

10-701: Introduction to Machine Learning

Lecture 9 – Neural Networks

Hoda Heidari

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

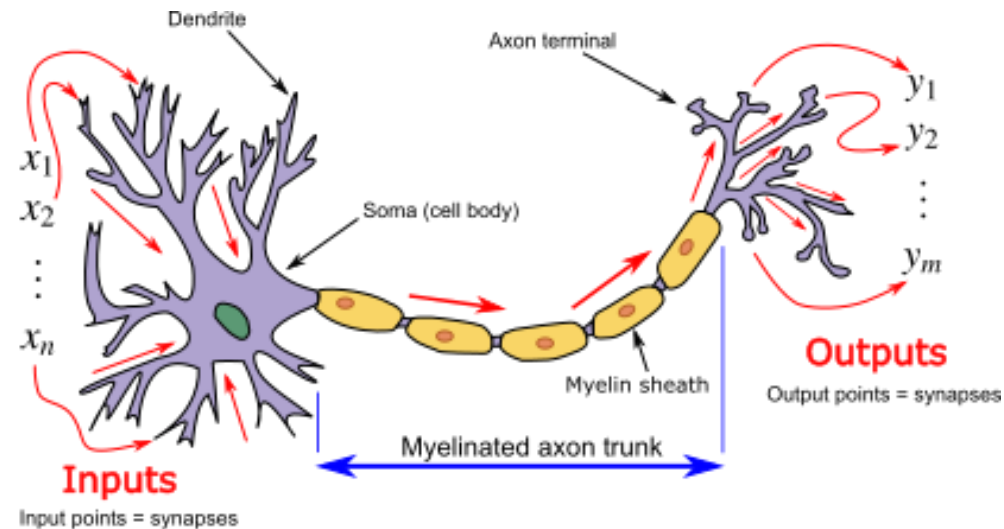
Biological Neural Network



Biological Neurons

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

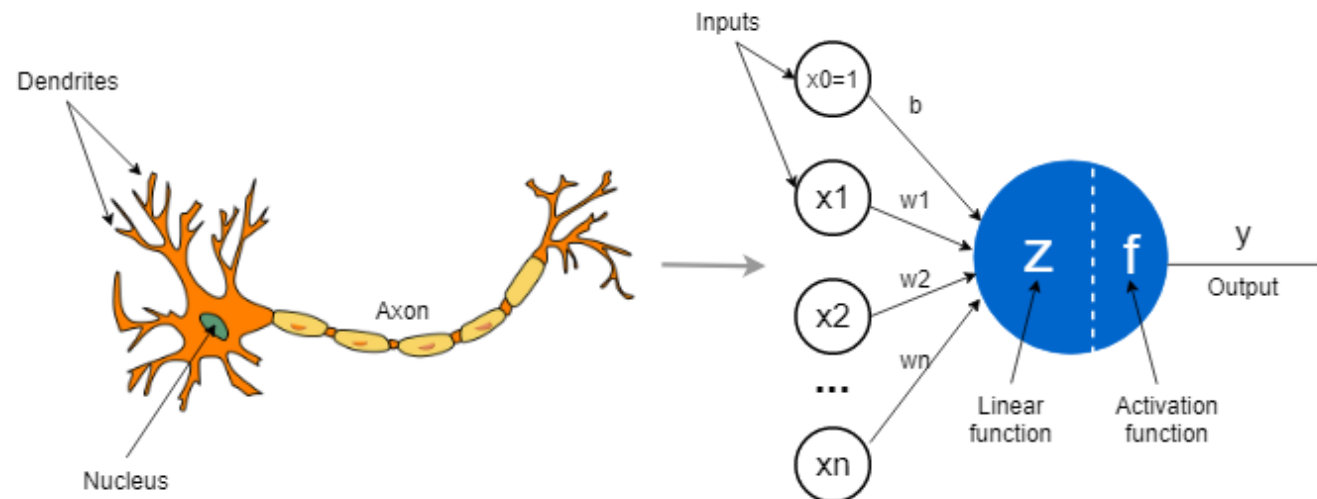
- Dendrites receive signals from other neurons.
- These signals are **combined** in the cell body through synapses.
- If the total signal is **strong enough**, the nucleus “fires” an electrical impulse down its axon to pass the message on to other neurons.



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

Artificial Neurons/ Perceptrons

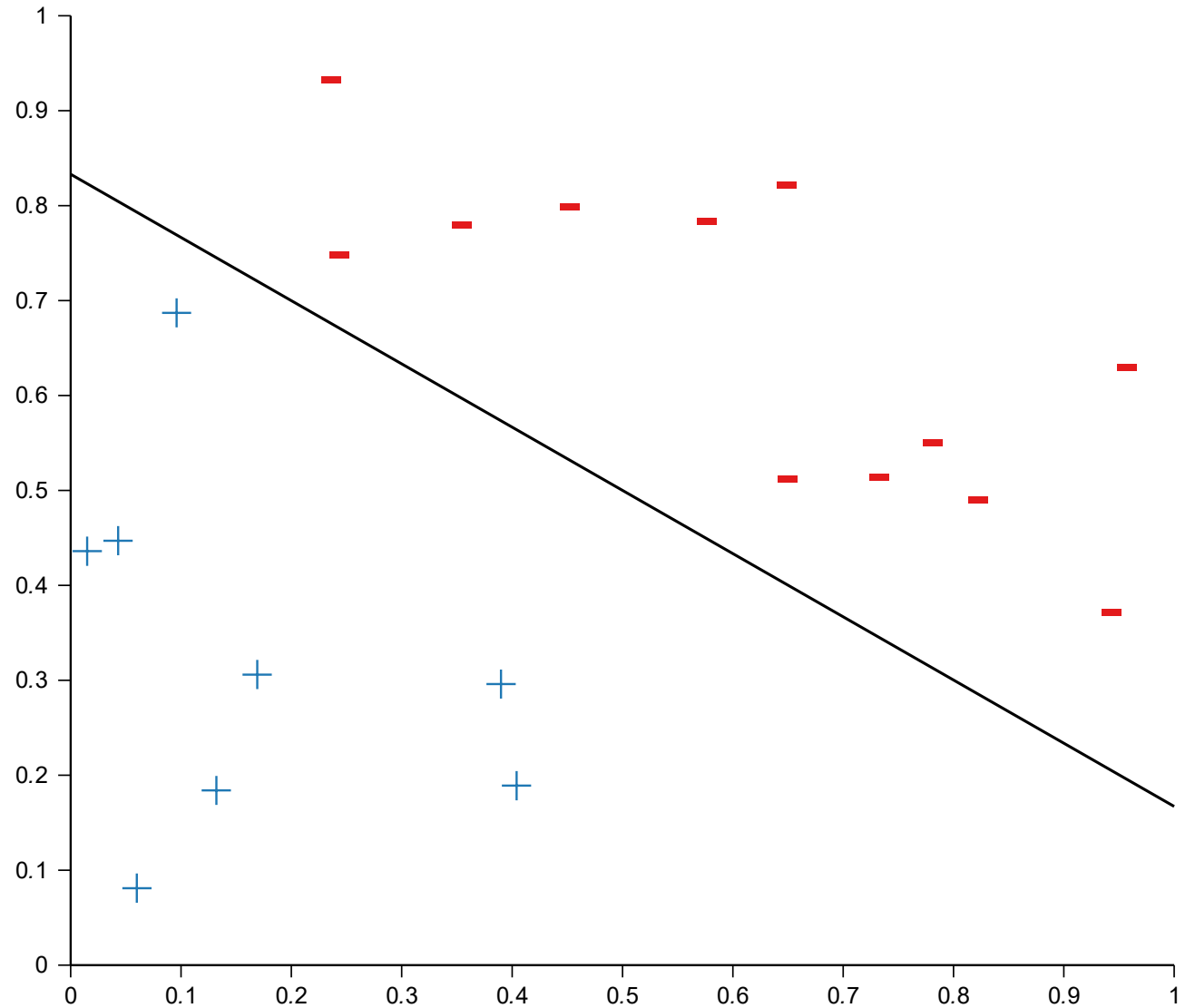
- **Inputs** come into the neuron, each multiplied by a **weight** (importance).
- The neuron adds these up and often adds a **bias**.
- It then passes the sum through an **activation function** (a mathematical rule that decides if the neuron “fires” and how strongly).
- The **output** is then sent to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



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

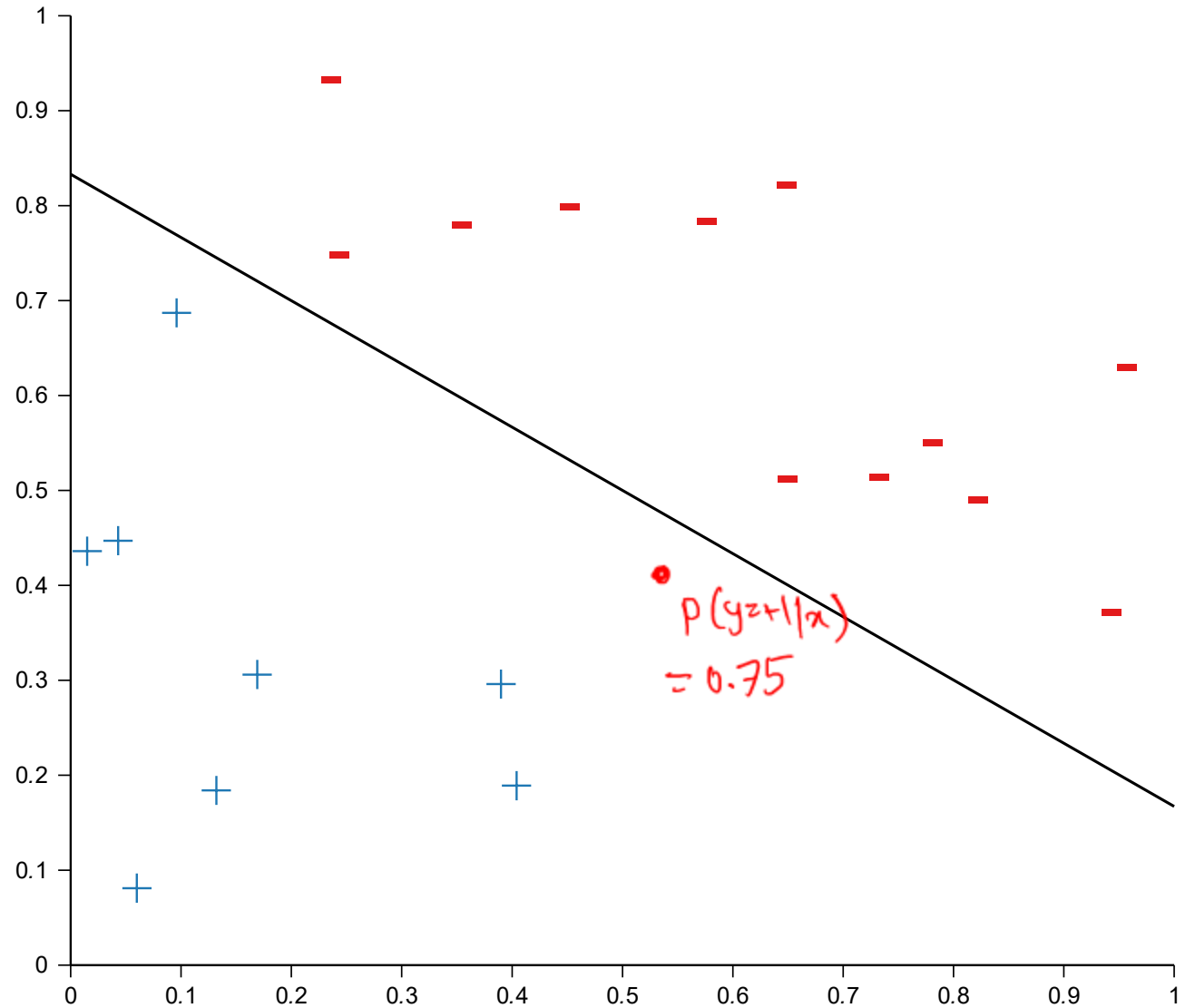
Recall: Linear Models

logistic regression: $\text{logistic}(w^T x)$
 $= p(y=1 | x, w)$



Where do linear decision boundaries come from?

