11.1 Single Perceptron

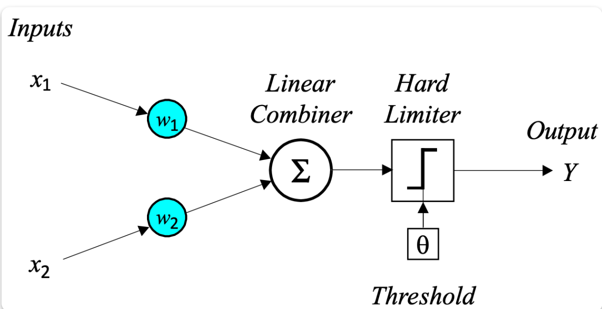
11.1.1 Basics

ℹ️ The neuron is an *abstract* concept of a simple computing unit.

1. Compute weighted sum of inputs.
2. Activate the weighted sum with activation function.

ℹ️ Perceptron (Frank Rosenblatt, 1958)

- Simplest form of neural network.



The single perceptron could be described with three key components:

1. Weights of inputs $\mathbf{w} \in \mathbb{R}^D$.
2. Threshold $\theta \in \mathbb{R}$.
3. Activation function f.

The activation of a perceptron is:

$$Y = f(\mathbf{w}^\top \mathbf{x} - \theta)$$

11.1.2 Classification with Perceptron

ℹ️ A perceptron mimics a *Hyperplane.*

- The output of a perceptron corresponds to the *separation result* of the hyperplane.

We see the activation process of a perceptron as two steps:

1. First, calculate the weighted sum.

$$g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} - \theta$$

- Here, $\mathbf{w}^\top \mathbf{x} - \theta = 0$ denotes a hyperplane in $\mathbb{R}^D$ space.

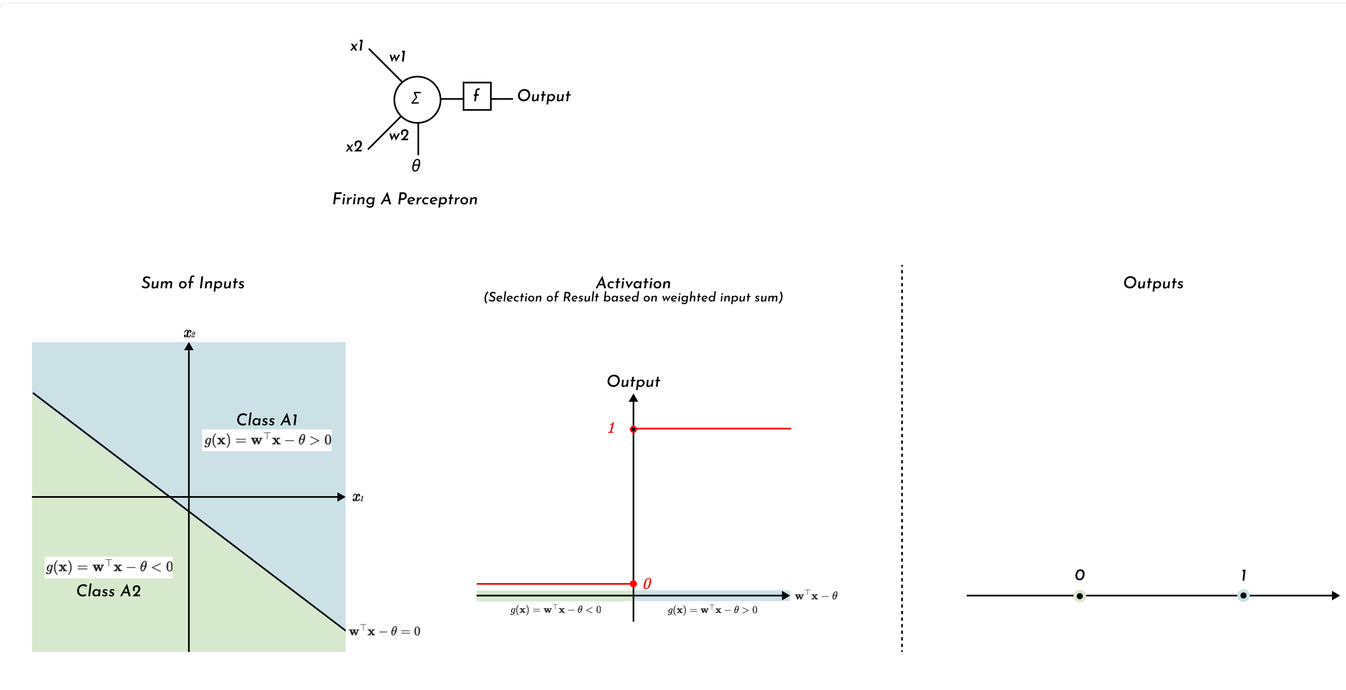
- The sign value of $g(\mathbf{x})$ with given data point of $\mathbf{x}$ is the classification result of the plane.
  - $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} - \theta > 0$, classify to A_1.
  - $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} - \theta < 0$, classify to A_2.
- However, since $g(\mathbf{x}) \in \mathbb{R}$, we need to organize the output according to the rule above.

