10-701: Introduction to Machine Learning

Lecture 9 – Neural Networks

Hoda Heidari

* Slides adopted from F24 offering of 10701 by Henry Chai.

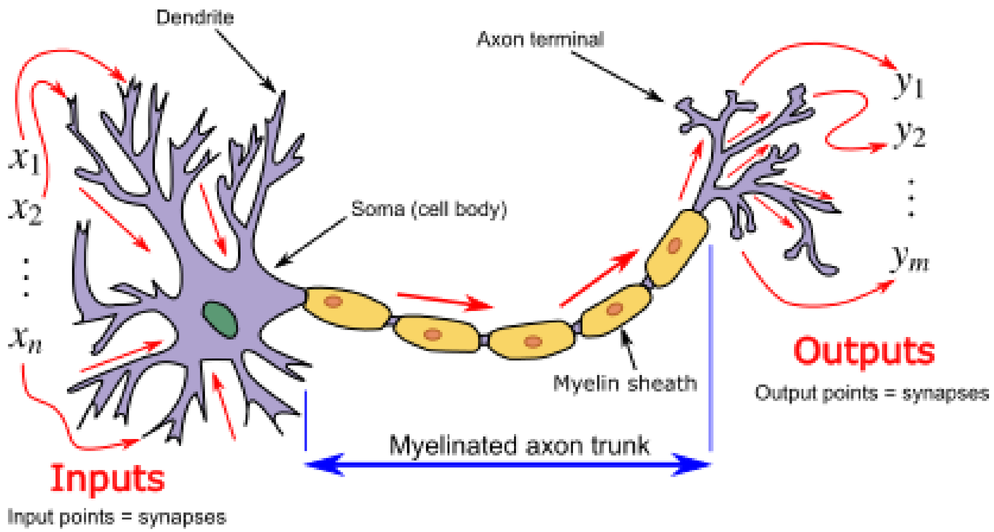
Biological Neural Network

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

Biological Neurons

A neuron has dendrites (**inputs**), a cell body (**processing unit**), and an axon (**output**).

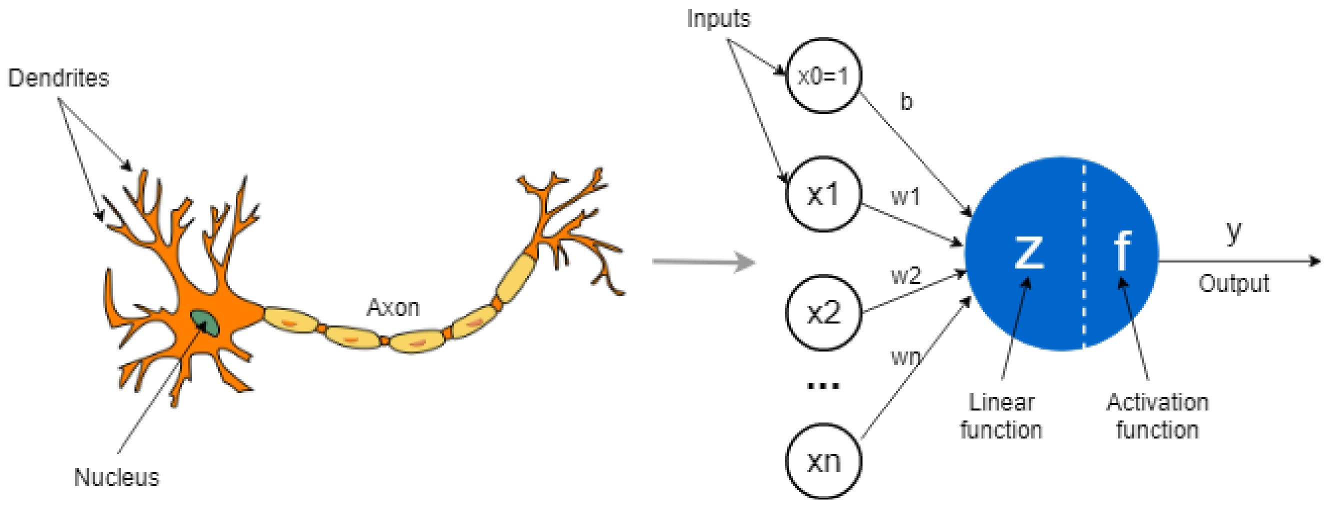
- Dendrites receive signals from other neurons.
- These signals are **combined** in the cell body through synapses.
- If the total signal is **strong enough**, the nucleus “fires” an electrical impulse down its axon to pass the message on to other neurons.



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

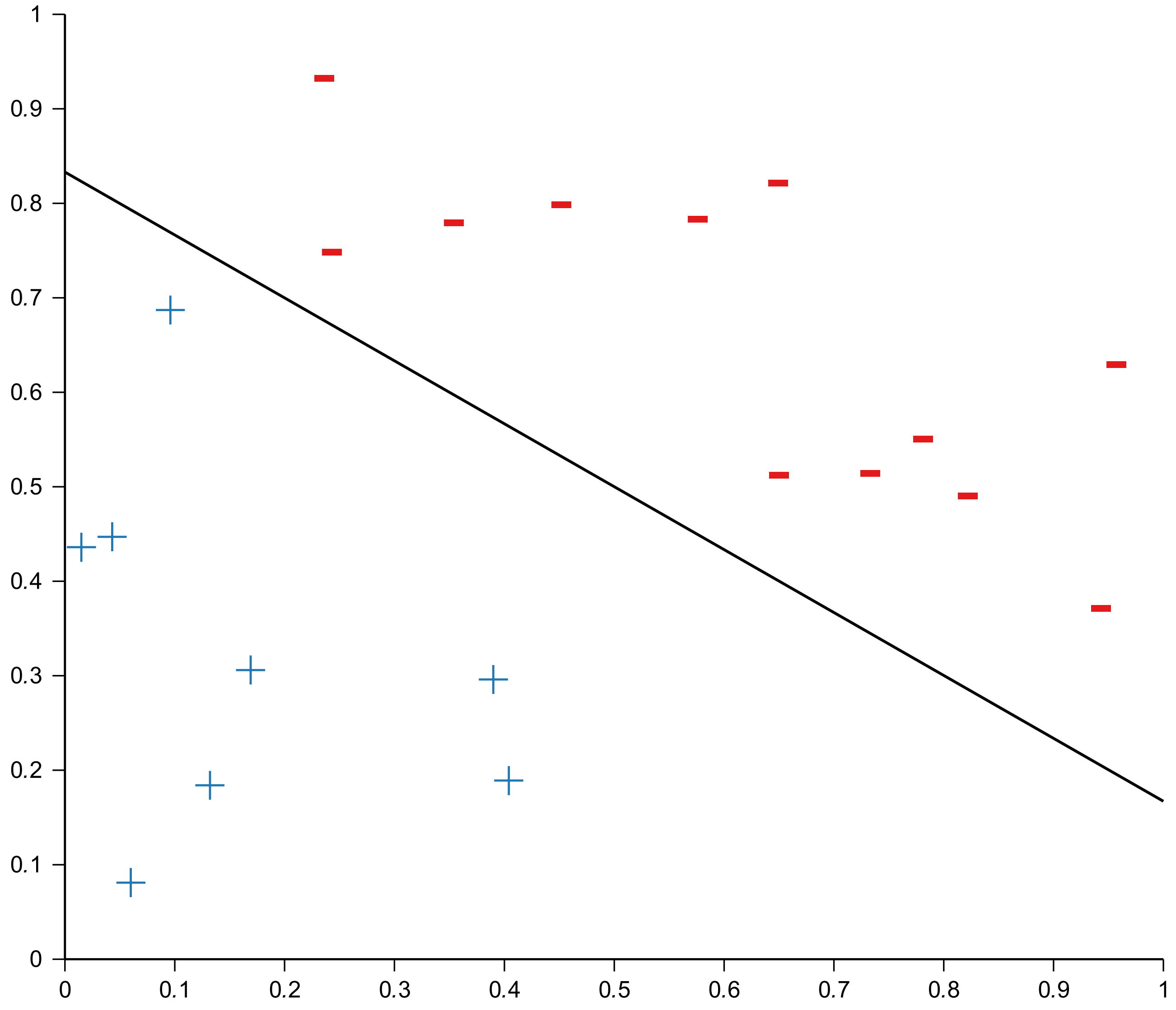
Artificial Neurons/ Perceptrons

- **Inputs** come into the neuron, each multiplied by a **weight** (importance).
- The neuron adds these up and often adds a **bias.**
- It then passes the sum through an **activation function** (a mathematical rule that decides if the neuron "fires" and how strongly).
- The **output** is then sent to the next layer of neurons.

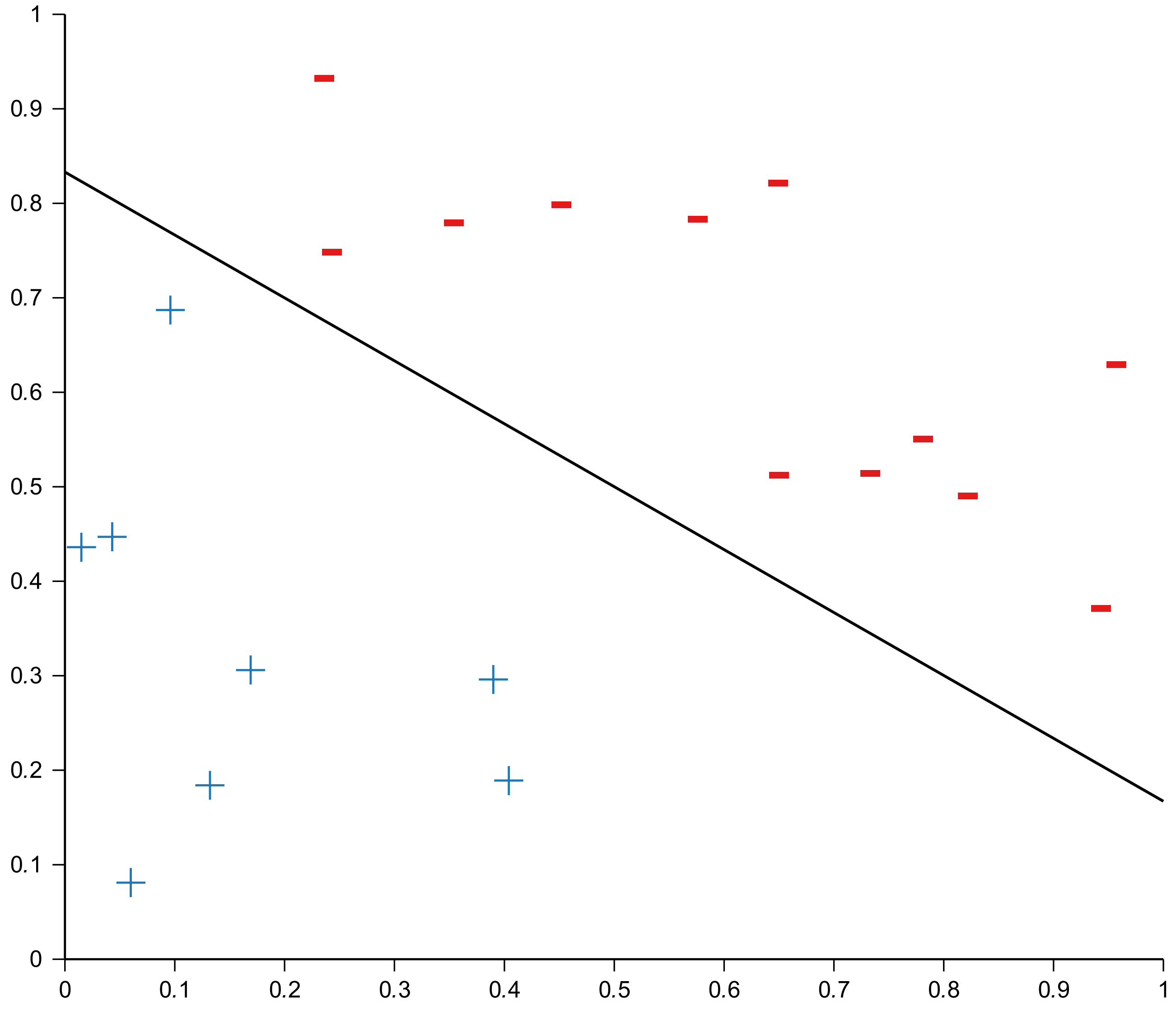


https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc/

Recall: Linear Models



Where do linear decision boundaries come from?