$$\hat{y} = \begin{cases} +1 & \text{if } h(x) \geq 0.5 \\ -1 & \text{otherwise} \end{cases}$$



The equation of a line is

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

(bias term prepended to \mathbf{w})

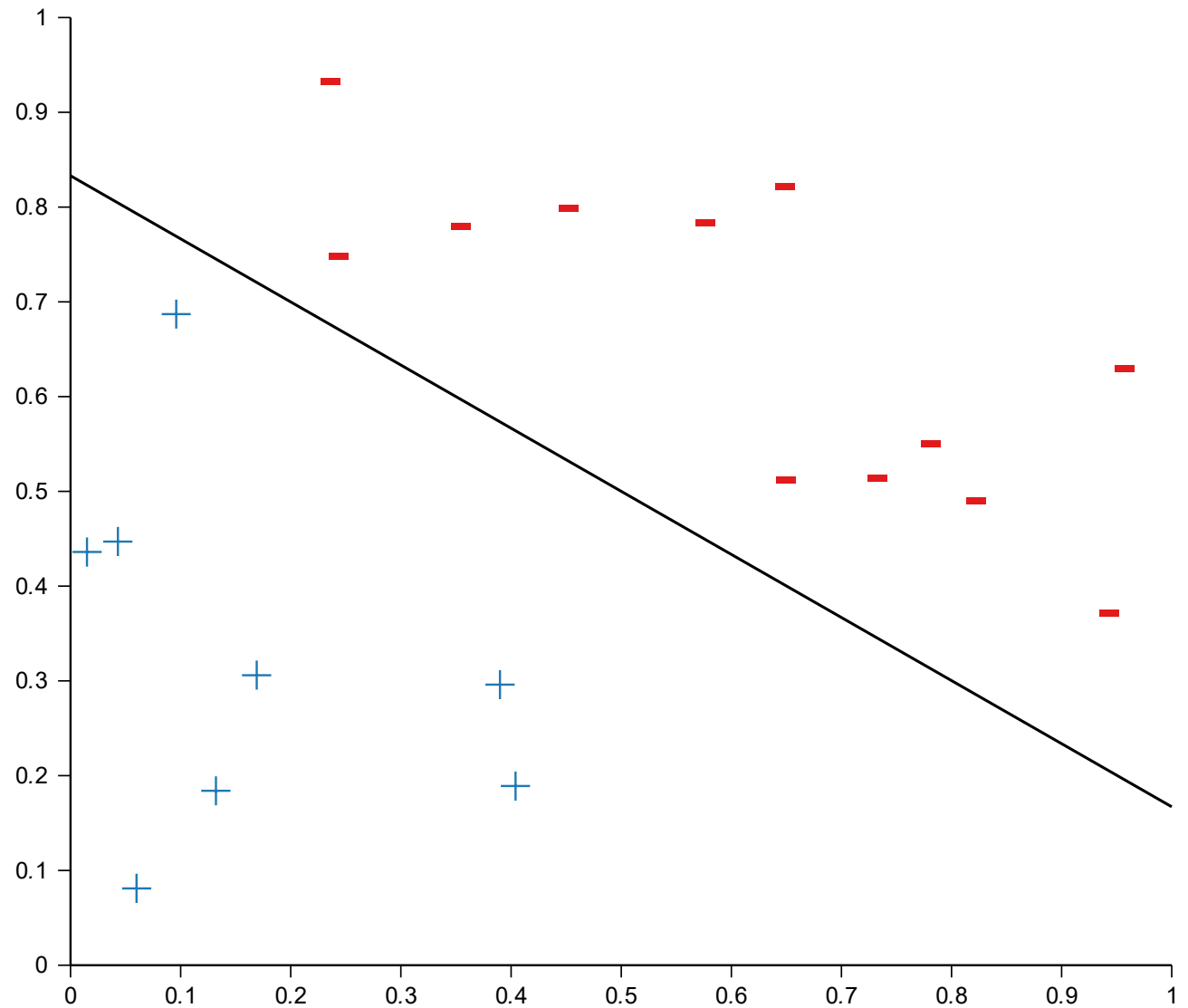
The line defines two half-spaces in \mathbb{R}^D :

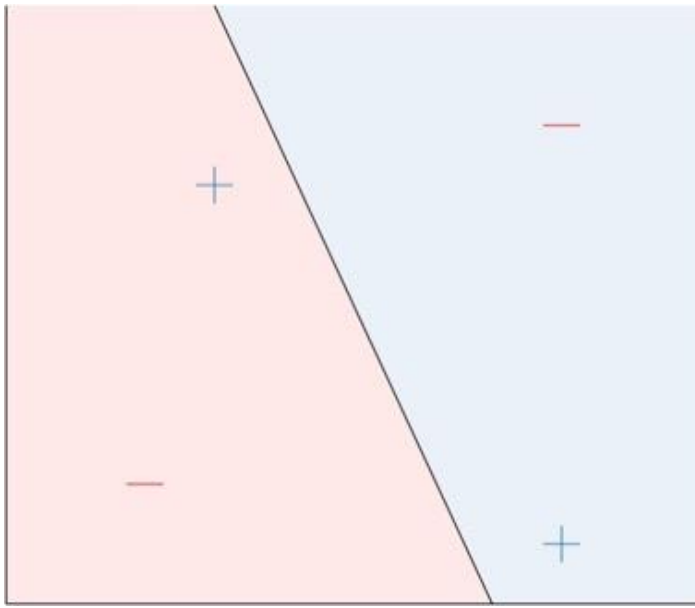
- $\mathcal{S}_+ = \{\mathbf{x}: \mathbf{w}^T \mathbf{x} > 0\}$
- $\mathcal{S}_- = \{\mathbf{x}: \mathbf{w}^T \mathbf{x} < 0\}$

So the model

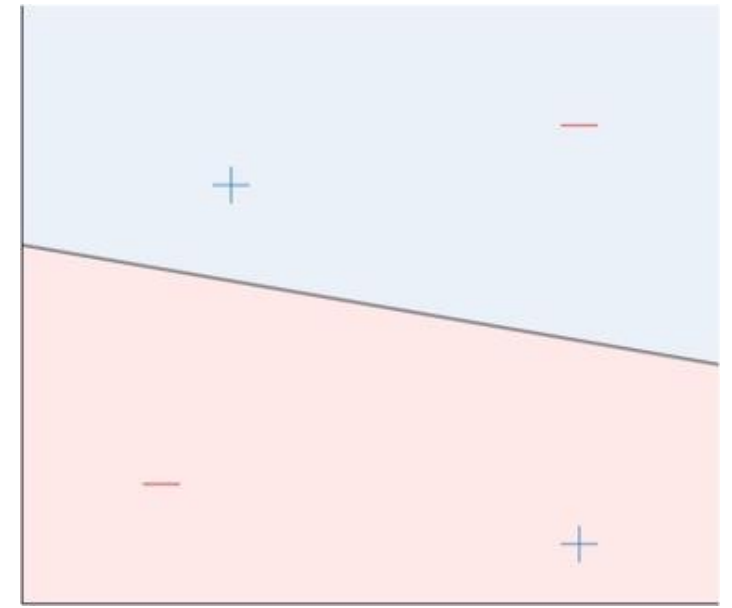
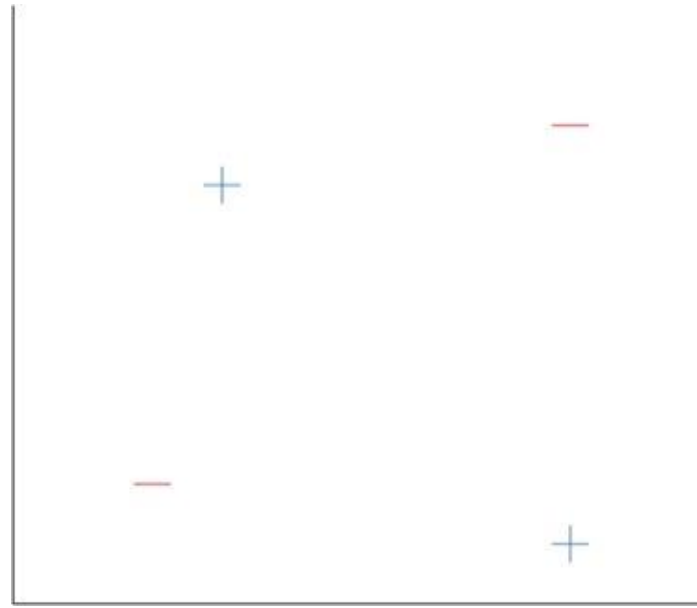
$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$$

gives rise to linear decision boundaries!





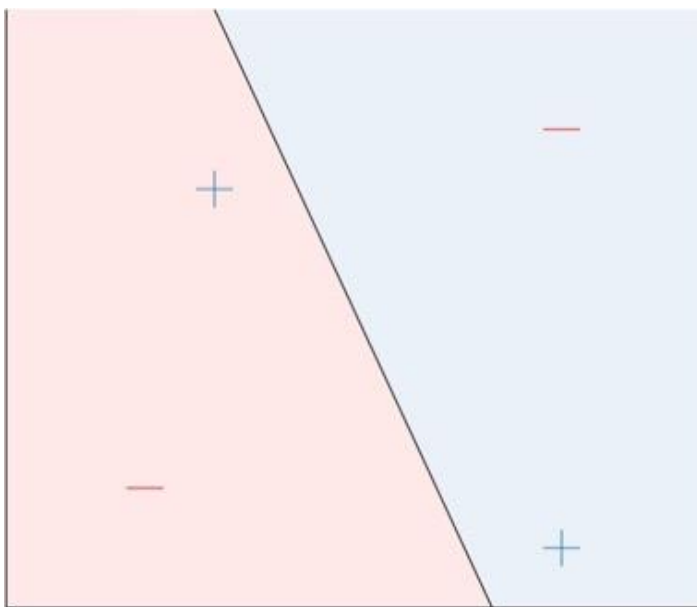
h_1



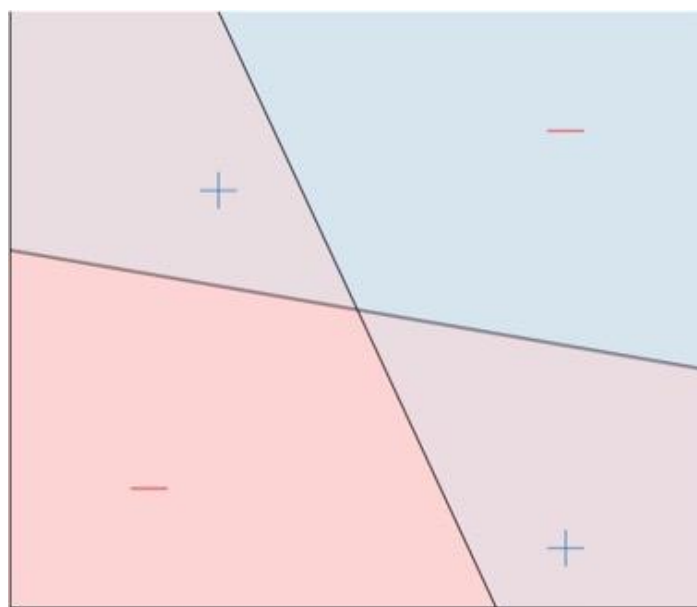
h_2

Perceptrons

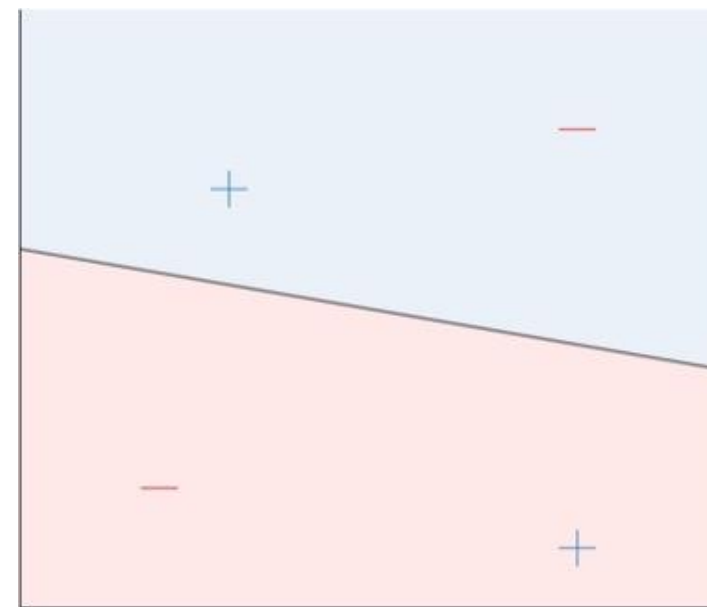
- Linear model for classification
- $h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$
- Predictions are $+1$ or -1