2. Then, activate the weighted sum.

$$Y = f(g)$$

- To realize the rule mentioned above, the activation function is chosen to be :

$$f(g(\mathbf{x})) = \begin{cases} 0 & g(\mathbf{x}) \le 0 \\ 1 & g(\mathbf{x}) > 0 \end{cases}$$



11.2 Multi-Level Perceptrons

ℹ️ The input data may not be linearly-separable.

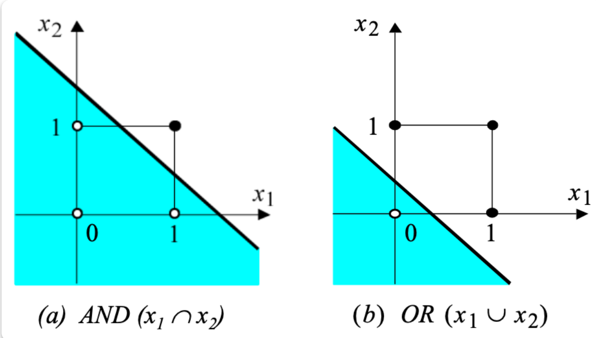
- That is, there's some classification tasks that can not be done by a single perceptron.

11.2.1 XOR Problem: Illustration

We let the inputs:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},\ x_1, x_2 \in \{0, 1\}$$

AND, OR: Can classify



A single hyperplane, i.e., a single perceptron, can classify the AND and OR case.

The AND case:

We regulate the label of $\mathbf{x}$ to be $y_{\mathrm{AND}}(\mathbf{x}) = x_1 \wedge x_2$.
The separating plane is thus:

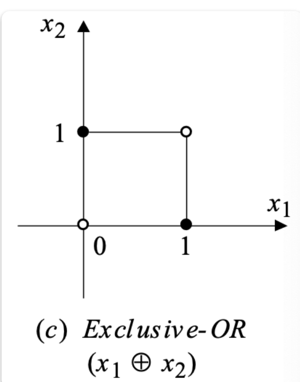
$$x_1 + x_2 - \frac{3}{2} = 0$$

The OR case:

We regulate the label of $\mathbf{x}$ to be $y_{\mathrm{OR}}(\mathbf{x}) = x_1 \vee x_2$.
The separating plane is thus:

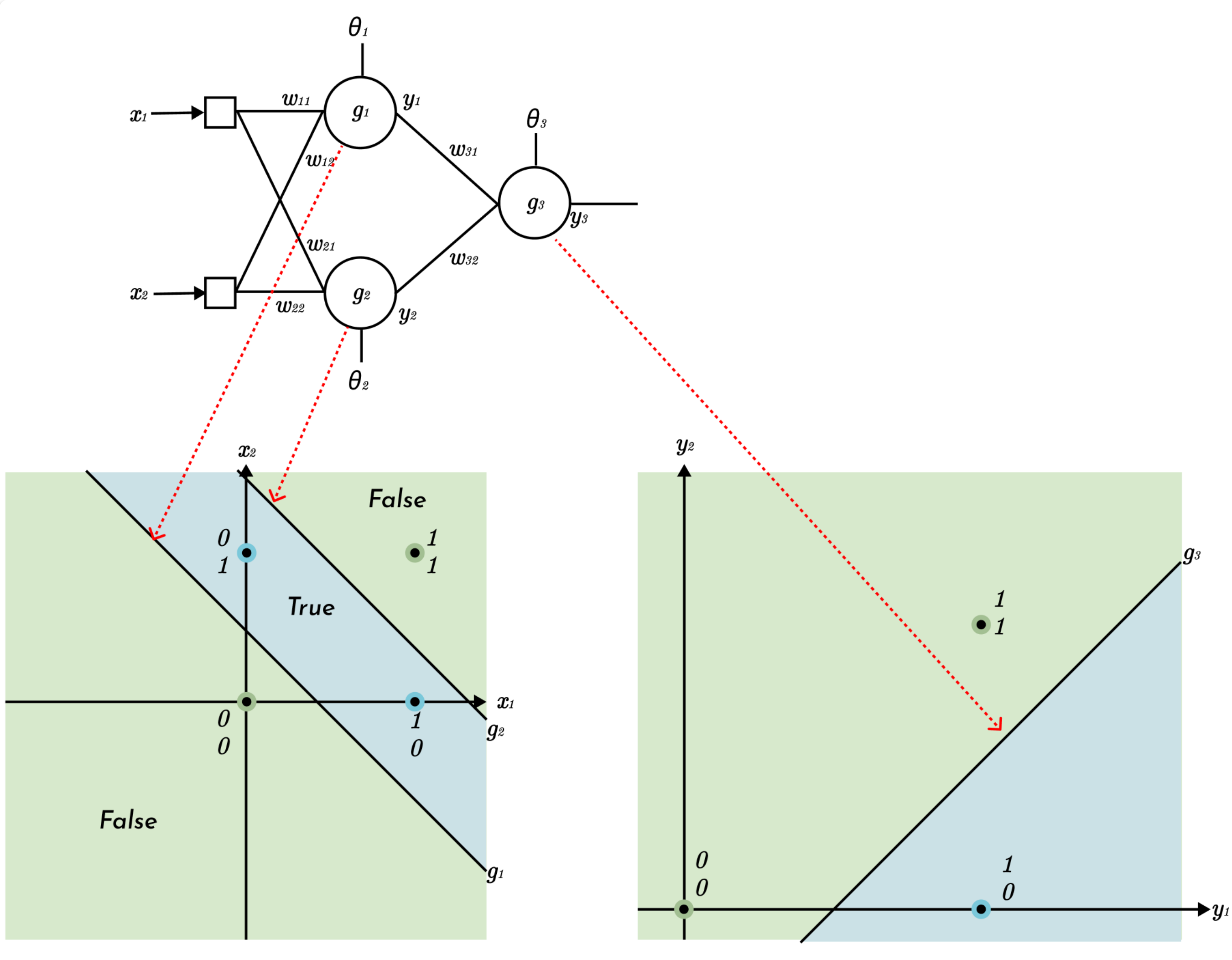
$$x_1 + x_2 - \frac{1}{2} = 0$$

XOR: Can't classify



We regulate the label of $\mathbf{x}$ to be $y_{\mathrm{XOR}}(\mathbf{x}) = x_1 \oplus x_2$.
Then, there doesn't exist such a hyperplane that could separate the points labeled by y_{XOR}.

