

Neural Network?

PinkLAB Edu

20 November, 2025

Table of Contents

1	Tensorflow란?	5
1.1	텐서플로우 소개	5
1.2	텐서플로우?	5
1.3	텐서플로우 설치	5
2	딥러닝의 기초 feat. Keras.....	7
2.1	신경망에서 아이디어를 얻어서 시작된 Neural Net	7
2.2	뉴런.....	8
2.3	레이어와 망(net)	8
2.4	딥러닝~.....	9
2.5	데이터 하나 무작정 읽어보자	9
2.6	어떻게 생겼나?	10
2.7	현재 간단한 딥러닝의 목표.....	10
2.8	먼저 모델을 주어진 데이터로 얻는 것	11
2.9	모델을 구한 후에는?	12
2.10	현재 우리의 목표	12
2.11	학습 대상 데이터를 추리고.....	12
2.12	원래 의도한 모델을 만들자.....	13
2.13	loss?.....	13
2.14	optimizer?	13
2.15	summary	13
2.16	모델을 다 구성했다	14
2.17	다음은?	15
2.18	fit~ 학습~	15
2.19	loss가 잘 떨어진다	16
2.20	predict 해볼까?.....	16
2.21	사용법은 sklearn과 비슷하다	17
2.22	가중치와 bias를 알고 싶다면	17
2.23	모델이 잘 만들어졌는지 확인하기 위해 데이터를 만들고	18
2.24	그리기를 시도하자.....	18

2.25 나쁘지 않다	19
3 XOR 문제	20
3.1 XOR	20
3.2 선형 모델로는 XOR를 풀 수가 없다	20
3.3 간단히 데이터를 준비하고	20
3.4 모델은 간단히	21
3.5 위 모델은 어떻게 생겼을까	21
3.6 model.compile.....	21
3.7 model.summary	22
3.8 학습.....	22
3.9 학습결과.....	23
3.10 loss 상황.....	23
3.11 학습에서 찾은 가중치	24
4 이번에는 분류로~	25
4.1 iris 데이터	25
4.2 그런데 y는 이렇게 생겼다	25
4.3 One hot encoding	25
4.4 sklearn의 one hot encoding.....	26
4.5 이제 학습 준비가 되었다	26
4.6 데이터 나누고	26
4.7 난망(net)을 이렇게 구성하기로 했다	27
4.8 이렇게 하면 된다	27
4.9 activation	28
4.10 역전파 back-propagation.....	28
4.11 역전파에서는 sigmoid가 문제가 있다	29
4.12 gradient vanishing.....	29
4.13 ReLU의 등장	30
4.14 softmax?.....	30
4.15 완성된 모델	31
4.16 adam?	31
4.17 gradient decent는 배웠다	31
4.18 복습차원에서~	32

4.19 SGD	32
4.20 GD vs SGD.....	33
4.21 옵티마이저의 선택.....	33
4.22 옵티마이저 계보	34
4.23 데이터가 복잡할 때는 일단 Adam을 써보자.....	35
4.24 summary 결과.....	35
4.25 학습.....	36
4.26 test 데이터에 대한 accuracy	36
4.27 loss와 acc의 변화	37

1 Tensorflow란?

1.1 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

1.2 텐서플로우?

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

1.3 텐서플로우 설치

- <https://www.tensorflow.org/install>

Download a package

Install TensorFlow with Python's pip package manager.

★ TensorFlow 2 packages require a pip version >19.0
(or >20.3 for macOS).

Official packages available for Ubuntu, Windows, and macOS.

[Read the pip install guide](#)

```
# Requires the latest pip
$ pip install --upgrade pip

# Current stable release for CPU and GPU
$ pip install tensorflow

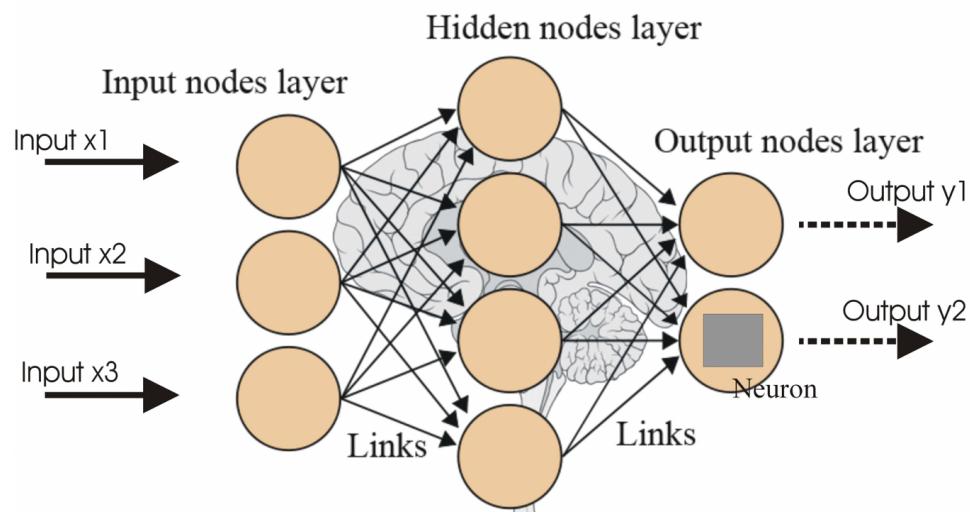
# Or try the preview build (unstable)
$ pip install tf-nightly
```

```
○ (ml) jt@ubuntu:~/dev_ws$ pip install tensorflow
Collecting tensorflow
  Downloading tensorflow-2.18.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.
    metadata (4.1 kB)
Collecting absl-py>=1.0.0 (from tensorflow)
  Downloading absl_py-2.1.0-py3-none-any.whl.metadata (2.3 kB)
Collecting astunparse>=1.6.0 (from tensorflow)
  Downloading astunparse-1.6.3-py2.py3-none-any.whl.metadata (4.4 kB)
Collecting flatbuffers>=24.3.25 (from tensorflow)
  Downloading flatbuffers-25.2.10-py2.py3-none-any.whl.metadata (875 bytes)
Collecting gast!=0.5.0,!0.5.1,!0.5.2,>=0.2.1 (from tensorflow)
  Downloading gast-0.6.0-py3-none-any.whl.metadata (1.3 kB)
```