The equation of a line is

$\boldsymbol{w}^T \boldsymbol{x} = 0$

(bias term prepended to $\boldsymbol{w}$)

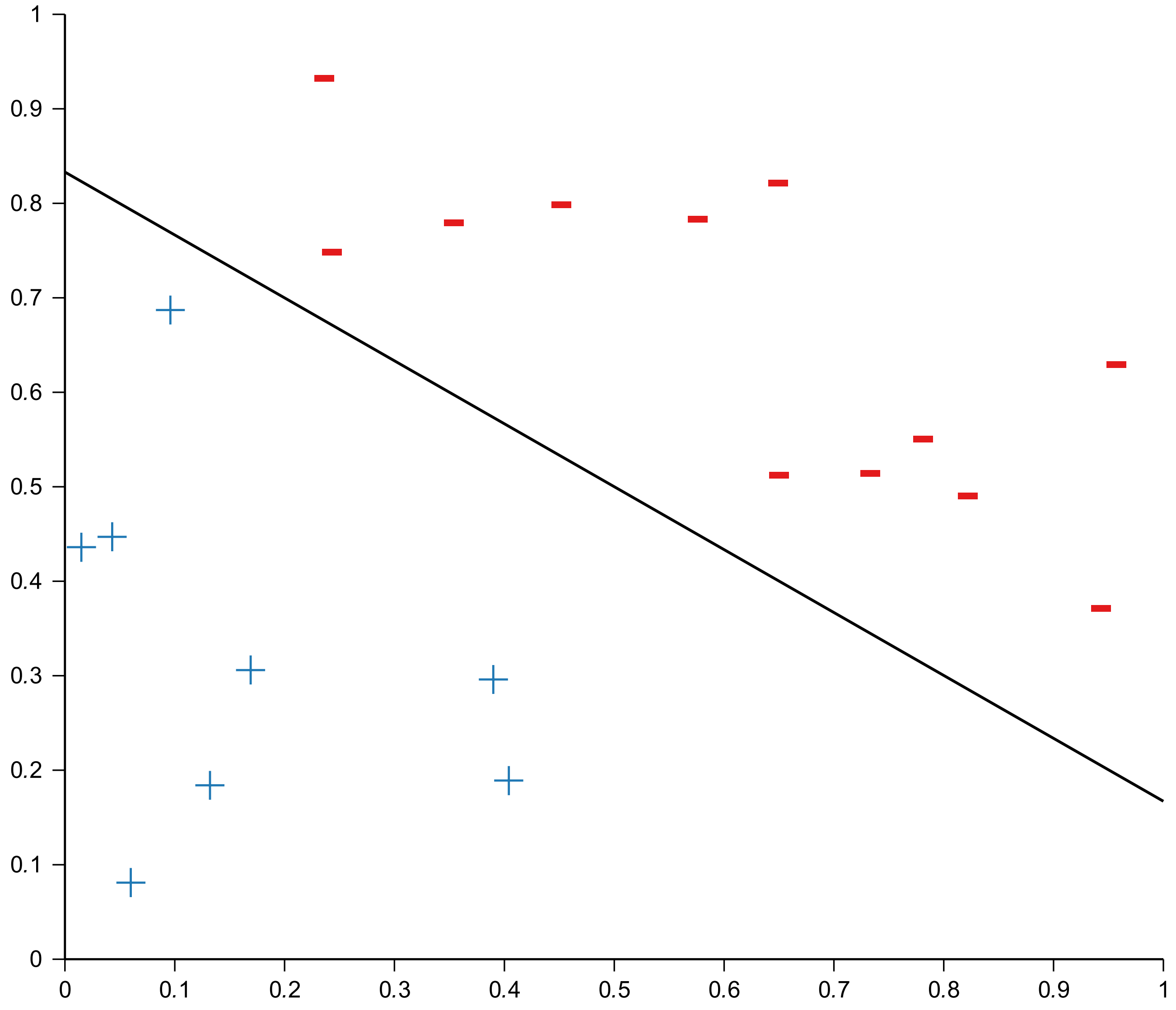
The line defines two half-spaces in $\mathbb{R}^D$:

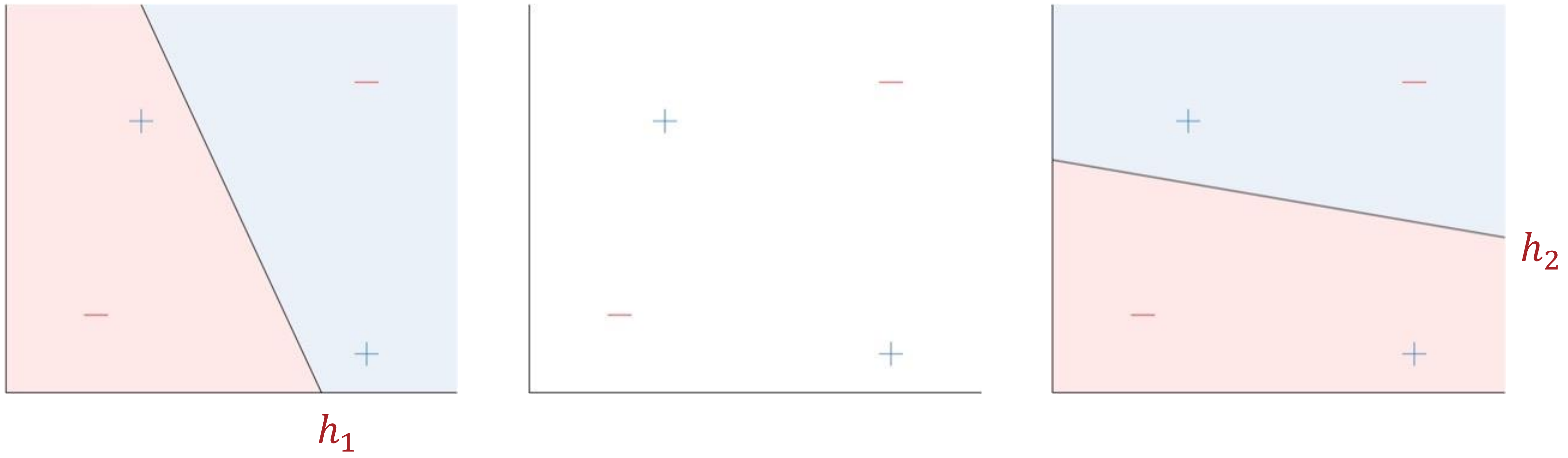
- $\mathcal{S}_+ = \{\boldsymbol{x}: \boldsymbol{w}^T \boldsymbol{x} > 0\}$
- $\mathcal{S}_- = \{\boldsymbol{x}: \boldsymbol{w}^T \boldsymbol{x} < 0\}$

So the model

$$h(x) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$$

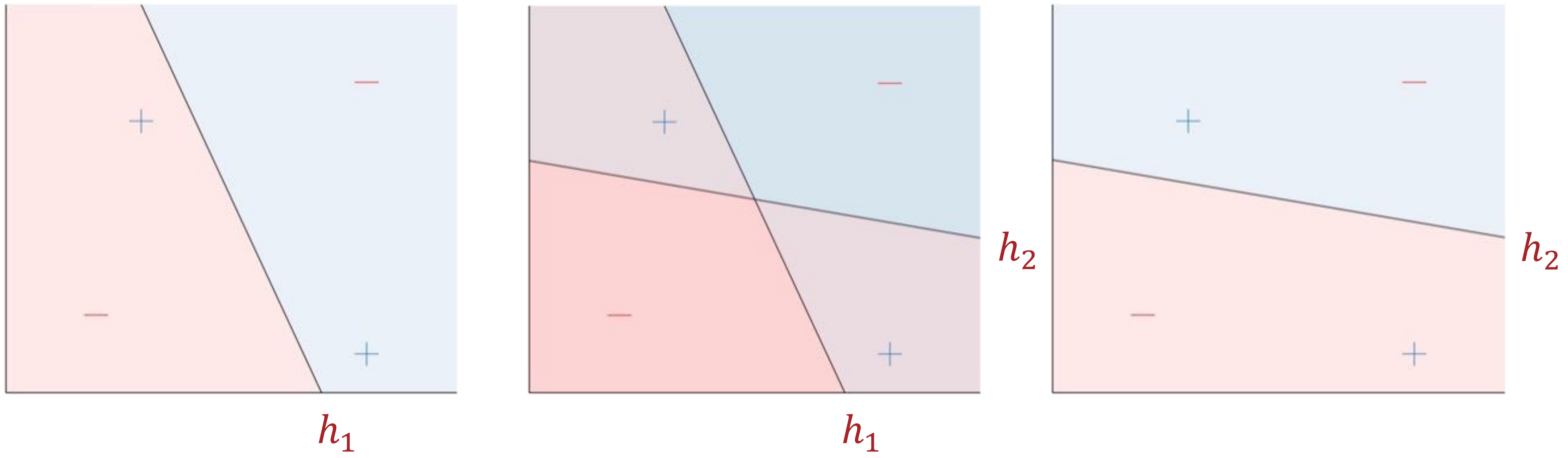
gives rise to linear decision boundaries!



