수학

목차

- 스칼라, 벡터, 행렬, 텐서
- 퍼셉트론
- 손실 함수
- 경사 하강법

AIoT

스칼라 vs 벡터

- 스칼라(Scala): 하나의 숫자로만 표시 가능. 크기를 의미.
  속력, 질량 등

- 벡터(Vector): 방향과 크기를 모두 가질 수 있음.
  속도, 힘 등

AIoT

스칼라 vs 벡터

문과

"인생은 속도가 아니라 방향이다" [괴테]

속도에 방향도 포함되어 있다!

AIoT

벡터의 표현

- $\mathbf{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ (크기가 2인 열 벡터)
- $\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ (크기가 3인 열 벡터)
- $\mathbf{w} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ (크기가 4인 열 벡터)

벡터의 표현

- $\mathbf{u} = [1 \quad 2]$ (크기가 2인 행 벡터)
- $\mathbf{v} = [1 \quad 2 \quad 3]$ (크기가 3인 행 벡터)
- $\mathbf{w} = [1 \quad 2 \quad 3 \quad 4]$ (크기가 4인 행 벡터)

행 벡터와 열 벡터의 차이는
가로(행)로 표현하거나,
세로(열)로 표현하거나의 차이

AIoT

벡터의 연산

두 벡터의 합, 차, 내적은 크기가 같은 두 벡터에서만 사용 가능

$\mathbf{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ 일 때,

- 두 벡터의 합

$$\mathbf{u} + \mathbf{v} = \begin{bmatrix} 1+3 \\ 2+4 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$$

- 두 벡터의 차

$$\mathbf{u} - \mathbf{v} = \begin{bmatrix} 1-3 \\ 2-4 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

벡터의 연산

$\mathbf{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}, k = 3$일 때,

- 스칼라 배

$$k\mathbf{u} = 3\begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

$$k\mathbf{v} = 3\begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 9 \\ 12 \end{bmatrix}$$

- 내적 (dot product)

$$\mathbf{u} \cdot \mathbf{v} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 3 \\ 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

행렬(matrix)

- 말 그대로 행(가로)과 열(세로)로 이루어짐.
  n x m 행렬은 행의 개수가 n, 열의 개수가 m인 행렬을 표현

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ (2 x 2 행렬)

$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ (2 x 3 행렬)

$C = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$ (3 x 2 행렬)

행렬의 연산

두 행렬의 합과 차는 크기가 같은 두 행렬에서만 사용 가능.

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$일 때,

- 두 행렬의 합(차도 동일)

$$A + B = \begin{bmatrix} 1+7 & 2+8 \\ 3+9 & 4+10 \\ 5+11 & 6+12 \end{bmatrix} = \begin{bmatrix} 8 & 10 \\ 12 & 14 \\ 16 & 18 \end{bmatrix}$$

행렬의 연산

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, k = 2$일 때,

- 스칼라 배

$$kA = 2\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{bmatrix}$$

행렬의 연산 – 두 행렬의 곱

- 두 행렬의 곱

두 행렬의 곱은 앞 행렬의 열의 크기와
뒤 행렬의 행의 크기가 같아야 함.
교환법칙이 성립하지 않는다.

n x k인 행렬 A와 k x m인 행렬 B의 곱 AxB의 크기는 n x m이 됨.

행렬의 연산 – 두 행렬의 곱

두 행렬의 곱을 구하는 식은 다음과 같다.

$$(A \times B)_{ij} = \sum_{l=1}^{k} A_{il} B_{lj}$$

행렬의 연산 – 두 행렬의 곱

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$에 대해

$$A \times B = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ \mathrm{A}_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

$$= \begin{bmatrix} 5 + 14 & 6 + 16 \\ 15 + 28 & 18 + 32 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

AIoT

여러가지 행렬

- 정사각 행렬: 행의 크기와 열의 크기가 같은 행렬

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- 전치 행렬: 기존 행렬의 행과 열을 서로 뒤바꾼 행렬, 원래 행렬에 윗첨자 T를 붙여 표현

