Week 4(1/3)

Perceptron

Machine Learning with Python

Handong Global University
Prof. Youngsup Kim
idebtor@gmail.com

Perceptron

- Goals
  - Understanding Perceptron

- Content
  - Perceptron Overview
  - Perceptron Binary Classification
  - Perceptron Learning
  - Overfitting and Underfitting

1. Perceptron History

Frank Rosenblatt(출처: Arvin Calspan Advanced Technology Center; Hecht-Nielsen, R. Neurocomputing)

1. Perceptron History

- Artificial Neuron → Neuron, Node, Perceptron
- Perceptron → The First Artificial Neural Network
  - Frank Rosenblatt, 1957
  - Cornell Aeronautical Laboratory

Frank Rosenblatt(출처: Arvin Calspan Advanced Technology Center; Hecht-Nielsen, R. Neurocomputing)

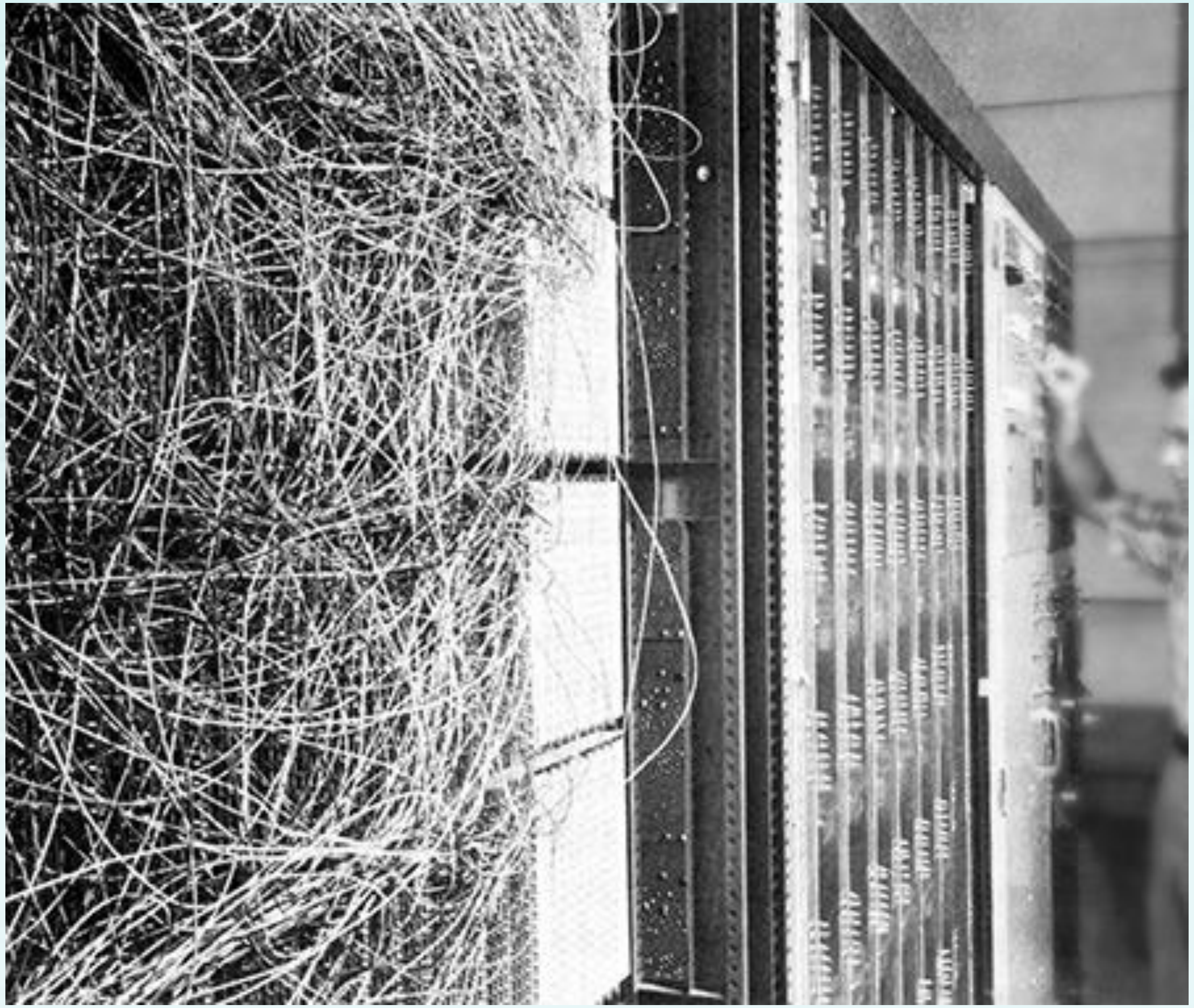
1. Perceptron History

- Artificial Neuron → Neuron, Node, Perceptron
- Perceptron → The First Artificial Neural Network
  - Frank Rosenblatt, 1957
  - Cornell Aeronautical Laboratory
  - The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain

Frank Rosenblatt(출처: Arvin Calspan Advanced Technology Center; Hecht-Nielsen, R. Neurocomputing)

1. Perceptron History

- **Artificial Neuron → Neuron, Node, Perceptron**
- **Perceptron → The First Artificial Neural Network**
  - **Frank Rosenblatt, 1957**
  - **Cornell Aeronautical Laboratory**
  - **The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain**
  -



마크 1 퍼셉트론 (출처: Arvin Calspan Advanced Technology Center; Hecht-Nielsen, R. Neurocomputing)

1. Perceptron History

ARCHIVES | 1958

NEW NAVY DEVICE LEARNS BY DOING; Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

JULY 8, 1958

1958

WASHINGTON, July 7 (UPI) -- The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

1. Perceptron History

1. Perceptron History

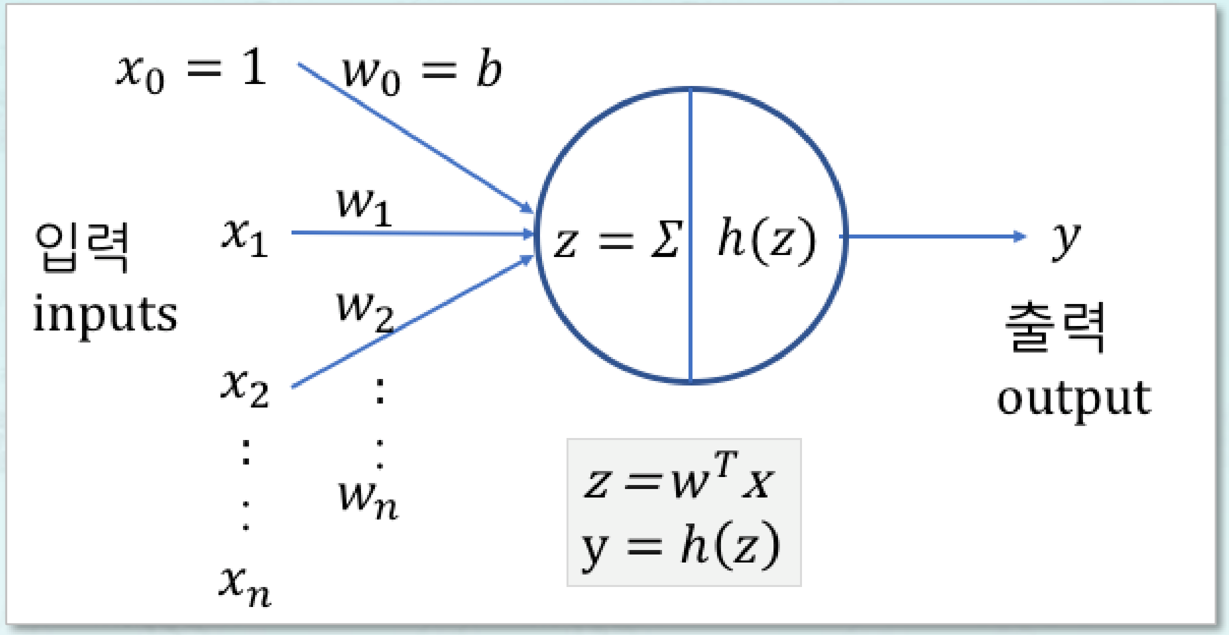
ARCHIVES | 1958

NEW NAVY DEVICE LEARNS BY DOING; Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

JULY 8, 1958

WASHINGTON, July 7 (UPI) -- The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