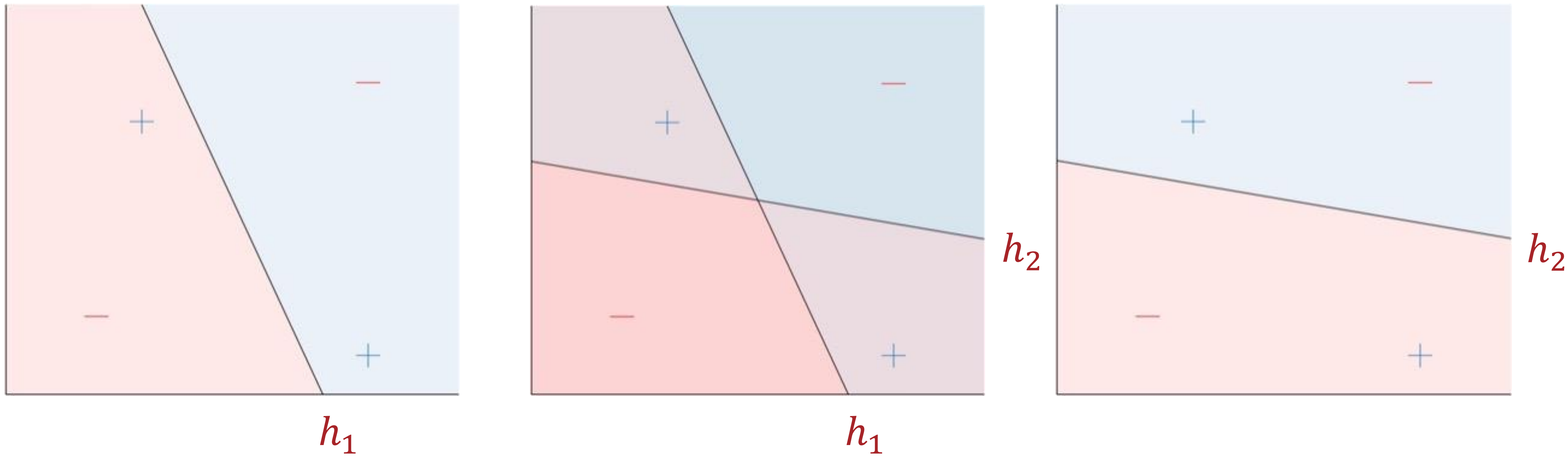


Perceptrons

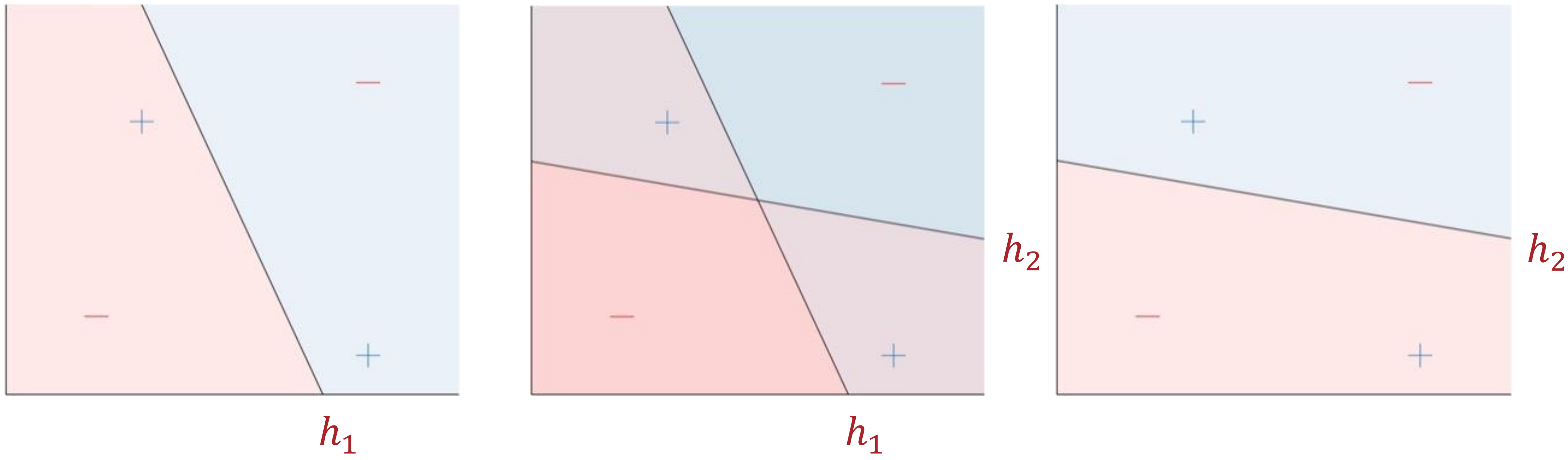
- Linear model for classification
- $h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$
- Predictions are $+1$ or -1



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

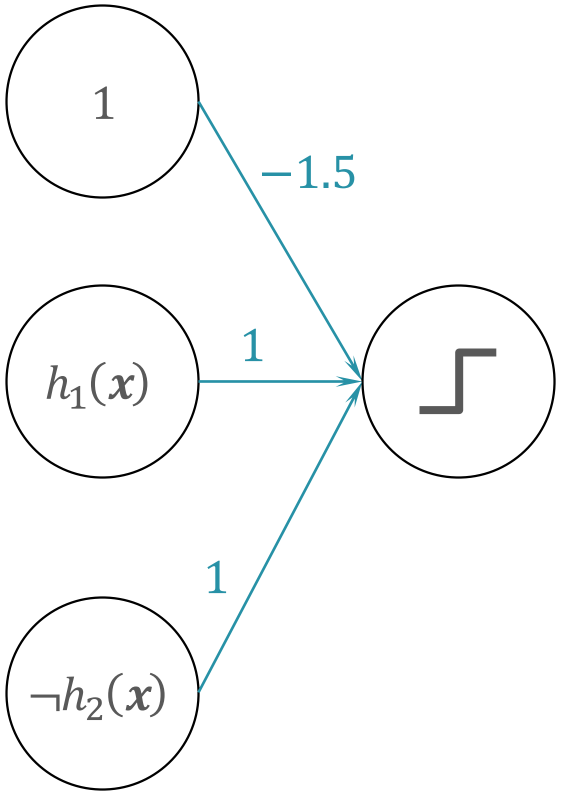
- Boolean variables are either $+1$ (”true”) or -1 (“false”)
- Basic Boolean operations
  - Negation: $\neg z = -1 * z$
  - And: $AND(z_1, z_2) = \text{sign}\left([-1.5, 1, 1]\begin{bmatrix} 1 \\ z_1 \\ z_2 \end{bmatrix}\right)$
  - Or: $OR(z_1, z_2) = \text{sign}\left([1.5, 1, 1]\begin{bmatrix} 1 \\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