11.2.2 Multi-Layer Perceptron



To solve the XOR problem, we construct two hyperplanes to classify such non-linear separable case.

$$g_1 : [1 \quad 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \frac{1}{2} = 0$$

$$g_2 : [1 \quad 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \frac{3}{2} = 0$$

Given $\mathbf{x}$, the classification results is decided by:

- $g_1(\mathbf{x}) > 0 \wedge g_2(\mathbf{x}) < 0$: Classify as True;
- $g_1(\mathbf{x}) < 0 \vee g_2(\mathbf{x}) > 0$: Classify as False.

First Layer.

The first layer has two perceptrons, representing the two hyperplanes.

$$g_1 : [1 \quad 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \frac{1}{2} = 0$$

$$g_2 : [1 \quad 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \frac{3}{2} = 0$$

The output of the first two perceptrons denotes the *regions* separated by the two spaces.

- The output y_1:
  - > 0, the point is above the hyperplane g_1.
  - < 0, the point is below the hyperplane g_1.
- The output y_2:
  - > 0, the point is above the hyperplane g_2.
  - < 0, the point is below the hyperplane g_2.

There are 2 hyperplanes defined in the input layer:

- There would be correspondingly 2 outputs y_1, y_2 from the input layer.
- The two output will be the inputs to the second layer.
- The second layer will be 2-d.

The first layer is a **Hidden Layer.**

- What it "hides": The intermediate results.
- Decision Regions that's not yet been mapped into output spaces.

Second Layer.

The two hyperplanes had divided the entire input space into 3 regions:

$$
\begin{aligned}
\text{Region 1: } & g_1(\mathbf{x}) < 0 \wedge g_2(\mathbf{x}) < 0 \\
& \implies y_1 = 0 \wedge y_2 = 0 \\
& \implies \text{label} = \text{False}
\end{aligned}
$$

$$
\begin{aligned}
\text{Region 2: } & g_1(\mathbf{x}) > 0 \wedge g_2(\mathbf{x}) < 0 \\
& \implies y_1 = 1 \wedge y_2 = 0 \\
& \implies \text{label} = \text{True}
\end{aligned}
$$

$$
\begin{aligned}
\text{Region 3: } & g_1(\mathbf{x}) > 0 \wedge g_2(\mathbf{x}) > 0 \\
& \implies y_1 = 1 \wedge y_2 = 1 \\
& \implies \text{label} = \text{False}
\end{aligned}
$$

Therefore, the new space could be separated by another hyperplane g_3:

$$
g_3 : \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \frac{1}{2} = 0
$$

Summary.

In summary, the neural network constructed to solve the XOR problem is:

$$
\begin{aligned}
\text{First Layer: } & g_1 : \mathbf{w}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \theta_1 = \frac{1}{2} \\
& g_2 : \mathbf{w}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \theta_2 = \frac{3}{2} \\
\text{Second Layer: } & g_3 : \mathbf{w}_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \ \theta_3 = \frac{1}{2}
\end{aligned}
$$

11.2 Perceptron Training

11.2.1 Single-Layer Perceptron Training

Error

- During a training iteration t, we have:
  - The desired, ground truth output $Y_d^{(t)}$.
  - The predicted output $Y^{(t)}$

  The error is thus calculated by:

$$e^{(t)} = Y_d^{(t)} - Y^{(t)}$$

ℹ️ Perceptron Learning Rule

$$\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} + \mathbf{x}_i^{(t)} \cdot e^{(t)}$$

Step 1. Initialization.

