Week 4(2/3)

Perceptron Algorithm

Machine Learning with Python

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

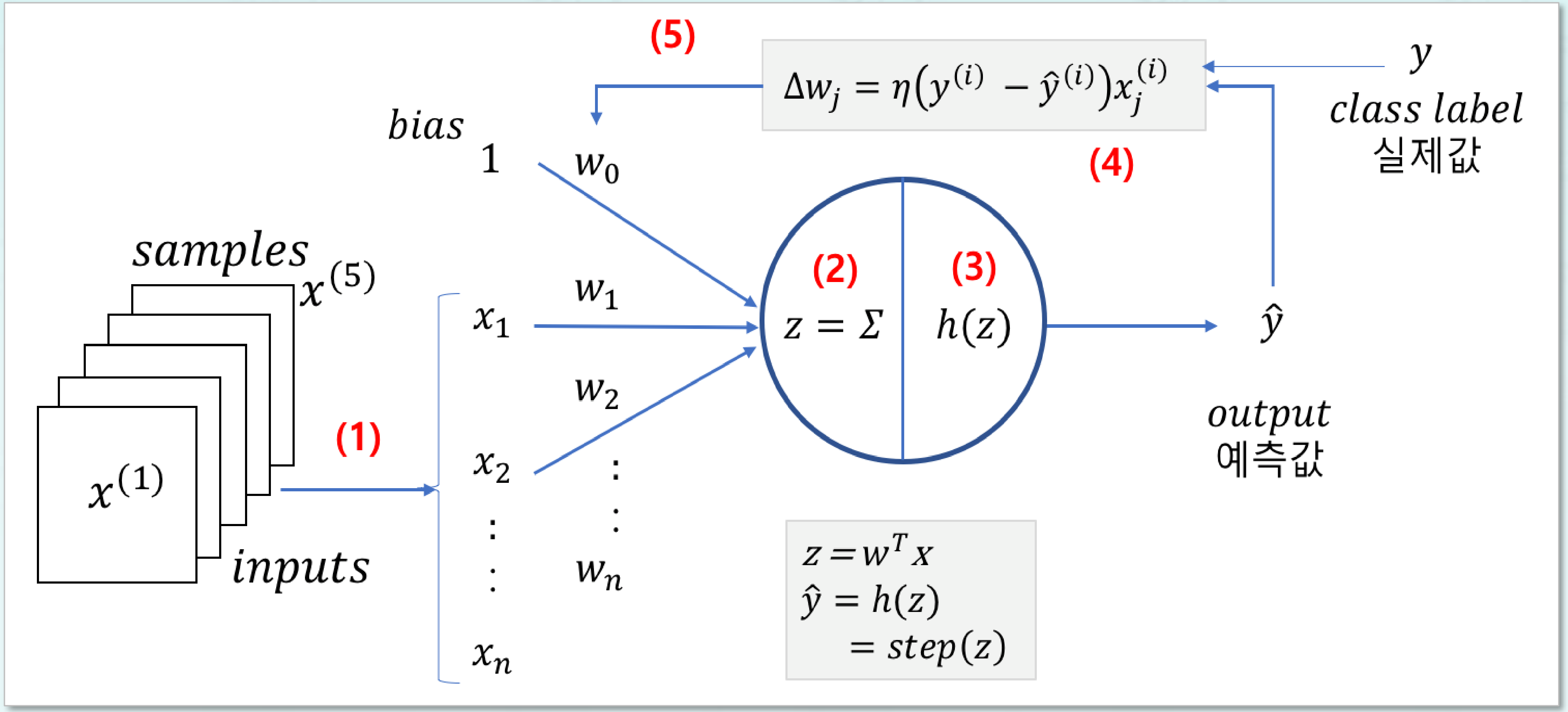
Perceptron Algorithm

- Goals
  - Understanding Perceptron Algorithm

- Topics
  - Perceptron Algorithm
  - Perceptron Weight Computation
  - Perceptron Learning Process
  - Perceptron Algorithm Limitation
  - Perceptron Example

1. Perceptron Algorithm: Purpose

- **Purpose:**
  - To compute **w** classifying **x**



1. Perceptron Algorithm: Algorithm

- **Purpose:**
  - To compute **w** classifying **x**

- **Algorithm:**
  - Initialize w with small random numbers

1. Perceptron Algorithm: Algorithm

- **Purpose:**
  - To compute **w** classifying **x**

- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$

1. Perceptron Algorithm: Algorithm

- **Purpose:**
  - To compute **w** classifying **x**

- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$

1. Perceptron Algorithm: Algorithm

- **Purpose:**
  - To compute **w** classifying **x**
- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$
- **Notation:**

1. Perceptron Algorithm: Notation

- **Purpose:**
  - To compute **w** classifying **x**
- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$
- **Notation:**
  - $x^{(i)}$
    (i)th training data, input

1. Perceptron Algorithm: Notation

- **Purpose:**
  - To compute **w** classifying **x**
- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$
- **Notation:**
  - $x^{(i)}$
    (i)th training data, input
  - $x_j^{(i)}$
    (i)th training data, j_th feature

1. Perceptron Algorithm: Notation

- **Purpose:**
  - To compute **w** classifying **x**
- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$
- **Notation:**
  - $x^{(i)}$
    (i)th training data, input
  - $x_j^{(i)}$
    (i)th training data, j_th feature
  - $\hat{y}$ (y hat)
    output, predicted value
  - y
    class label, actual value

1. Perceptron Algorithm: Notation

- **Purpose:**
  - To compute **w** classifying **x**
- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$
- **Notation:**
  - $x^{(i)}$
    (i)th training data, input
  - $x_j^{(i)}$
    (i)th training data, j_th feature
  - $\hat{y}$ (y hat)
    output, predicted value
  - y
    class label, actual value

1. Perceptron Algorithm: Notation

- **Purpose:**
  - To compute **w** classifying **x**

- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$

- **Notation:**
  - $x^{(i)}$
    (i)th training data, input
  - $x_j^{(i)}$
    (i)th training data, j_th feature
  - $\hat{y}$ (y hat)
    output, predicted value
  - y
    class label, actual value
  - w_j
    weight for j_th feature

1. Perceptron Algorithm: Notation

- **Purpose:**
  - To compute **w** classifying **x**

- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$

- **Notation:**
  - $x^{(i)}$
    (i)th training data, input
  - $x_j^{(i)}$
    (i)th training data, j_th feature
  - $\hat{y}$ (y hat)
    output, predicted value
  - y
    class label, actual value
  - w_j
    weight for j_th feature
  - Δw_j
    delta weight to adjust

1. Perceptron Algorithm: Weight Computation

- **Purpose:**
  - To compute **w** classifying **x**

- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$

- **Delta weight expression**

1. Perceptron Algorithm: Weight Computation

- **Purpose:**
  - To compute **w** classifying **x**

- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$

- **Delta weight expression**

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \qquad (1)$$

  - η (eta) learning rate [0..1]
  - j features including bias

$$\Delta w_0 = \eta(y^{(i)} - \hat{y}^{(i)})$$
$$\Delta w_1 = \eta(y^{(i)} - \hat{y}^{(i)})x_1^{(i)}$$
$$\Delta w_2 = \eta(y^{(i)} - \hat{y}^{(i)})x_2^{(i)}$$

1. Perceptron Algorithm: Weight Computation

- **Purpose:**
  - To compute **w** classifying **x**

- **Algorithm:**
  - Initialize w with small random numbers
  - For each training data $x^{(i)}$
    - compute output $\hat{y} = h(w^T x)$
    - adjust weight $w_j := w_j + \Delta w_j$

- **Delta weight expression**

$$\underbrace{\Delta w_j}_{\text{delta weight}} = \underbrace{\eta}_{\text{learning rate}}(\underbrace{y^{(i)}}_{\text{class label}} - \underbrace{\hat{y}^{(i)}}_{\text{predicted}})\underbrace{x_j^{(i)}}_{\text{input}} \qquad (1)$$

1. Perceptron Algorithm: Weight Computation

- Test the expression (1):
  - Bipolar step function returns -1 or 1
- $\boxed{\text{Case 1: } \hat{y} = y}$
  - $\Delta w_j =$
- Case 2: $\hat{y} \neq y$
  - $\Delta w_j =$

- Delta weight expression

$$\underbrace{\Delta w_j}_{\text{delta weight}} = \underbrace{\eta}_{\text{learning rate}}(\underbrace{y^{(i)}}_{\text{class label}} - \underbrace{\hat{y}^{(i)}}_{\text{predicted}})\underbrace{x_j^{(i)}}_{\text{input}} \qquad (1)$$

1. Perceptron Algorithm: Weight Computation

- Test the expression (1):
  - Bipolar step function returns -1 or 1
- Case 1: $\hat{y} = y$
  - $\Delta w_j = 0$
  - Therefore, no change in weight
- Case 2: $\hat{y} \neq y$
  - $\Delta w_j =$

- Delta weight expression

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \qquad (1)$$

predicted
input
class label
learning rate
delta weight

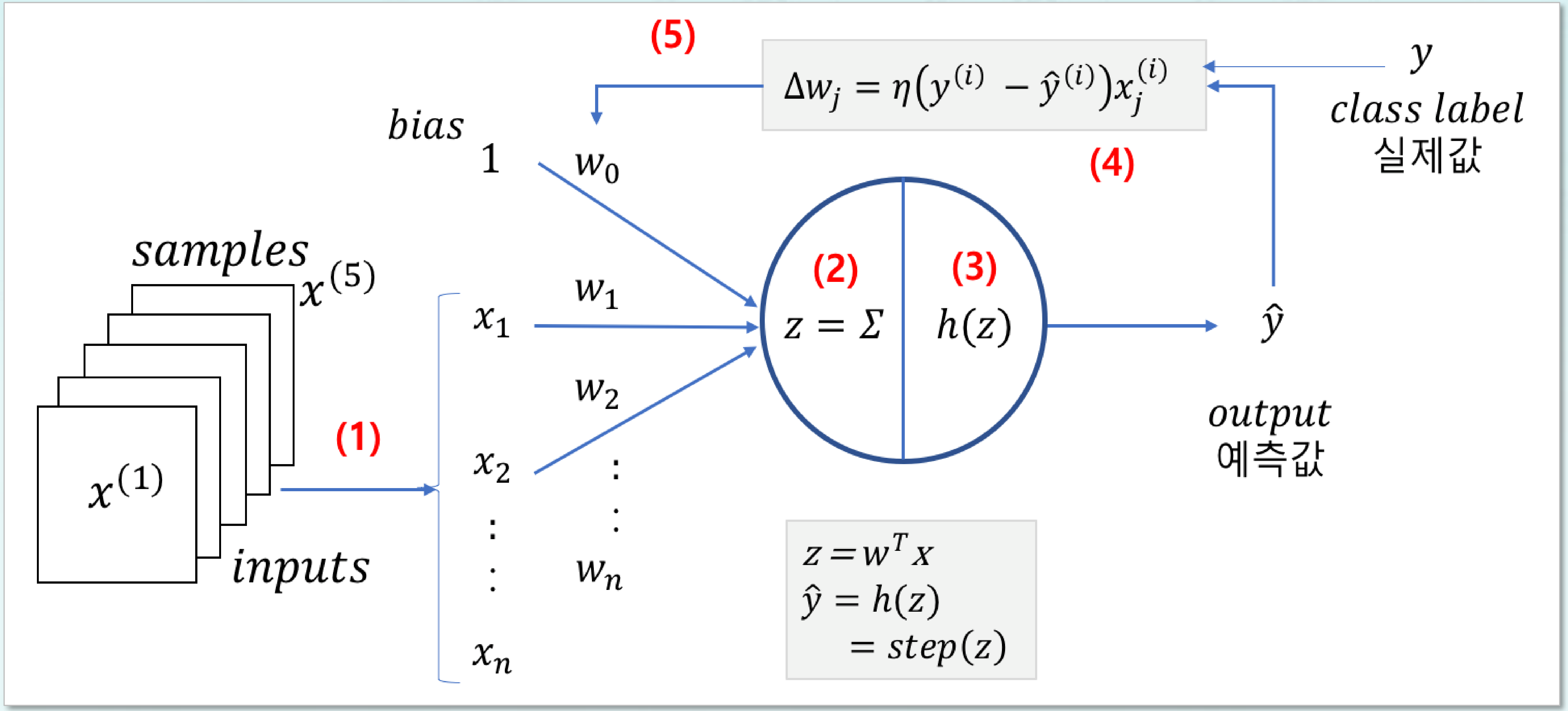
1. Perceptron Algorithm: Weight Computation