h_1



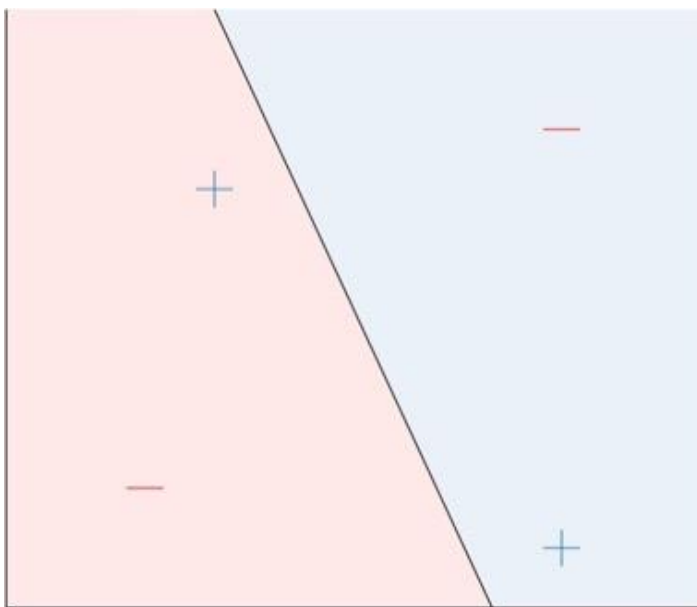
h_1



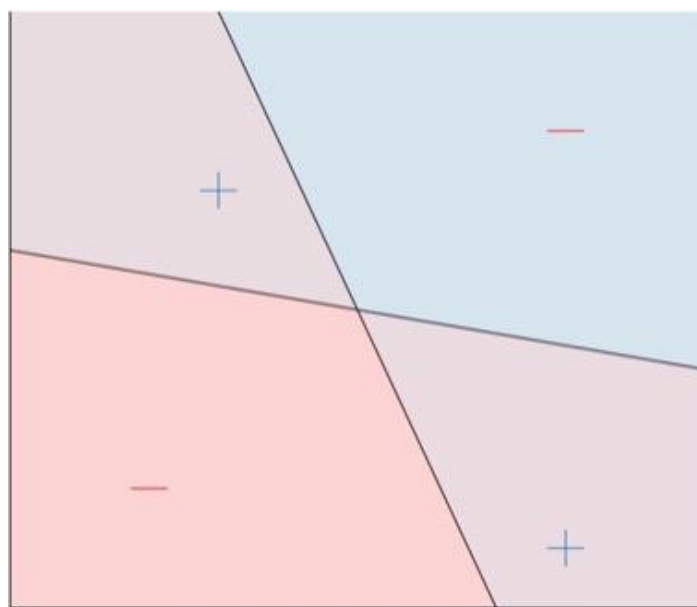
h_2

combined classifier $h_3(x) = ((h_1(x)=+1 \& h_2(x)=-1) \text{ or } (h_1(x)=-1 \& h_2(x)=+1))$

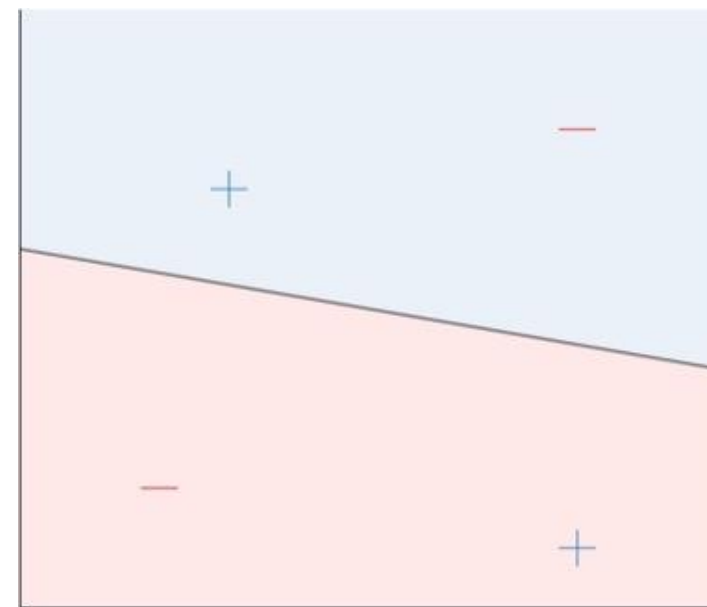
Combining Perceptrons



h_1



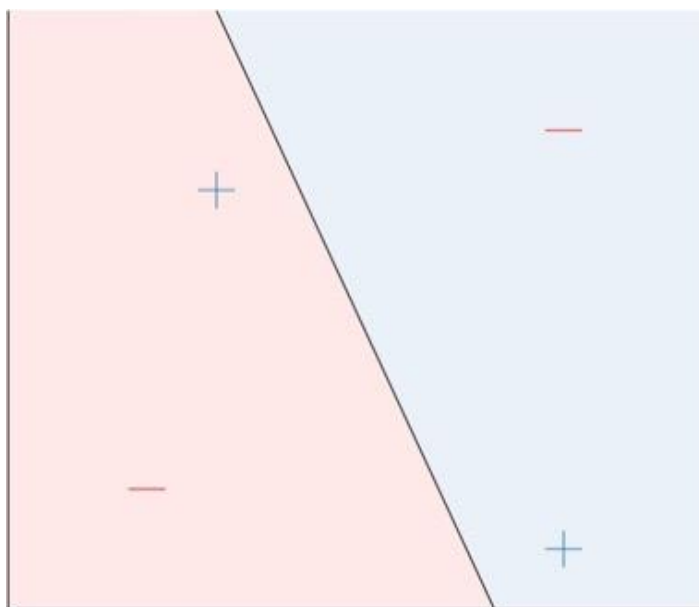
h_1



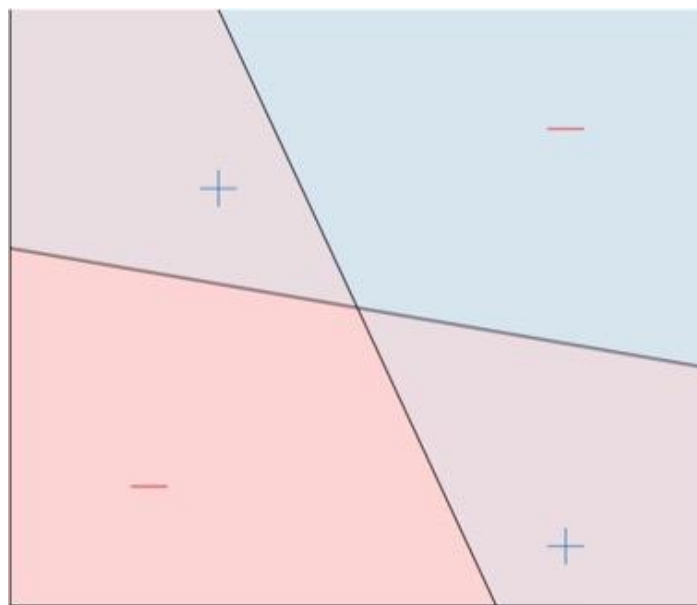
h_2

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

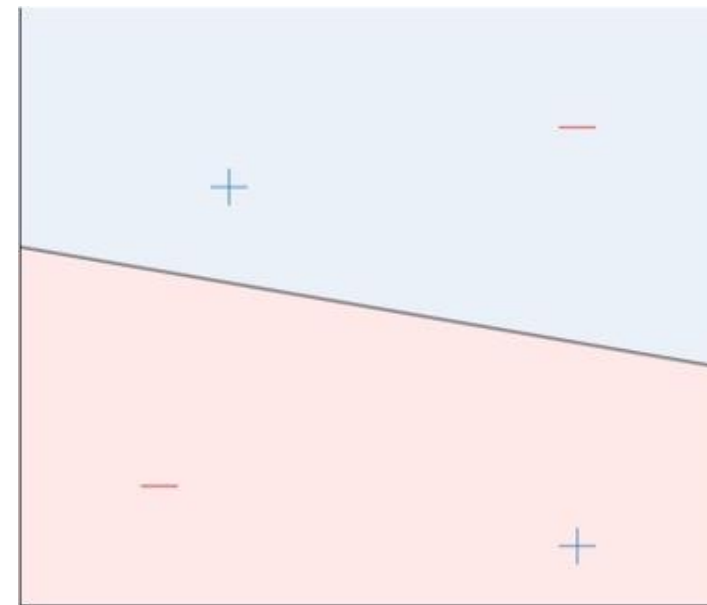
OR (AND ($h_1(x)$, $\neg h_2(x)$) , AND ($\neg h_1(x)$, $h_2(x)$))



h_1



h_1



h_2

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

$AND(b_1, b_2) \rightarrow \text{sign}(\omega_0 + \omega_1 b_1 + \omega_2 b_2) \rightarrow \begin{cases} \omega_0 = ? & -1.5 \\ \omega_1 = ? & +1 \\ \omega_2 = ? & +1 \end{cases}$
 $OR(b_1, b_2) \rightarrow \text{sign}(\omega_0 + \omega_1 b_1 + \omega_2 b_2) \rightarrow \begin{cases} \omega_0 = 1 \\ \omega_1 = +1 \\ \omega_2 = +1 \end{cases}$

Boolean Algebra

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

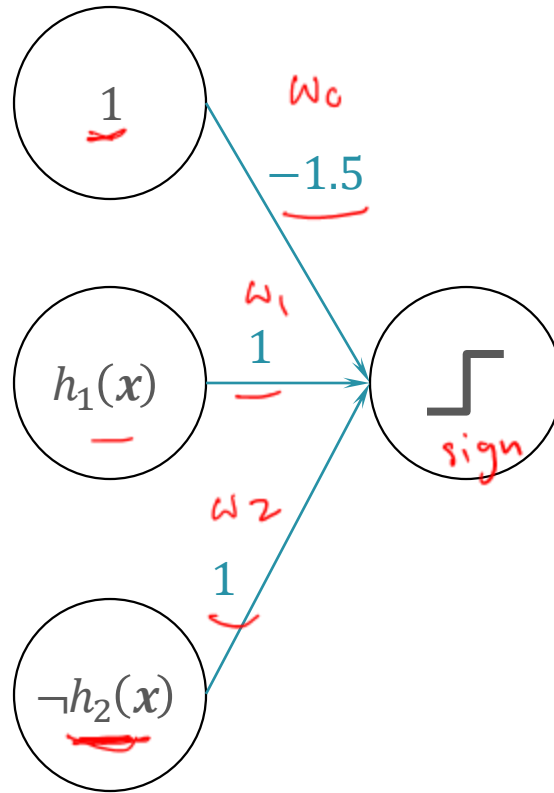
- Boolean variables are either $+1$ ("true") or -1 ("false")
- Basic Boolean operations
 - Negation: $\neg z = -1 * z$
 - And: $AND(z_1, z_2) = \text{sign} \left([-1.5, 1, 1] \begin{bmatrix} 1 \\ z_1 \\ z_2 \end{bmatrix} \right)$
 - Or: $OR(z_1, z_2) = \text{sign} \left([1.5, 1, 1] \begin{bmatrix} 1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