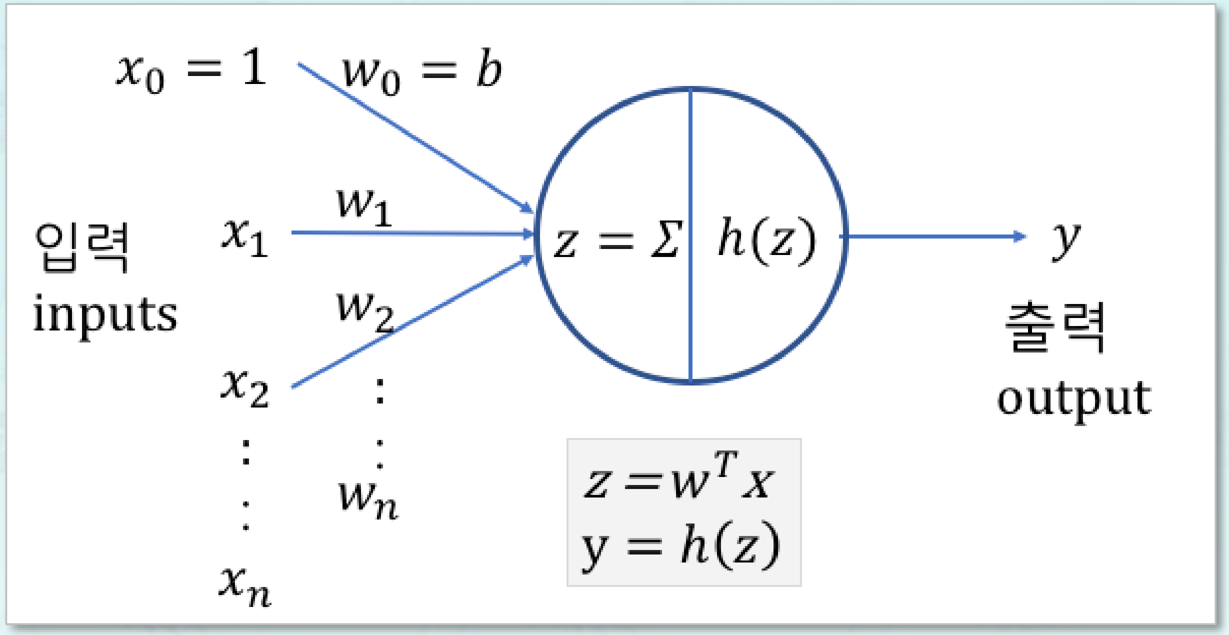
2. Perceptron Structure



2. Perceptron Structure

- **input x**

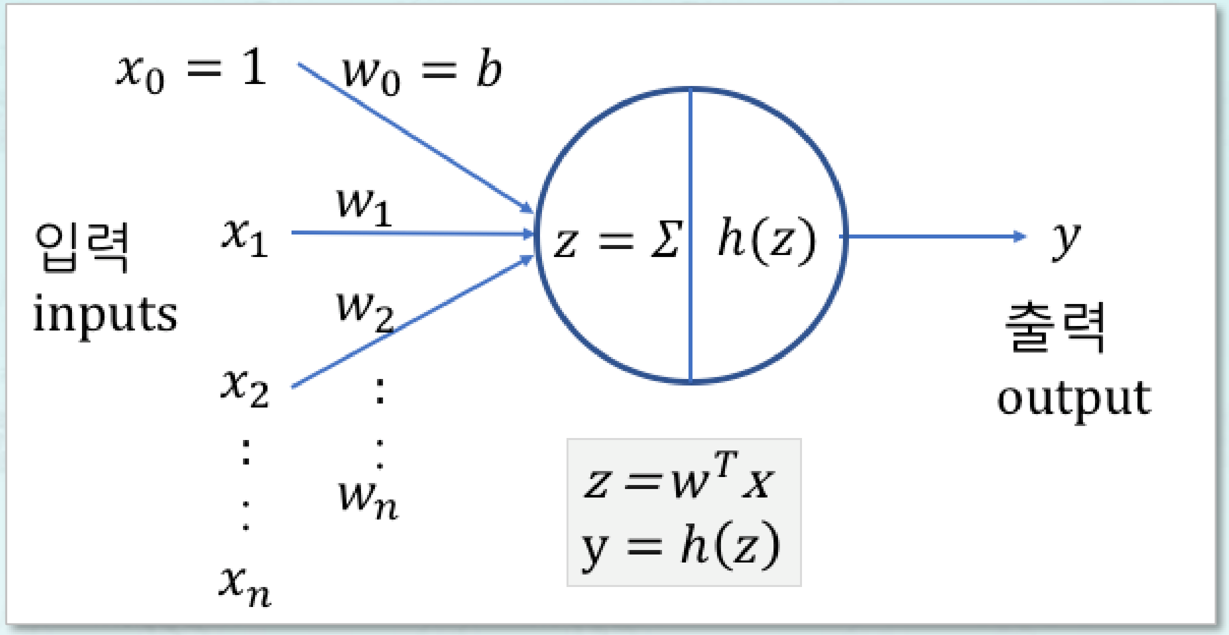
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