- Test the expression (1):
  - Bipolar step function returns -1 or 1
- Case 1: $\hat{y} = y$
  - $\Delta w_j = 0$
  - Therefore, no change in weight
- Case 2: $\hat{y} \neq y$
  - $\Delta w_j = \eta\left(1^i - \left(-1^i\right)\right) x_j^i = \eta(2) x_j^i$
  - $\Delta w_j = \eta\left(-1^i - 1^i\right) x_j^i = \eta(-2) x_j^i$

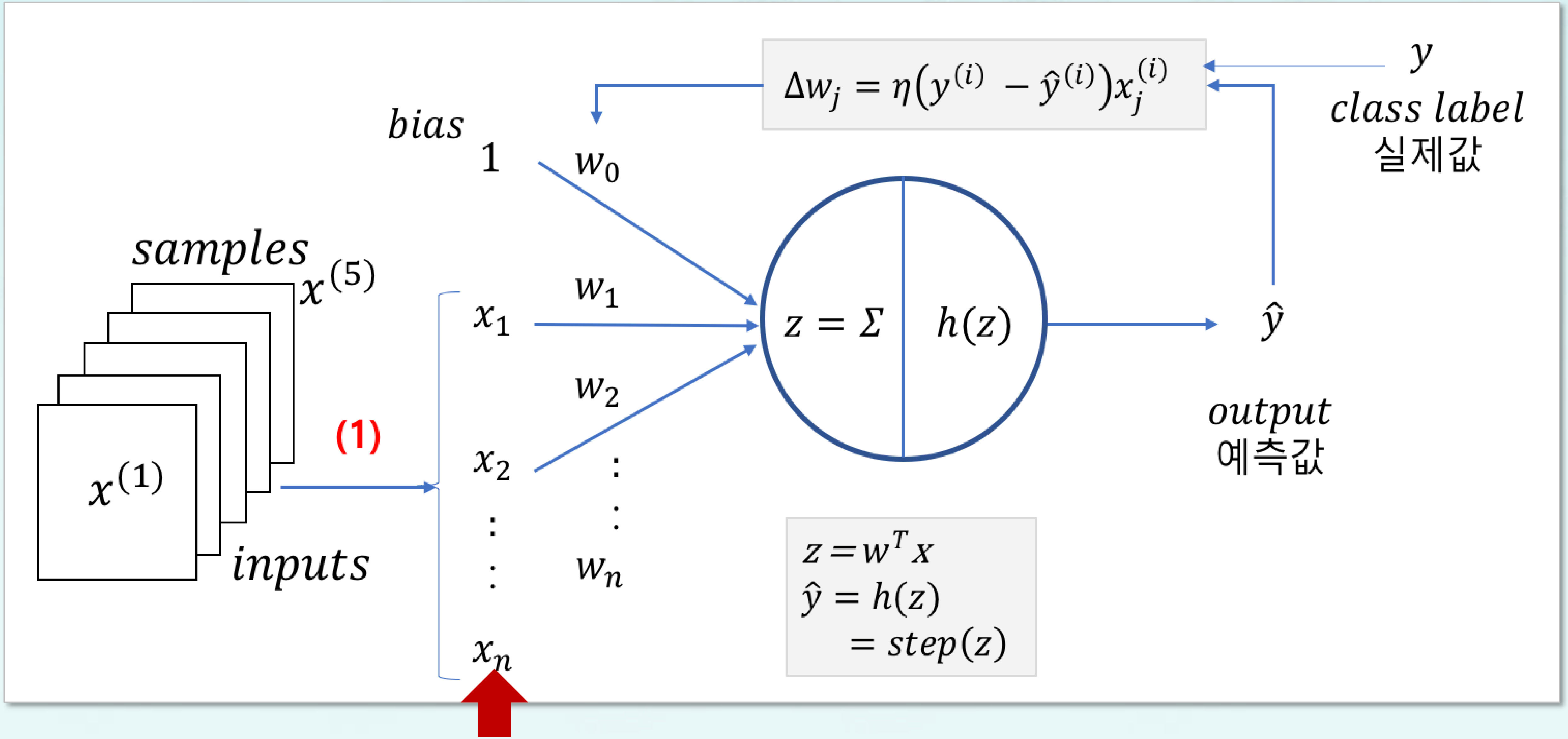
- Delta weight expression

$$\underbrace{\Delta w_j}_{\text{delta weight}} = \underbrace{\eta}_{\text{learning rate}}(\underbrace{y^{(i)}}_{\text{class label}} - \underbrace{\hat{y}^{(i)}}_{\text{predicted}})\underbrace{x_j^{(i)}}_{\text{input}} \qquad (1)$$

