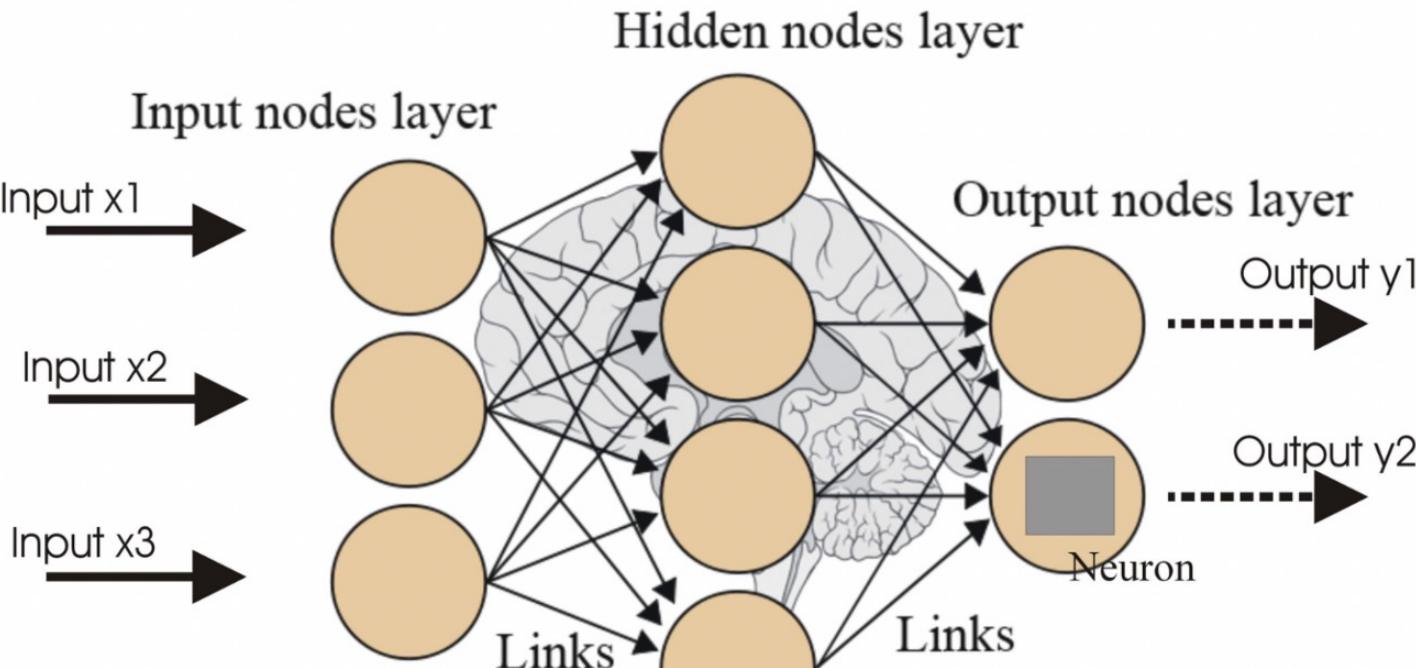
# 딥러닝의 기초 feat. Keras

딥러닝의 기초  
feat. Keras

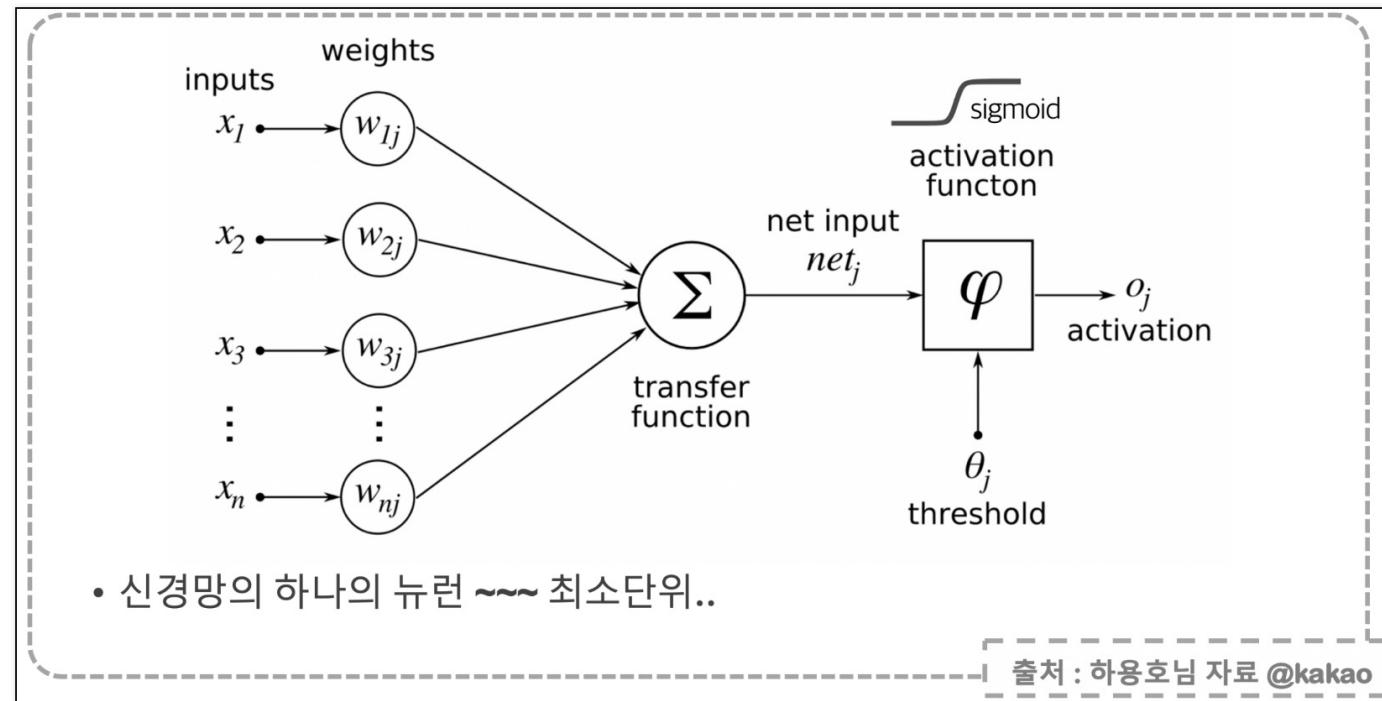
# Deep Learning and Keras



## 신경망에서 아이디어를 얻어서 시작된 Neural Net

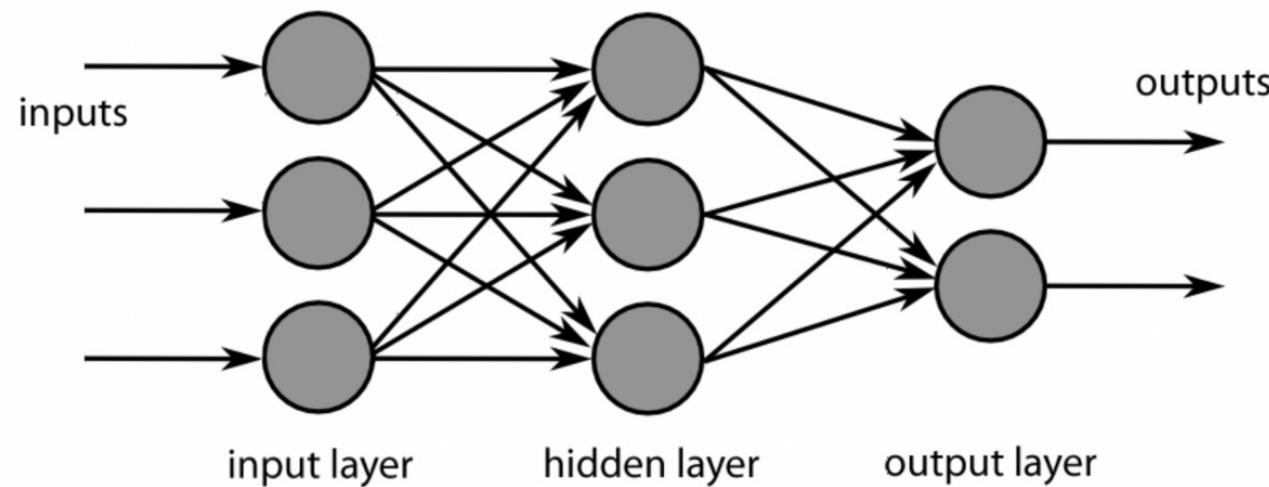


## 뉴런



- 뉴런은 입력, 가중치, 활성화함수, 출력으로 구성
- 뉴런에서 학습할 때 변하는 것은 가중치. 처음에는 초기화를 통해 랜덤값을 넣고, 학습과정에서 일정한 값으로 수렴

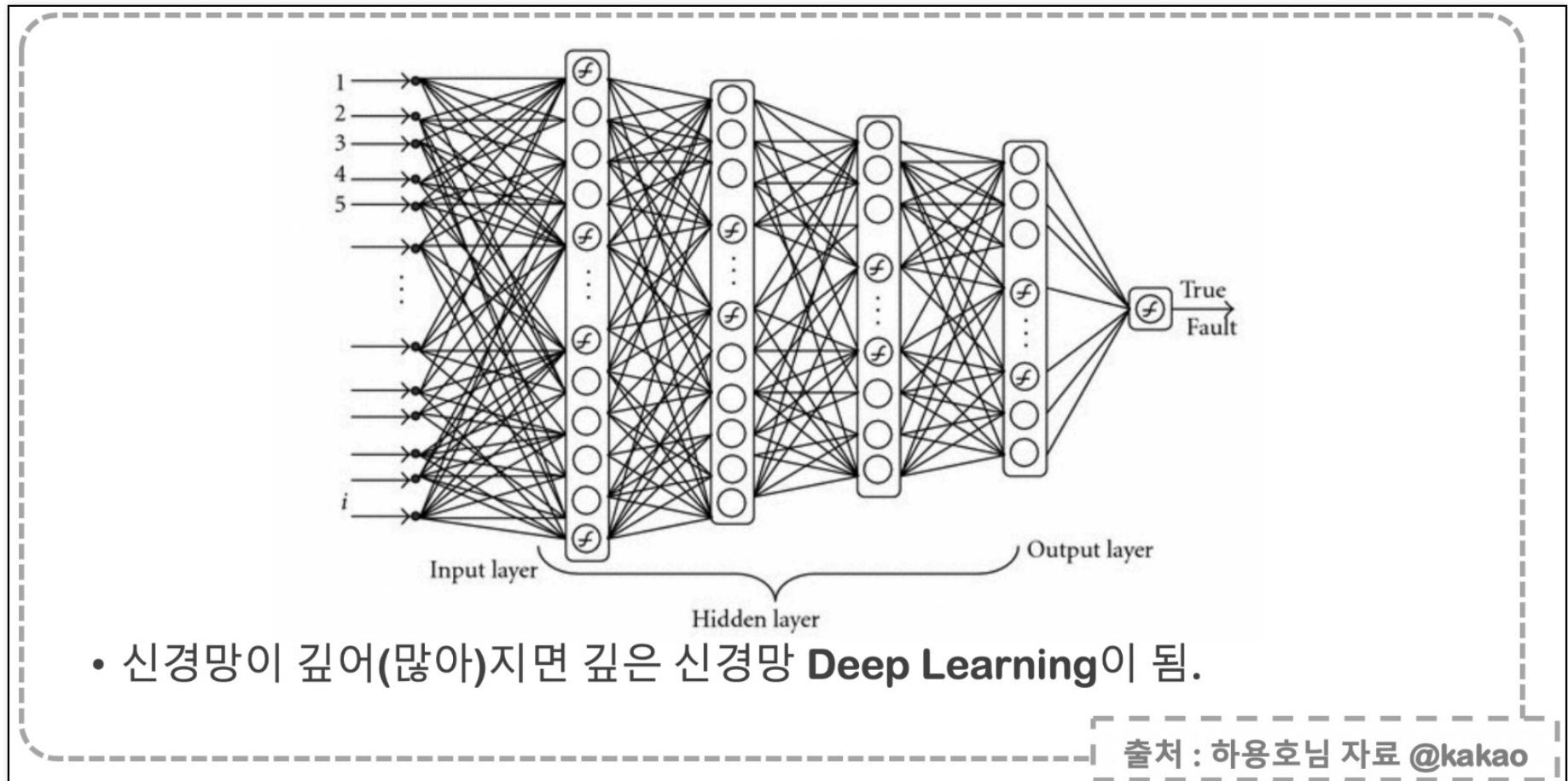
## 레이어와 망(net)