2. Perceptron Structure

- input x

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



- weight w

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

2. Perceptron Structure

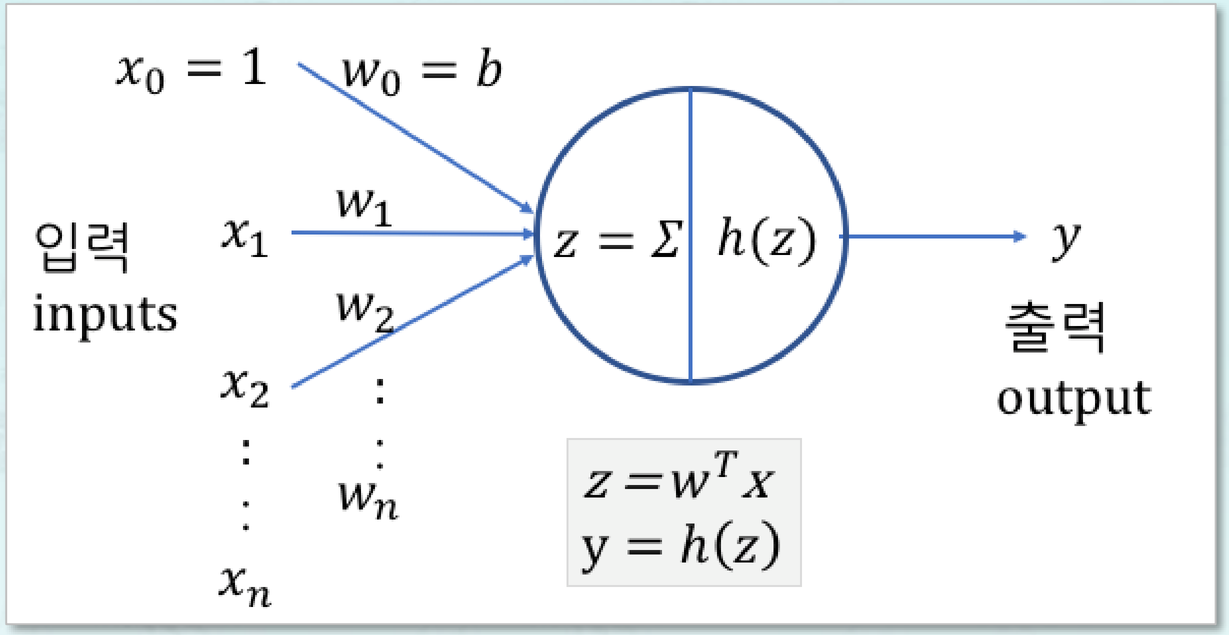
- input x

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

bias
b

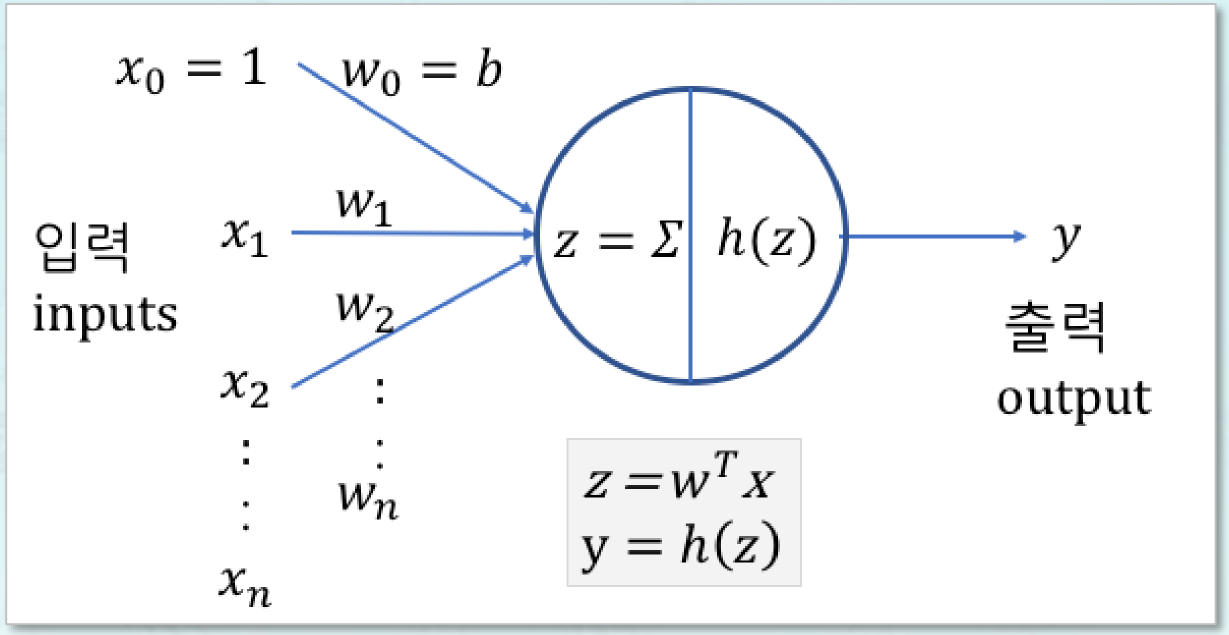
- weight w

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$



2. Perceptron Structure: net input

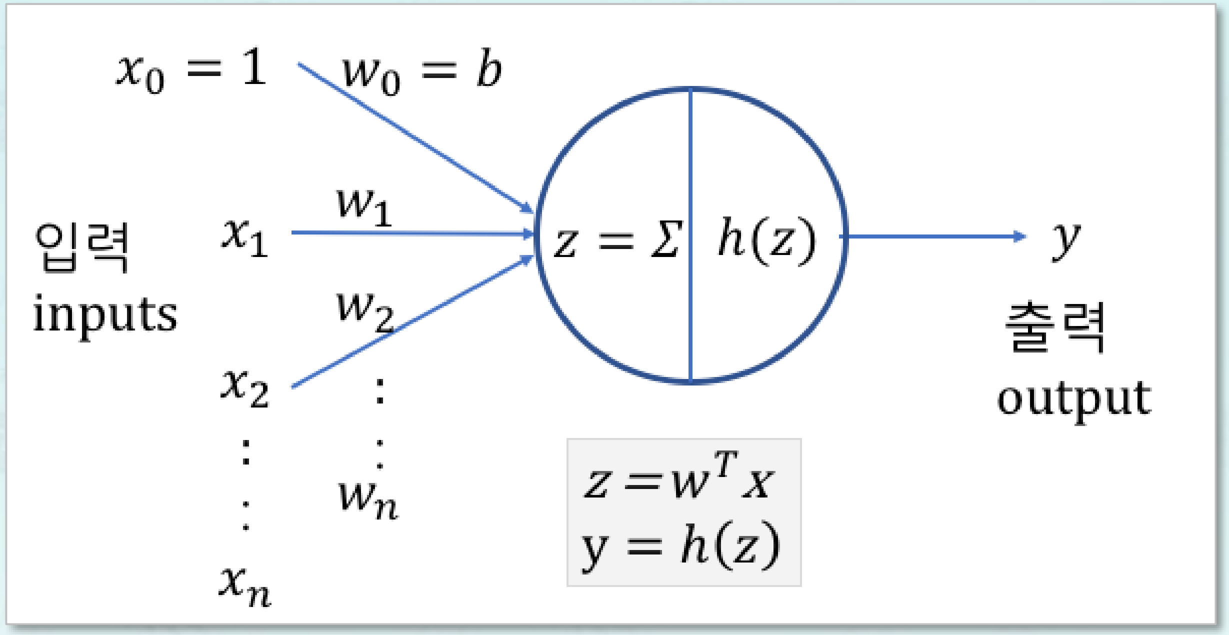
- **net input z**



2. Perceptron Structure: net input

- **net input z**

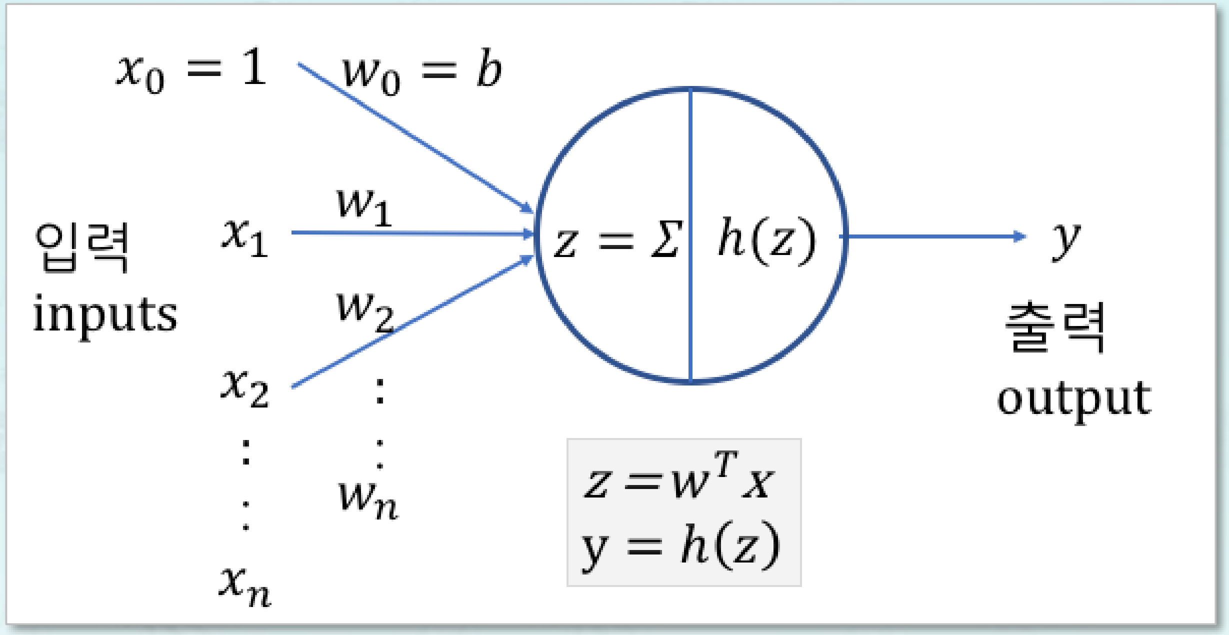
$$z = w_0 x_0 + w_1 w_1 + \ \ldots + w_n x_n$$



2. Perceptron Structure: net input

- **net input z**

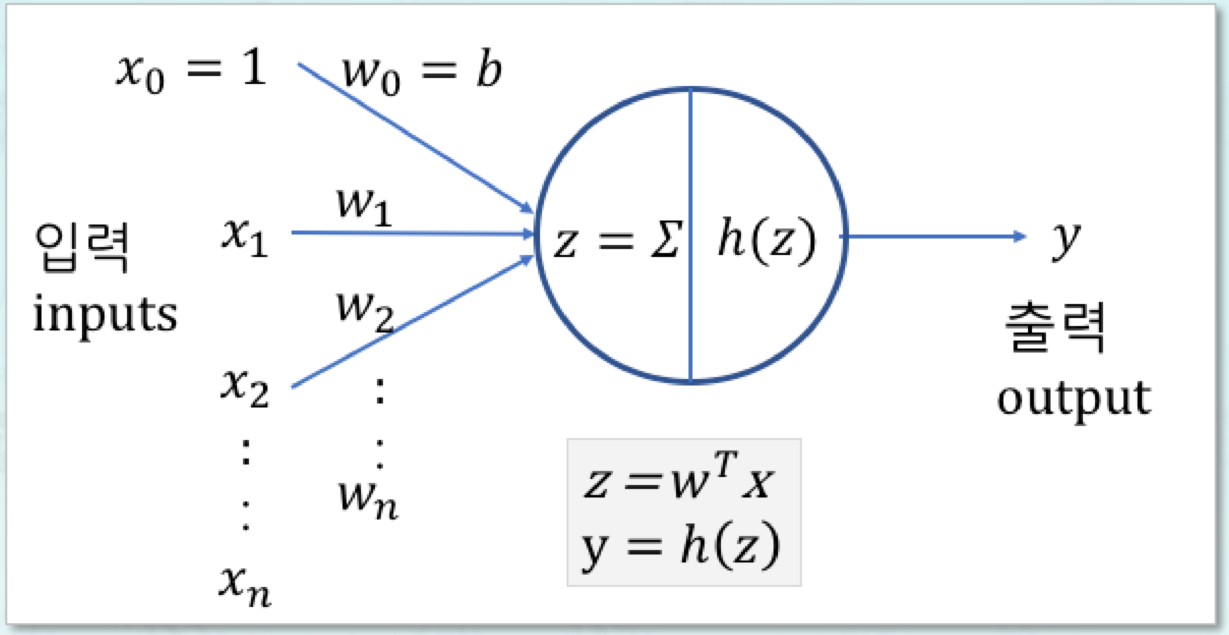
$$z = w_0x_0 + w_1w_1 + \ \dots + w_nx_n$$
$$= \sum_{j=0}^{n} x_jw_j$$



2. Perceptron Structure: net input

- **net input z**

$$z = w_0 x_0 + w_1 w_1 + \ \ldots + w_n x_n$$
$$= \sum_{j=0}^{n} x_j w_j$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$



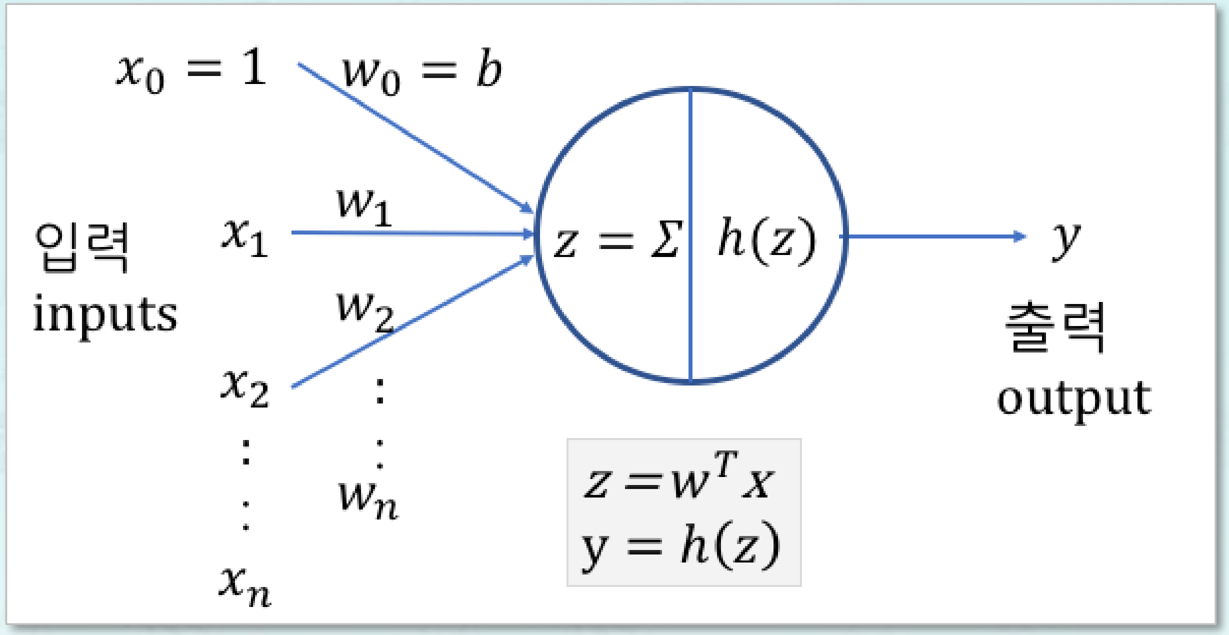
2. Perceptron Structure: net input

- **net input z**

$$z = w_0 x_0 + w_1 w_1 + \ \ldots + w_n x_n$$

$$= \sum_{j=0}^{n} x_j w_j$$

$$= \mathbf{w}^{\mathbf{T}} \mathbf{x}$$



$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

2. Perceptron Structure: net input computation

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.

