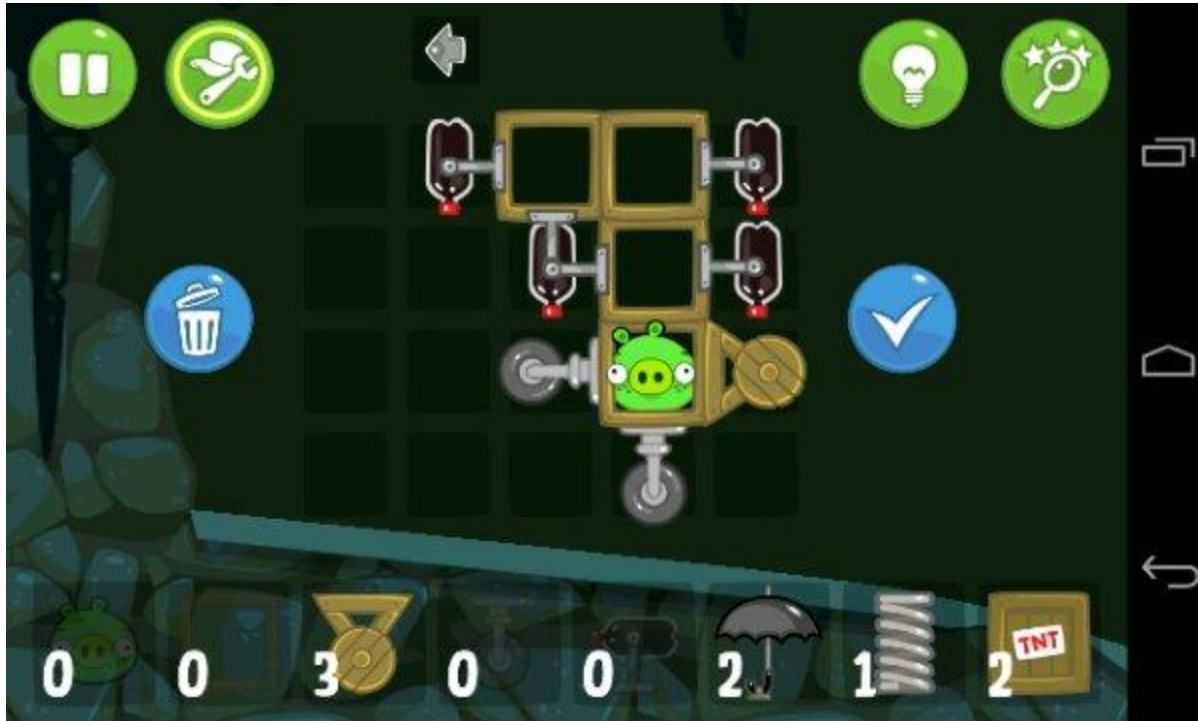


딥러닝 이해하기

leejeyeol92@gmail.com

Introduction

0.Introduction



0.Introduction

딥러닝 프로젝트

Convolution
layer

Batch
Normalization

Whitening

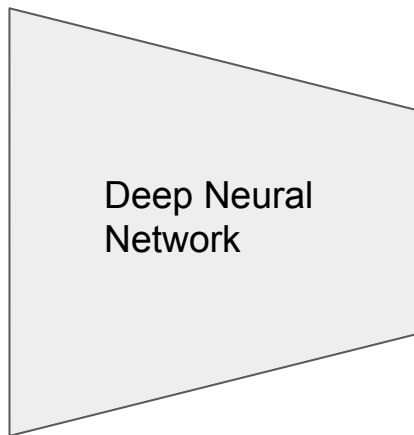
...

Weight
Decay

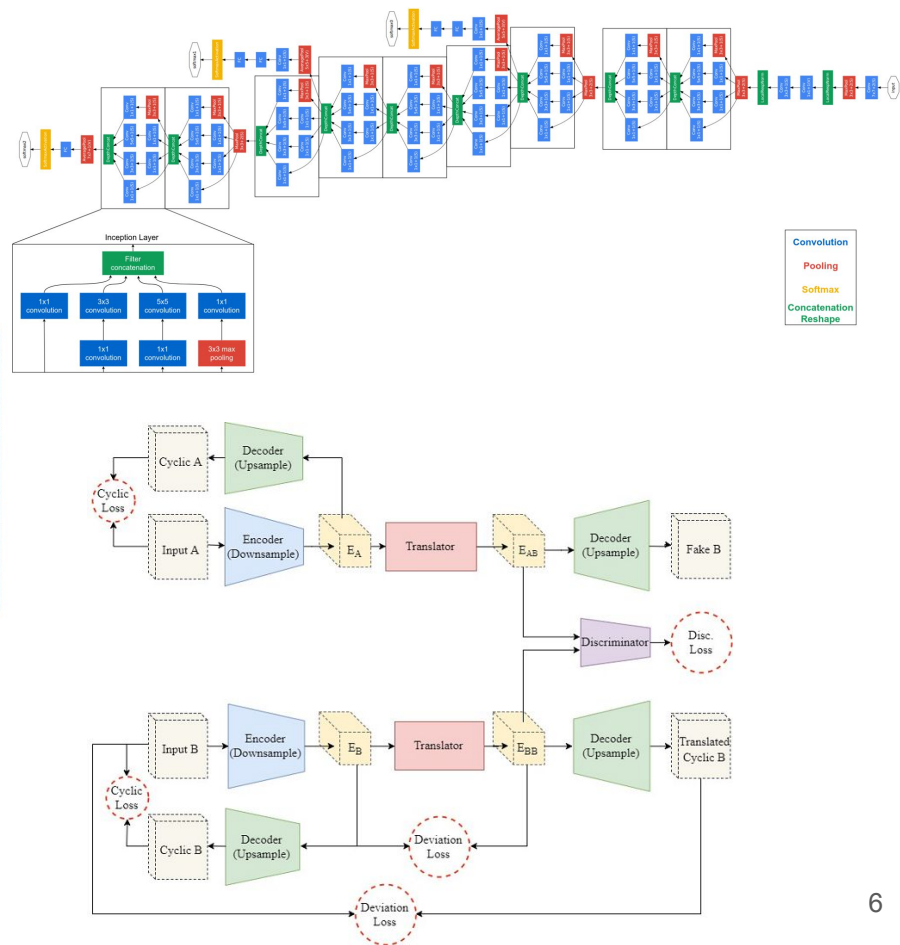
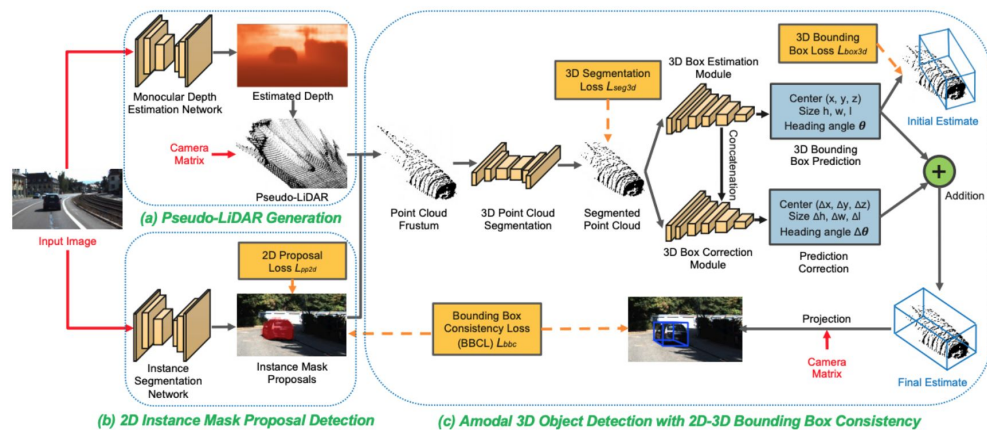
Resnet

