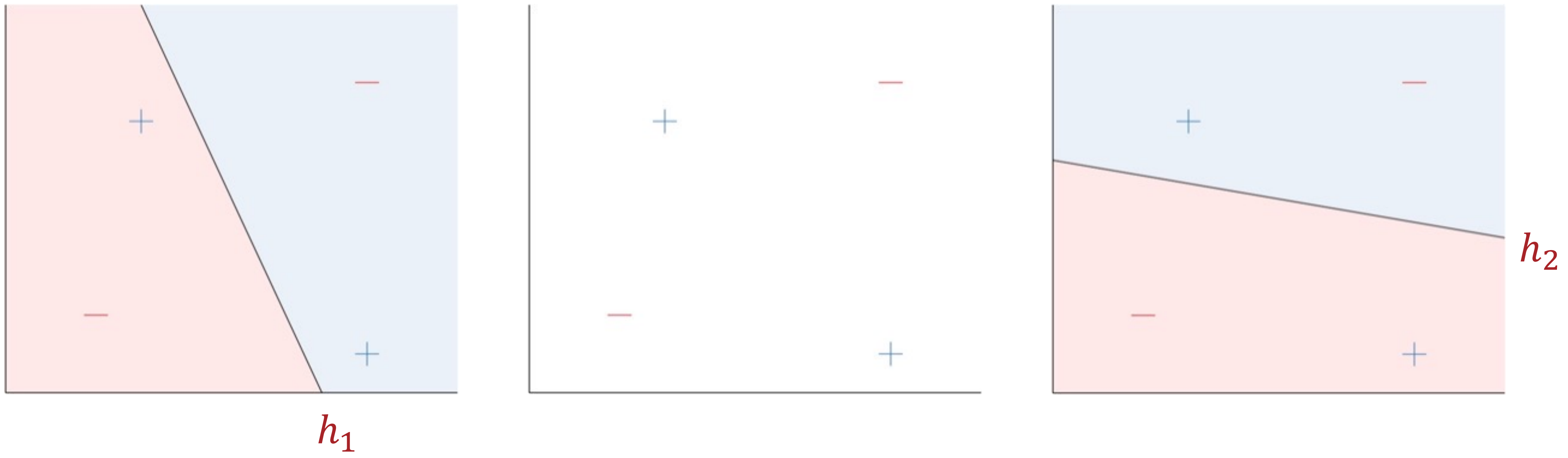
10-301/601: Introduction to Machine Learning Lecture 13 – Neural Networks

Henry Chai

5/21/25

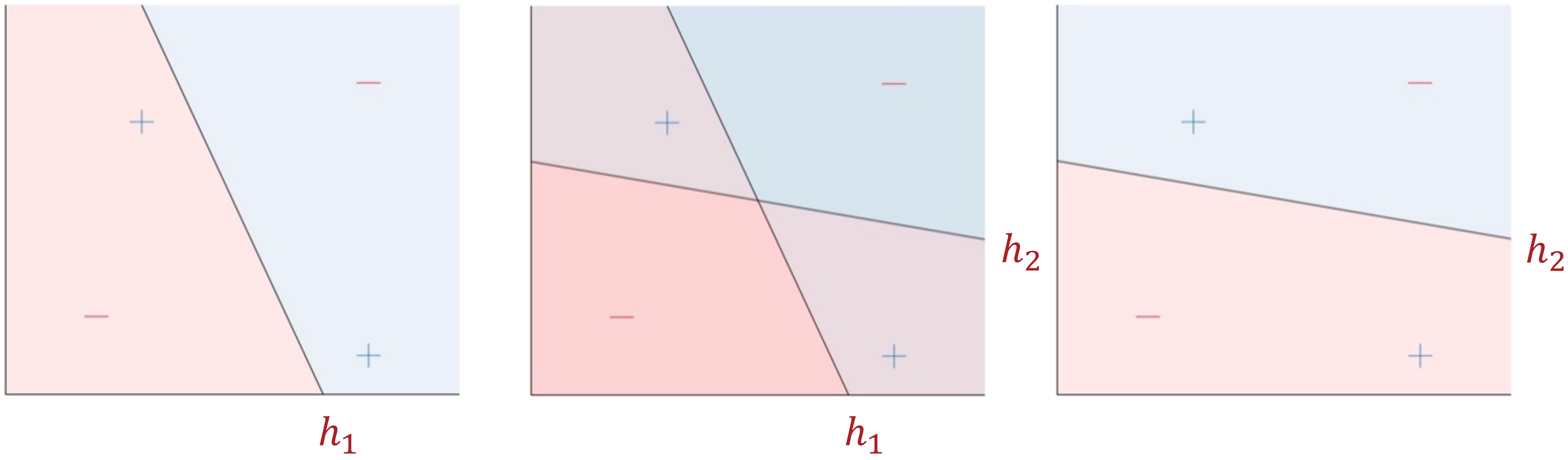
Biological Neural Network

Source: https://science-art.com/image/?id=2971&m=168&pagename=neural_network_3d#.W_wrzJNKhUM

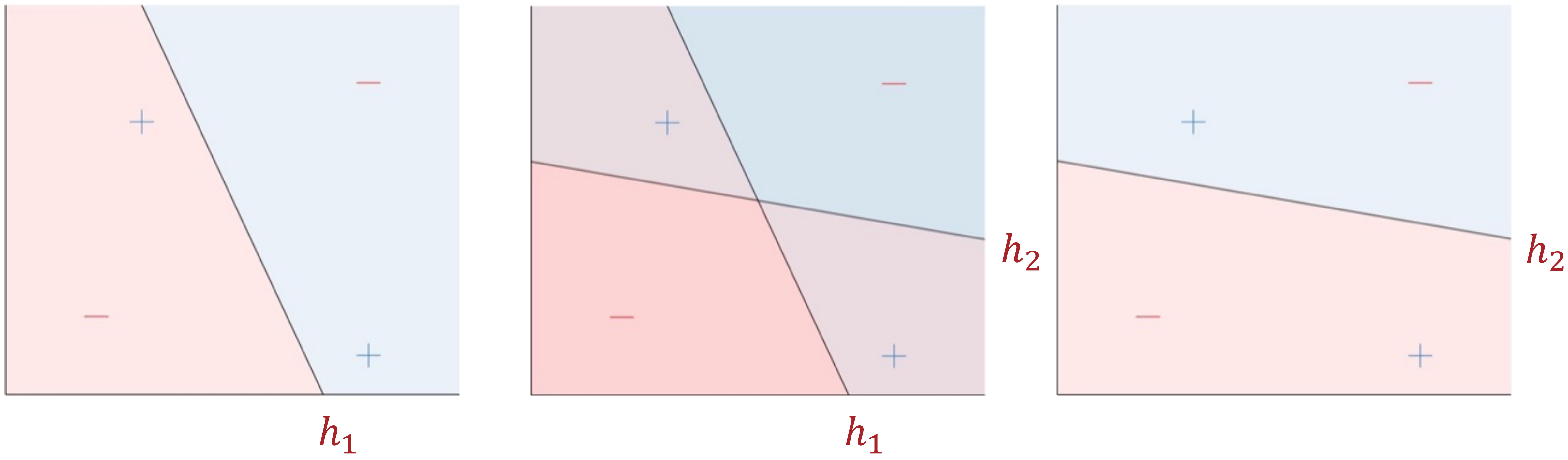


Perceptrons

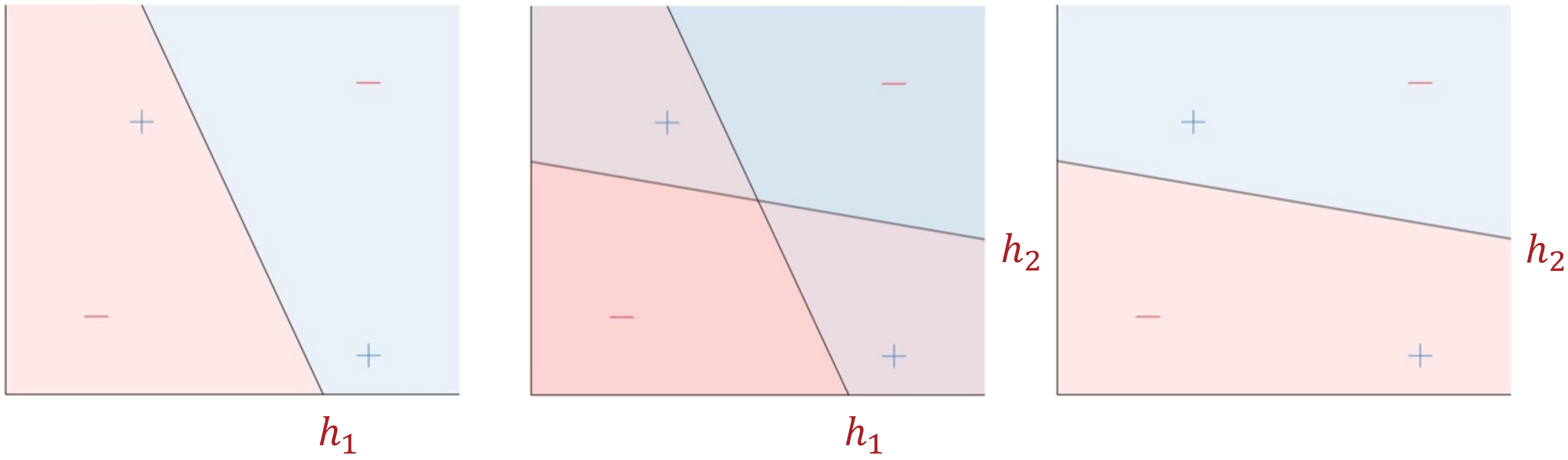
- Linear model for classification
- $h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$
- Predictions are $+1$ or -1