2. Perceptron Structure: net input computation

```
x = np.array([0, 1, 2, 3])
w = np.array([0, 0.1, 0.2, 0.3])
z = np.dot(x, w)
print(z)
```

```
1.4
```

Solution(1)

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.

2. Perceptron Structure: net input computation

```
x = np.array([0, 1, 2, 3])
w = np.array([0, 0.1, 0.2, 0.3])
z = np.dot(x, w)
print(z)
```

```
1.4
```

Solution(1)

```
import numpy as np
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w, x)
print(z)
```

```
1.329056653793057
```

Solution(2)

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.

2. Perceptron Structure: net input computation

- net input z

$$z = w_0x_0 + w_1w_1 + \ ... + w_nx_n$$
$$= \sum_{j=0}^{n} x_jw_j$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- Thoughts on Solution(2):

```
import numpy as np
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w, x)
print(z)
```

```
1.329056653793057
```

Solution(2)

2. Perceptron Structure: net input computation

- **net input z**

$$z = w_0 x_0 + w_1 w_1 + \ \dots + w_n x_n$$
$$= \sum_{j=0}^{n} x_j w_j$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

```python
import numpy as np
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w, x)
print(z)
```

```
1.329056653793057
```

Solution(2)

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- Thoughts on Solution(2):

2. Perceptron Structure: net input computation

- **net input z**

$$z = w_0 x_0 + w_1 w_1 + \ \ldots + w_n x_n$$
$$= \sum_{j=0}^{n} x_j w_j$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- **Thoughts on Solution(2):**

```python
import numpy as np
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w, x)
print(z)
```

```
1.329056653793057
```

Solution(2)

```python
import numpy as np
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w.T, x)
print(z)
```

```
1.4781818847304011
```

Solution(3)

2. Perceptron Structure: net input computation

- **net input z**

$$z = w_0 x_0 + w_1 w_1 + \ \ldots + w_n x_n$$
$$= \sum_{j=0}^{n} x_j w_j$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- **Thoughts on Solution(2):**

```
import numpy as np
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w, x)
print(z)
```

```
1.329056653793057
```

Solution(2)

```
import numpy as np
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w.T, x)
print(z)
```

```
1.4781818847304011
```

Solution(3)

2. Perceptron Structure: net input computation

- **net input z**

$$z = w_0x_0 + w_1w_1 + \ \dots + w_nx_n$$
$$= \sum_{j=0}^{n} x_j w_j$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- **Thoughts on Solution(2):**

```python
import numpy as np
np.random.seed(0)
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w, x)
print(z)
```

```
3.553656675063983
```

Solution(2A)

```python
import numpy as np
np.random.seed(0)
x = np.array(np.arange(4))
w = np.array(np.random.random(4))
z = np.dot(w.T, x)
print(z)
```

```
3.553656675063983
```

Solution(3A)

2. Perceptron Structure: net input computation

- net input z

$$z = w_0 x_0 + w_1 w_1 + \ \ldots + w_n x_n$$
$$= \sum_{j=0}^{n} x_j w_j$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

```
print('x.shape={}, w.shape{}, w.T.shape{}'.
      format(x.shape, w.shape, w.T.shape))
```

```
x.shape=(4,), w.shape(4,), w.T.shape(4,)
```

- :
- :
-

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- Thoughts on Solution(2):

2. Perceptron Structure: net input computation

- net input z

$$z = w_0x_0 + w_1w_1 + \ \dots + w_nx_n$$
$$= \sum_{j=0}^{n} x_jw_j$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

```
print('x.shape={}, w.shape{}, w.T.shape{}'.
      format(x.shape, w.shape, w.T.shape))
```

```
x.shape=(4,), w.shape(4,), w.T.shape(4,)
```

- .
- .
-

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- Thoughts on Solution(2):

2. Perceptron Structure: net input computation

- input x, w:
  - row vector(행 벡터)
  - shape n x 1 or (n, 1)

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

- **net input z example:**
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- Thoughts on Solution(2):

2. Perceptron Structure: net input computation

- input x, w:
  - row vector(행 벡터)
  - shape n x 1 or (n, 1)

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

- net input z example:
  1. input x = [0, 1, 2, 3]
  2. weight w = [0..1]
  3. Compute net input z.
- Thoughts on Solution(2):

```
np.random.seed(0)
x = np.array(np.arange(4)).reshape(4,1)
w = np.array(np.random.random(4)).reshape(4,1)
z = np.dot(w.T, x)    #.random((4,1)) is OK!
print(z)
```

```
[[3.55536567]]
```

2. Perceptron Structure: net input computation

```
print(x)
print(w.T)
print(z)
print('shapes: x{}, w{}, w.T{}, z{}'.
  format(x.shape, w.shape, w.T.shape, z.shape))
```

```
[[0]
 [1]
 [2]
 [3]]
[[0.5488135  0.71518937 0.60276338 0.54488318]]
[[3.55536567]]
shapes: x(4, 1), w(4, 1), w.T(1, 4), z(1, 1)
```

2. Perceptron Structure: net input computation

```
print(x)
print(w.T)
print(z)
print('shapes: x{}, w{}, w.T{}, z{}'.
  format(x.shape, w.shape, w.T.shape, z.shape))
```

```
[[0]
 [1]
 [2]
 [3]]
[[0.5488135  0.71518937 0.60276338 0.54488318]]
[[3.55536567]]
shapes: x(4, 1), w(4, 1), w.T(1, 4), z(1, 1)
```

2. Perceptron Structure: net input computation

```
print(x)
print(w.T)
print(z)
print('shapes: x{}, w{}, w.T{}, z{}'.
  format(x.shape, w.shape, w.T.shape, z.shape))
```

```
[[0]
 [1]
 [2]
 [3]]
[[0.5488135  0.71518937 0.60276338 0.54488318]]
[[3.55536567]]
shapes: x(4, 1), w(4, 1), w.T(1, 4), z(1, 1)
```

2. Perceptron Structure: net input computation

```
print(x)
print(w.T)
print(z)
print('shapes: x{}, w{}, w.T{}, z{}'.
  format(x.shape, w.shape, w.T.shape, z.shape))
```

```
[[0]
 [1]
 [2]
 [3]]
[[0.5488135  0.71518937 0.60276338 0.54488318]]
[[3.55536567]]
shapes: x(4, 1), w(4, 1), w.T(1, 4), z(1, 1)
```

```
z = np.dot(w.T, x).squeeze()
print(z)
```