Xavier
Initialization

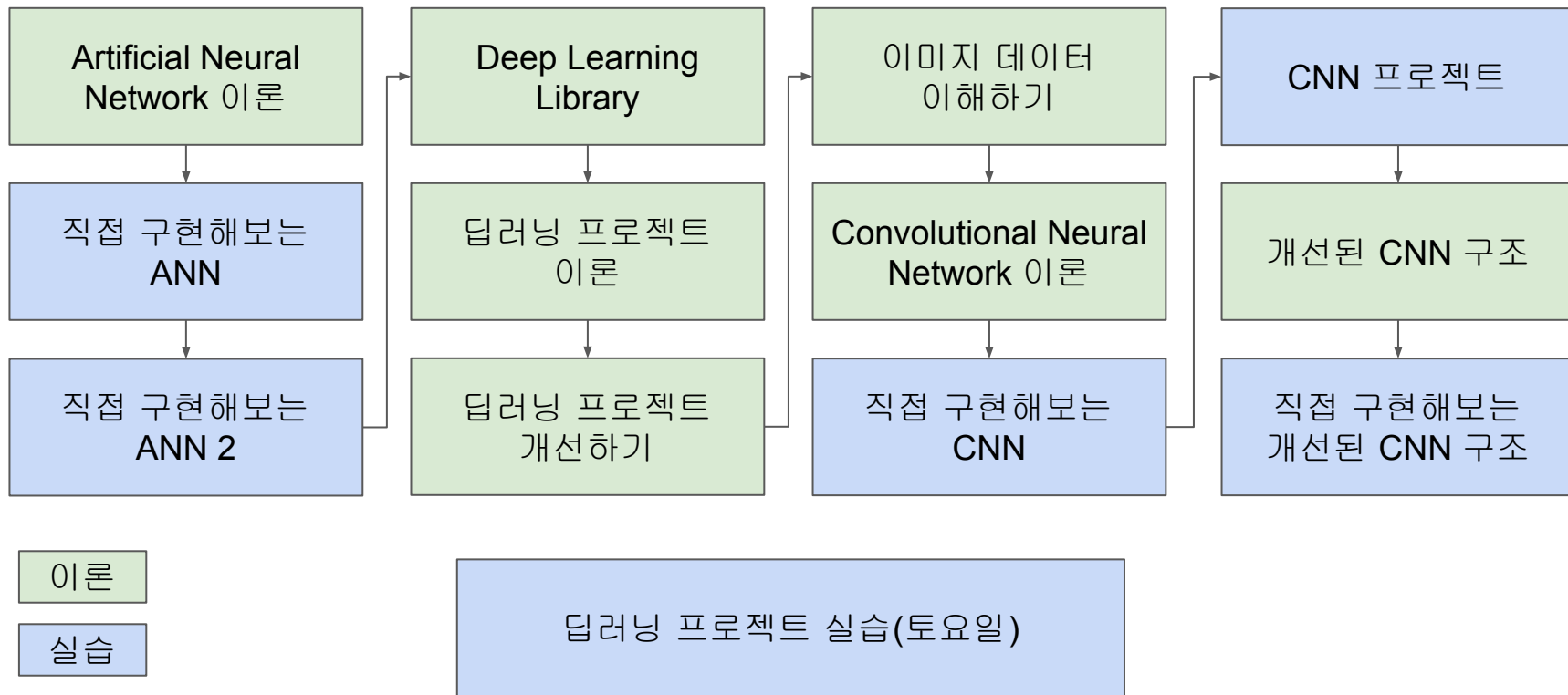
0.Introduction



0.Introduction



0-.진행



0.Introduction

요구사항

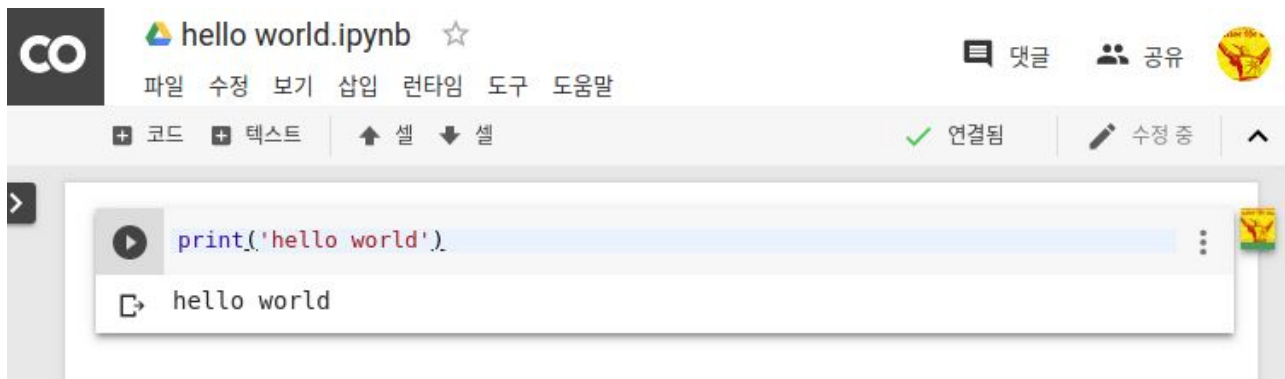
Python3

기초적인 선형 대수학

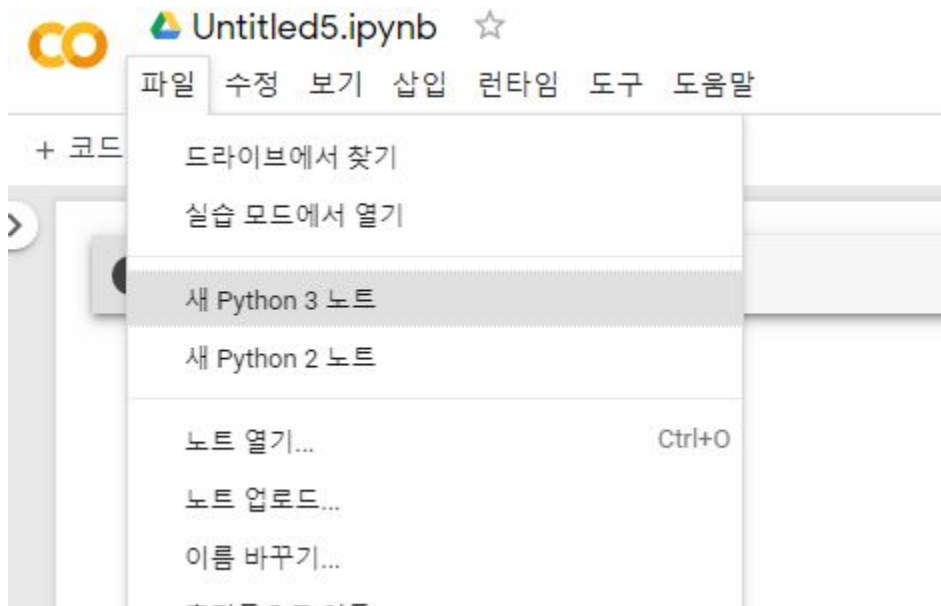
기초적인 미적분

0-.구현 환경 - google colab

- 구글이 제공하는 머신러닝 교육+데이터 분석 도구
- Python 2,3 지원
- Google CoLab = Jupyter Notebook + Google Virtual Machine + Google Drive
- <https://colab.research.google.com>

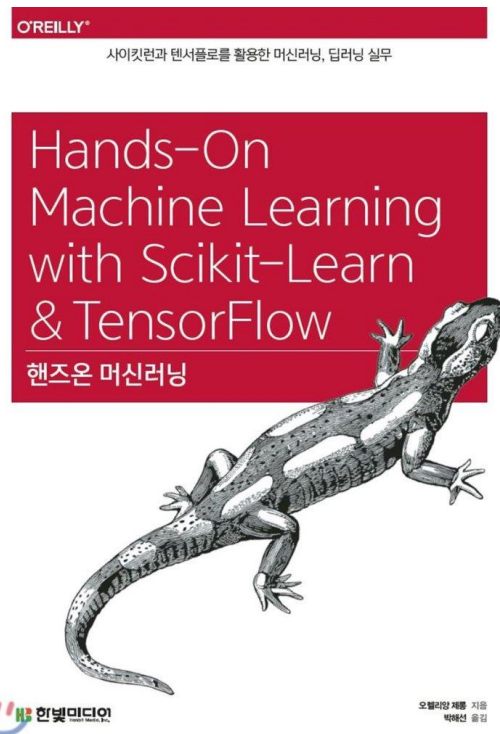
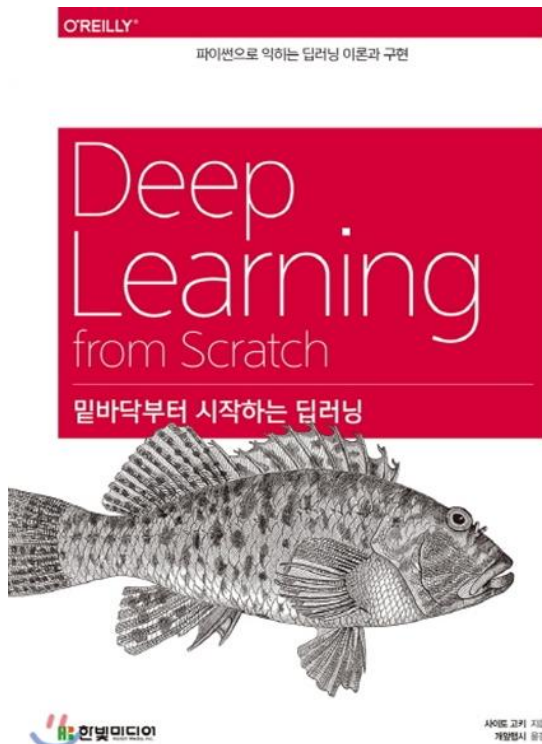