Building a Network

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

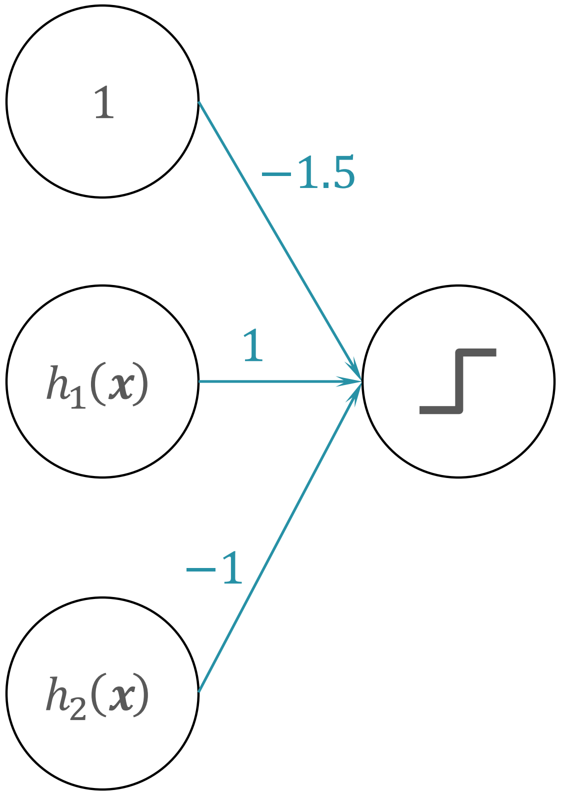
Building a Network

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



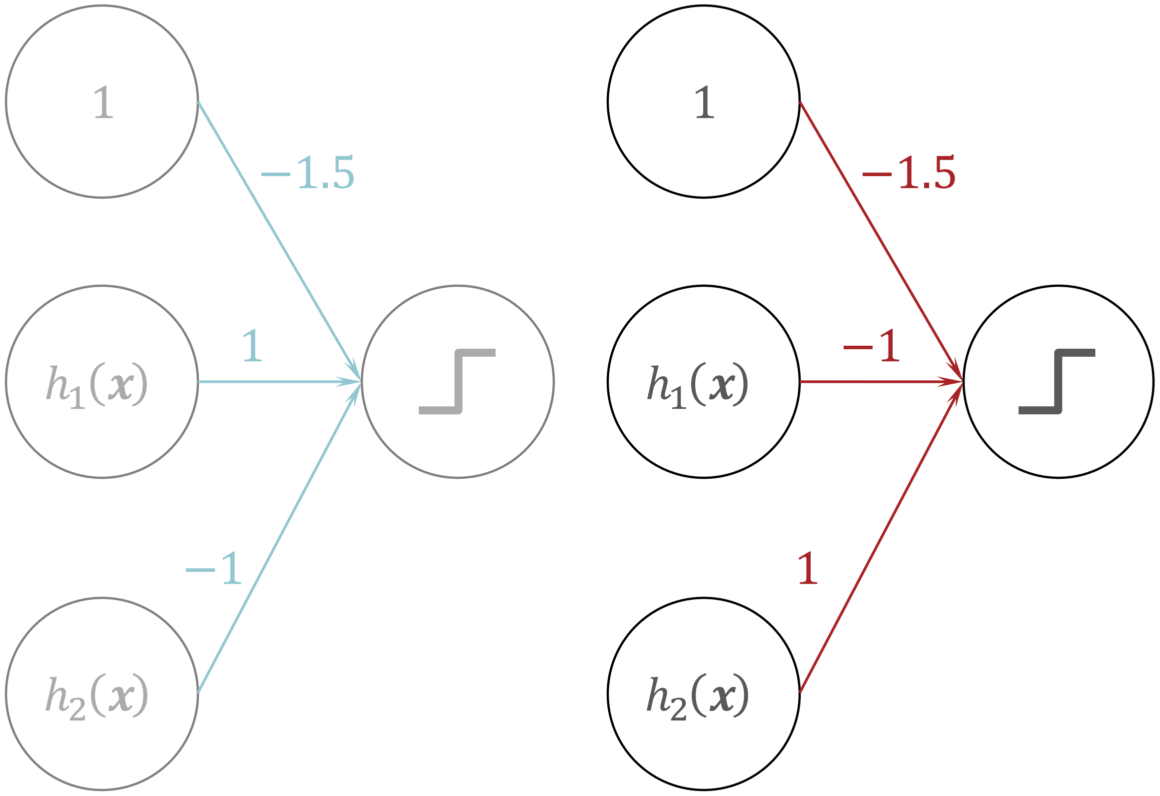
Building a Network

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



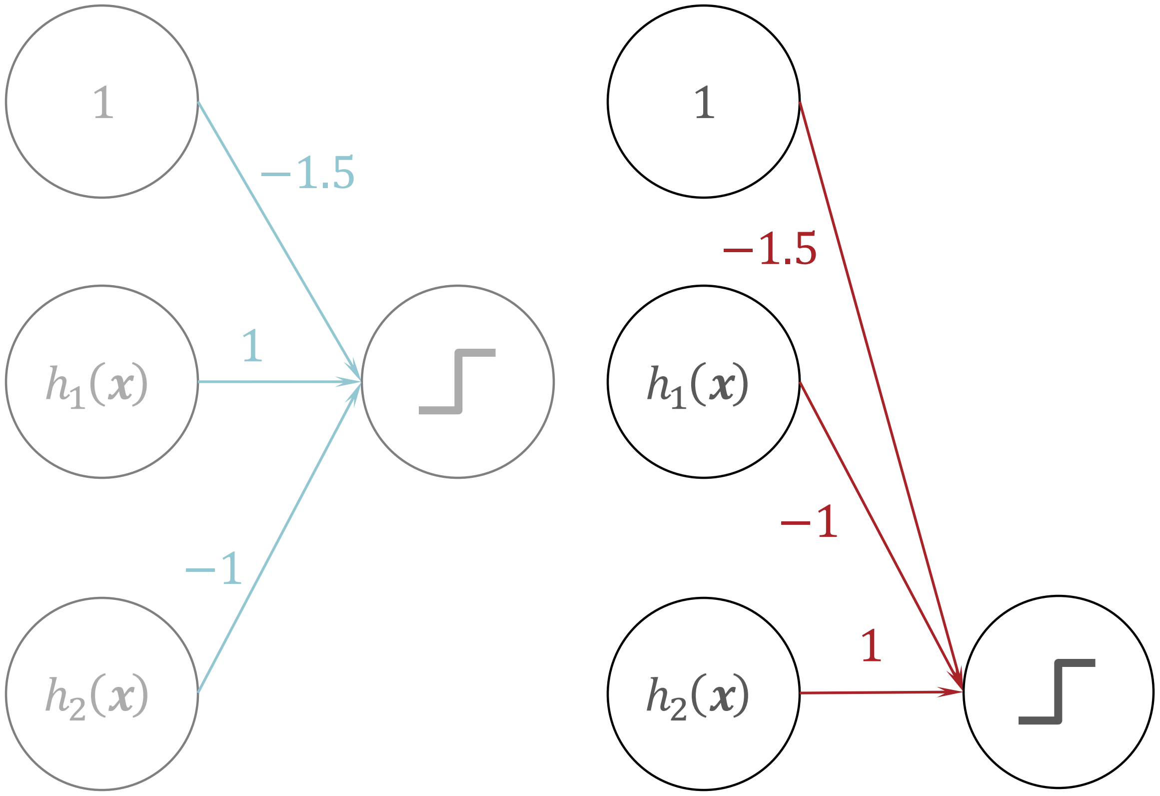
Building a Network

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



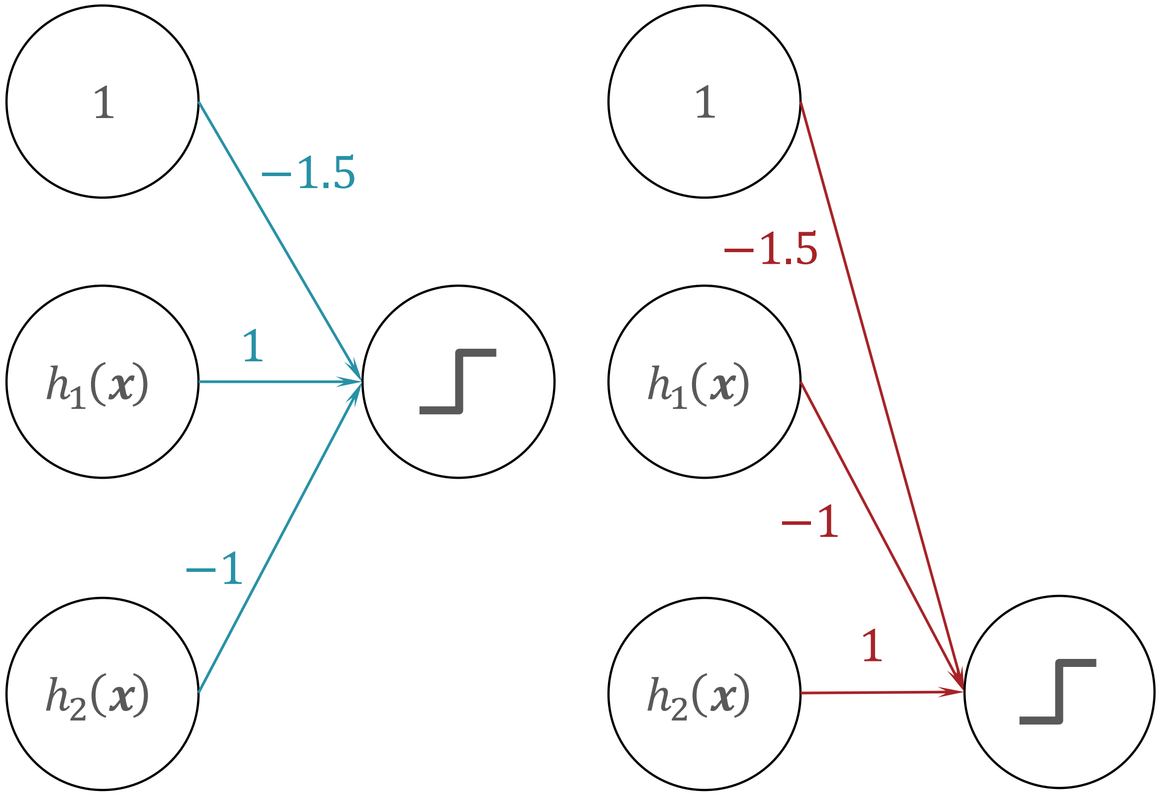
Building a Network

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



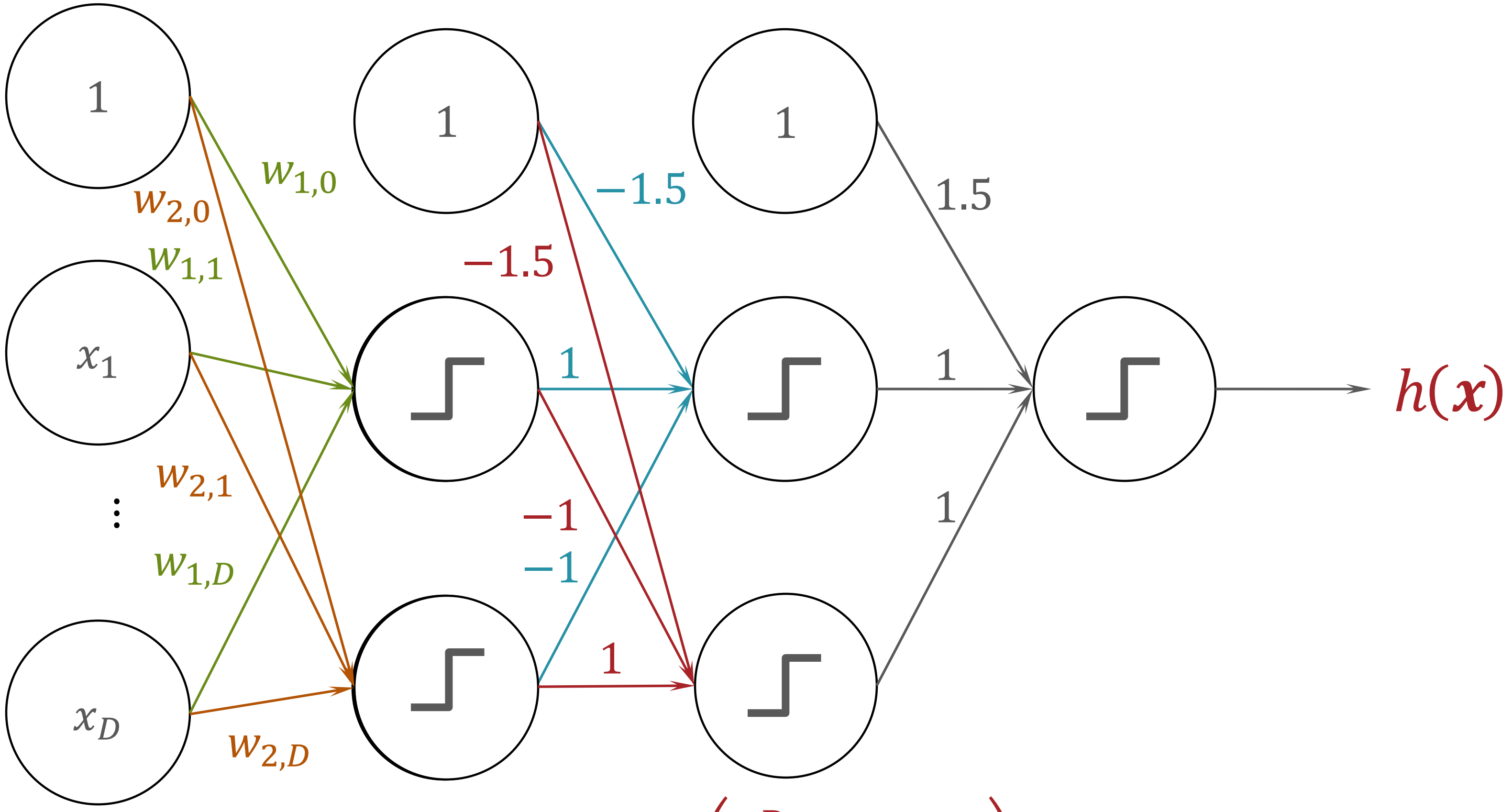
Building a Network

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



Building a Network

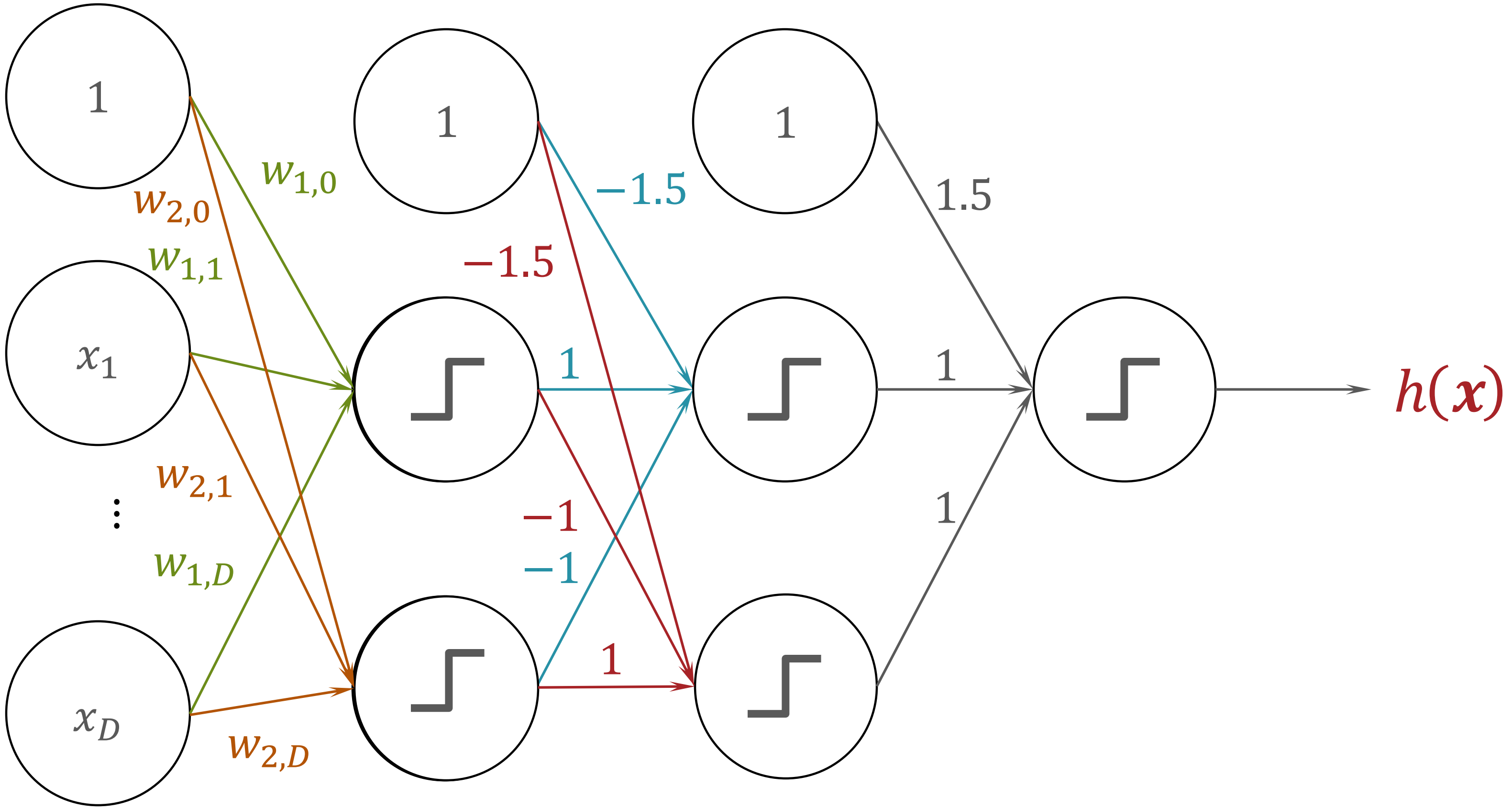
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



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

Building a Network

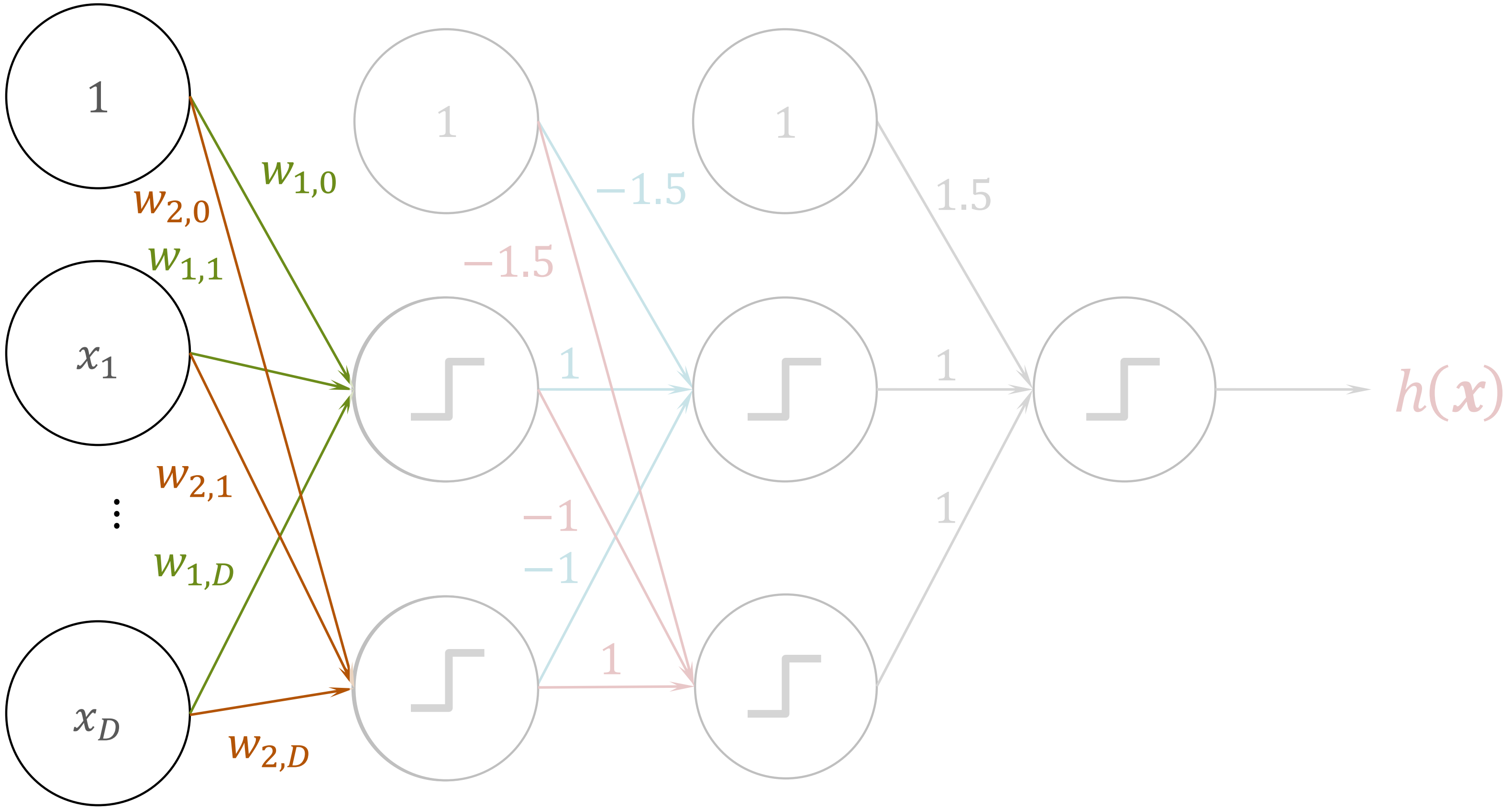
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