Combining Perceptrons



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



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

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

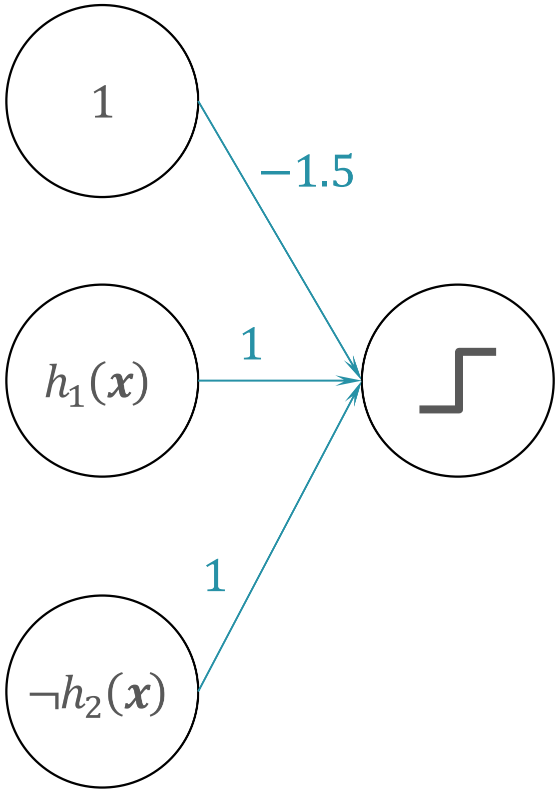
- Boolean variables are either $+1$ (”true”) or -1 (“false”)
- Basic Boolean operations
  - Negation: $\neg z = -1 * z$
  - And: $AND(z_1, z_2) = \text{sign}\left([-1.5, 1, 1]\begin{bmatrix}1\\ z_1\\ z_2\end{bmatrix}\right)$
  - Or: $OR(z_1, z_2) = \text{sign}\left([1.5, 1, 1]\begin{bmatrix}1\\ z_1\\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