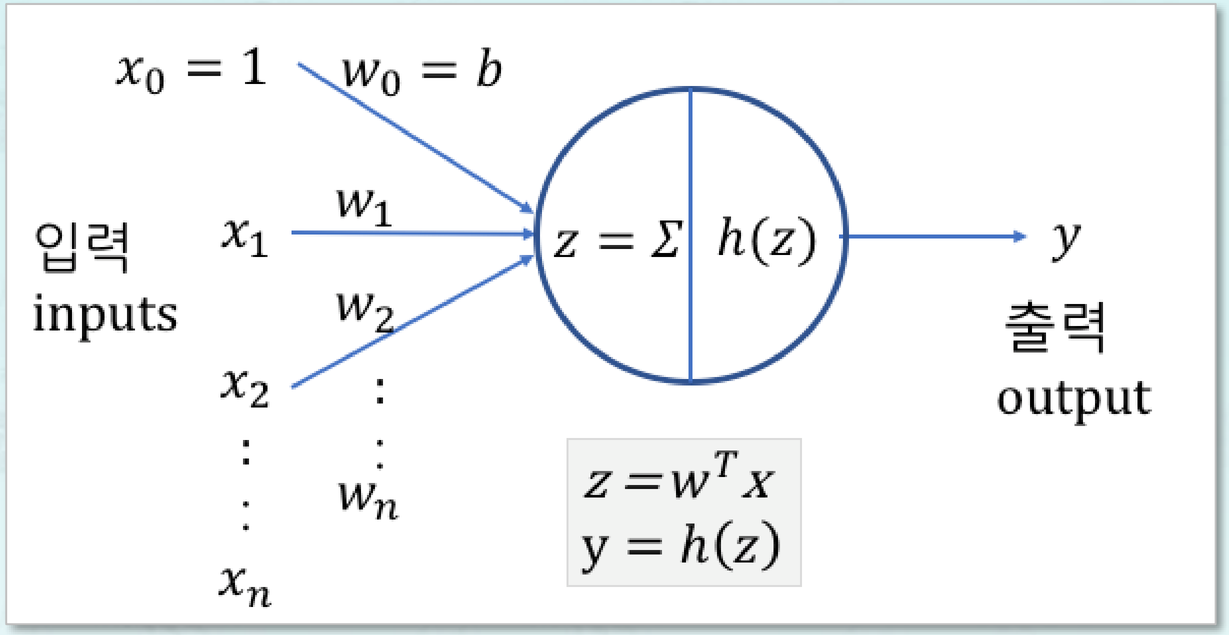
```
3.555365667506398
```

3. Perceptron Binary Classification

- Binary classification
- Linear binary classifier

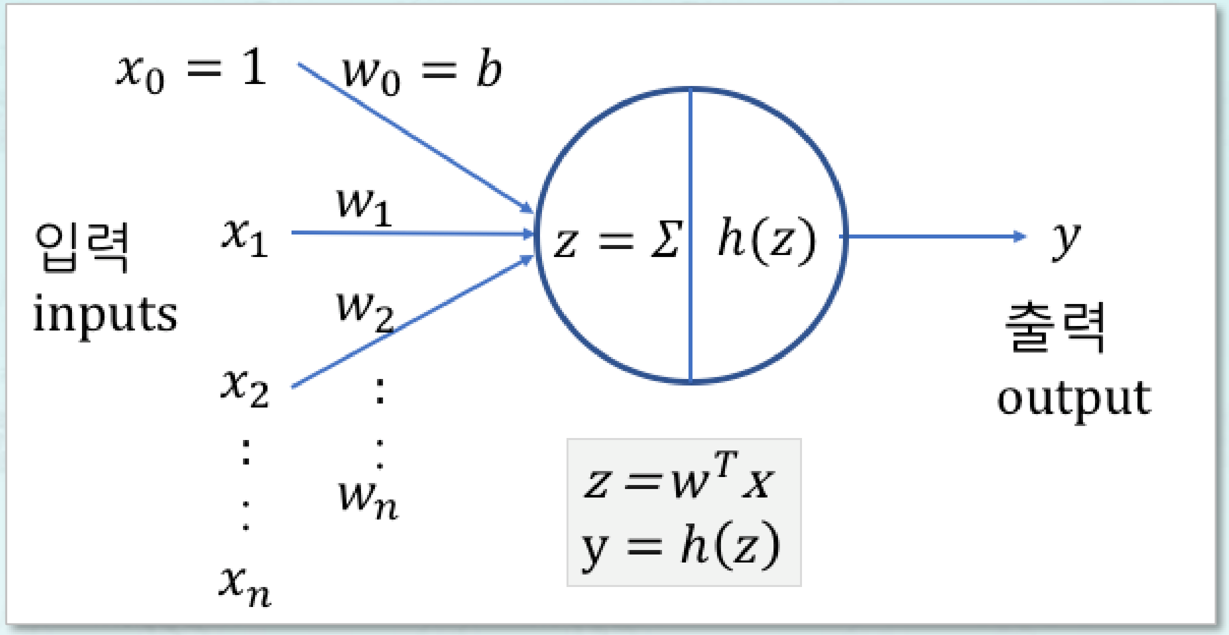
3. Perceptron Binary Classification

- input
- weight
- ?



3. Perceptron Binary Classification

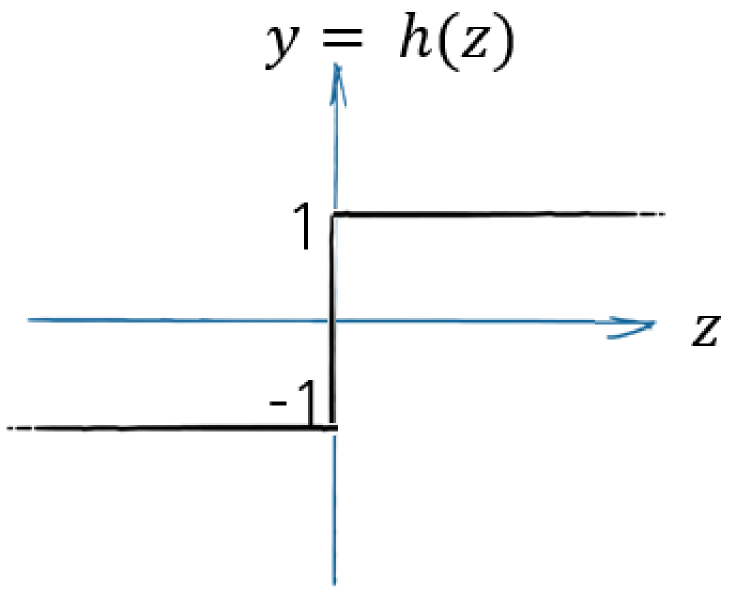
- input
- weight
- Activation Function
  - Sigmoid Function
  - Step Function
  - tanh Function
  - ReLU Function



3. Perceptron Binary Classification

- **Activation Function for Binary Classification**

$$h(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{otherwise.} \end{cases}$$

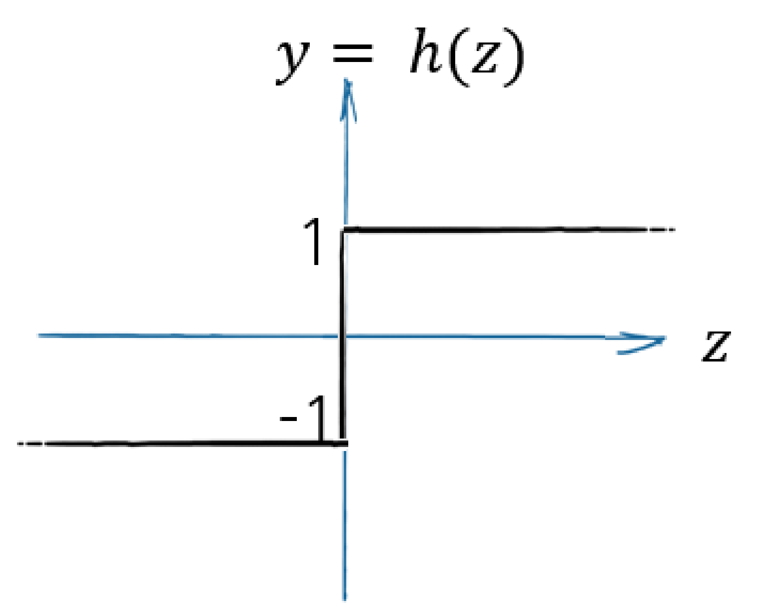


계단함수(양극성)
bipolar step function

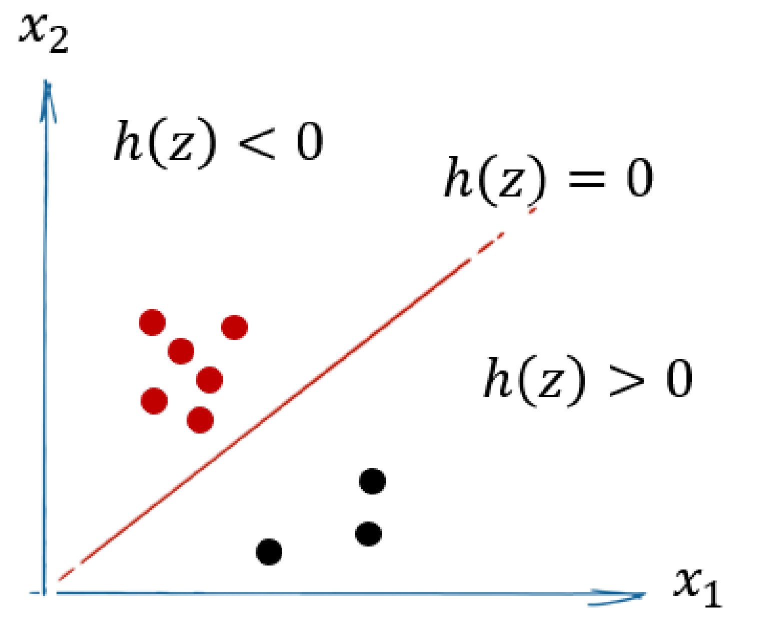
3. Perceptron Binary Classification

- **Activation Function for Binary Classification**

$$h(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{otherwise.} \end{cases}$$



계단함수(양극성)
bipolar step function



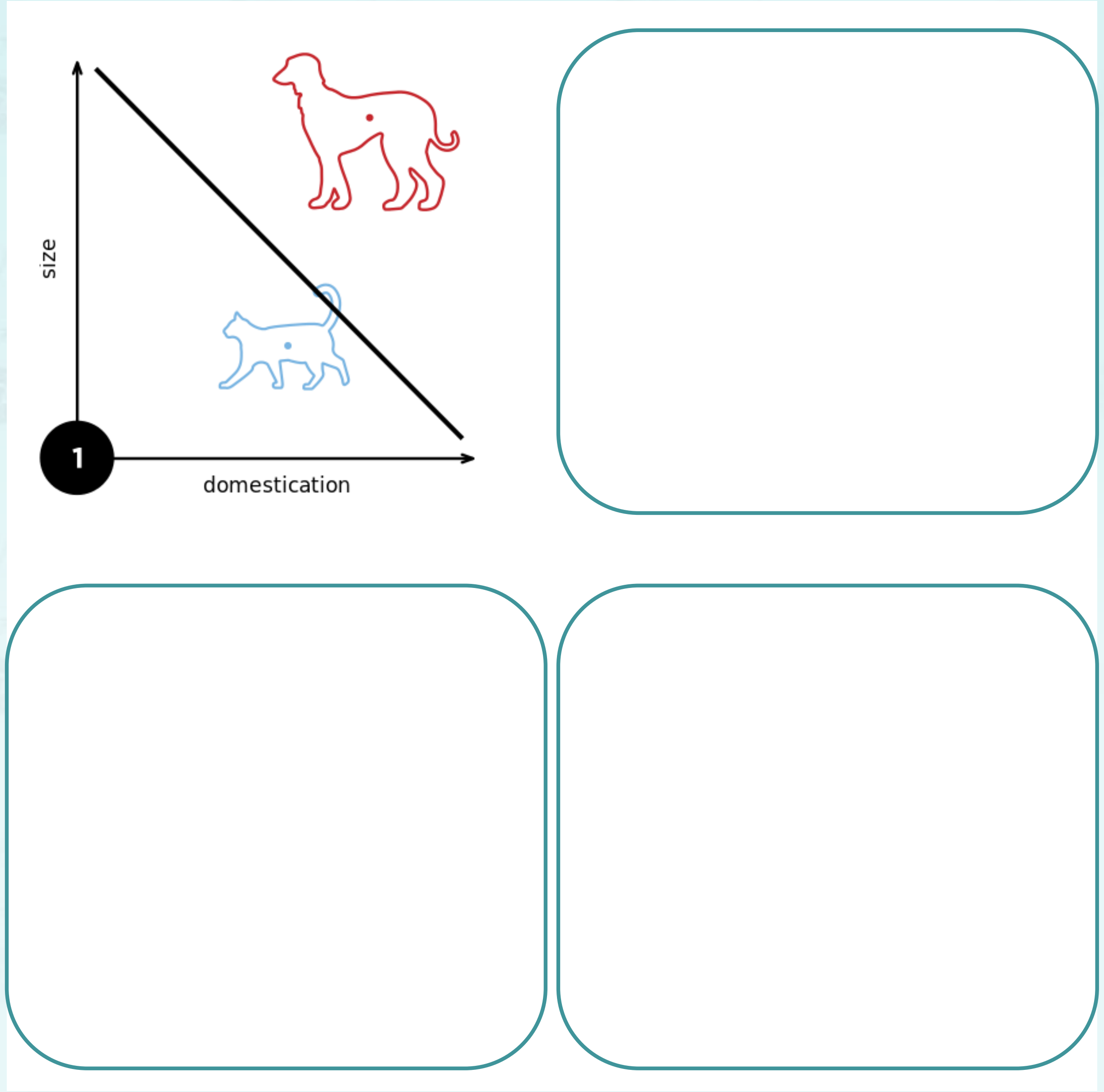
선형 이진 분류기($z = w^T x$)
linear binary classifier