Step 1.1 Weights & Thresholds

Set initial weights and thresholds:

$$\mathbf{w}^{(0)} = \begin{bmatrix} w_1^{(0)} \\ w_2^{(0)} \\ \vdots \\ w_n^{(0)} \end{bmatrix}$$

$$\theta = \text{random}(-0.5, 0.5)$$

Step 1.2 Hyper-Parameters

Learning rate: η
Activation Function: f
Error metric: e

Step 2. Iteration: For All $\mathbf{x}_t$

Step 2.1 Activation

Activate the perceptron by applying inputs and the weights.

$$Y^{(t)} = f(\mathbf{w}^{(t)}\mathbf{x}^{(t)} - \theta^{(t)})$$

Step 2.2 Weight Training

Calculate the error:

$$e^{(t)} = Y_d^{(t)} - Y^{(t)}$$

Delta Rule that gives the change in weights:

$$\Delta\mathbf{w}^{(t)} = \eta \cdot e^{(t)} \cdot \mathbf{x}^{(t)}$$

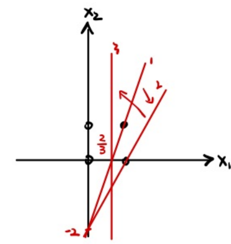
Update the weight of the perceptron:

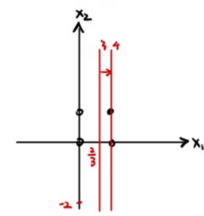
$$
\begin{aligned}
\mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} + \Delta\mathbf{w}^{(t)} \\
&= \mathbf{w}^{(t)} + \eta \cdot e^{(t)} \cdot \mathbf{w}^{(t)} \\
&= \mathbf{w}^{(t)} + \eta \cdot \left(Y_d^{(t)} - Y^{(t)}\right) \cdot \mathbf{w}^{(t)}
\end{aligned}
$$

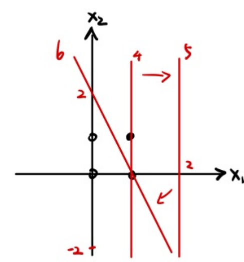
Perceptron Training Algorithm for Learning to Perform the "AND" Operation

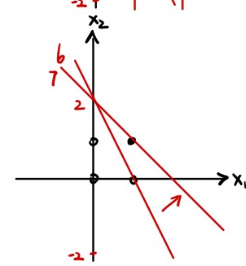
$\theta = 0.2$
$\eta = 0.1$

Epoch	Cur Weights W	X	Y_d	$W^T X - \theta$	Y	e	ΔW	Update
1	$\begin{bmatrix} 0.3 \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	0	-0.2	0	0		
		$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$	0	-0.1	0	0		
		$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	0	0.1	1	-1	$\begin{bmatrix} -0.1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$
	$\begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	1	-0.1	0	1	$\begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.3 \\ 0 \end{bmatrix}$
2	$\begin{bmatrix} 0.3 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	0	-0.2	0	0		
		$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$	0	-0.2	0	0		
		$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	0	0.1	1	-1	$\begin{bmatrix} -0.1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.2 \\ 0 \end{bmatrix}$
	$\begin{bmatrix} 0.2 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	1	0	1	0		
3	$\begin{bmatrix} 0.2 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	0	-0.2	0	0		
		$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$	0	-0.2	0	0		
		$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	0	0	1	-1	$\begin{bmatrix} -0.1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.1 \\ 0 \end{bmatrix}$
	$\begin{bmatrix} 0.1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	1	-0.1	0	1	$\begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}$
4	$\begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	0	-0.2	0	0		
		$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$	0	-0.1	0	0		
		$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	0	0	1	-1	$\begin{bmatrix} -0.1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$
	$\begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	1	0	1	0		
5	$\begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$		-0.2	0	0		
		$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$		-0.1	0	0		
		$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$		-0.1	0	0		
		$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$		0	1	0		



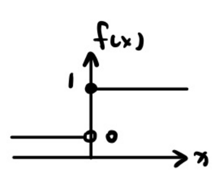






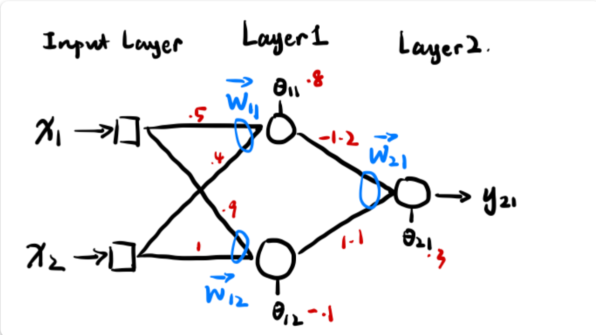
Hyper-Parameters:

- threshold $\theta = 0.2$
- learning rate $\eta = 0.1$
- activation function: $f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$



11.2.2 Multi-Layer Perceptron Training

Suppose that we construct a neural network as follows.