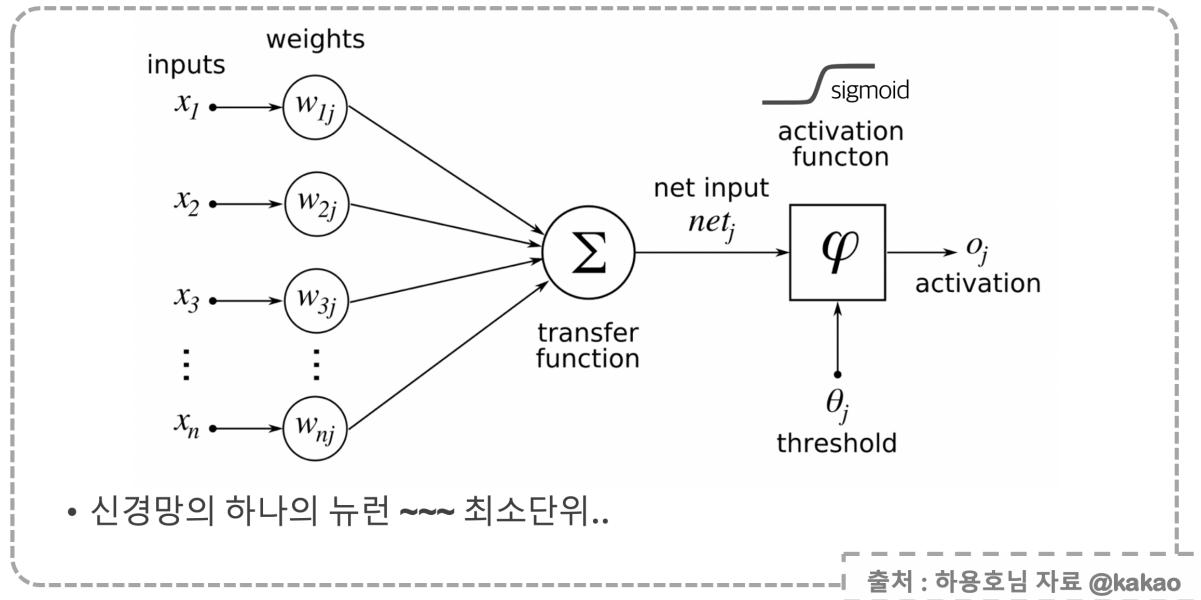
2 딥러닝의 기초 feat. Keras



2.1 신경망에서 아이디어를 얻어서 시작된 Neural Net

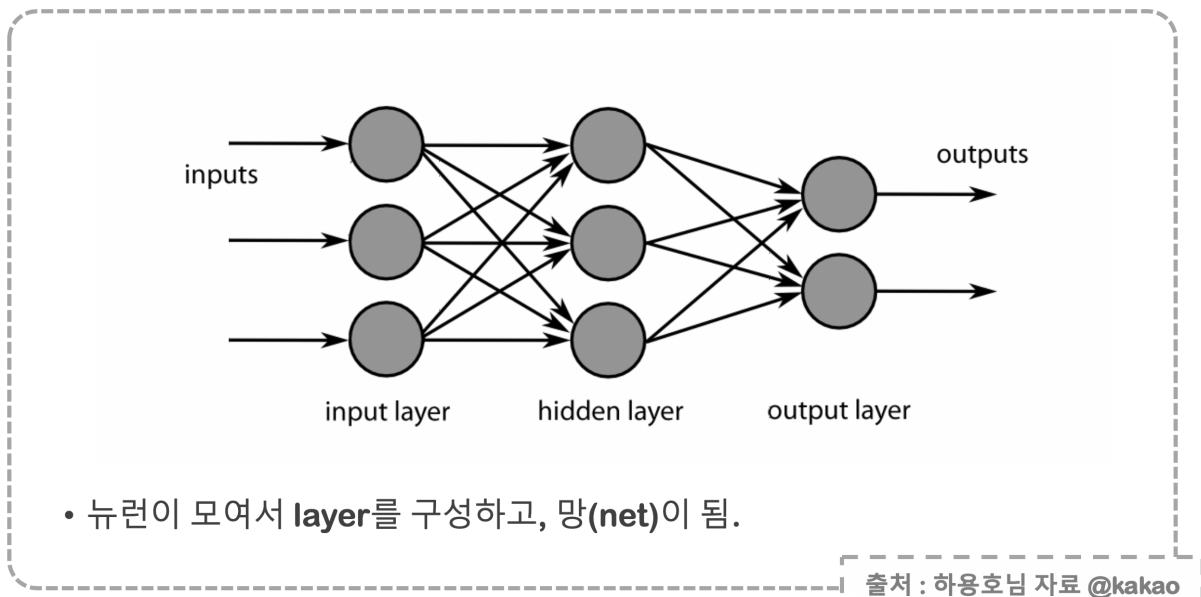


2.2 뉴런

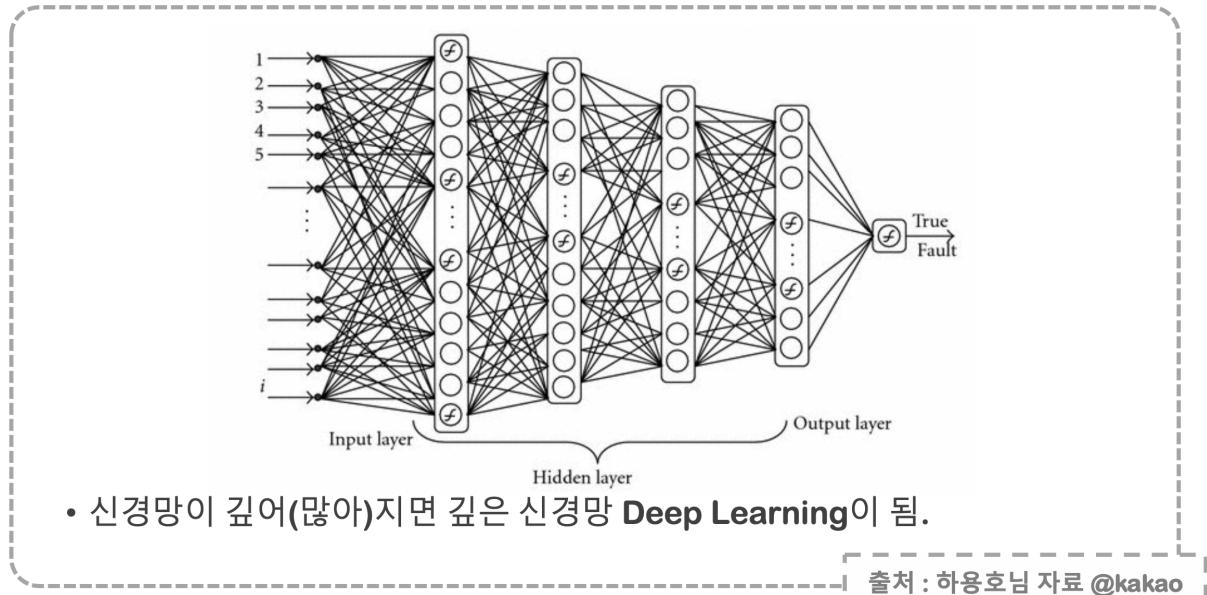


- 뉴런은 입력, 가중치, 활성화함수, 출력으로 구성
- 뉴런에서 학습할 때 변하는 것은 가중치. 처음에는 초기화를 통해 랜덤값을 넣고, 학습과정에서 일정한 값으로 수렴

2.3 레이어와 망(net)



2.4 딥러닝~



2.5 데이터 하나 무작정 읽어보자

- <https://people.sc.fsu.edu/~jb Burkardt/datasets/regression/regression.html>

Datasets:

- [x01.txt](#), brain and body weight, 62 rows, 3 columns;
- [x02.txt](#), height, weight, catheter length, 12 rows, 4 columns;
- [x03.txt](#), age, blood pressure, 30 rows, 4 columns;
- [x04.txt](#), catalog print run versus orders, 38 rows, 4 columns;
- [x05.txt](#), catalog print run versus orders, 38 rows, 5 columns;
- [x06.txt](#), age, water temperature, length of fish, 44 rows, 4 columns;
- [x07.txt](#), retardation, doctor distrust, degree of illness, 53 rows, 4 columns;
- [x08.txt](#), poverty, unemployment, murder rate, 20 rows, 5 columns;
- [x09.txt](#), age, weight, blood fat, 25 rows, 5 columns;
- [x10.txt](#), factory operation parameters, percent of unprocessed ammonia, 21 rows, 5 columns;
- [x11.txt](#), pasturage properties and price, 67 rows, 5 columns;
- [x12.txt](#), electrical utility data, 16 rows, 6 columns;
- [x13.txt](#), production, imports, and consumption data, 18 rows, 7 columns;

```
import numpy as np

raw_data = np.loadtxt("x09.txt", skip_header=36)
raw_data
```

Python

```
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.],
```

2.6 어떻게 생겼나?

```
import matplotlib.pyplot as plt

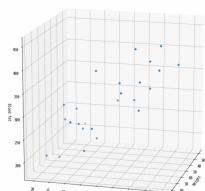
xs = np.array(raw_data[:,2], dtype=np.float32)
ys = np.array(raw_data[:,3], dtype=np.float32)
zs = np.array(raw_data[:,4], dtype=np.float32)

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()
```

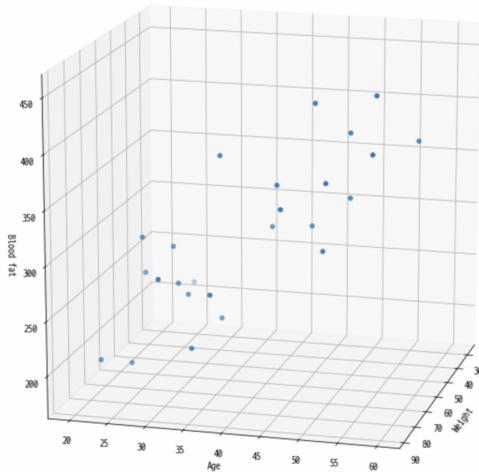
✓ 0.3s

Python

2.7 현재 간단한 딥러닝의 목표

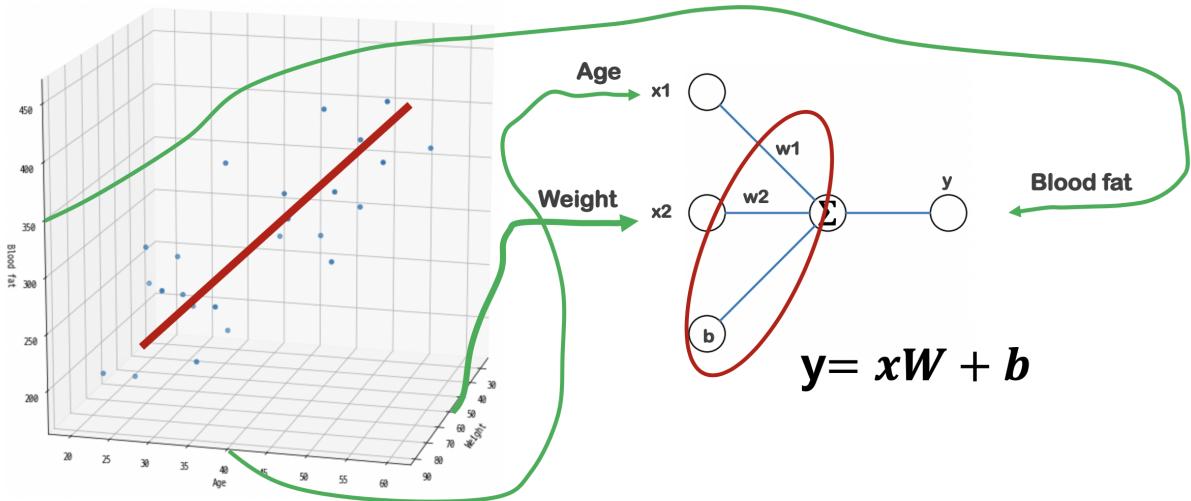


- 이 예제의 목적은 입력인 나이와 몸무게를 알려주면, 주어진 데이터 기준의 blood fat를 얻는 것이다.
- 즉, 40살, 100키로인 사람의 데이터 기준 blood fat은? 하고 물으면, 얼마입니 다~~~ 하고 답이 나와야 하는 것이다.
- Linear Regression ~~



