

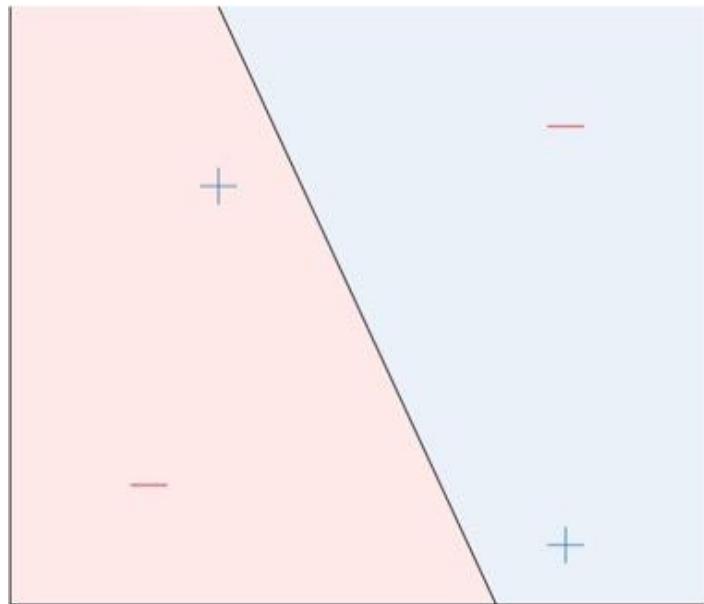
# 10-301/601: Introduction to Machine Learning Lecture 13 – Neural Networks

Henry Chai

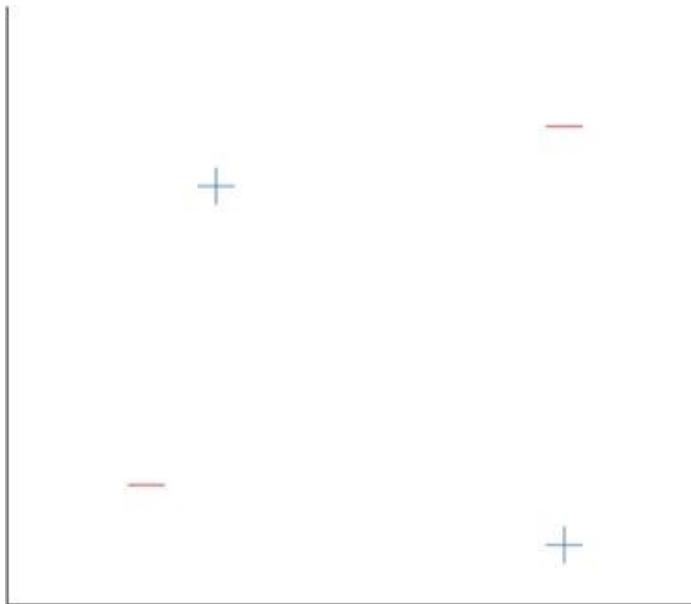
5/21/25

# Biological Neural Network



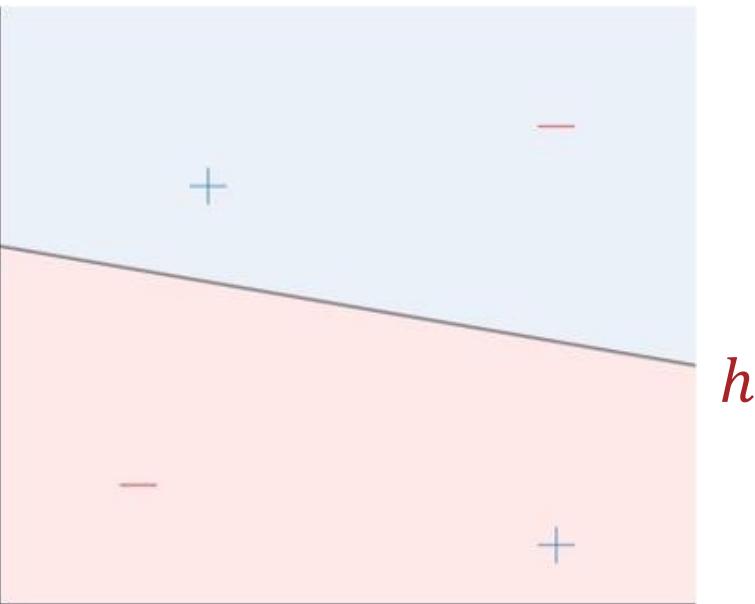


$h_1$

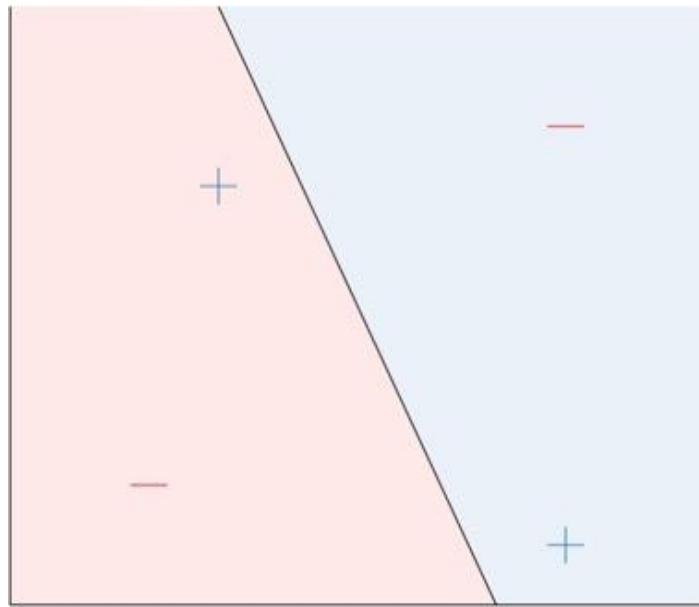


# Perceptrons

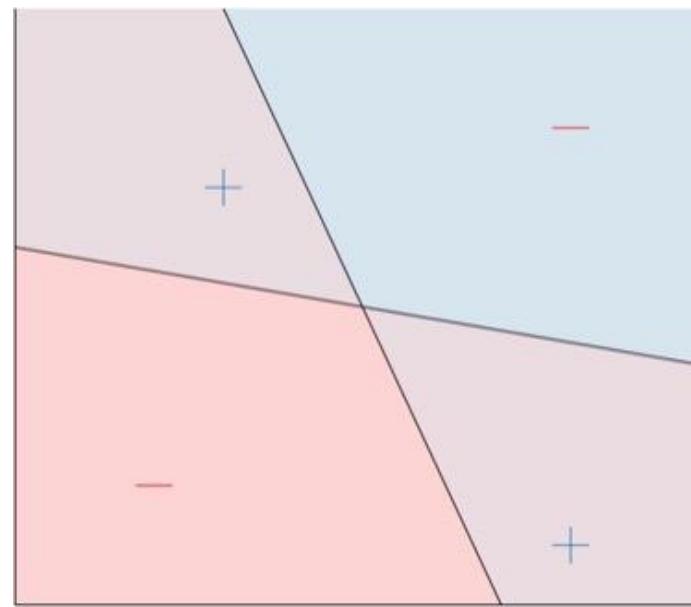
- Linear model for classification
- $h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$
- Predictions are  $+1$  or  $-1$