Step 1. Initialization

Step 1.1 Weights and Thresholds

We combine weights and thresholds for a more simplified computation.
For the description of a perceptron below:

$$\text{Weights \& Threshold: } \mathbf{w}_{\text{before}} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \ \theta_{\text{before}}$$

$$\text{Input Format: } \mathbf{x}_{\text{before}} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

We convert it to:

$$\text{Weights \& Threshold: } \mathbf{w}_{\text{after}} = \begin{bmatrix} w_1 \\ w_2 \\ \theta_{\text{before}} \end{bmatrix}$$

$$\text{Input Format: } \mathbf{x}_{\text{after}} = \begin{bmatrix} x_1 \\ x_2 \\ -1 \end{bmatrix},$$

Therefore:

$$\mathbf{w}_{\text{before}}^\top \mathbf{x}_{\text{before}} - \theta_{\text{before}} \equiv \mathbf{w}_{\text{after}}^\top \mathbf{x}_{\text{after}}$$

The weights and thresholds of this network is initialized as:

$$\mathbf{w}_1 = \begin{bmatrix} - & \mathbf{w}_{11}^\top & - \\ - & \mathbf{w}_{12}^\top & - \end{bmatrix}$$

$$= \begin{bmatrix} w_{11,1} & w_{11,2} & \theta_{11} \\ w_{12,1} & w_{12,2} & \theta_{12} \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0.4 & 0.8 \\ 0.9 & 1 & -0.1 \end{bmatrix}$$

$$\mathbf{w}_2 = \begin{bmatrix} - & \mathbf{w}_{21}^\top & - \end{bmatrix}$$

$$= \begin{bmatrix} -1.2 & 1.1 & 0.3 \end{bmatrix}$$

Step 1.2 Hyper-Parameters

- Learning rate: $\eta = 0.1$
- Activation function: $f = \text{sigmoid as } \sigma$

Step 2. Iteration

Take the first input of $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ as an example.

Step 2.1 Forward Propagation

Pass through first layer:

$$g_{11} = \mathbf{w}_{11}^\top \mathbf{x}$$

$$= \begin{bmatrix} 0.5 & 0.4 & 0.8 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

$$= 0.1$$

$$g_{12} = \mathbf{w}_{12}^\top \mathbf{x}$$

$$= \begin{bmatrix} 0.9 & 1.0 & -0.1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

$$= 2.0$$

$$y_{11} = \sigma(g_{11}) = \sigma(0.1) = 0.5250$$

$$y_{12} = \sigma(g_{12}) = \sigma(2.0) = 0.8808$$

Pass through second layer:

$$
\begin{aligned}
g_{21} &= \mathbf{w}_{21}^{\top} \begin{bmatrix} y_{11} \\ y_{12} \\ -1 \end{bmatrix} \\
&= \begin{bmatrix} -1.2 & 1.1 & 0.3 \end{bmatrix} \begin{bmatrix} 0.5250 \\ 0.8808 \\ -1 \end{bmatrix} \\
&= 0.03888
\end{aligned}
$$

$$
y_{21} = \sigma(g_{21}) = \sigma(0.03888) = 0.5097
$$

Here, $y_{21} = 0.5097$ is the final output of the neural network.