$$A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

AIoT

여러가지 행렬

- 대각 행렬: 정사각 행렬이면서 주 대각선(좌상->우하)의 성분만 존재하는 행렬

$$D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

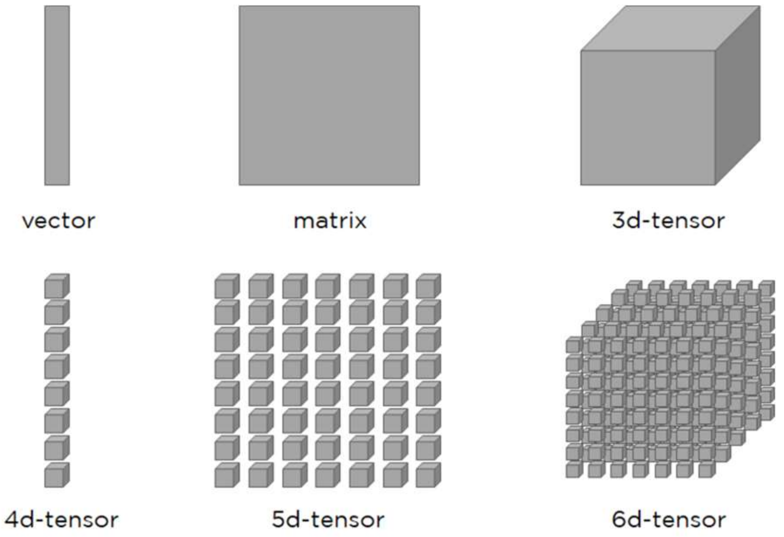
- 단위(또는 항등) 행렬: 주 대각선의 성분이 1인 대각 행렬

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

텐서(Tensor)

- 스칼라는 수 하나, 벡터는 1차원, 행렬은 2차원을 표현할 수 있었다.

- 텐서는 3차원 이상을 표현한다.



AIoT

텐서(Tensor)

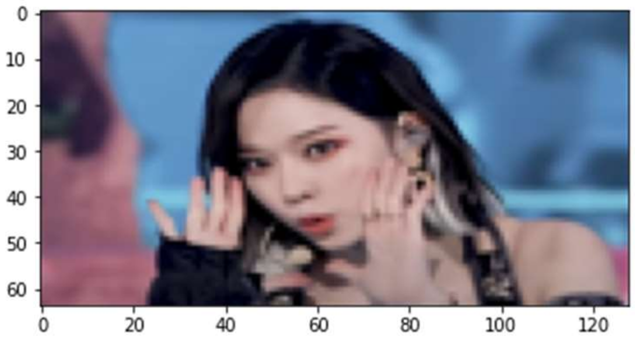
- 텐서를 이용하면 RGB 이미지를 표현할 수 있게 된다.

가로 128 세로 64의 RGB 이미지 한장은
3(색상)x128(가로)x64(세로)의 텐서로 표현 가능하다.

텐서(Tensor)

- 예시

128x64의 RGB



```
torch.Size([3, 128, 64])
```

텐서로 변환한 후 텐서의 크기(shape)

```
tensor([[[0.3176, 0.3961, 0.5686,  ..., 0.2353, 0.2118, 0.1961],
         [0.3098, 0.3843, 0.5490,  ..., 0.2353, 0.2157, 0.2000],
         [0.3098, 0.3765, 0.5294,  ..., 0.2235, 0.2039, 0.1922],
         ...,
         [0.5608, 0.5647, 0.5529,  ..., 0.1255, 0.2471, 0.6118],
         [0.5333, 0.5608, 0.5608,  ..., 0.1176, 0.1098, 0.4980],
         [0.5020, 0.5608, 0.5725,  ..., 0.1216, 0.1137, 0.3608]],

        [[0.2431, 0.2863, 0.4118,  ..., 0.3725, 0.3412, 0.3255],
         [0.2353, 0.2784, 0.4000,  ..., 0.3765, 0.3451, 0.3294],
         [0.2353, 0.2745, 0.3804,  ..., 0.3647, 0.3373, 0.3216],
         ...,
         [0.3294, 0.3333, 0.3255,  ..., 0.1216, 0.1961, 0.4667],
         [0.3137, 0.3373, 0.3373,  ..., 0.1137, 0.0784, 0.3961],
         [0.2980, 0.3333, 0.3451,  ..., 0.1216, 0.0941, 0.2902]],

        [[0.3216, 0.3529, 0.4588,  ..., 0.5098, 0.4745, 0.4510],
         [0.3137, 0.3490, 0.4471,  ..., 0.5137, 0.4784, 0.4549],
         [0.3137, 0.3451, 0.4353,  ..., 0.5020, 0.4706, 0.4471],
         ...,
         [0.4039, 0.4078, 0.3961,  ..., 0.1882, 0.2431, 0.5412],
         [0.3882, 0.4078, 0.4039,  ..., 0.1804, 0.1294, 0.4784],
         [0.3725, 0.4078, 0.4118,  ..., 0.1882, 0.1490, 0.3725]]])
```