Building a Network

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

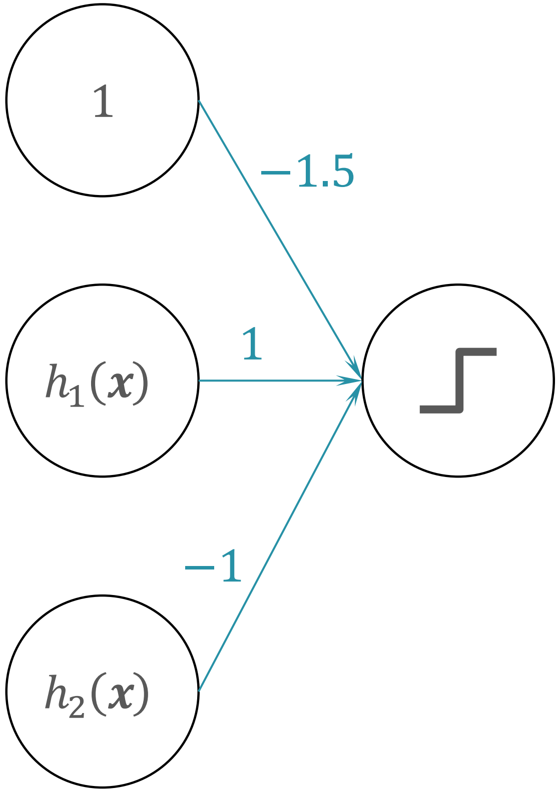
Building a Network

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



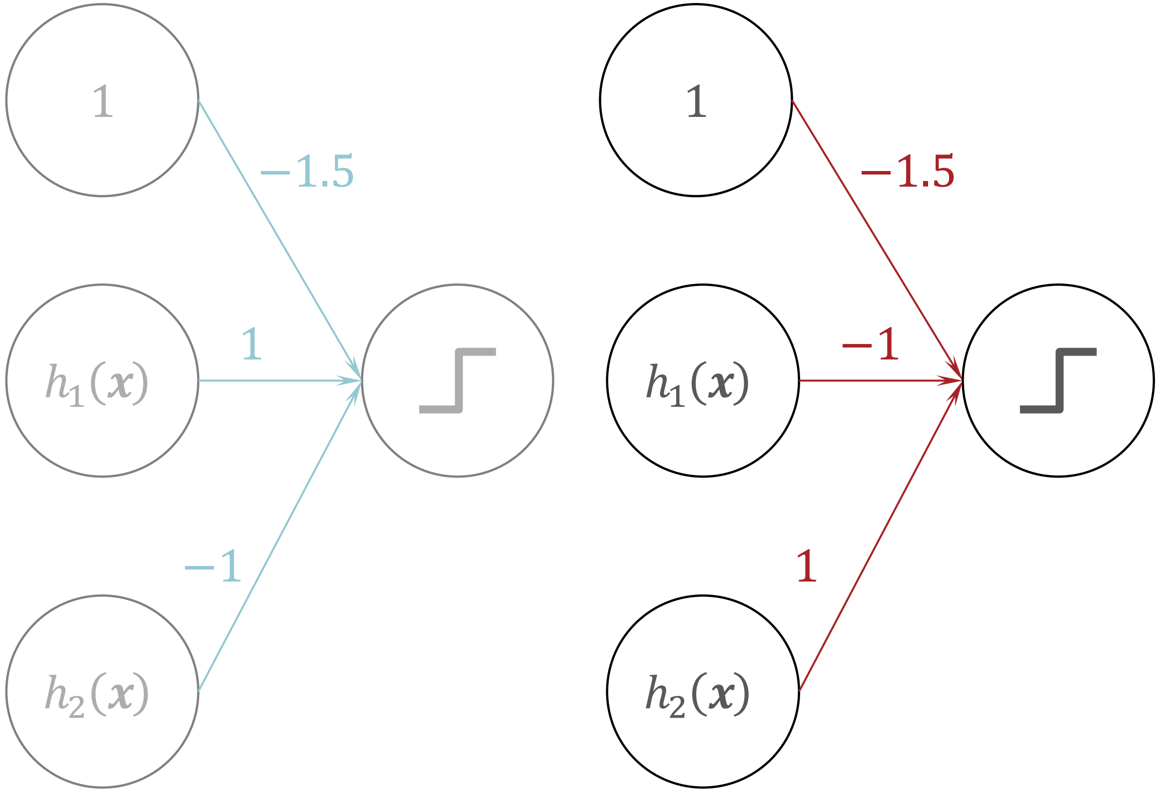
Building a Network

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



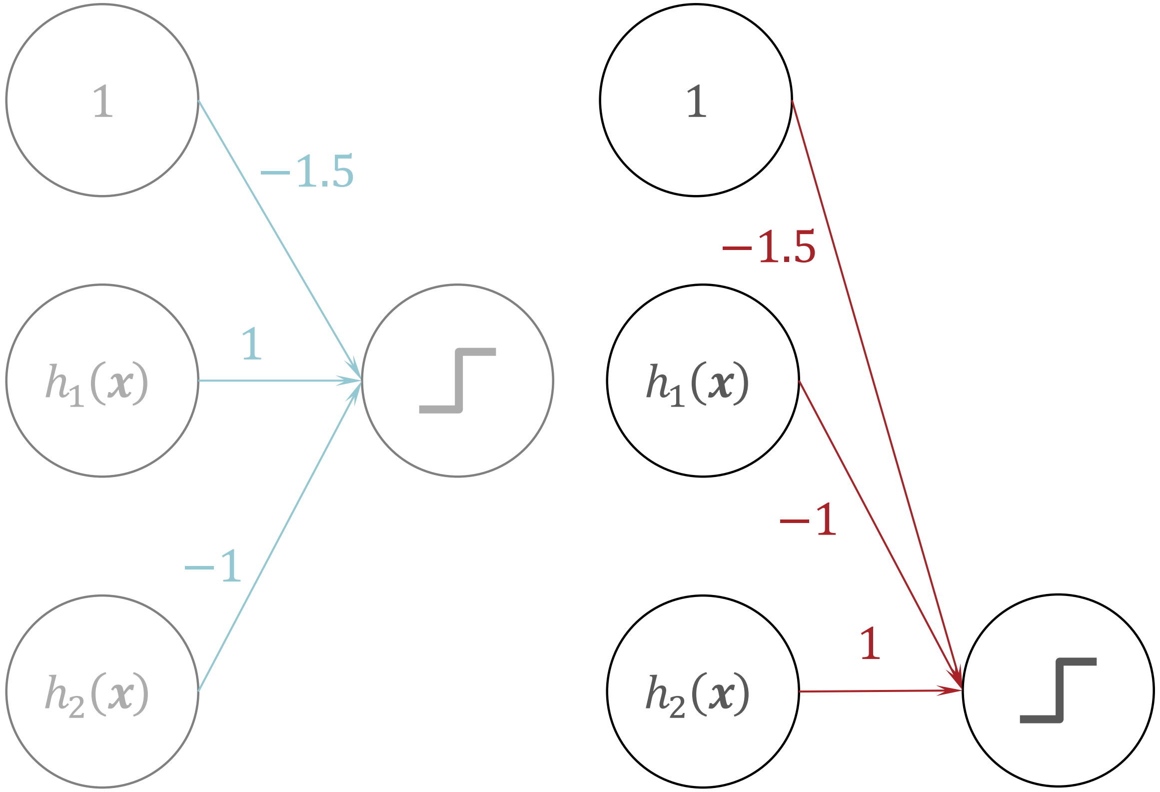
Building a Network

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



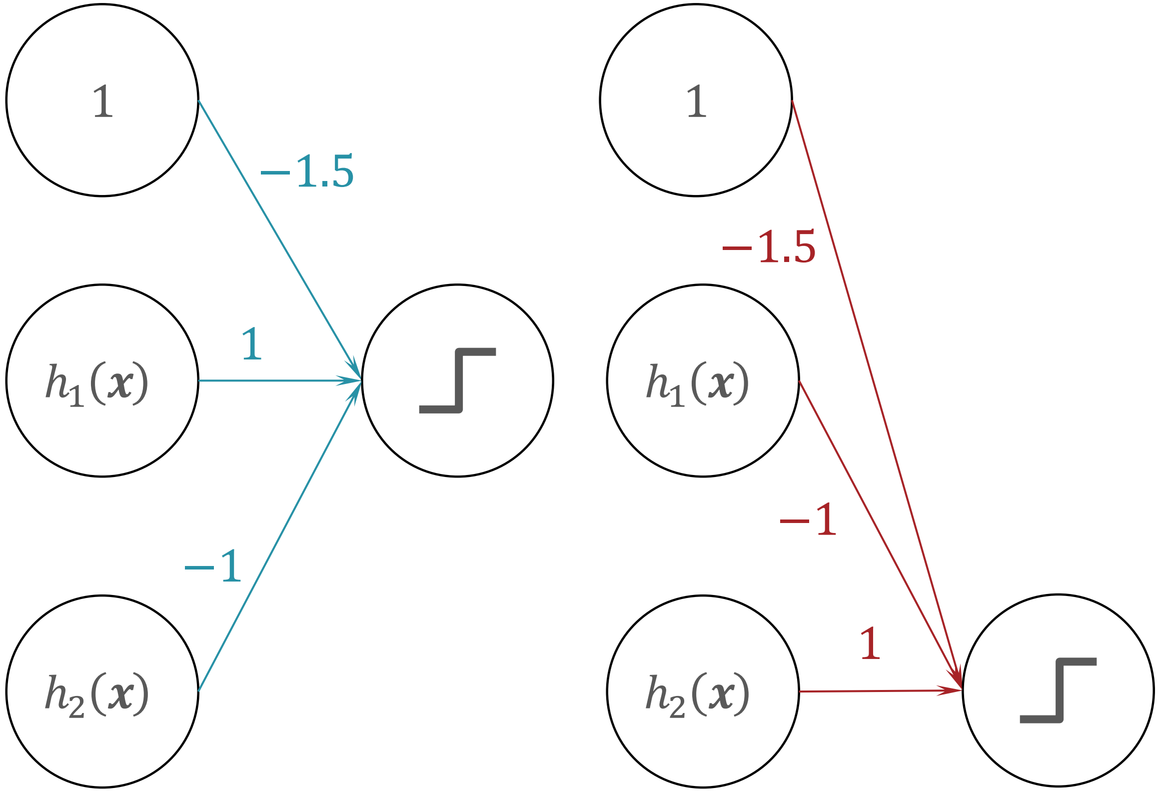
Building a Network

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



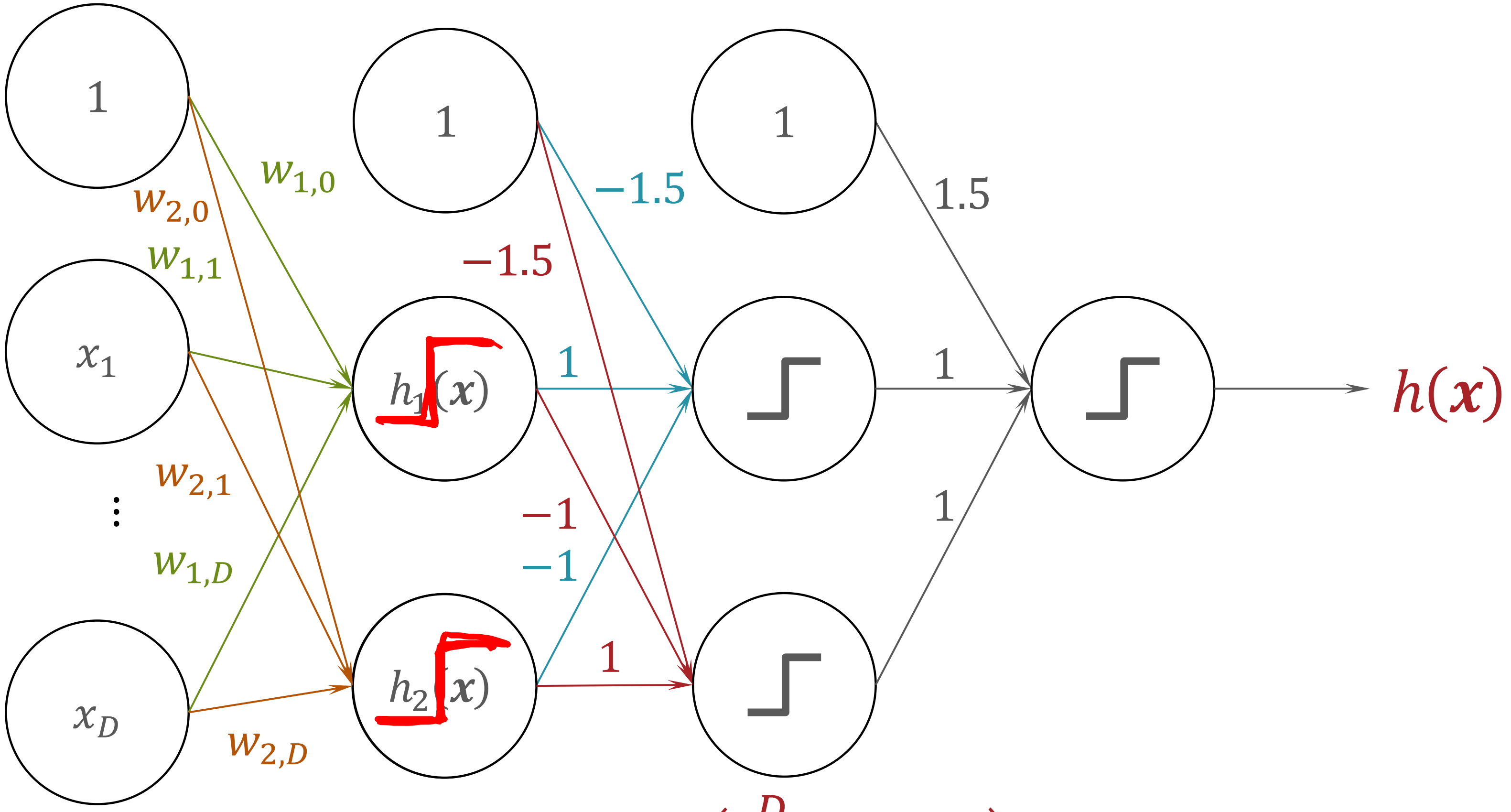
Building a Network

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



Building a Network

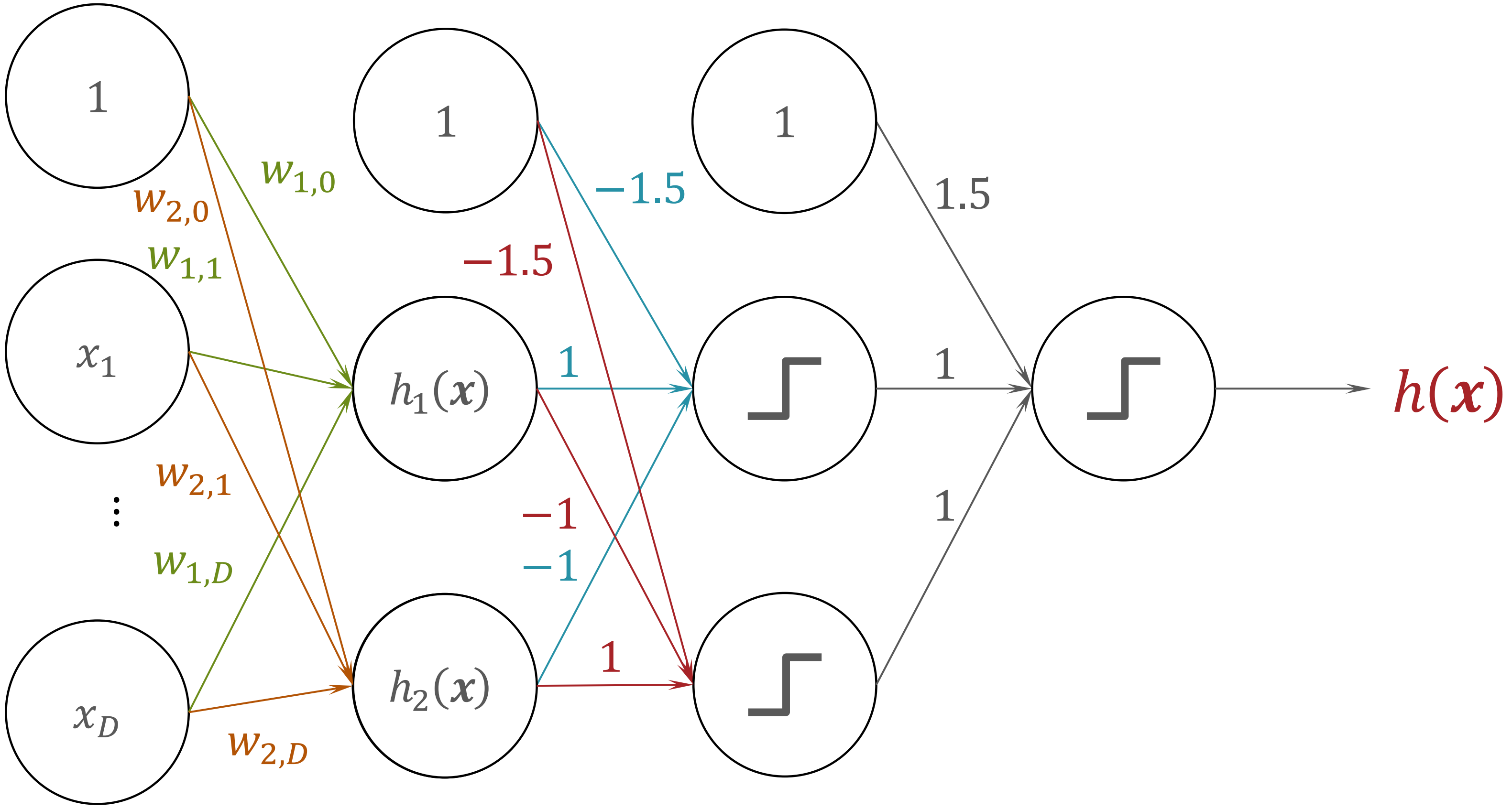
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



$$h_i(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w}_i^T \boldsymbol{x}\right) = \text{sign}\left(\sum_{d=0}^{D} w_{i,d}\, x_d\right)$$

