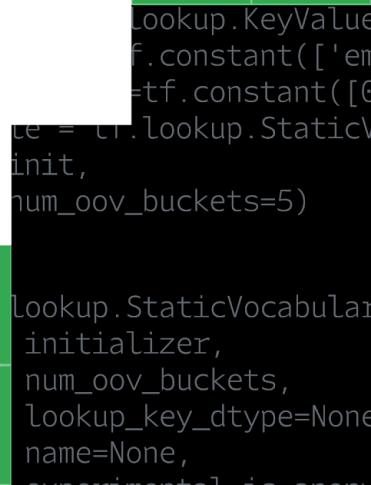
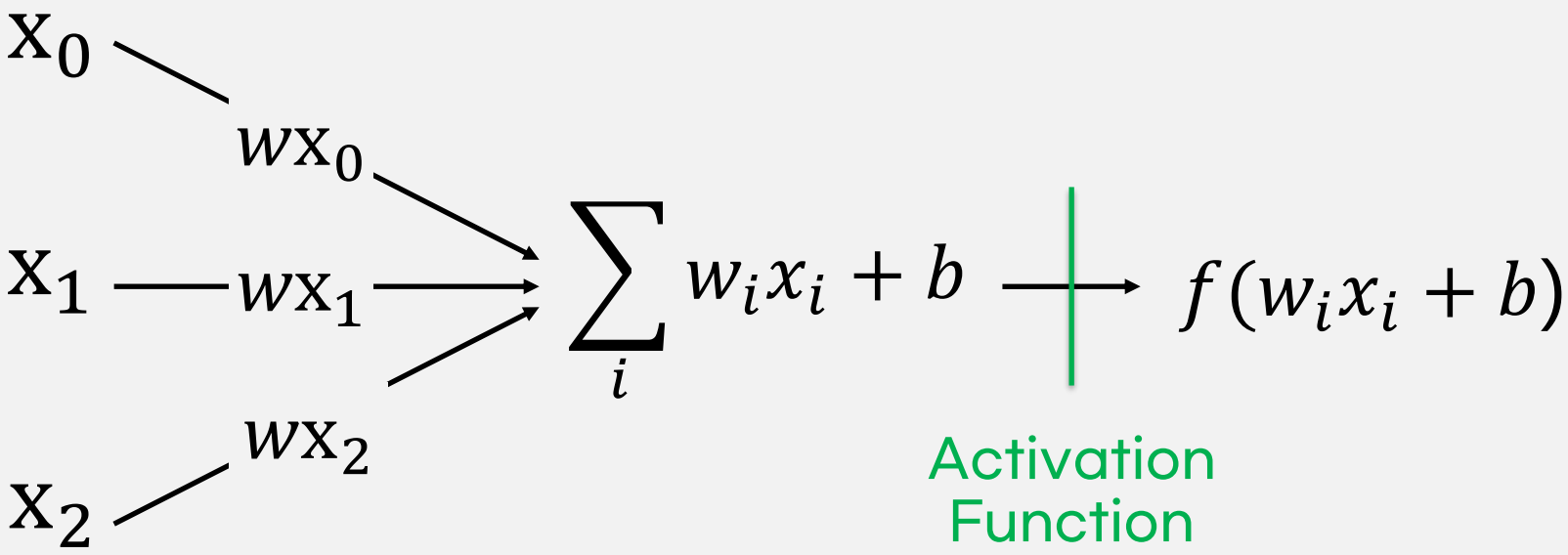
Google Developer Student Clubs