텐서의 값

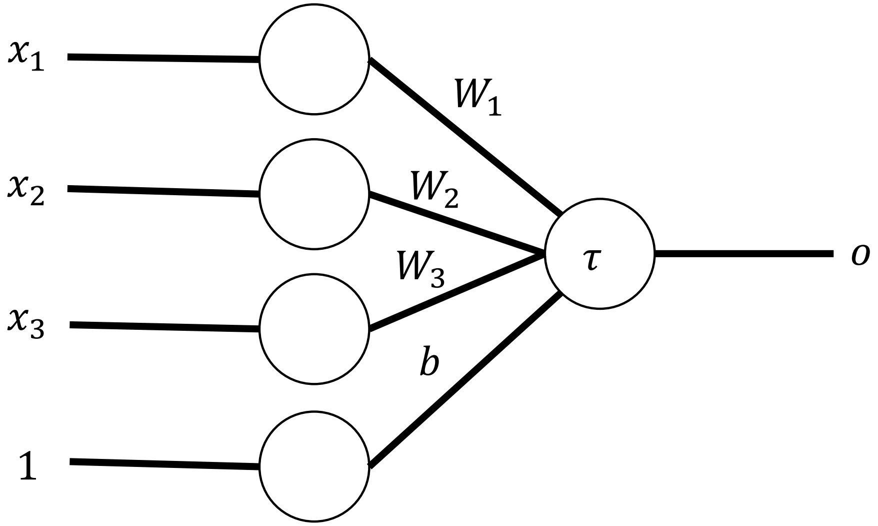
AIoT

텐서(Tensor)

- 6장에서 다룰 Convolutional Neural Network에서는 이러한 RGB 이미지를 텐서로 변환하여 모델의 입력으로 사용할 예정.

퍼셉트론(Perceptron)

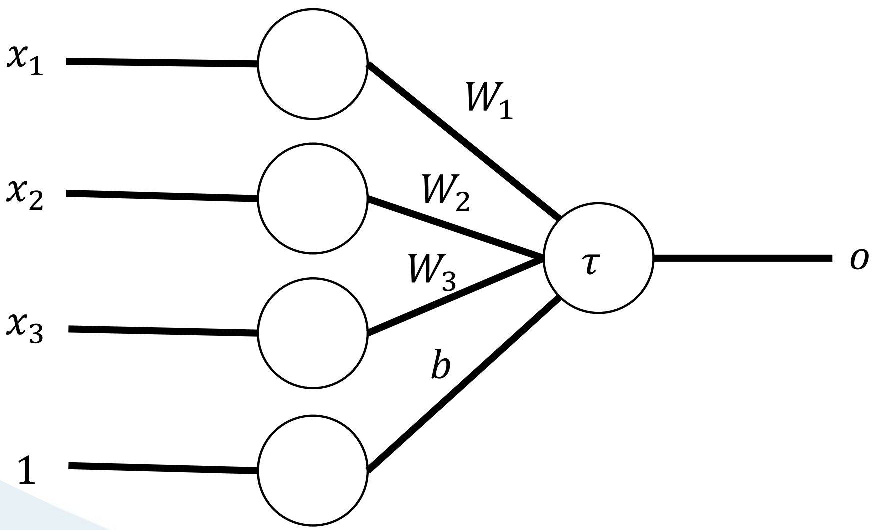
- 인공신경망의 한 종류. 딥러닝의 가장 기본적인 구조
  입력층(x)과 출력층(o)이 존재