0-.구현 환경 - google colab



0-.구현 환경 - google colab

참고한 도서

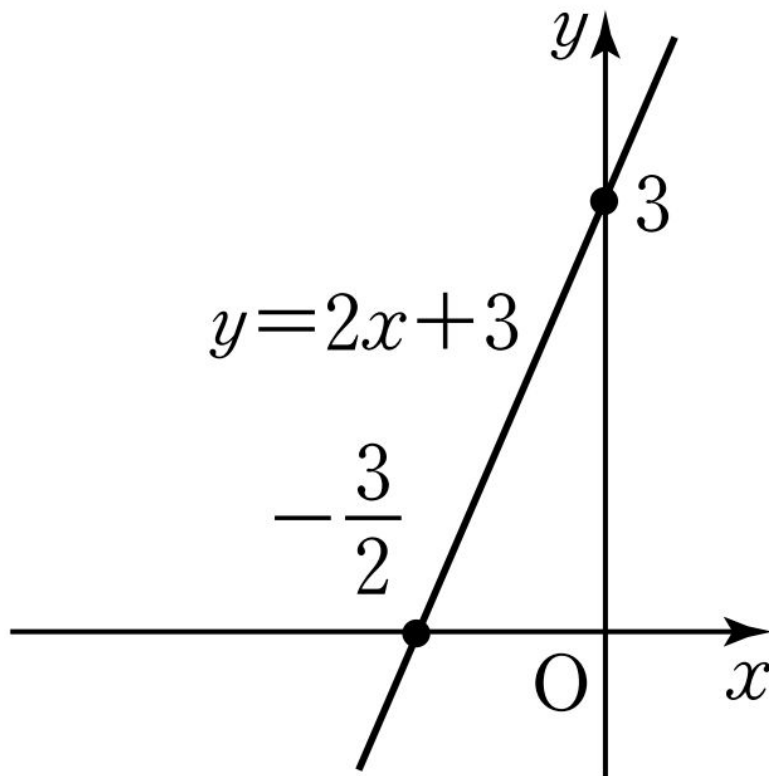


Artificial Neural Network

1차 방정식

$$y = 2x + 3$$

1차 방정식 그래프



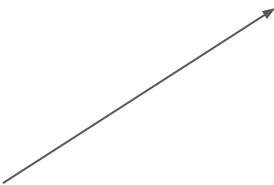
parameter를 가진 1차 방정식


$$y = ax + b$$

parameter

$$y = \boxed{a}x + \boxed{b}$$

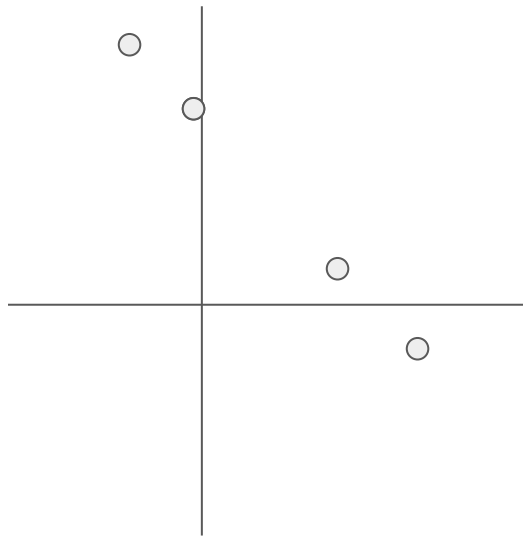
\square : 파라미터


$$y = x$$

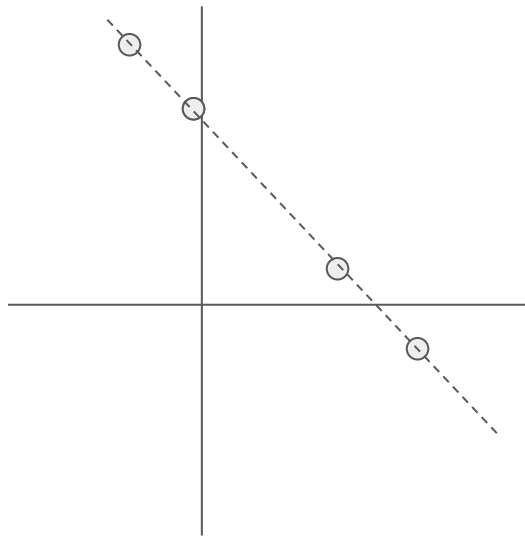

$$y = 2x + 3$$


$$y = 100x + 1000$$

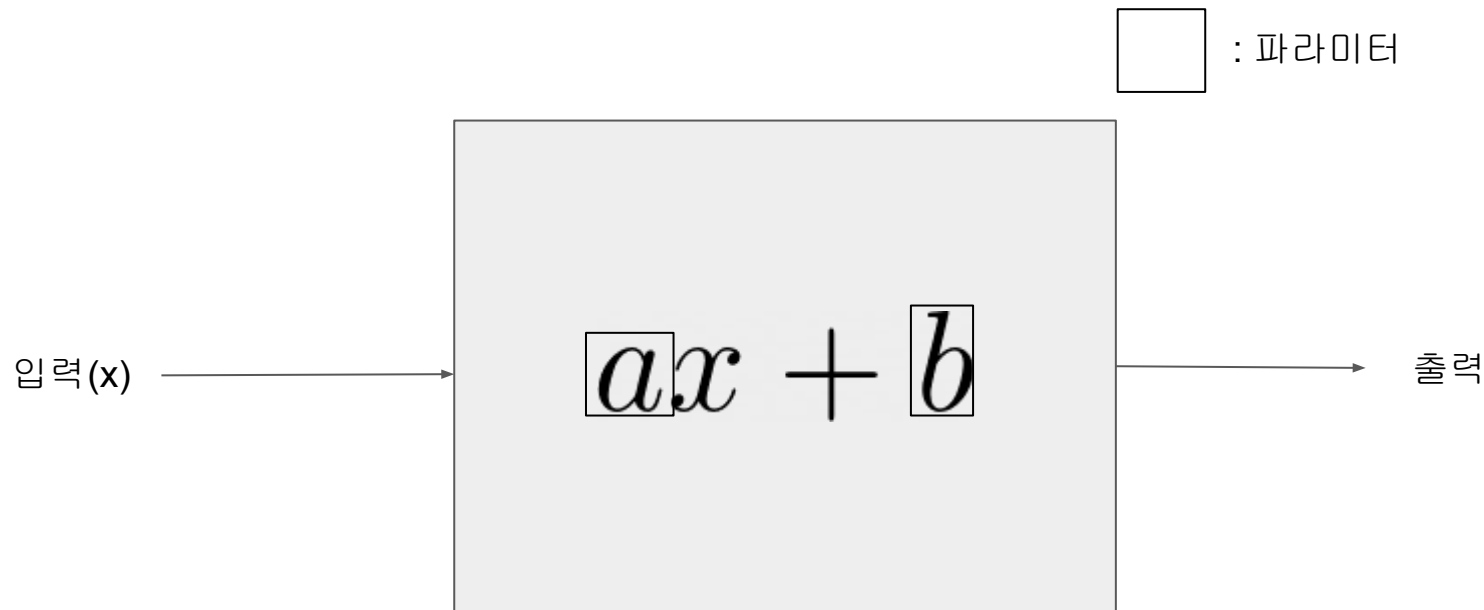
선형 회귀



선형 회귀

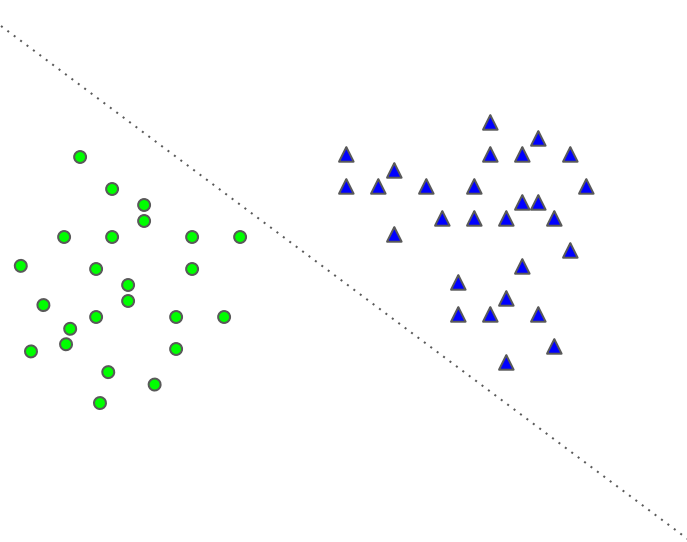


parameter를 가진 1차 방정식 추상화

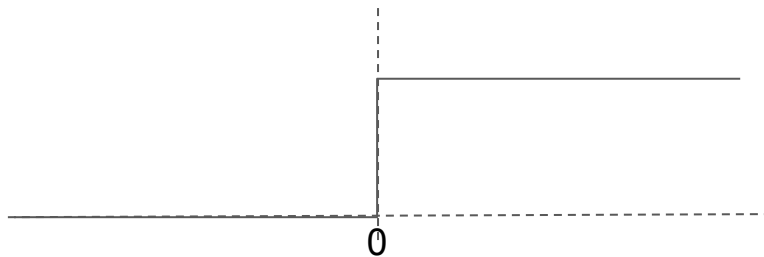


분류 문제

$$h(\boxed{a}x + \boxed{b})$$



$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$



선형방정식 표현

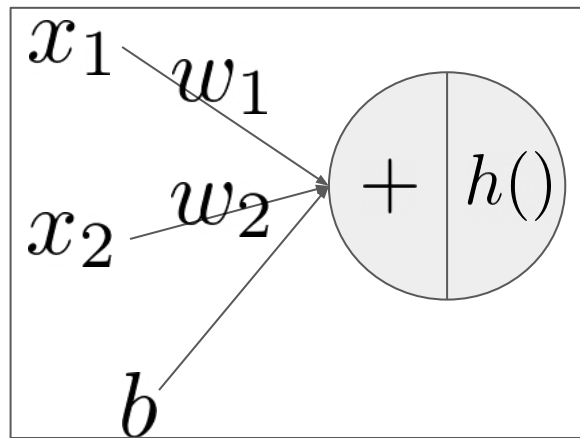
$$\boxed{a}x + \boxed{b}$$



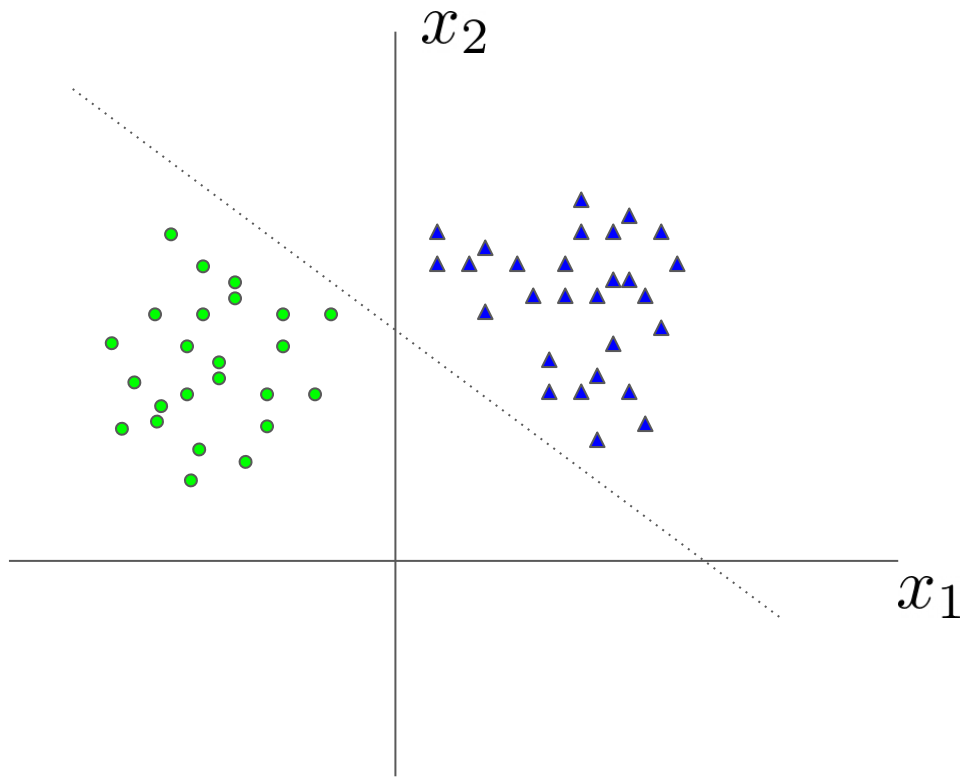
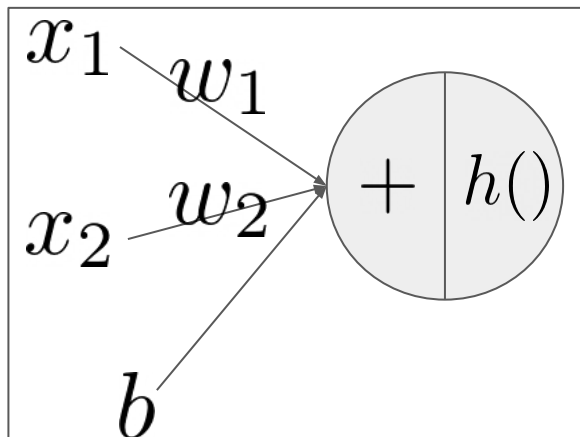
$$\boxed{w_1}x_1 + \boxed{w_2}x_2 + \boxed{b}$$