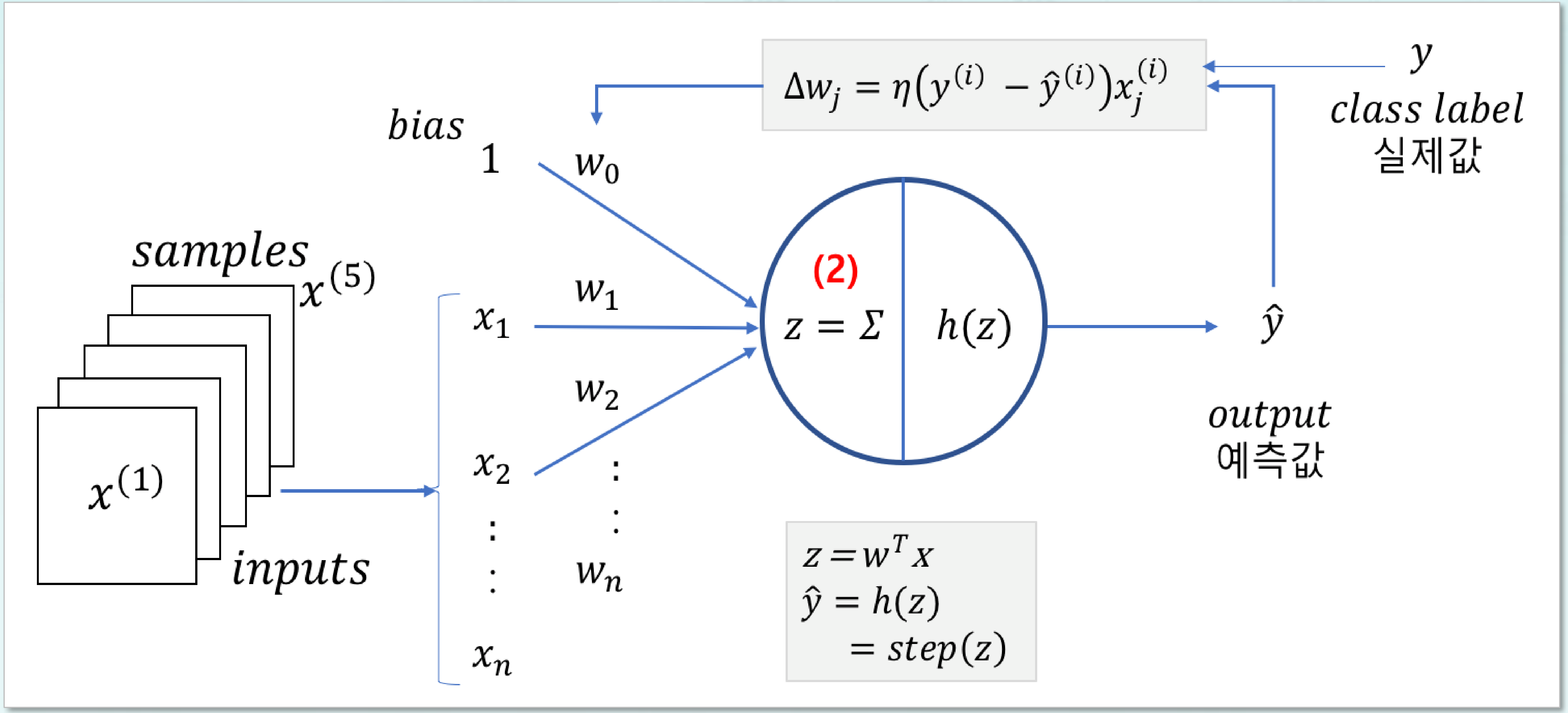
2. Perceptron Learning Process



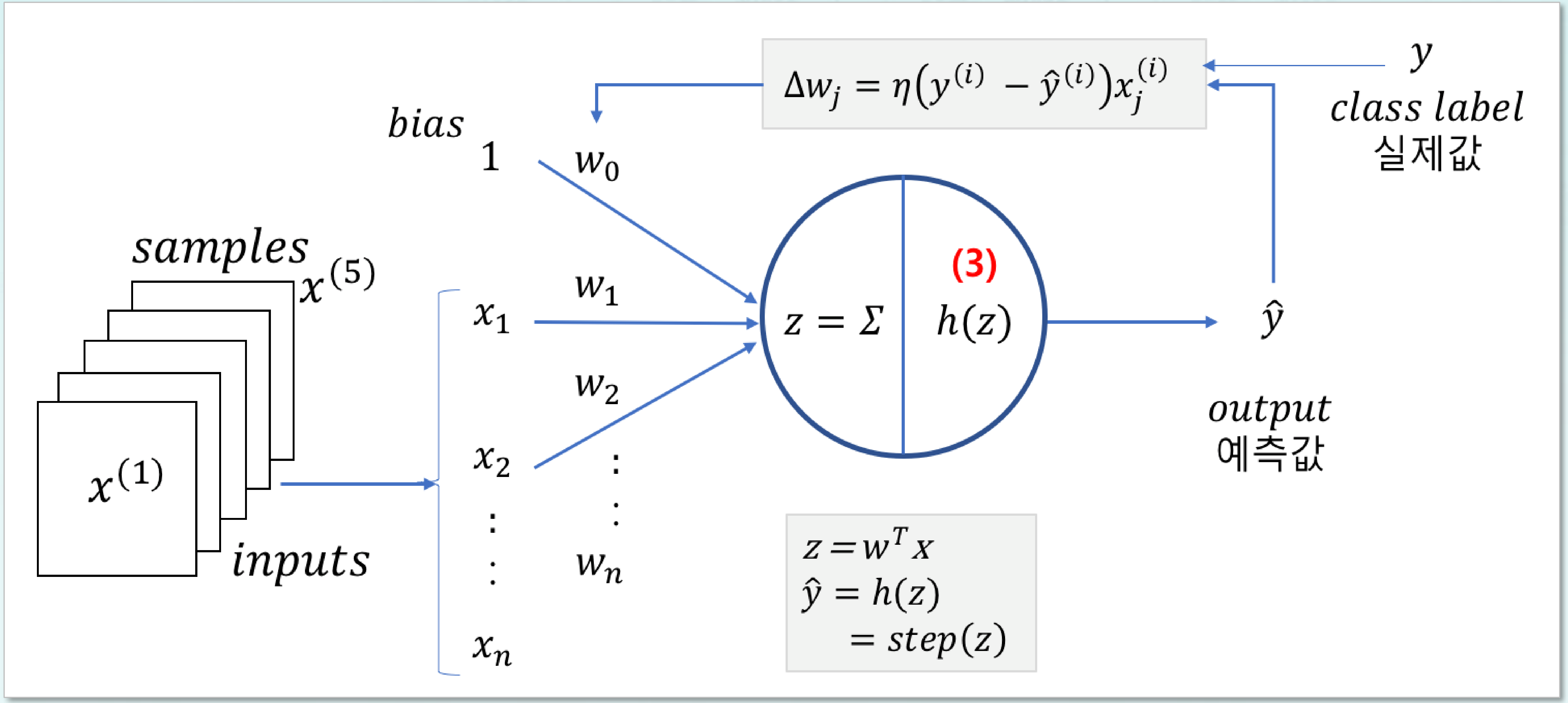
2. Perceptron Learning Process: Input



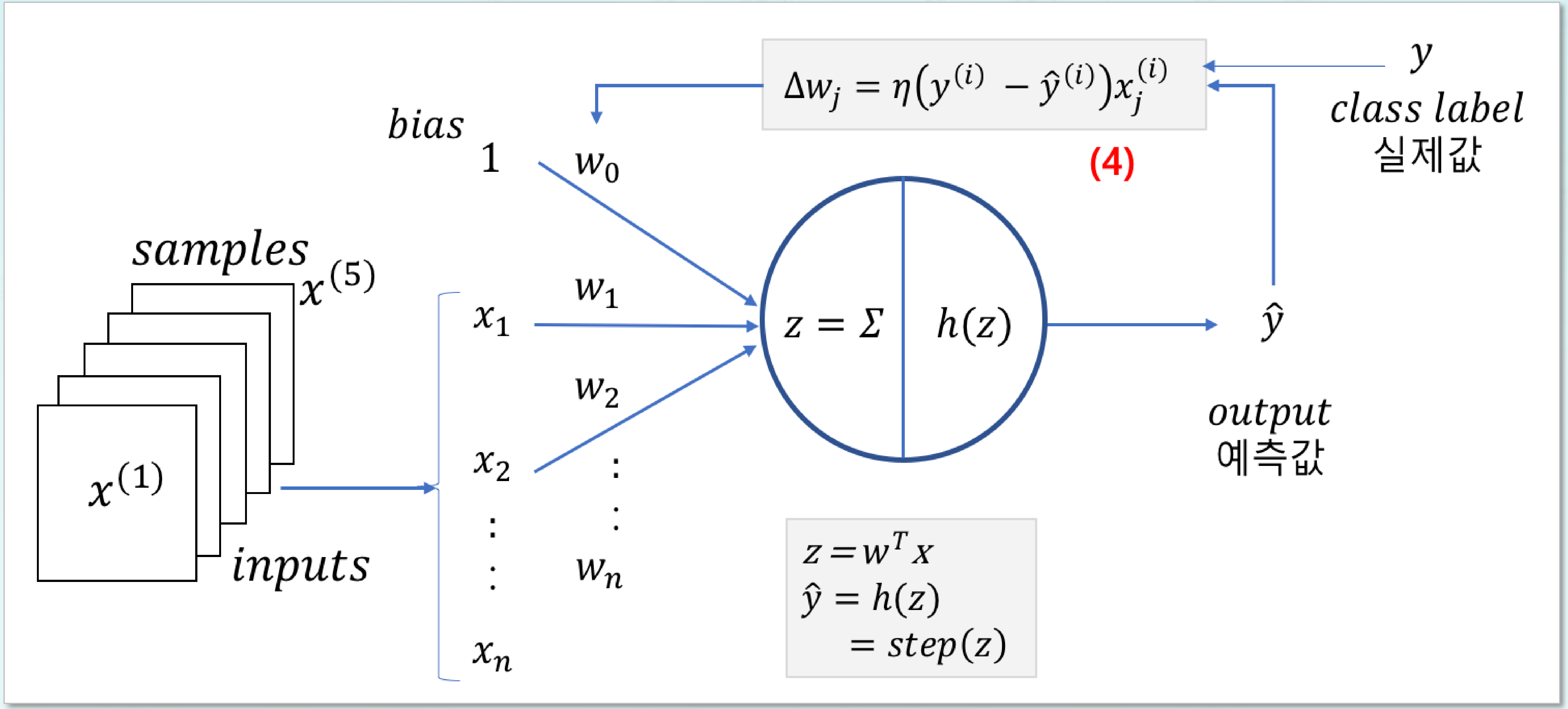
2. Perceptron Learning Process: Net input



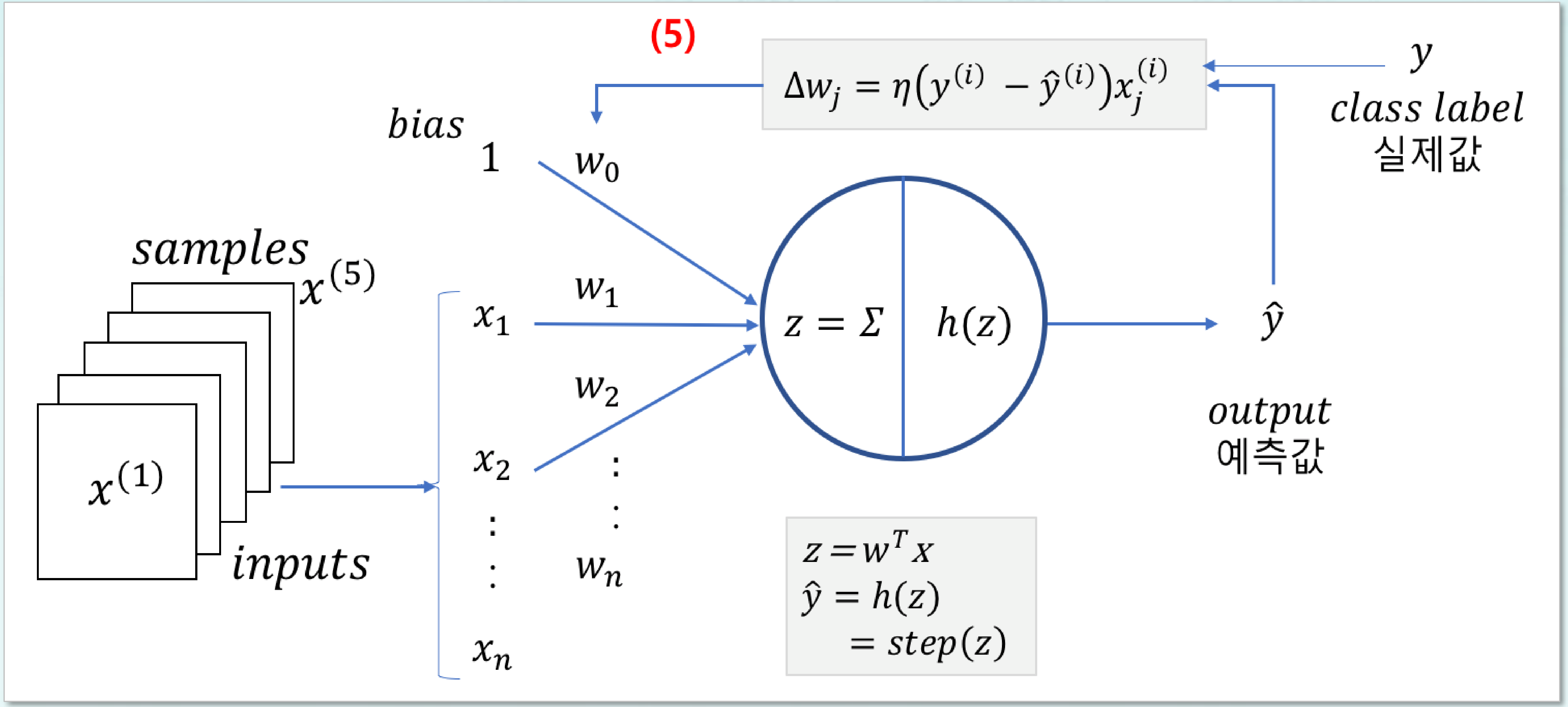
2. Perceptron Learning Process: Output



2. Perceptron Learning Process: Compute error



2. Perceptron Learning Process: Adjust weight



3. Perceptron Algorithm's Limit

- 1943: McCulloc-Pitt neuron
  - Warren McCullock, Wlater Pitts
- 1957: Perceptron by Rosenblatt
- 1958: New York Times
- 1969: Prof. Marvin Minsky at MIT
  - Perceptron's limit: XOR
  - Possible by Multi-Layer Perceptron
    But no solution found.
- 1974: Paul Worbros at Harvard
  - Graduate student
  - Backpropagation for MLP found
  - 1974, 1982, 1986(Hinton)
- 1980: LeCun
  - Convolutional Neural Network
- 2006, 2007: Hinton, Benjio
  - Breakthrough
  - Use Deep Learning instead of MLP
- 2018: LeCun, Hinton, Benjio

3. Perceptron Algorithm's Limit

- 1943: McCulloc-Pitt neuron
  - Warren McCullock, Wlater Pitts
- 1957: Perceptron by Rosenblatt
- 1958: New York Times
- 1969: Prof. Marvin Minsky at MIT
  - Perceptron's limit: XOR
  - Possible by Multi-Layer Perceptron But no solution found.
- 1974: Paul Worbros at Harvard
  - Graduate student
  - Backpropagation for MLP found
  - 1974, 1982, 1986(Hinton)
- 1980: LeCun
  - Convolutional Neural Network
- 2006, 2007: Hinton, Benjio
  - Breakthrough
  - Use Deep Learning instead of MLP
- 2018: LeCun, Hinton, Benjio