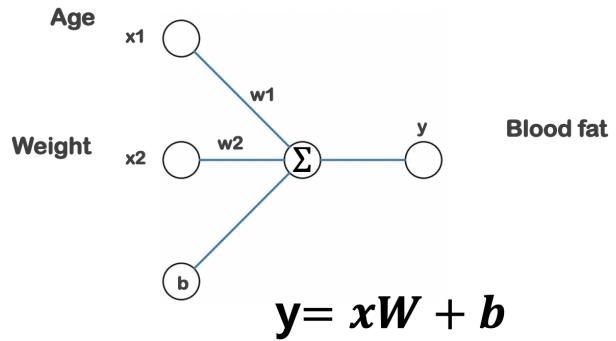
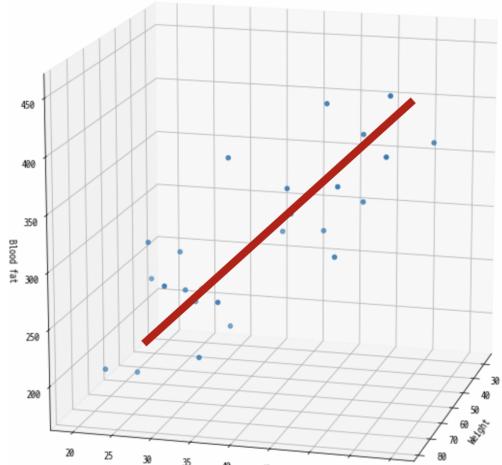
- 이 예제의 목적은 입력인 나이와 몸무게를 알려주면, 주어진 데이터 기준의 **blood fat**을 얻는 것이다.
- 즉, **40살, 100키로인 사람의 데이터 기준 blood fat은?** 하고 물으면, 얼마입니~다~~~하고 답이 나와야 하는 것이다.
- **Linear Regression ~~**

2.8 먼저 모델을 주어진 데이터로 얻는 것



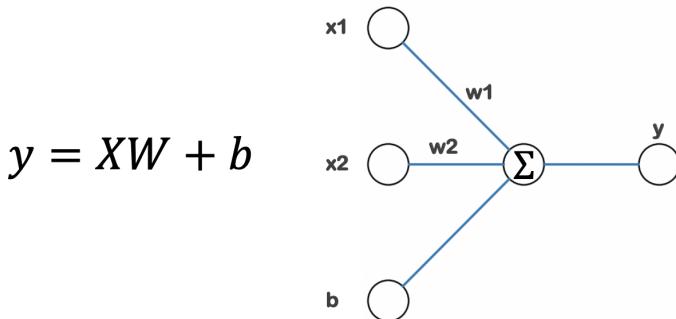
- 직선 모델을 얻는 것으로 하면, 주어진 입출력 데이터로 **W**와 **b** 즉, 모델을 얻는 것

2.9 모델을 구한 후에는?



- 이제 모델(W, b)을 이용해서, 질문을 하는 것, 즉, age 40, weight 80인 사람의 Blood fat은 얼마? 하면, 그 대답을 얻는 것이 방금 학습을 한 이유

2.10 현재 우리의 목표



- 목적: x_1, x_2 를 입력해서 y 가 나오게 하는 Weight와 bias를 구하는 것

2.11 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))
    
```

Python

2.12 원래 의도한 모델을 만들자

```
import tensorflow as tf

model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Dense(1, input_shape=(2,)),
    ])

model.compile(optimizer="rmsprop", loss="mse")
```

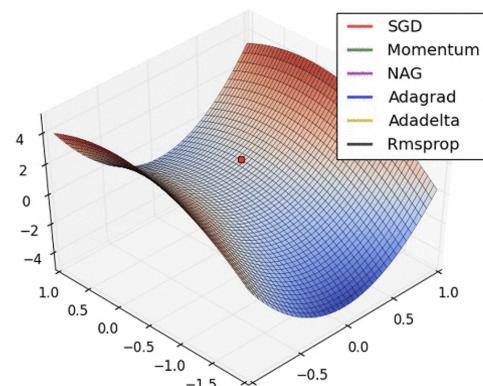
Python

2.13 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.

2.14 optimizer?

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



2.15 summary

```
model.summary()
```

Python

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	3

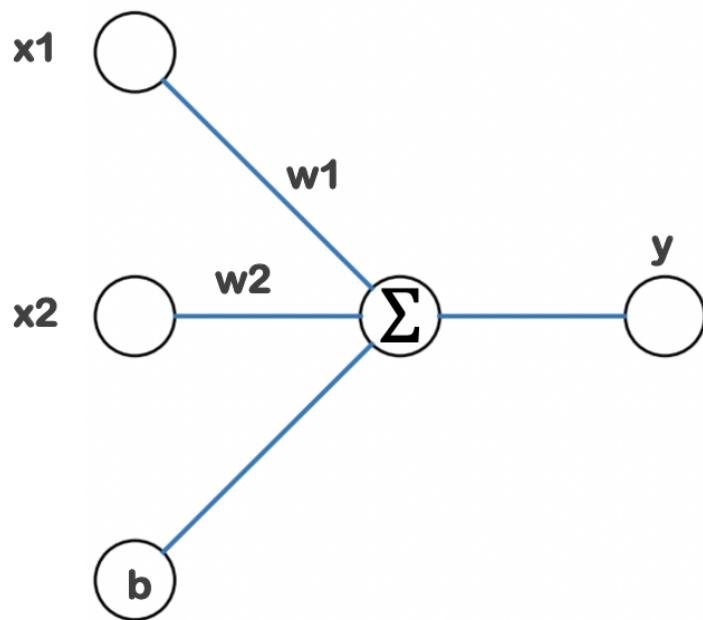
Total params: 3 (12.00 B)

Trainable params: 3 (12.00 B)

Non-trainable params: 0 (0.00 B)

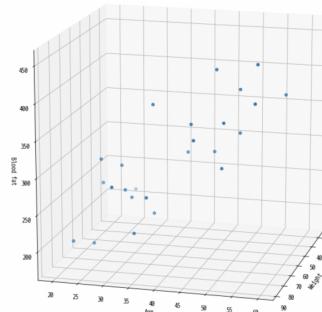
2.16 모델을 다 구성했다

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	3



2.17 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??



2.18 fit~ 학습~