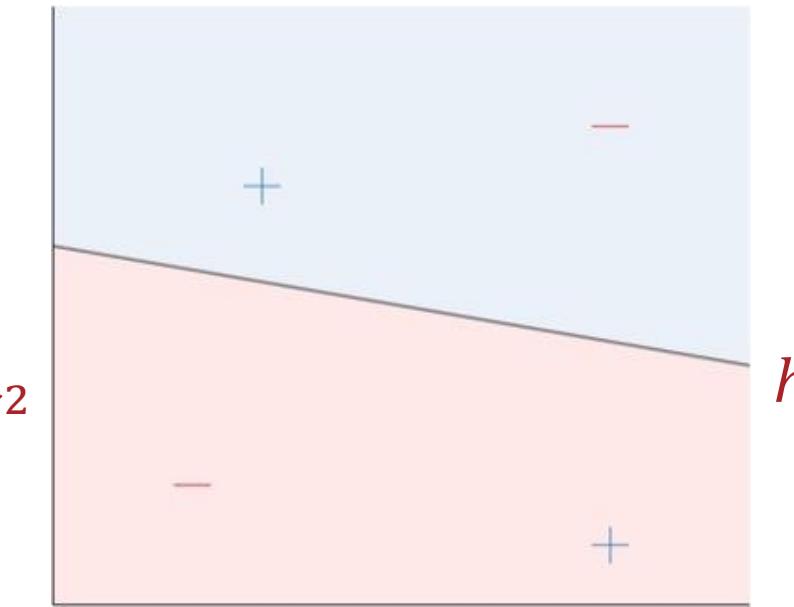
$h_2$



$h_1$

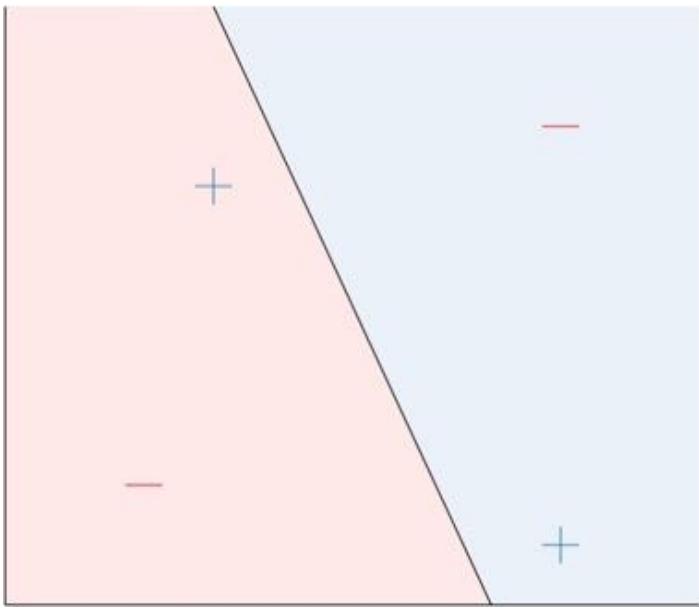


$h_1$

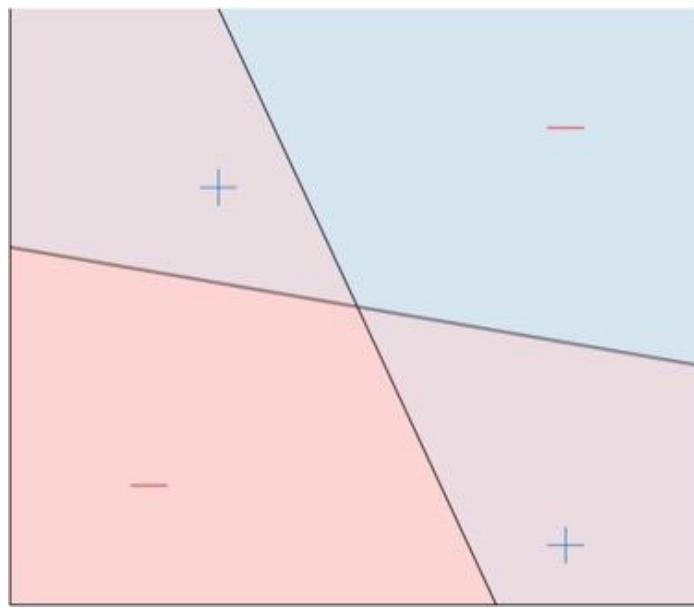


$h_2$

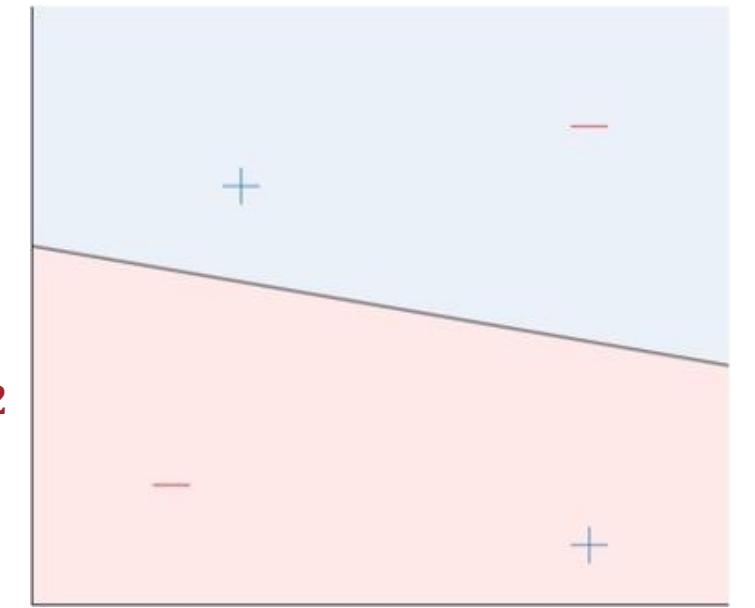
# Combining Perceptrons



$h_1$



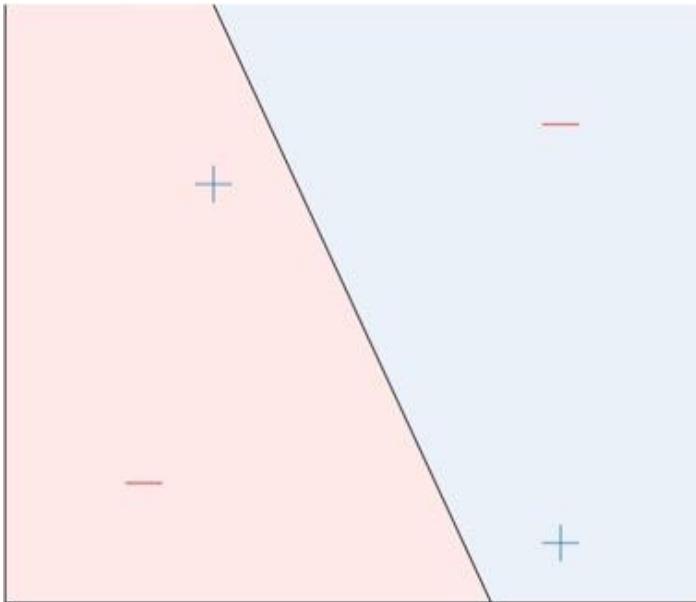
$h_1$



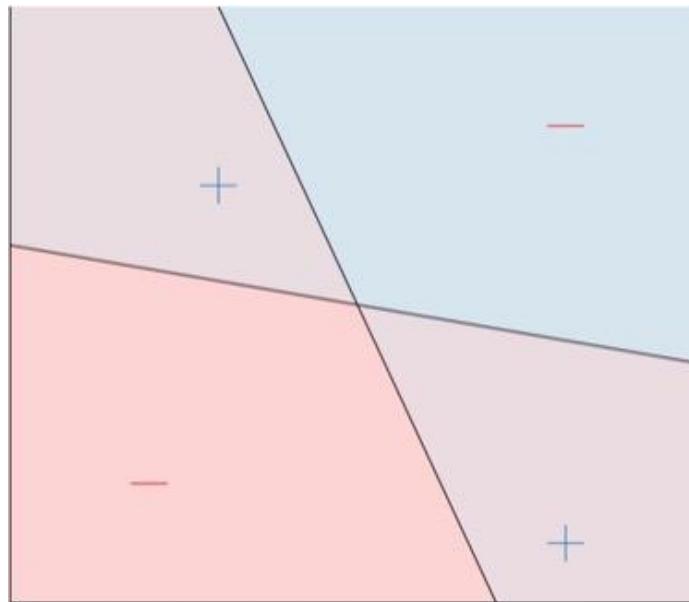
$h_2$

$h_2$

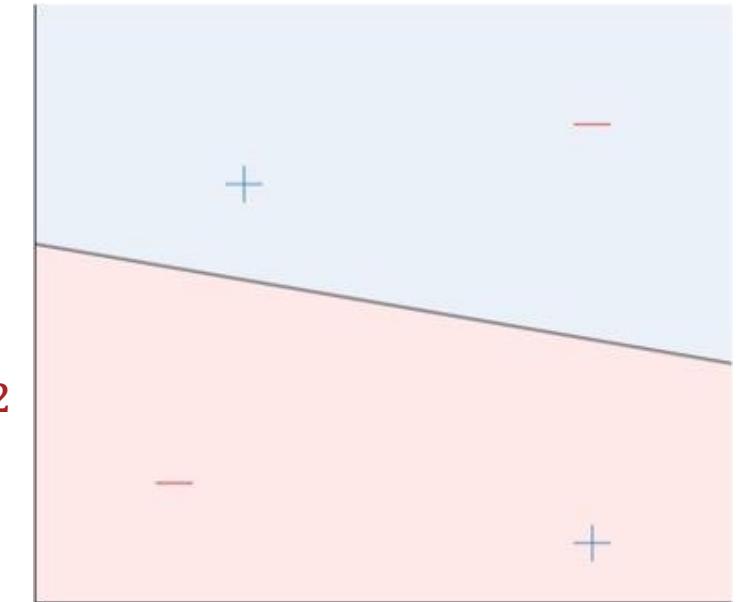
$$h(\mathbf{x}) = \begin{cases} +1 & \text{if } (h_1(\mathbf{x}) = +1 \text{ and } h_2(\mathbf{x}) = -1) \text{ or } (h_1(\mathbf{x}) = -1 \text{ and } h_2(\mathbf{x}) = +1) \\ -1 & \text{otherwise} \end{cases}$$



$h_1$



$h_1$



$h_2$

$h_2$

$$h(\mathbf{x}) = OR \left( AND(h_1(\mathbf{x}), \neg h_2(\mathbf{x})), AND(\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$

# Boolean Algebra

- Boolean variables are either  $+1$  ("true") or  $-1$  ("false")
- Basic Boolean operations
  - Negation:  $\neg z = -1 * z$
- And:  $AND(z_1, z_2) = \begin{cases} +1 & \text{if both } z_1 \text{ and } z_2 \text{ equal } +1 \\ -1 & \text{otherwise} \end{cases}$
- Or:  $OR(z_1, z_2) = \begin{cases} +1 & \text{if either } z_1 \text{ or } z_2 \text{ equals } +1 \\ -1 & \text{otherwise} \end{cases}$

# Boolean Algebra

- Boolean variables are either  $+1$  ("true") or  $-1$  ("false")
- Basic Boolean operations
  - Negation:  $\neg z = -1 * z$
  - And:  $AND(z_1, z_2) = \text{sign}(z_1 + z_2 - 1.5)$
  - Or:  $OR(z_1, z_2) = \text{sign}(z_1 + z_2 + 1.5)$

# Boolean Algebra

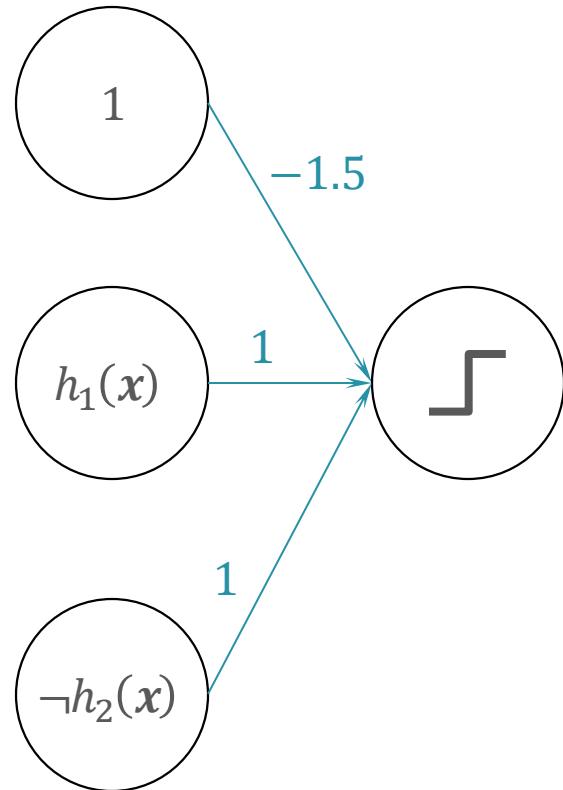
- Boolean variables are either  $+1$  ("true") or  $-1$  ("false")
- Basic Boolean operations
  - Negation:  $\neg z = -1 * z$
- And:  $AND(z_1, z_2) = \text{sign} \left( [-1.5, 1, 1] \begin{bmatrix} 1 \\ z_1 \\ z_2 \end{bmatrix} \right)$
- Or:  $OR(z_1, z_2) = \text{sign} \left( [1.5, 1, 1] \begin{bmatrix} 1 \\ z_1 \\ z_2 \end{bmatrix} \right)$

# Building a Network

$$h(\mathbf{x}) = OR \left( AND(h_1(\mathbf{x}), \neg h_2(\mathbf{x})), AND(\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$

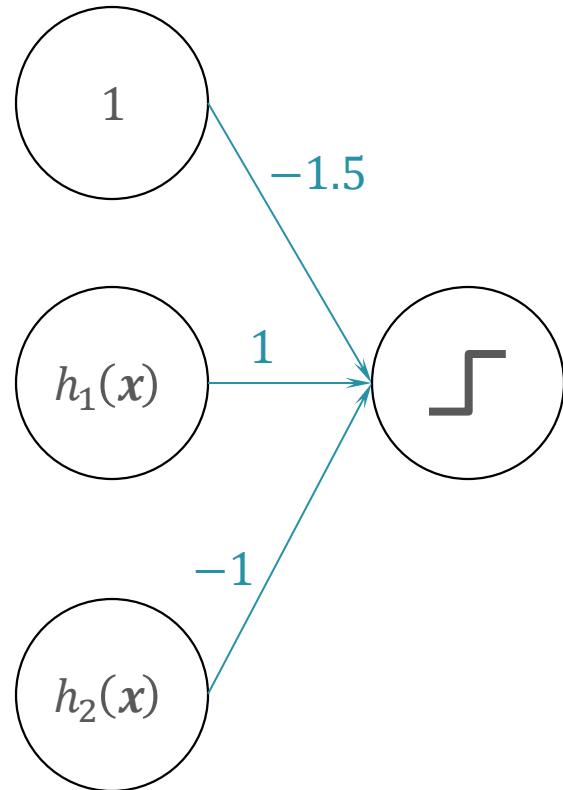
# Building a Network

$$h(\mathbf{x}) = OR \left( AND(h_1(\mathbf{x}), \neg h_2(\mathbf{x})), AND(\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$