분류문제 + 선형방정식 다이어그램

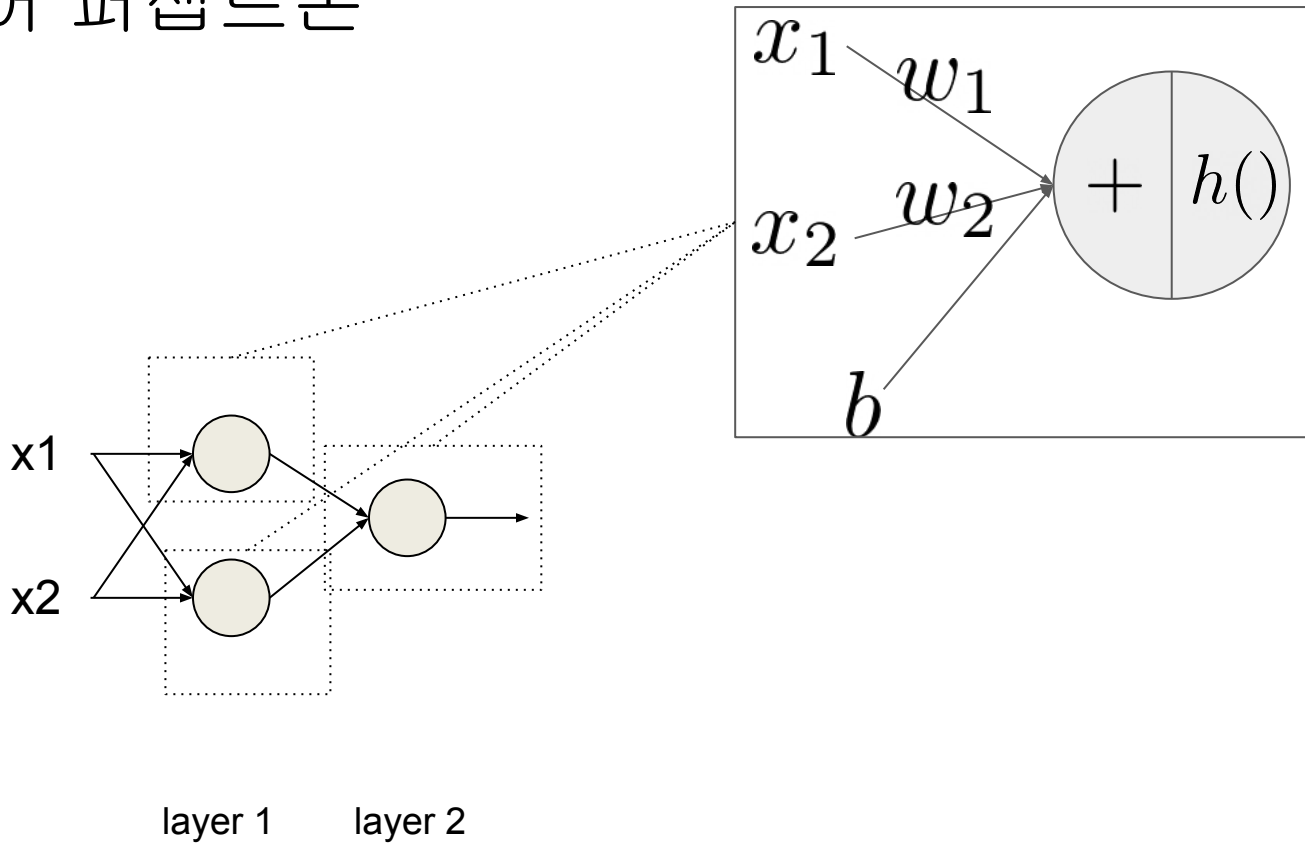
$$h(\boxed{w_1}x_1 + \boxed{w_2}x_2 + \boxed{b})$$



퍼셉트론

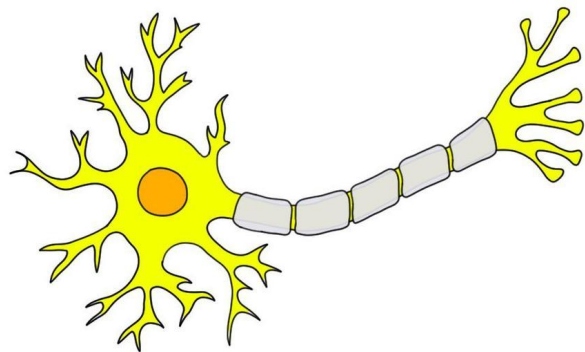


멀티 레이어 퍼셉트론



신경망과 인공신경망

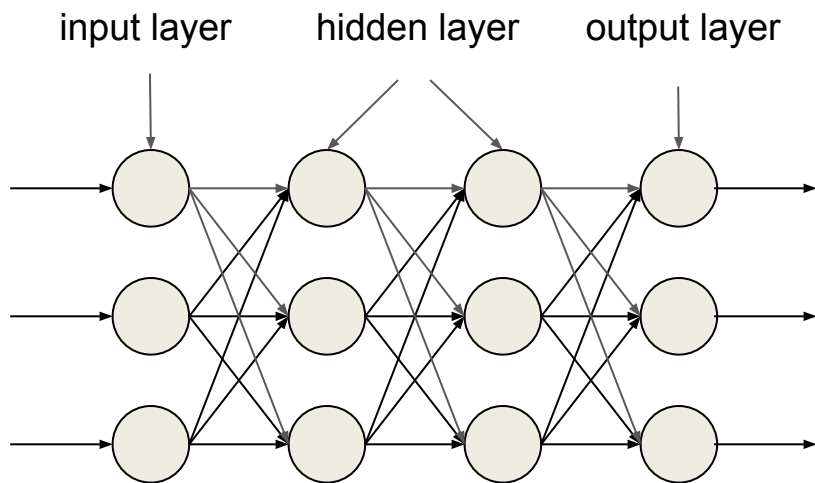
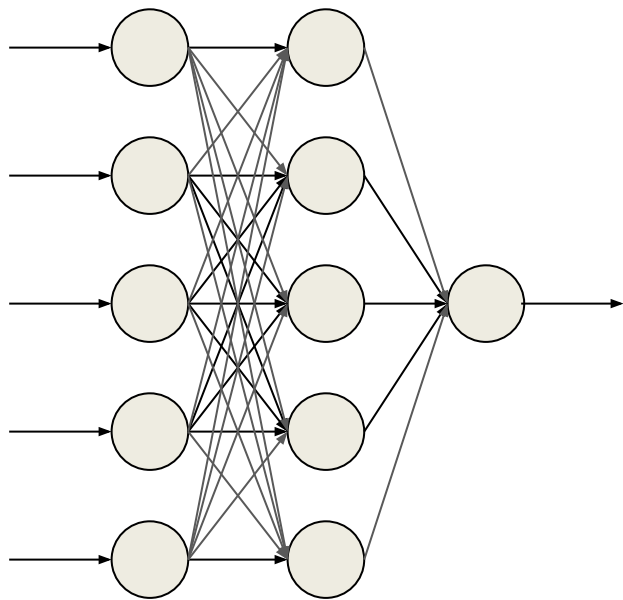
신경 세포(Neuron)



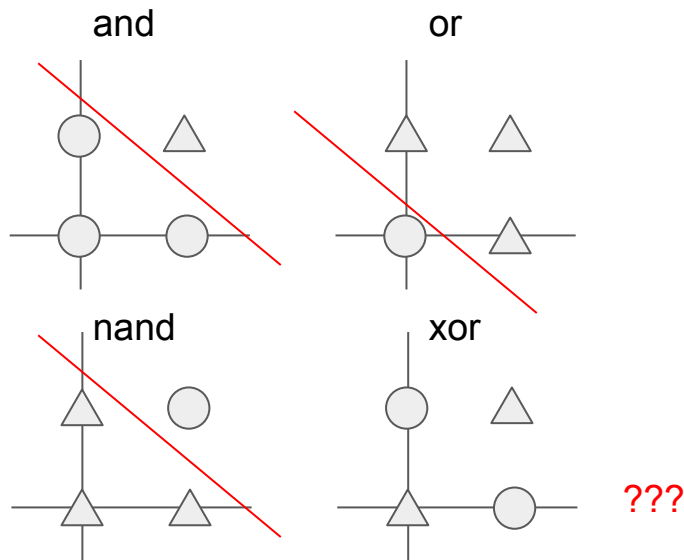
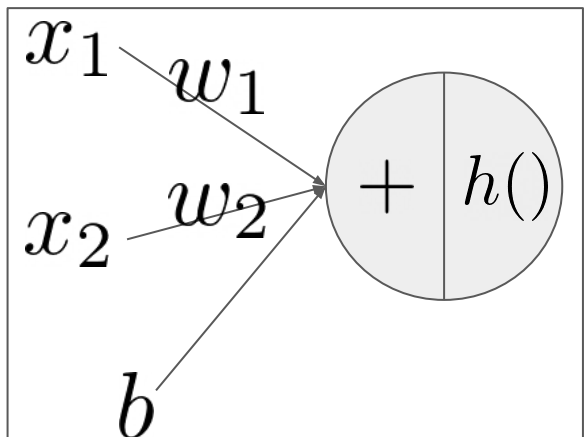
신경망(Neural Network)