AIoT

퍼셉트론

• 출력 o는 다음과 같이 계산됨



$$o = \tau(x_1 W_1 + x_2 W_2 + x_3 W_3 + b)$$
$$= \tau(x^T \cdot W + b)$$

활성 함수(τ)로 다음과 같은 계단 함수를 사용한다면

$$\tau(x) = \{ \begin{matrix} 1 \ (x \geq 0) \\ -1 \ (x < 0) \end{matrix}$$

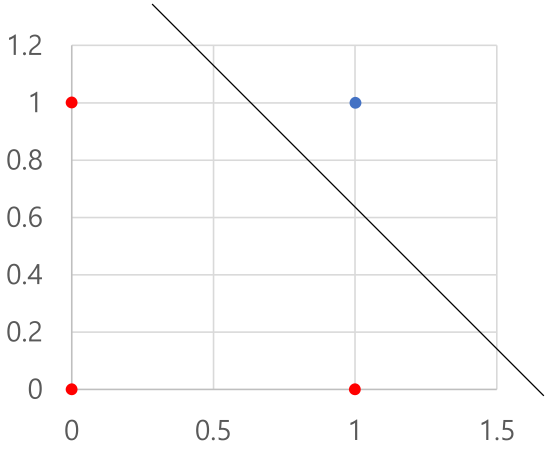
활성 함수의 출력에 따라 두 가지로 분류 가능

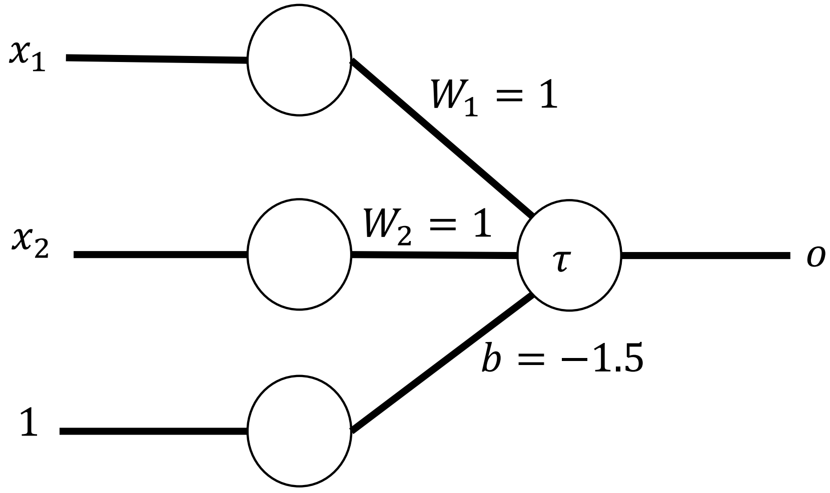
AIoT

퍼셉트론

- AND 분류기

x_1	x_2	o
0	0	-1
0	1	-1
1	0	-1
1	1	1





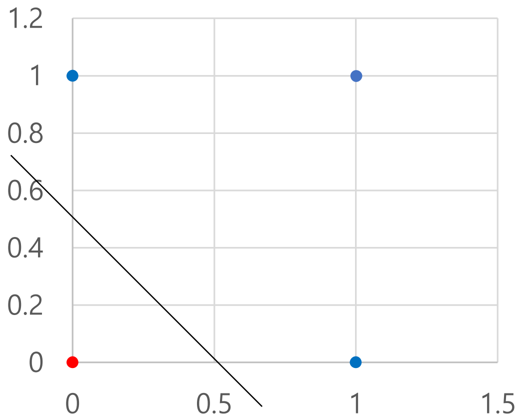
$$\tau(x) = \begin{cases} 1 \ (x \geq 0) \\ -1 \ (x < 0) \end{cases}$$

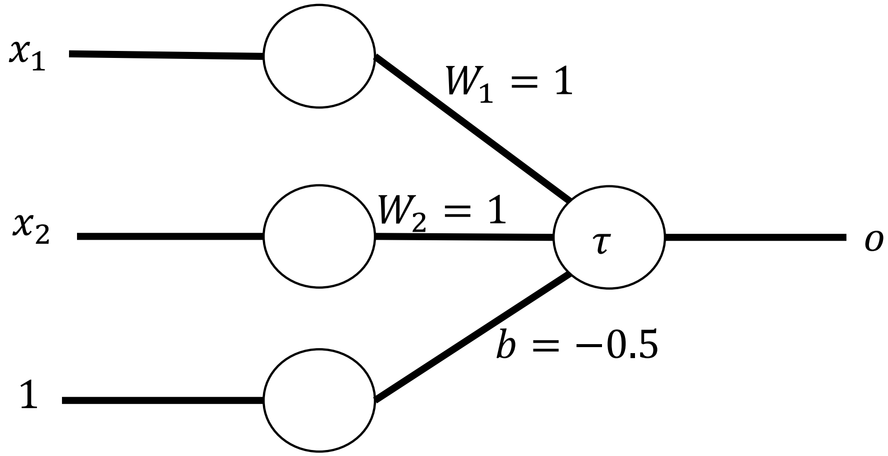
AIoT

퍼셉트론

- OR 분류기

x_1	x_2	o
0	0	-1
0	1	1
1	0	1
1	1	1





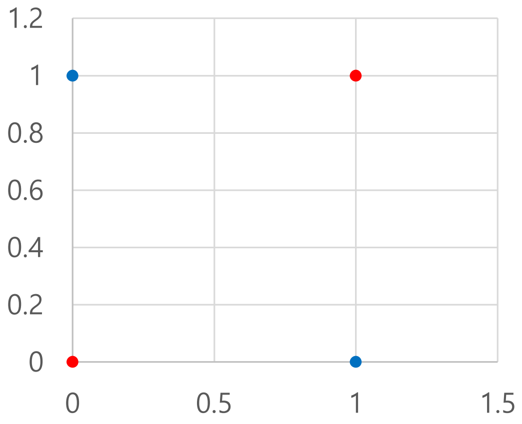
$$\tau(x) = \begin{cases} 1 \ (x \geq 0) \\ -1 \ (x < 0) \end{cases}$$

AIoT

퍼셉트론

- XOR 분류기 ?