```
hist = model.fit(x_data, y_data, epochs=10000)
✓ 8m 46.7s
```

Python

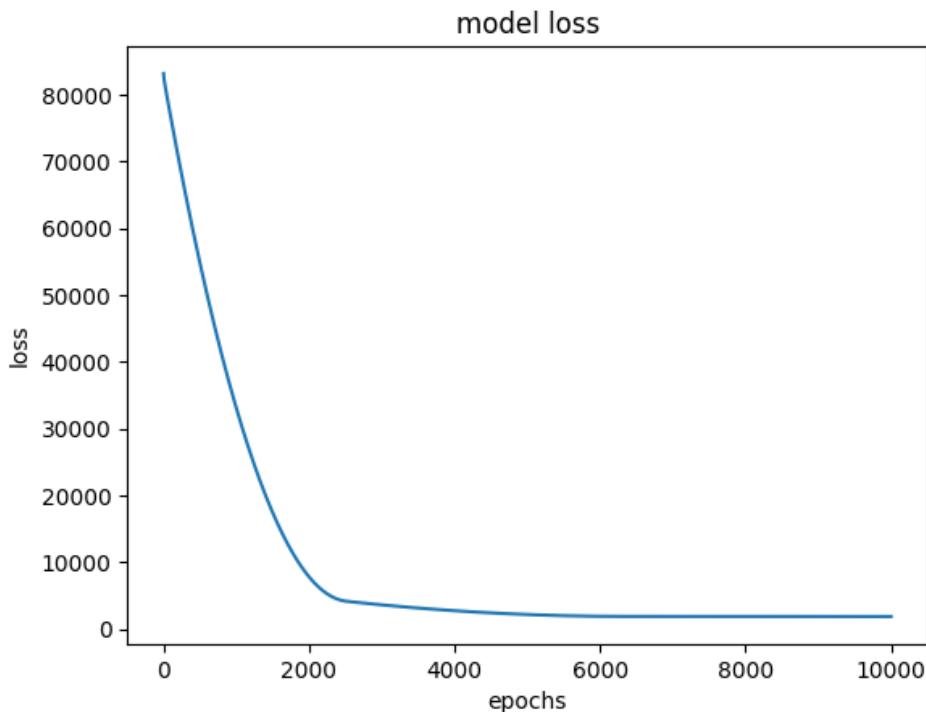
```
1/1 ━━━━━━━━━━ 0s 43ms/step - loss: 1843.3488
Epoch 9977/10000
1/1 ━━━━━━━━ 0s 38ms/step - loss: 1843.3448
Epoch 9978/10000
1/1 ━━━━━━ 0s 39ms/step - loss: 1843.3413
Epoch 9979/10000
1/1 ━━━━ 0s 41ms/step - loss: 1843.3374
Epoch 9980/10000
1/1 ━━ 0s 46ms/step - loss: 1843.3333
Epoch 9981/10000
1/1 ━ 0s 54ms/step - loss: 1843.3291
```

2.19 loss가 잘 떨어진다

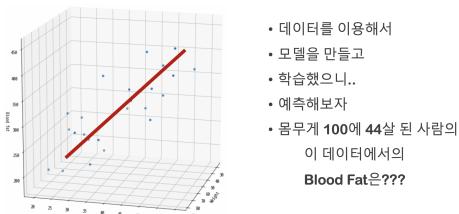
```
plt.plot(hist.history["loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epochs")
plt.show()
```

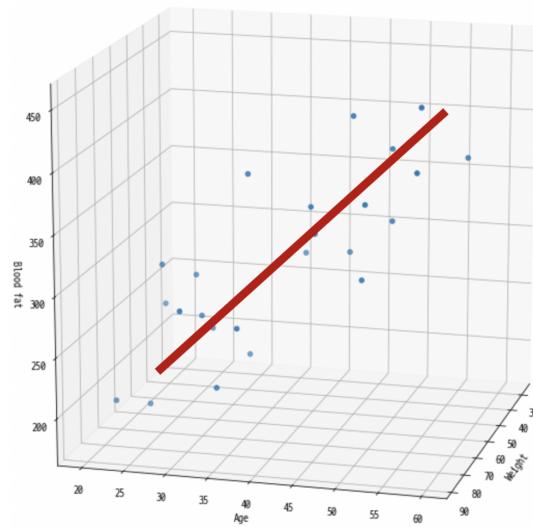
✓ 0.0s

Python



2.20 predict 해볼까?





- 데이터를 이용해서
- 모델을 만들고
- 학습했으니..
- 예측해보자
- 몸무게 100에 44살 된 사람의
이 데이터에서의
Blood Fat은???

2.21 사용법은 sklearn과 비슷하다

```
model.predict(np.array([100, 44]).reshape(1, 2))
✓ 0.2s
1/1 ━━━━━━━━ 0s 49ms/step
array([[373.2614]], dtype=float32)
```

Python

```
model.predict(np.array([60, 25]).reshape(1, 2))
✓ 0.0s
1/1 ━━━━━━━━ 0s 34ms/step
array([[220.17584]], dtype=float32)
```

Python

2.22 가중치와 bias를 알고 싶다면

```
W_, b_ = model.get_weights()
print("Weight is : ", W_)
print("bias is : ", b_)
✓ 0.0s
Weight is : [[1.1923258]
[5.546977 ]]
bias is : [9.961866]
```

Python

2.23 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_

```

✓ 0.0s

Python

2.24 그리기를 시도하자

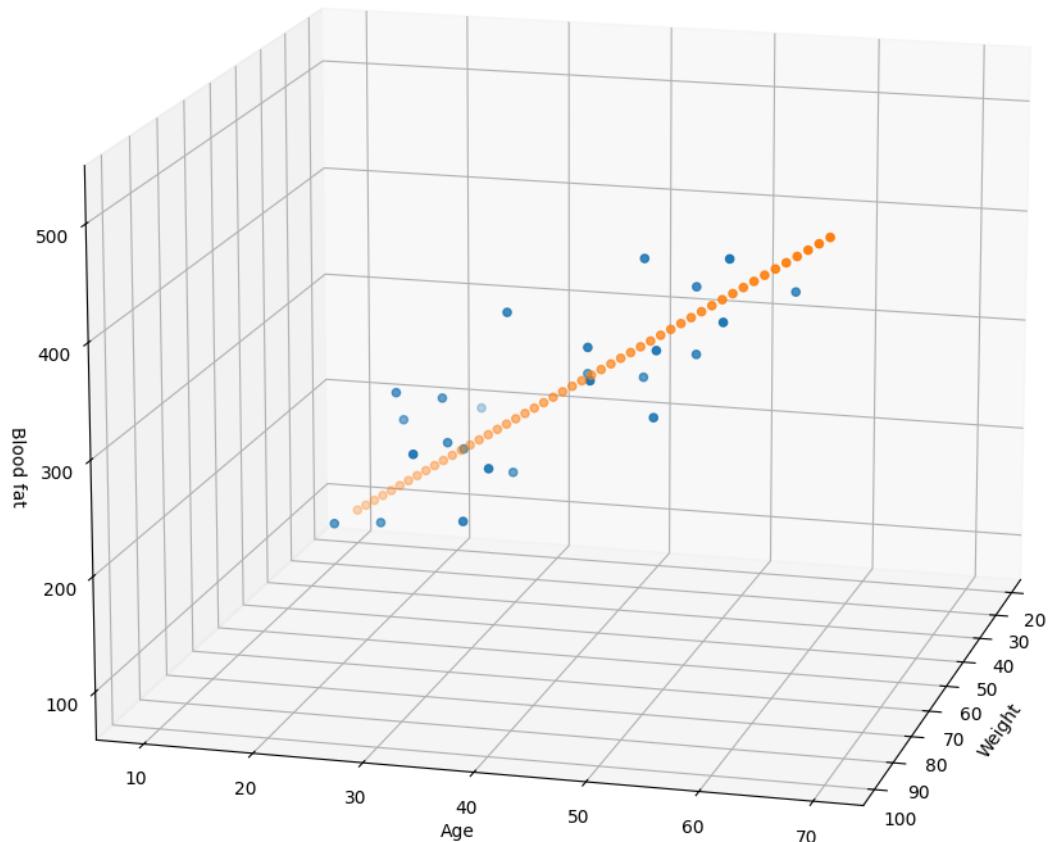
```
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(xs, ys, zs)
ax.scatter(x, y, Z)
ax.set_xlabel("Weight")
ax.set_ylabel("Age")
ax.set_zlabel("Blood fat")
ax.view_init(15, 15)
plt.show()

```

✓ 0.1s

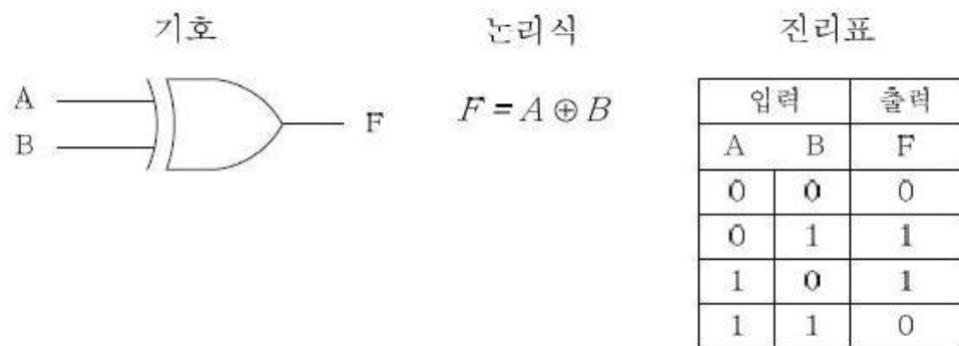
Python

2.25 나쁘지 않다

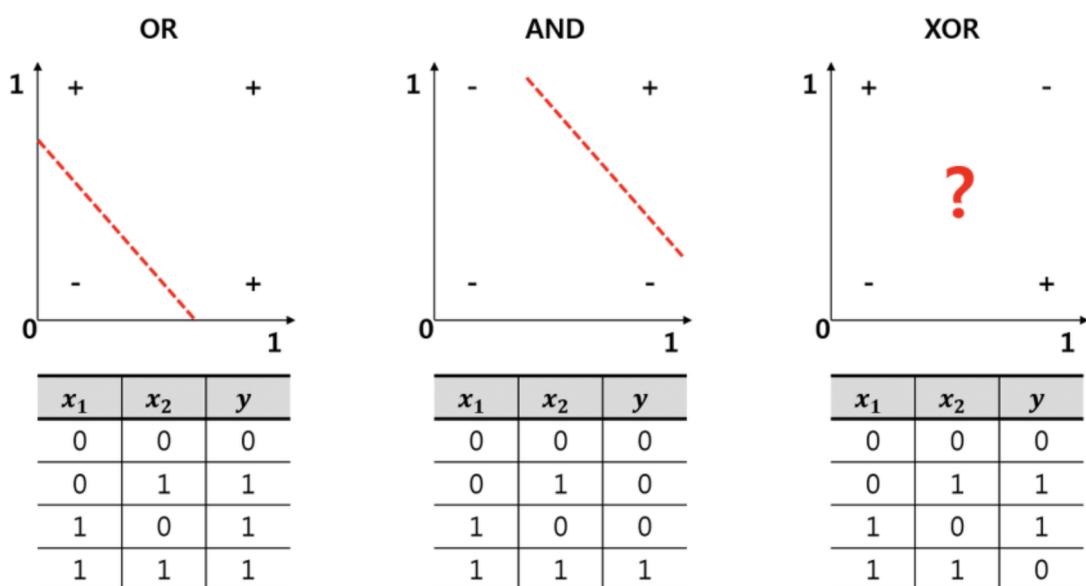


3 XOR 문제

3.1 XOR



3.2 선형 모델로는 XOR를 풀 수가 없다



3.3 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]])