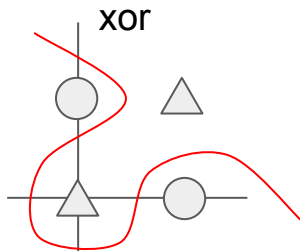
신경망



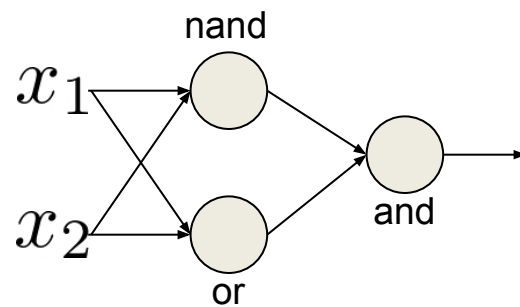
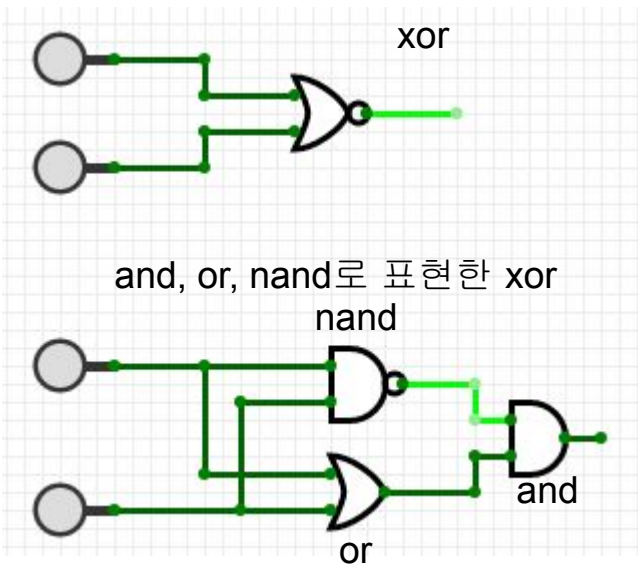
퍼셉트론으로 logic gate 문제 풀기



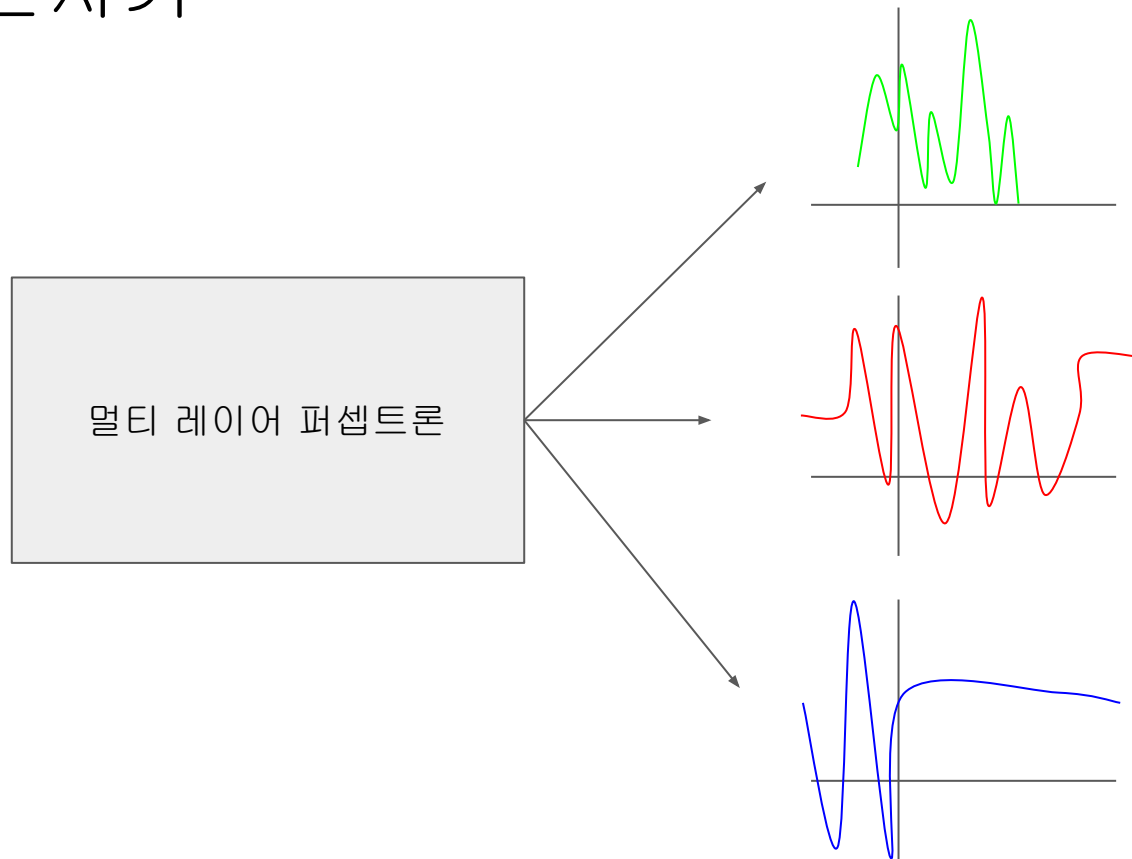
비선형 문제 - xor gate



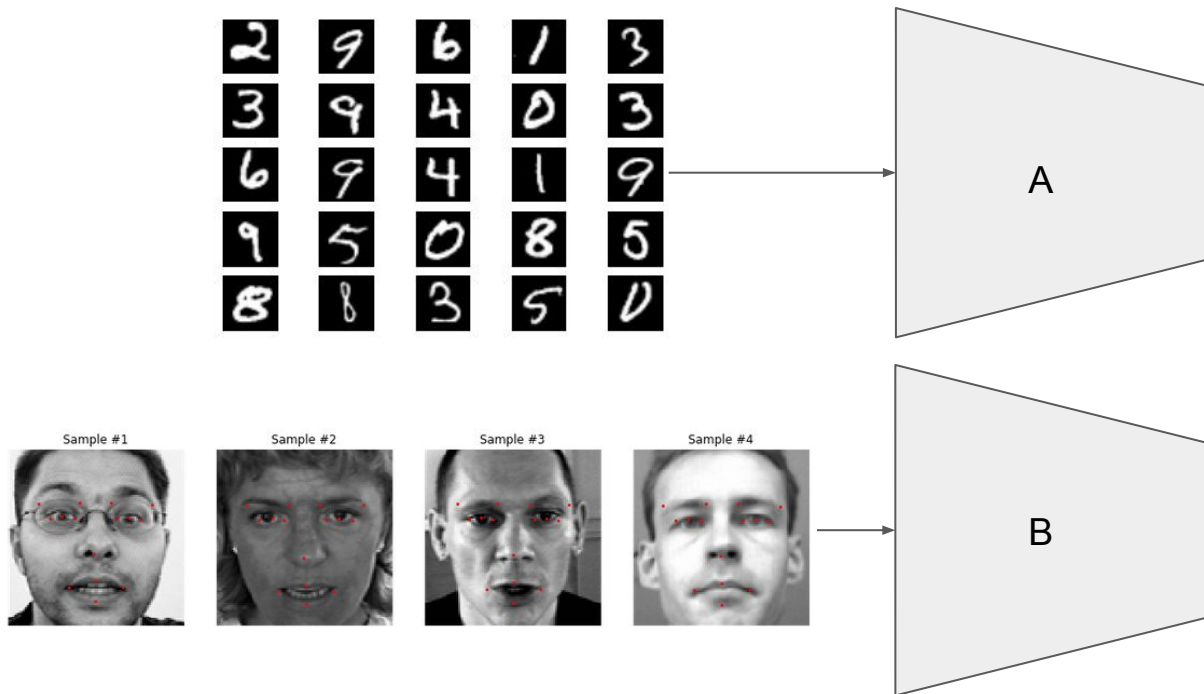
xor문제를 푸는 멀티 레이어 퍼셉트론



범용 함수 근사기



인공신경망의 weight



실습 1

목표 : numpy를 이용한 multi layer perceptron

- logic gate를 근사하는 2 layer multilayer perceptron을 직접 구현해봅시다.
- bias는 문제를 쉽게 하기 위해 생략합니다.

numpy

- scientific computing을 위해 제공되는 python의 기본 패키지입니다.
- 행렬연산 등을 지원합니다.
- 다음과 같이 import하면 np.으로 numpy의 기능들을 사용할 수 있습니다.

```
import numpy as np
```

실습 1

행렬의 곱과 인공 신경망

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

Diagram illustrating matrix multiplication with element-wise calculations:

- Top row calculation: $1 * 5 + 2 * 7 = 19$
- Bottom row calculation: $3 * 5 + 4 * 7 = 22$

```
import numpy as np
A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])
C = np.dot(A,B)

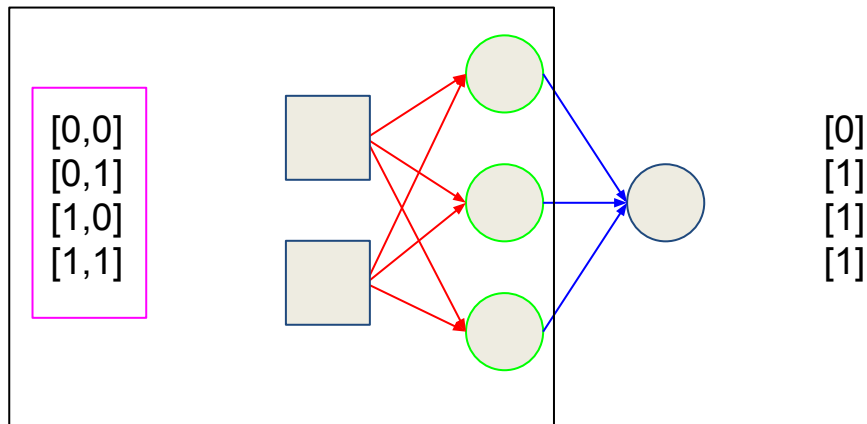
print(C)
```

```
[[19 22]
 [43 50]]
```