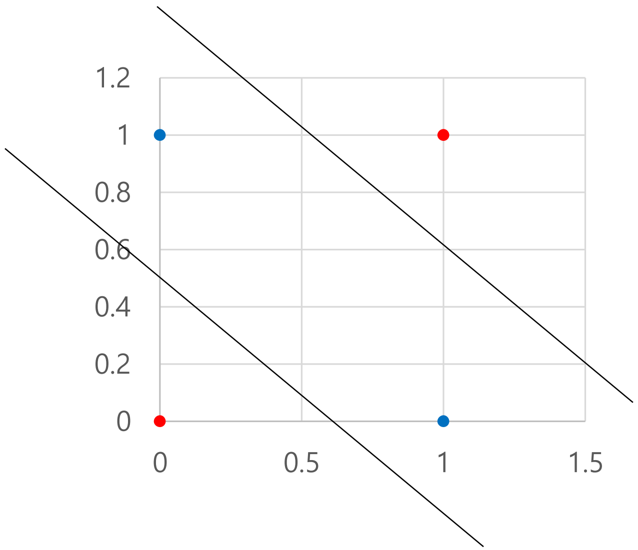
x_1	x_2	o
0	0	-1
0	1	1
1	0	1
1	1	-1



하나의 퍼셉트론,
직선 하나로는
XOR 분류 불가

퍼셉트론

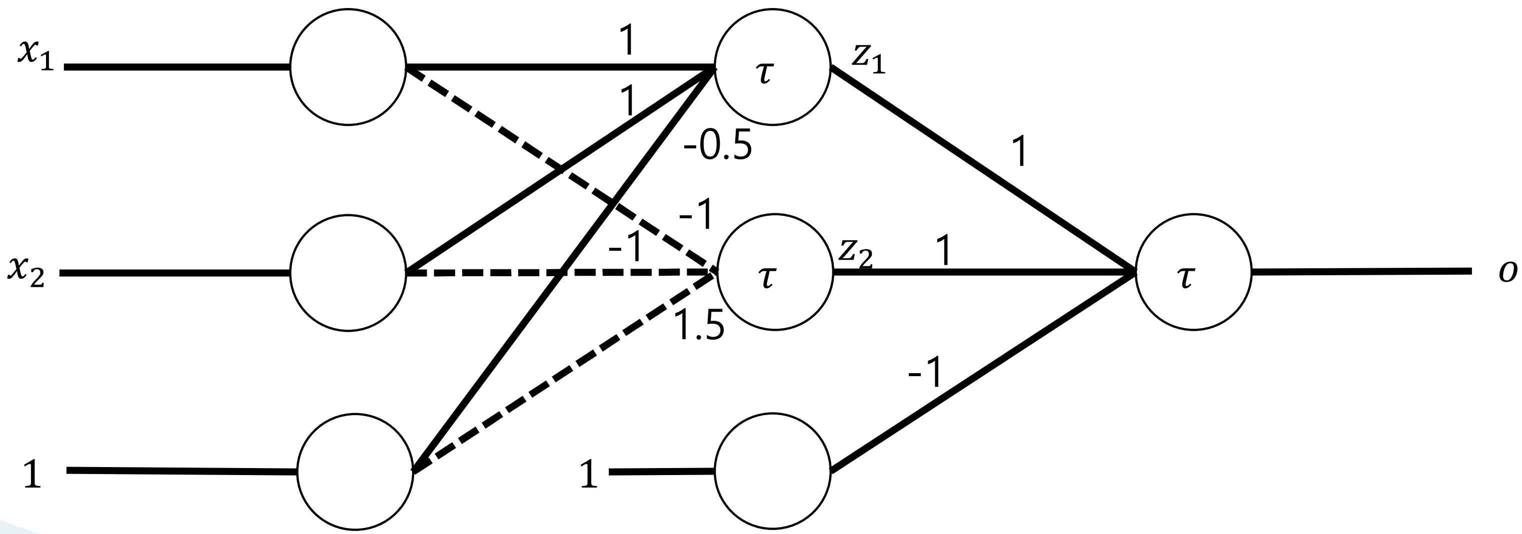
- 다음과 같은 두개의 직선을 사용한다면 분류 가능



AIoT

다층 퍼셉트론(Multi Layered Perceptron)