Building a Network

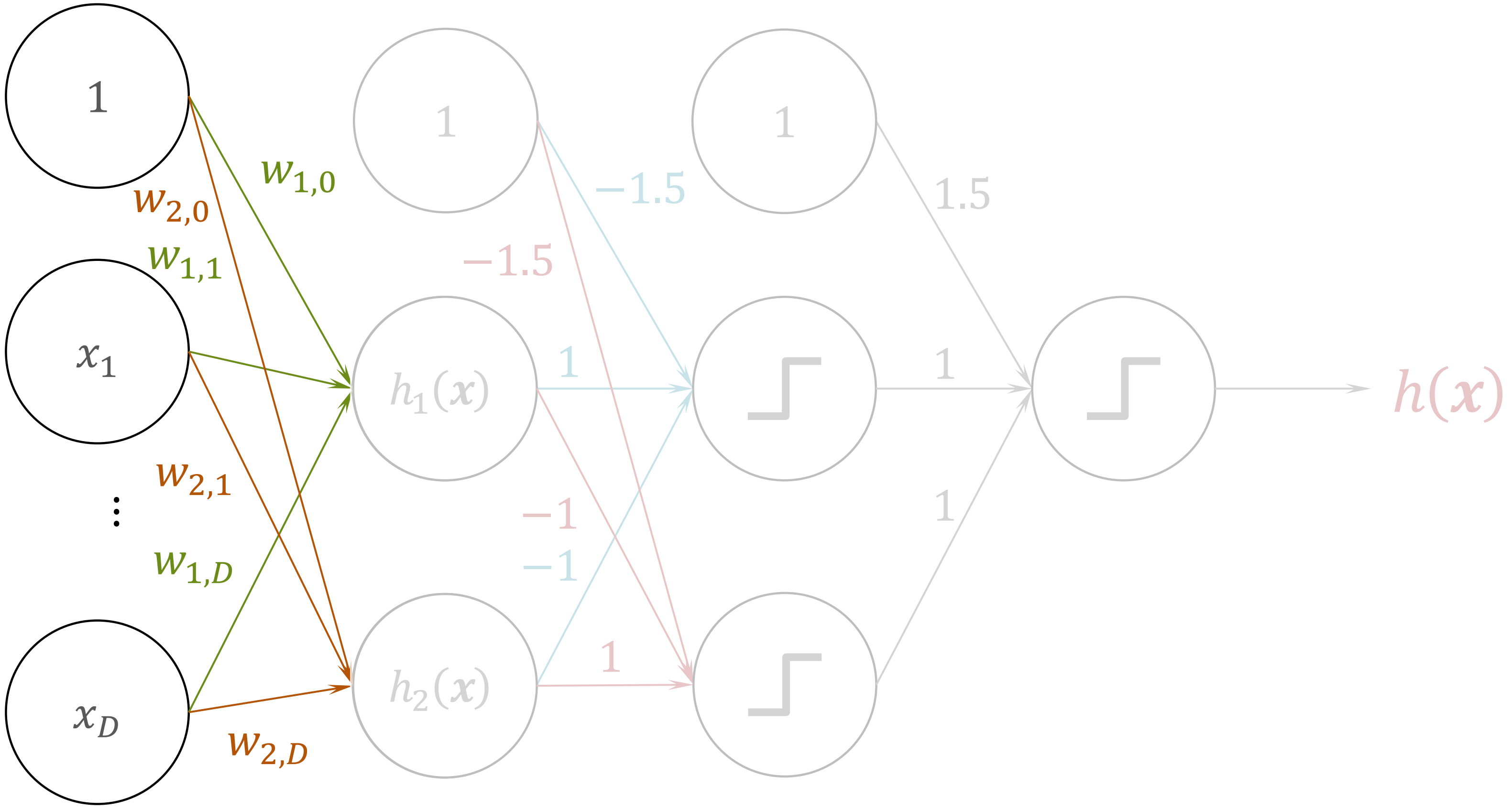
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



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

Building a Network

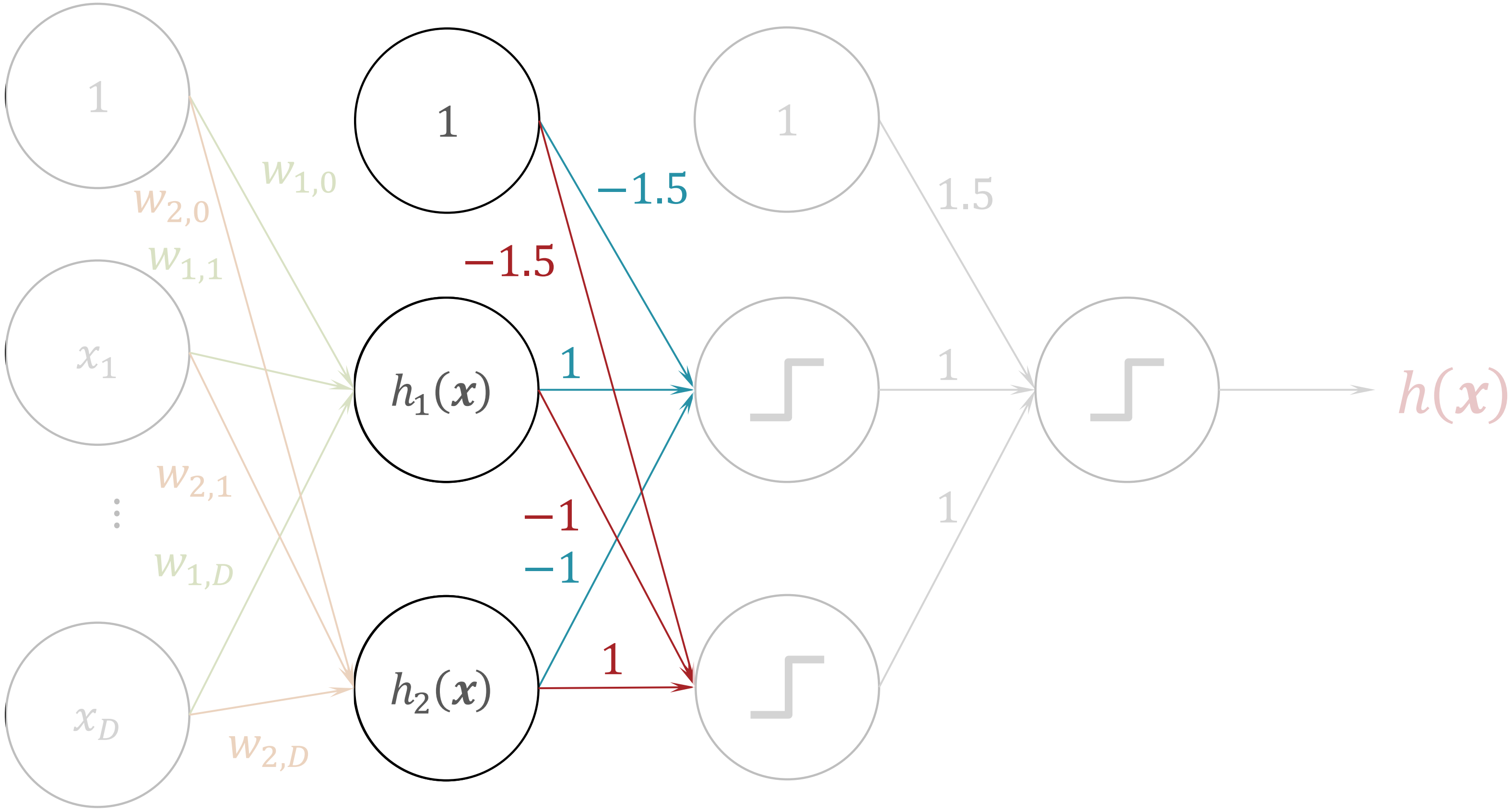
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



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

Building a Network

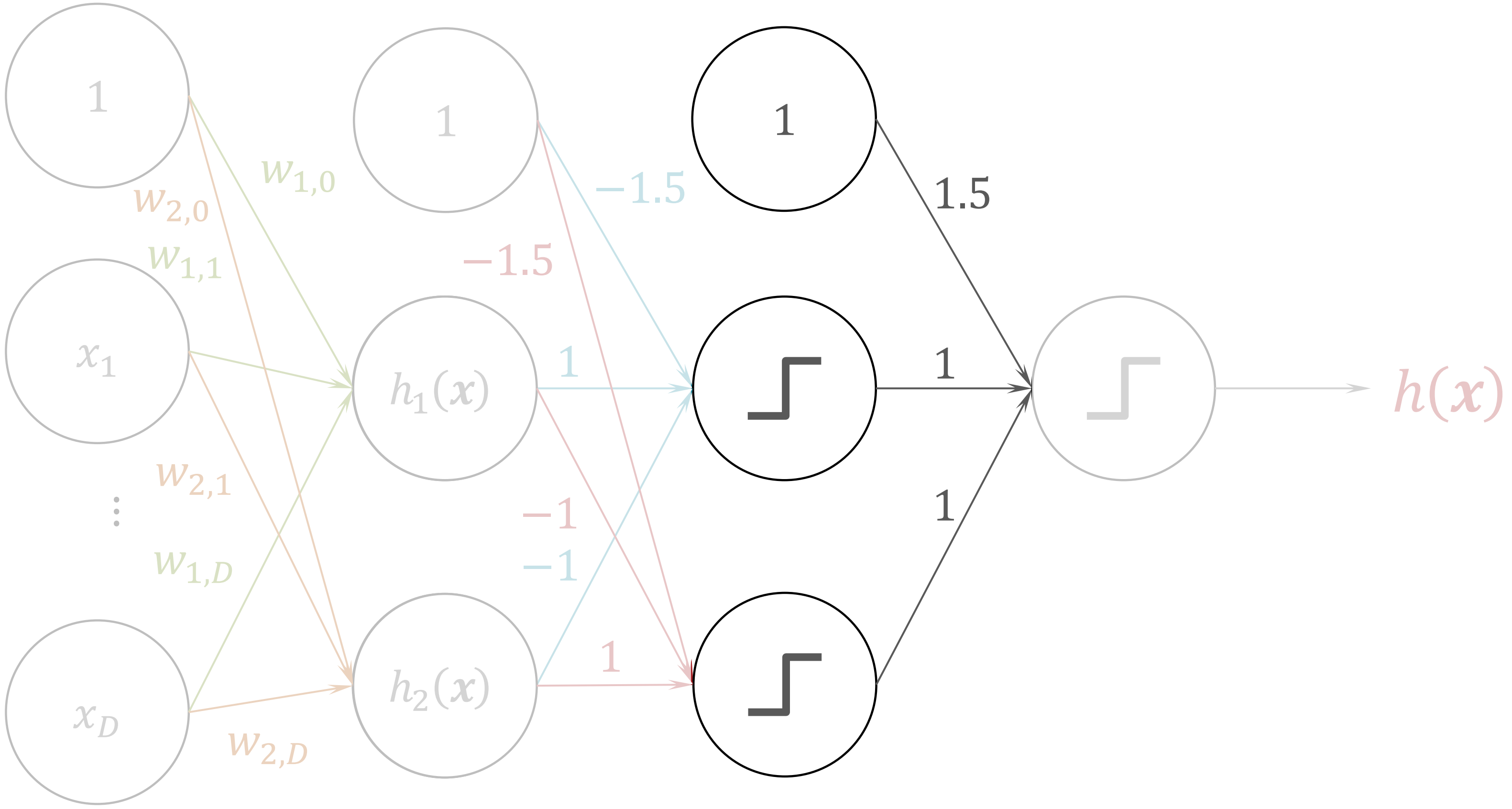
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