ACM A.M. Turing Award 2018:
Yan Lecun, Geoffrey Hinton and Yoshua Bengio,

Godfathers of AI' honored with Turing Award, the Nobel Prize of computing

3. Perceptron Algorithm’s Limit

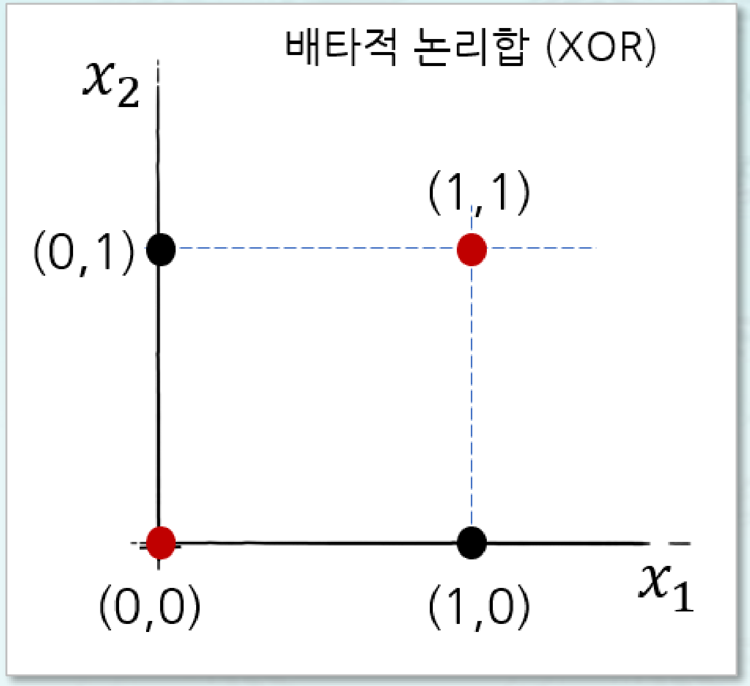
XOR 진리표

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

3. Perceptron Algorithm's Limit

XOR 진리표

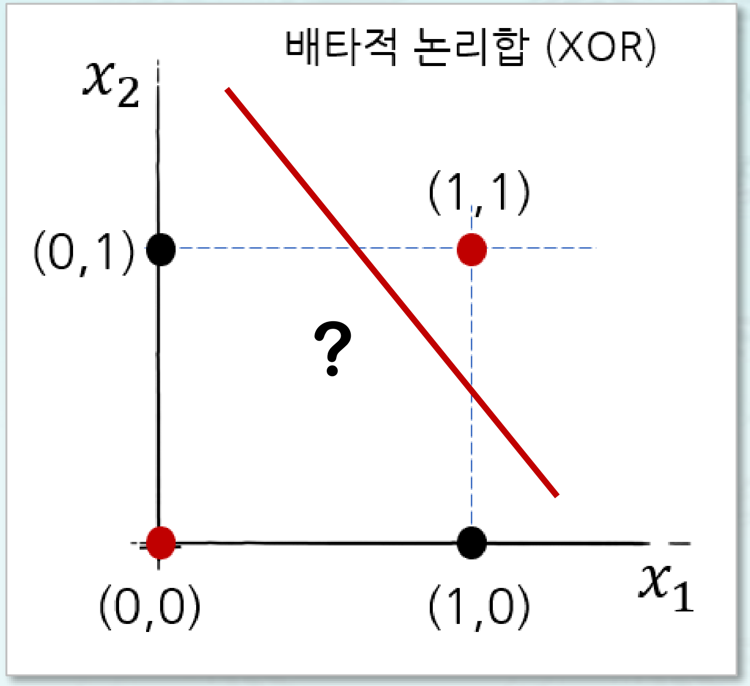
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



3. Perceptron Algorithm’s Limit

XOR 진리표

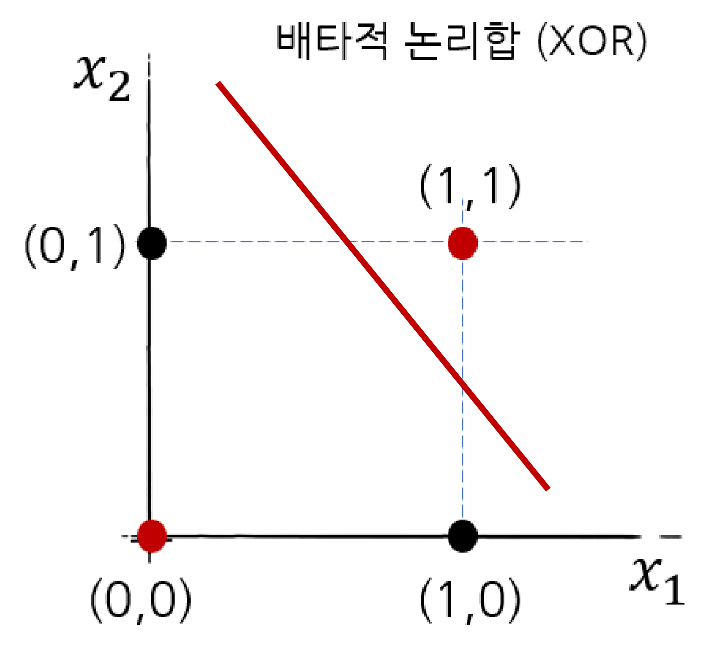
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

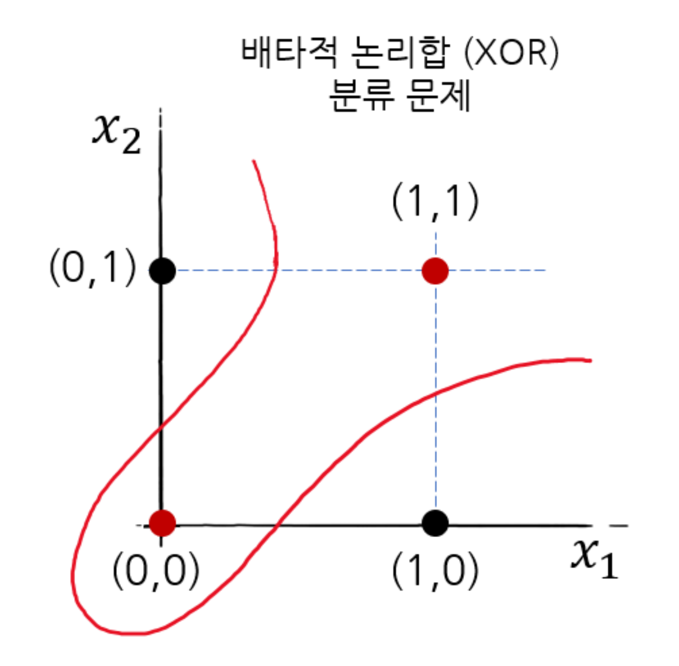


3. Perception Algorithm's Limit

XOR 진리표

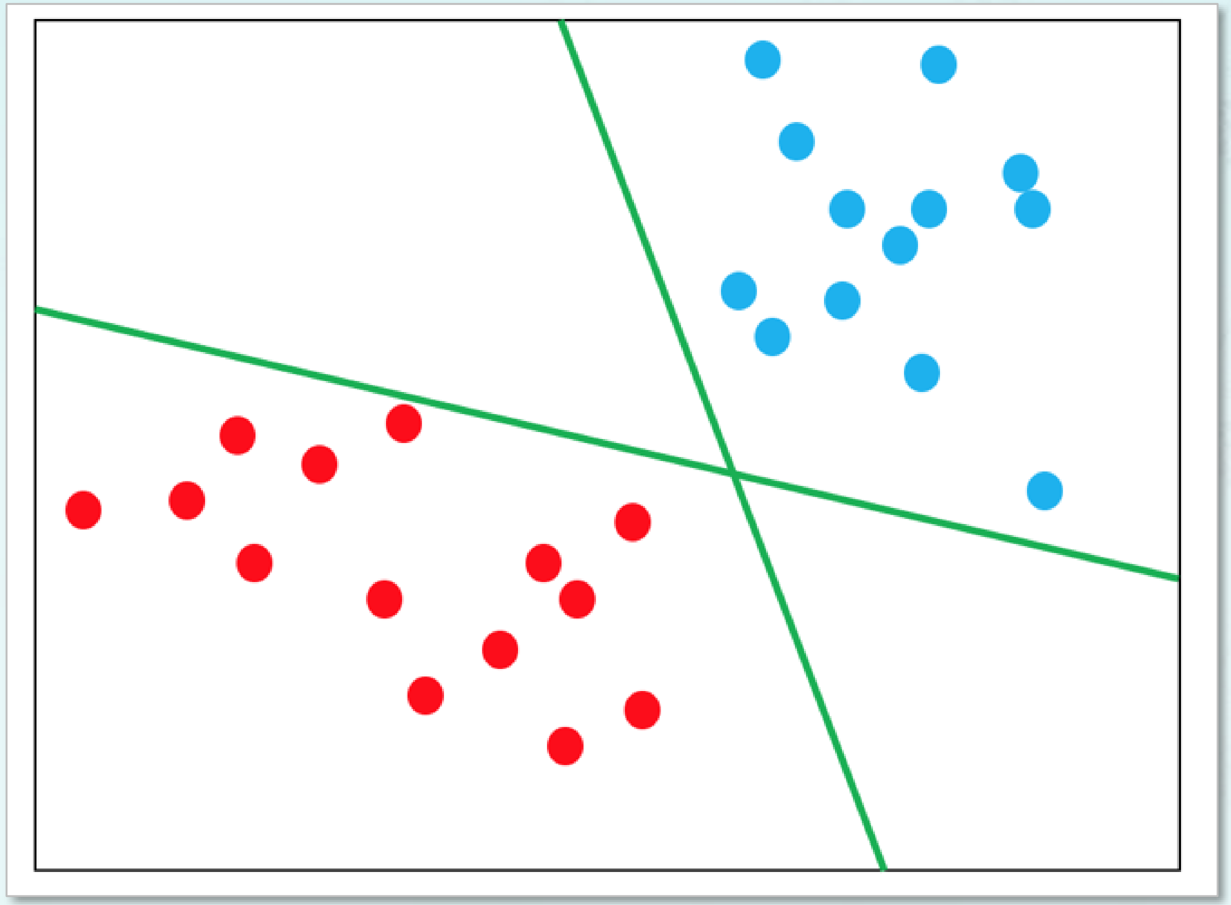
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