✓ 0.0s
```

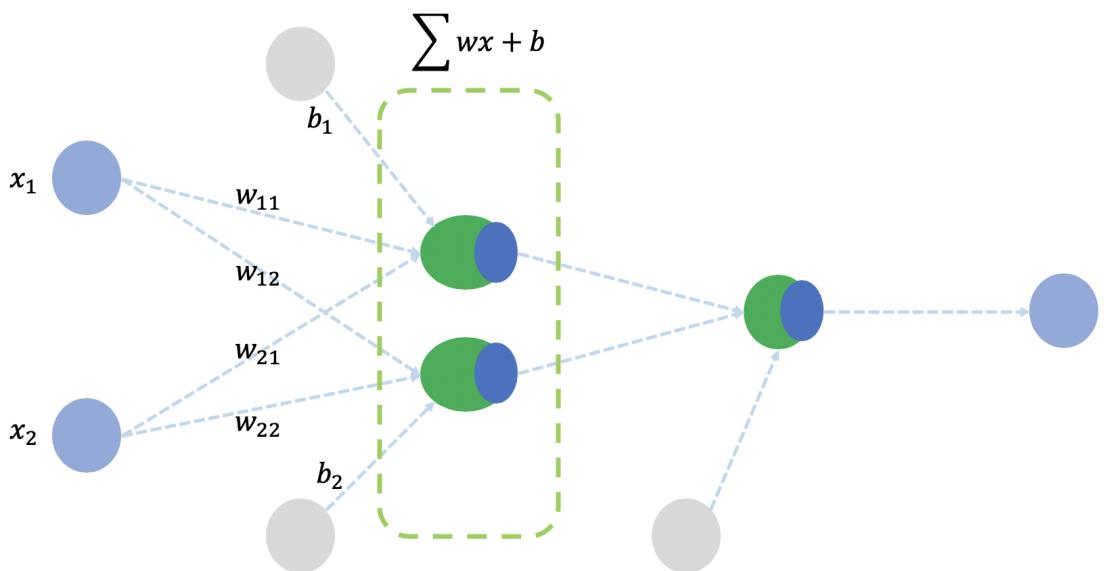
Python

3.4 모델은 간단히

```
model = tf.keras.Sequential(
    [
        tf.keras.layers.Dense(2, activation="sigmoid", input_shape=(2,)),
        tf.keras.layers.Dense(1, activation="sigmoid"),
    ]
)
✓ 0.0s
```

Python

3.5 위 모델은 어떻게 생겼을까



3.6 model.compile

```
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.1), loss="mse")
✓ 0.0s
```

Python

- 옵티마이저를 설정하고, 학습률을 설정한다.
- loss 함수는 mse로 한다 mean squared error

3.7 model.summary

```
model.summary()
✓ 0.0s
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2)	6
dense_2 (Dense)	(None, 1)	3

Total params: 9 (36.00 B)

Trainable params: 9 (36.00 B)

Non-trainable params: 0 (0.00 B)

3.8 학습

```
hist = model.fit(X, y, epochs=5000, batch_size=1)
✓ 3m 36.4s
```

Epoch 4986/5000
 4/4 ━━━━━━━━ 0s 9ms/step - loss: 0.1469
 Epoch 4987/5000
 4/4 ━━━━━━ 0s 10ms/step - loss: 0.0701
 Epoch 4988/5000
 4/4 ━━━━ 0s 6ms/step - loss: 0.1128
 Epoch 4989/5000
 4/4 ━━━━ 0s 7ms/step - loss: 0.0708
 Epoch 4990/5000

- epochs는 지정된 횟수만큼 학습을 하는 것
- batch_size는 한번의 학습에 사용될 데이터의 수를 지정

3.9 학습결과

```
model.predict(X)
✓ 0.0s
1/1 ━━━━━━━━ 0s 56ms/step

array([[0.04668765],
       [0.9411334 ],
       [0.49895597],
       [0.5069579 ]], dtype=float32)
```

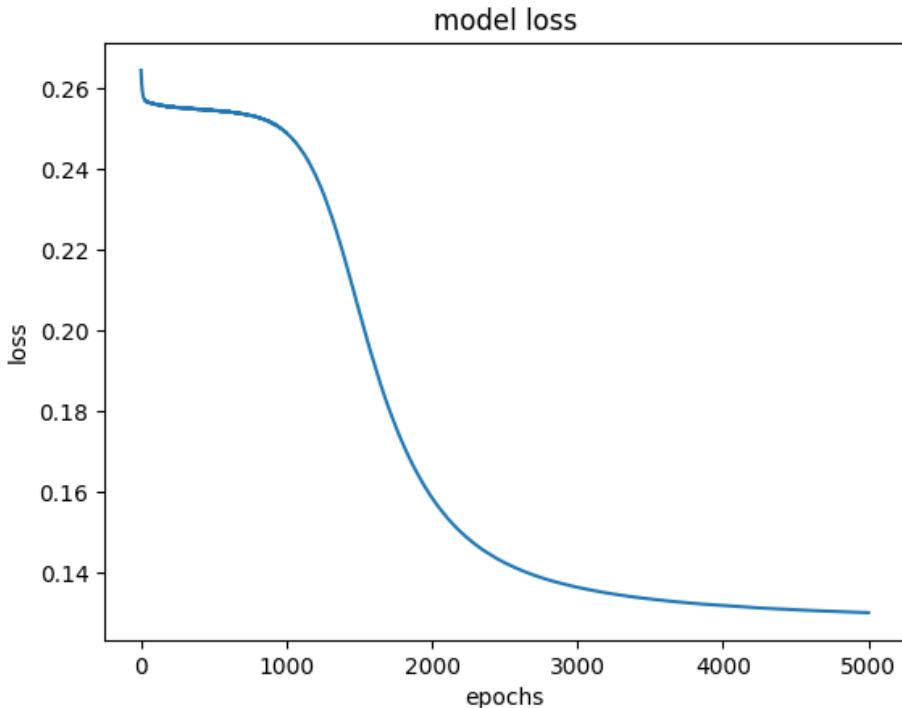
Python

3.10 loss 상황

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

✓ 0.0s

Python



3.11 학습에서 찾은 가중치

```
for w in model.weights:  
    print("----")  
    print(w)  
    ✓ 0.0s  
---  
<Variable path=sequential_1/dense_1/kernel, shape=(2, 2), dtype=float32, value=  
[[-4.428503  2.0893295]  
 [-6.7301097 -6.11057 ]]>  
---  
<Variable path=sequential_1/dense_1/bias, shape=(2,), dtype=float32, value=[  
 0.81099373 -1.3783121 ]>  
---  
<Variable path=sequential_1/dense_2/kernel, shape=(2, 1), dtype=float32, value=  
[[-5.63109 ]  
 [ 4.3400626]]>  
---  
<Variable path=sequential_1/dense_2/bias, shape=(1,), dtype=float32, value=  
[0.00849132]>
```

4 이번에는 분류로~

4.1 iris 데이터

```
from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data
y = iris.target
```

✓ 0.0s

Python

4.2 그런데 y는 이렇게 생겼다

y

✓ 0.0s

Python

```
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, 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])
```

4.3 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										

4.4 sklearn의 one hot encoding

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(sparse_output=False, handle_unknown="ignore")
enc.fit(y.reshape(len(y), 1))
```

✓ 0.0s

Python

OneHotEncoder
OneHotEncoder(handle_unknown='ignore', sparse_output=False)

4.5 이제 학습 준비가 되었다