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

Building a Network

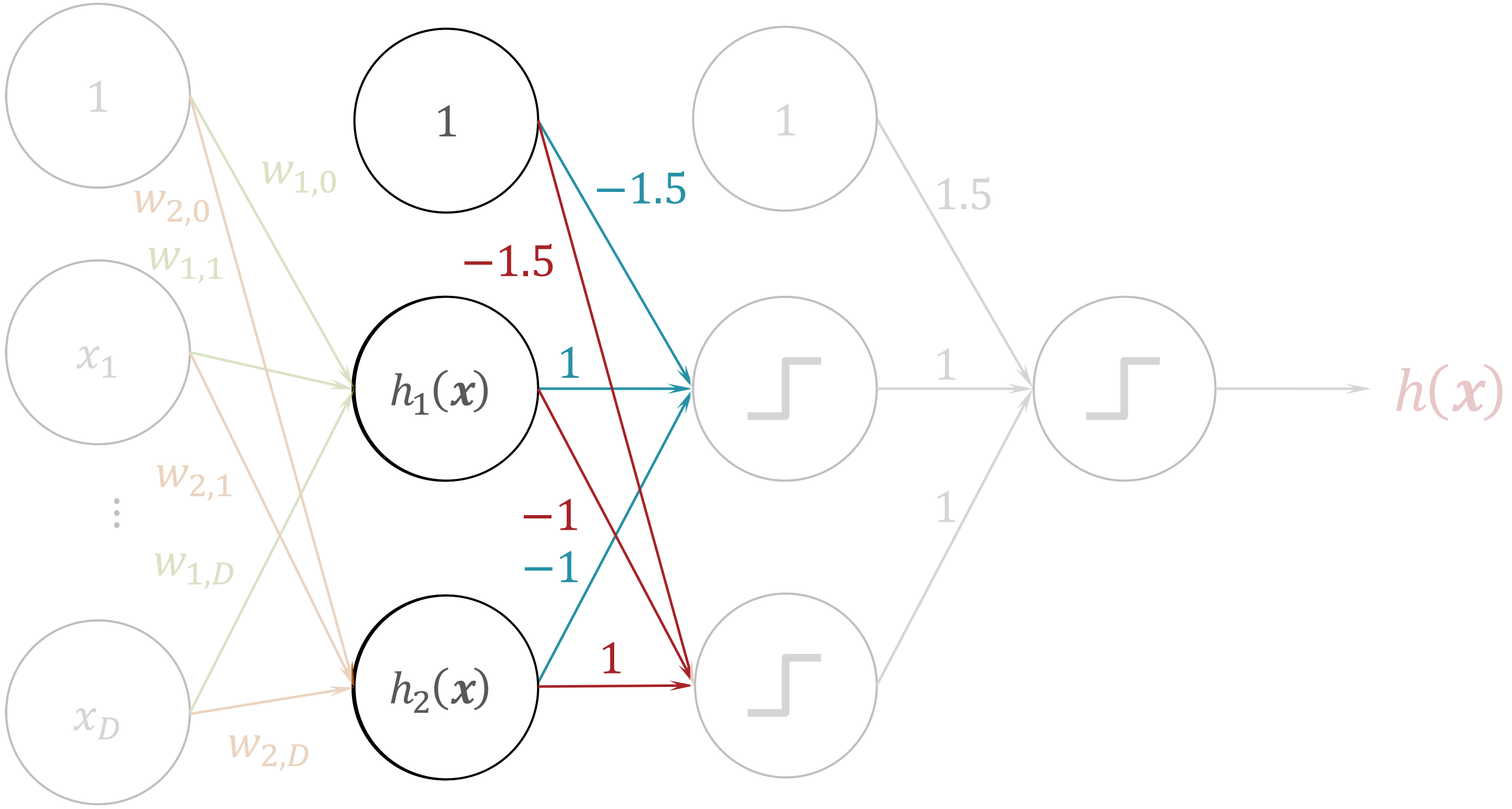
$$h(\boldsymbol{x}) = OR\Big(AND\big(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\big), AND\big(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\big)\Big)$$



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

Building a Network

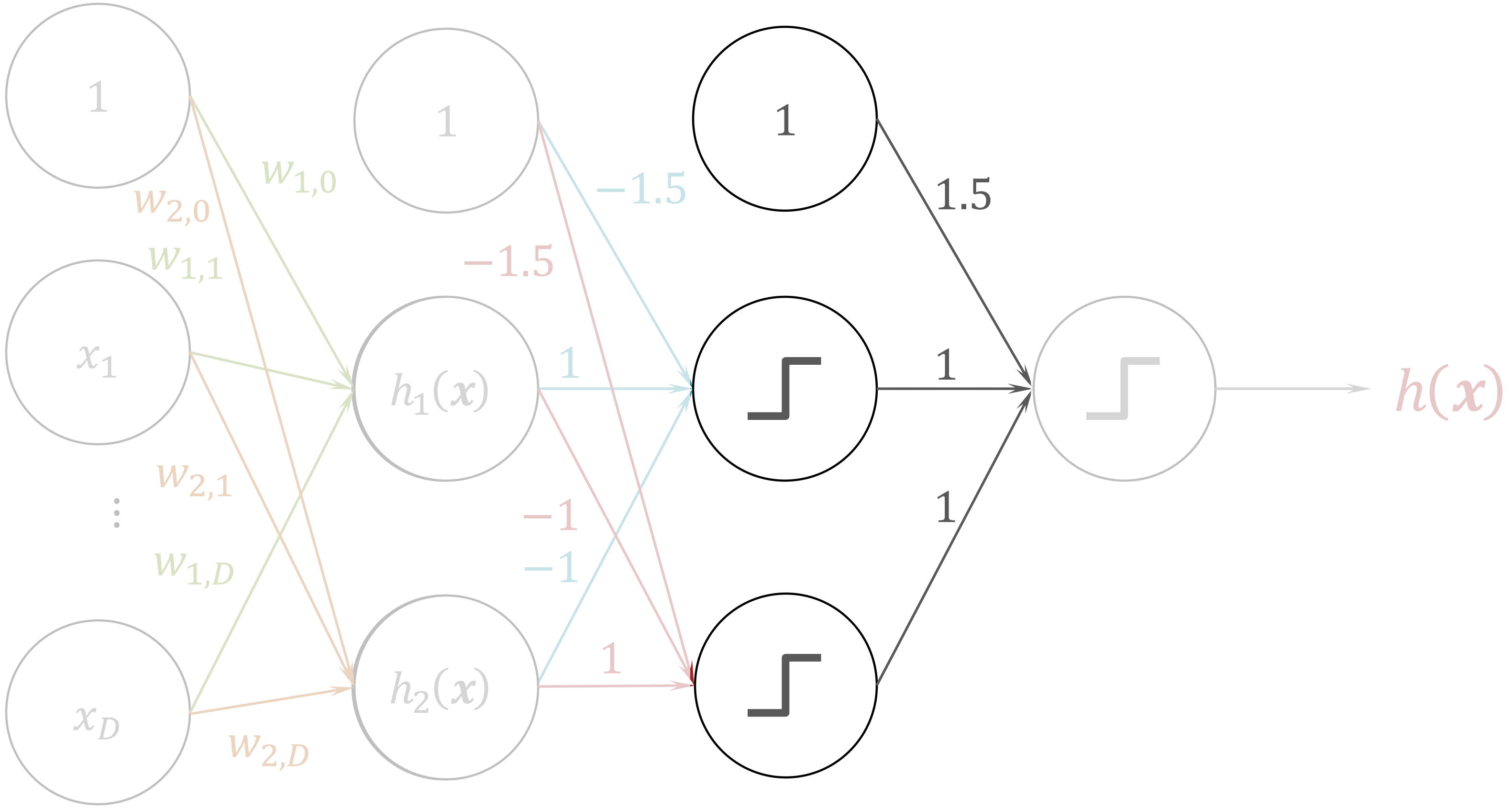
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



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

Building a Network

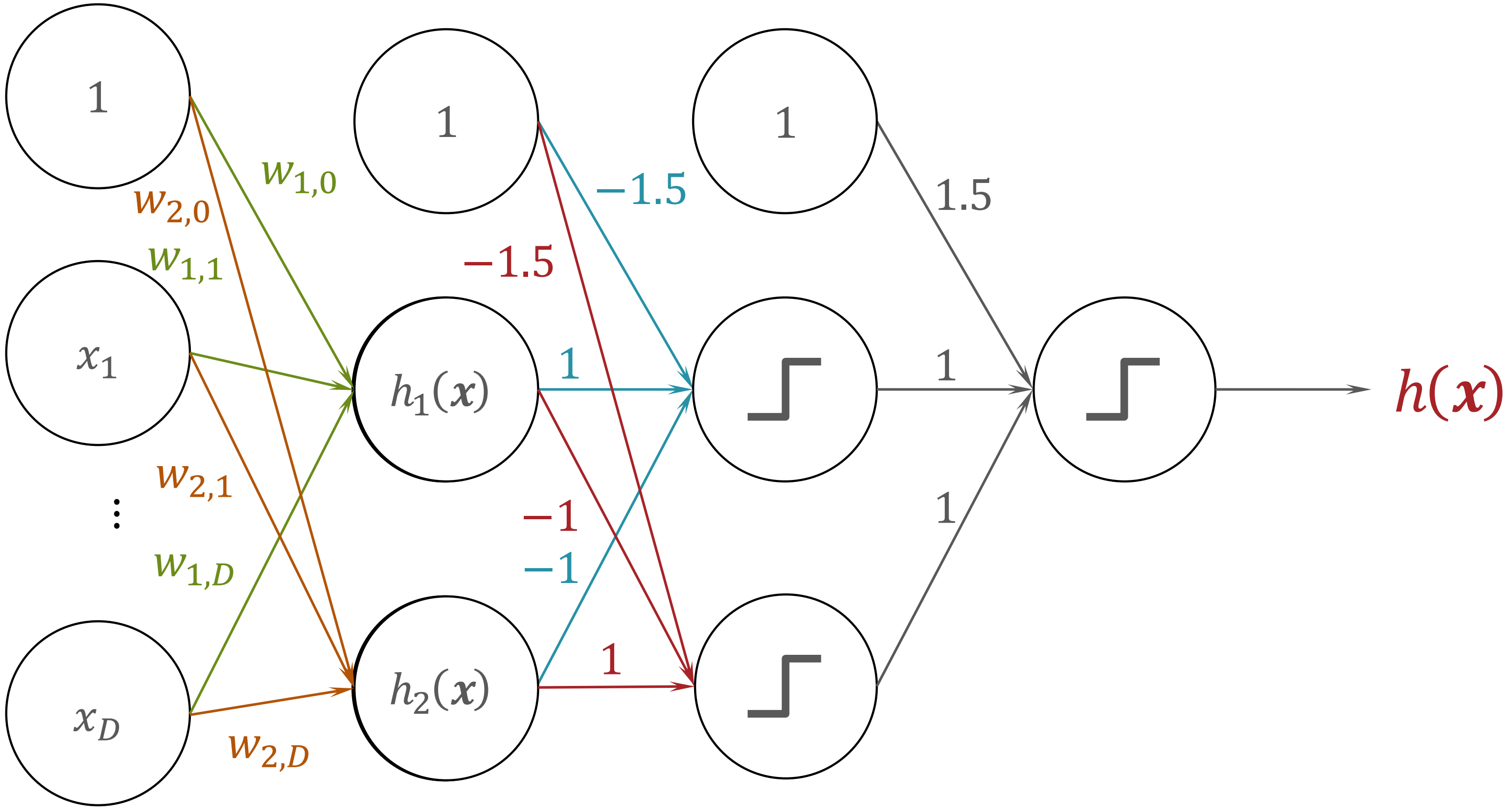
$$h(\boldsymbol{x}) = OR\left(AND\left(h_1(\boldsymbol{x}), \neg h_2(\boldsymbol{x})\right), AND\left(\neg h_1(\boldsymbol{x}), h_2(\boldsymbol{x})\right)\right)$$