# ML/DL Basic Week03



# Activation Function




axon from a neuron

synapse

cell body

output axon

# Activation Function

사용 이유: Data를 비선형으로 바꾸기 위해서

은닉층 증가 -> 많은 양의 매개변수 필요X, 연산의 수 감소

# Activation Function

1. Sigmoid Function
- 실수 값을 0과 1 사이의 확률 값으로 변환
- 입력 값의 절댓값이 커질수록 기울기가 0으로 수렴

2. Hyperbolic Tangent Function
- 기울기가 작아지지 않아 양수와 음수 모두 학습 효율성 뛰어남
- 기울기가 0으로 수렴하는 구간 존재 -> 기울기 소실 문제

# Activation Function

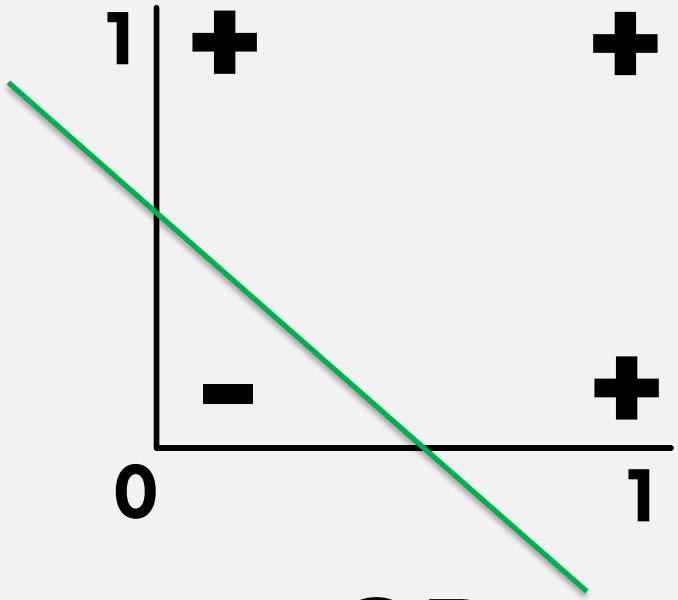