- 뉴런이 모여서 **layer**를 구성하고, 망(**net**)이 됨.

출처 : 하용호님 자료 @kakao

딥러닝~



## 데이터 하나 무작정 읽어보자

```
import numpy as np

raw_data = np.genfromtxt('x09.txt', skip_header=36)
raw_data
```

```
array([[ 1.,  1.,  84.,  46., 354.],
       [ 2.,  1.,  73.,  20., 190.],
       [ 3.,  1.,  65.,  52., 405.],
       [ 4.,  1.,  70.,  30., 263.],
       [ 5.,  1.,  76.,  57., 451.],
       [ 6.,  1.,  69.,  25., 302.],
       [ 7.,  1.,  63.,  28., 288.],
       [ 8.,  1.,  72.,  36., 385.],
       [ 9.,  1.,  79.,  57., 402.],
```

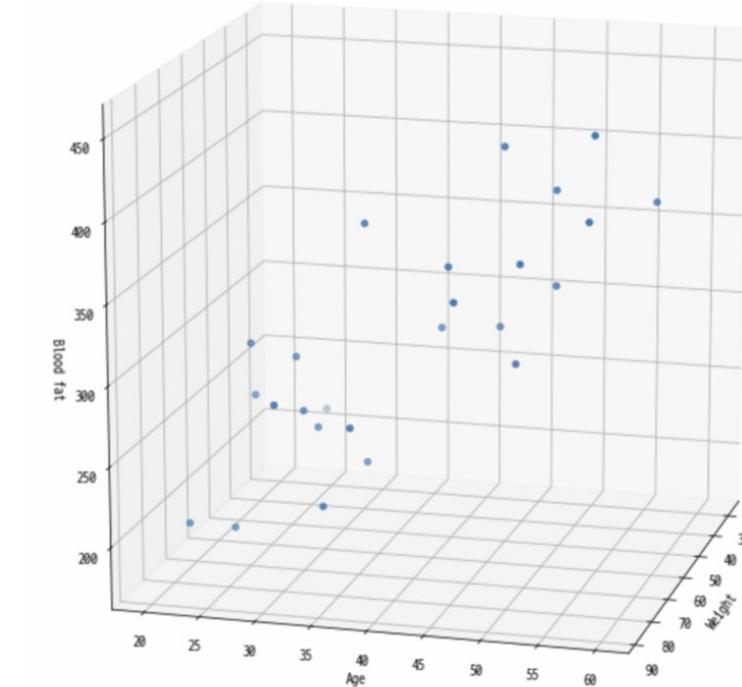
## 어떻게 생겼나?

```
| from mpl_toolkits.mplot3d import Axes3D
| import matplotlib.pyplot as plt
| %matplotlib inline

xs = np.array(raw_data[:,2], dtype=np.float32)
ys = np.array(raw_data[:,3], dtype=np.float32)
zs = np.array(raw_data[:,4], dtype=np.float32)

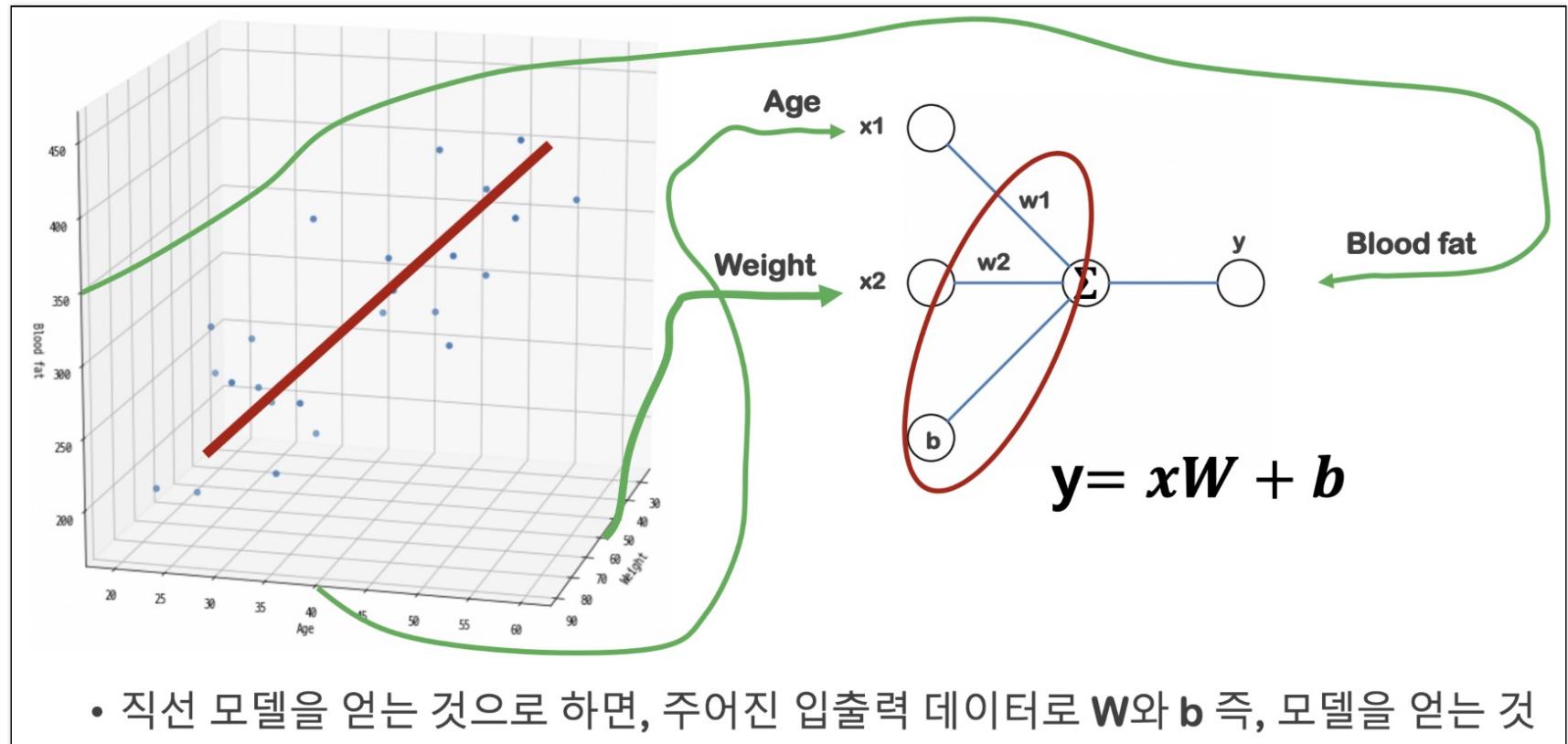
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs, ys, zs)
ax.set_xlabel('Weight')
ax.set_ylabel('Age')
ax.set_zlabel('Blood fat')
ax.view_init(15,15)
plt.show()
```