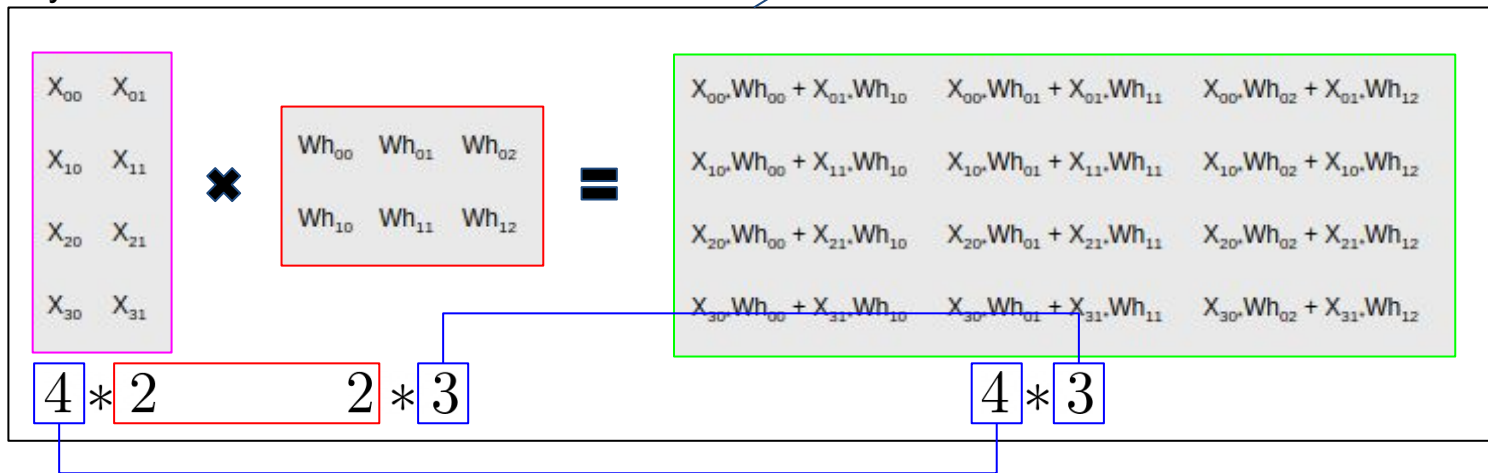
$$w_1x_1 + w_2x_2$$

실습 1

행렬의 곱과 인공 신경망
(and gate)



layer 1의 신경망 계산



실습 1

```
# -*- coding: utf-8 -*-
import numpy as np

# 활성화 함수로 사용할 계단 함수입니다.
def step(x): return np.array(x>0, dtype=np.int)

# 인공적으로 입력과 출력 만들기 X : 입력 Y : 정답(label)
# and 데이터입니다.
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
Y = np.array([[0], [1], [1], [1]])

# 적절한 weight들을 결정합니다.
W_layer_1 =
W_layer_2 =

# 순전파
layer_1_output =
layer_2_output =

# 학습종료 후 결과물 확인 (optional)
print("Input is")
print(X)
print("expected output is")
print(Y)
print("actual output is ")
print(layer_2_output)
```

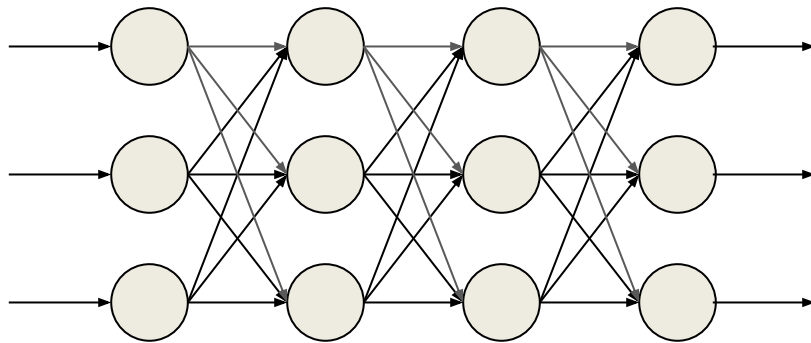
weight의 값은 임의로 넣어보세요. 결과는 틀려도 됩니다. 코드가 돌아가는지 확인해보세요.

행렬곱(np.dot())을 이용하여 신경망 계산을 구현하세요.

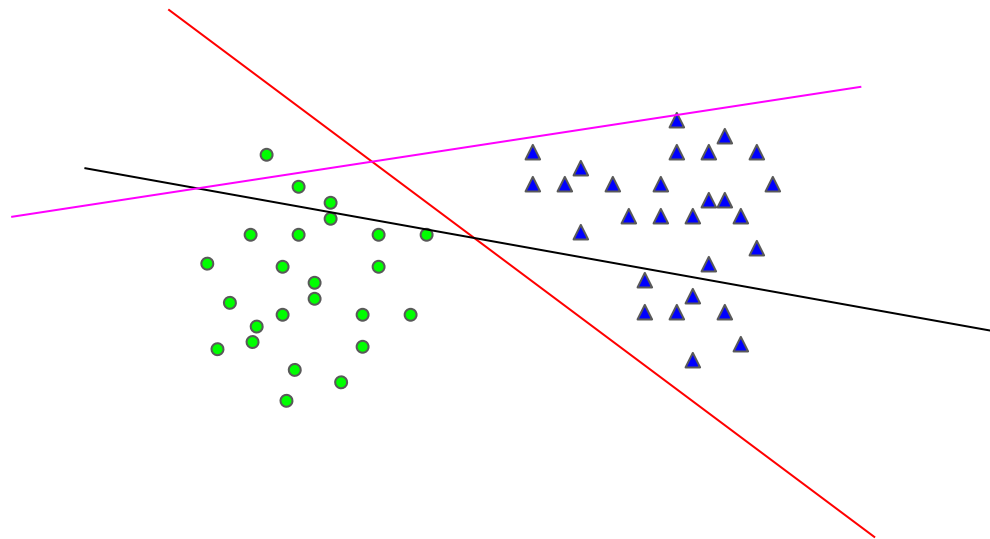
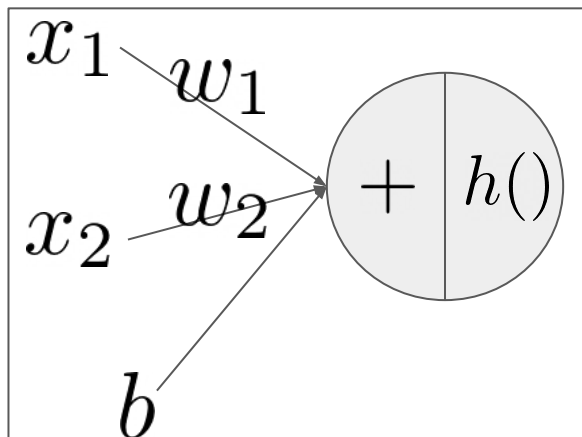
Backpropagation

적절한 weight 찾기

적절한 weight를 찾을 수 있을까?



Loss function



Loss function

Mean Squared Error(MSE)

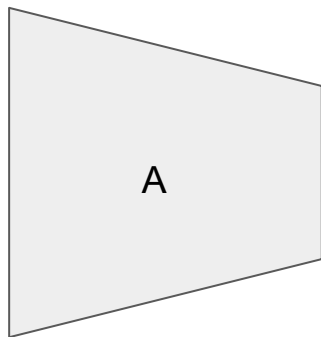
$$E = \frac{1}{2} \sum_k (y_k - t_k)^2$$

y_k 신경망에 입력 데이터를 넣었을 때 나온 신경망 출력

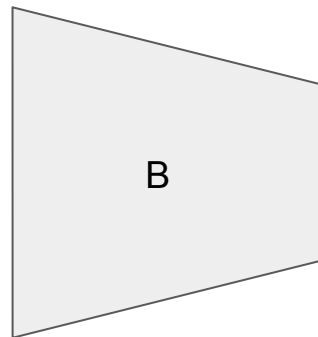
t_k 신경망에 입력 데이터를 넣었을 때 나오기를 기대하는 출력(정답)

Loss Function = Cost function = Error Function

loss function의 의의



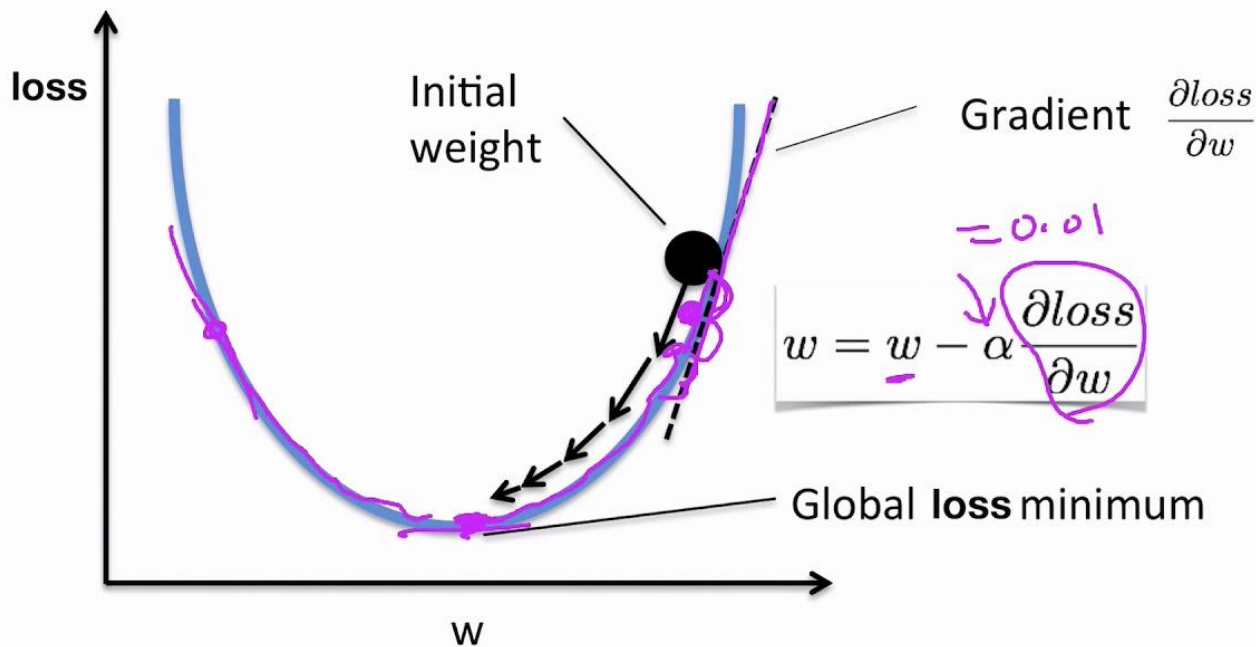
MSE : 120



MSE : 10

Gradient descent

Gradient descent algorithm



$$h(w_1x_1 + w_2x_2 + b)$$

$$E = \frac{1}{2} \sum_k (y_k - t_k)^2$$

Gradient descent

$$E = \frac{1}{2} \sum_k (h(XW) - t_k)^2 \quad (1)$$

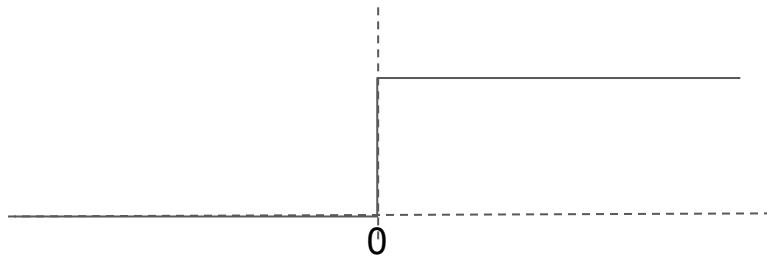
$$\frac{\partial E}{\partial W} \quad (2)$$