- 두개의 직선은 두개의 퍼셉트론(z_1, z_2)로 표현가능
  z를 은닉층이라 함



다층 퍼셉트론

- 은닉층의 노드를 각각 z_1, z_2라고 하자,
  두 노드의 출력을 구하는 식은 다음과 같다.

$$z_1 = \tau(x_1 + x_2 - 0.5)$$
$$z_2 = \tau(-x_1 - x_2 + 1.5)$$

- 두 은닉층 노드의 출력을 이용하여 최종 출력은
  다음과 같이 구할 수 있다.

$$o = \tau(z_1 + z_2 - 1)$$

다층 퍼셉트론

- 이제 XOR 분류를 제대로 수행할 수 있는지 확인해보자

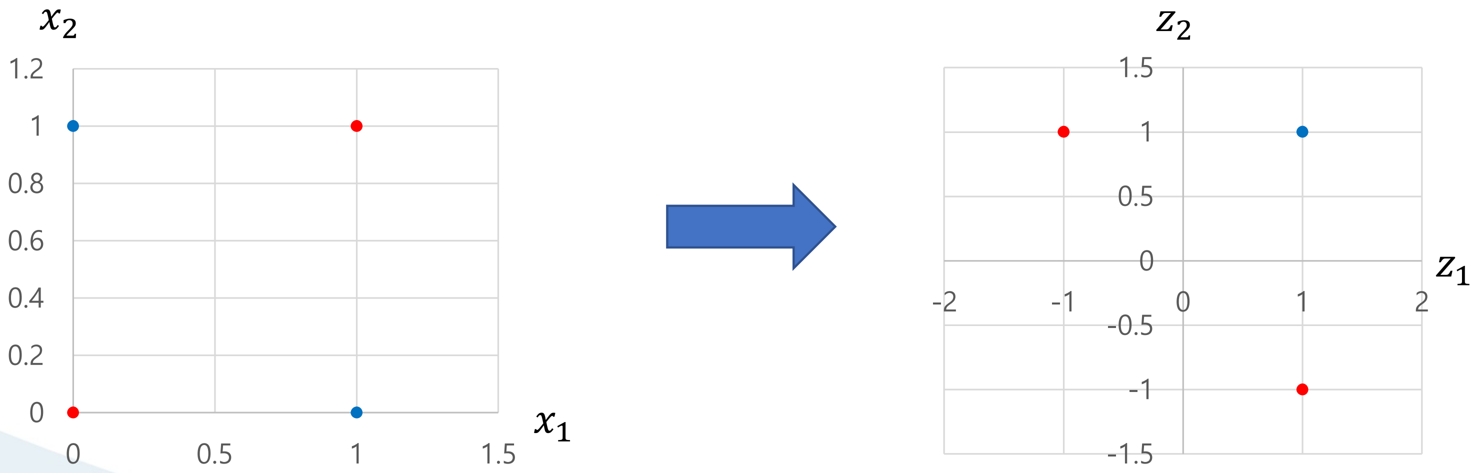
x_1	x_2	z_1	z_2	o
0	0	-1	1	-1
0	1	1	1	1
1	0	1	1	1
1	1	1	-1	-1

$$z_1 = \tau(x_1 + x_2 - 0.5)$$
$$z_2 = \tau(-x_1 - x_2 + 1.5)$$
$$o = \tau(z_1 + z_2 - 1)$$

$$\tau(x) = \begin{cases} 1 \ (x \geq 0) \\ -1 \ (x < 0) \end{cases}$$

다층 퍼셉트론

- 어떻게 가능해 졌을까?
  =>은닉층을 거치면서 입력이 변환됨
  우측 그래프는 직선 하나로 분류 가능



다층 퍼셉트론

- 퍼셉트론을 여러 층으로 쌓으면 모델의 표현 능력이 향상됨.
- 이러한 퍼셉트론이 3~4층 이상 쌓이면 딥러닝이라고 부름

다양한 활성 함수

함수 이름	함수	범위	적용
계단 함수	$\tau(x) = \begin{cases} 1 \ (x \geq 0) \\ -1 \ (x < 0) \end{cases}$	1 또는 -1	단층 퍼셉트론
로지스틱 시그모이드	$\tau(x) = \frac{1}{1+e^{-x}}$	(0,1)	다층 퍼셉트론
하이퍼볼릭 탄젠트	$\tau(x) = \tanh(x) = \frac{2}{1+e^{-x}} - 1$	(-1,1)	다층 퍼셉트론
ReLU	$\tau(x) = \max(0, x)$	[0,∞)	딥러닝

AIoT

손실 함수

1장에서,

우리가 한 과정은 **테스트 데이터**를 가장 잘 표현하는 **모델**의 **매개 변수**를 **학습 데이터**를 이용하여 찾아낸 것이다.