```
enc.categories_
```

✓ 0.0s

Python

```
[array([0, 1, 2])]
```

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

✓ 0.0s

Python

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

4.6 데이터 나누고

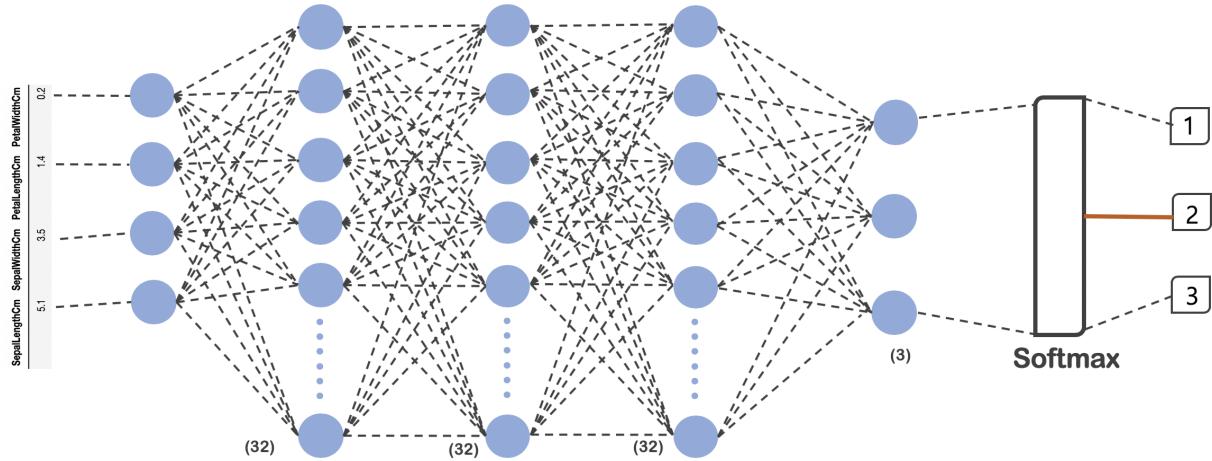
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y_onehot, test_size=0.2, random_state=13
)
```

✓ 0.0s

Python

4.7 난망(net)을 이렇게 구성하기로 했다

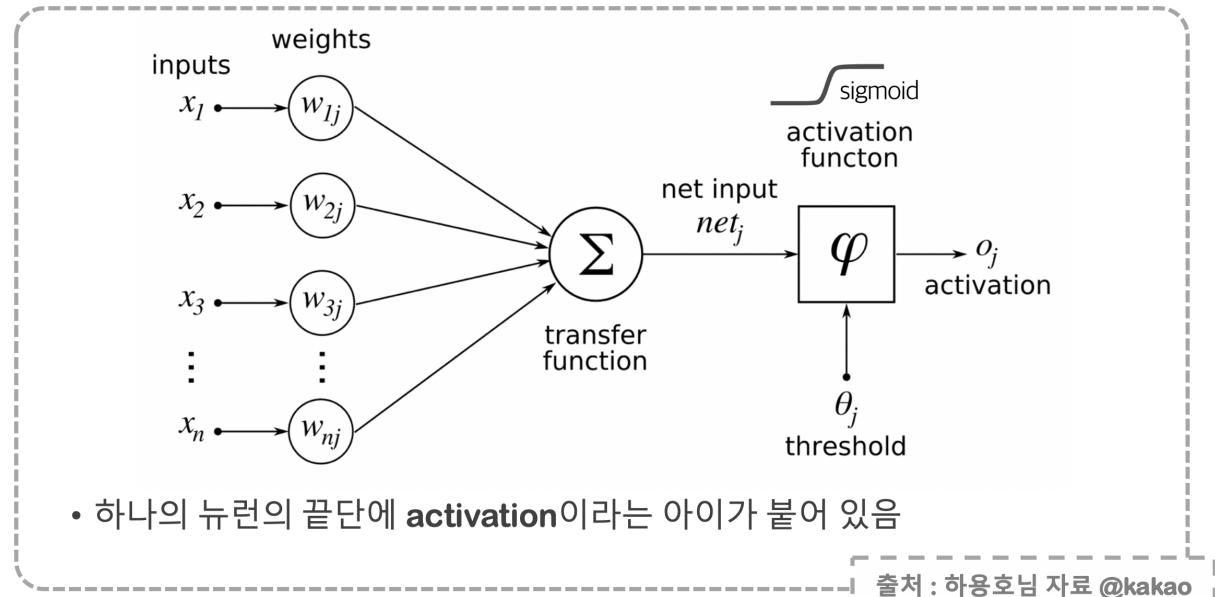


4.8 이렇게 하면 된다

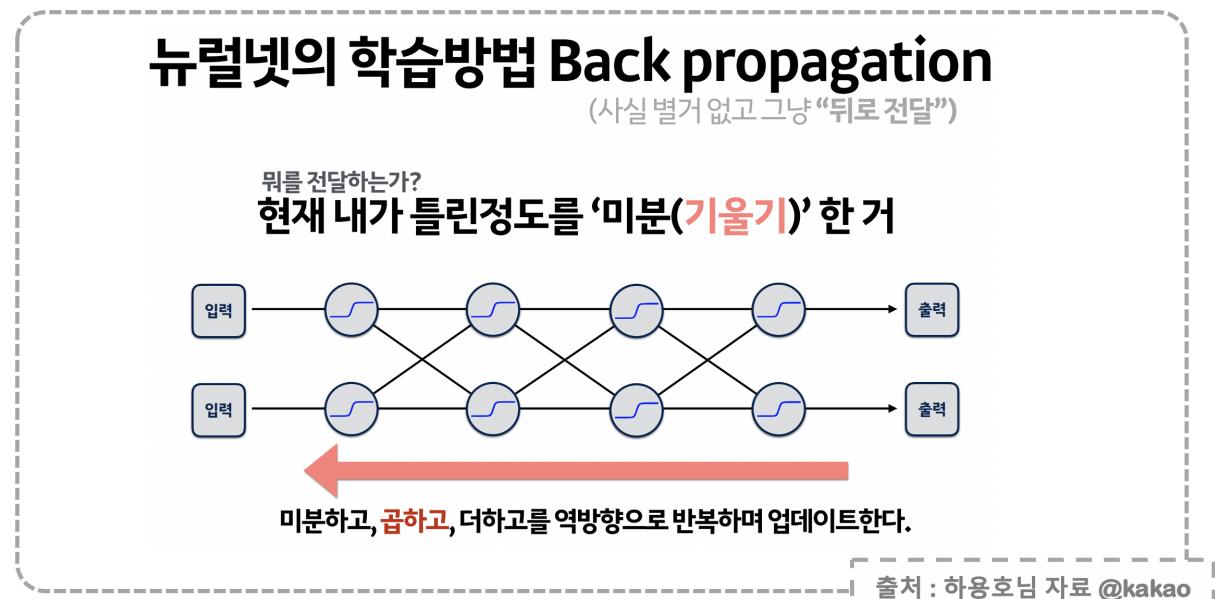
```
model = tf.keras.models.Sequential(
    [
        tf.keras.layers.Dense(32, input_shape=(4,), activation="relu"),
        tf.keras.layers.Dense(32, activation="relu"),
        tf.keras.layers.Dense(32, activation="relu"),
        tf.keras.layers.Dense(3, activation="softmax"),
    ]
)
✓ 0.0s
```