3. ReLU 함수

- 입력값이 음수이면 0 출력, 양수이면 입력값 그대로 출력
- 기울기 소실 문제 해결 but, 입력값이 너무 커지면 편향

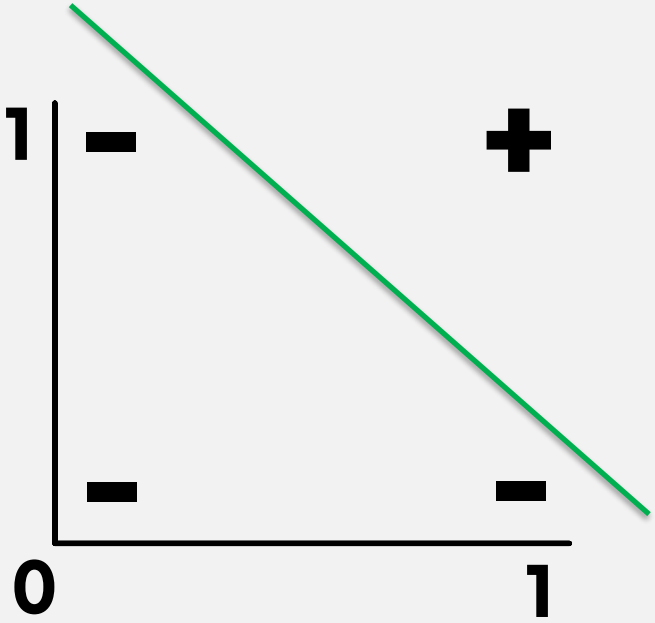
4. Leaky ReLU 함수

- ReLU 함수의 단점을 보완하기 위해 나온 함수
- 음수 값에서 입력 데이터에 아주 작은 값을 곱하여 반환

# AND / OR Problem: Linearly Separable



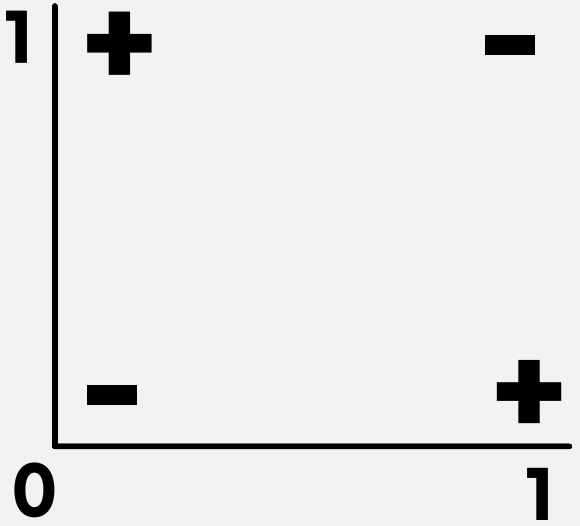

OR




AND

# XOR Problem: Linearly Separable?

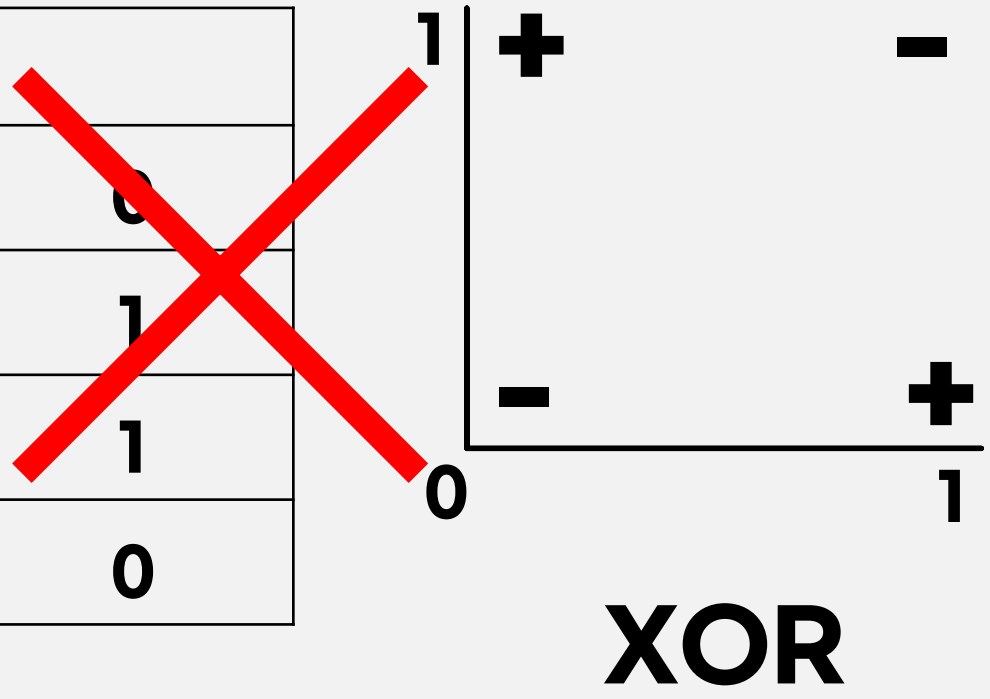
| $x_1$ | $x_2$ | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