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

Building a Network

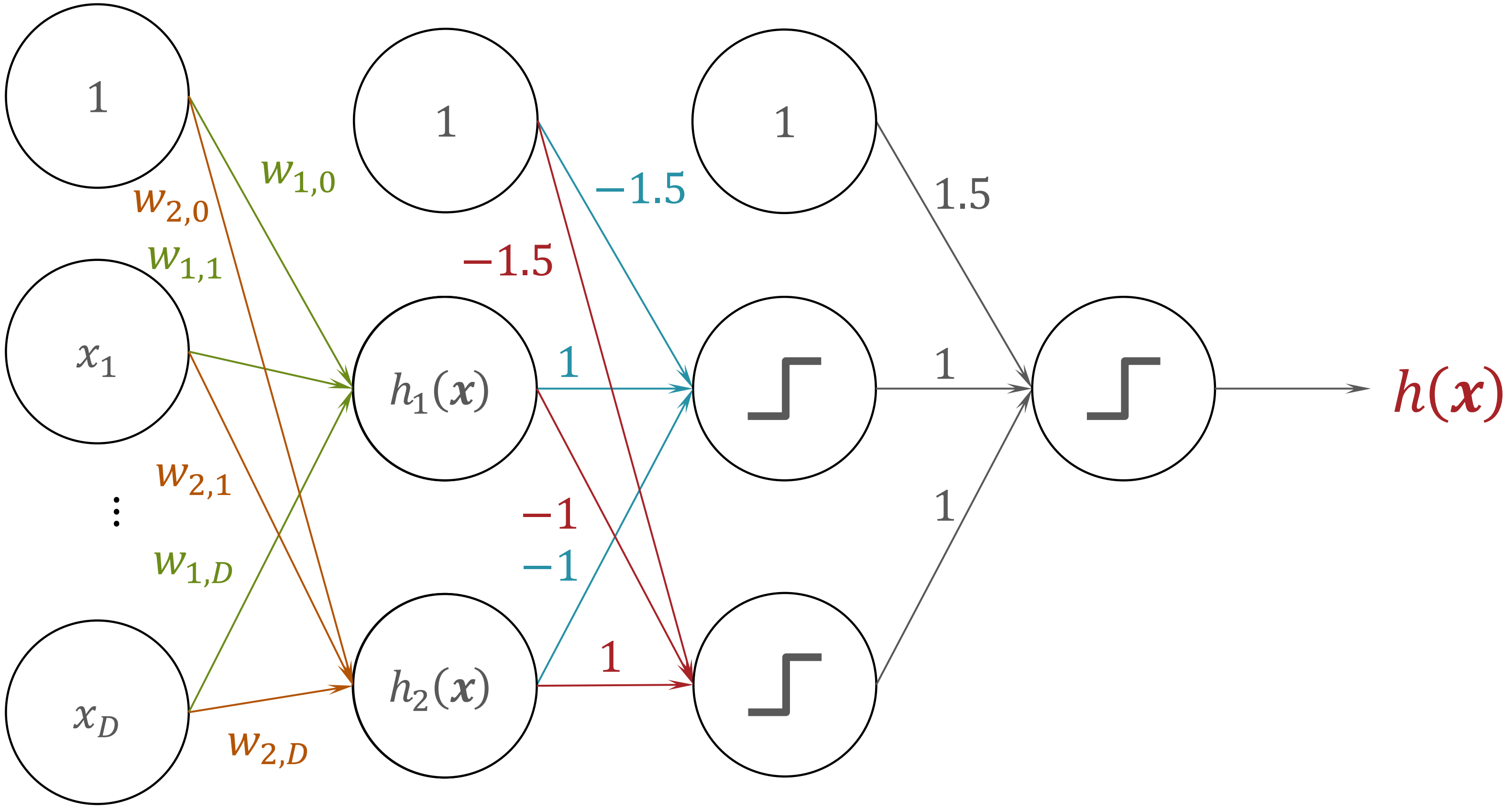
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



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

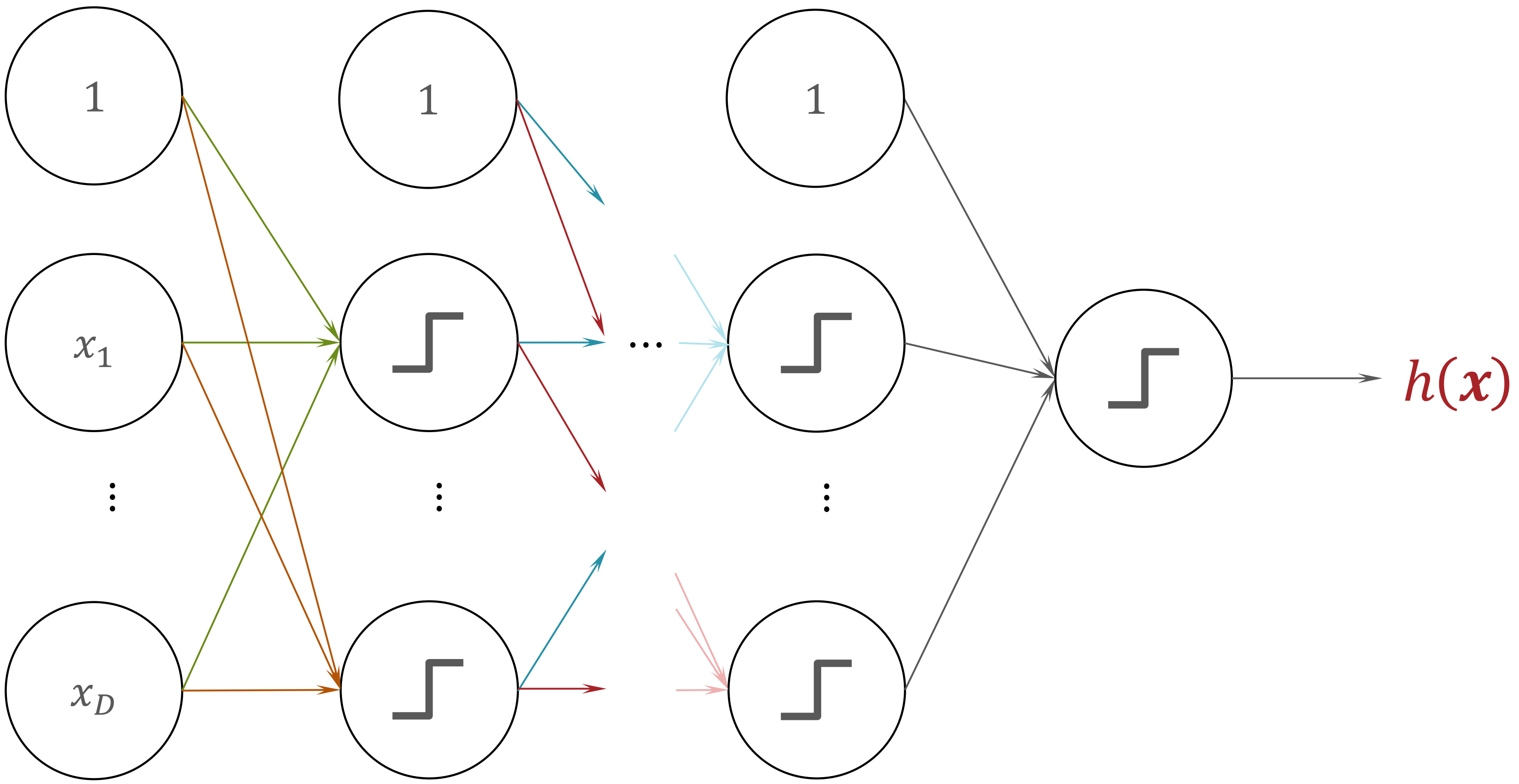
Building a Network

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

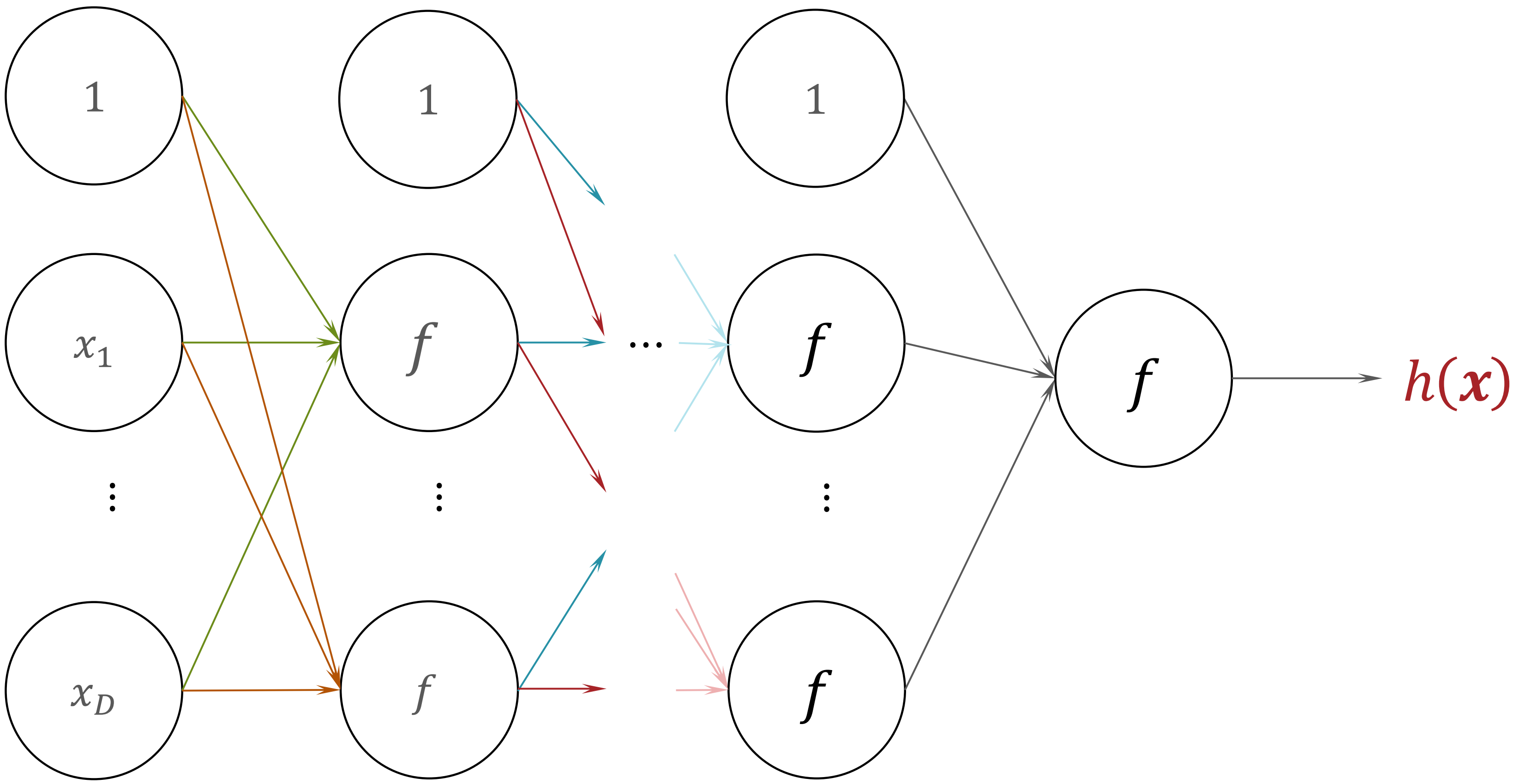


$$h(\boldsymbol{x}) = \text{sign}(\text{sign}(\text{sign}(\boldsymbol{w}_1^T\boldsymbol{x}) - \text{sign}(\boldsymbol{w}_2^T\boldsymbol{x}) - 1.5) +$$
$$\text{sign}(-\text{sign}(\boldsymbol{w}_1^T\boldsymbol{x}) + \text{sign}(\boldsymbol{w}_2^T\boldsymbol{x}) - 1.5) + 1.5)$$