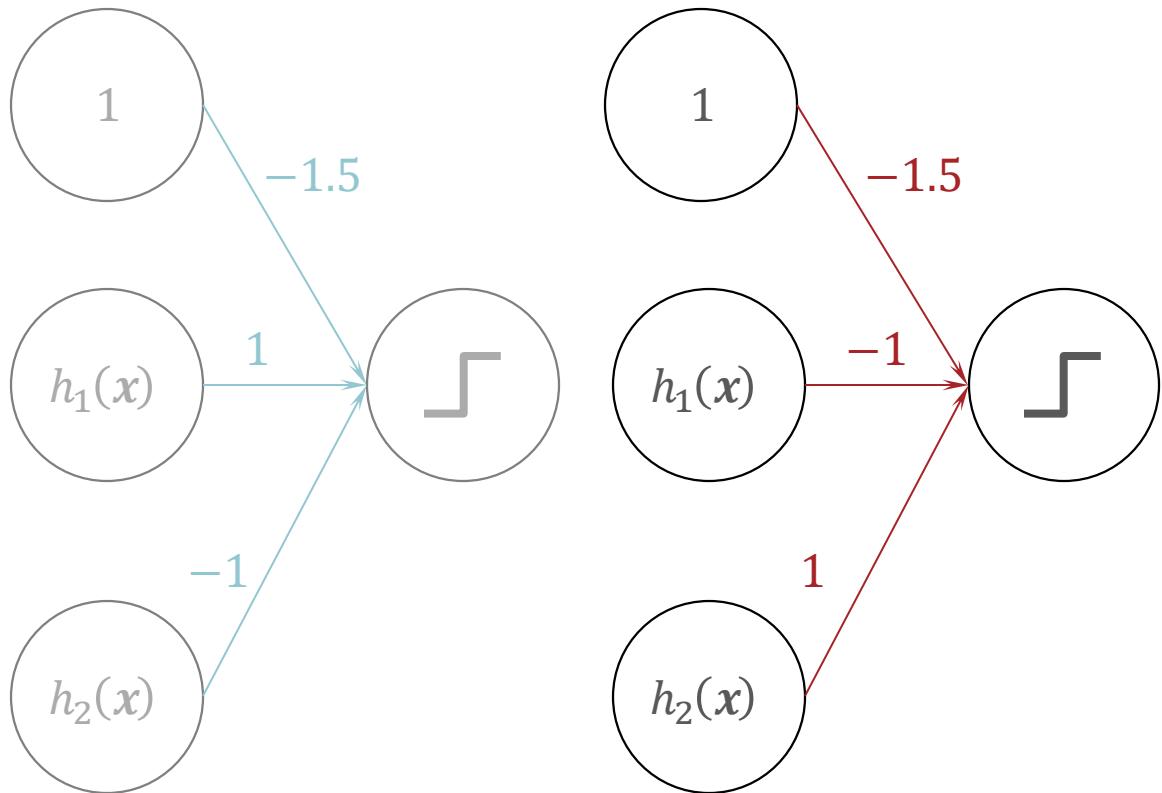
# Building a Network

$$h(\mathbf{x}) = OR \left( AND(h_1(\mathbf{x}), \neg h_2(\mathbf{x})), AND(\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$