## 현재 간단한 딥러닝의 목표

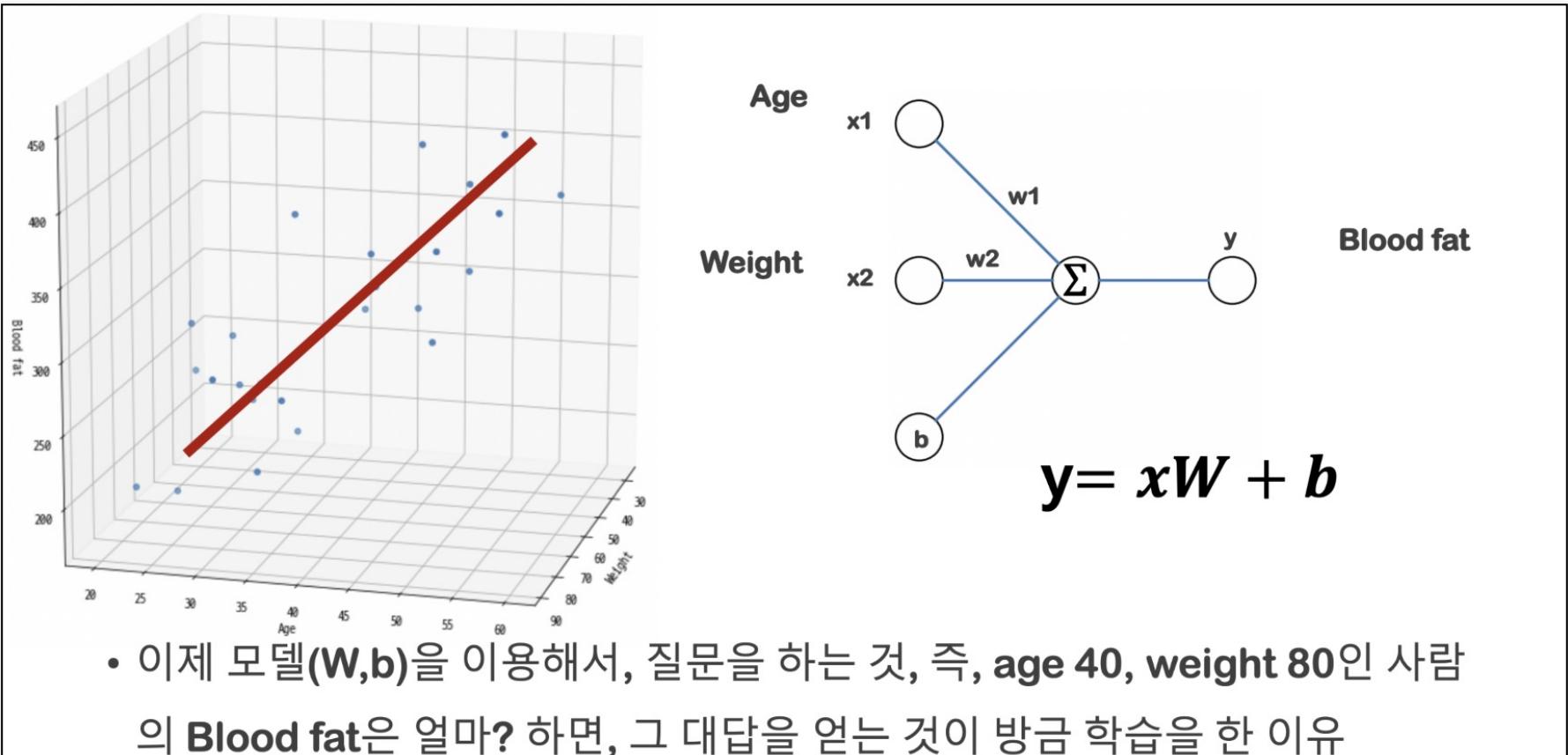


- 이 예제의 목적은 입력인 나이와 몸무게를 알려주면, 주어진 데이터 기준의 **blood fat**을 얻는 것이다.
- 즉, **40살, 100키로인 사람의 데이터 기준 blood fat은?** 하고 물으면, 얼마입니다~~~하고 답이 나와야 하는 것이다.
- **Linear Regression ~~**

## 먼저 모델을 주어진 데이터로 얻는 것

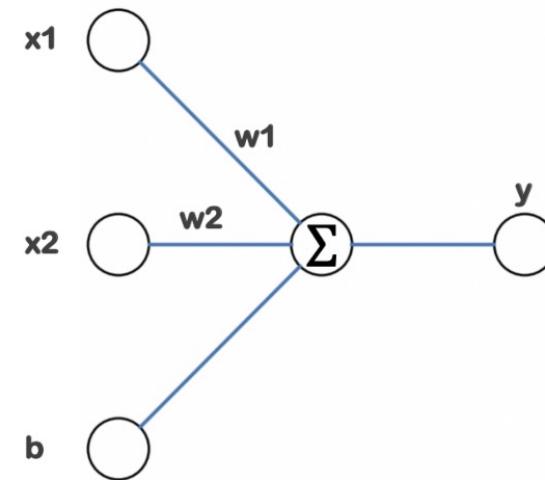


## 모델을 구한 후에는?



## 현재 우리의 목표

$$y = XW + b$$



- 목적:  $x_1, x_2$ 를 입력해서  $y$ 가 나오게 하는 **Weight**와 **bias**를 구하는 것

## 학습 대상 데이터를 추리고

```
| x_data = np.array(raw_data[:, 2:4], dtype=np.float32)
| y_data = np.array(raw_data[:, 4], dtype=np.float32)
|
| y_data = y_data.reshape((25, 1))
```

## 원래 의도한 모델을 만들자

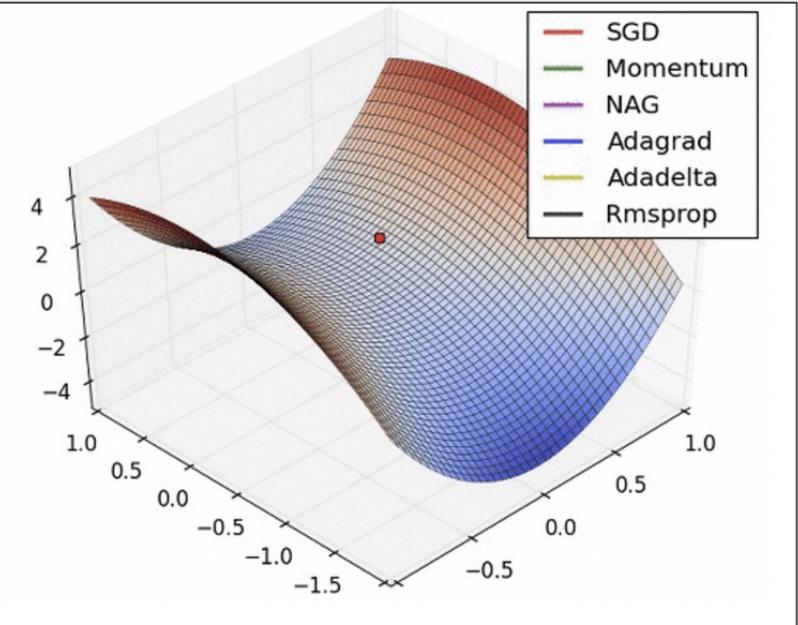
```
| model = tf.keras.models.Sequential([
|     tf.keras.layers.Dense(1, input_shape=(2, )),
| ])
|
model.compile(optimizer='rmsprop', loss='mse')
```

## loss?

- 학습을 위해서는 **loss (cost)** 함수를 정해주어야 한다
- **loss** 함수는 간략히 말해서, 정답까지 얼마나 멀리 있는지를 측정하는 함수이다.
- 이번에는 **mse:mean square error** 오차 제곱의 평균을 사용
- 그리고, 옵티마이저를 선정한다. 옵티마이저는 **loss**를 어떻게 줄일 것인지 를 결정하는 방법을 선택하는 것이다.

## optimizer?

- **optimizer**는 loss 함수를 최소화하는  
가중치를 찾아가는 과정에 대한  
알고리즘이다.
- 여기서는 **rmsprop**을 사용



## summary

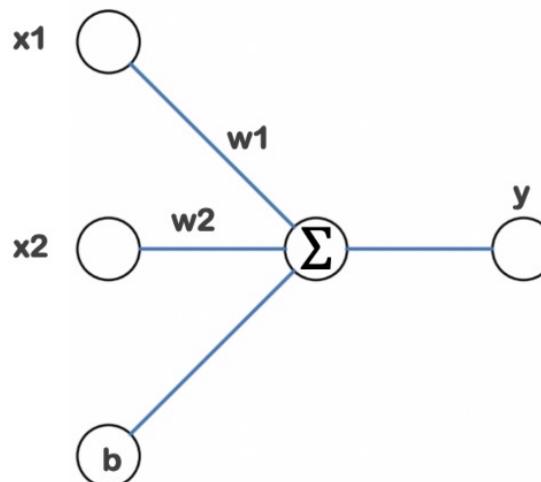
```
| model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
dense (Dense)	(None, 1)	3
<hr/>		
Total params: 3		
Trainable params: 3		
Non-trainable params: 0		
<hr/>		

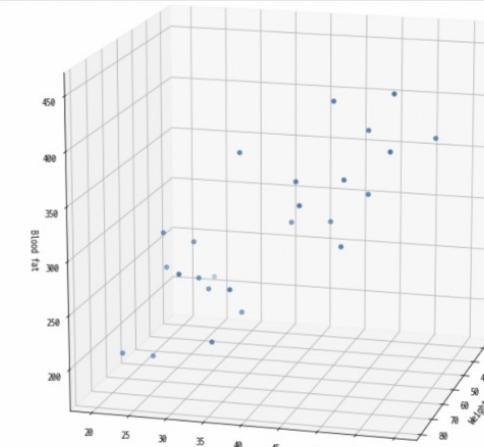
## 모델을 다 구성했다

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 1)	3
<hr/>		
Total params: 3		
Trainable params: 3		
Non-trainable params: 0		
<hr/>		



## 다음은?

- 현재 우리는 나이와 몸무게를 받아서
  - **Blood fat**을 추정하는 모델을
  - 학습을 통해 얻으려고 한다.
- 이를 위해
  - 모델(네트워크)을 구성했고
  - 모델의 **loss function**을 선정하고, **loss**의 감소를 위한 **optimizer**도 선정을 했다.
- 이제 학습을 시작해야지...~~ 어떻게??



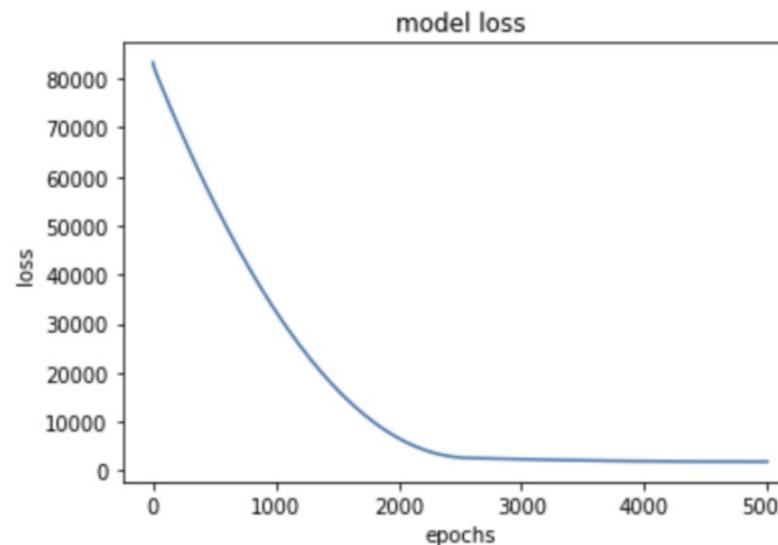
## fit~ 학습~

```
hist = model.fit(x_data, y_data, epochs=5000)