1장에서 다뤘던 예시들은 무작정 매개 변수를 대입하다 보면 오차가 0인 최적의 해를 금방 찾을 수 있었음.

하지만 실제 세계의 데이터는 오차가 반드시 존재.

이러한 경우에도 100% 정답은 아니지만 정답에 가까운 최적의 매개변수를 찾아야 하는 방법이 필요함

손실 함수

- 손실 함수(Loss Function): 예측 값(o)과 실제 값(y)사이에 오차(차이)가 얼마나 발생 했는 지 수치적으로 표현하는 함수

- 손실 함수의 값이 작을 수록 예측 값과 실제 값 사이의 오차가 적다고 해석할 수 있음.

손실 함수

- 손실 함수를 사용하여 1장에서 언급한 머신러닝(딥러닝)에서 해야할 일을 다시 정리하자면

테스트 데이터와의 **손실 함수 값(오차)**이 **가장 작게** 나오는 **모델**의 **매개 변수**를 **학습 데이터**를 이용하여 찾아내는 것

손실 함수

- 매개 변수를 Θ, 손실 함수를 J라고 했을 때
  최적의 매개변수 $\hat{\Theta}$는 다음과 같다

$$\hat{\Theta} = \underset{\Theta}{argmin}\, J(\Theta)$$

$\underset{\Theta}{argmin}\, J(\Theta)$ 이 의미하는 것은 J(Θ)를 가장 작게 하는 Θ이다.

AIoT

여러 손실 함수

4장 Regression에서 주로 다룰 예정

- Mean Absolute Error(MAE)

$$J(\Theta) = \frac{1}{n}\sum_{i=1}^{n}|o_i - y_i|$$

- Mean Squared Error(MSE)

$$J(\Theta) = \frac{1}{n}\sum_{i=1}^{n}(o_i - y_i)^2$$

여러 손실 함수

- 5장 Classification에서 주로 다룰 예정
- Binary Cross Entropy(BCE): 2개의 클래스에 대한 분류문제에 사용

$$J(\Theta) = -\frac{1}{n}\sum_{i=1}^{n}\{y_i \log(\sigma(o_i)) + (1 - y_i)\log(1 - \sigma(o_i))\}$$

- Sigmoid(σ)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

AIoT

여러 손실 함수

- Cross Entropy(CE)
  k개 클래스에 대한 예측 문제라면, o는 n x k 크기의 형태

$$J(\Theta) = -\frac{1}{n}\sum_{i=1}^{n} \log(softmax(o_i[y_i]))$$

y_i는 정답 클래스의 인덱스를 나타냄

- Softmax

$$softmax(x[i]) = \frac{e^{x[i]}}{\sum_j^k e^{x[j]}}$$

AIoT

최적화

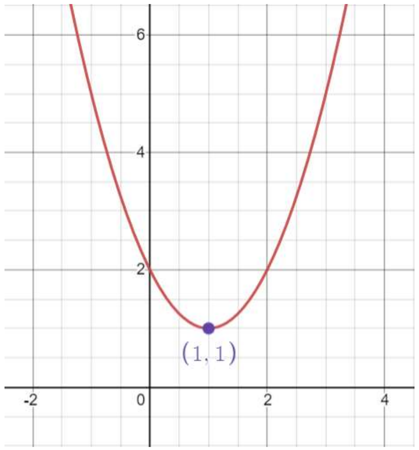
- 딥러닝에서 최적화란 모델의 매개 변수를 찾아가는 과정
- 최적의 매개변수란, 손실 함수의 값을 최소로 하는 매개변수
- 1장에서는 무작정 대입하여 최적의 매개변수를 찾아냈지만, 실제 학습 과정에서는 미분을 이용한 수학적 방법 이용

AIoT

최적화

- 다음과 같은 2차 함수 개형의 손실 함수 $J(\Theta)$가 있다고 하자.
  다음 손실 함수의 최소 값은 $\Theta = 1$일 때, 1이다.