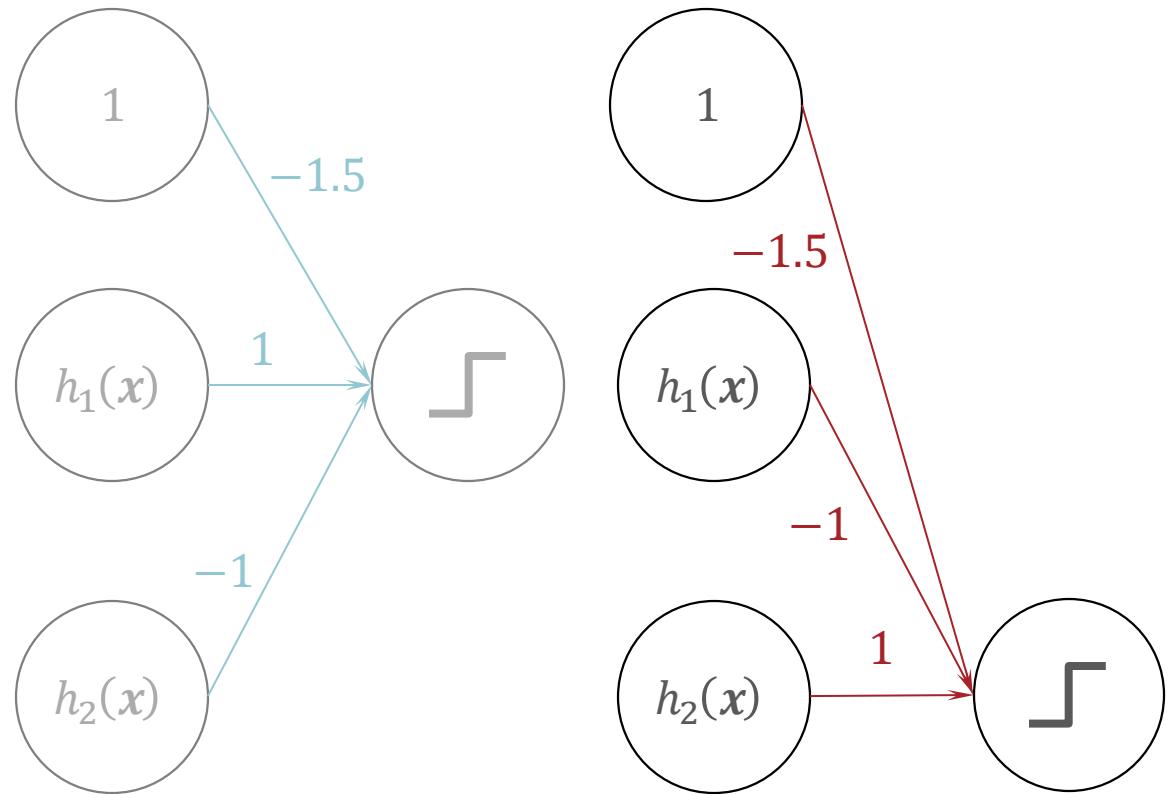
# Building a Network

$$h(x) = OR \left( AND(h_1(x), \neg h_2(x)), AND(\neg h_1(x), h_2(x)) \right)$$



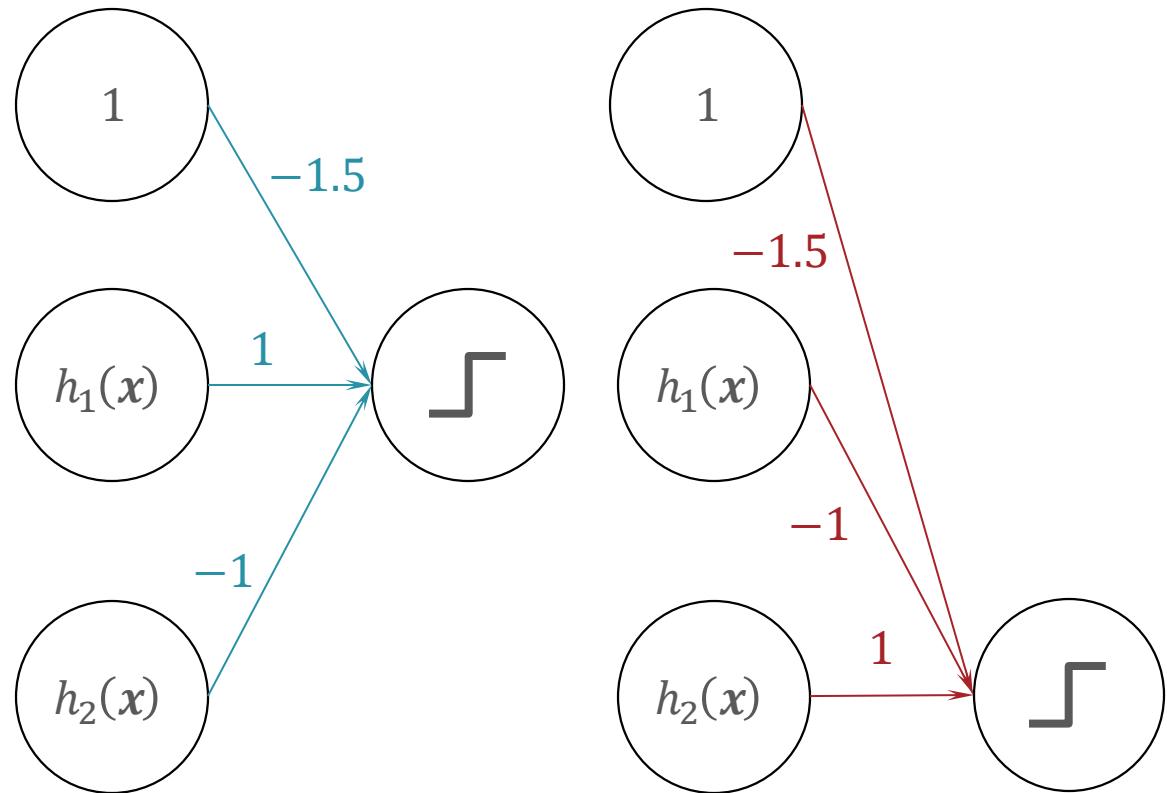
# Building a Network

$$h(x) = OR \left( AND(h_1(x), \neg h_2(x)), AND(\neg h_1(x), h_2(x)) \right)$$

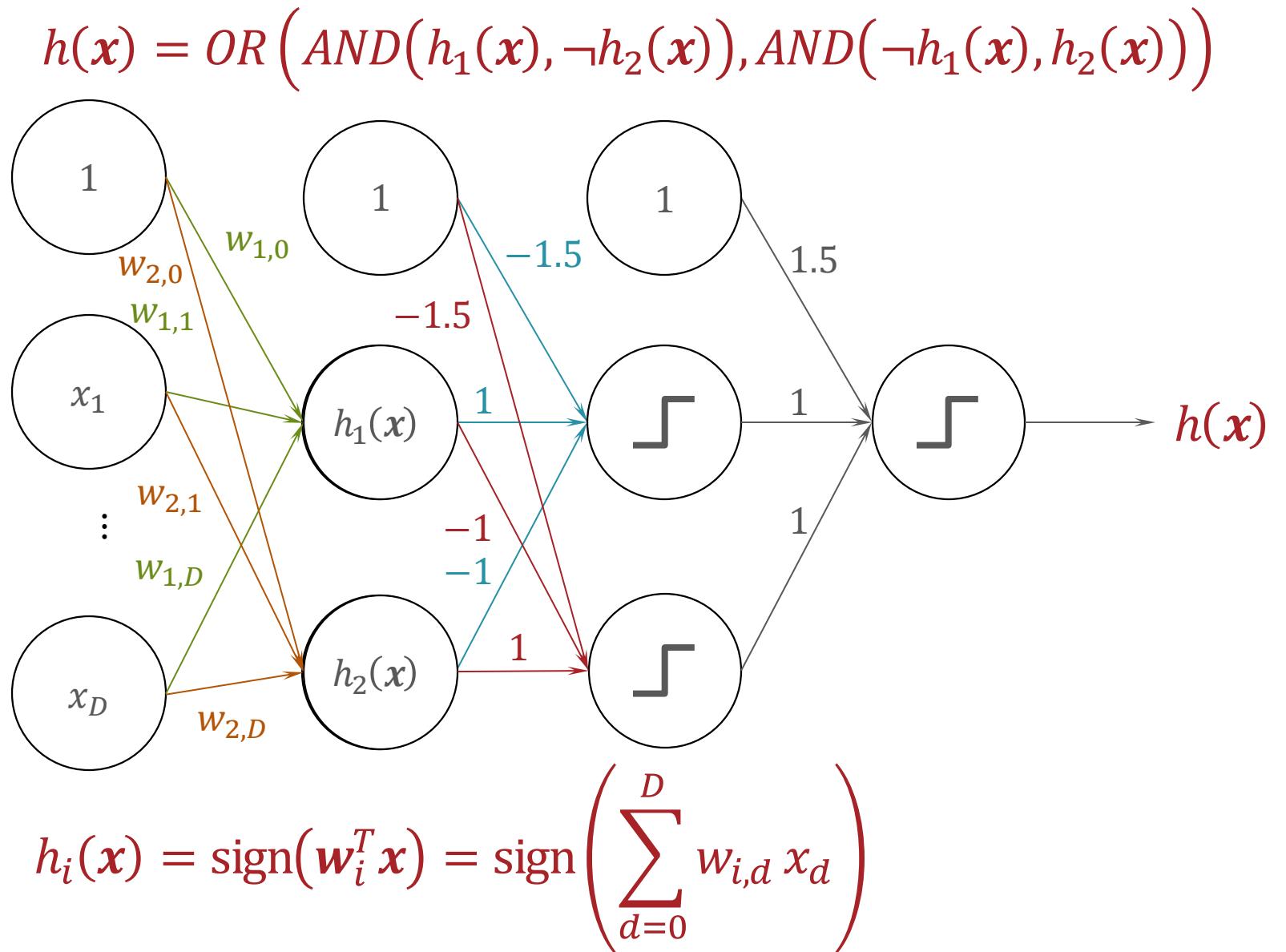


# Building a Network

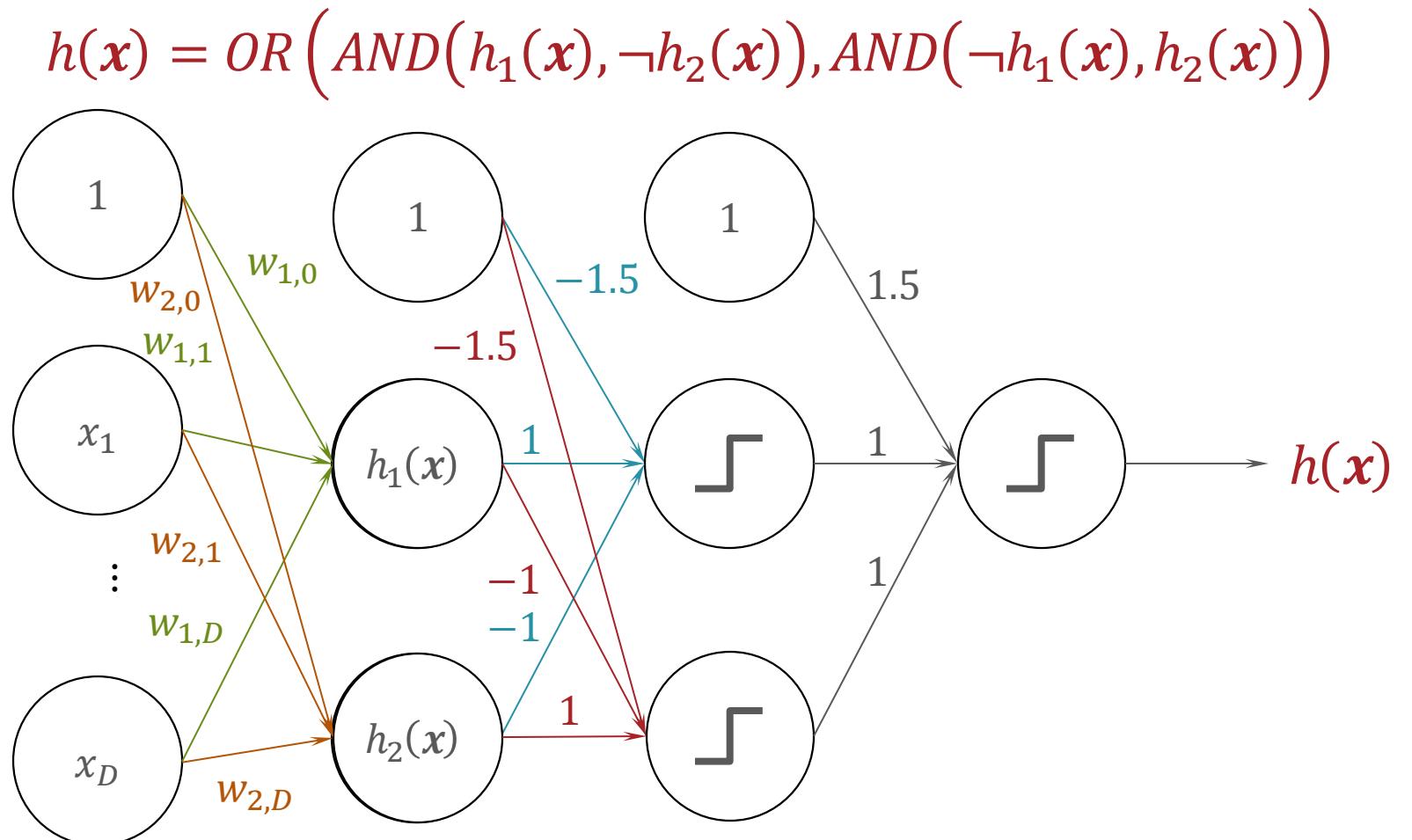
$$h(x) = OR \left( AND(h_1(x), \neg h_2(x)), AND(\neg h_1(x), h_2(x)) \right)$$



# Building a Network

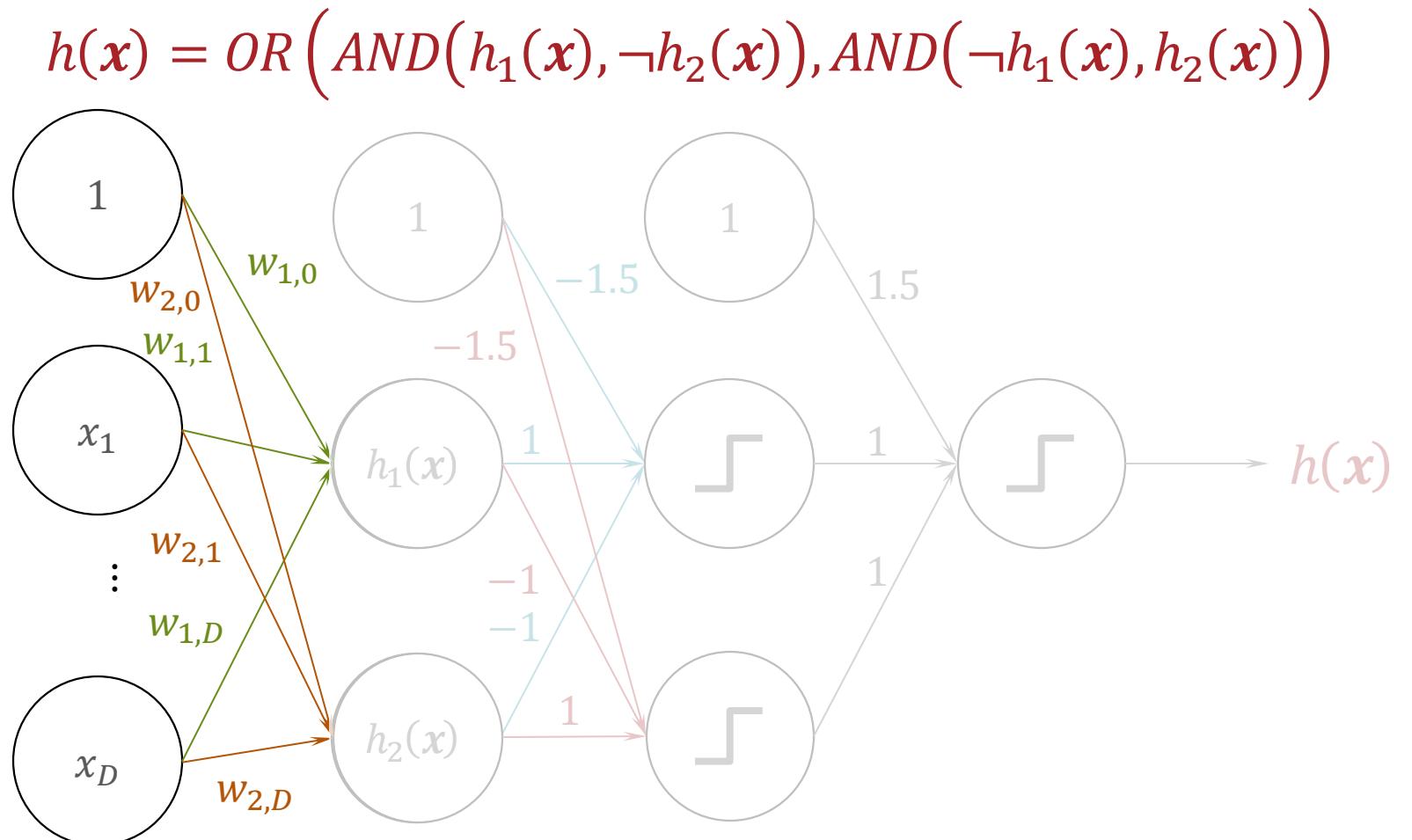


# Building a Network



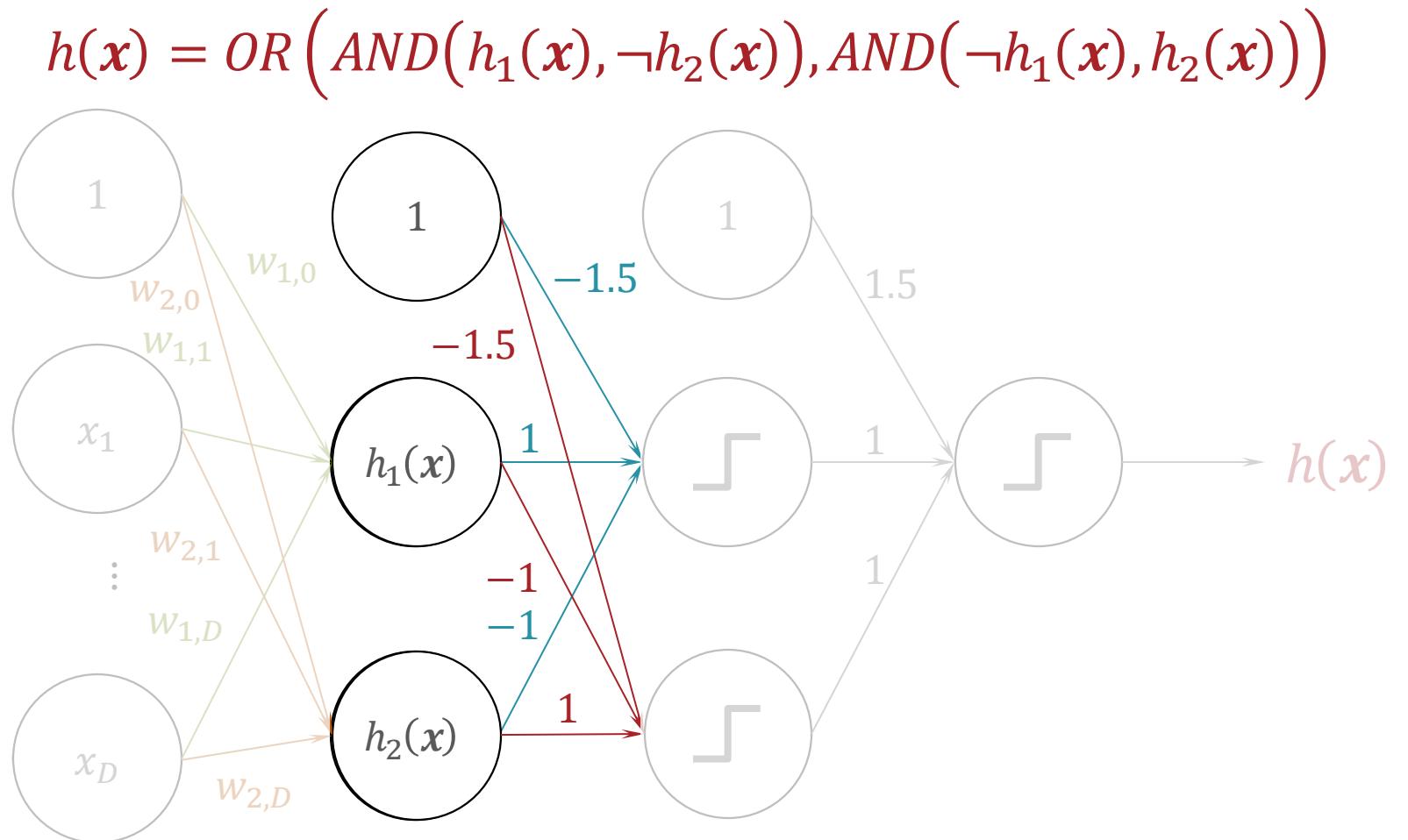
$$h(\mathbf{x}) = \text{sign}(\text{sign}(\text{sign}(\mathbf{w}_1^T \mathbf{x}) - \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + \text{sign}(-\text{sign}(\mathbf{w}_1^T \mathbf{x}) + \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + 1.5)$$