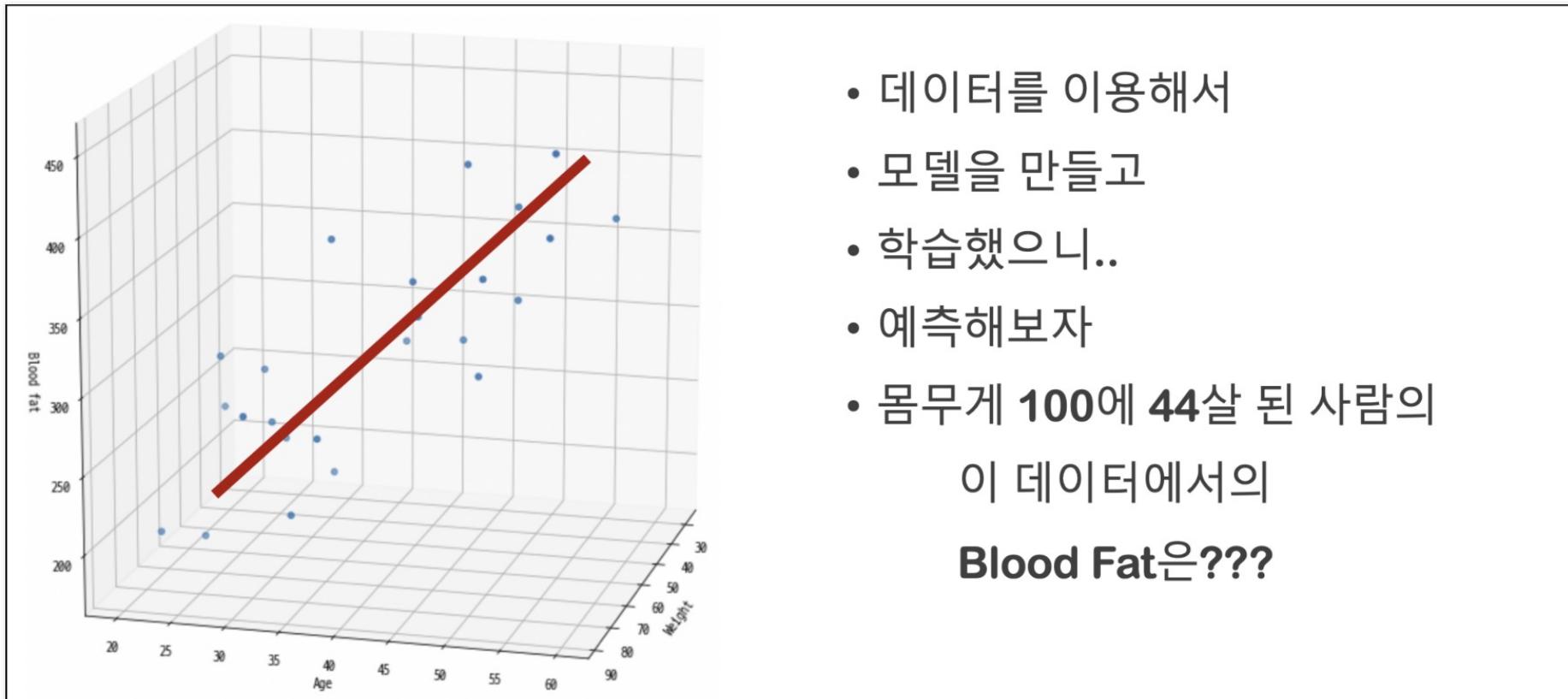
25/25 [=====] - 0s 33us/sample - loss: 79987.3906
Epoch 46/5000
25/25 [=====] - 0s 38us/sample - loss: 79925.8672
Epoch 47/5000
25/25 [=====] - 0s 32us/sample - loss: 79864.3984
Epoch 48/5000
25/25 [=====] - 0s 34us/sample - loss: 79802.9844
Epoch 49/5000
25/25 [=====] - 0s 35us/sample - loss: 79741.6094
Epoch 50/5000
25/25 [=====] - 0s 36us/sample - loss: 79680.2812
Epoch 51/5000
25/25 [=====] - 0s 37us/sample - loss: 79619.0000
```

## loss가 잘 떨어진다

```
| plt.plot(hist.history[ 'loss' ])
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epochs')
  plt.show()
```



## predict 해볼까?



## 사용법은 sklearn과 비슷하다

```
| model.predict(np.array([100, 44]).reshape(1, 2))
```

```
array([[374.87653]], dtype=float32)
```

```
| model.predict(np.array([60, 25]).reshape(1, 2))
```

```
array([[219.11299]], dtype=float32)
```

## 가중치와 bias를 알고 싶다면

```
| W_, b_ = model.get_weights()  
|     print('Weight is : ', W_)  
|     print('bias is : ', b_)
```

```
Weight is : [[1.2482319]  
[5.570223 ]]  
bias is : [4.963518]
```

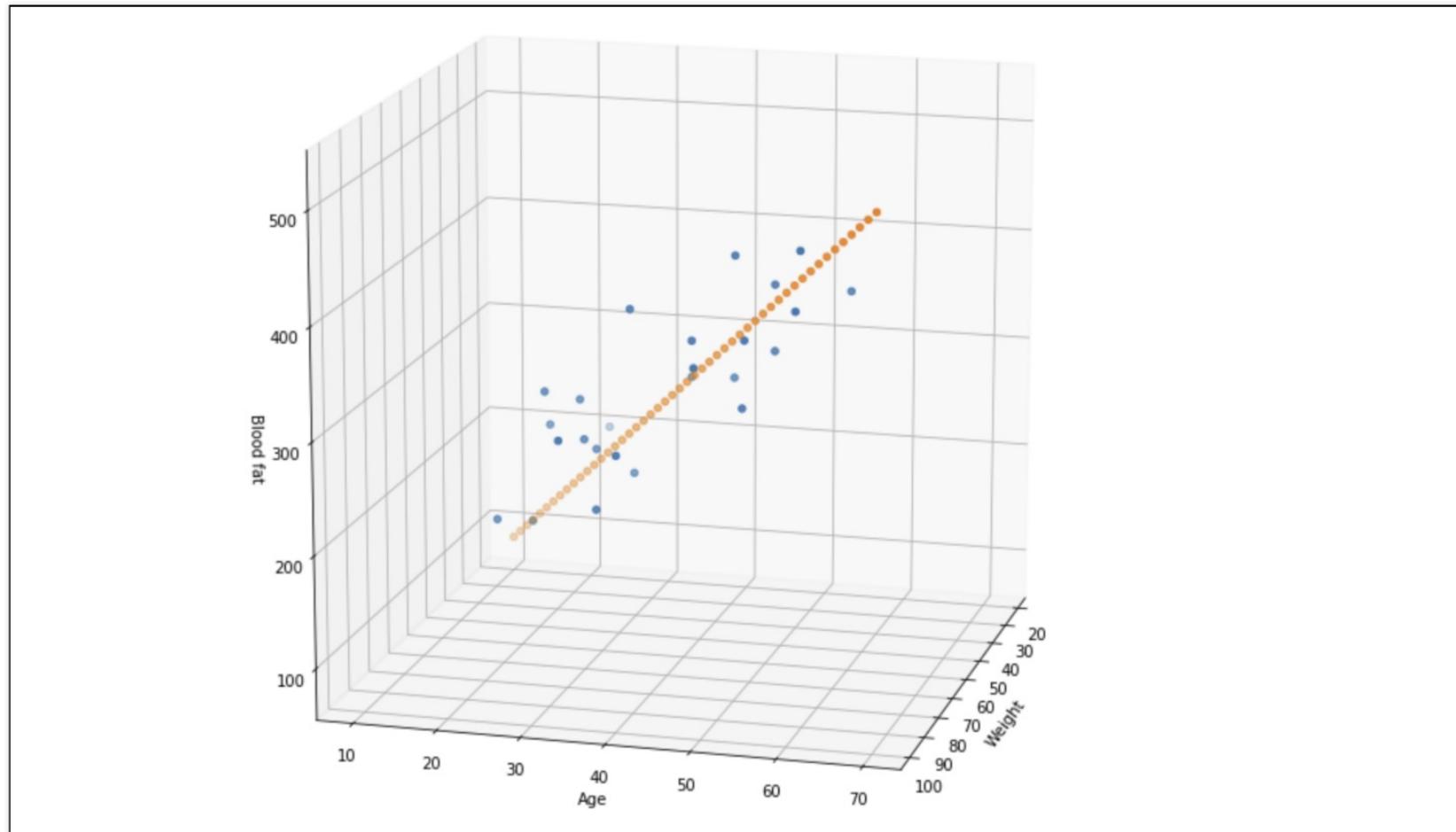
모델이 잘 만들어졌는지 확인하기 위해 데이터를 만들고

```
| x = np.linspace(20, 100, 50).reshape(50,1)
| y = np.linspace(10, 70, 50).reshape(50,1)
|
| X = np.concatenate((x, y), axis=1)
| Z = np.matmul(X, W_) + b_
```

## 그리기를 시도하자

```
| fig = plt.figure(figsize=(12,12))
| ax = fig.add_subplot(111, projection='3d')
| ax.scatter(xs, ys, zs)
| ax.scatter(x, y, Z)
| ax.set_xlabel('Weight')
| ax.set_ylabel('Age')
| ax.set_zlabel('Blood fat')
| ax.view_init(15,15)
| plt.show()
```

나쁘지 않다



# XOR 문제

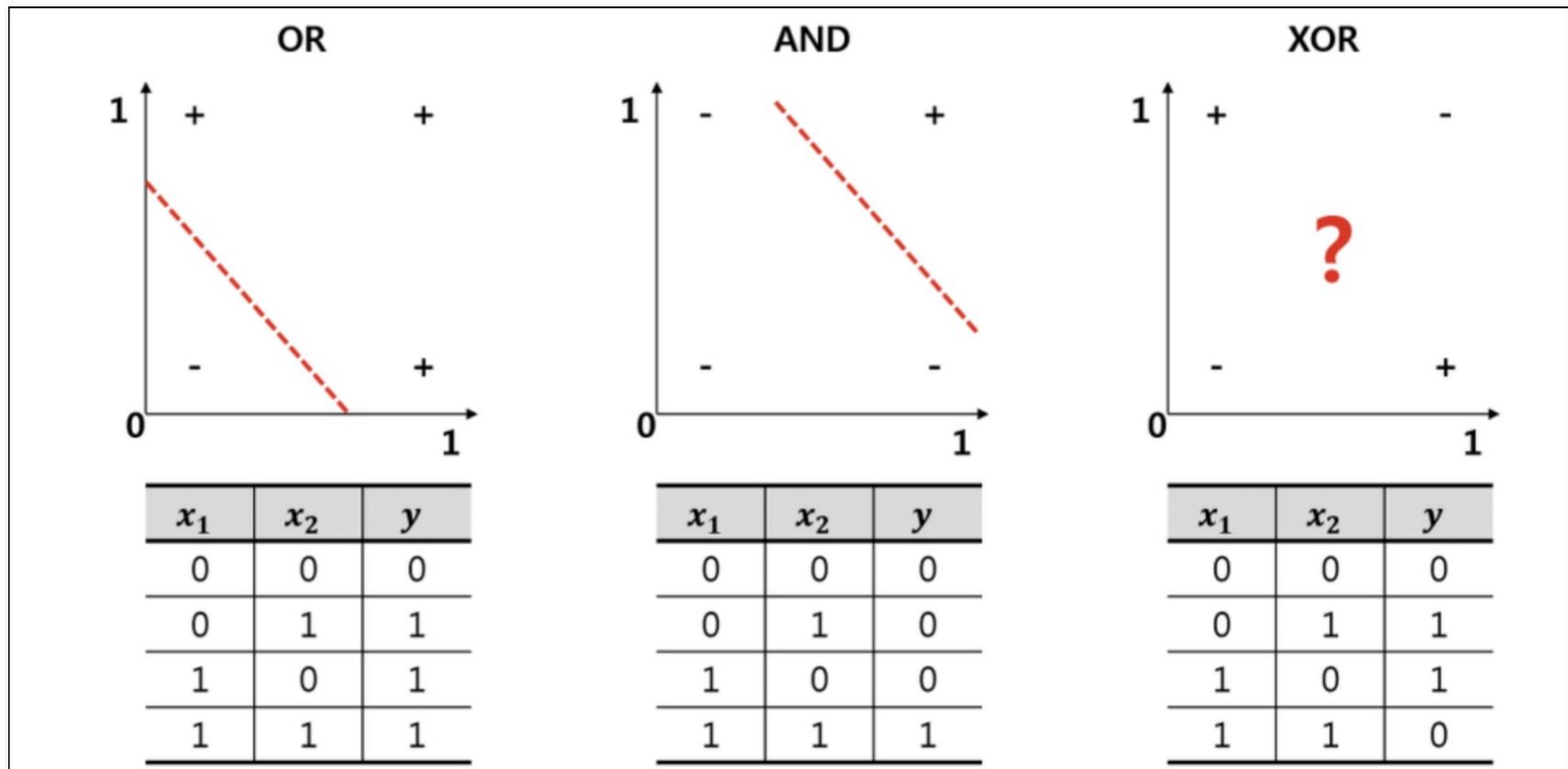
## XOR 문제

## XOR

기호	논리식	진리표																	
	$F = A \oplus B$	<table border="1"><thead><tr><th>입력</th><th>출력</th></tr><tr><th>A</th><th>B</th><th>F</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	입력	출력	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
입력	출력																		
A	B	F																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	

## XOR 문제

선형 모델로는 XOR를 풀 수가 없다



## XOR 문제

간단히 데이터를 준비하고