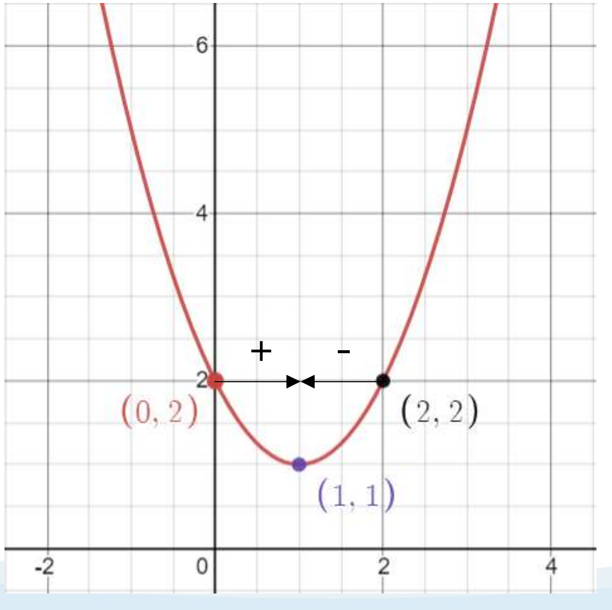
$$J(\Theta) = (\Theta - 1)^2 + 1 = \Theta^2 - 2\Theta + 2$$



AIoT

최적화

- 만약에 $\Theta = 2$라면, $\Theta = 1$을 향해 음수를 더해줘야 하고 $\Theta = 0$이라면 $\Theta = 1$을 향해 양수를 더해주는 방향으로 매개변수의 갱신이 필요하다.



AIoT

최적화

- 일반화 하자면

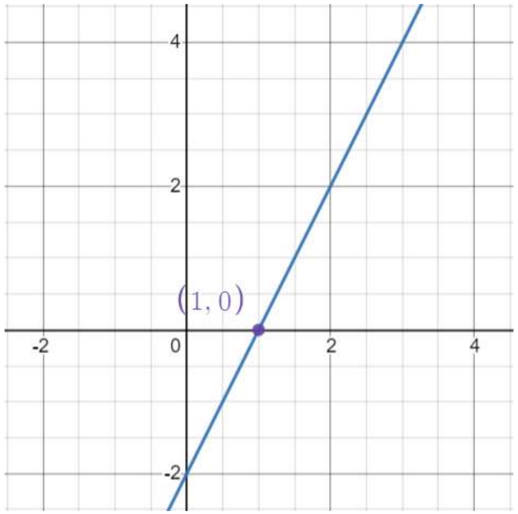
매개 변수	최적화 방향
$\Theta < 1$	양의 방향
$\Theta = 1$	최적의 매개변수
$\Theta > 1$	음의 방향

최적화

- 이번에는 손실 함수 $J(\Theta)$를 Θ에 대해 미분해보자.

$$J'(\Theta) = 2\Theta - 2$$

해당 도함수는 $\Theta = 1$을 기준으로 왼쪽은 음수이고,
오른쪽은 양수이다.



최적화

- 도함수의 값(Gradient)에 음을 취해 기존 매개 변수에 더해주면 최적의 매개 변수 방향으로 매개 변수를 업데이트 할 수 있다.
  =**경사하강법**(Gradient Descent)

$$\Theta_{new} = \Theta_{old} - J'(\Theta_{old})$$

최적화

- 그러나 $\Theta = 0$의 경우와 $\Theta = 2$의 경우를 보면

$$\Theta_{new} = 0 - J'(0) = 0 - (-2) = 2,$$
$$\Theta_{new} = 2 - J'(2) = 2 - (2) = 0$$

으로 최적의 매개 변수 $\Theta = 1$을 지나쳐 버린다.

- 그래서 매개 변수의 갱신은 도함수의 값을 그대로 빼 주는 것이 아닌 (0,1)의 실수를 곱하여 갱신해 준다.
  이를 학습률(Learning rate)이라고 하고 주로 ρ로 표기한다.

최적화

- 따라서, 경사 하강법은 다음과 같다.

$$\Theta_{new} = \Theta_{old} - \rho J'(\Theta_{old})$$

보통은 도함수를 다음과 같이 표기하기도 함

$$\Theta_{new} = \Theta_{old} - \rho \nabla J \Big|_{\Theta_{old}}$$

AIoT

Review

- 스칼라, 벡터, 행렬, 텐서
  데이터의 차원에 따른 분류, RGB 이미지는 텐서로 표현 가능
- 퍼셉트론
  딥러닝의 가장 기본적인 구조
- 손실 함수
  문제(회귀, 분류 등)에 따라 여러 손실 함수 존재
- 최적화 - 경사 하강법
  손실 함수를 미분하여 최적의 매개 변수를 찾음

AIoT

3장 Preview

- Python의 딥러닝 프레임워크 Pytorch에서 텐서 다뤄보기
- Pytorch에서 제공하는 다양한 함수와 클래스 살펴보기
- 1장 딥러닝의 간단한 예시를 Pytorch에서 적용해보기

AIoT

수고하셨습니다!

AIoT