# Building a Network



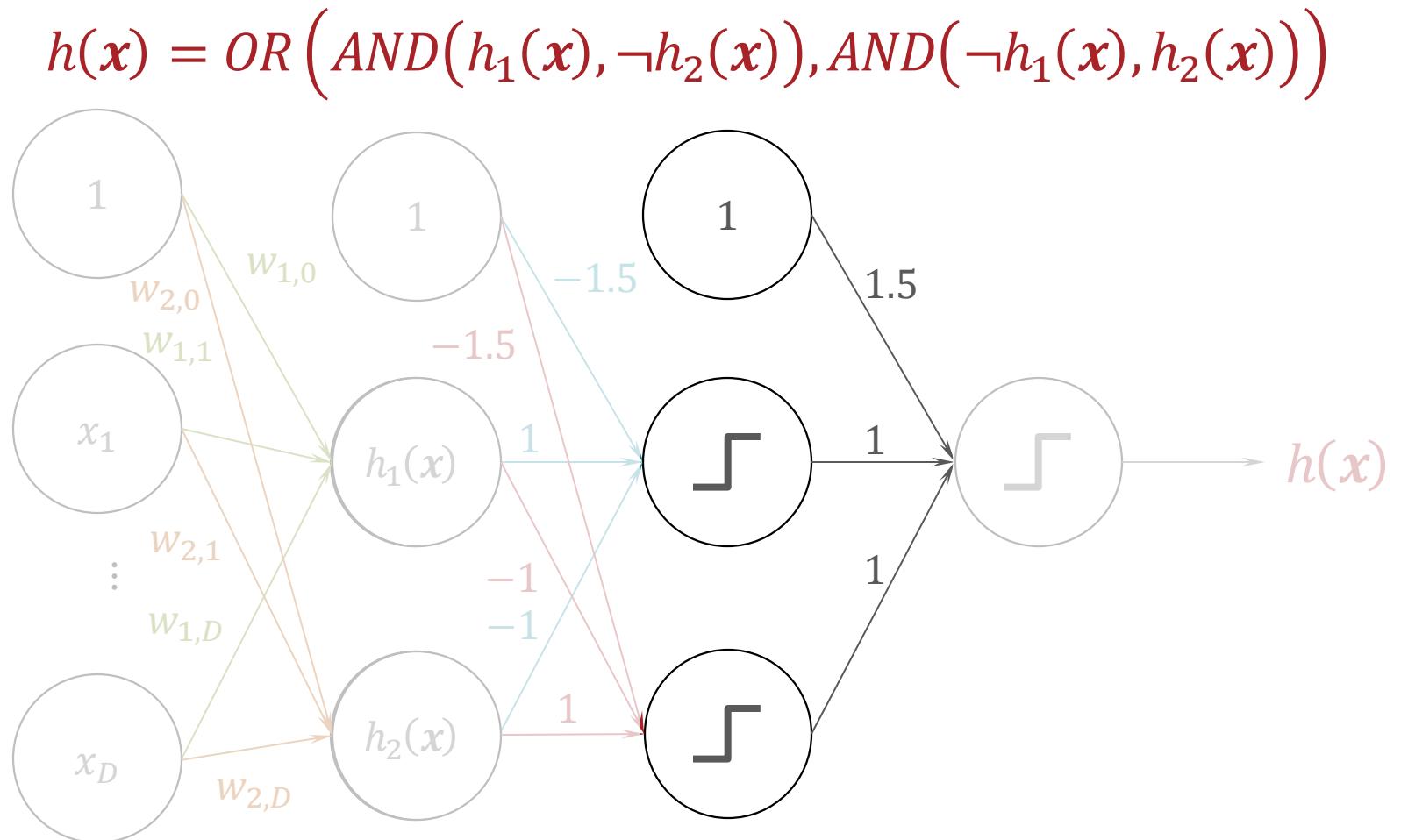
$$h(\mathbf{x}) = \text{sign}(\text{sign}(\text{sign}(\mathbf{w}_1^T \mathbf{x}) - \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + \text{sign}(-\text{sign}(\mathbf{w}_1^T \mathbf{x}) + \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + 1.5)$$

# Building a Network



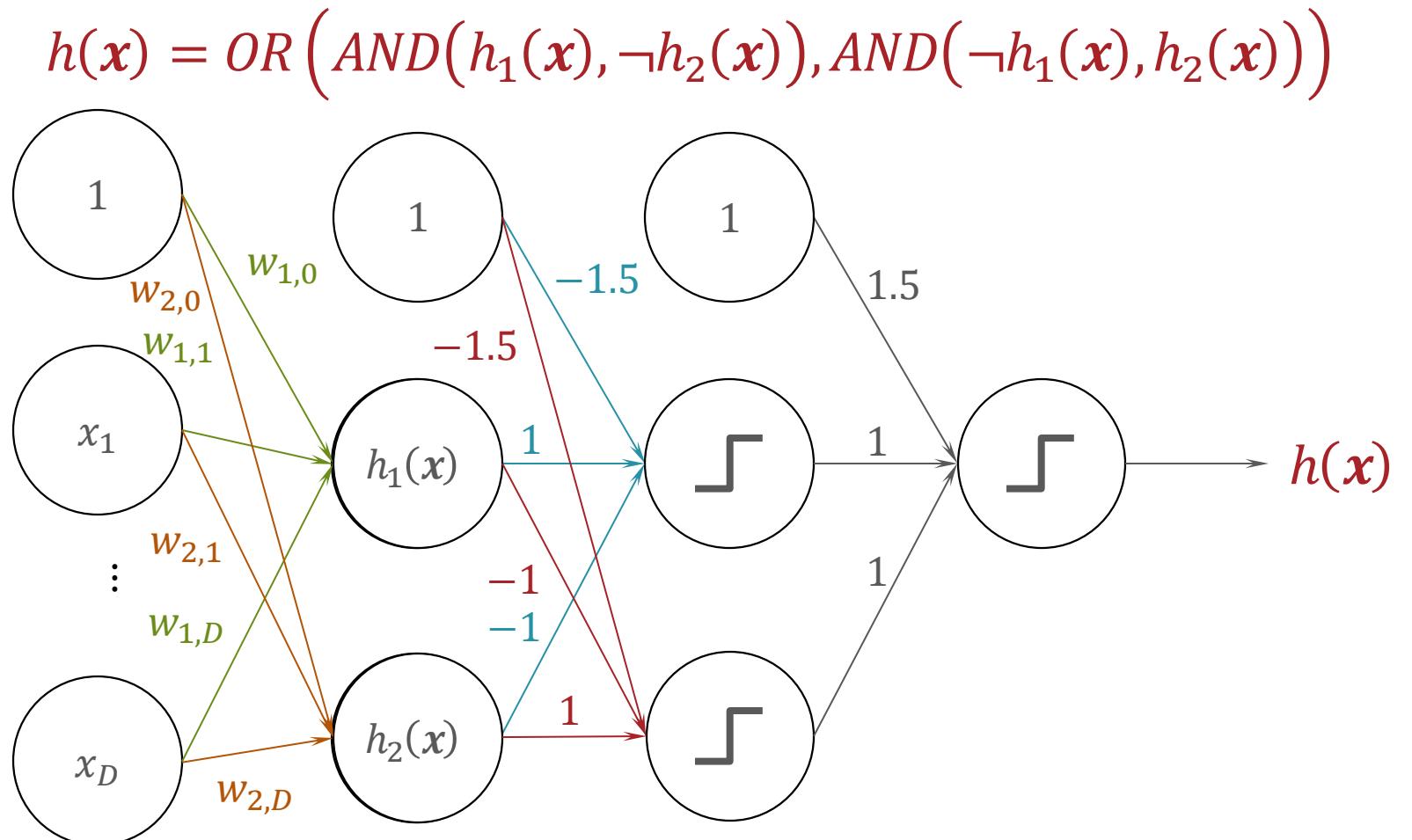
$$h(\mathbf{x}) = \text{sign}(\text{sign}(\text{sign}(\mathbf{w}_1^T \mathbf{x}) - \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + \text{sign}(-\text{sign}(\mathbf{w}_1^T \mathbf{x}) + \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + 1.5)$$

# Building a Network



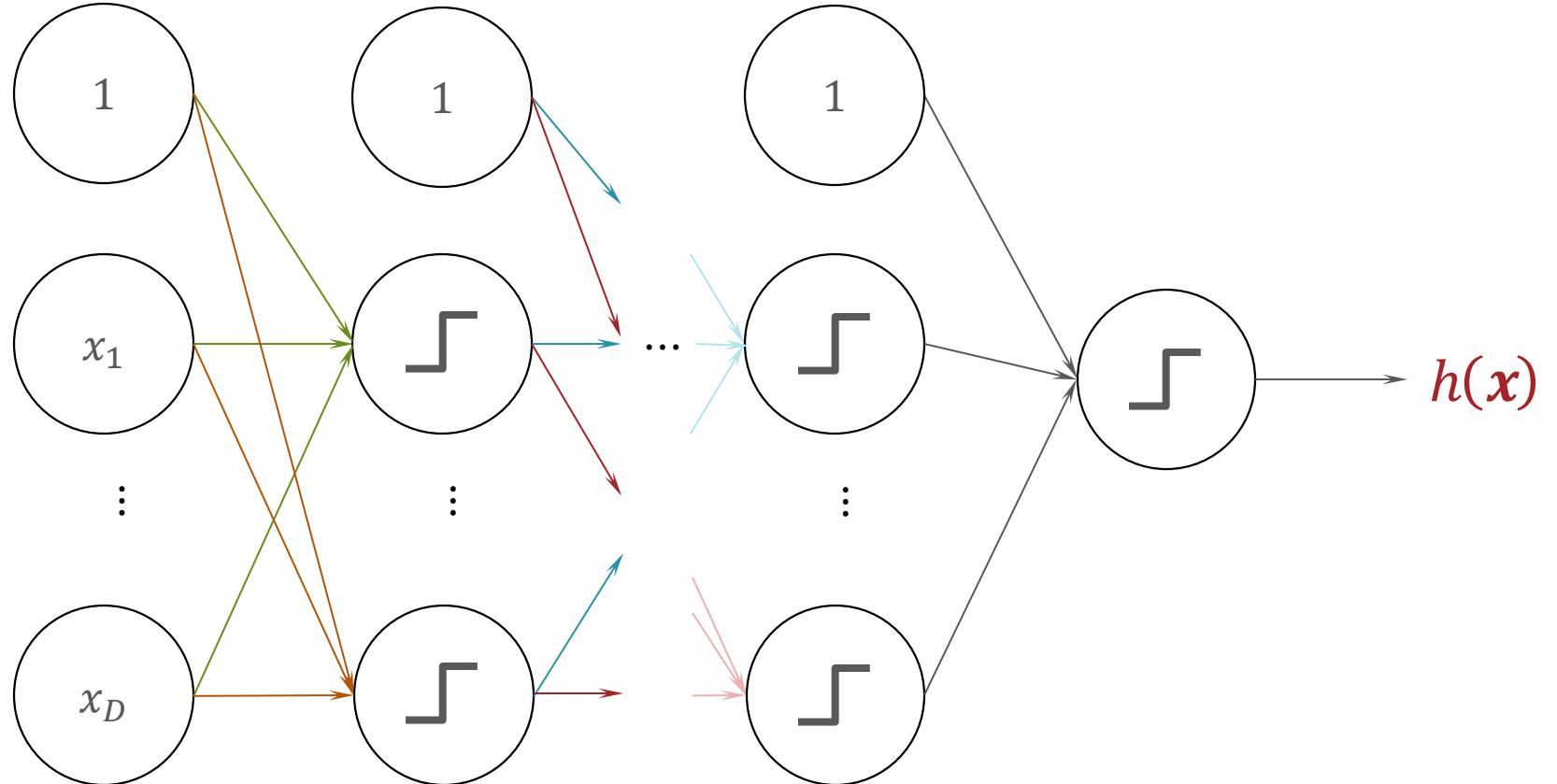
$$h(\mathbf{x}) = \text{sign}(\text{sign}(\text{sign}(\mathbf{w}_1^T \mathbf{x}) - \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + \text{sign}(-\text{sign}(\mathbf{w}_1^T \mathbf{x}) + \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + 1.5)$$