Multi-Layer Perceptron (MLP)



(Fully-Connected) Feed Forward Neural Network

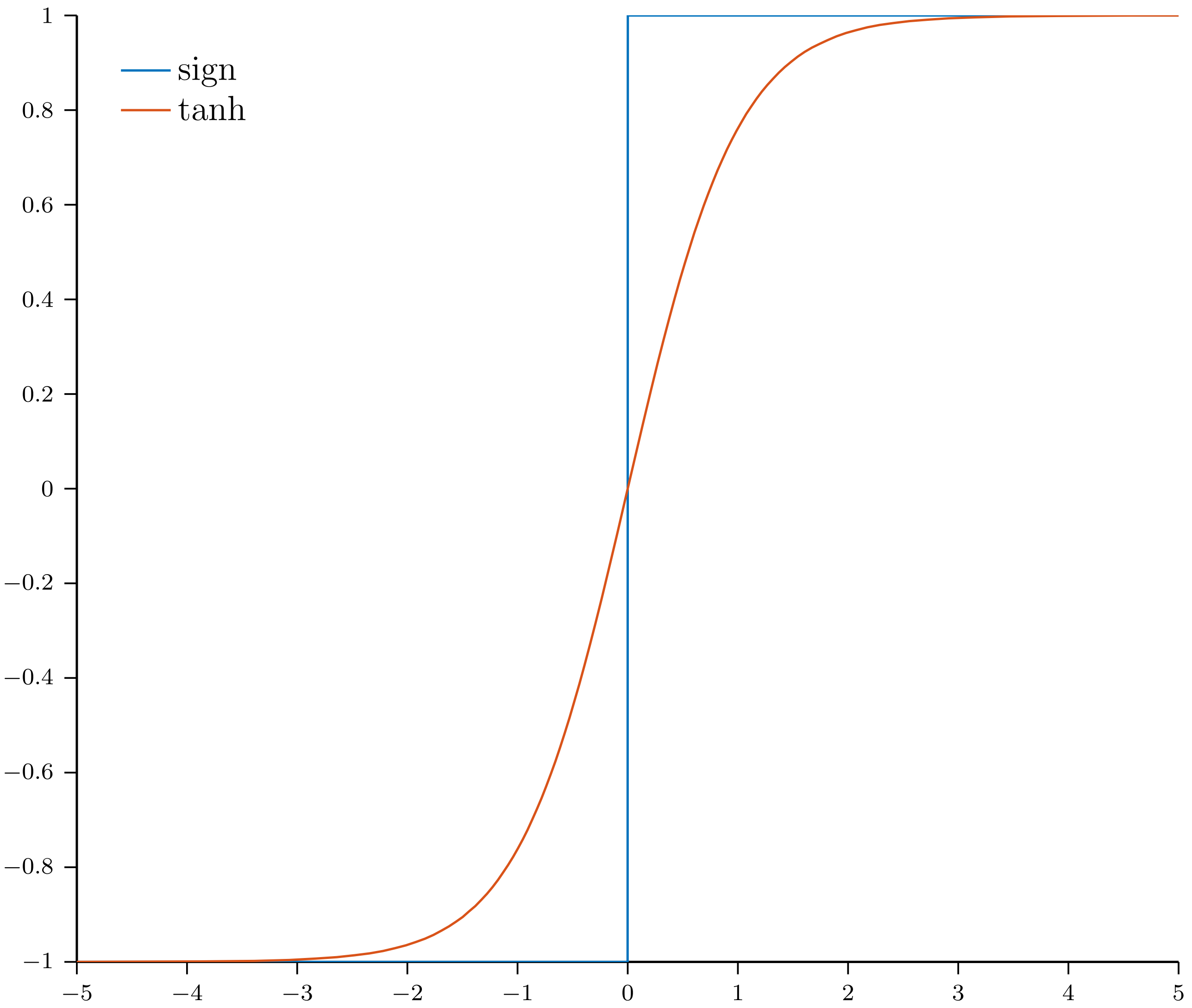


$f(\cdot)$

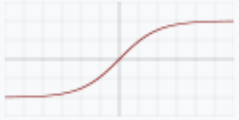
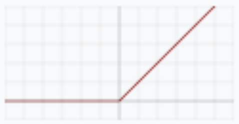
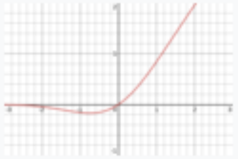
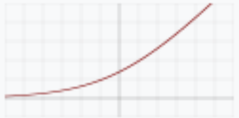
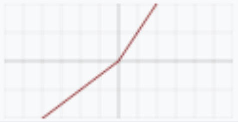
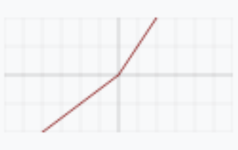
- Hyperbolic tangent:

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

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



Other Activation Functions

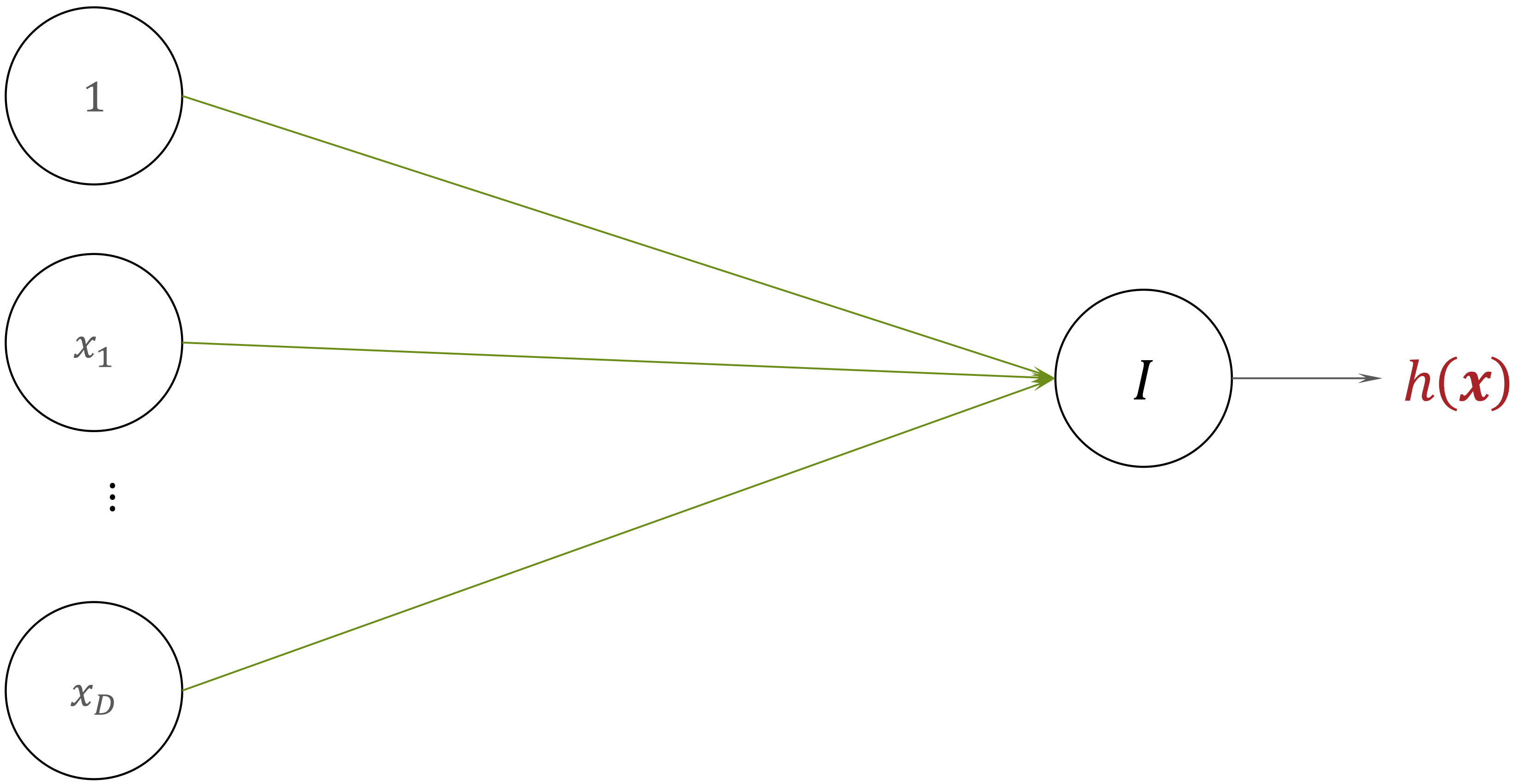
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases} = \max\{0, x\} = x\mathbf{1}_{x>0}$
Gaussian Error Linear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