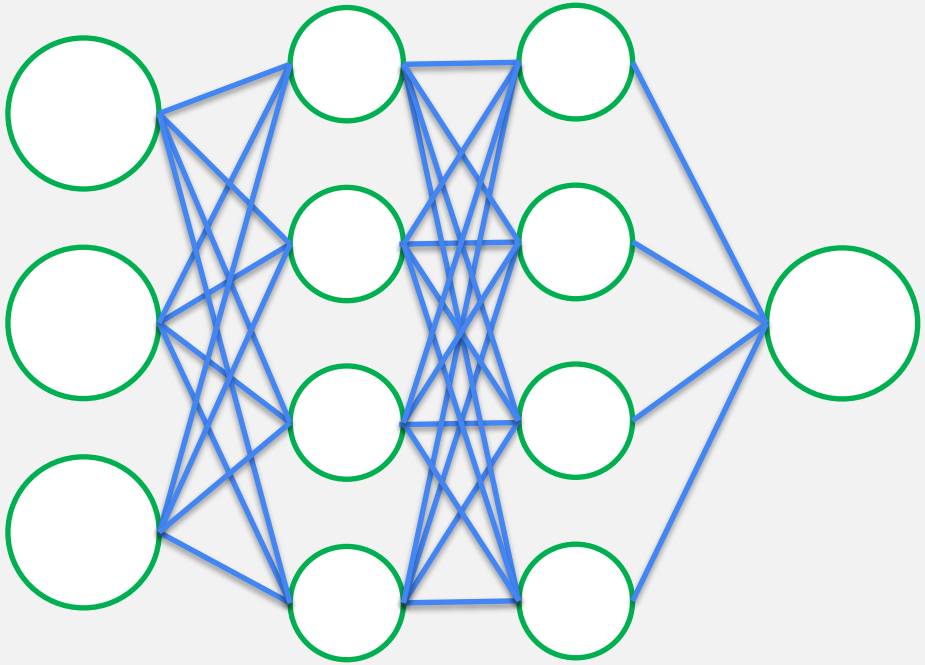

XOR

# XOR Problem: Linearly Separable?

| $x_1$ | $x_2$ | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



# Multilayer Perceptron(MLP)



input layer

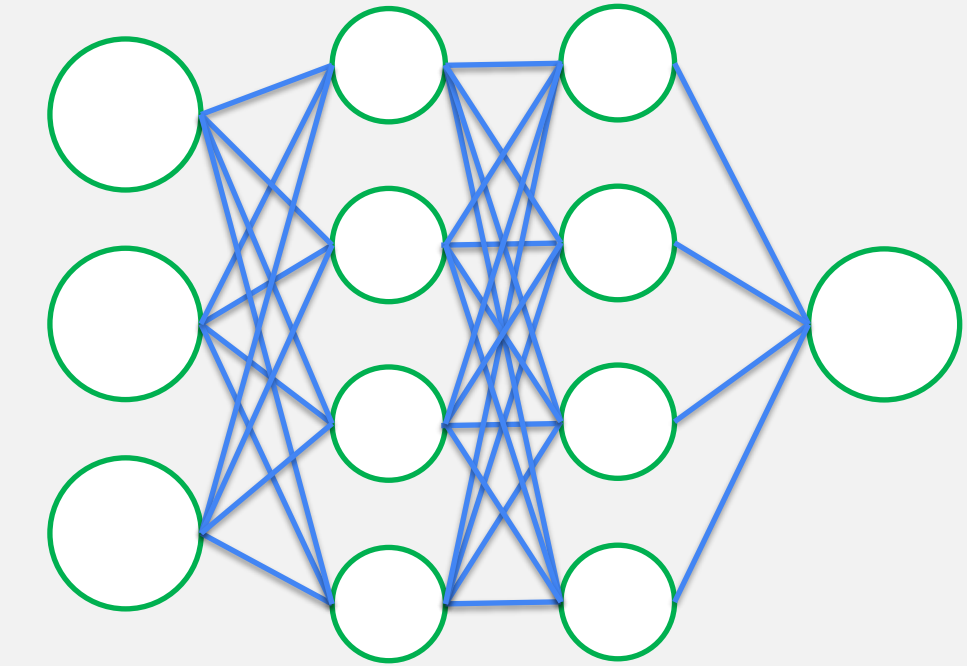
hidden layer

output layer

Minsky
: No one had found a way to **train MLPs.**

**Backpropagation**

# Backpropagation

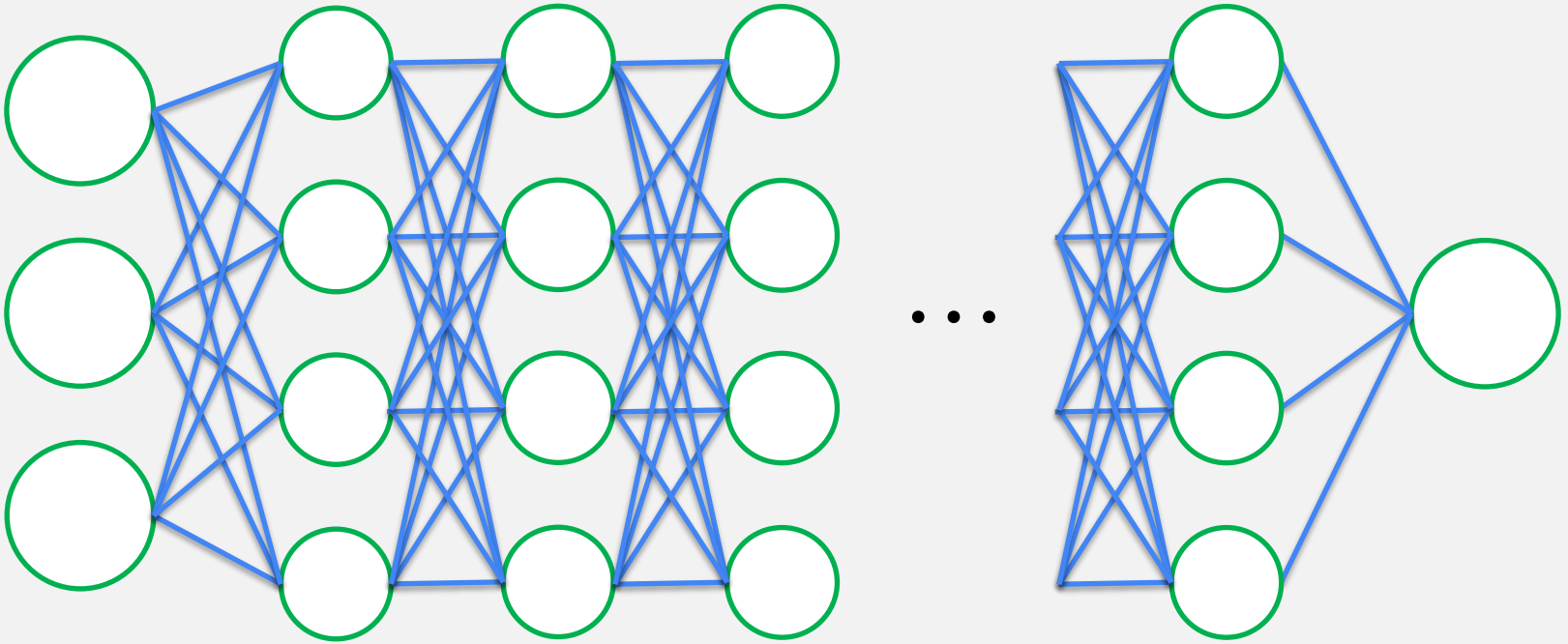


Backward

Error를
반대 반향으로 전파시키며
뉴런의 W와 b 갱신