4. Perceptron Learning Method

- **Learning – Change in Weights**

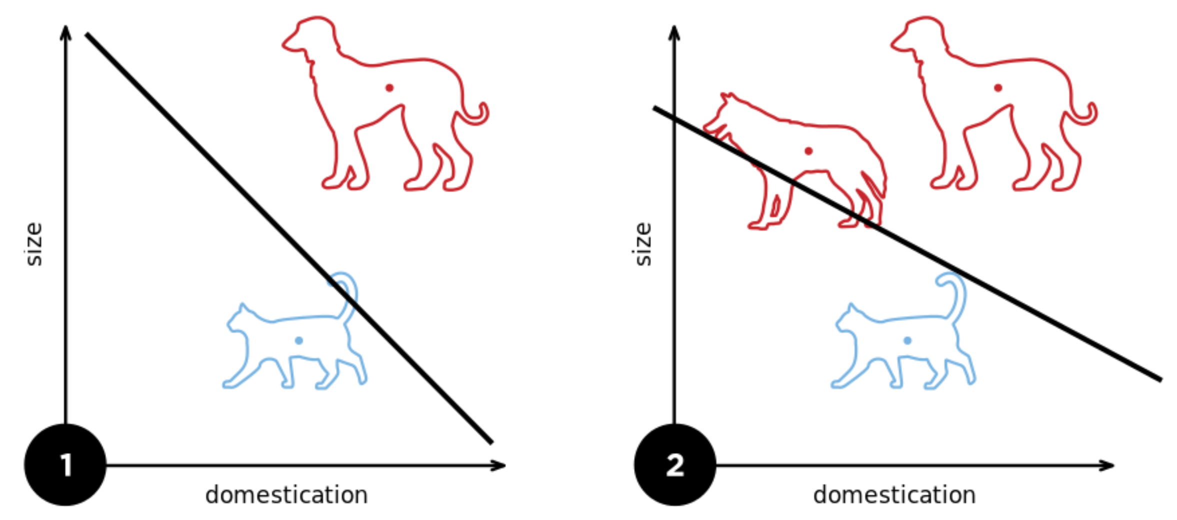
4. Perceptron Learning Method

- **Learning – Change in Weights**



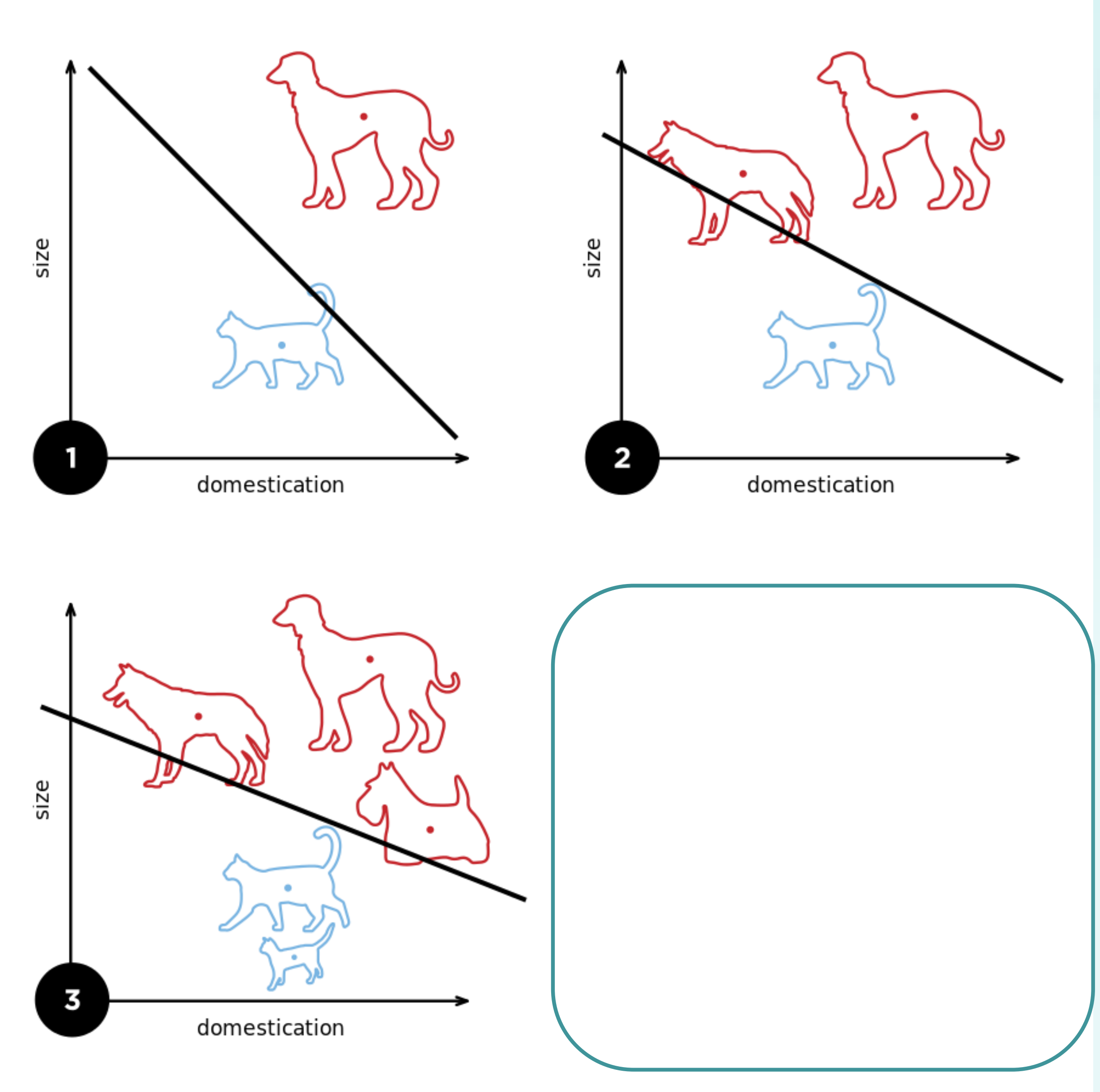
4. Perceptron Learning Method

- **Learning – Change in Weights**



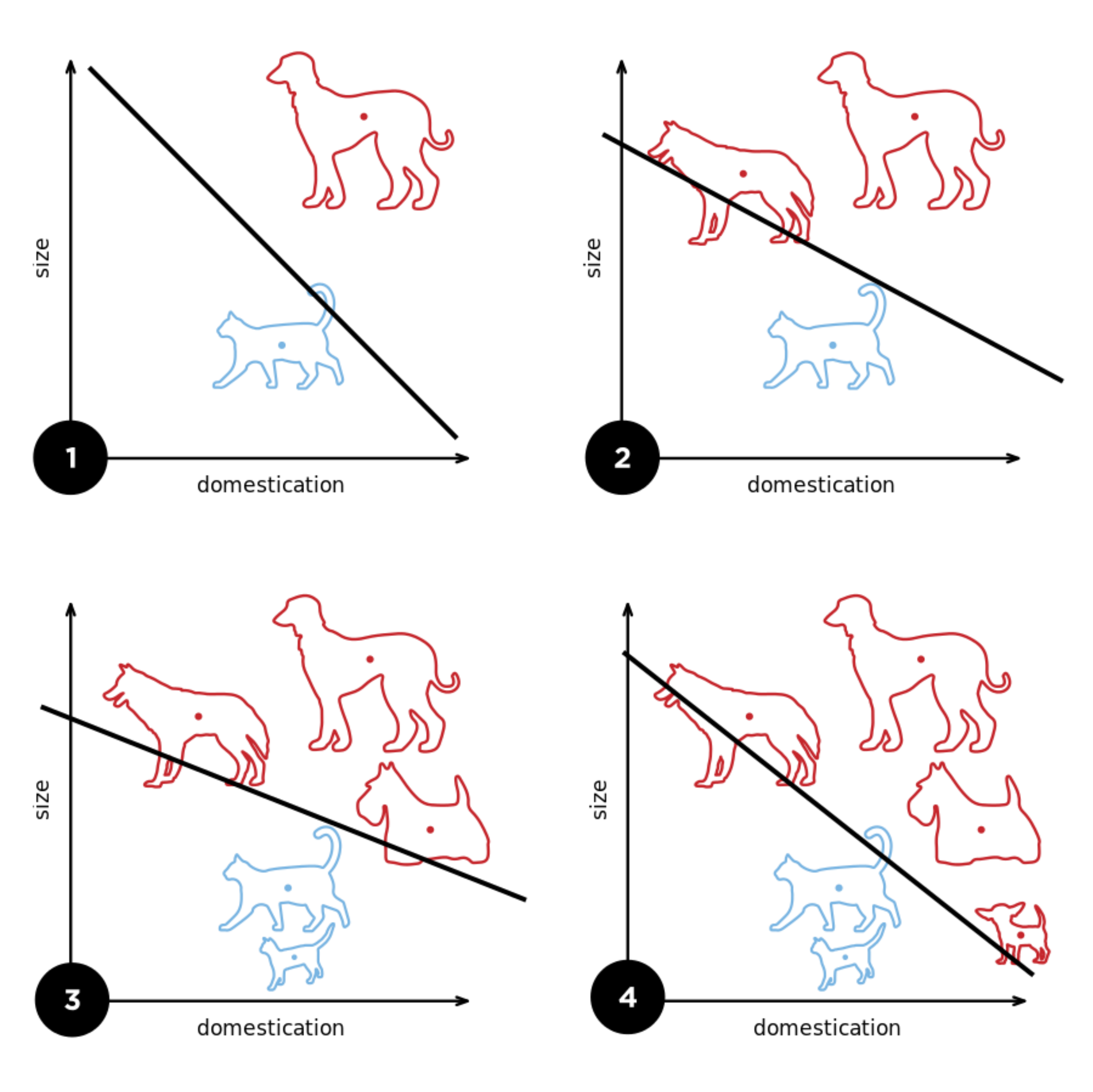
4. Perceptron Learning Method

- **Learning – Change in Weights**