```
| import numpy as np  
  
X = np.array([ [0, 0],  
              [1, 0],  
              [0, 1],  
              [1, 1] ])  
y = np.array([[0], [1], [1], [0]])
```

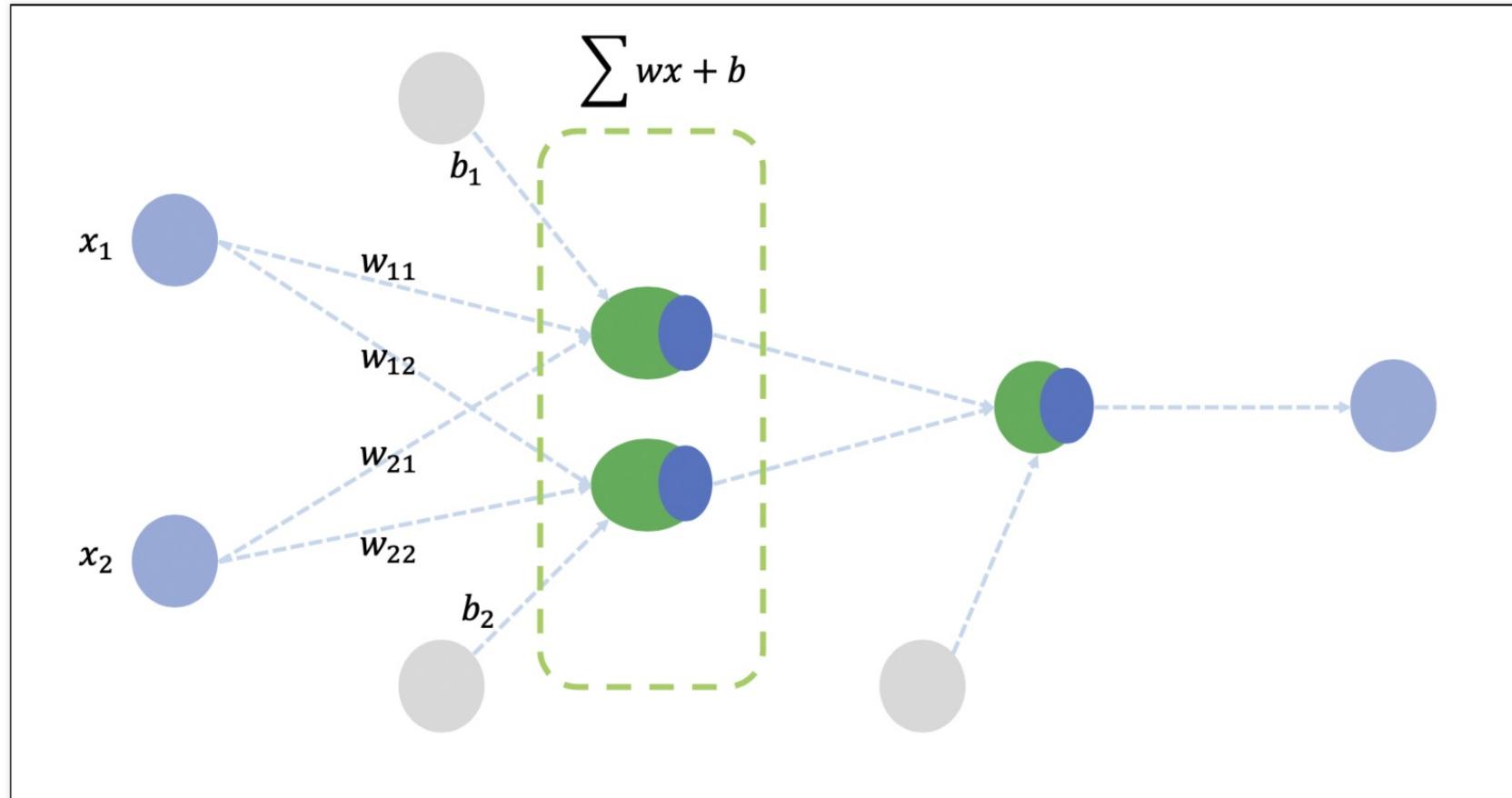
## XOR 문제

모델은 간단하

```
| model = tf.keras.Sequential([
|     tf.keras.layers.Dense(2, activation='sigmoid', input_shape=(2,)),
|     tf.keras.layers.Dense(1, activation='sigmoid')
| ])
```

## XOR 문제

위 모델은 어떻게 생겼을까



## XOR 문제

## model.compile

```
| model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')
```

- 옵티마이저를 선정하고, 학습률을 선정한다.
- loss 함수는 mse로 한다 mean squared error

## XOR 문제

## model.summary

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 2)	6
dense_1 (Dense)	(None, 1)	3
<hr/>		
Total params: 9		
Trainable params: 9		
Non-trainable params: 0		

## XOR 문제

## 학습

```
hist = model.fit(X, y, epochs=5000, batch_size=1)

Train on 4 samples
Epoch 1/5000
4/4 [=====] - 0s 33ms/sample - loss: 0.3187
Epoch 2/5000
4/4 [=====] - 0s 1ms/sample - loss: 0.3131
Epoch 3/5000
4/4 [=====] - 0s 1ms/sample - loss: 0.3077
Epoch 4/5000
4/4 [=====] - 0s 1ms/sample - loss: 0.3023
```

- epochs는 지정된 횟수만큼 학습을 하는 것
- batch\_size는 한번의 학습에 사용될 데이터의 수를 지정

## XOR 문제

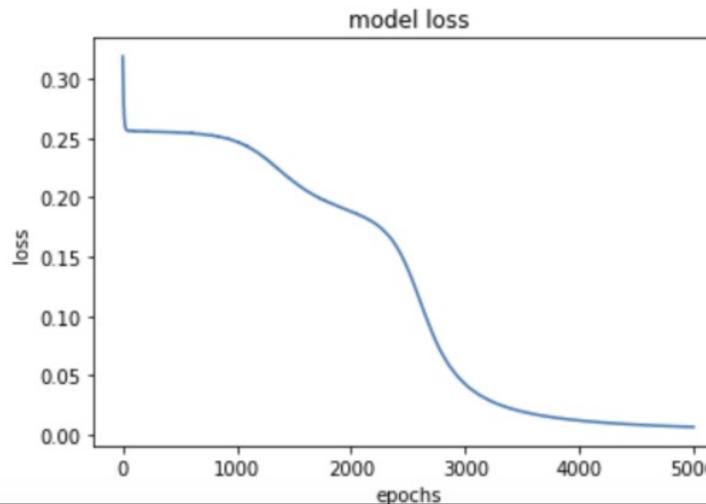
## 학습결과

```
| model.predict(X)  
  
array([[ 0.07790035],  
       [ 0.9116049 ],  
       [ 0.9153757 ],  
       [ 0.07007294]], dtype=float32)
```

## XOR 문제

## loss 상황

```
import matplotlib.pyplot as plt  
%matplotlib inline  
  
plt.plot(hist.history['loss'])  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epochs')  
plt.show()
```



## XOR 문제

## 학습에서 찾은 가중치

```
| for w in model.weights:  
|     print('---')  
|     print(w)  
  
---  
<tf.Variable 'dense/kernel:0' shape=(2, 2) dtype=float32, numpy=  
array([[-4.84964   ,  6.2534823],  
       [ 5.1338177, -6.2427473]], dtype=float32)>  
---  
<tf.Variable 'dense/bias:0' shape=(2,) dtype=float32, numpy=array([2.4333632,  
3.2707038], dtype=float32)>  
---  
<tf.Variable 'dense_1/kernel:0' shape=(2, 1) dtype=float32, numpy=  
array([[-5.9917865],  
      [-5.829812 ]], dtype=float32)>  
---  
<tf.Variable 'dense_1/bias:0' shape=(1,) dtype=float32, numpy=array([8.65374  
3], dtype=float32)>
```

이번에는 분류로~

이번에는 분류로~

## iris 데이터

```
| from sklearn.datasets import load_iris  
  
iris = load_iris()  
  
X = iris.data  
y = iris.target
```

이번에는 분류로~

그런데 y는 이렇게 생겼다

```
| y  
array([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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
      2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

이번에는 분류로~

## One hot encoding

	A	B	C	D	E	F	G	H	I
1	Original data:		One-hot encoding format:						
2	id	Color		id	White	Red	Black	Purple	Gold
3	1	White		1	1	0	0	0	0
4	2	Red		2	0	1	0	0	0
5	3	Black		3	0	0	1	0	0
6	4	Purple		4	0	0	0	1	0
7	5	Gold		5	0	0	0	0	1
8									

이번에는 분류로~

## sklearn의 one hot encoding

```
| from sklearn.preprocessing import OneHotEncoder  
  
enc = OneHotEncoder(sparse=False, handle_unknown='ignore')  
enc.fit(y.reshape(len(y), 1))  
  
OneHotEncoder(categories='auto', drop=None, dtype=<class 'numpy.float64'>,  
              handle_unknown='ignore', sparse=False)
```