Building a Network

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

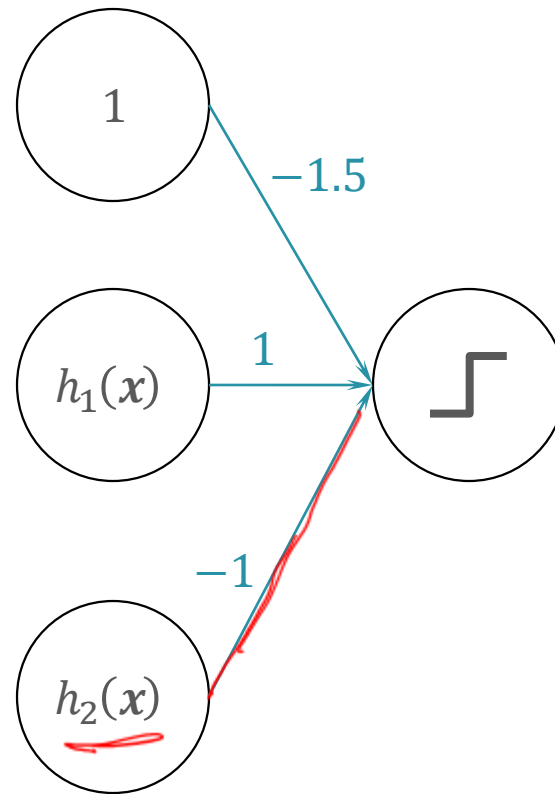
Building a Network

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



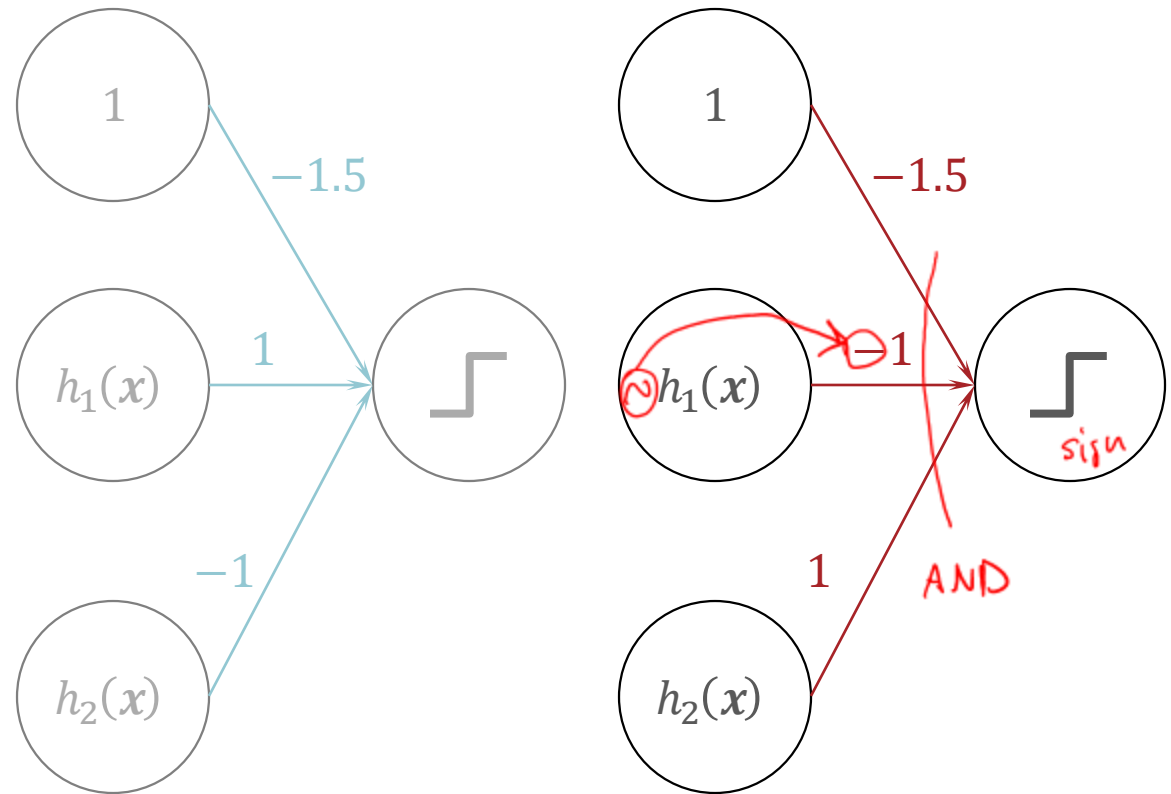
Building a Network

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



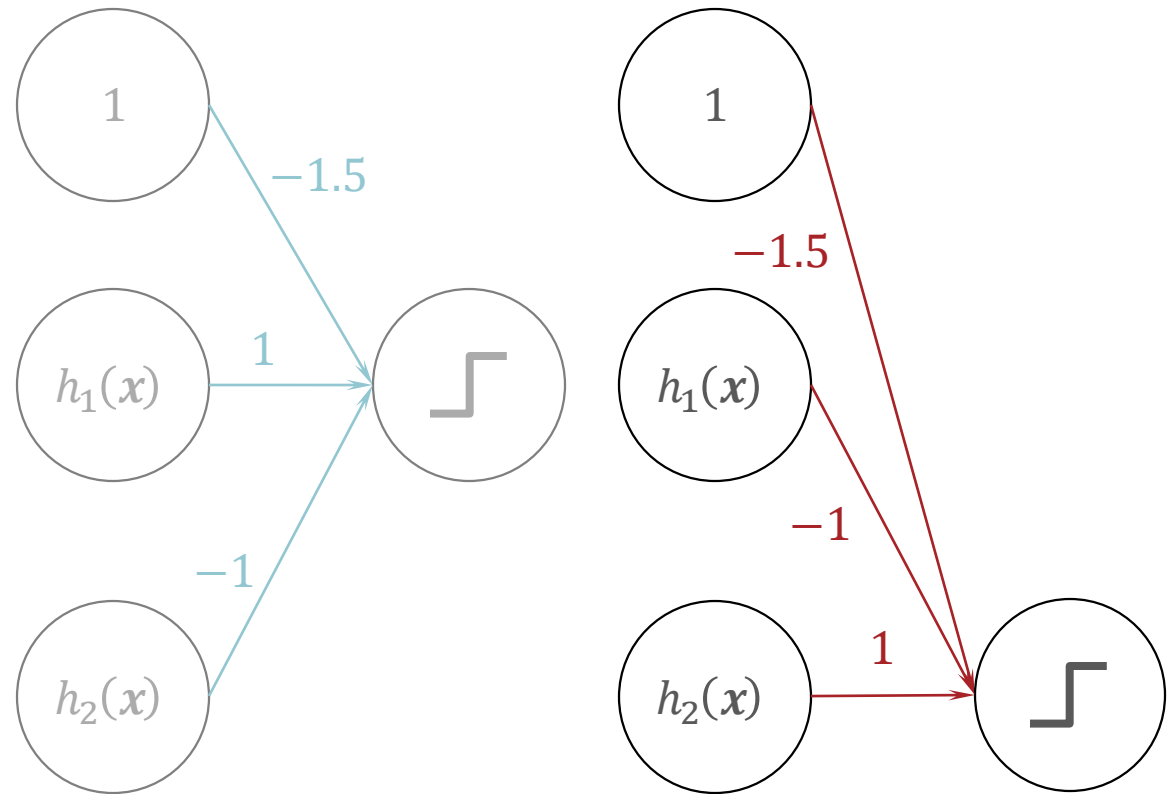
Building a Network

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



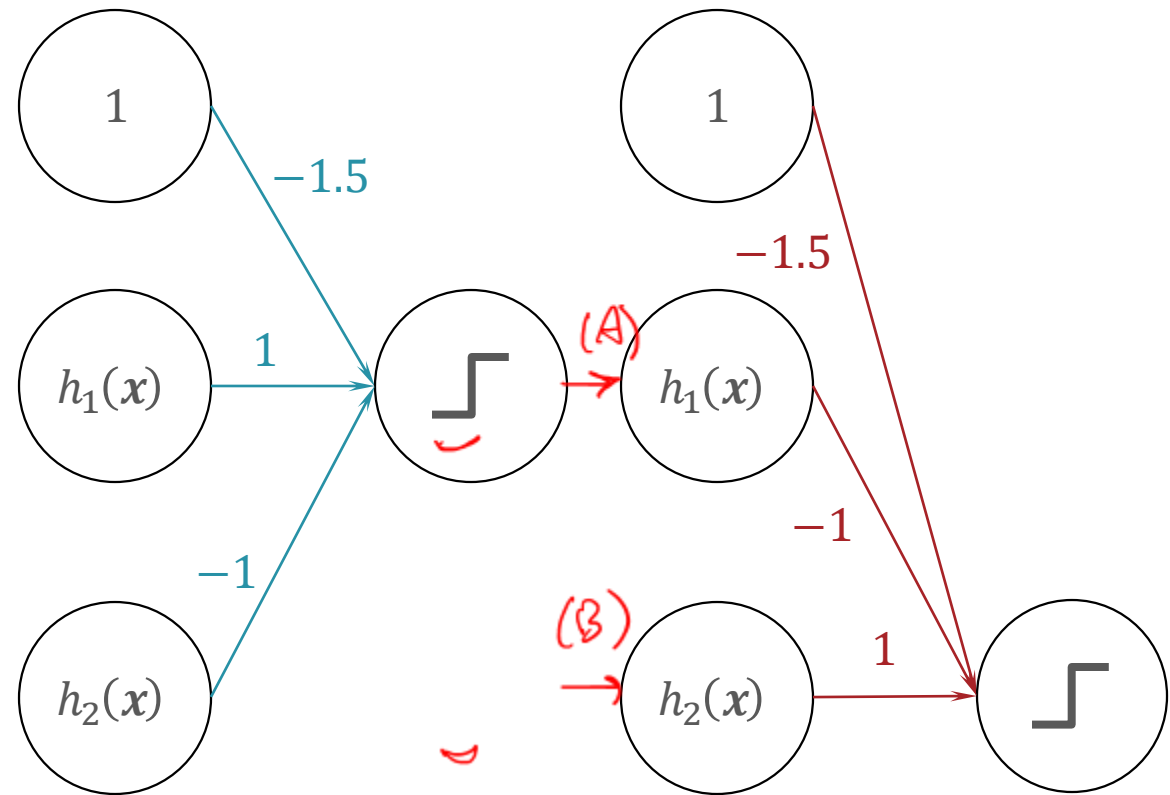
Building a Network

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



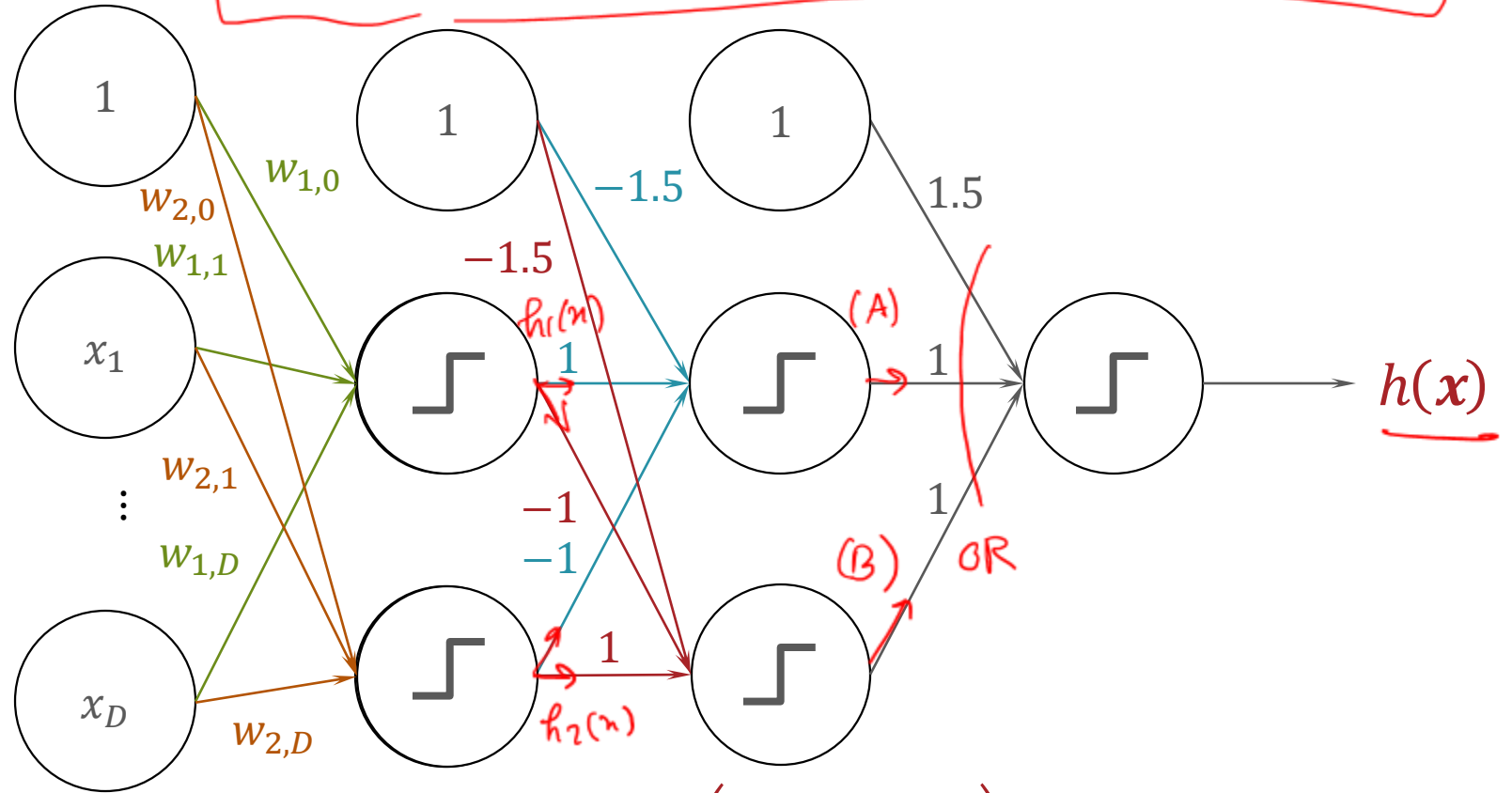
Building a Network