Source: https://en.wikipedia.org/wiki/Activation_function

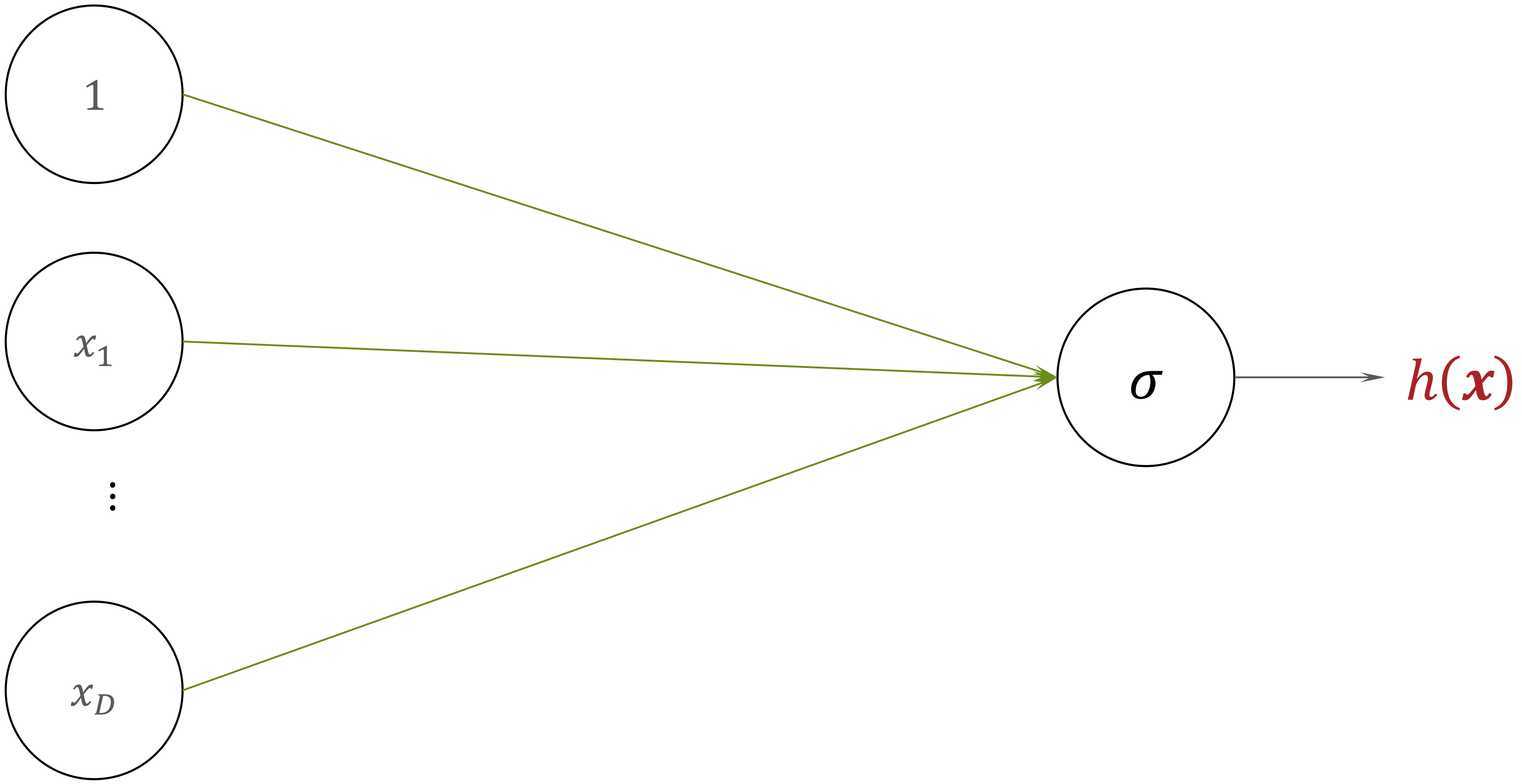
True or False: both the linear regression model and the logistic regression model can be expressed as neural networks.

- True for both
- True for linear regression, false for logistic regression
- False for linear regression, true for logistic regression
- False for both

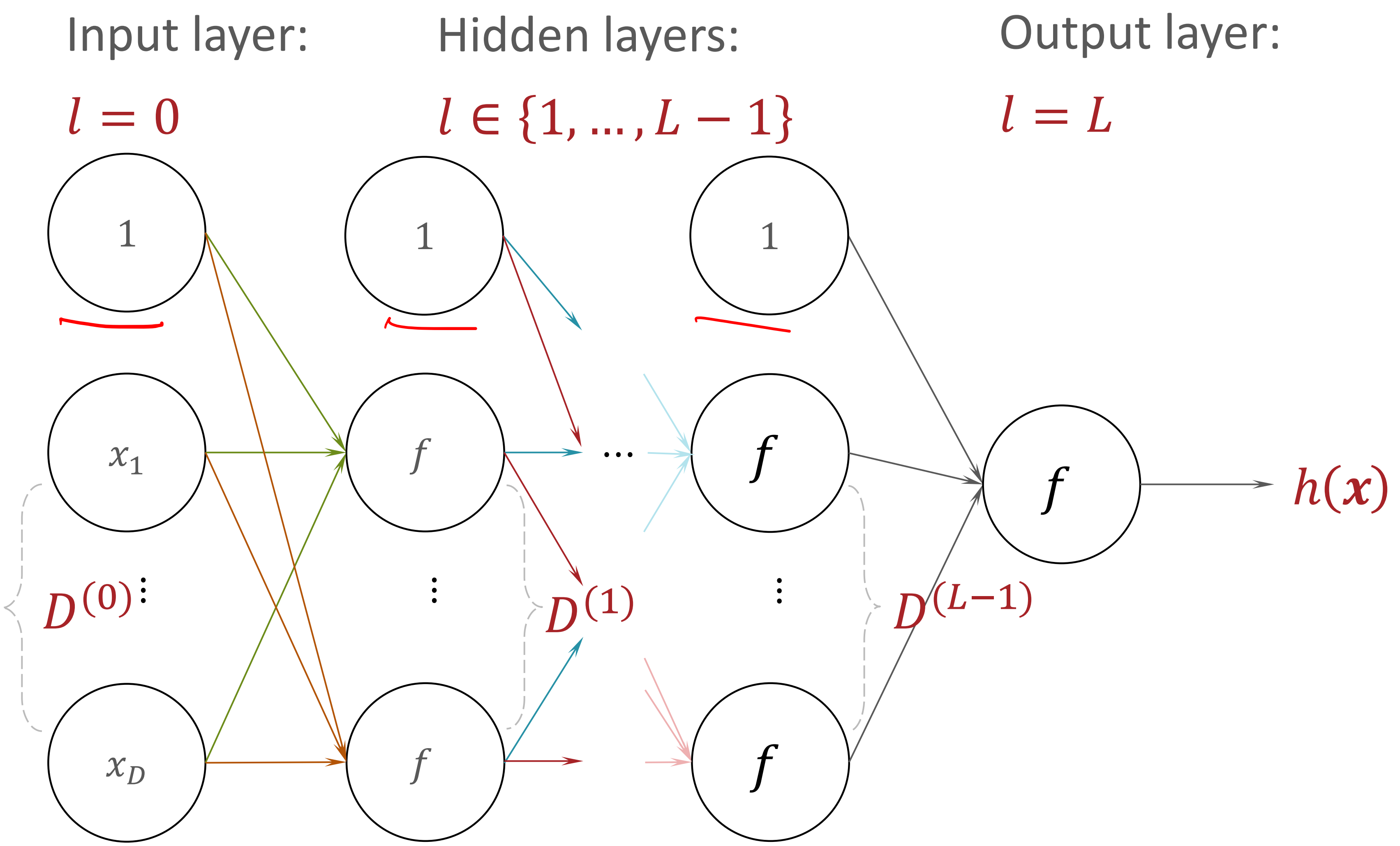
Linear Regression as a Neural Network



Logistic Regression as a Neural Network



(Fully-Connected) Feed Forward Neural Network

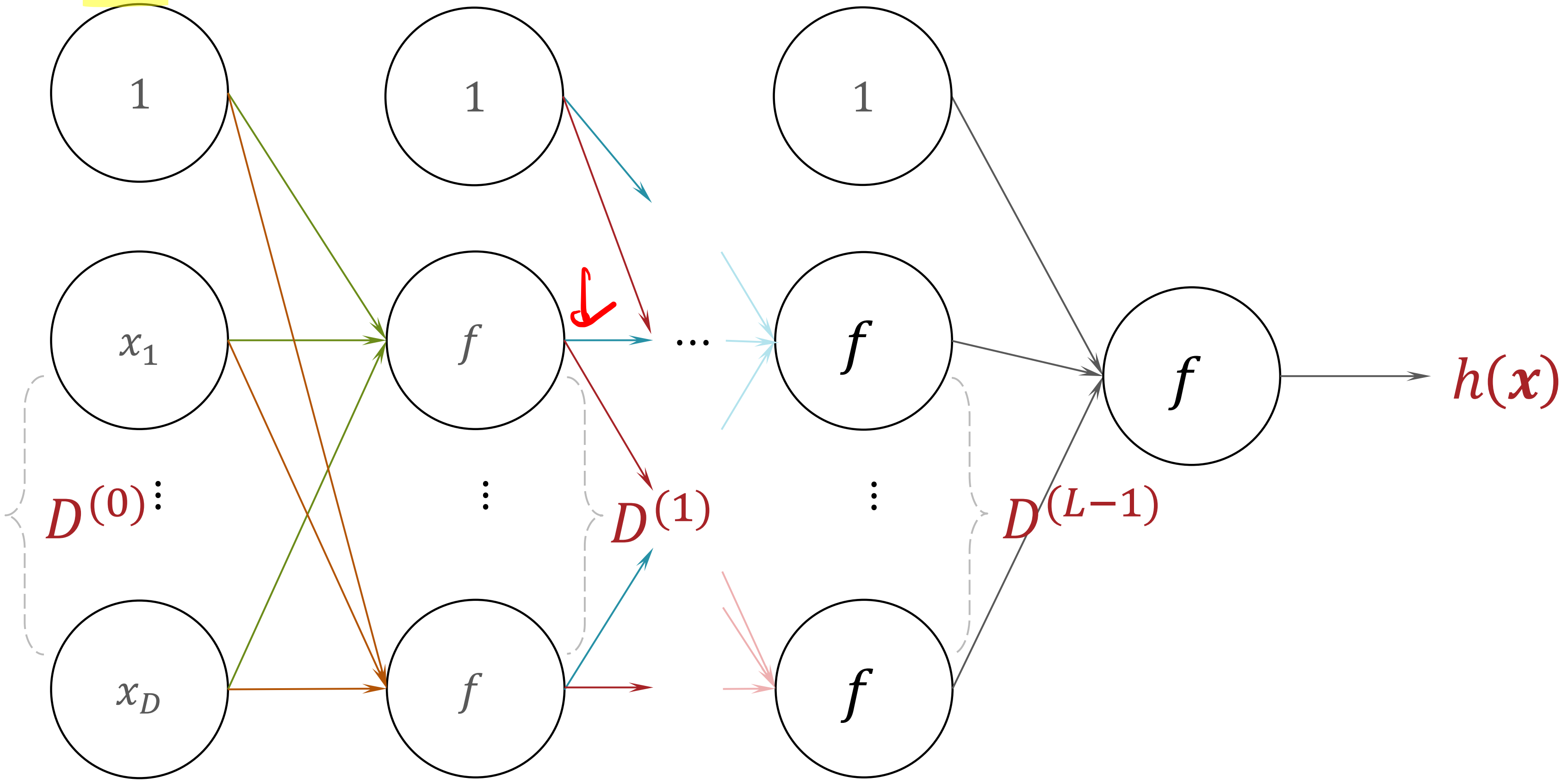


Layer l has dimension $D^{(l)}$ → Layer l has $D^{(l)} + 1$ nodes, counting the bias node