이번에는 분류로~

## 이제 학습 준비가 되었다

```
enc.categories_
[array([0, 1, 2])]

y_onehot = enc.transform(y.reshape(len(y), 1))
y_onehot

array([[1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

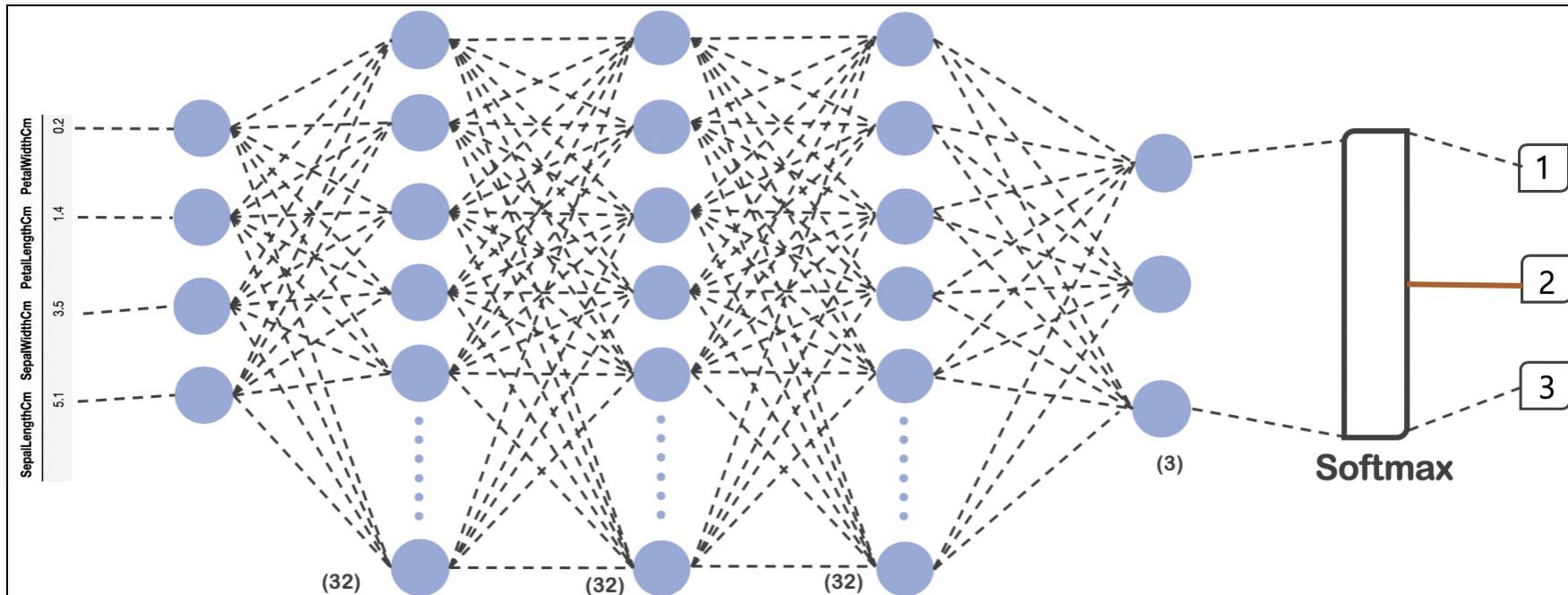
이번에는 분류로~

## 데이터 나누고

```
| from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot,  
                                                 test_size=0.2,  
                                                 random_state=13)
```

이번에는 분류로~

난 망(net)을 이렇게 구성하기로 했다



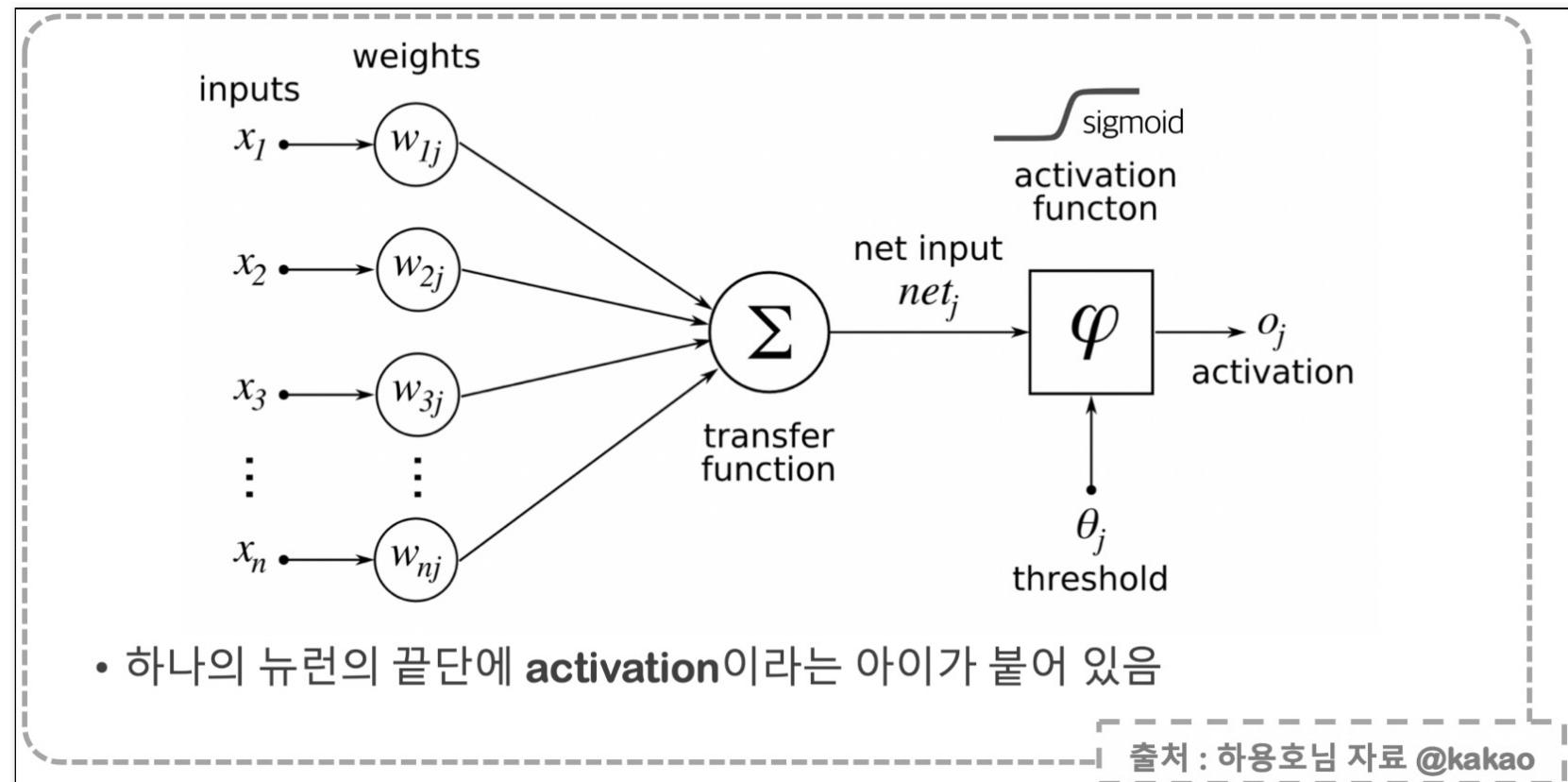
이번에는 분류로~

## 이렇게 하면 된다