# Building a Network

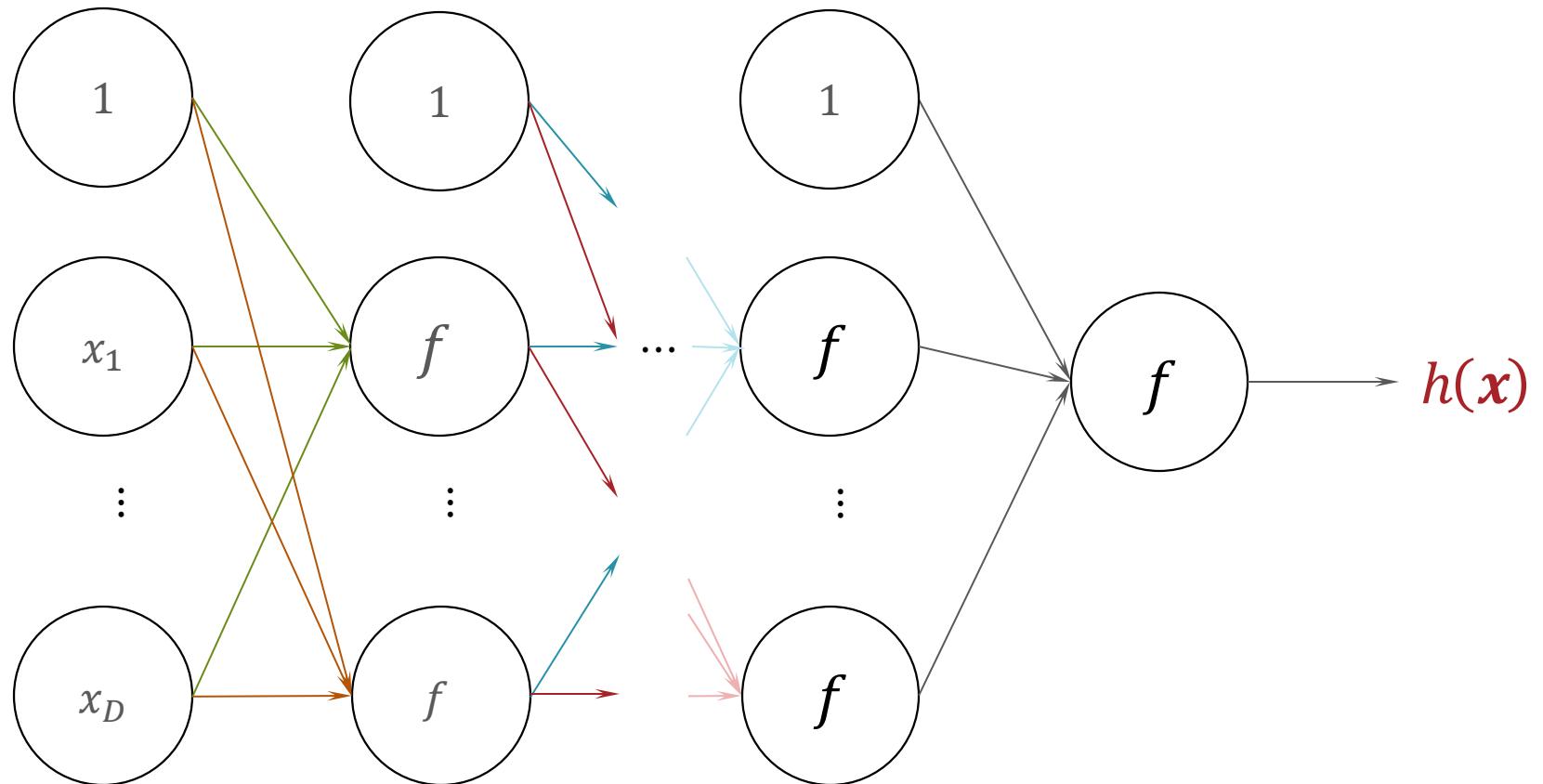


$$h(\mathbf{x}) = \text{sign}(\text{sign}(\text{sign}(\mathbf{w}_1^T \mathbf{x}) - \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + \text{sign}(-\text{sign}(\mathbf{w}_1^T \mathbf{x}) + \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + 1.5)$$

# Multi-Layer Perceptron (MLP)



# (Fully- Connected) Feed Forward Neural Network

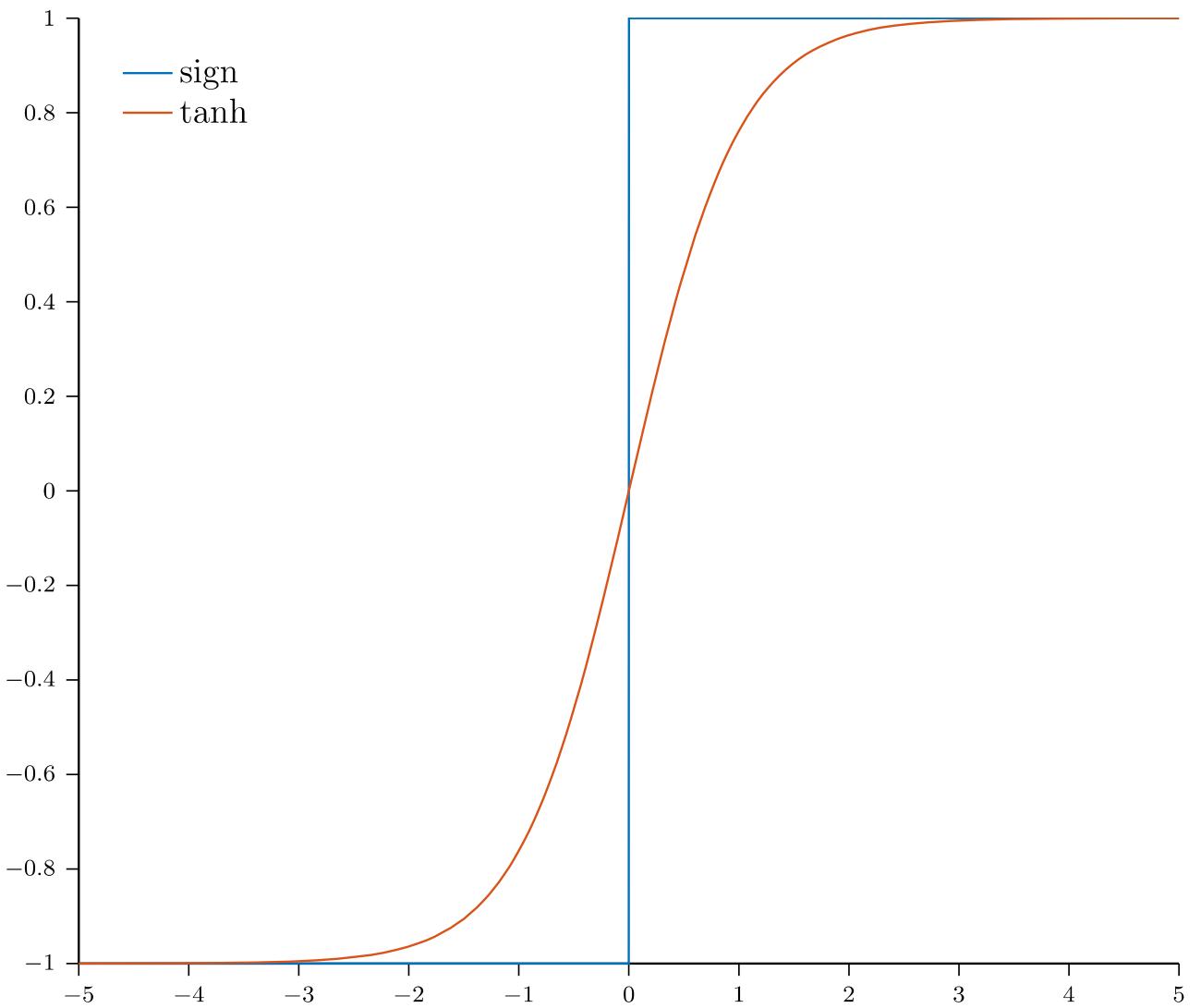


$f(\cdot)$

- Hyperbolic tangent:

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- $\frac{\partial \tanh(z)}{\partial z} = 1 - \tanh(z)^2$



# Other Activation Functions

Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1 + e^{-x}}$
Hyperbolic tangent ( $\tanh$ )		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU) <sup>[7]</sup>		$\begin{cases} 0 & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases} = \max\{0, x\} = x \mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU) <sup>[4]</sup>		$\frac{1}{2}x \left( 1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right) \right) = x\Phi(x)$
Softplus <sup>[8]</sup>		$\ln(1 + e^x)$
Exponential linear unit (ELU) <sup>[9]</sup>		$\begin{cases} \alpha(e^x - 1) & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter $\alpha$
Leaky rectified linear unit (Leaky ReLU) <sup>[11]</sup>		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \ge 0 \end{cases}$
Parametric rectified linear unit (PReLU) <sup>[12]</sup>		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \ge 0 \end{cases}$ with parameter $\alpha$

True or False: both the linear regression model and the logistic regression model can be expressed as neural networks.

True for both

0%

True for linear regression, false for logistic regression

0%

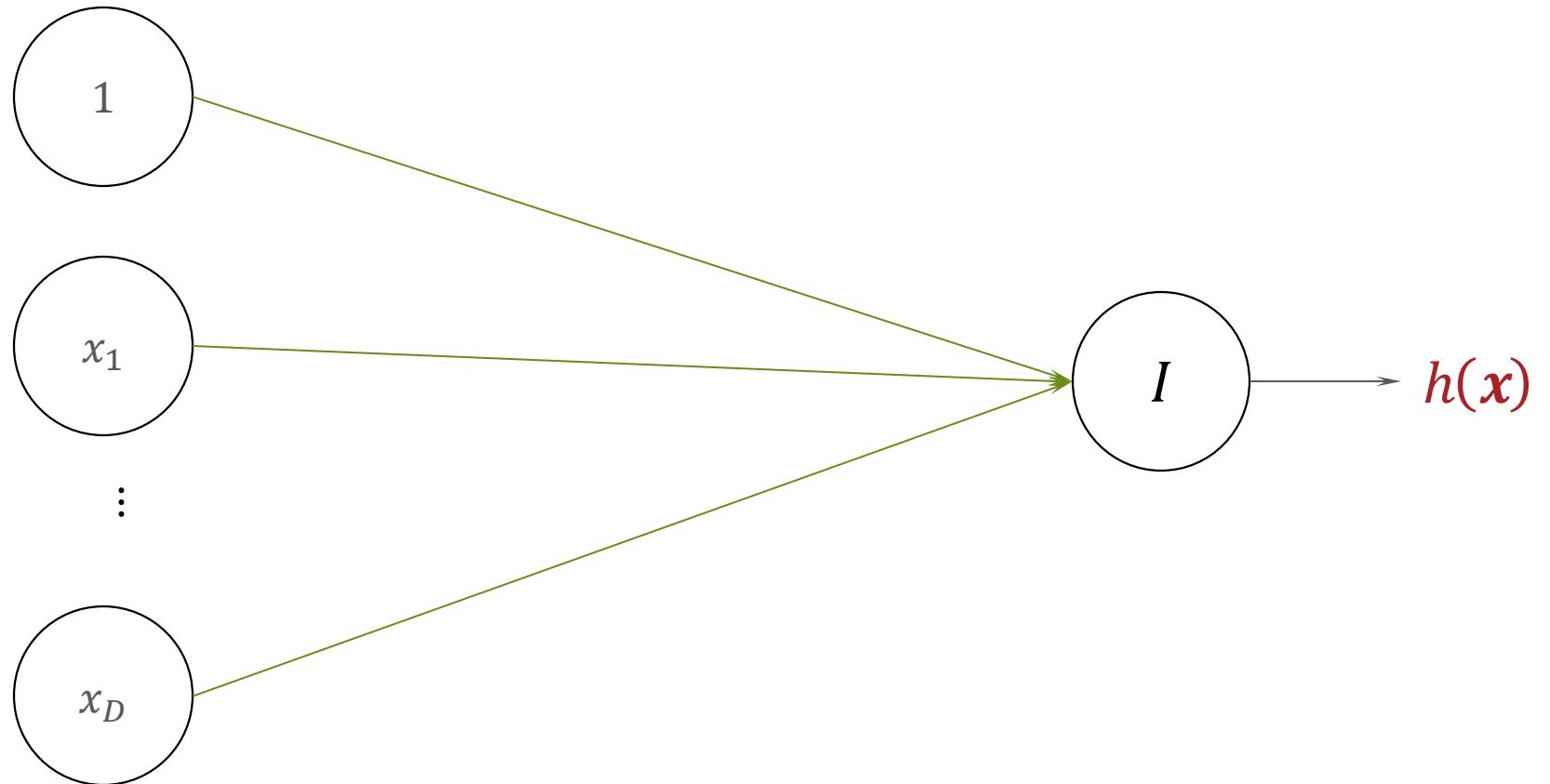
False for linear regression, true for logistic regression