$$h(x) = \text{OR} \left(\overset{(A)}{AND(h_1(x), \neg h_2(x))}, \overset{(B)}{AND(\neg h_1(x), h_2(x))} \right)$$



Building a Network

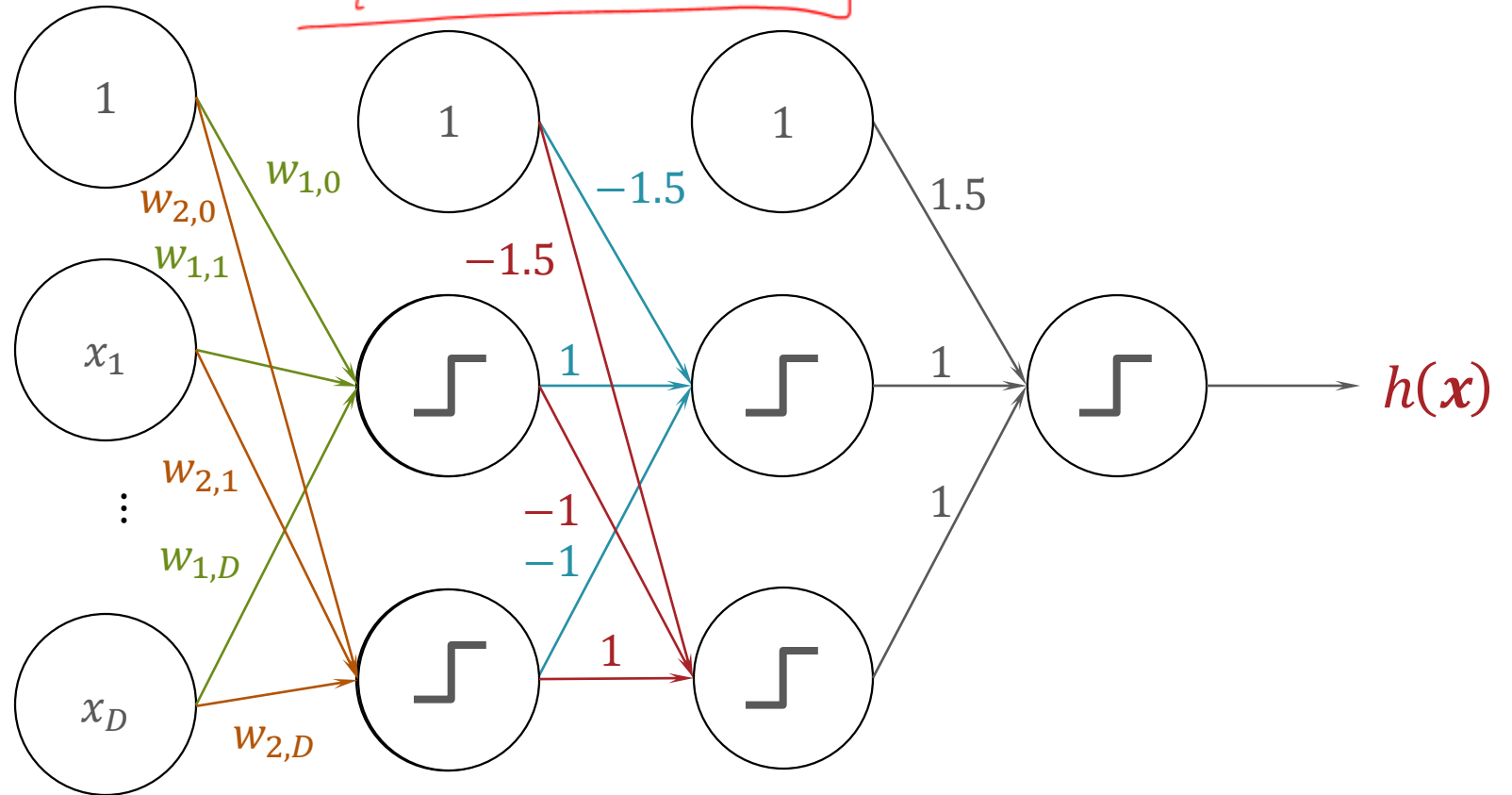
$$h(\mathbf{x}) = \text{OR} \left(\text{AND}(h_1(\mathbf{x}), \neg h_2(\mathbf{x})), \text{AND}(\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$



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

Building a Network

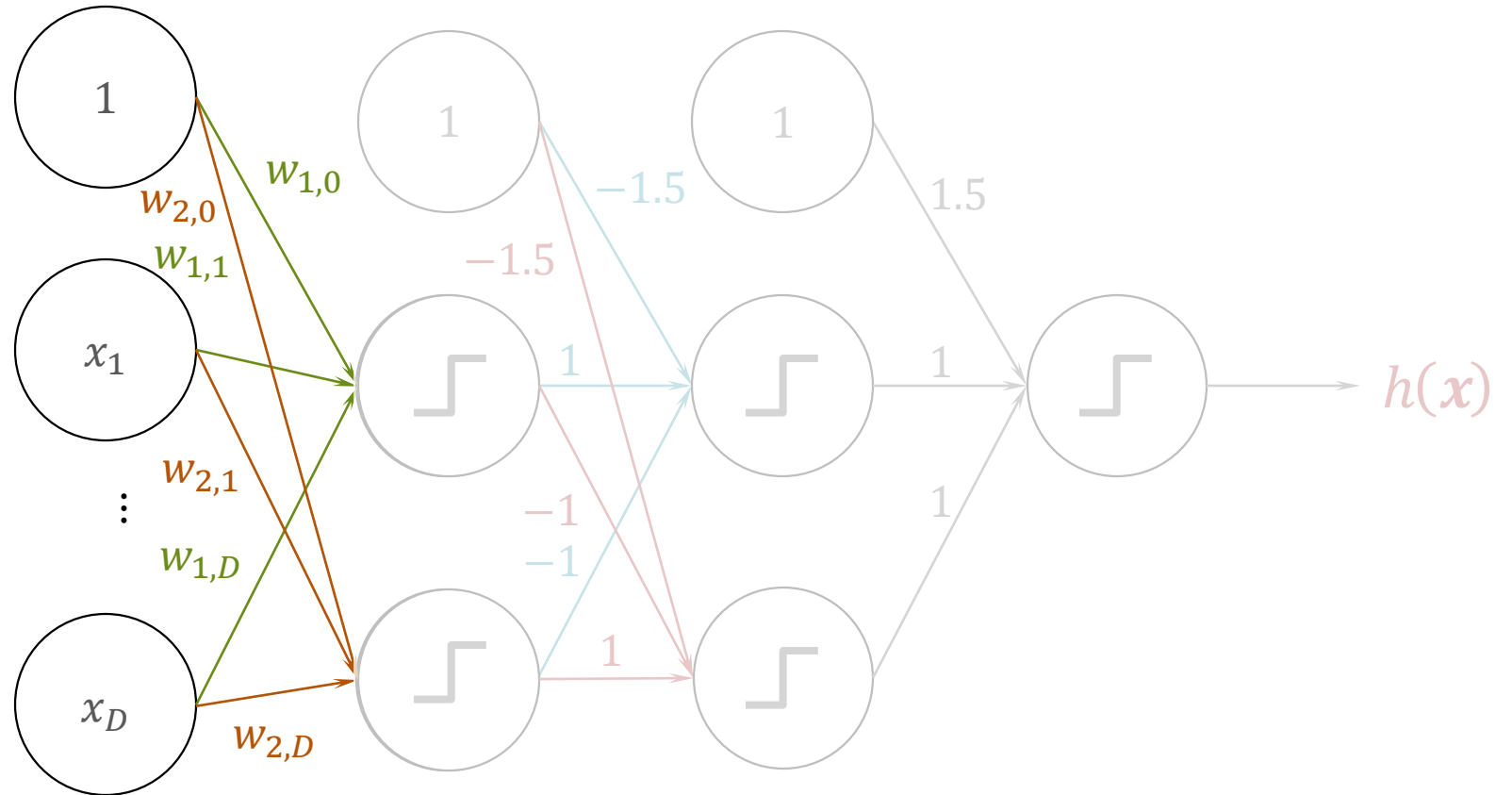
$$h(\mathbf{x}) = OR \left(\underbrace{AND(h_1(\mathbf{x}), \neg h_2(\mathbf{x}))}_{\text{red box}}, AND(\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$