Big Problem

# Backpropagation: Big problem



Error를 반대 방향으로 전파시킬수록 의미가 약해져 학습 X
-> SVM, Randomforest

# Backpropagation: Breakthrough

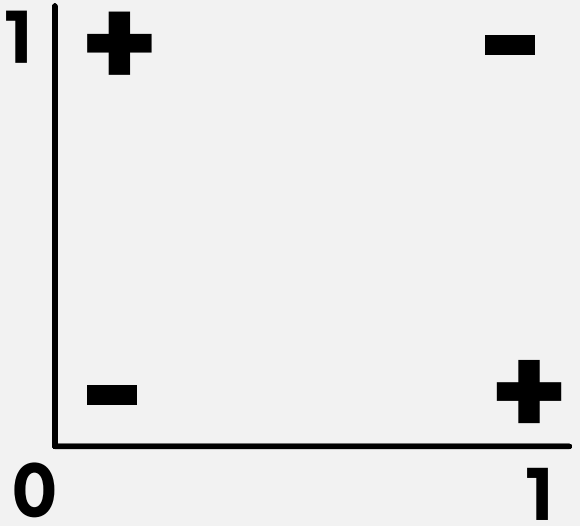
1. NN with many layers could be trained well,
   if the weights are initialized in a clever way.

2. Deep machine learning methods are efficient
   for difficult problems.

-> 유튜브 자막, 페이스북, 넷플릭스, 알파고에 활용

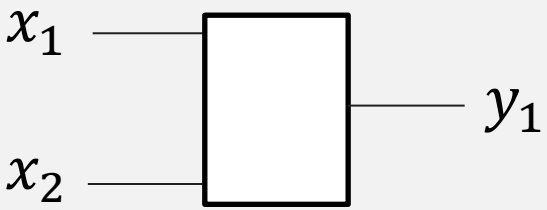
# XOR Problem: Linearly Separable?