$$W = W - \alpha \frac{\partial E}{\partial W} \quad (3)$$

Activation function

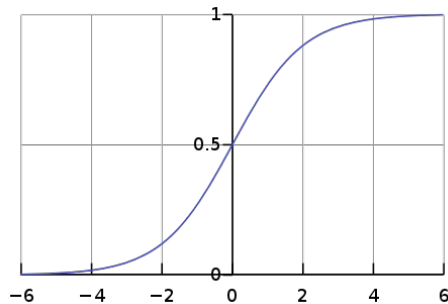
Step function

$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$



Sigmoid function

$$h(x) = \frac{1}{1 + \exp^{-x}}$$



Chain Rule

$$t = x + y$$

$$z = t^2$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \frac{\partial t}{\partial x}$$

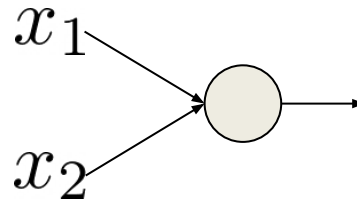
$$\frac{\partial z}{\partial t} = 2t$$

$$\frac{\partial t}{\partial x} = 1$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x + y)$$

Chain Rule - Single Neuron

Chain Rule - single perceptron



$$E = \frac{1}{2} \sum_k g_k^2, g_k = h - t_k$$

$$\frac{\partial g}{\partial h} = 1$$

$$E = \frac{1}{2} \sum_k (h - t_k)^2$$

$$h = \frac{1}{1 + \exp^{-f}}$$

$$\begin{aligned} f &= w_1 x_1 + w_2 x_2 + b \\ &= XW \end{aligned}$$

$$E = \frac{1}{2} \sum_k (h(XW) - t_k)^2$$

↓
 $E(h(f(W)))$

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial h} \frac{\partial h}{\partial f} \frac{\partial f}{\partial W}$$

$$\frac{\partial E}{\partial h} \quad h - t$$

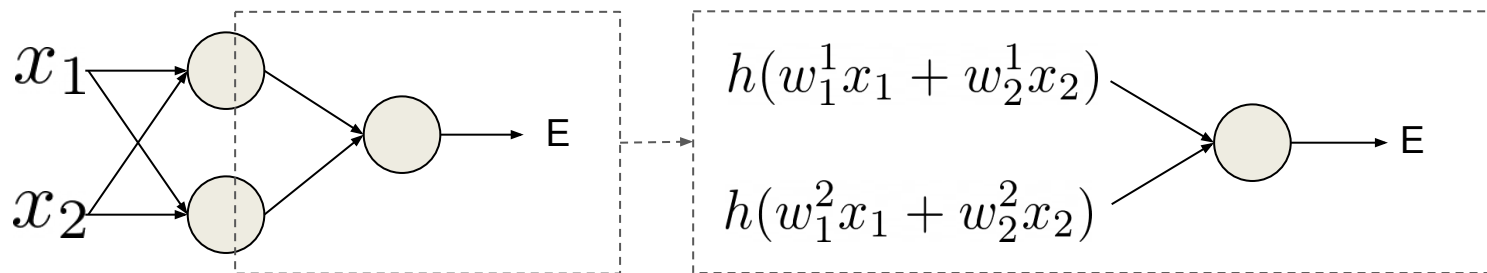
$$\frac{\partial h}{\partial f} \quad f(1 - f)$$

$$\frac{\partial f}{\partial W} \quad X^T$$

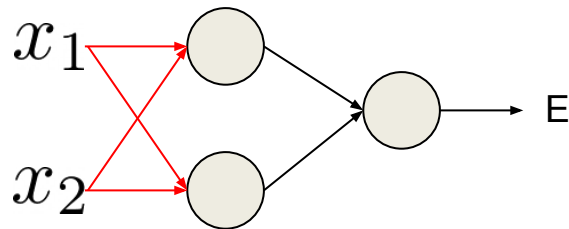
정확히는 미분값이 아니라
입력으로 들어온 X 인데 이를
backpropagation 행렬 계산을 쉽게
하기 위해 전치한것. 계산할 때는
행렬곱의 앞에 위치하게 됨.

$$W = W - \alpha \frac{\partial E}{\partial W}$$

Chain Rule - Multi layer Neural network



$$E(h(f(h(f(W)))))) ???$$



Chain Rule - Multi layer Neural network

$$E(h(f(h(f(W))))))$$

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial h} \frac{\partial h}{\partial f} \frac{\partial f}{\partial h} \frac{\partial h}{\partial f} \frac{\partial f}{\partial W} X^T$$

두번째 레이어 weight gradient 계산 시
사용한 것을 저장.

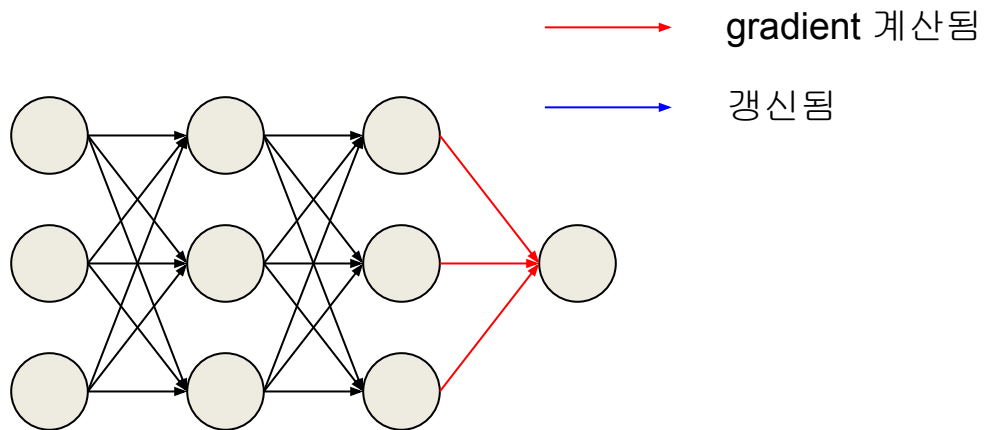
$$f(1 - f)$$

$$f = w_1x_1 + w_2x_2 + b \\ = XW$$

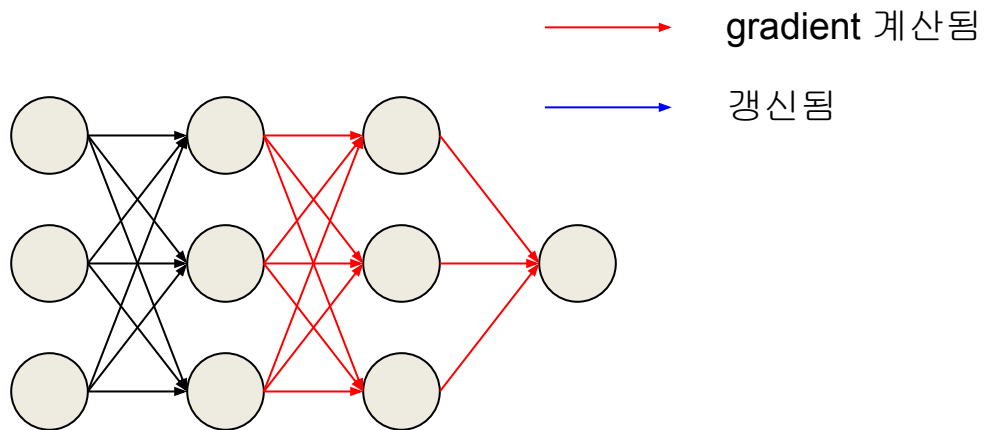
여기서 h 는 이전 레이어의 출력. 즉 식
 f 에서 x 에 해당함. 즉, $\frac{\partial f}{\partial X}$

$$\frac{\partial f}{\partial W} X^T \quad \frac{\partial f}{\partial X} W_{48}^T$$