Python

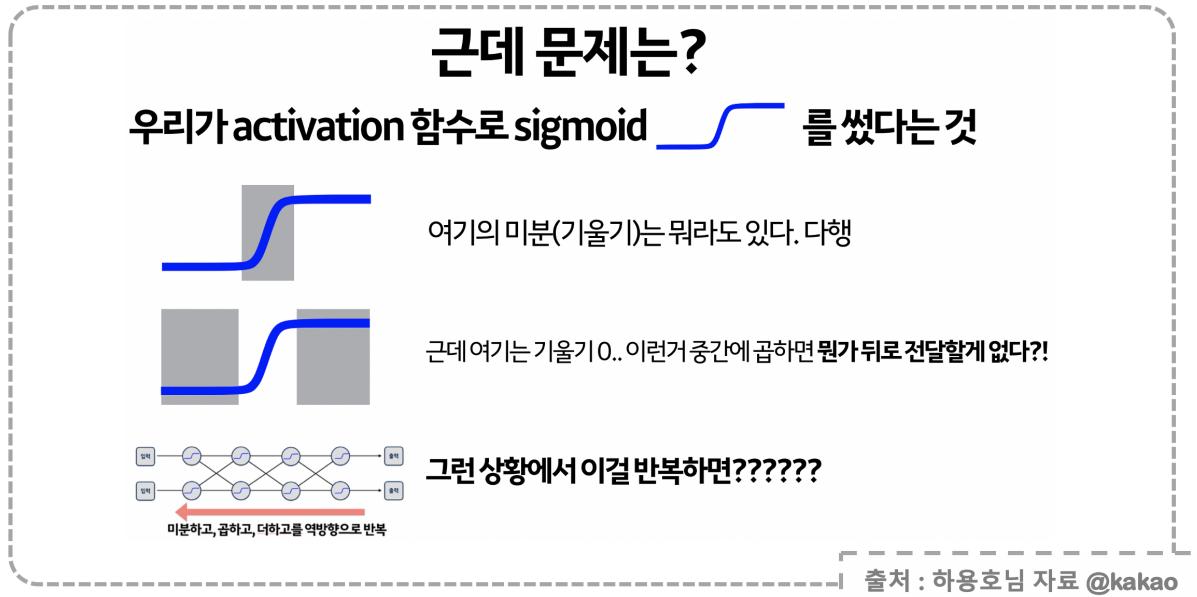
4.9 activation



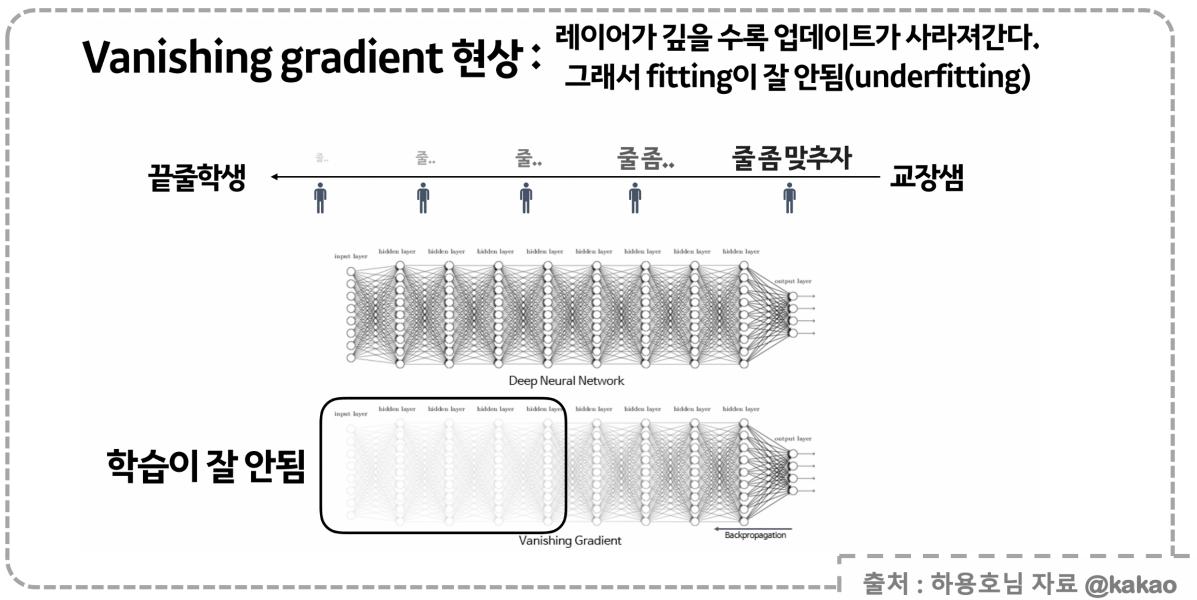
4.10 역전파 back-propagation



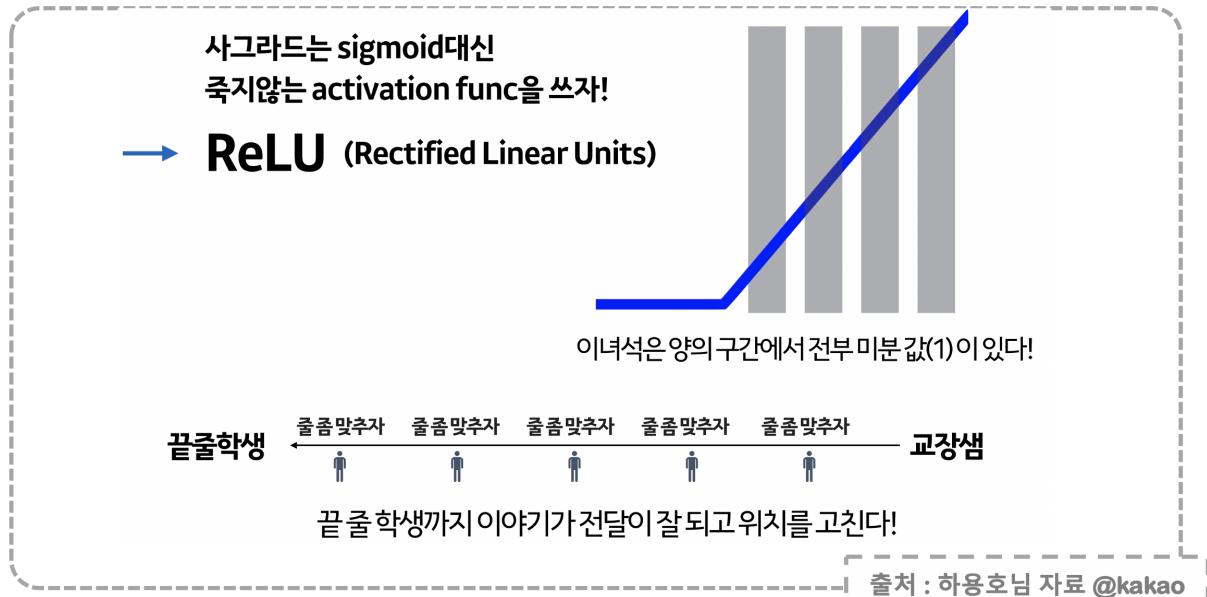
4.11 역전파에서는 sigmoid가 문제가 있다



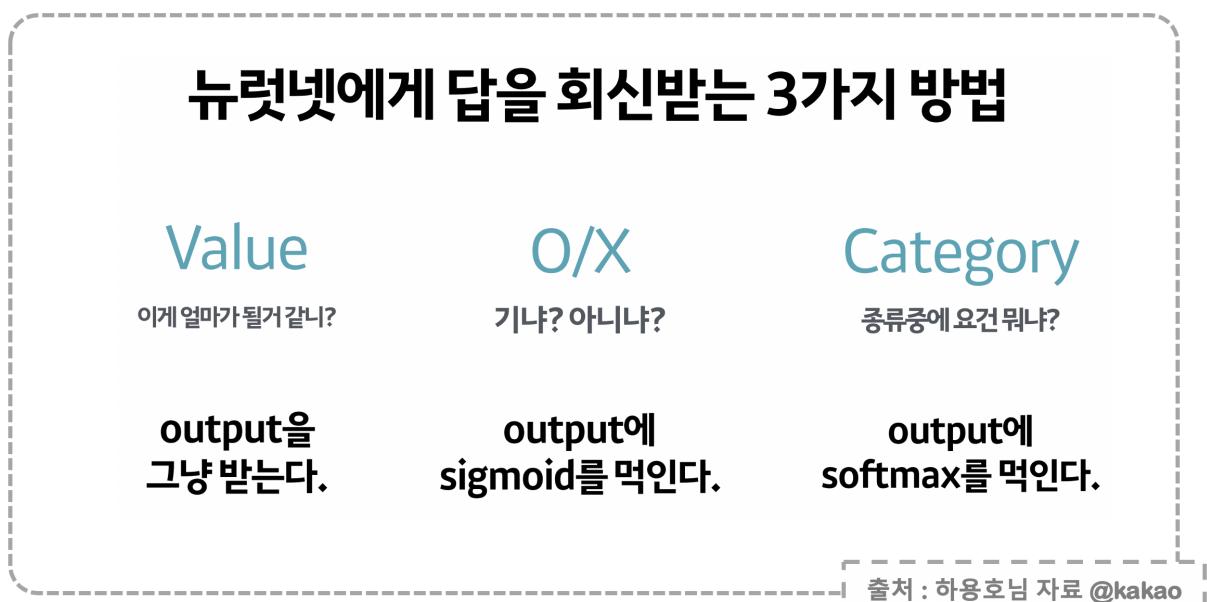
4.12 gradient vanishing



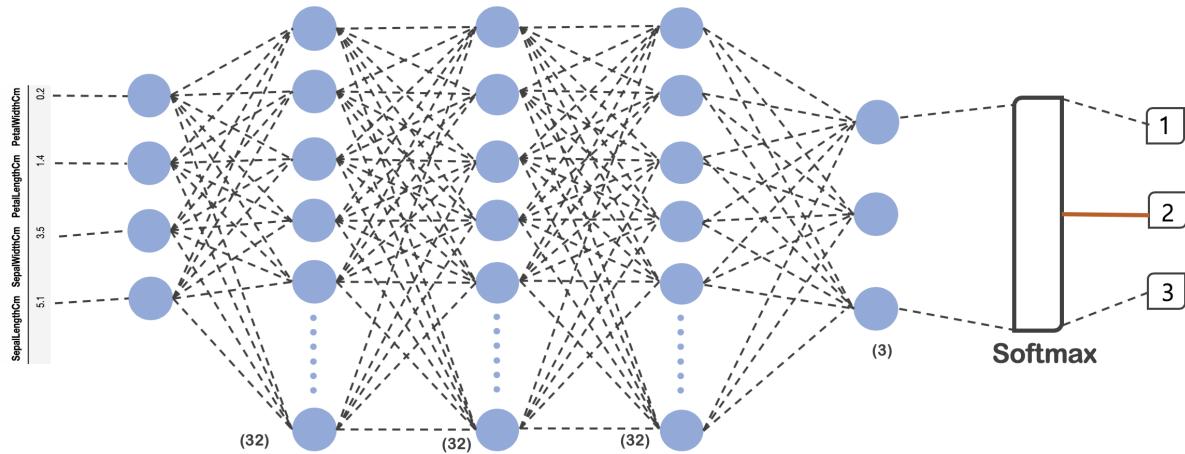
4.13 ReLU의 등장



4.14 softmax?