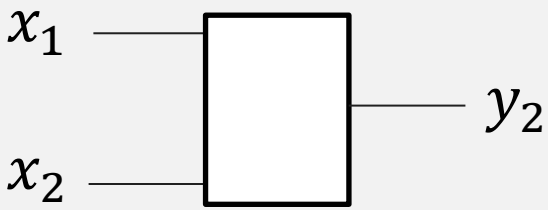
| $x_1$ | $x_2$ | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

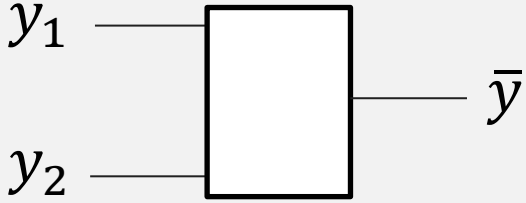



XOR

# XOR Problem: Linearly Separable?

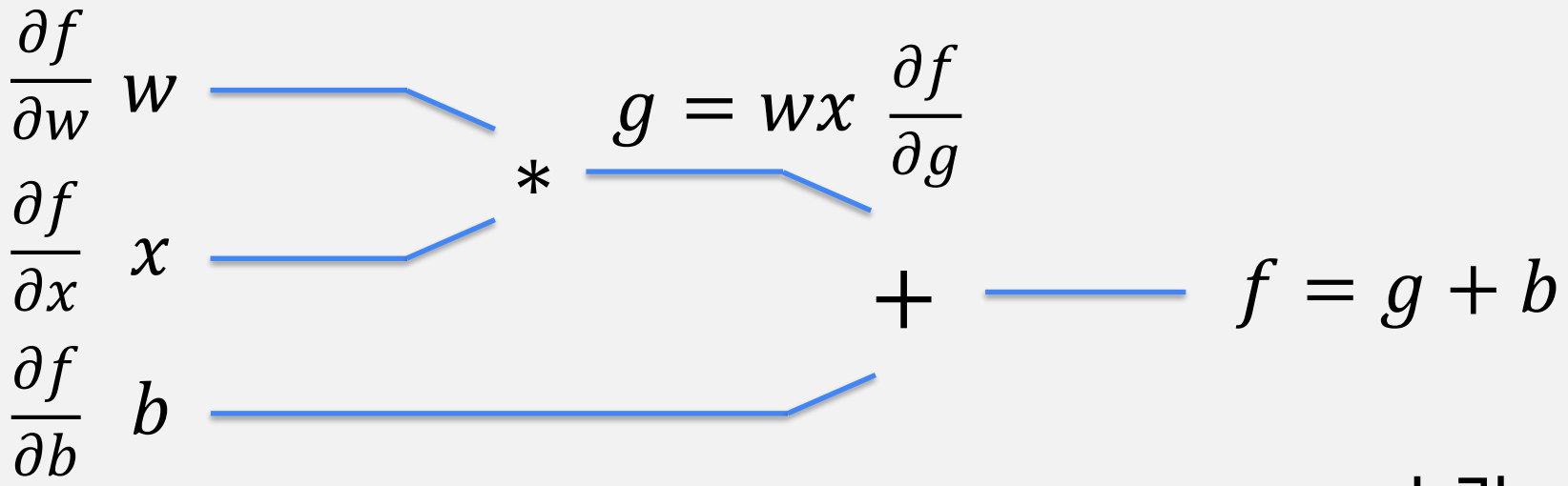



$$w = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \; b = -8 \qquad w = \begin{bmatrix} -7 \\ -7 \end{bmatrix} \; b = 3 \qquad w = \begin{bmatrix} -11 \\ -11 \end{bmatrix} \; b = -6$$