4. Perceptron Learning Method

- **Learning – Change in Weights**



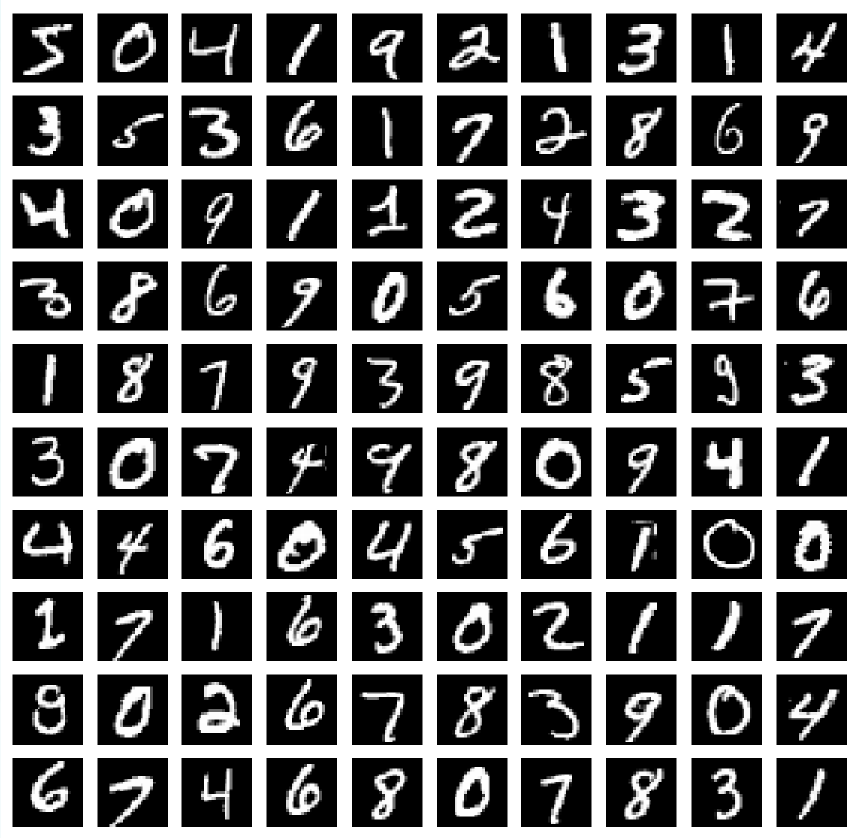
5. Overfitting

- **Perfect Perceptron**?

5. Overfitting

- Perfect Perceptron?

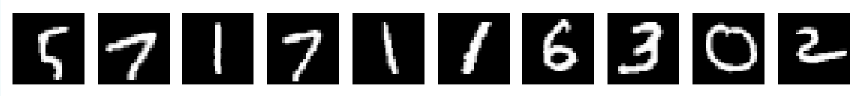
Training Data:



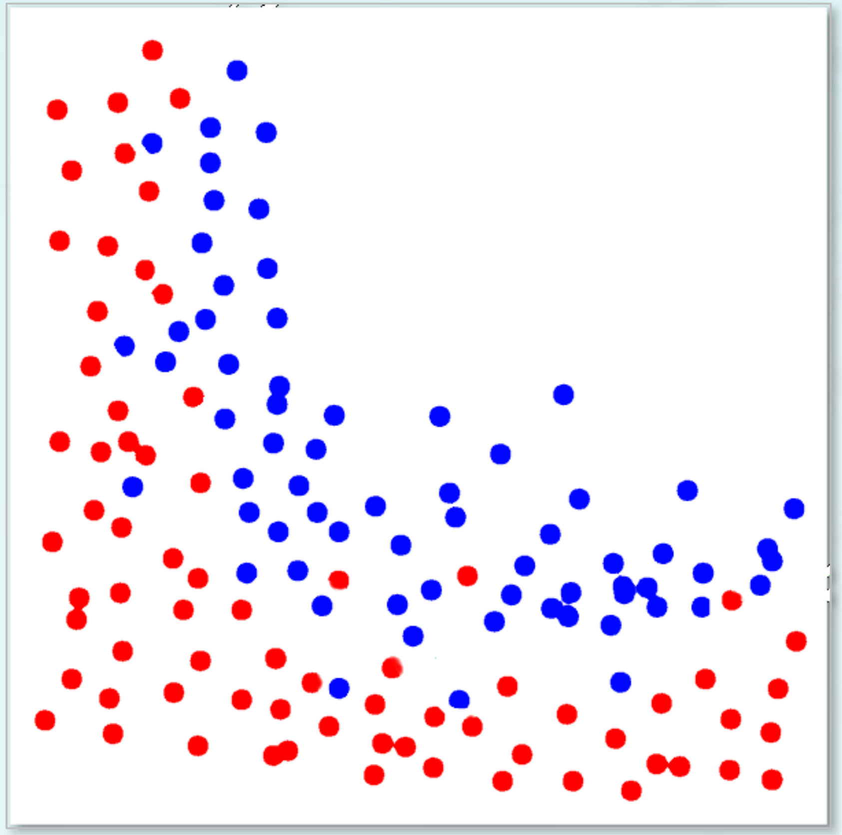
Label:

5 0 4 1 9 2 1 3 1 4
3 5 3 6 1 7 2 8 6 9
4 0 9 1 ...
...

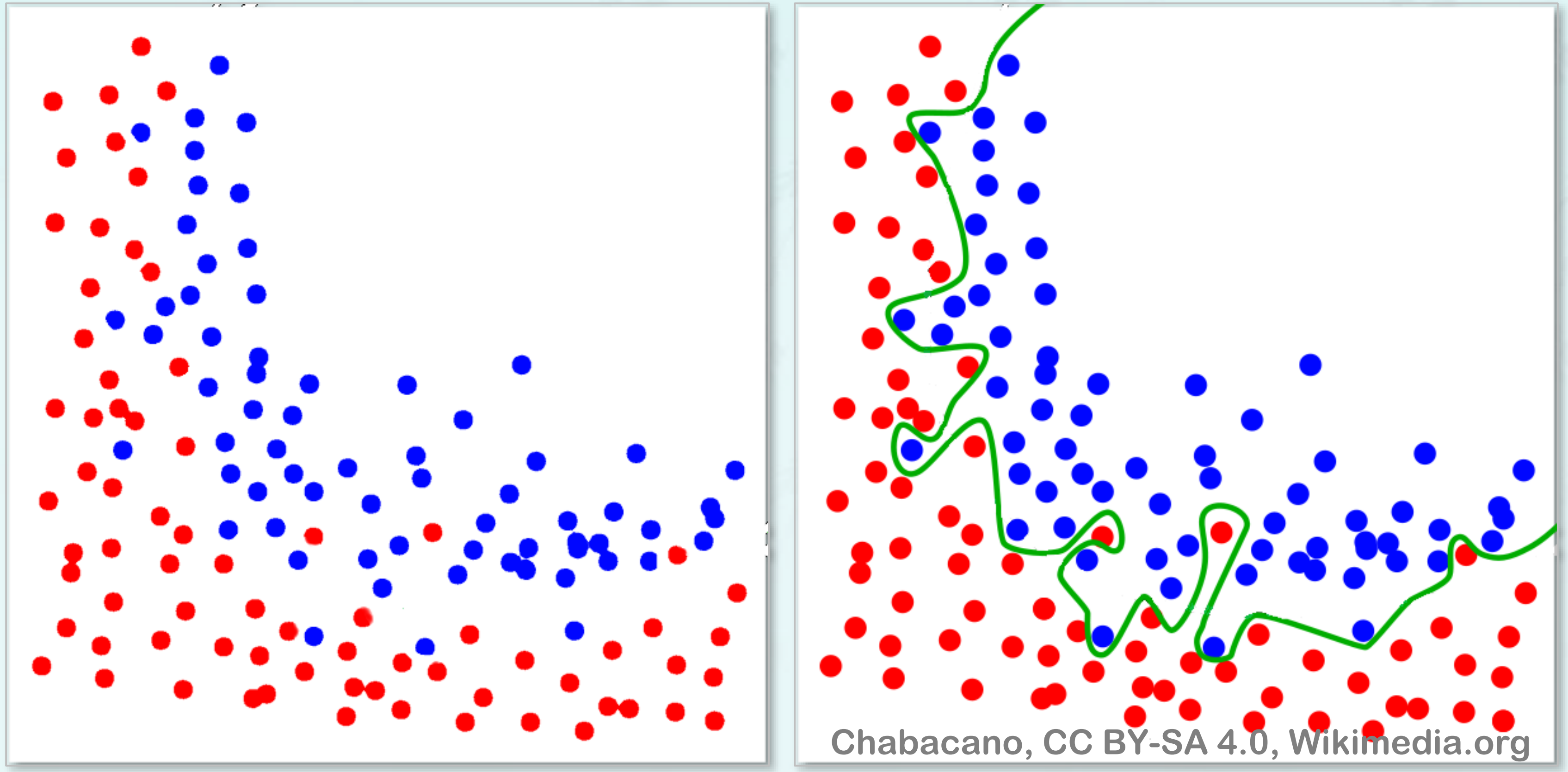
Test Data:



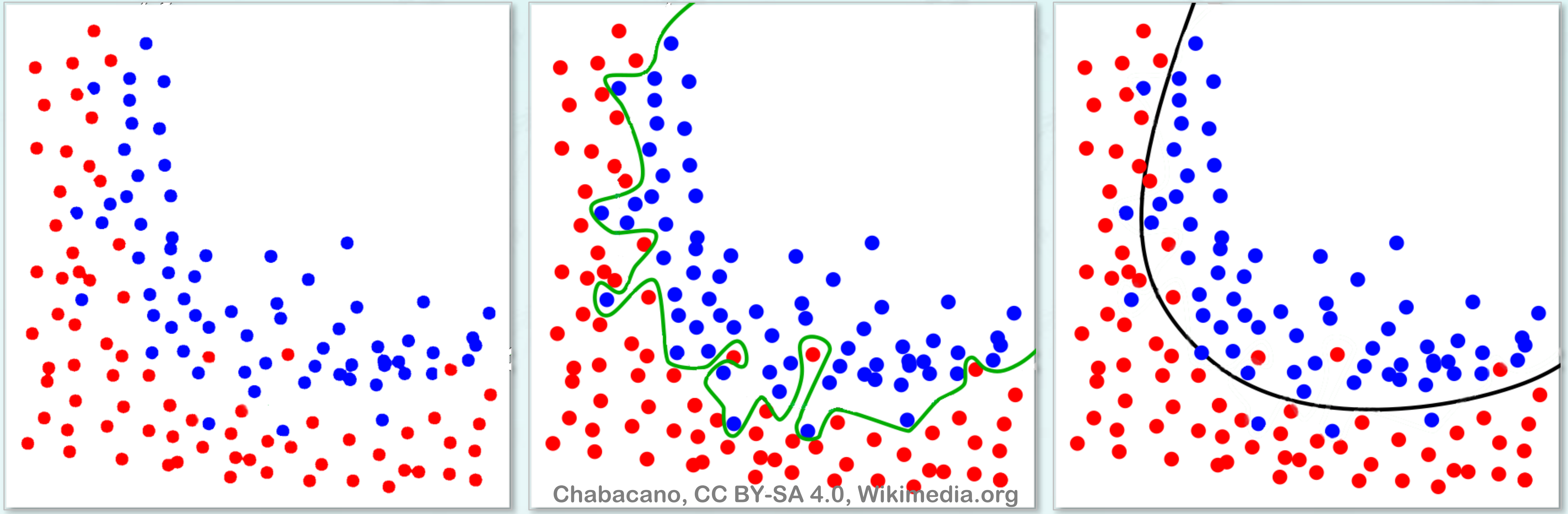
5. Overfitting



5. Overfitting

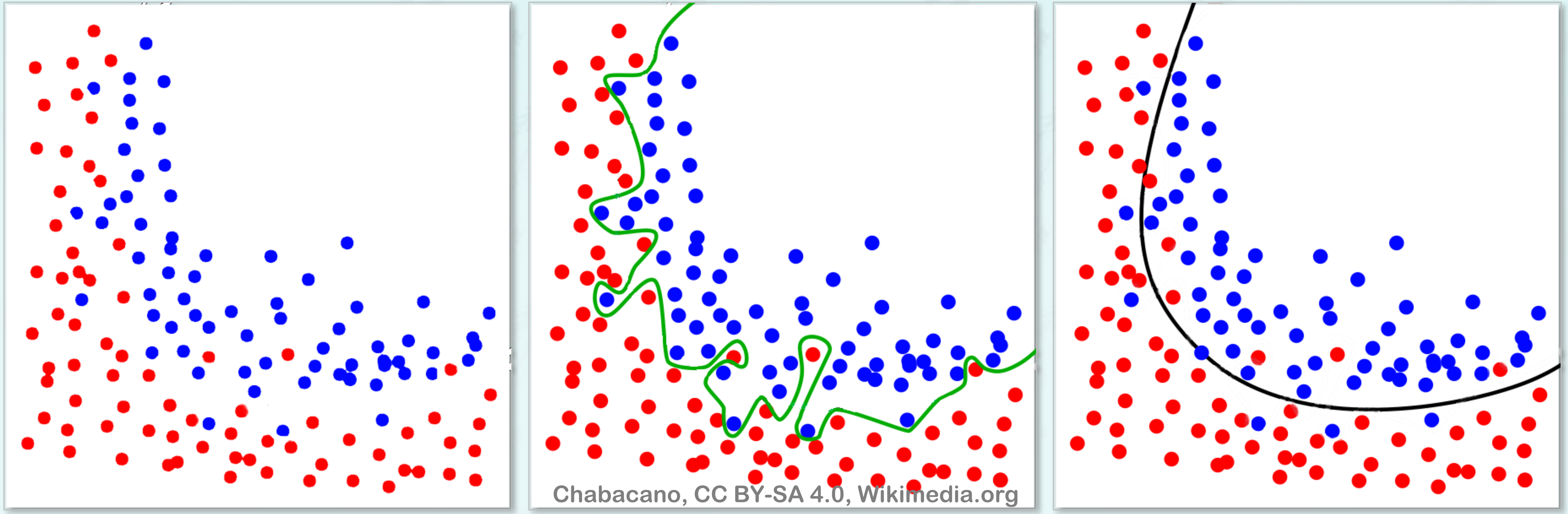


5. Overfitting



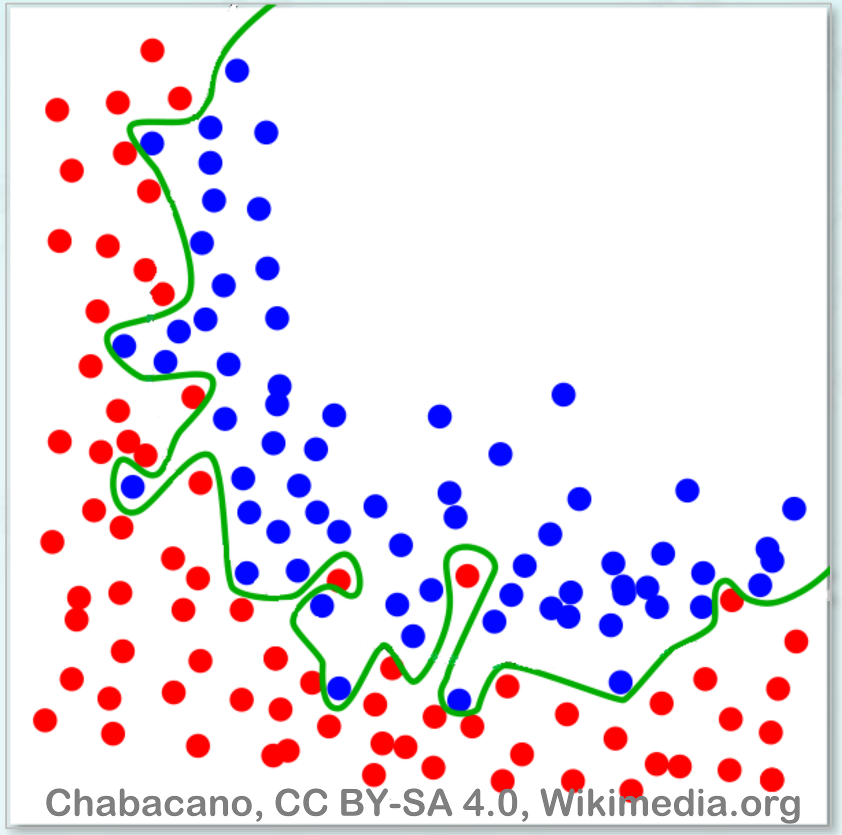
5. Overfitting

- A Better Classifier?
  1. Green line
  2. Black line



5. Overfitting

- Overfitting
- Underfitting



Perceptron

- Summary
  - Perceptron History
  - Perceptron Structure
  - Perceptron Learning
  - Binary Classifier and Activation Function
  - Overfitting and Underfitting
- Next
  - 4-2 Perceptron Algorithm

Week 4(1/3)

Activation Function

Machine Learning with Python

Handong Global University
Prof. Youngsup Kim
idebtor@gmail.com

여러분 곁에 항상 열려 있는 K-MOOC 강의실에서 만나 뵙기를 바랍니다.