Step 2.2 Back Propagation

Total Error:

$$
\begin{aligned}
\varepsilon &= Y_d - y_{21} \\
&= 0 - 0.5097 \\
&= -0.5097
\end{aligned}
$$

Error gradient of the second layer:

$$
\begin{aligned}
\delta_{21} &= y'_{21} \cdot \varepsilon \\
&= y_{21}(1 - y_{21}) \cdot \varepsilon \\
&= 0.5097 \cdot (1 - 0.5097) \cdot (-0.5097) \\
&= -0.1274
\end{aligned}
$$

Error gradients of the first layer:

$$
\begin{aligned}
\delta_{11} &= \delta_{21} \cdot y'_{11} \cdot w_{21,1} \\
&= \delta_{21} \cdot y_{11} \cdot (1 - y_{11}) \cdot w_{21,1} \\
&= (-0.1274) \times 0.5250 \times (1 - 0.5250) \times (-1.2) \\
&= 0.0381
\end{aligned}
$$

$$
\begin{aligned}
\delta_{12} &= \delta_{21} \cdot y'_{12} \cdot w_{21,2} \\
&= \delta_{21} \cdot y_{21} \cdot (1 - y_{12}) \cdot w_{21,2} \\
&= (-0.1274) \times 0.8808 \times (1 - 0.8808) \times (1.1) \\
&= -0.015
\end{aligned}
$$

Step 2.3 Update Weights

Update weights in Layer 1:

$$\text{Changes in weights:}$$

$$\begin{aligned}
\Delta \mathbf{w}_{11} &= \eta \cdot \delta_{11} \cdot \mathbf{x} \\
&= 0.1 \times 0.0381 \times \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \\
&= \begin{bmatrix} 0.00381 \\ 0.00381 \\ -0.00381 \end{bmatrix} \\
\\
\Delta \mathbf{w}_{12} &= \eta \cdot \delta_{12} \cdot \mathbf{x} \\
&= 0.1 \times -(0.015) \times \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \\
&= \begin{bmatrix} -0.0015 \\ -0.0015 \\ 0.0015 \end{bmatrix}
\end{aligned}$$

$$\text{Apply Changes:}$$

$$\begin{aligned}
\mathbf{w}_{11}^{\text{new}} &= \mathbf{w}_{11} + \Delta \mathbf{w}_{11} \\
&= \begin{bmatrix} 0.5 \\ 0.4 \\ 0.8 \end{bmatrix} + \begin{bmatrix} 0.00381 \\ 0.00381 \\ -0.00381 \end{bmatrix} \\
&= \begin{bmatrix} 0.50381 \\ 0.40381 \\ -0.08381 \end{bmatrix} \\
\mathbf{w}_{12}^{\text{new}} &= \mathbf{w}_{12} + \Delta \mathbf{w}_{12} \\
&= \begin{bmatrix} 0.9 \\ 1.0 \\ -0.1 \end{bmatrix} + \begin{bmatrix} -0.0015 \\ -0.0015 \\ -0.0015 \end{bmatrix} \\
&= \begin{bmatrix} 0.8985 \\ 0.9985 \\ -0.1015 \end{bmatrix}
\end{aligned}$$

Update weights in Layer 2:

Changes in weights:

$$\Delta \mathbf{w}_{21} = \eta \cdot \delta_{21} \cdot \begin{bmatrix} y_{11} \\ y_{12} \\ -1 \end{bmatrix}$$

$$= 0.1 \times (-0.1274) \times \begin{bmatrix} 0.5250 \\ 0.8808 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 6.6885 \times 10^{-3} \\ -0.0112 \\ -0.01274 \end{bmatrix}$$

Apply Changes:

$$\mathbf{w}_{21}^{(new)} = \mathbf{w}_{21} + \Delta \mathbf{w}_{21}$$

$$= \begin{bmatrix} -1.2 \\ 1.1 \\ 0.3 \end{bmatrix} + \begin{bmatrix} -6.6885 \times 10^{-3} \\ 0.0112 \\ 0.01274 \end{bmatrix}$$

$$= \begin{bmatrix} -1.206885 \\ 1.11122 \\ 0.31274 \end{bmatrix}$$