```
| model = tf.keras.models.Sequential([
|     tf.keras.layers.Dense(32, input_shape=(4, ), activation='relu'),
|     tf.keras.layers.Dense(32, activation='relu'),
|     tf.keras.layers.Dense(32, activation='relu'),
|     tf.keras.layers.Dense(3, activation='softmax'),
| ])
```

이번에는 분류로~

## activation



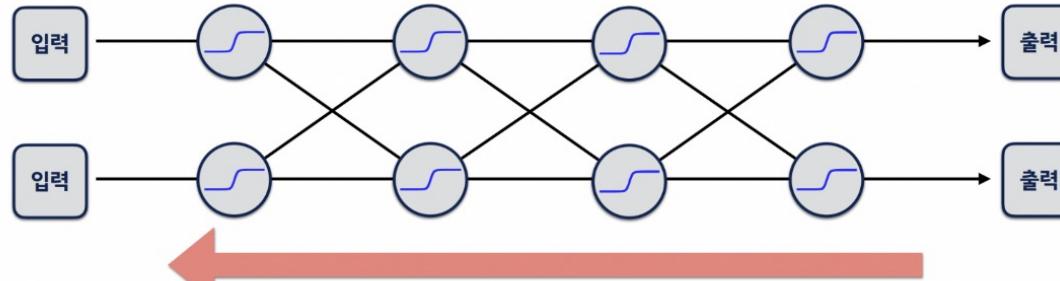
이번에는 분류로~

## 역전파 back-propagation

### 뉴럴넷의 학습방법 Back propagation

(사실 별거 없고 그냥 “뒤로 전달”)

뭐를 전달하는가?  
현재 내가 틀린정도를 ‘미분(기울기)’ 한 거



미분하고, 곱하고, 더하고를 역방향으로 반복하며 업데이트한다.

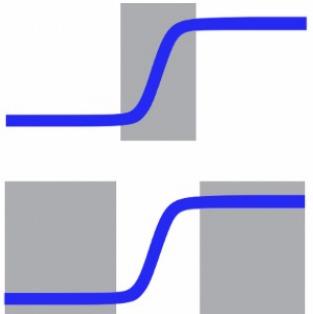
출처 : 하용호님 자료 @kakao

이번에는 분류로~

## 역전파에서는 sigmoid가 문제가 있다

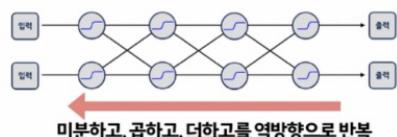
### 근데 문제는?

우리가 activation 함수로 sigmoid  를 썼다는 것



여기의 미분(기울기)는 뭐라도 있다. 다행

근데 여기는 기울기 0.. 이런거 중간에 곱하면 뭔가 뒤로 전달할게 없다?!



그런 상황에서 이걸 반복하면??????

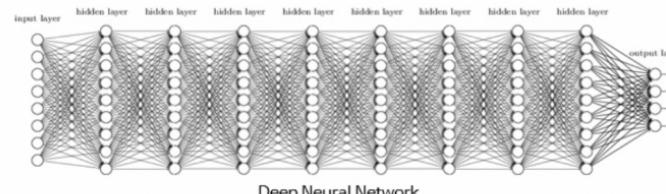
출처 : 하용호님 자료 @kakao

이번에는 분류로~

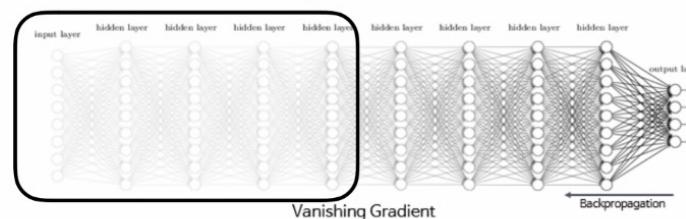
## gradient vanishing

**Vanishing gradient 현상 :** 레이어가 깊을 수록 업데이트가 사라져간다.  
그래서 fitting이 잘 안됨(underfitting)

끌줄학생 ← 출.. 출.. 출.. 출.. 출.. 맞추자 → 교장샘



학습이 잘 안됨



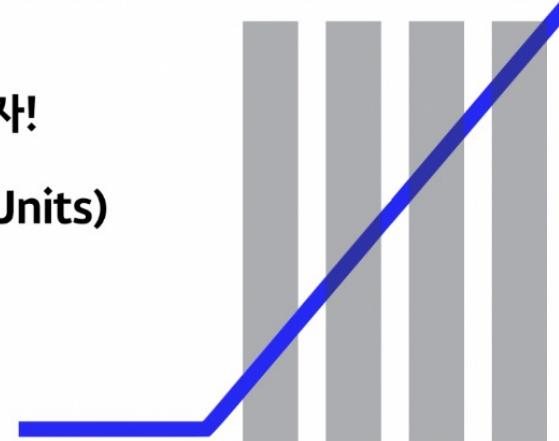
출처 : 하용호님 자료 @kakao

이번에는 분류로~

## ReLU의 등장

사그라드는 sigmoid 대신  
죽지 않는 activation func 을 쓰자!

## → **ReLU** (Rectified Linear Units)



이녀석은 양의 구간에서 전부 미분값(1)이 있다!

A horizontal scale diagram illustrating the concept of '줄맞추기' (fitting in). It features five student icons arranged along a line. On the far left is the label '끌줄학생' (Student who pulls a line) with an arrow pointing towards the center. In the middle is the label '줄맞추기' (Fitting in) with an arrow pointing towards the center. On the far right is the label '교장샘' (Principal) with an arrow pointing towards the center.

끝줄 학생까지 이야기가 전달이 잘 되고 위치를 고친다!

출처 : 하용호님 자료 @kakao

이번에는 분류로~

softmax?

## 뉴런넷에게 답을 회신받는 3가지 방법

Value

이게얼마가될거같니?

output을  
그냥 받는다.

O/X

기냐? 아니냐?

output에  
sigmoid를 먹인다.

Category

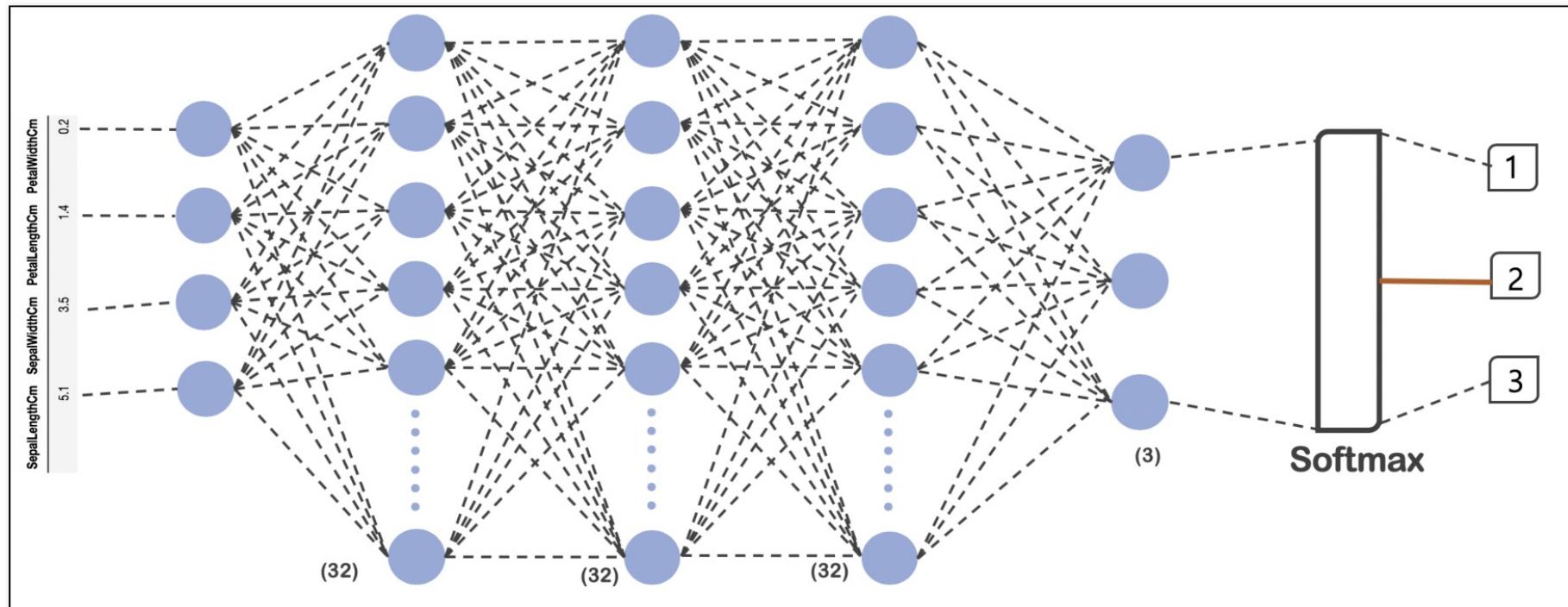
종류중에요건뭐나?

output에  
softmax를 먹인다.

출처 : 하용호님 자료 @kakao

이번에는 분류로~

## 완성된 모델



이번에는 분류로~

## adam?

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

이번에는 분류로~