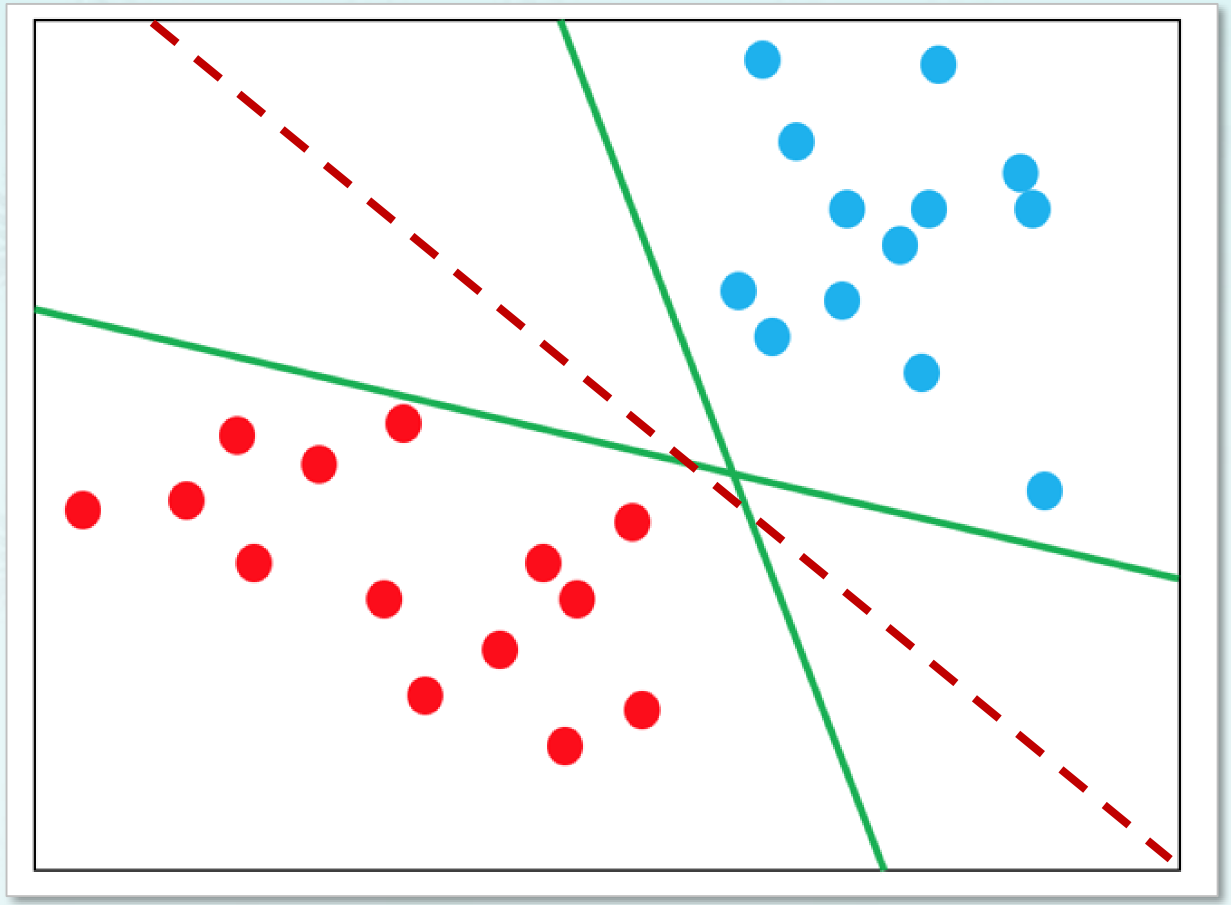




3. Perceptron Algorithm’s Limit: Classification

- **The green straight lines classify two classes, but not the best fit.**

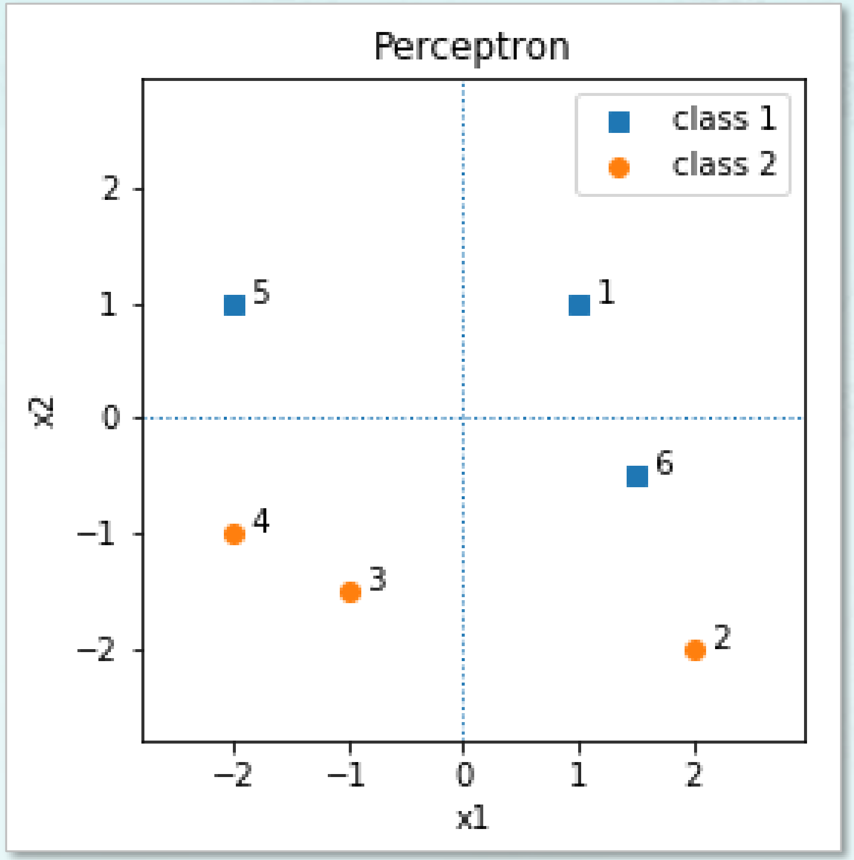




4. Perceptron Example

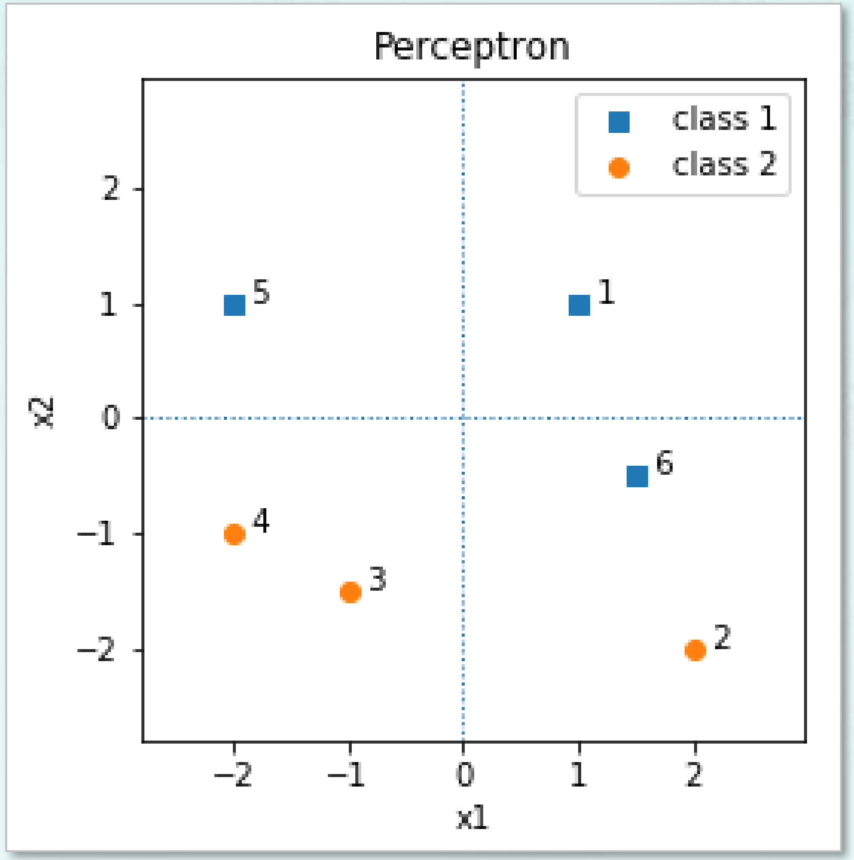
4. Perceptron Example: Training data

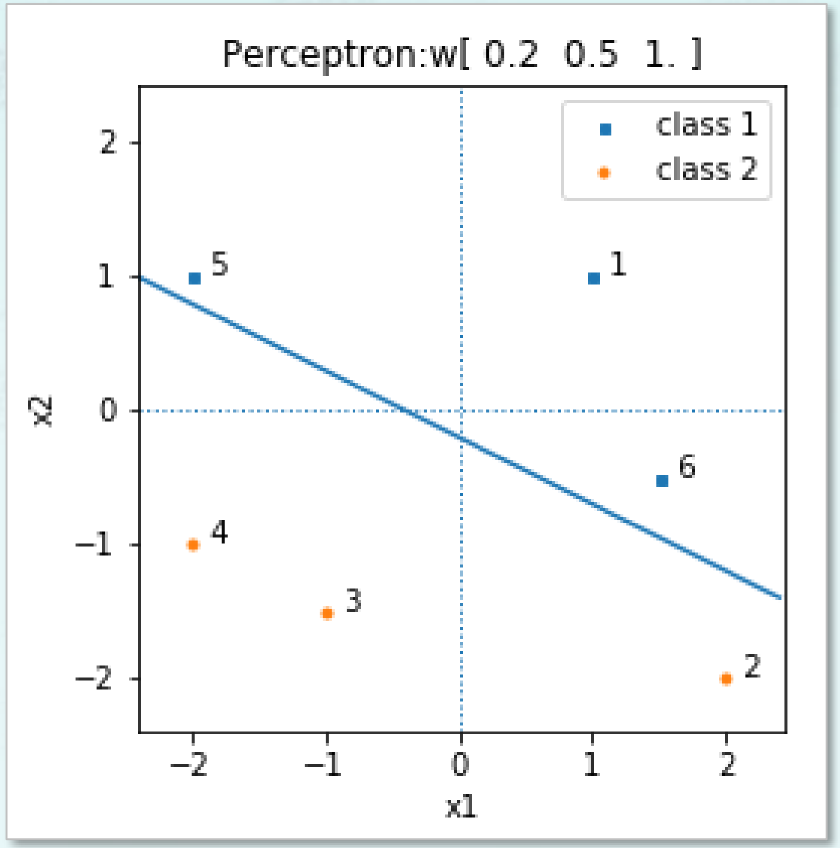
- 6 training data
- Class label: y = [1, -1, -1, -1, 1, 1]



4. Perceptron Example: Training data

- 6 training data
- Class label: y = [1, -1, -1, -1, 1, 1]





4. Perceptron Example: Weight computation

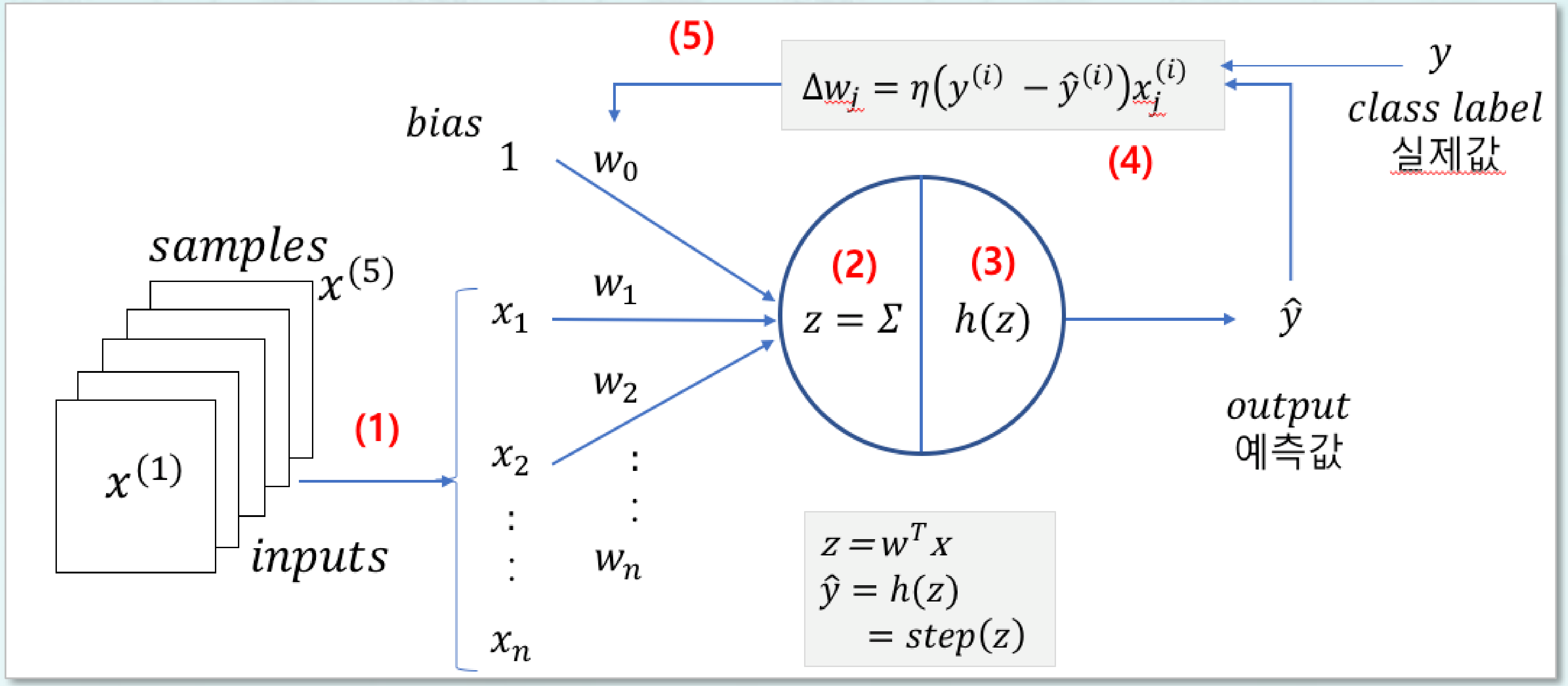
- Step 1: Compute weight w
  - Initial weights:
    - $w^T = [0 \quad 1 \quad 0.5]$
  - Learning rate $\eta = 0.1$