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

Building a Network

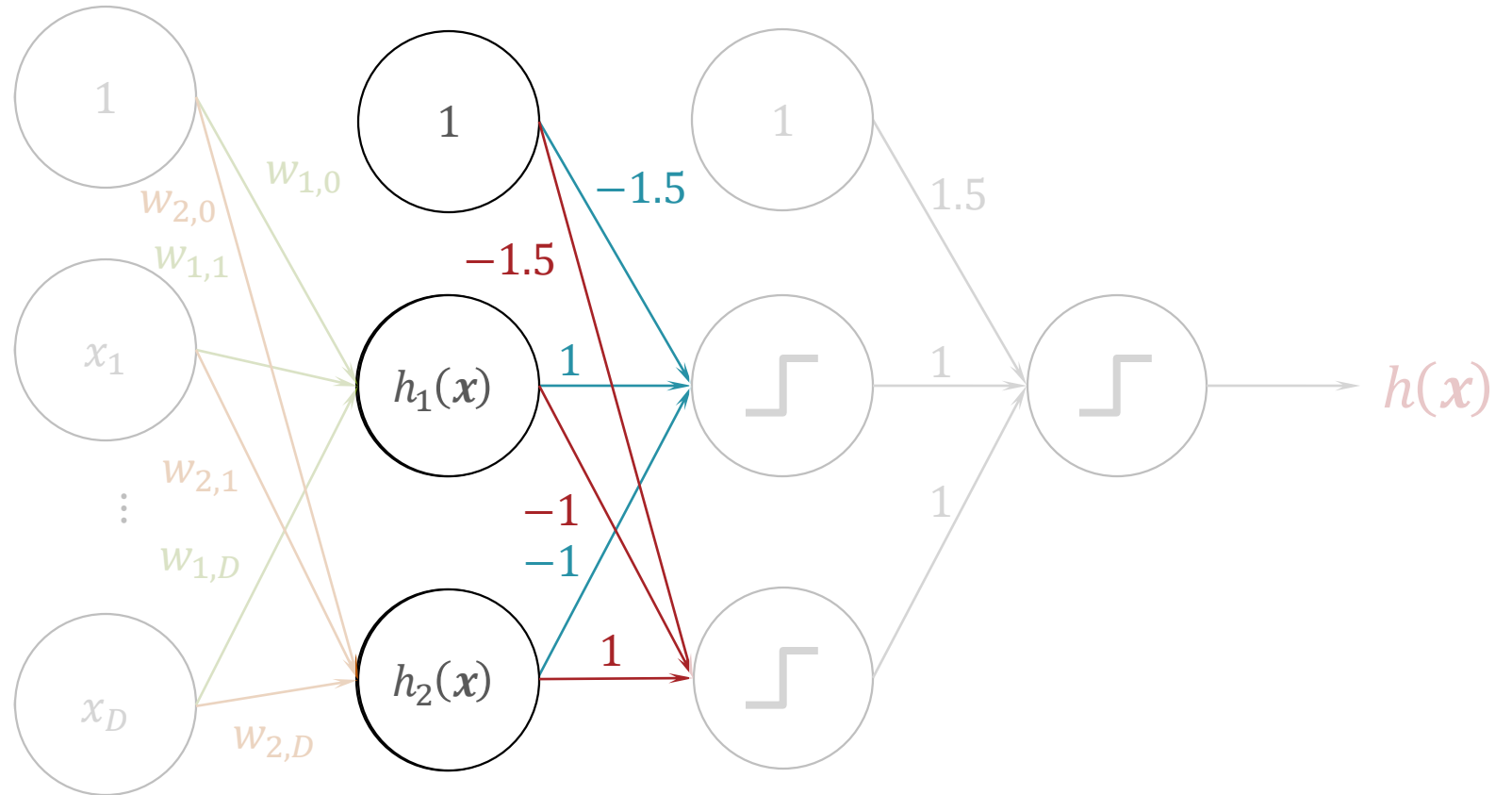
$$h(\mathbf{x}) = \text{OR} \left(\text{AND} (h_1(\mathbf{x}), \neg h_2(\mathbf{x})), \text{AND} (\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$



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

Building a Network

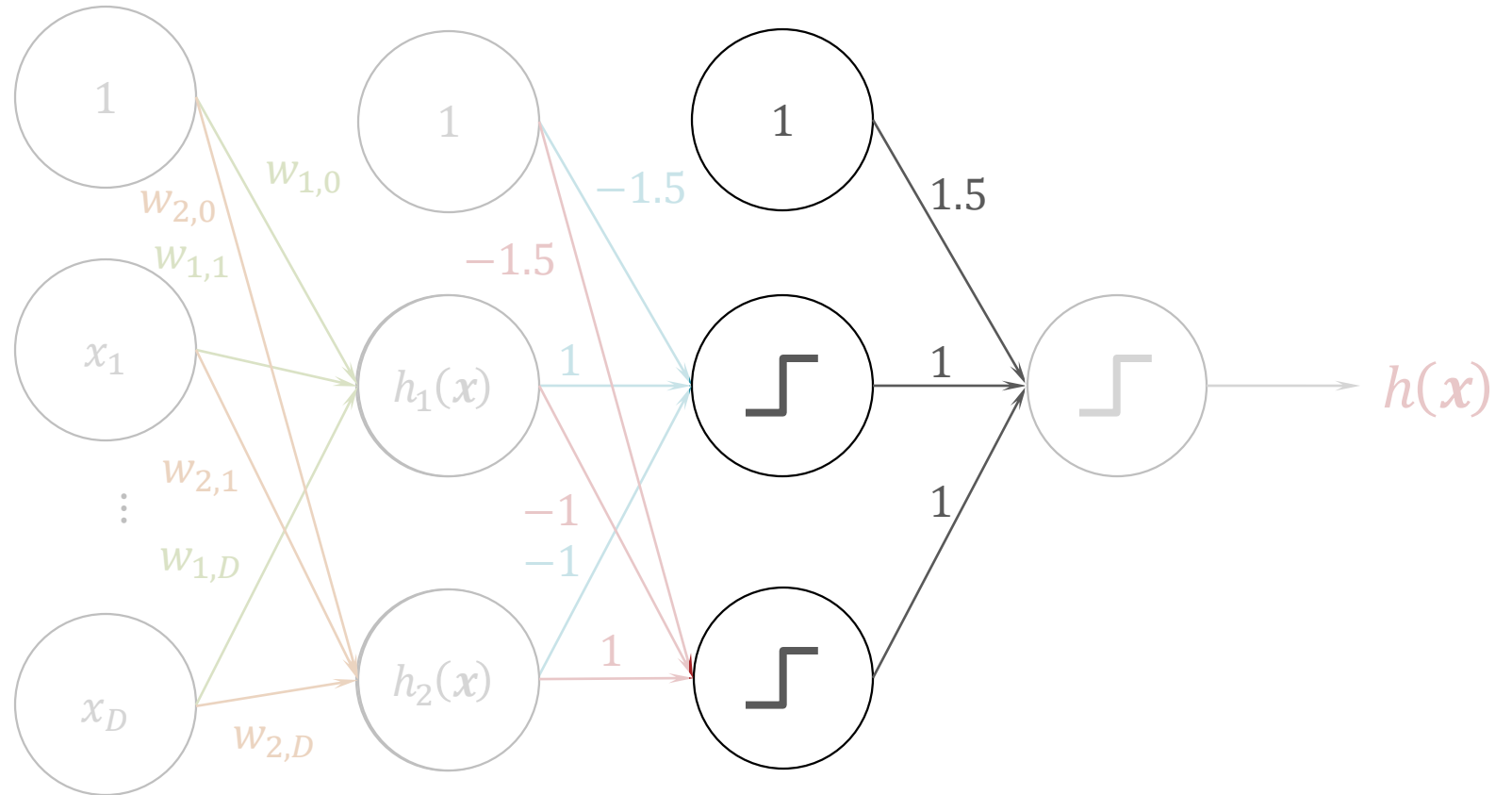
$$h(\mathbf{x}) = OR \left(AND(h_1(\mathbf{x}), \neg h_2(\mathbf{x})), AND(\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$



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

Building a Network

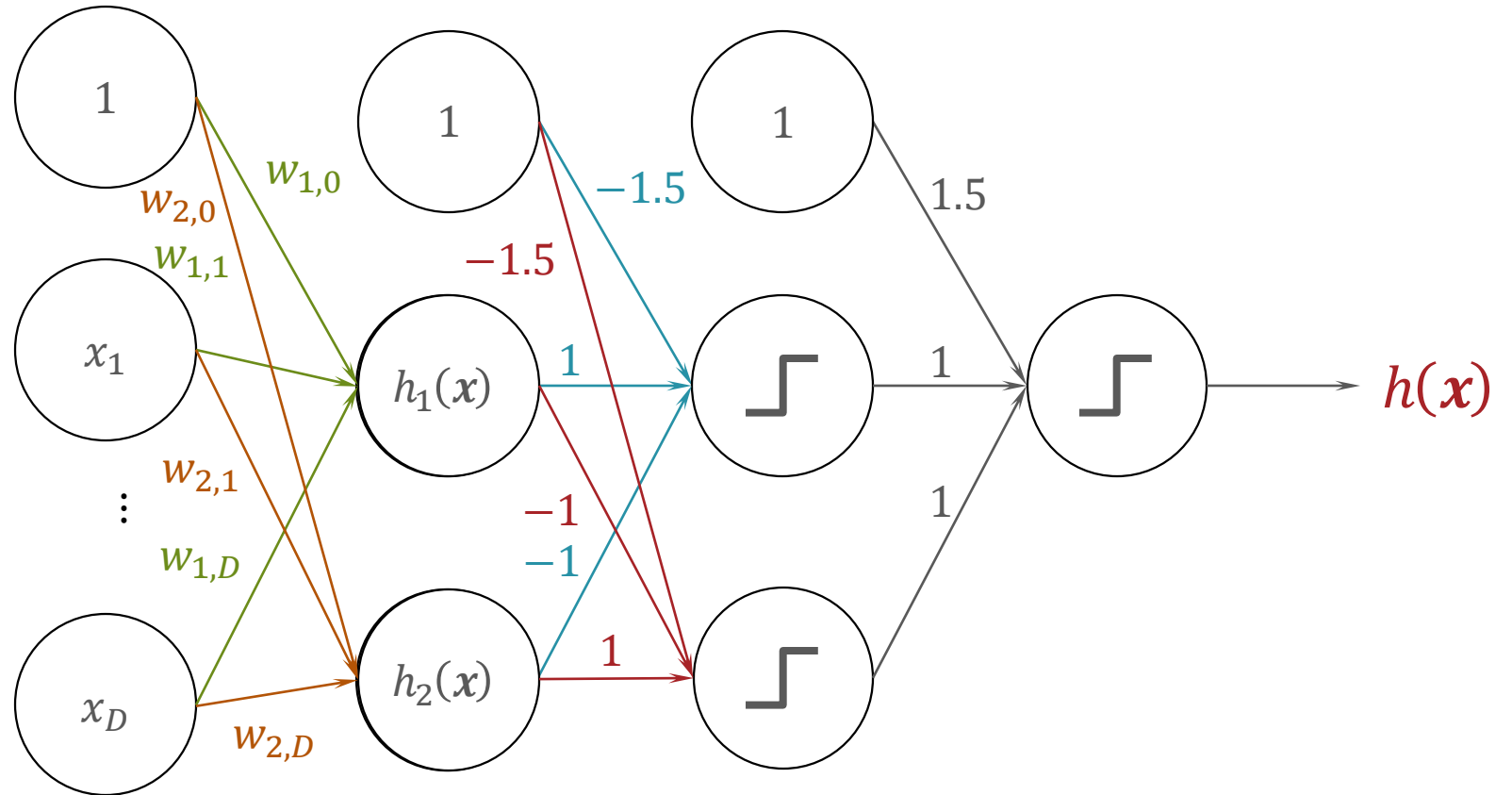
$$h(\mathbf{x}) = OR \left(AND(h_1(\mathbf{x}), \neg h_2(\mathbf{x})), AND(\neg h_1(\mathbf{x}), h_2(\mathbf{x})) \right)$$