4.15 완성된 모델



4.16 adam?

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

model.summary()
```

Python

4.17 gradient decent는 배웠다

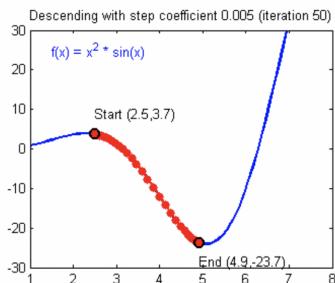
기존 뉴럴넷이 가중치 parameter들을
최적화(optimize)하는 방법

Gradient Decent

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

출처 : 하용호님 자료 @kakao

4.18 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~~

$$\text{weight의 업데이트} = \frac{\text{에러 낮추는 방향}}{\text{(decent)}} \times \frac{\text{한발자국 크기}}{\text{(learning rate)}} \times \frac{\text{현 지점의 기울기}}{\text{(gradient)}}$$

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

출처 : 하용호님 자료 @kakao

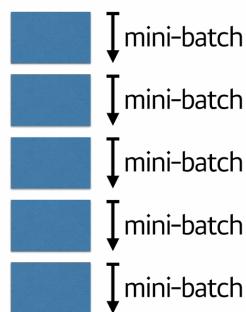
4.19 SGD

Gradient Decent

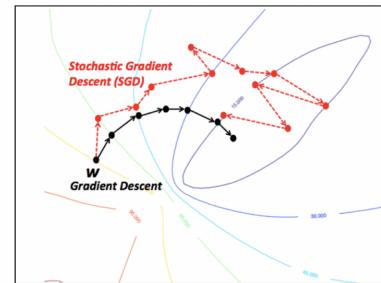


전부다 읽고나서
최적의 1스텝 간다.

Stochastic Gradient Decent

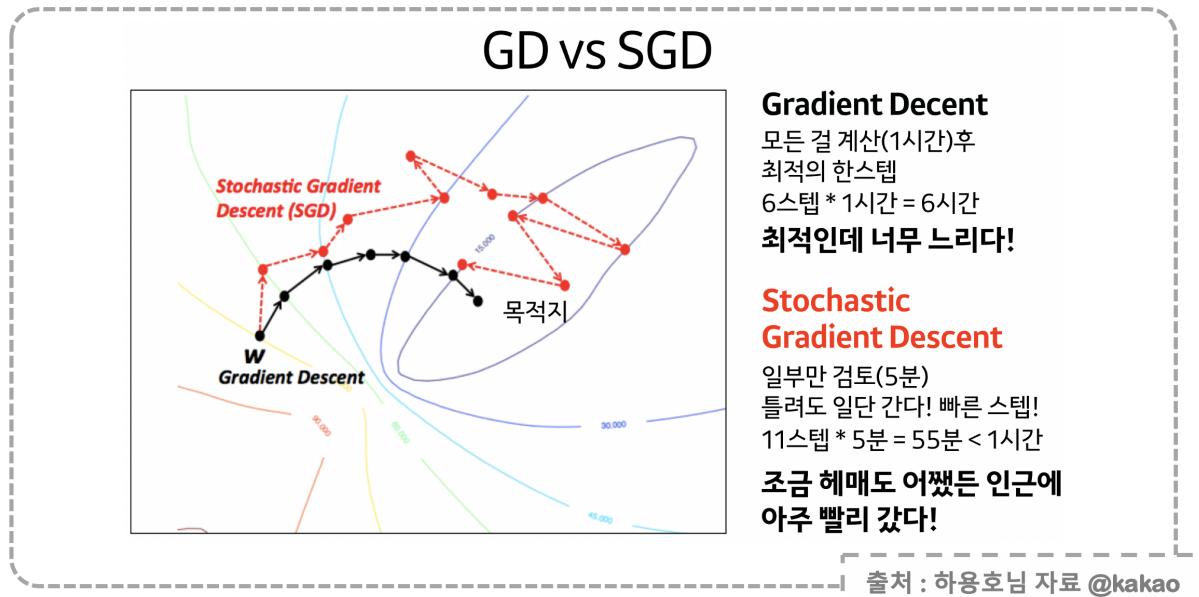


작은 토막마다
일단 1스텝간다.



출처 : 하용호님 자료 @kakao

4.20 GD vs SGD



4.21 옵티마이저의 선택

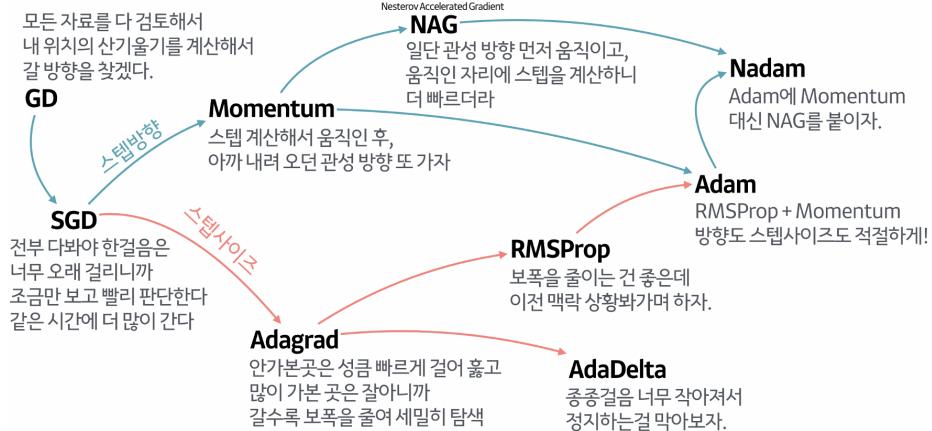
$-\gamma \nabla F(\mathbf{a}^n)$ **산을 잘 타고 내려오는 것은**
 $\nabla F(\mathbf{a}^n)$ **어느 방향으로** 발을 디딜지
 γ **얼마 보폭으로** 발을 디딜지
두가지를 잘 잡아야 빠르게 타고 내려온다.

SGD를 더 개선한 멋진 optimizer가 많다!
SGD의 개선된 후계자들

출처 : 하용호님 자료 @kakao

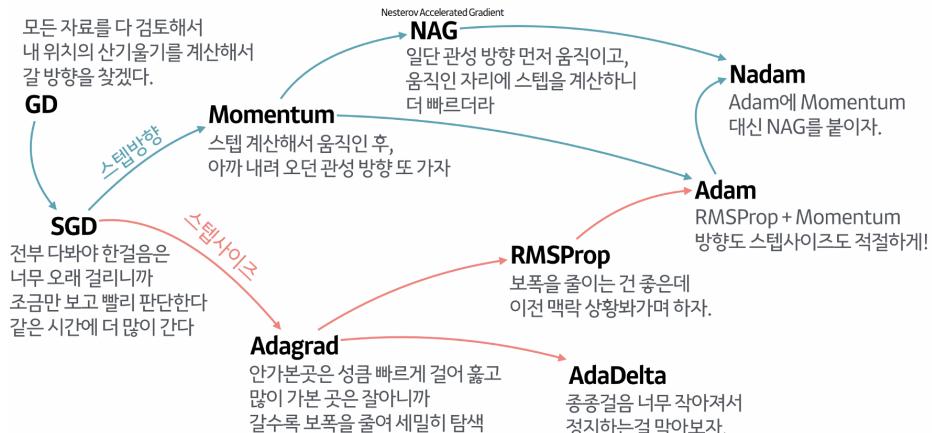
4.22 옵티마이저 계보

산내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



출처 : 하용호님 자료 @kakao

산내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



출처 : 하용호님 자료 @kakao

4.23 데이터가 복잡할 때는 일단 Adam을 써보자

잘 모르겠으면 Adam!

출처 : 하용호님 자료 @kakao

4.24 summary 결과

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 32)	160
dense_4 (Dense)	(None, 32)	1,056
dense_5 (Dense)	(None, 32)	1,056
dense_6 (Dense)	(None, 3)	99

Total params: 2,371 (9.26 KB)

Trainable params: 2,371 (9.26 KB)

Non-trainable params: 0 (0.00 B)

4.25 학습

```
hist = model.fit(X_train, y_train, epochs=100)
✓ 5.5s
```

Epoch 86/100
4/4 0s 7ms/step - accuracy: 0.9712 - loss: 0.0792
Epoch 87/100
4/4 0s 7ms/step - accuracy: 0.9819 - loss: 0.0850
Epoch 88/100
4/4 0s 7ms/step - accuracy: 0.9852 - loss: 0.0857
Epoch 89/100
4/4 0s 7ms/step - accuracy: 0.9852 - loss: 0.0834
Epoch 90/100
4/4 0s 8ms/step - accuracy: 0.9806 - loss: 0.0650
Epoch 91/100

4.26 test 데이터에 대한 accuracy

```
model.evaluate(X_test, y_test, verbose=2)
✓ 0.1s
```

1/1 - 0s - 153ms/step - accuracy: 1.0000 - loss: 0.0804
[0.08044213801622391, 1.0]

4.27 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()
```

✓ 0.1s

Python