4. Perceptron Example: Weight computation

- **Step 1: Compute weight w**
  - **Initial weights:**
    - **w^T = [0 1 0.5]**
  - **Learning rate $\eta = 0.1$**
  - **Training data:**
    - $x^{(1)} = [1, 1]$
    - $x^{(2)} = [2, -2]$
    - $x^{(3)} = [-1, -1.5]$
    - $x^{(4)} = [-2, -1.0]$
    - $x^{(5)} = [1, -2.0, 1.0]$
    - $x^{(6)} = [1, 1.5, -0.5]$

4. Perceptron Example: Weight computation

- **Step 1: Compute weight w**
  - **Initial weights:**
    - $w^T = [0 \quad 1 \quad 0.5]$
  - **Learning rate** $\eta = 0.1$
  - **Training data:**
    - $x^{(1)} = [1, 1]$
    - $x^{(2)} = [2, -2]$
    - $x^{(3)} = [-1, -1.5]$
    - $x^{(4)} = [-2, -1.0]$
    - $x^{(5)} = [1, -2.0, 1.0]$
    - $x^{(6)} = [1, 1.5, -0.5]$



4. Perceptron Example: Weight computation

- Step 1: Compute weight w

i	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^{\mathrm{T}}\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)					
2	(1.0, 2.0, -2.0)						
3	(1.0, -1.0, -1.5)						
4	(1.0, -2.0, -1.0)						
5	(1.0, -2.0, 1.0)						
6	(1.0, 1.5, -0.5)						
$final$							

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

$\boldsymbol{i}$	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^{\mathrm{T}}\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)			1	0.1	
2	(1.0, 2.0, -2.0)				-1	0.1	
3	(1.0, -1.0, -1.5)				-1	0.1	
4	(1.0, -2.0, -1.0)				-1	0.1	
5	(1.0, -2.0, 1.0)				1	0.1	
6	(1.0, 1.5, -0.5)				1	0.1	
$final$							

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

i	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^{\mathrm{T}}\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)			1	0.1	
2	(1.0, 2.0, -2.0)				-1	0.1	
3	(1.0, -1.0, -1.5)				-1	0.1	
4	(1.0, -2.0, -1.0)				-1	0.1	
5	(1.0, -2.0, 1.0)				1	0.1	
6	(1.0, 1.5, -0.5)				1	0.1	
$final$							

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

i	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^T\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)	1.5	1.0	1	0.1	0
2	(1.0, 2.0, -2.0)	(0.0, 1.0, 0.5)			-1	0.1	
3	(1.0, -1.0, -1.5)				-1	0.1	
4	(1.0, -2.0, -1.0)				-1	0.1	
5	(1.0, -2.0, 1.0)				1	0.1	
6	(1.0, 1.5, -0.5)				1	0.1	
$final$							

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

$\boldsymbol{i}$	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^{\mathrm{T}}\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)	1.5	1.0	1	0.1	0
2	(1.0, 2.0, -2.0)	(0.0, 1.0, 0.5)	1.0	1.0	-1	0.1	
3	(1.0, -1.0, -1.5)				-1	0.1	
4	(1.0, -2.0, -1.0)				-1	0.1	
5	(1.0, -2.0, 1.0)				1	0.1	
6	(1.0, 1.5, -0.5)				1	0.1	
$\boldsymbol{final}$							

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

i	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^T\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)	1.5	1.0	1	0.1	0
2	(1.0, 2.0, -2.0)	(0.0, 1.0, 0.5)	1.0	1.0	-1	0.1	
3	(1.0, -1.0, -1.5)						
4	(1.0, -2.0, -1.0)						
5	(1.0, -2.0, 1.0)						
6	(1.0, 1.5, -0.5)						
$final$							

$$\begin{aligned}\Delta w_j &= \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \\ &= 0.1(-1 - 1)x_j^{(2)} \\ &= -0.2x_j^{(2)}\end{aligned}$$

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

i	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^T\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)	1.5	1.0	1	0.1	0
2	(1.0, 2.0, -2.0)	(0.0, 1.0, 0.5)	1.0	1.0	-1	0.1	
3	(1.0, -1.0, -1.5)						
4	(1.0, -2.0, -1.0)						
5	(1.0, -2.0, 1.0)						
6	(1.0, 1.5, -0.5)						
$final$							

$$\begin{aligned}\Delta w_j &= \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \\ &= 0.1(-1 - 1)x_j^{(2)} \\ &= -0.2x_j^{(2)}\end{aligned}$$

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

i	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^{\mathrm{T}}\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)	1.5	1.0	1	0.1	0
2	(1.0, 2.0, -2.0)	(0.0, 1.0, 0.5)	1.0	1.0	-1	0.1	(-.2, -.4, .4)
3	(1.0, -1.0, -1.5)						
4	(1.0, -2.0, -1.0)						
5	(1.0, -2.0, 1.0)						
6	(1.0, 1.5, -0.5)						
$final$							

$$
\begin{aligned}
\Delta w_j &= \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \\
&= 0.1(-1 - 1)x_j^{(2)} \\
&= -0.2x_j^{(2)}
\end{aligned}
$$

$$
\boxed{
\begin{aligned}
\Delta w &= -0.2(1.0, 2.0, -2.0) \\
&= (-0.2, -0.4, 0.4)
\end{aligned}
}
$$

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

i	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^{\mathrm{T}}\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)	1.5	1.0	1	0.1	0
2	(1.0, 2.0, -2.0)	W (0.0, 1.0, 0.5)	1.0	1.0	-1	0.1	ΔW (-.2, -.4, .4)
3	(1.0, -1.0, -1.5)	(-2.0, 0.6, 0.9)					
4	(1.0, -2.0, -1.0)	$W + \Delta W$					
5	(1.0, -2.0, 1.0)						
6	(1.0, 1.5, -0.5)						
$final$							

$$
\begin{aligned}
\Delta w_j &= \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)} \\
&= 0.1(-1 - 1)x_j^{(2)} \\
&= -0.2x_j^{(2)} \\
\Delta w &= -0.2(1.0, 2.0, -2.0) \\
&= (-0.2, -0.4, 0.4)
\end{aligned}
$$

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

$\boldsymbol{i}$	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^{\mathrm{T}}\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)	1.5	1.0	1	0.1	0
2	(1.0, 2.0, -2.0)	(0.0, 1.0, 0.5)	1.0	1.0	-1	0.1	(-.2, -.4, .4)
3	(1.0, -1.0, -1.5)	(-2.0, 0.6, 0.9)	-2.15	-1	-1	0.1	0
4	(1.0, -2.0, -1.0)	(-0.2, 0.6, 0.9)	-2.3	-1	-1	0.1	0
5	(1.0, -2.0, 1.0)	(0.0, 0.2, 1.1)	-0.25	-1	1	0.1	(.2, -.4, .2)
6	(1.0, 1.5, -0.5)	(0.0, 0.2, 1.1)	-0.25	-1	1	0.1	(.2, .3, -.1)
$final$	-	(0.2, 0.5, 1.0)		-	-	-	

4. Perceptron Example: Weight computation

- Step 1: Compute weight w

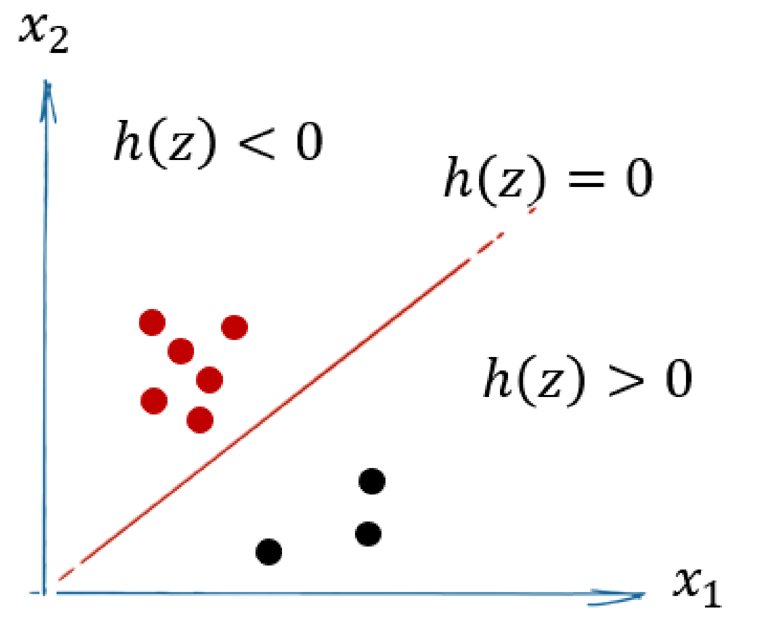
i	$(x_0^{(i)}, x_1^{(i)}, x_0^{(i)})$	(w_0, w_1, w_2)	$\mathbf{w}^{\mathrm{T}}\mathbf{x}$	$\hat{y}^{(i)}$	$y^{(i)}$	η	Δw
1	(1.0, 1,0, 1.0)	(0.0, 1.0, 0.5)	1.5	1.0	1	0.1	0
2	(1.0, 2.0, -2.0)	(0.0, 1.0, 0.5)	1.0	1.0	-1	0.1	(-.2, -.4, .4)
3	(1.0, -1.0, -1.5)	(-2.0, 0.6, 0.9)	-2.15	-1	-1	0.1	0
4	(1.0, -2.0, -1.0)	(-0.2, 0.6, 0.9)	-2.3	-1	-1	0.1	0
5	(1.0, -2.0, 1.0)	(0.0, 0.2, 1.1)	-0.25	-1	1	0.1	(.2, -.4, .2)
6	(1.0, 1.5, -0.5)	(0.0, 0.2, 1.1)	-0.25	-1	1	0.1	(.2, .3, -.1)
$final$	-	(0.2, 0.5, 1.0)		-	-	-	

4. Perceptron Example: Decision boundary

- Step 1: Compute weight w
  - w = [0.2, 0.5, 1.0]
- Step 2: Compute decision boundary

4. Perceptron Example: Decision boundary

- Step 1: Compute weight w
  - w = [0.2, 0.5, 1.0]
- Step 2: Compute decision boundary
  - $h(z) = 0$ or $h(w^T x) = 0$



선형 이진 분류기($z = w^T x$)