0%

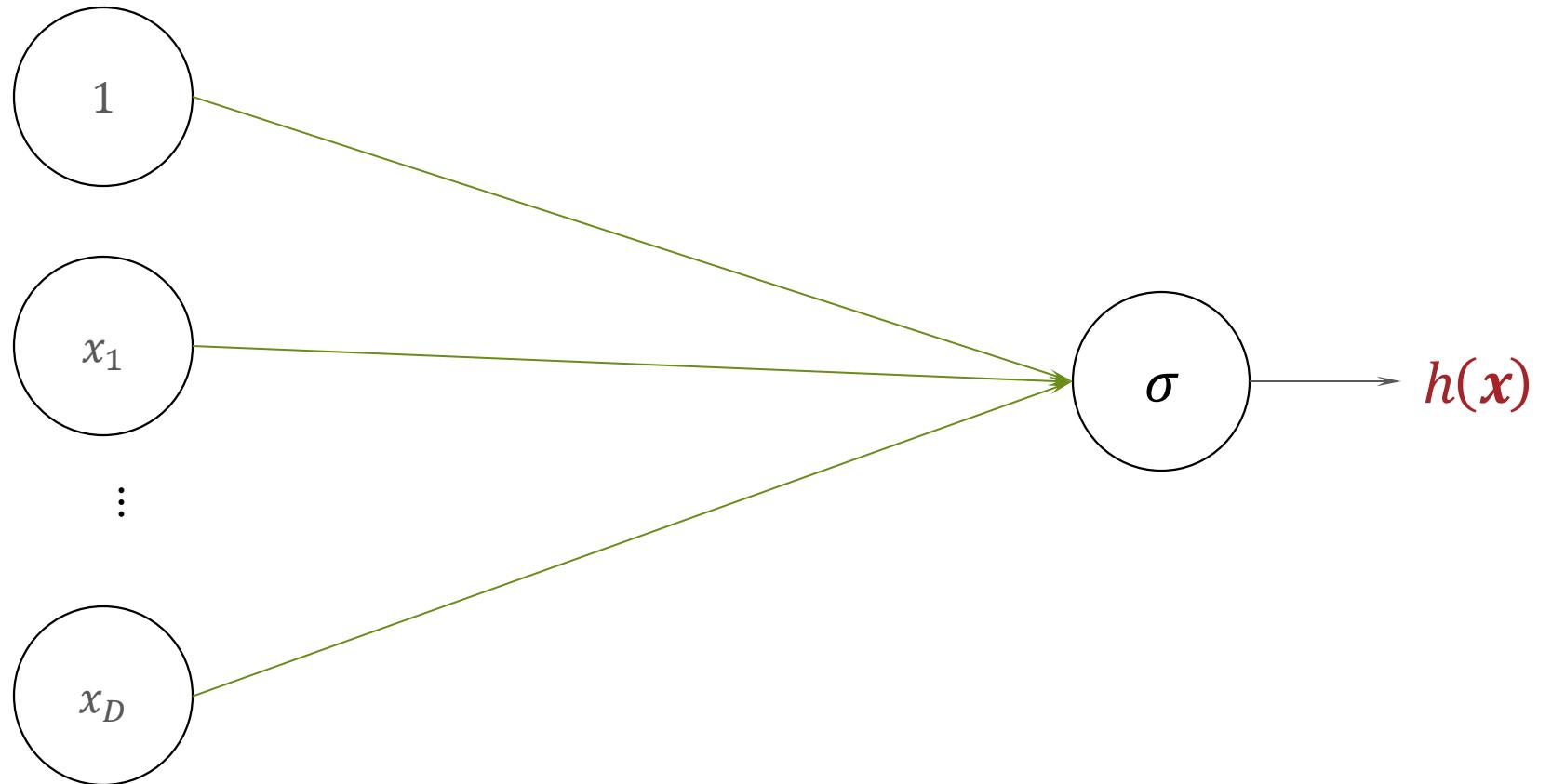
False for both

0%

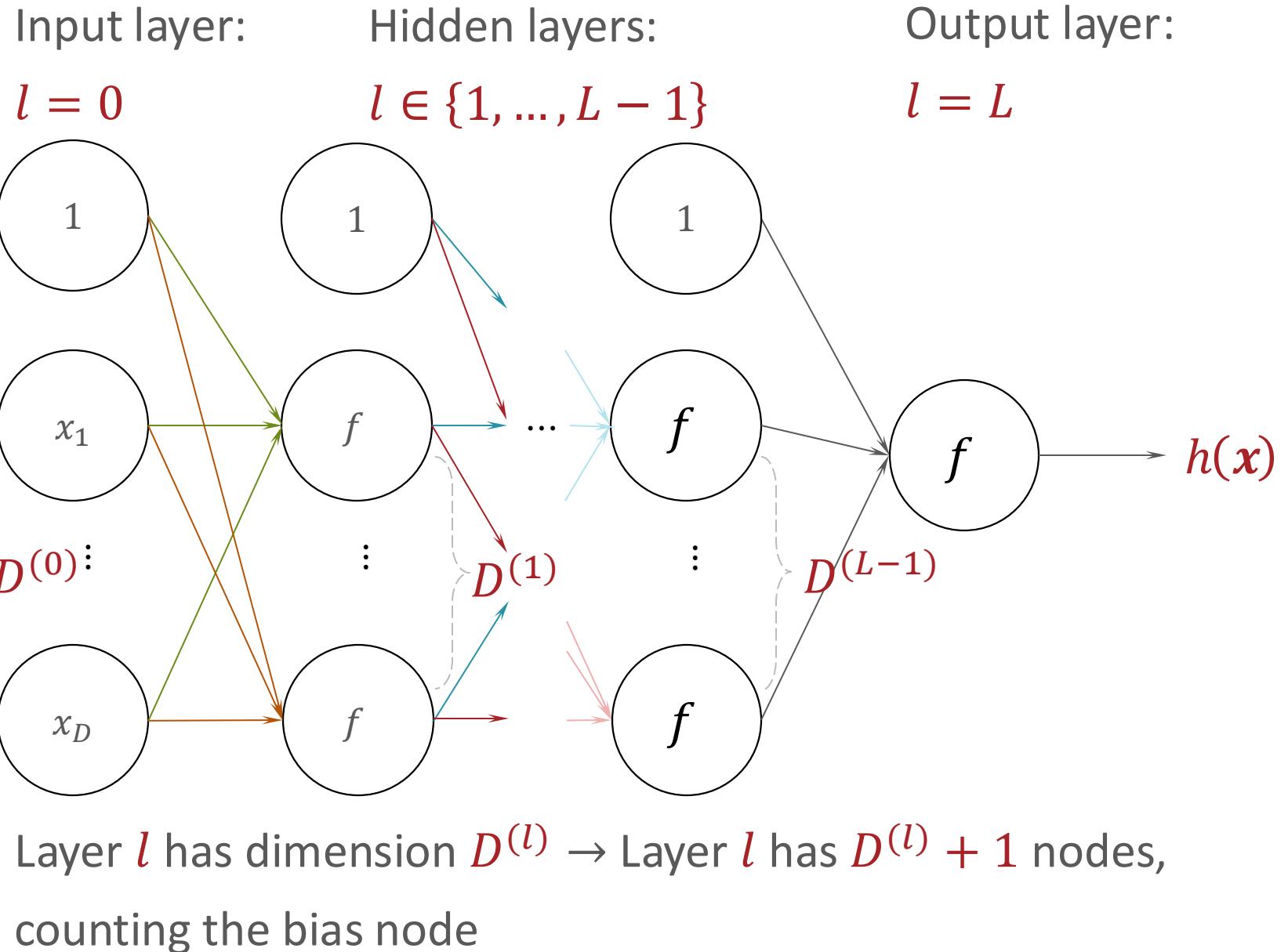
# Linear Regression as a Neural Network



# Logistic Regression as a Neural Network



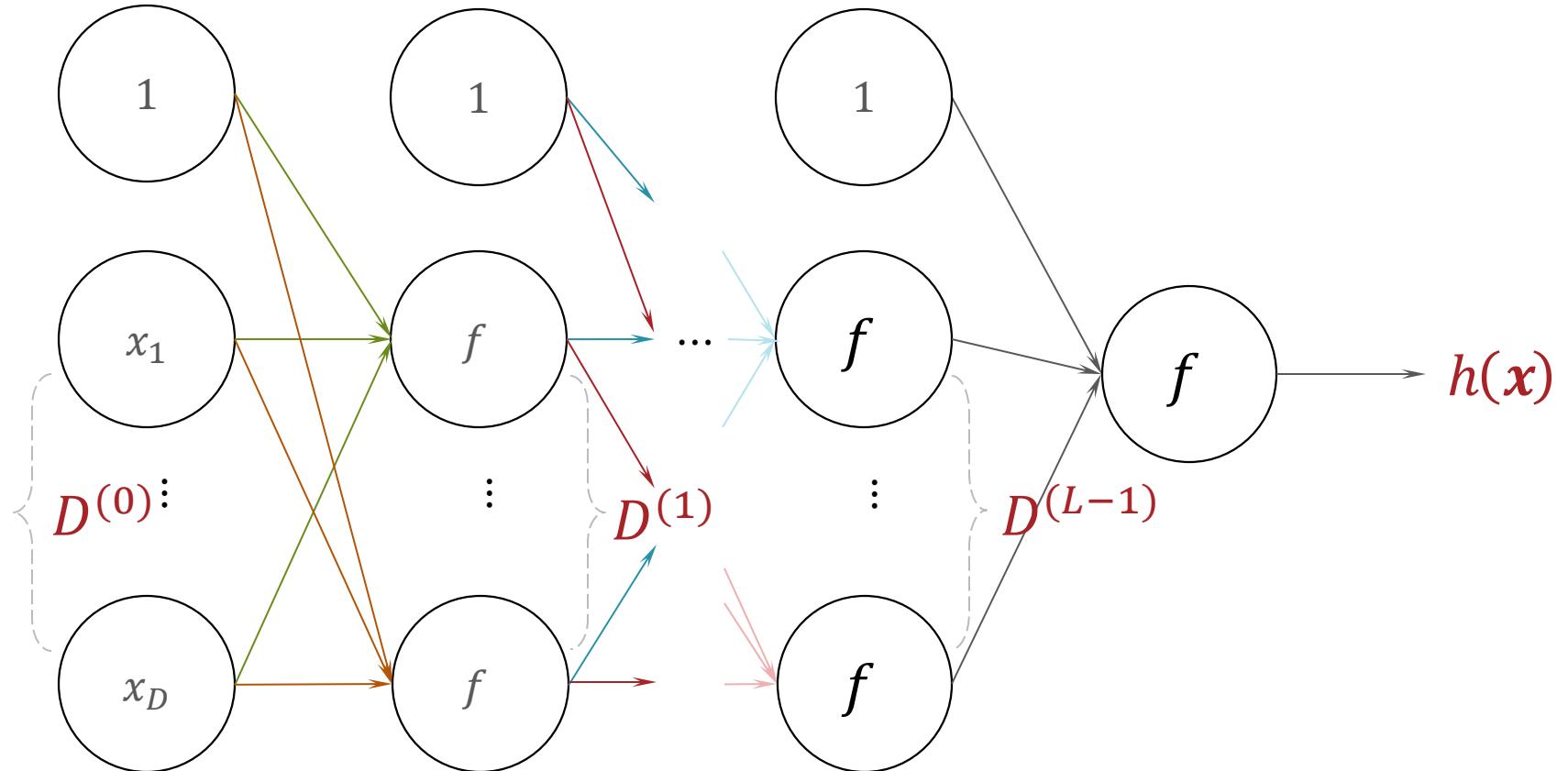
# (Fully- Connected) Feed Forward Neural Network