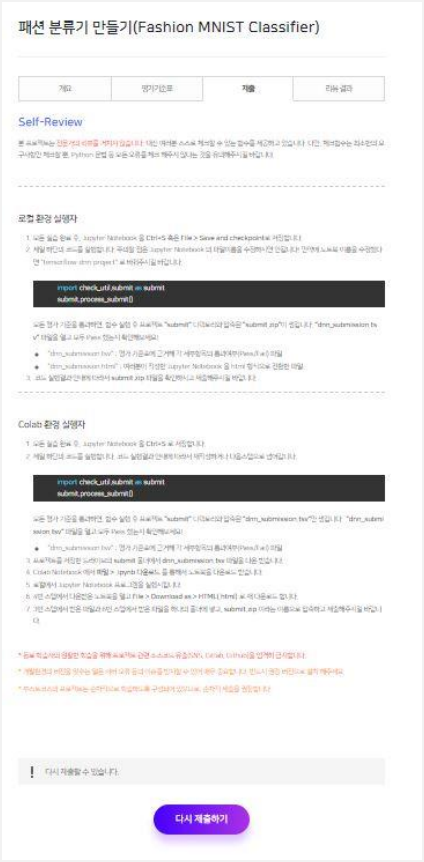
# Backpropagation Algorithm(Chain Rule)




$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial w} = x \qquad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} = w$$

w, x, b가
f에 미치는 영향을
알기 위해 편미분

# Fashion MNIST Classifier




## 패션 분류기 만들기(Fashion MNIST Classifier)

개요 | 평가기준표 | 제출 | 리뷰 결과

### 리뷰어 지정 전

리뷰 #1 | -

프로젝트 평가 | 코드 리뷰 | 수강생 메모 | 리뷰 목록

| 리뷰 | 결과 | 리뷰일 | 리뷰어 |
|---|---|---|---|
| 리뷰 #1 (현재 리뷰) | 리뷰어 지정 전 | 2023.09.27 | - |

# Fashion MNIST Classifier

```
[ ]  # for train
     N = len(train_data)

     ## 코드 시작 ##
     train_dataset = tf.data.Dataset.from_tensor_slices((train_data, train_labels))
     train_dataset = train_dataset.batch(batch_size)
     train_dataset = train_dataset.shuffle(N)
     train_dataset = train_dataset.repeat()
     ## 코드 종료 ##

     print(train_dataset)

     # for test
     ## 코드 시작 ##
     test_dataset = tf.data.Dataset.from_tensor_slices((test_data, test_labels))
     test_dataset = test_dataset.batch(batch_size)
     test_dataset = test_dataset.repeat()
     ## 코드 종료 ##

     print(test_dataset)
```

```
<_RepeatDataset element_spec=(TensorSpec(shape=(None, 784), dtype=tf.float32, name=None),
<_RepeatDataset element_spec=(TensorSpec(shape=(None, 784), dtype=tf.float32, name=None),
```

```
from tensorflow.keras.layers import Dense, BatchNormalization, ReLU
model.add(Dense(512, input_shape = (784,)))
model.add(BatchNormalization())
model.add(ReLU())
model.add(Dense(num_classes, activation = 'softmax'))
```

# Fashion MNIST Classifier

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics = 'accuracy')
```

```
[ ]  ## 코드 시작 ##
     steps_per_epoch = len(train_data) // batch_size
     model.fit(train_dataset, epochs = max_epochs, steps_per_epoch = steps_per_epoch)
     ## 코드 종료 ##
```

```
Epoch 1/5
468/468 [==============================] - 10s 11ms/step - loss: 0.4338 - accuracy: 0.8436
Epoch 2/5
468/468 [==============================] - 9s 18ms/step - loss: 0.3276 - accuracy: 0.8804
Epoch 3/5
468/468 [==============================] - 6s 13ms/step - loss: 0.2860 - accuracy: 0.8952
Epoch 4/5
468/468 [==============================] - 6s 12ms/step - loss: 0.2627 - accuracy: 0.9046
Epoch 5/5
468/468 [==============================] - 10s 20ms/step - loss: 0.2384 - accuracy: 0.9126
<keras.callbacks.History at 0x797f1c0d89d0>
```