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

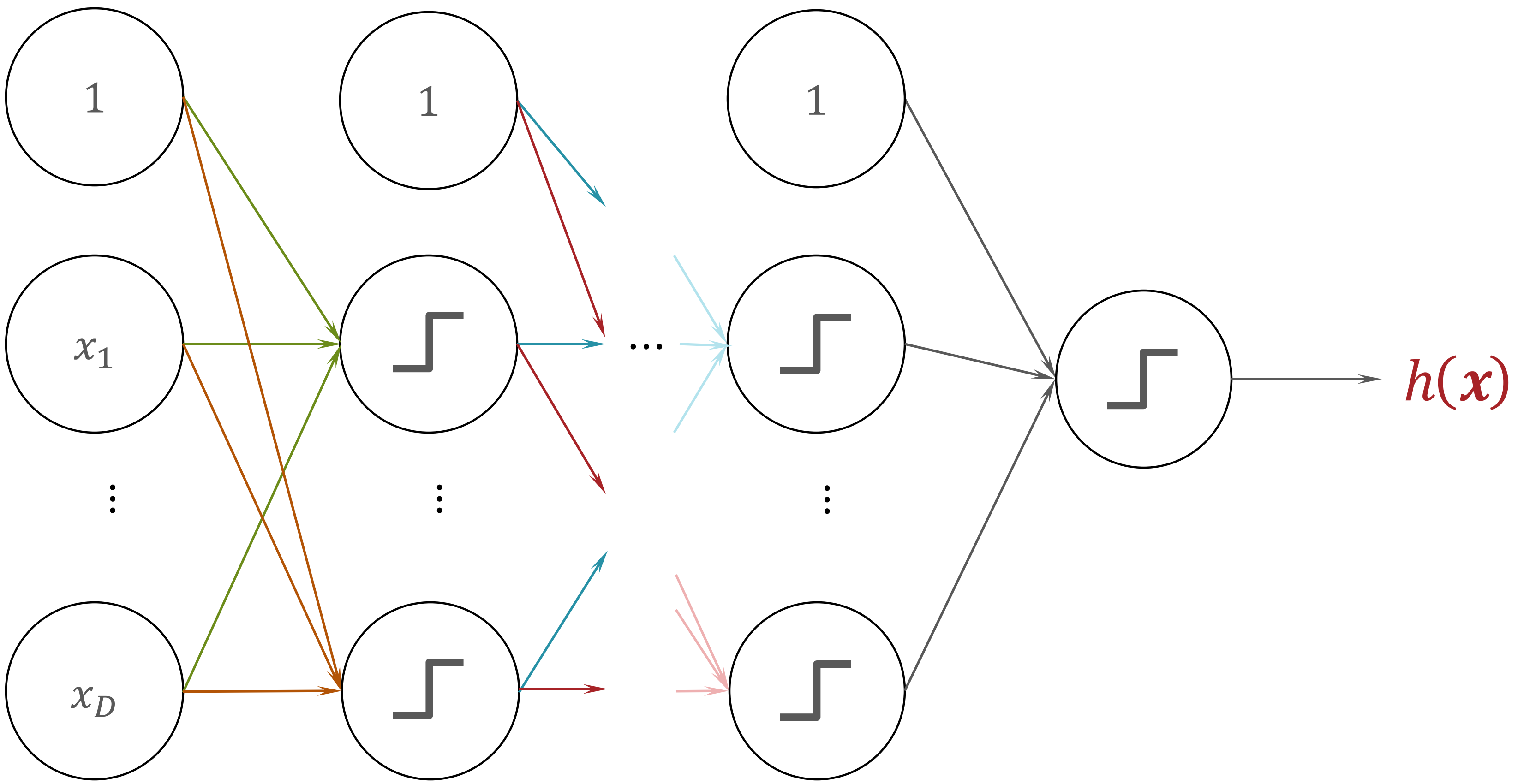
Building a Network

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

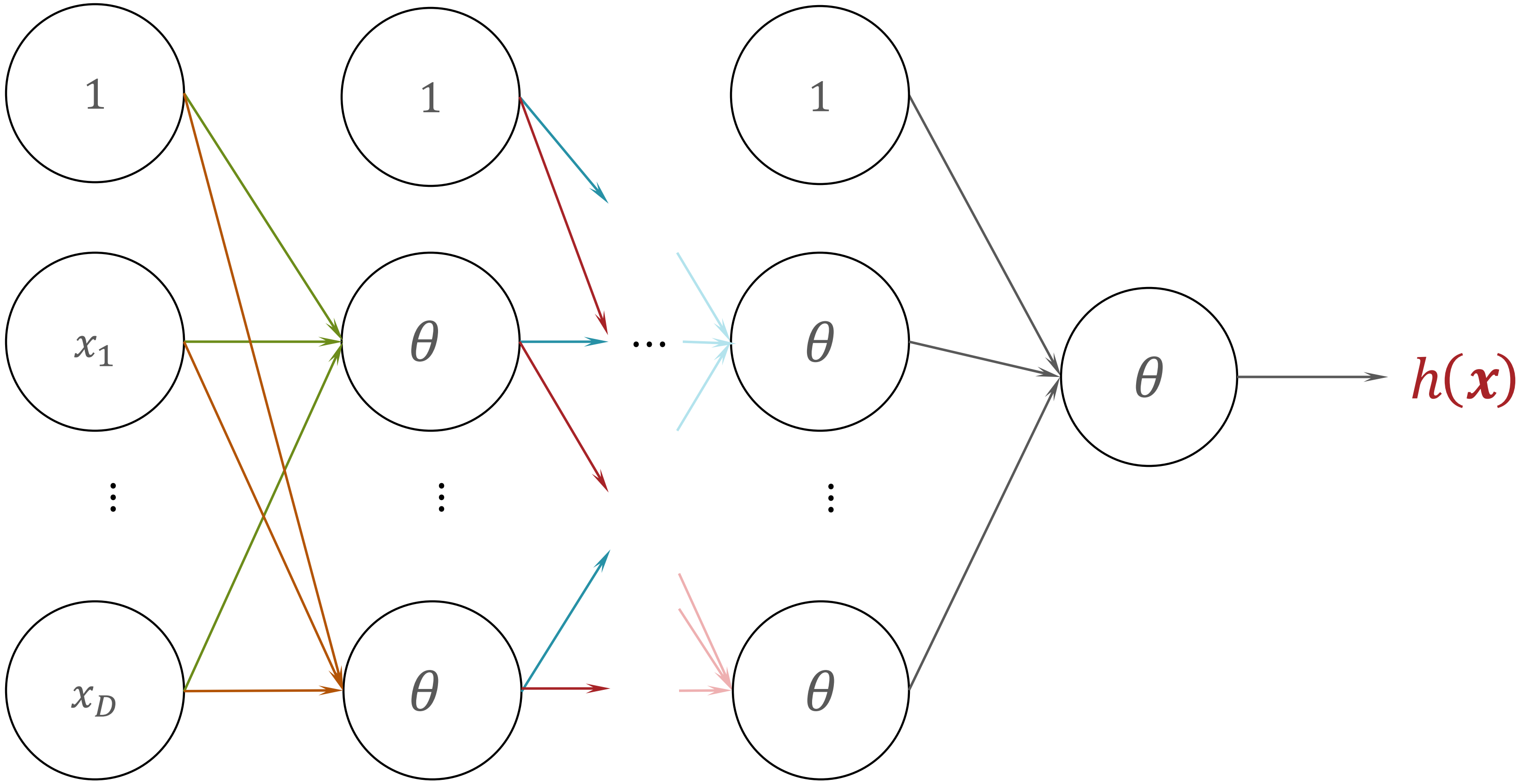


$$h(\boldsymbol{x}) = \text{sign}(\text{sign}(\text{sign}(\boldsymbol{w}_1^T\boldsymbol{x}) - \text{sign}(\boldsymbol{w}_2^T\boldsymbol{x}) - 1.5) +$$
$$\text{sign}(-\text{sign}(\boldsymbol{w}_1^T\boldsymbol{x}) + \text{sign}(\boldsymbol{w}_2^T\boldsymbol{x}) - 1.5) + 1.5)$$

Multi-Layer Perceptron (MLP)