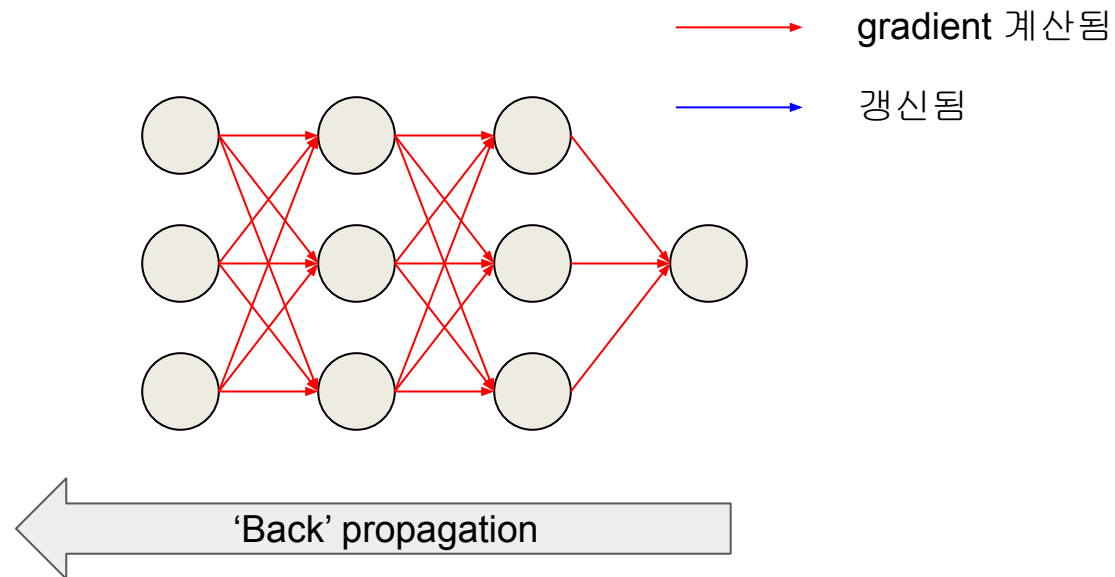
backpropagation



backpropagation

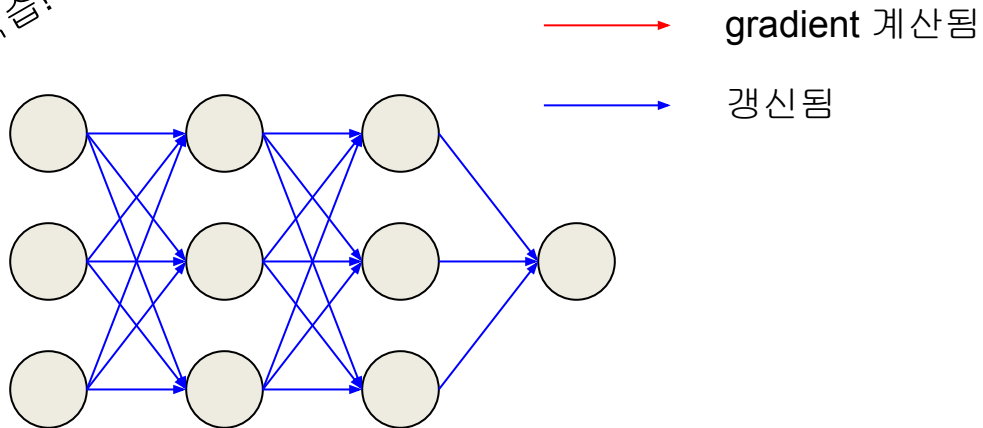


backpropagation



backpropagation

1 step 학습!



실습 2

목표 : **numpy**를 이용한 **backpropagation** 구현

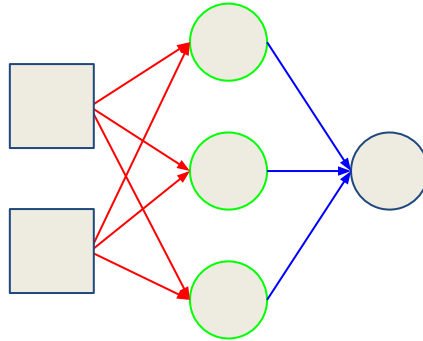
- 실습 1에서 만든 인공신경망을 만들었습니다.
- **weight**를 학습하지 않고 따로 지정했었습니다.
- **backpropagation**을 구현하여 데이터로부터 적절한 **weight**를 학습하도록 만듭니다.
- **bias**는 문제를 쉽게 하기 위해 생략합니다.

실습 2

인공 신경망 학습
(xor gate)

입력

[0,0]
[0,1]
[1,0]
[1,1]



출력

[0]
[1]
[1]
[0]

실습 2

실습1에서 바뀌는 것들

- epoch을 지정합니다. (몇번 학습할지를 결정합니다.)
- learning rate를 지정합니다. (찾은 gradient로 weight를 갱신할때 그 갱신 수준을 결정합니다.)
- activation function으로 step function 대신 sigmoid function을 사용합니다.
- weight 값을 따로 지정하지 않고 랜덤으로 초기화 한 후, 학습으로 개선해나갑니다.
- 순전파(신경망에 입력을 넣고 출력을 확인하는것) 이후 역전파(학습)을 수행합니다.
- layer1의 gradient를 계산하기 위해 layer2 gradient의 일부가 필요합니다.

실습 2

```
# -*- coding: utf-8 -*-
import numpy as np

# epoch은 모든 데이터를 전부 사용하여 한번 학습하는 것을 말합니다. epoch을 20000으로 두면 학습을 20000만번 진행합니다.
epochs = 20000

# 입력, 은닉, 출력 레이어의 노드 수입니다.
inputLayerSize, hiddenLayerSize, outputLayerSize = 2, 3, 1

# weight 갱신 때 사용할 학습률입니다.
learning_rate = 0.1

# 인공적으로 입력과 출력 만들기 X : 입력 Y : 정답(label)
# xor 데이터입니다.
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
t = np.array([[0], [1], [1], [0]])

# 활성화 함수로 사용할 sigmoid와 그 미분을 함수로 선언.
def sigmoid(x): return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x): return x * (1 - x)

# weight matrix를 생성합니다. 수동으로 설정할 필요 없이 uniform 분포에서 샘플링한 랜덤값으로 초기화됩니다.
W_layer_1 = np.random.uniform(size=(inputLayerSize, hiddenLayerSize))
W_layer_2 = np.random.uniform(size=(hiddenLayerSize, outputLayerSize))
...

랜덤으로 만들기 때문에 값은 매번 다르지만, 다음과 같은 형태로 나옵니다.
W_layer_1
array([[0.2157437, 0.88022068, 0.14593536],
       [0.04643253, 0.21375877, 0.94012982]])
W_layer_2
array([[0.41842079,
       [0.40828852],
       [0.09722518]])
...
```


실습 2

```
for i in range(epochs):
    # 순전파
    layer_1_output = sigmoid(np.dot(X, W_layer_1)) # layer 1
    layer_2_output = sigmoid(np.dot(layer_1_output, W_layer_2)) # layer 2

    # loss function 계산. loss function으로 mean squared error를 사용하였습니다.
    E = 1 / 2 * np.square(t - layer_2_output).sum()

    # 역전파 시작
    layer_2_W_grad = page 43 참고
    # delta layer 2(output layer) : partial E/h * partial h/f (partial f/x 는 weight 갱신할 때 곱함)

    layer_1_W_grad = page 45 참고
    # delta layer 1 : partial E/(layer1의)h * partial h/f (partial f/x 는 weight 갱신할 때 곱함)

    # weight 갱신. 갱신된 W = 기존 W - 출발노드의 output * 도착 노드의 delta
    # 각각 partial f/x에 해당하는 해당 레이어로의 입력데이터 행렬이 행렬곱됩니다.
    W_layer_2 += page 43 참고
    W_layer_1 += page 45 참고

# 학습종료 후 결과물 확인 (optional)
print("Input is")
print(X)
print("expected output is")
print(t)
print("actual output is ")
print(layer_2_output)
```

실습 2

결과

```
Input is
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
expected output is
[[0]
 [1]
 [1]
 [0]]
actual output is
[[0.07607478]
 [0.95087552]
 [0.95083983]
 [0.02672599]]
```

실습 2-1

Weight 실험해보기

```
print(W_layer_1)  
print(W_layer_2)
```

```
[[ 5.43873189 -3.11375556  5.29522101]  
 [-3.07384198  5.53159189  5.30037098]]  
[[-8.35124937]  
 [-8.33291174]  
 [11.69026761]]
```

```
# -*- coding: utf-8 -*-  
import numpy as np  
  
# 활성화 함수로 사용할 계단 함수입니다.  
def step(x): return np.array(x>0, dtype=np.int)  
  
# 인공적으로 입력과 출력 만들기 X : 입력 Y : 정답(label)  
# and 데이터입니다.  
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  
Y = np.array([[0], [1], [1], [1]])  
  
# 적절한 weight들을 결정합니다.  
W_layer_1 = np.array([[ 5.43873189, -3.11375556,  5.29522101],  
                      [-3.07384198,  5.53159189,  5.30037098]])  
W_layer_2 = np.array([[-8.35124937],  
                      [-8.33291174],  
                      [11.69026761]])  
  
# 순전파  
layer_1_output = step(np.dot(X, W_layer_1)) # layer 1  
layer_2_output = step(np.dot(layer_1_output, W_layer_2)) # layer 2  
  
# 학습종료 후 결과물 확인 (optional)  
print("Input is")  
print(X)  
print("expected output is")  
print(Y)  
print("actual output is ")  
print(layer_2_output)
```

comma(,)위치 주의

```
Input is  
[[0 0]  
 [0 1]  
 [1 0]  
 [1 1]]  
expected output is  
[[0]  
 [1]  
 [1]  
 [1]]
```

```
actual output is  
[[0]  
 [1]  
 [1]  
 [0]]
```

XOR 근사함수

실습 2-2

학습 시각화

추가

```
Error_list = []  
for i in range(epochs):  
    # 순전파  
    layer_1_output = sigmoid(np.dot(X, W_layer_1)) # layer 1  
    layer_2_output = sigmoid(np.dot(layer_1_output, W_layer_2)) # layer 2  
  
    # loss function 계산. loss function으로 mean squared error를 사용하였습니다.  
    E = 1 / 2 * np.square(t - layer_2_output).sum()  
    Error_list.append(E)  
    # 역전파 시작  
    layer_2_W_grad = page 43 참고  
    # delta layer 2(output layer) : partial E/h + partial h/f (partial f/x 는 weight 갱신할 때 곱함)  
  
    layer_1_W_grad = page 45 참고  
    # delta layer 1 : partial E/(layer1의)h + partial h/f (partial f/x 는 weight 갱신할 때 곱함)  
  
    # weight 갱신. 갱신된 W = 기존 W - 출발노드의 output * 도착 노드의 delta  
    # 각각 partial f/x에 해당하는 해당 레이어로의 입력데이터 행렬이 행렬곱됩니다.  
    W_layer_2 += page 43 참고  
    W_layer_1 += page 45 참고
```

실습 2-2

학습 시각화



```
import matplotlib.pyplot as plt  
plt.plot(Error_list)
```



```
[<matplotlib.lines.Line2D at 0x7fa41c23d2b0>]
```