4. Perceptron Example: Decision boundary

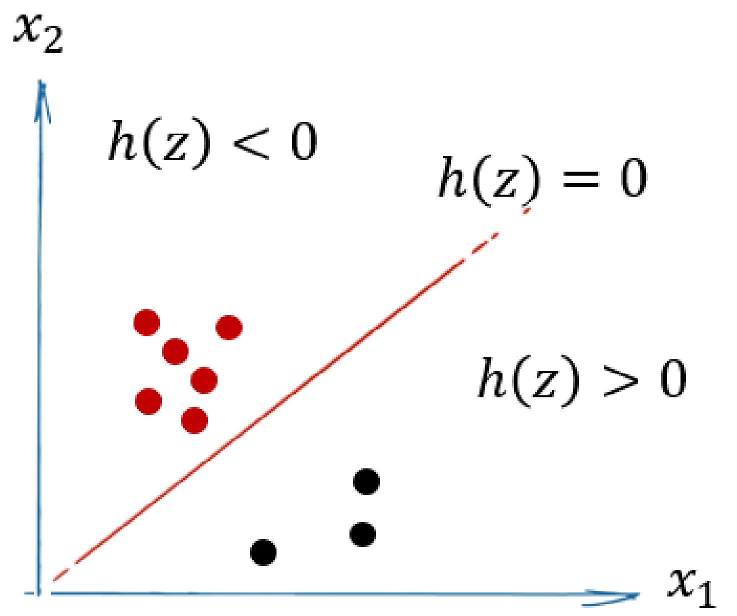
- Step 1: Compute weight w
  - w = [0.2, 0.5, 1.0]
- Step 2: Compute decision boundary
  - $h(z) = 0$ or $h(\mathbf{w}^T x) = 0$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} = 0$$

$$\begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = 0$$

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$0.2 + 0.5 x_1 + 1.0 x_2 = 0$$



선형 이진 분류기($z = w^T x$)

4. Perceptron Example: Decision boundary

- Step 1: Compute weight w
  - w = [0.2, 0.5, 1.0]
- Step 2: Compute decision boundary
  - h(z) = 0 or h(w^Tx) = 0

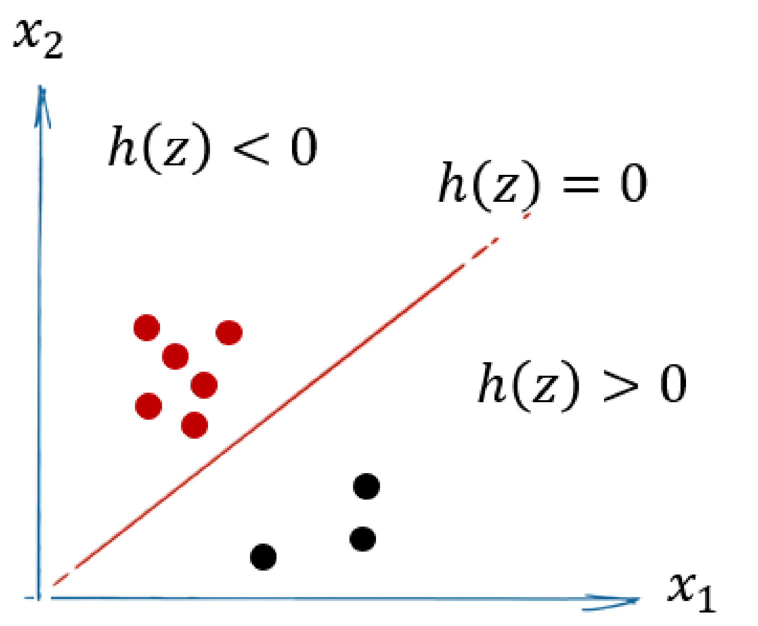
$$\mathbf{w}^{\mathrm{T}}\mathbf{x} = 0$$

$$\begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = 0$$

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$0.2 + 0.5 x_1 + 1.0 x_2 = 0$$

$$\boxed{x_2 = -0.5 x_1 - 0.2}$$

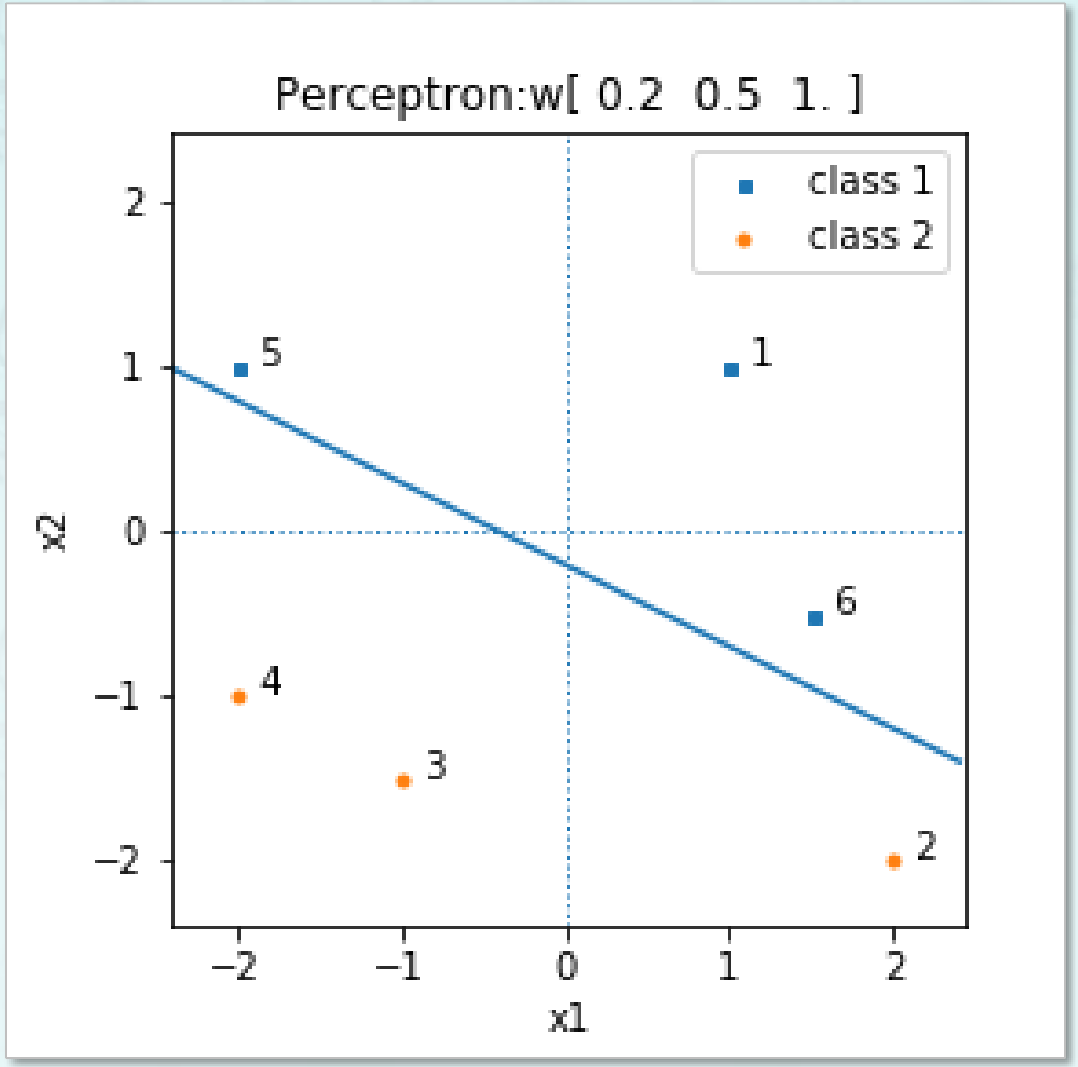


선형 이진 분류기($z = w^T x$)

4. Perceptron Example: Visualization

- Step 1: Compute weight w
  - w = [0.2, 0.5, 1.0]
- Step 2: Compute decision boundary
  - $x_2 = -.5x_1 - 0.2$
- Step 3: Visualization
  - plot_xyw()

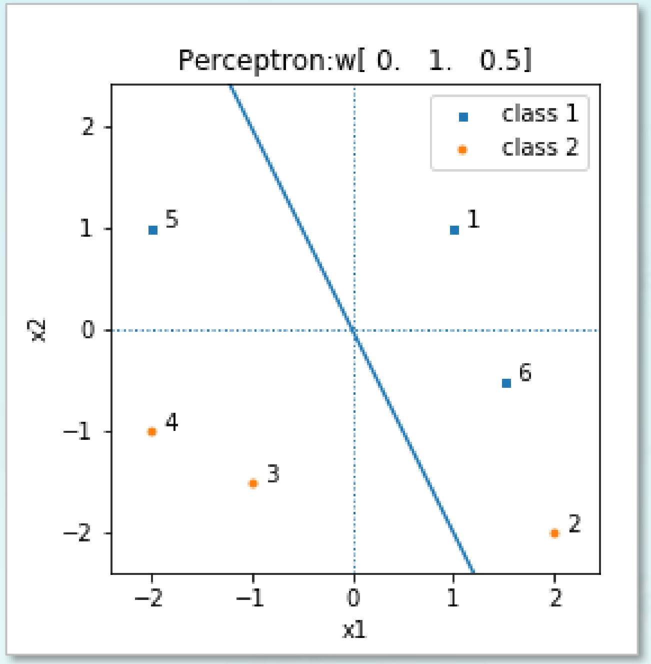
- plot_xyw()



4. Perceptron Example: Visualization

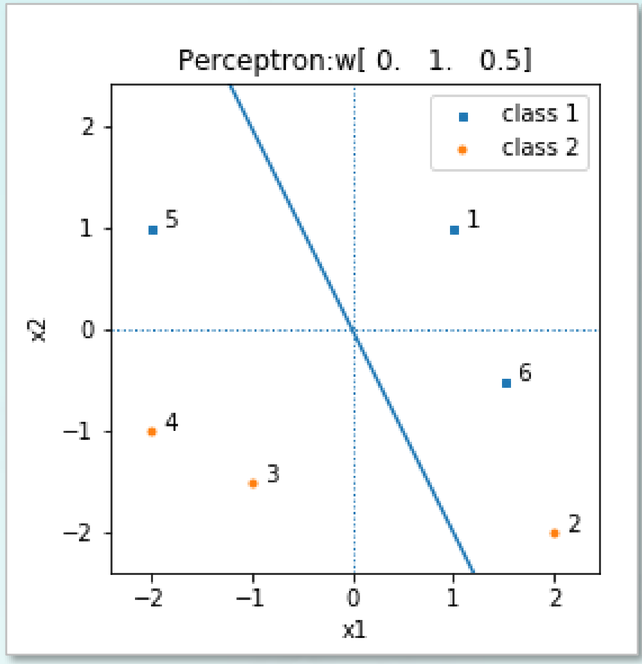
```python
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
%run code/plot_xyw.py

x = np.array([[1.0, 1.0], [2.0, -2.0], [-1.0, -1.5],
              [-2.0, -1.0], [-2.0,1.0], [1.5, -0.5]])
X = np.c_[ np.ones(len(x)), x ]
y = np.array([1, -1, -1, -1, 1, 1])
w = np.array([0.2, 0.5, 1.0])
plot_xyw(X, y, w, X0=True, annotate=True)
```