(Fully-Connected) Feed Forward Neural Network

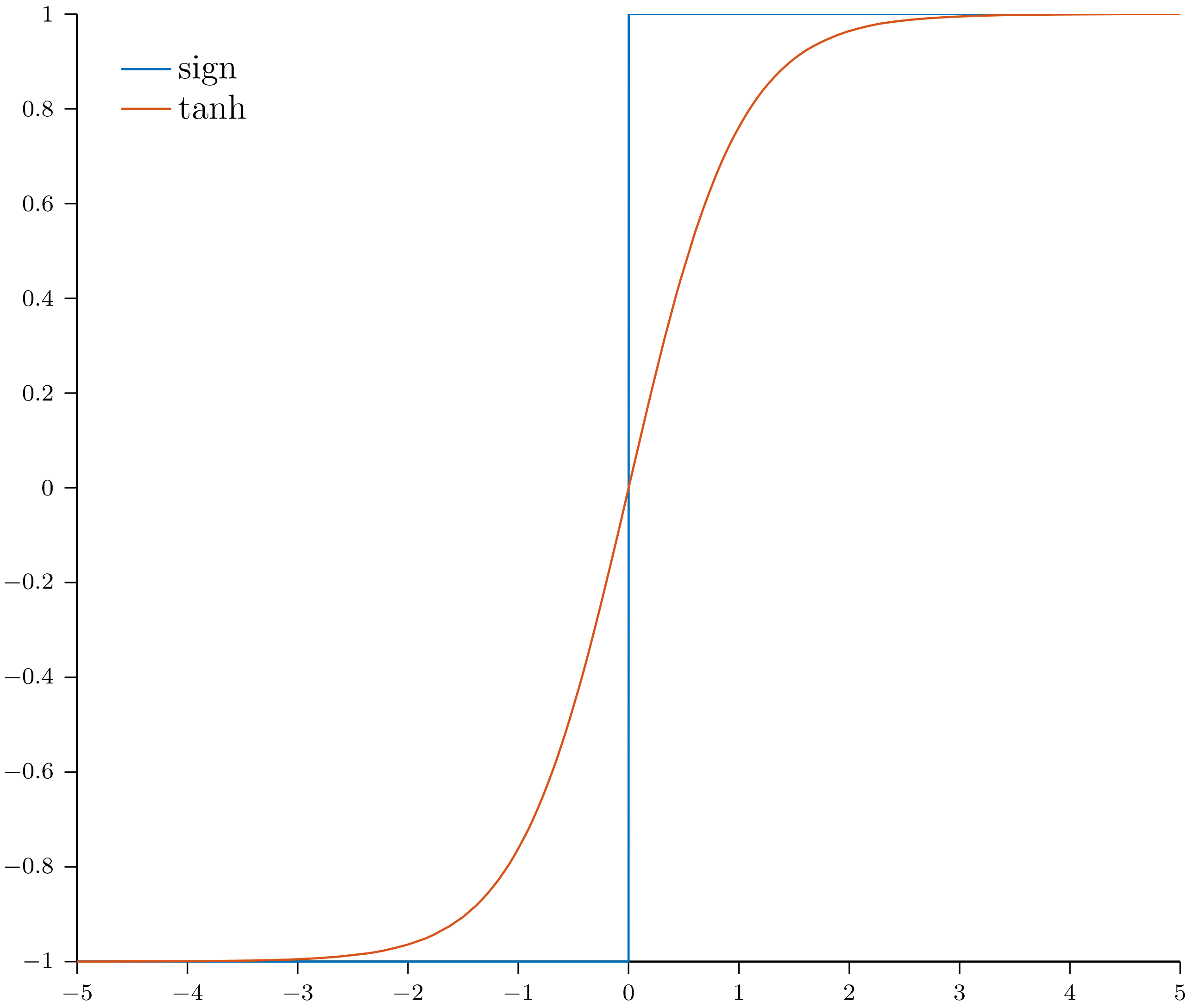


$\theta(\cdot)$

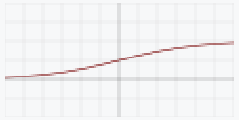
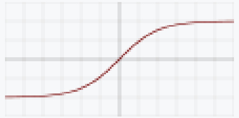
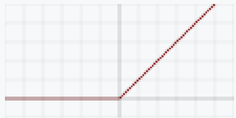
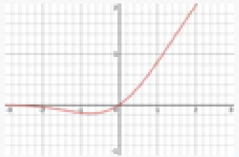
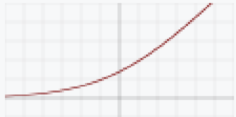
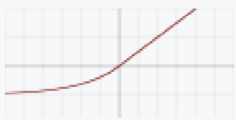
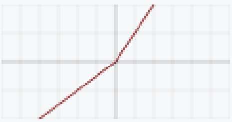
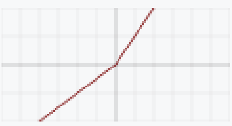
- Hyperbolic tangent:

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

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

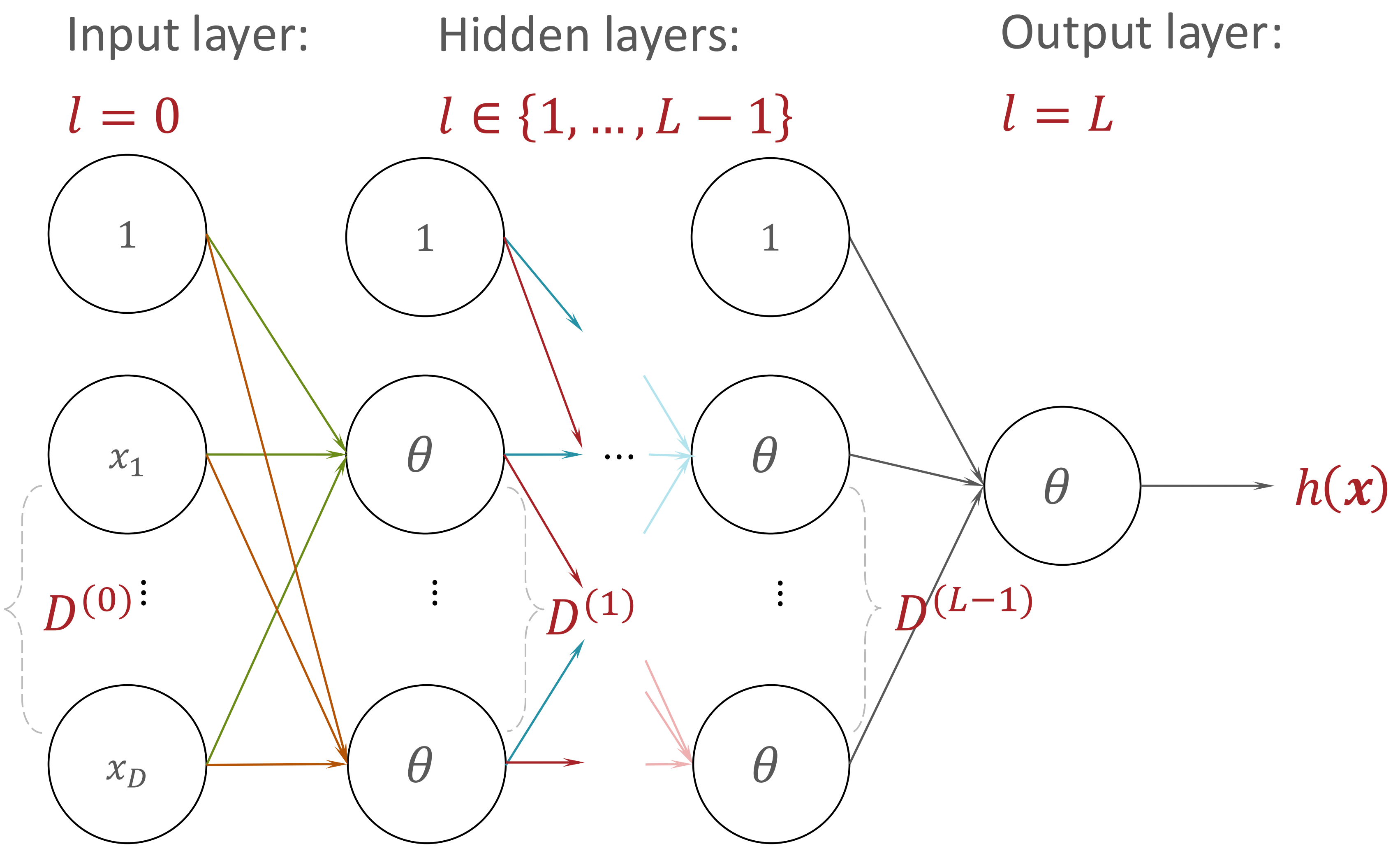


Other Activation Functions

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]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$
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]		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Leaky rectified linear unit (Leaky ReLU)[11]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Parametric rectified linear unit (PReLU)[12]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α

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

(Fully-Connected) Feed Forward Neural Network

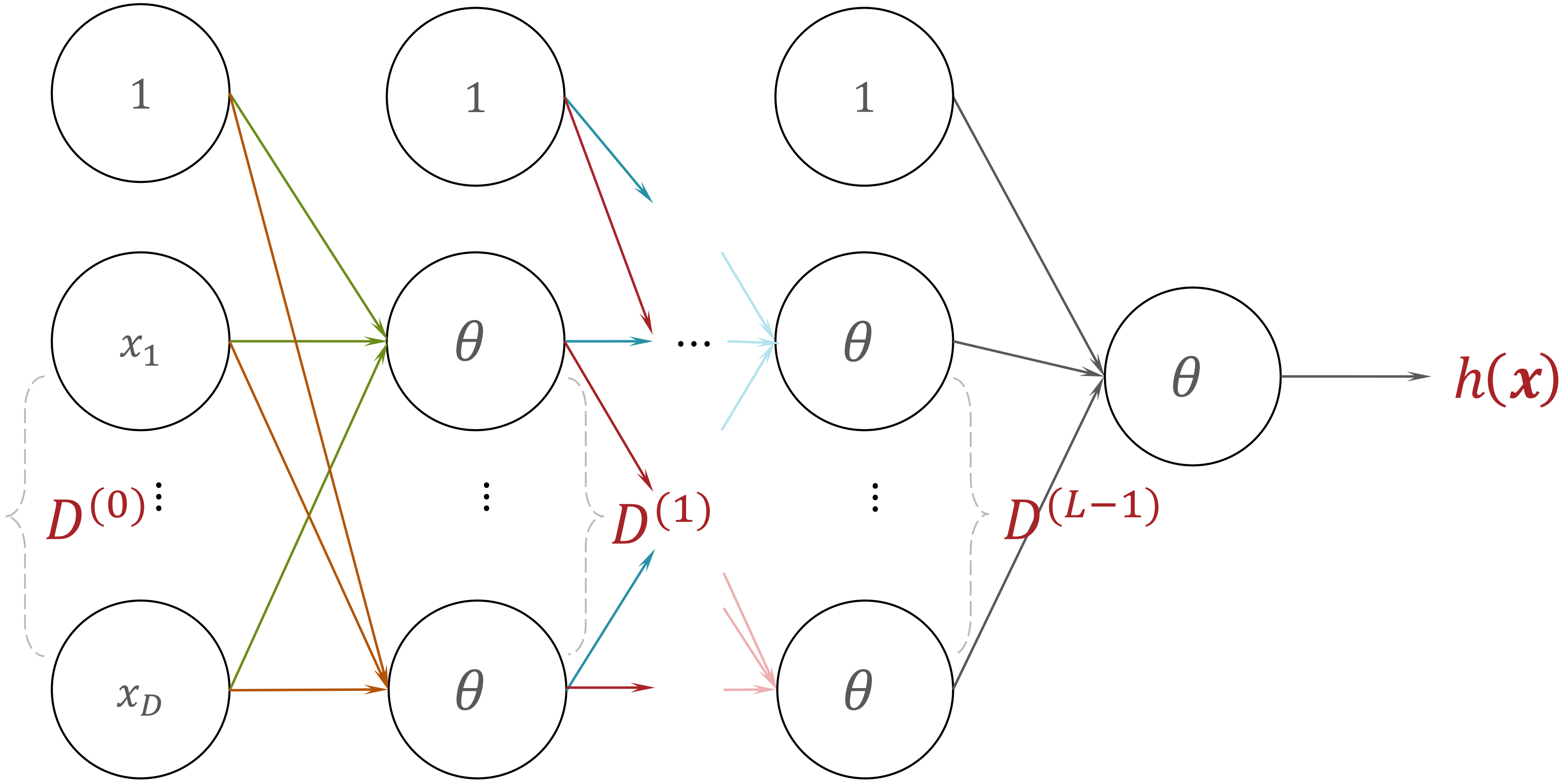


Layer l has dimension $D^{(l)}$ → Layer l has $D^{(l)} + 1$ nodes, counting the bias node

(Fully-Connected) Feed Forward Neural Network

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

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



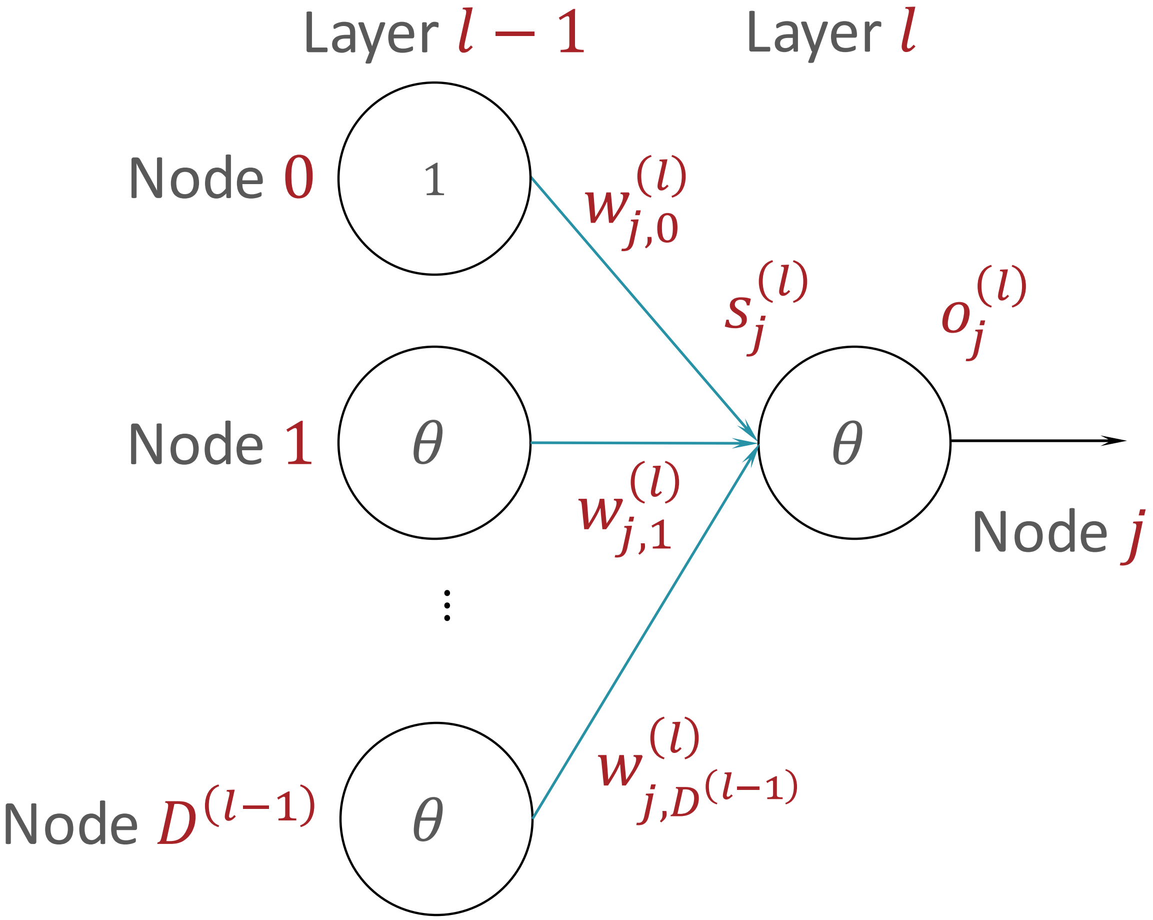
$w_{j,i}^{(l)}$ is the weight between node i in layer $l-1$ and node j in layer l

- How many hidden layers are necessary to simulate logistic regression using a NN?