gradient decent는 배웠다

기존 뉴럴넷이 가중치 parameter들을  
최적화(optimize)하는 방법

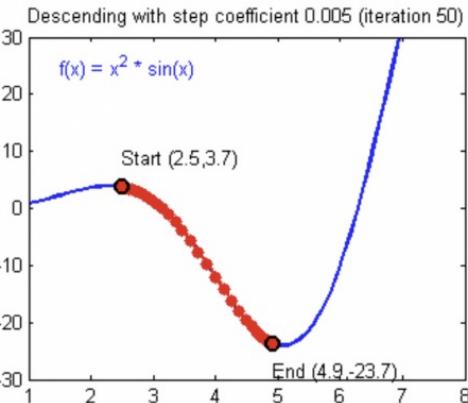
## Gradient Decent

loss function의 현 가중치에서의 기울기(gradient)를 구해서  
loss를 줄이는 방향으로 업데이트해 나간다.

출처 : 하용호님 자료 @kakao

이번에는 분류로~

## 복습차원에서~



**뉴럴넷은 loss(or cost) function을 가지고 있습니다. 쉽게 말하면 “틀린 정도”**

현재 가진 weight 세팅(내 자리)에서,  
내가 가진 데이터를 다 넣으면  
전체 에러가 계산됩니다.

거기서 미분하면 에러를 줄이는 방향을 알 수 있습니다.  
(내자리의 기울기 \* 반대방향)

그 방향으로 정해진 스텝량(learning rate)을  
곱해서 weight을 이동시킵니다. 이걸 반복~~

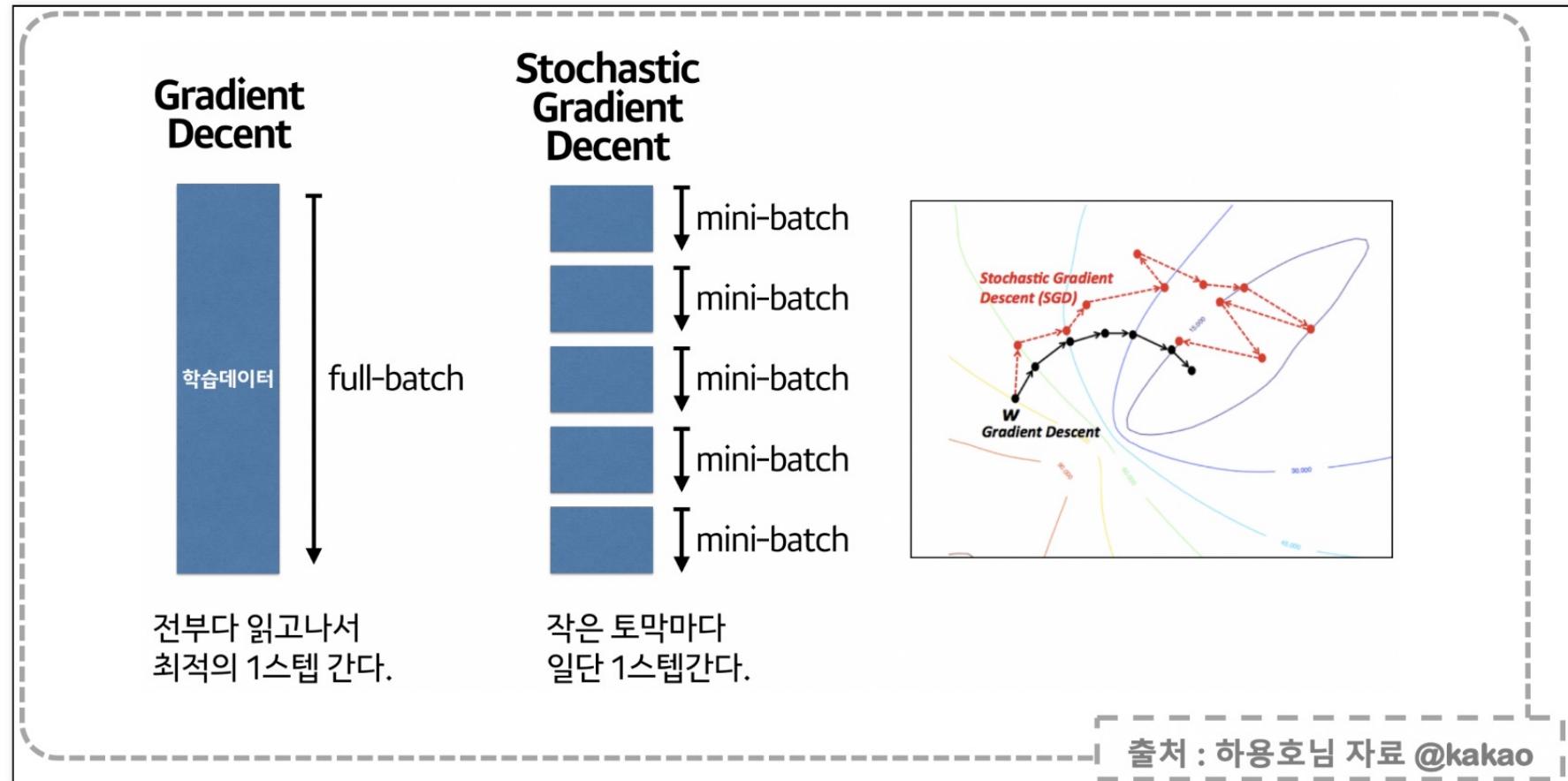
weight의 업데이트 =  $\text{에러 낮추는 방향} \times \text{한발자국 크기} \times \text{현 지점의 기울기}$

$$-\gamma \nabla F(\mathbf{a}^n)$$

출처 : 하용호님 자료 @kakao

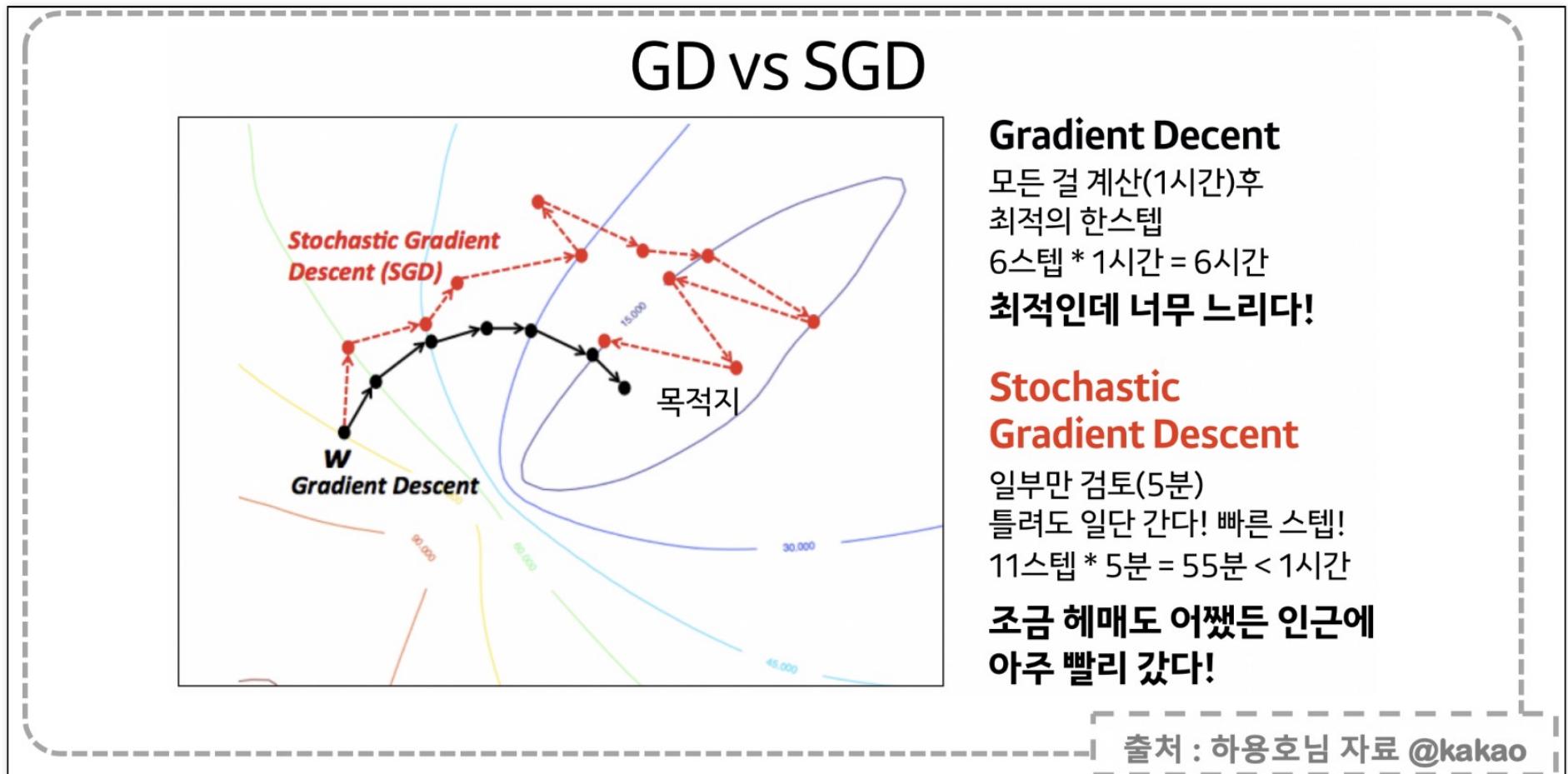
이번에는 분류로~

## SGD



이번에는 분류로~

## GD vs SGD



이번에는 분류로~

## 옵티マイ저의 선택

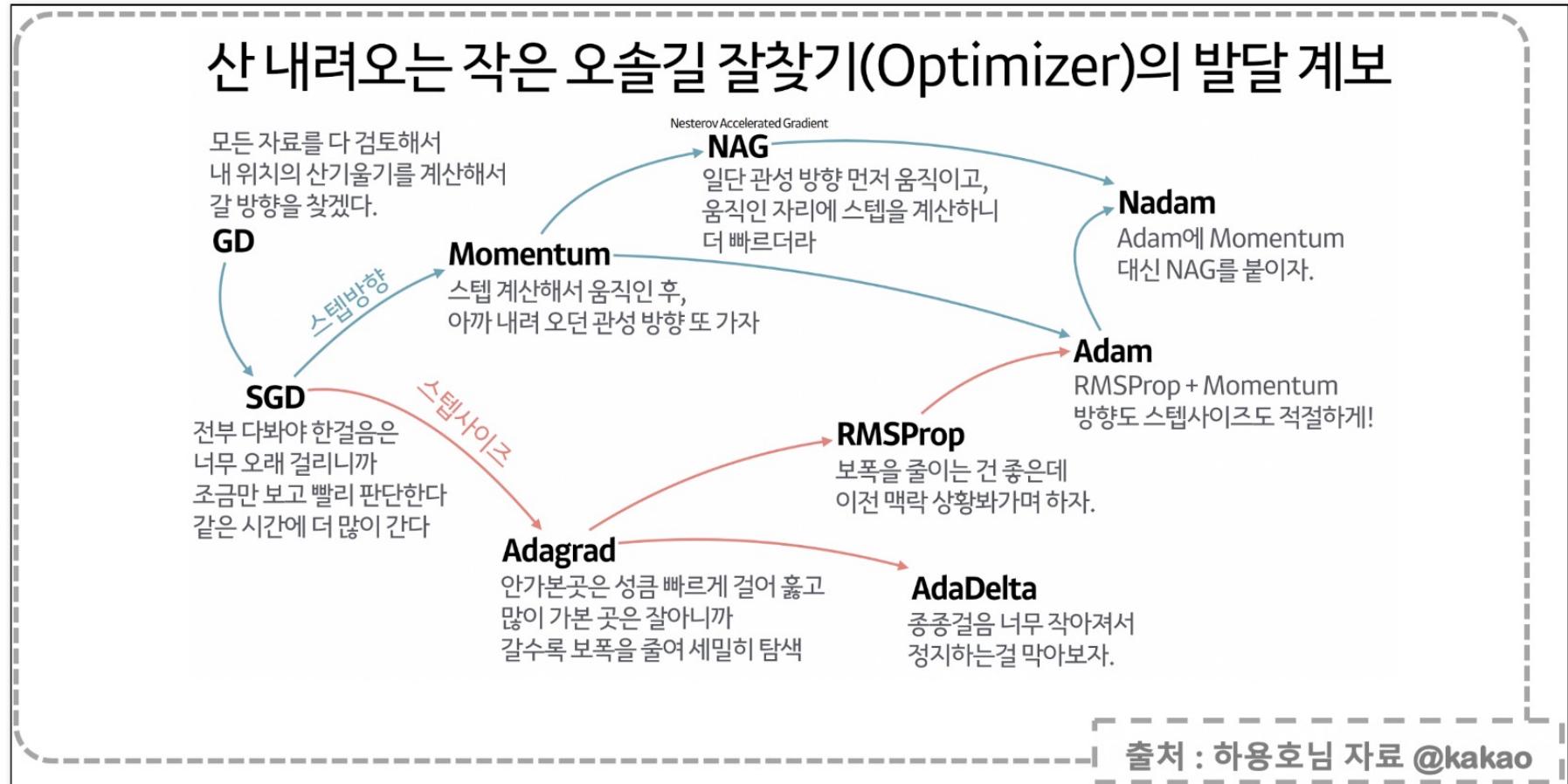
–  $\gamma \nabla F(\mathbf{a}^n)$  산을 잘 타고 내려오는 것은  
 $\nabla F(\mathbf{a}^n)$  어느 방향으로 발을 디딜지  
 $\gamma$  얼마 보폭으로 발을 디딜지  
두 가지를 잘 잡아야 빠르게 타고 내려온다.

SGD를 더 개선한 멋진 optimizer가 많다!  
SGD의 개선된 후계자들

출처 : 하용호님 자료 @kakao

이번에는 분류로~

## 옵티마이저 계보



이번에는 분류로~

데이터가 복잡할 때는 일단 Adam을 써보자

잘 모르겠으면 Adam!

출처 : 하용호님 자료 @kakao

이번에는 분류로~

## summary 결과

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 32)	160
dense_2 (Dense)	(None, 32)	1056
dense_3 (Dense)	(None, 32)	1056
dense_4 (Dense)	(None, 3)	99
<hr/>		
Total params: 2,371		
Trainable params: 2,371		
Non-trainable params: 0		
<hr/>		

이번에는 분류로~

## 학습

```
| hist = model.fit(X_train, y_train, epochs=100)

Epoch 24/100
120/120 [=====] - 0s 42us/sample - loss: 0.3268 -
accuracy: 0.9667
Epoch 25/100
120/120 [=====] - 0s 41us/sample - loss: 0.3122 -
accuracy: 0.9750
Epoch 26/100
120/120 [=====] - 0s 43us/sample - loss: 0.2982 -
accuracy: 0.9750
```

이번에는 분류로~

## test 데이터에 대한 accuracy

```
| model.evaluate(X_test, y_test, verbose=2)
```

```
30/30 - 0s - loss: 0.0960 - accuracy: 0.9667
```

```
[0.0960172712802887, 0.96666664]
```

이번에는 분류로~

## loss와 acc의 변화

```
| plt.plot(hist.history['loss'])
| plt.plot(hist.history['accuracy'])
| plt.title('model loss')
| plt.ylabel('loss')
| plt.xlabel('epochs')
| plt.show()
```