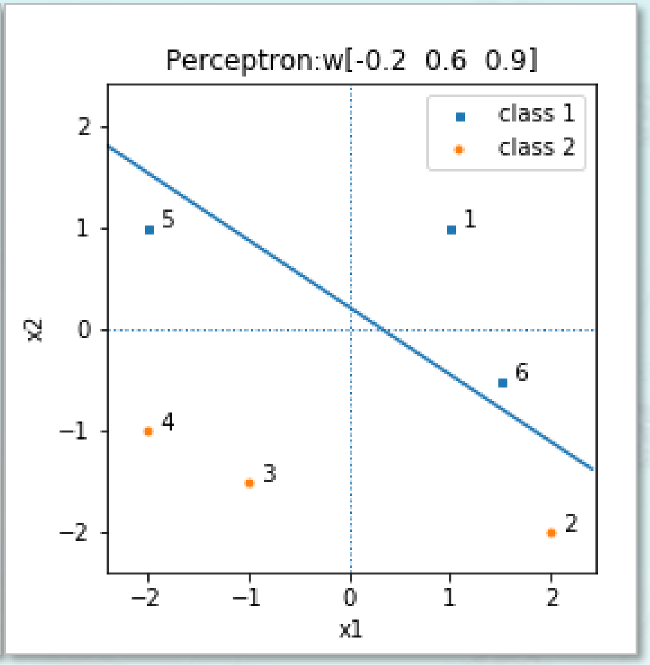
4. Perceptron Example: Convergence of decision boundary



w[0.0   1.0   0.5]

4. Perceptron Example: Convergence of decision boundary

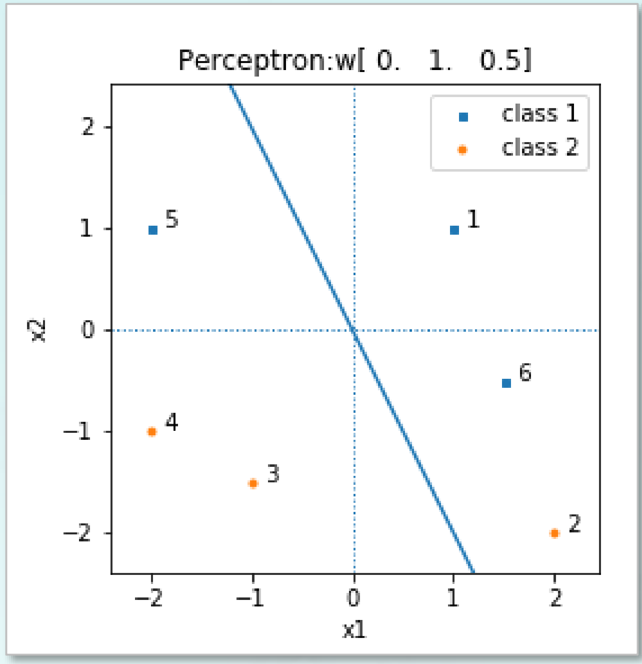


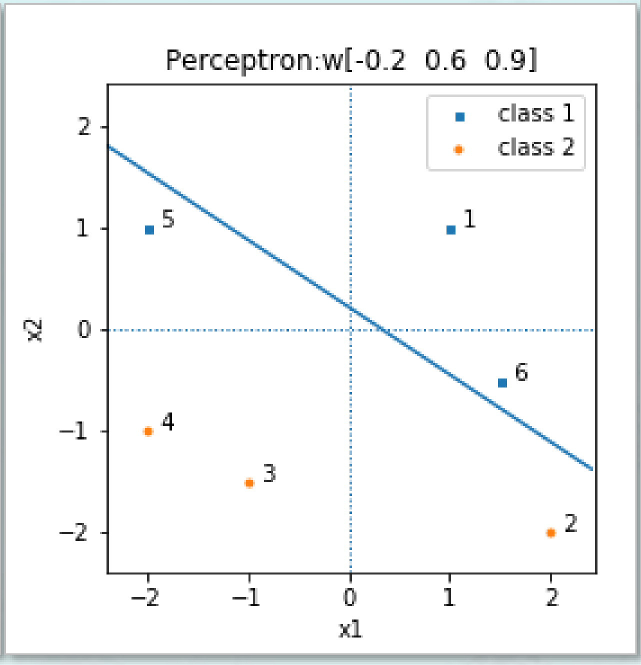


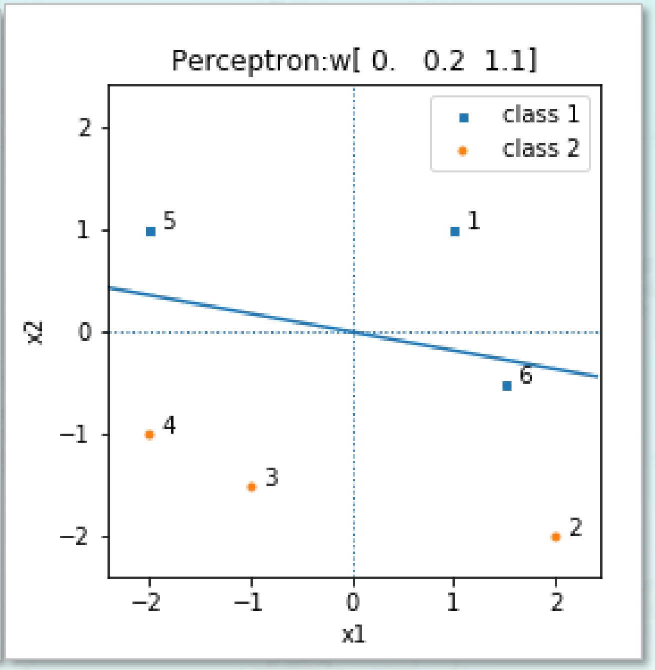
w[0.0   1.0   0.5]

w[0.2   0.6   0.9]

4. Perceptron Example: Convergence of decision boundary

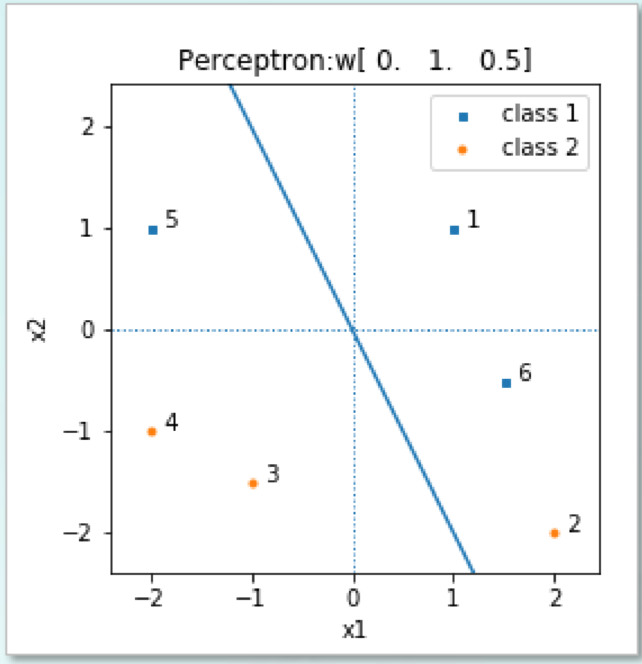




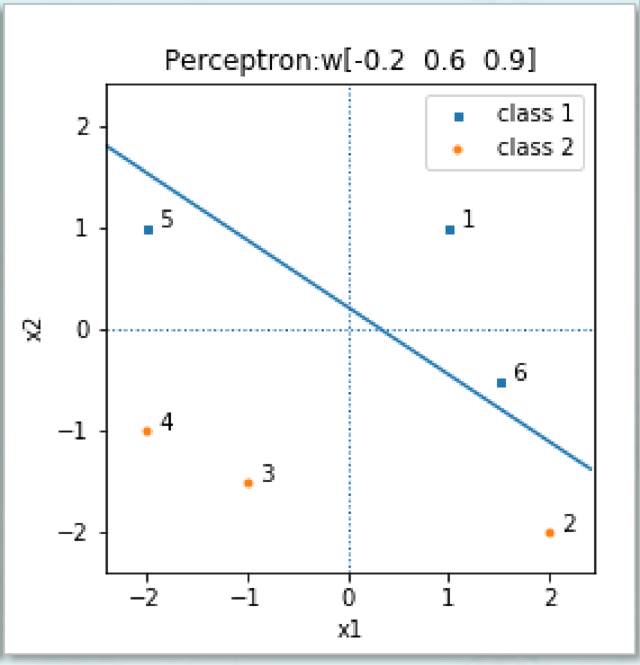


w[0.0   1.0   0.5]   w[0.2   0.6   0.9]   w[0.0   0.2   1.1]

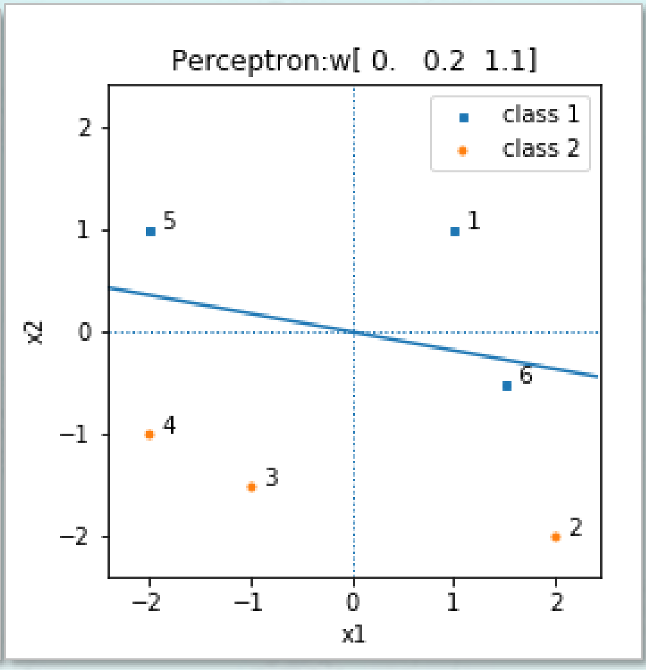
4. Perceptron Example: Convergence of decision boundary



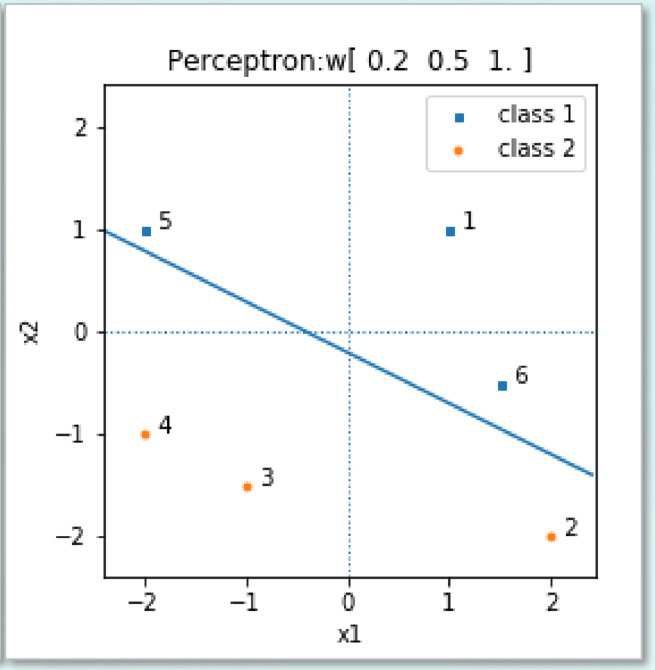
w[0.0   1.0   0.5]



w[0.2   0.6   0.9]



w[0.0   0.2   1.1]



w[0.2   0.5   1.0]

Perceptron Algorithm

- Summary
  - Perceptron Algorithm
  - Perceptron Weight Computation
  - Perceptron Learning Process
  - Perceptron Algorithm Limitation
  - Perceptron Example

- Next
  - 4-3 Perceptron Algorithm Implementation

Week 4(2/3)

Perceptron

Machine Learning with Python

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