- How many hidden layers are necessary to simulate linear regression using a NN?

Signal and Outputs

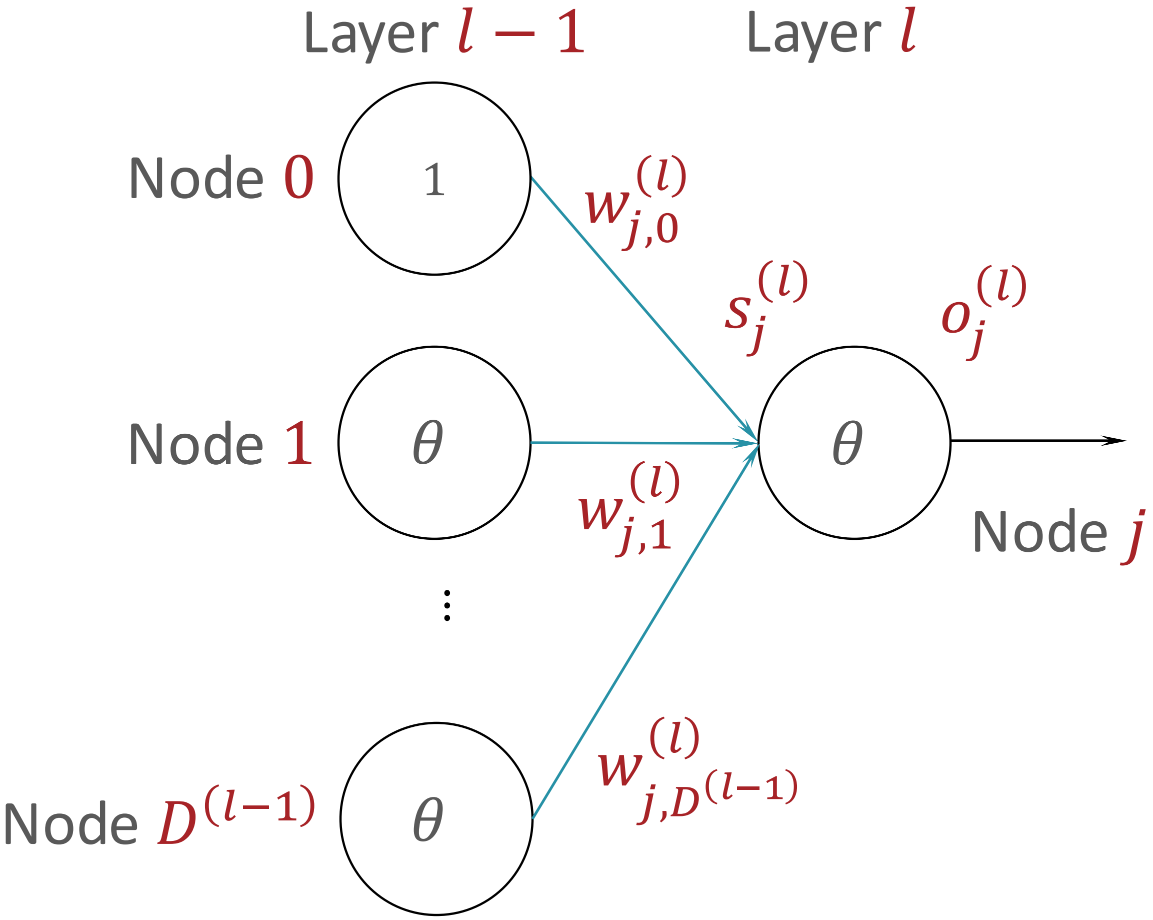
Every node has an incoming *signal* and outgoing *output*



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

Signal and Outputs

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



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

Forward Propagation for Making Predictions

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

Stochastic Gradient Descent for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta^{(0)}$
- Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
- While TERMINATION CRITERION is not satisfied
  - For $i \in \text{shuffle}(\{1, \ldots, N\})$
    - For $l = 1, \ldots, L$
      - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell^{(i)}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right)$
      - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta^{(0)} G^{(l)}$
    - Increment t: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}$

Two questions:

1. What is this loss function $\ell^{(i)}$?

2. How on earth do we take these gradients?

- Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta^{(0)}$
- 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 $i \in \text{shuffle}(\{1, \dots, N\})$
    - For $l = 1, \dots, L$
      - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell^{(i)} \left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$ (???)
      - 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!)

$$\ell^{(i)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = \left(h_{W^{(1)},\dots,W^{(L)}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

Loss Functions for Neural Networks

- Binary classification - cross-entropy loss
  - measures the difference between two probability distributions:
    - The **true distribution** (the labels, usually one-hot encoded).
    - The **predicted distribution** (the model's output probabilities)
  - penalizes the model more when it assigns low probability to the correct class.
  - Example: Binary classification, one instance with true label y and predicted probability p

Loss Functions for Neural Networks

- Binary classification - cross-entropy loss

Loss Functions for Neural Networks

- Binary classification - cross-entropy loss
  - Assume $P\left(Y=1 \middle| \boldsymbol{x}, W^{(1)}, \dots, W^{(L)}\right) = h_{W^{(1)},\dots,W^{(L)}}(\boldsymbol{x})$

$$\ell^{(i)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = -\log P\left(y^{(i)} | \boldsymbol{x}^{(i)}, W^{(1)}, \dots, W^{(L)}\right)$$

$$= -\log\left(h_{W^{(1)},\dots,W^{(L)}}\left(\boldsymbol{x}^{(i)}\right)^{y^{(i)}} \left(1 - h_{W^{(1)},\dots,W^{(L)}}\left(\boldsymbol{x}^{(i)}\right)\right)^{1-y^{(i)}}\right)$$

$$= -\, y^{(i)} \log\left(h_{W^{(1)},\dots,W^{(L)}}\left(\boldsymbol{x}^{(i)}\right)\right)$$

$$-\left(1 - y^{(i)}\right)\log\left(1 - h_{W^{(1)},\dots,W^{(L)}}\left(\boldsymbol{x}^{(i)}\right)\right)$$

Loss Functions for Neural Networks

- Multi-class classification - also the 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 neural network output is also a vector of length C

$$P\left(y[c] = 1 \middle| \boldsymbol{x}, W^{(1)}, \dots, W^{(L)}\right) = h_{W^{(1)},\dots,W^{(L)}}(\boldsymbol{x})[c]$$

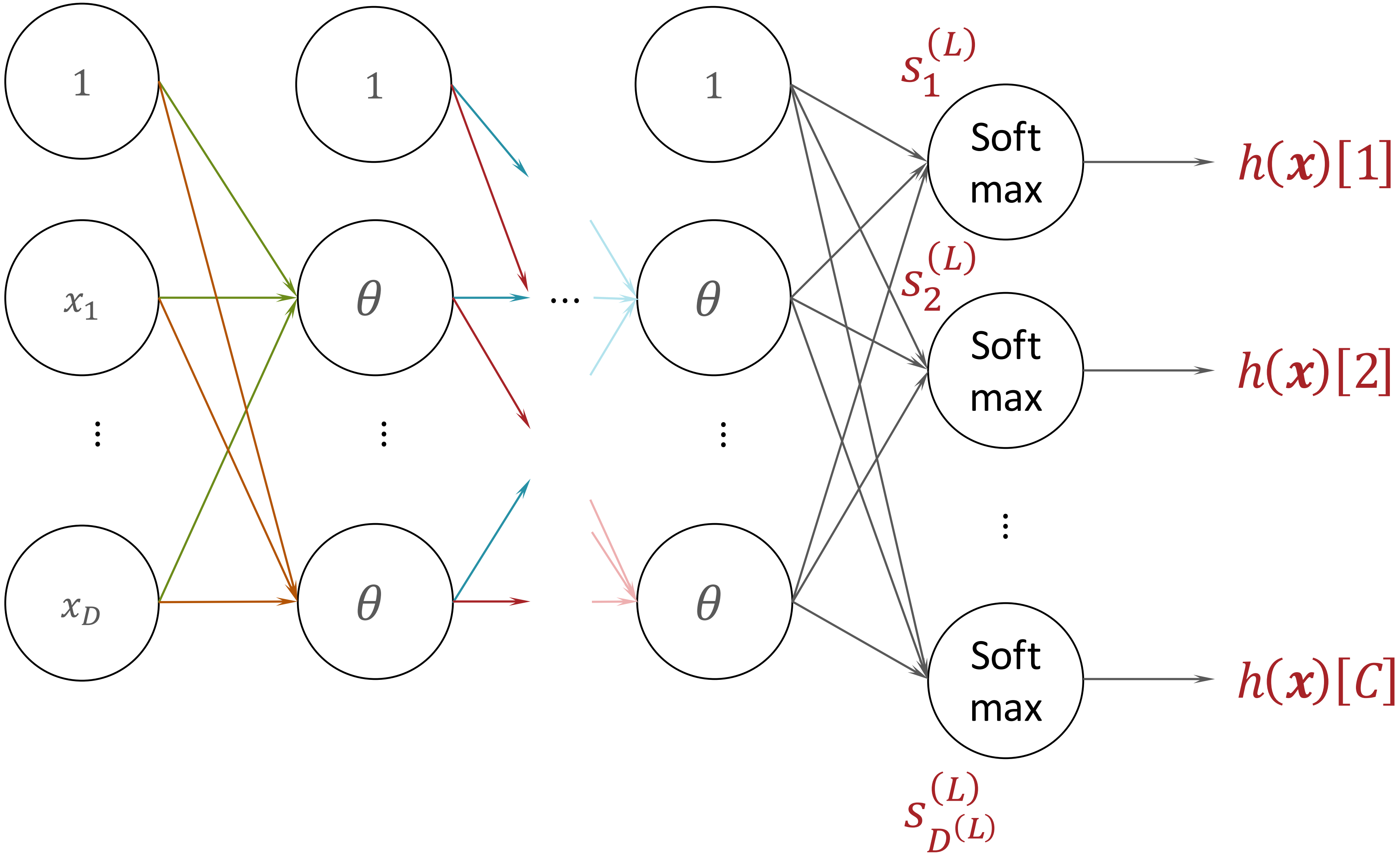
  - Then the cross-entropy loss is

$$\ell^{(i)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = -\log P\left(y^{(i)} | \boldsymbol{x}^{(i)}, W^{(1)}, \dots, W^{(L)}\right)$$

$$= -\sum_{c=1}^{C} y[c] \log h_{W^{(1)},\dots,W^{(L)}}\left(\boldsymbol{x}^{(n)}\right)[c]$$

Multi-dimensional Outputs

$$h(\boldsymbol{x})[c] = \frac{\exp s_c^{(L)}}{\sum_{k=1}^{C} s_k^{(L)}}$$



Key Takeaways

- Many common machine learning models can be represented as neural networks.
- 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