(Fully-Connected) Feed Forward Neural Network

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

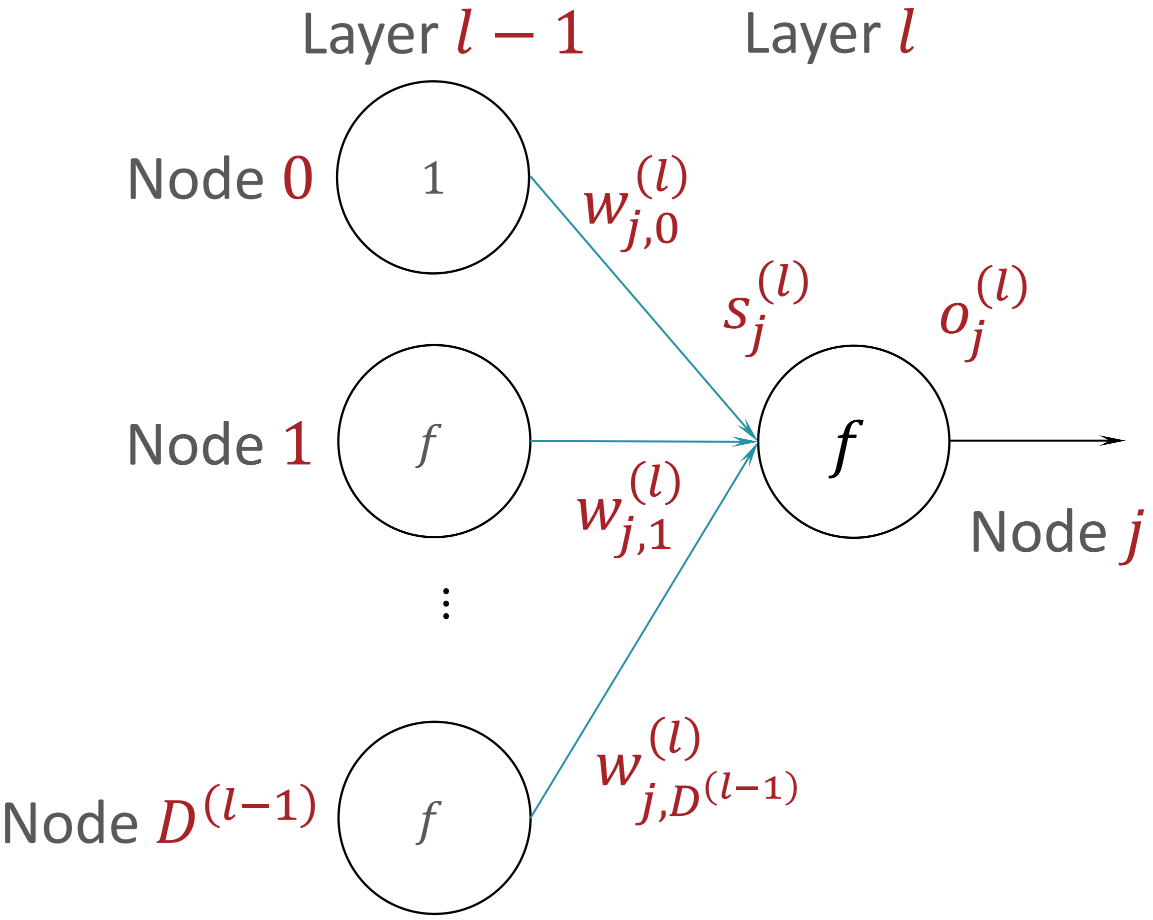
$W^{(l)} \in \mathbb{R}^{D^{(l)} \times (D^{(l-1)}+1)}$



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

Signal and Outputs

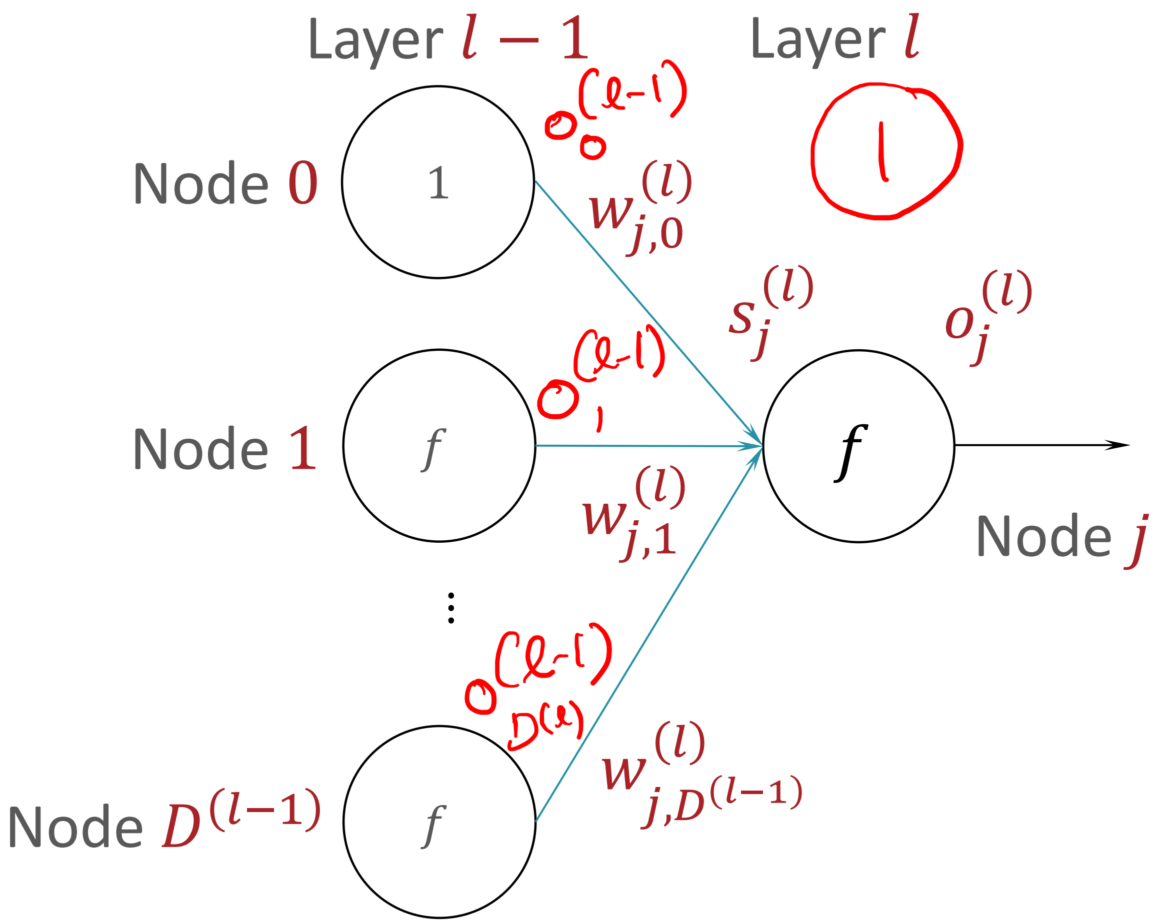
Every node has an incoming *signal* and outgoing *output*



$$s_j^{(l)} = \sum_{i=0}^{D^{(l-1)}} w_{j,i}^{(l)} o_i^{(l-1)} \text{ and } o_j^{(l)} = f\left(s_j^{(l)}\right)$$

Signal and Outputs

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



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

Forward Propagation for Making Predictions

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

Gradient Descent for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta^{(0)}$
- 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 $G^{(l)} = \nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$
    - 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!)

$$\ell_{\mathcal{D}}\left(W^{(1)}_{(t)}, \ldots, W^{(L)}_{(t)}\right) = \frac{1}{N}\sum_{n=1}^{N}\left(h_{W^{(1)}_{(t)}, \ldots, W^{(L)}_{(t)}}\left(\vec{x}^{(n)}\right) - y^{(n)}\right)^2$$

- Binary classification - cross-entropy loss (same as logistic regression!)

Assume $P\left(Y=1 \mid \vec{x}, W^{(1)}, \ldots, W^{(L)}\right) = h(\vec{x})$

$$\ell_{\mathcal{D}}\left(W^{(1)}_{(t)}, \ldots, W^{(L)}_{(t)}\right) = -\sum_{n=1}^{N} \log P\left(y^{(n)} \mid \vec{x}^{(n)}, W^{(1)}_{(t)}, \ldots, W^{(L)}_{(t)}\right)$$

$$= -\sum_{n=1}^{N} \log\left(h\left(\vec{x}^{(n)}\right)^{y^{(n)}}\left(1-h\left(\vec{x}^{(n)}\right)\right)^{\left(1-y^{(n)}\right)}\right)$$

$$= -\sum_{n=1}^{N} y^{(n)} \log h\left(\vec{x}^{(n)}\right) + \left(1-y^{(n)}\right)\log\left(1-h\left(\vec{x}^{(n)}\right)\right)$$

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.,

$$y = [0 \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0]$$

Assume the network outputs a vector of length C

where $P(y[k] = 1 \mid \vec{x}, W^{(1)}, \ldots, W^{(L)}) = h(\vec{x})[k]$

The cross-entropy loss is

$$\ell_{\mathcal{D}}(W^{(1)}_{(t)}, \ldots, W^{(L)}_{(t)}) = -\sum_{n=1}^{N}\left(\sum_{k=1}^{C} y^{(n)}[k] \log h(\vec{x}^{(n)})[k]\right)$$

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)