# (Fully- Connected) Feed Forward Neural Network

The weights between layer  $l - 1$  and layer  $l$  are a matrix:

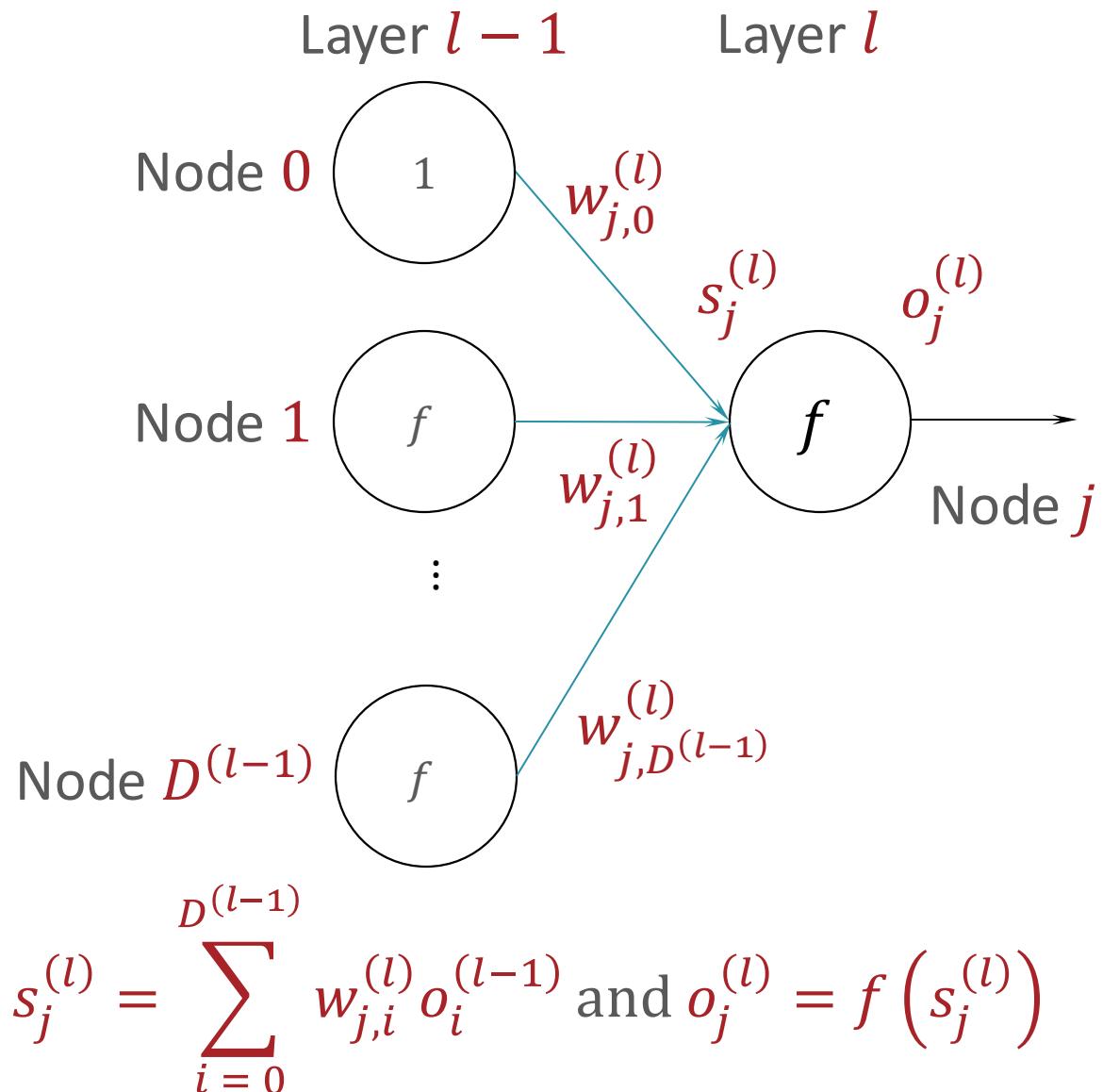
$$W^{(l)} \in \mathbb{R}^{D^{(l)} \times (D^{(l-1)} + 1)}$$



$w_{j,i}^{(l)}$  is the weight between node  $i$  in layer  $l - 1$  and node  $j$  in layer  $l$

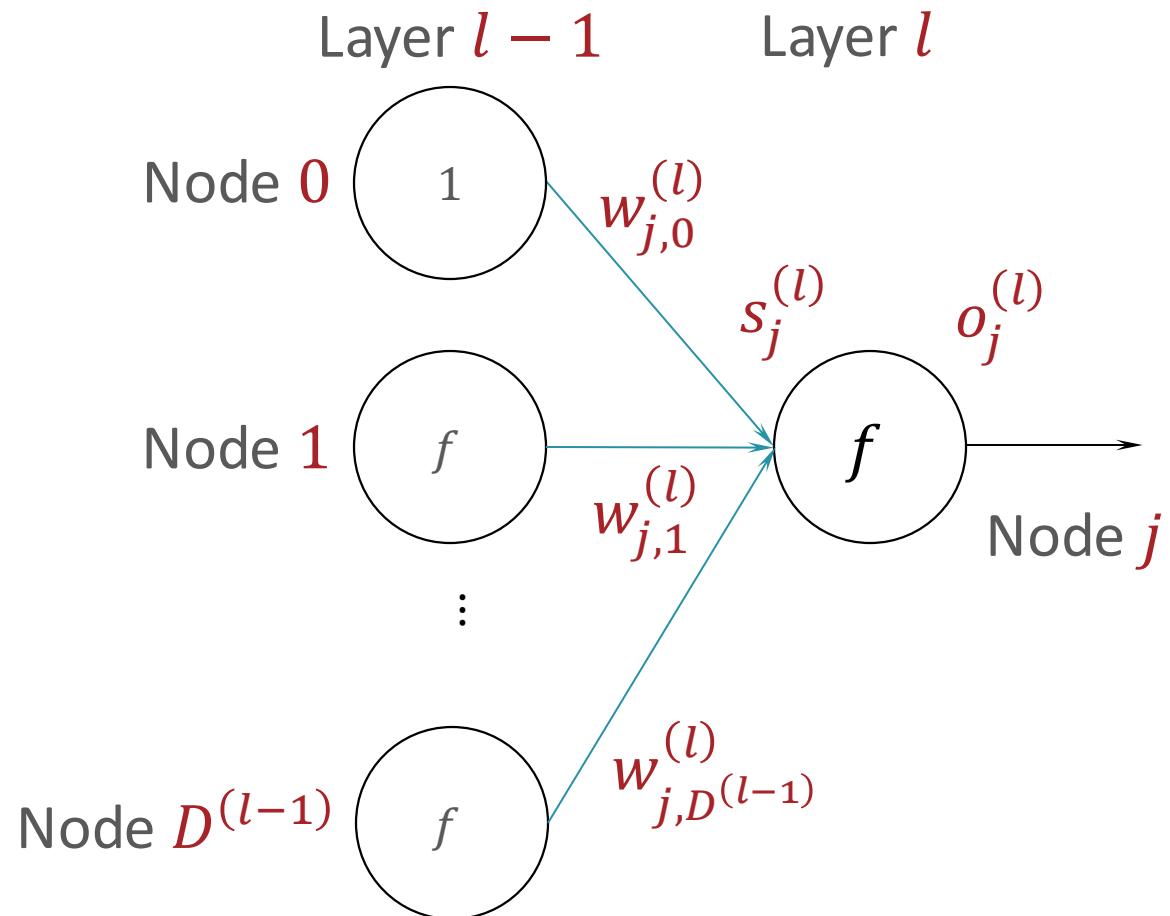
# Signal and Outputs

Every node has an incoming *signal* and outgoing *output*



# Signal and Outputs

Every node has an incoming *signal* and outgoing *output*



$$\mathbf{s}^{(l)} = W^{(l)} \mathbf{o}^{(l-1)} \text{ and } \mathbf{o}^{(l)} = [1, f(\mathbf{s}^{(l)})]^T$$

# Forward Propagation for Making Predictions

- Input: weights  $W^{(1)}, \dots, W^{(L)}$  and a query data point  $\mathbf{x}$
- Initialize  $\mathbf{o}^{(0)} = [1, \mathbf{x}]^T$
- For  $l = 1, \dots, L$ 
  - $\mathbf{s}^{(l)} = W^{(l)} \mathbf{o}^{(l-1)}$
  - $\mathbf{o}^{(l)} = [1, f(\mathbf{s}^{(l)})]^T$
- Output:  $h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}) = \mathbf{o}^{(L)}$

# Gradient Descent for Learning

- Input:  $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N, \eta^{(0)}$
- Initialize all weights  $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$  to small, random numbers and set  $t = 0$  (???)
- While TERMINATION CRITERION is not satisfied (???)
  - For  $l = 1, \dots, L$ 
    - Compute  $G^{(l)} = \nabla_{W^{(l)}} \ell_{\mathcal{D}}(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)})$  (???)
    - Update  $W^{(l)}$ :  $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
    - Increment  $t$ :  $t = t + 1$
- Output:  $W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}$

# Loss Functions for Neural Networks

- Regression - squared error (same as linear regression!)
$$\ell_{\mathcal{D}} \left( W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)} \right) = \sum_{n=1}^N \left( h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)}) - y^{(n)} \right)^2$$
- Binary classification - cross-entropy loss (same as logistic regression!)
  - Assume  $P(Y = 1 | \mathbf{x}, W^{(1)}, \dots, W^{(L)}) = h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)})$

$$\begin{aligned}\ell_{\mathcal{D}} \left( W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)} \right) &= - \sum_{n=1}^N \log P(y^{(n)} | \mathbf{x}^{(n)}, W^{(1)}, \dots, W^{(L)}) \\ &= - \sum_{n=1}^N \log \left( h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)})^{y^{(n)}} \left( 1 - h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)}) \right)^{1-y^{(n)}} \right) \\ &= - \sum_{n=1}^N y^{(n)} \log \left( h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)}) \right) \\ &\quad + (1 - y^{(n)}) \log \left( 1 - h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)}) \right)\end{aligned}$$

# Loss Functions for Neural Networks

- Multi-class classification - cross-entropy loss
  - Express the label as a one-hot or one-of- $C$  vector e.g.,
$$y = [0 \quad 0 \quad 1 \quad 0 \quad \dots \quad 0]$$
  - Assume the neural network output is also a vector of length  $C$

$$P(y[c] = 1 | \mathbf{x}, W^{(1)}, \dots, W^{(L)}) = h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)})[c]$$

- Then the cross-entropy loss is

$$\begin{aligned}\ell_{\mathcal{D}}(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}) &= - \sum_{n=1}^N \log P(y^{(n)} | \mathbf{x}^{(n)}, W^{(1)}, \dots, W^{(L)}) \\ &= - \sum_{n=1}^N \sum_{c=1}^C y[c] \log h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)})[c]\end{aligned}$$

# Key Takeaways

- Perceptrons can be combined to achieve non-linear decision boundaries
- Feed-forward neural network model
  - Activation function
  - Layers: input, hidden & output
  - Weight matrices
  - Signals & outputs
- Forward propagation for making predictions
- Neural networks can use the same loss functions as other machine learning models (e.g., squared error for regression, cross-entropy for classification)