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

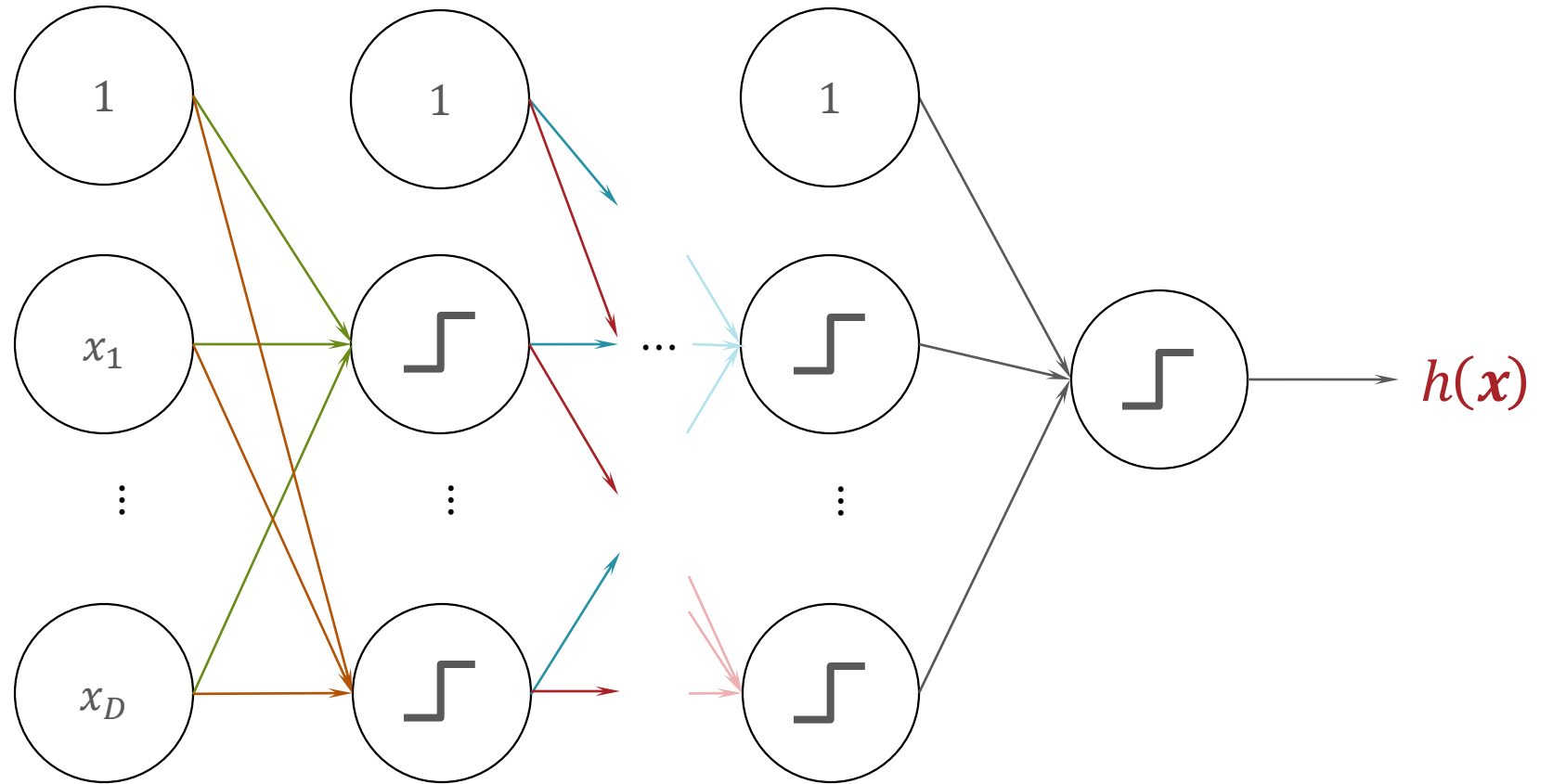
Building a Network

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

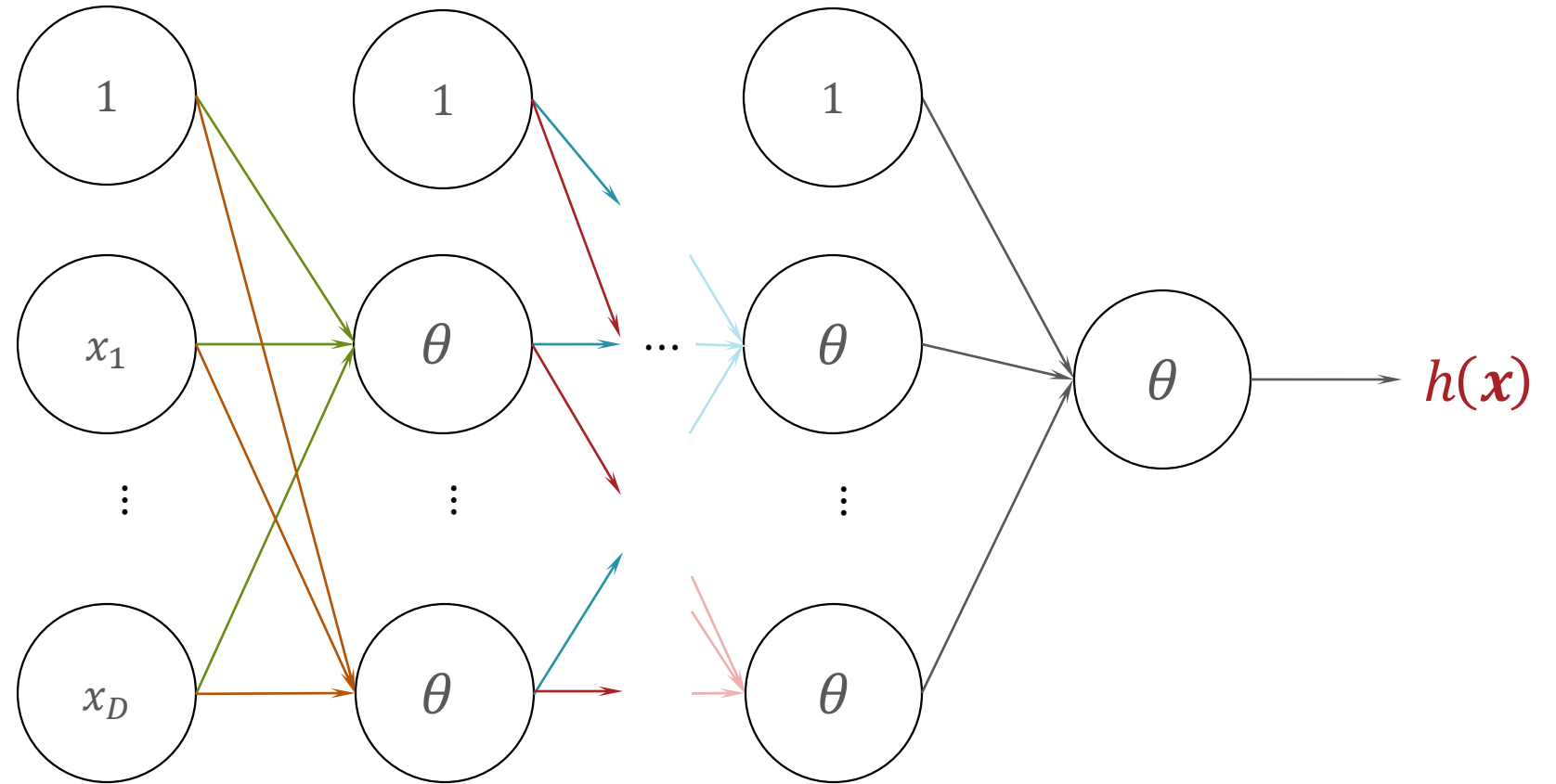


$$h(\mathbf{x}) = \text{sign}(\text{sign}(\text{sign}(\mathbf{w}_1^T \mathbf{x}) - \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + \text{sign}(-\text{sign}(\mathbf{w}_1^T \mathbf{x}) + \text{sign}(\mathbf{w}_2^T \mathbf{x}) - 1.5) + 1.5)$$

Multi-Layer Perceptron (MLP)



(Fully-Connected) Feed Forward Neural Network

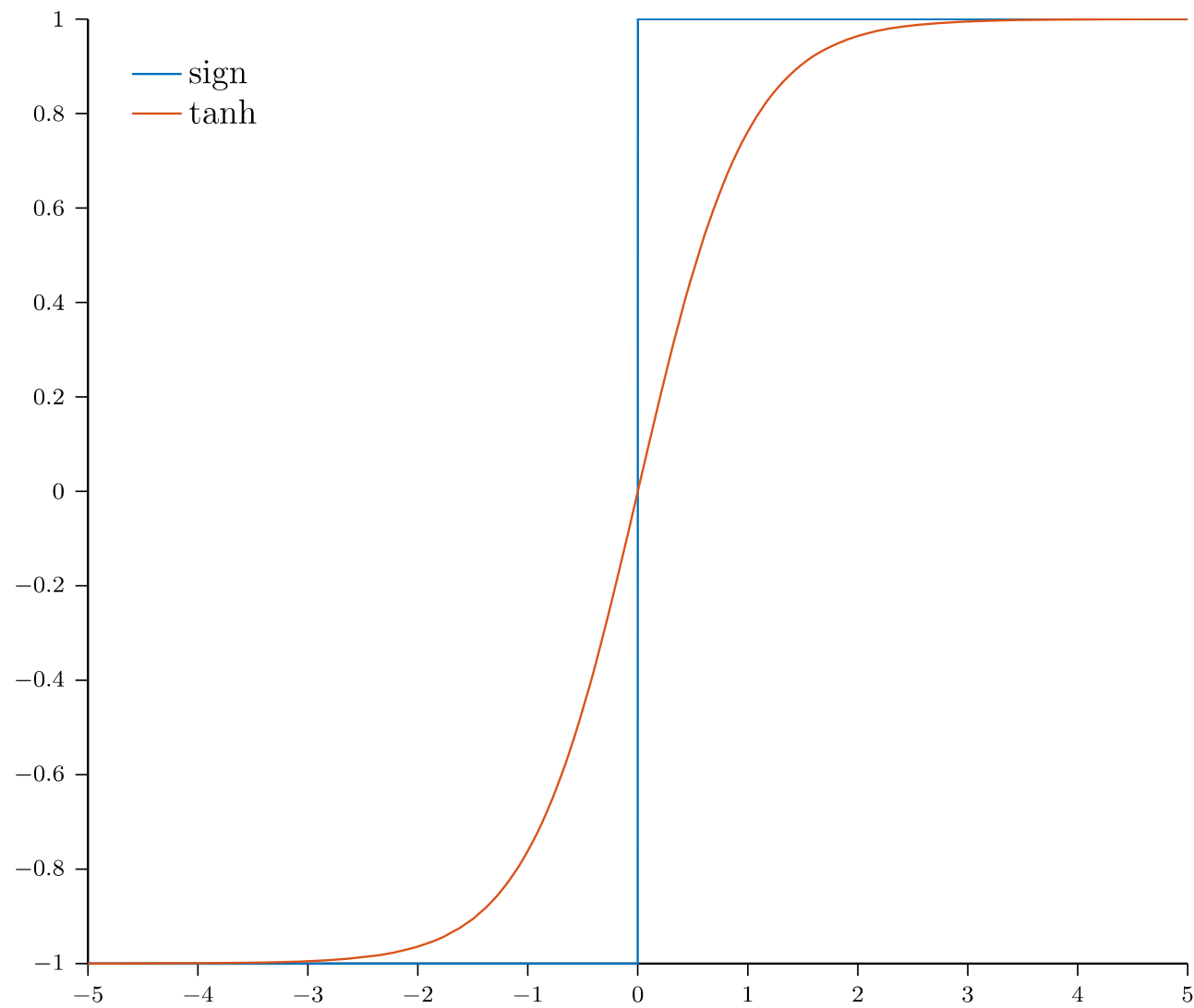


$\theta(\cdot)$

- Hyperbolic tangent:


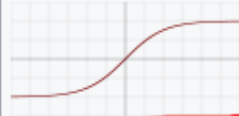

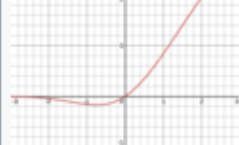


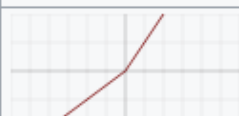

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

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



Other Activation Functions

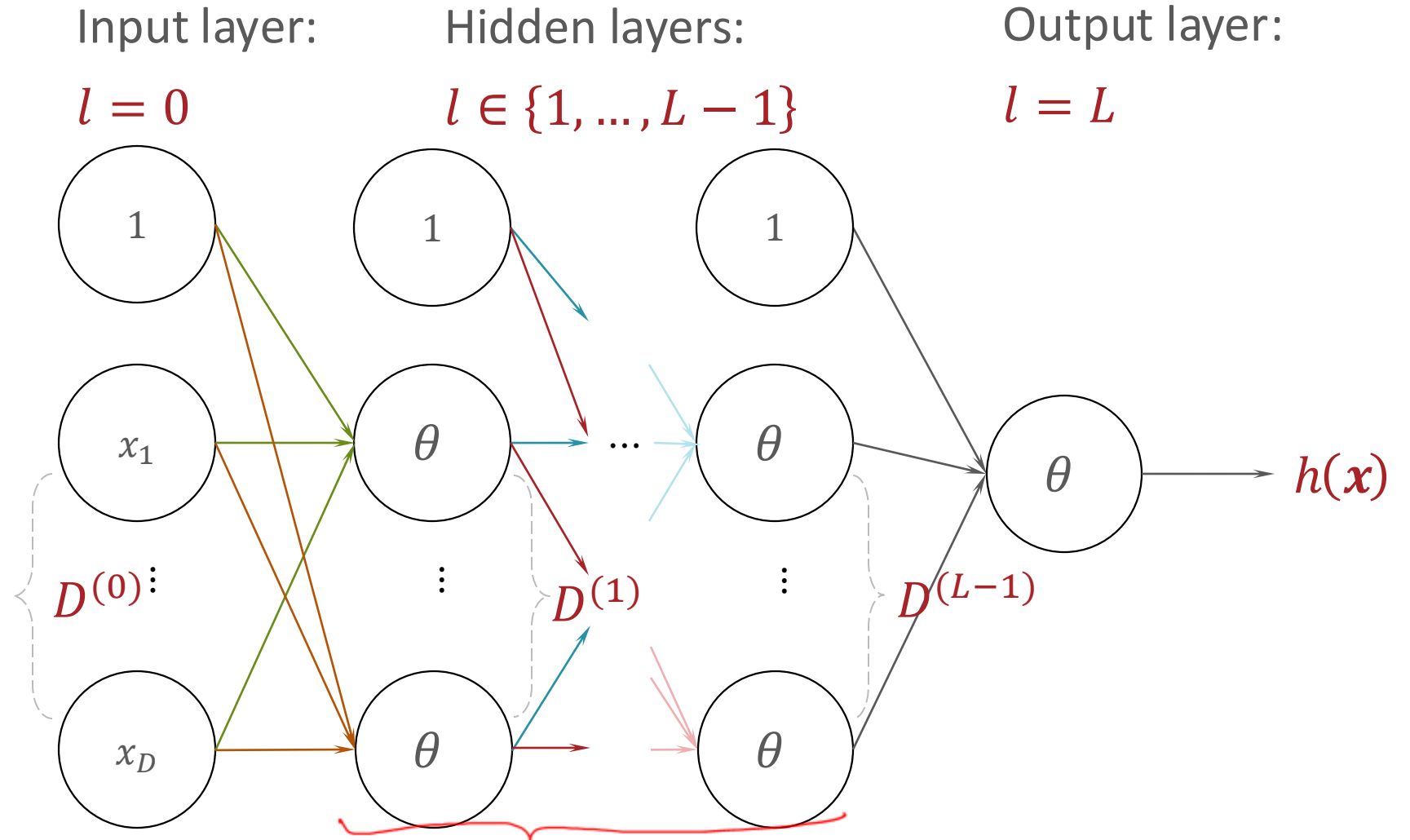
Multiclass classification \rightarrow softmax activation
 output Binary output

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] ✓		$\frac{1}{2}x \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$ $= x\Phi(x)$
Softplus ^[8]		$\ln(1 + e^x)$
Exponential linear unit (ELU) ^[9]		$\begin{cases} \alpha (e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ <p>with parameter α</p>
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] ✓		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ <p>with parameter α</p>

Regression

~~✗~~ x

(Fully-Connected) Feed Forward Neural Network

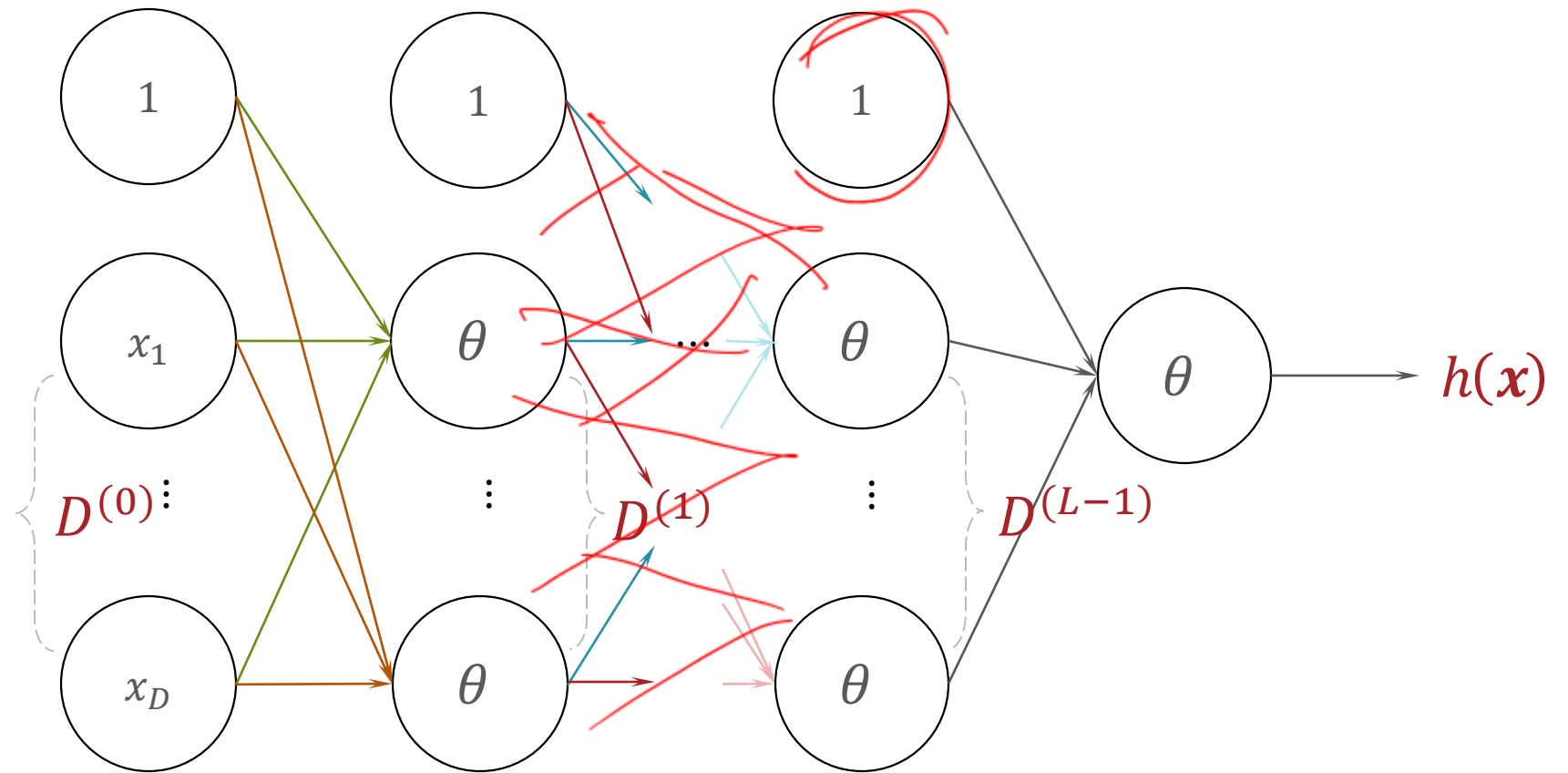


Layer l has dimension $D^{(l)}$ → Layer l has $D^{(l)} + 1$ nodes, counting the bias node

(Fully-Connected) Feed Forward Neural Network

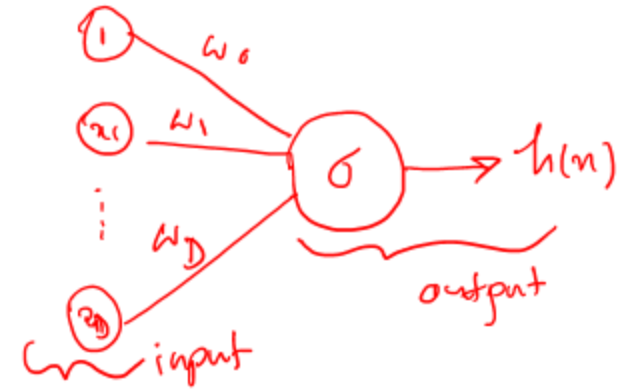
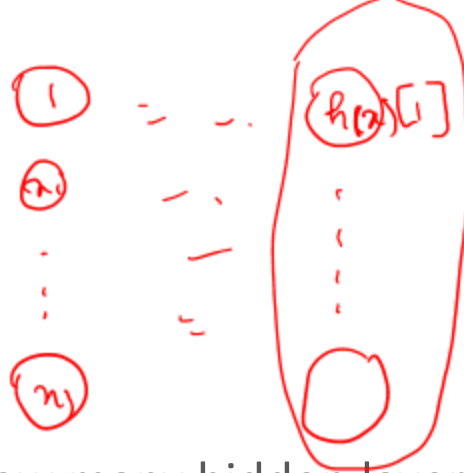
The weights between layer $l - 1$ and layer l are a matrix:

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



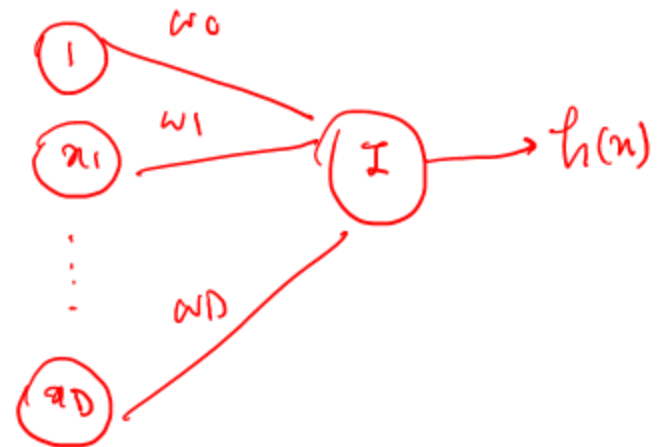
$w_{j,i}^{(l)}$ is the weight between node i in layer $l - 1$ and node j in layer l

In-class Poll



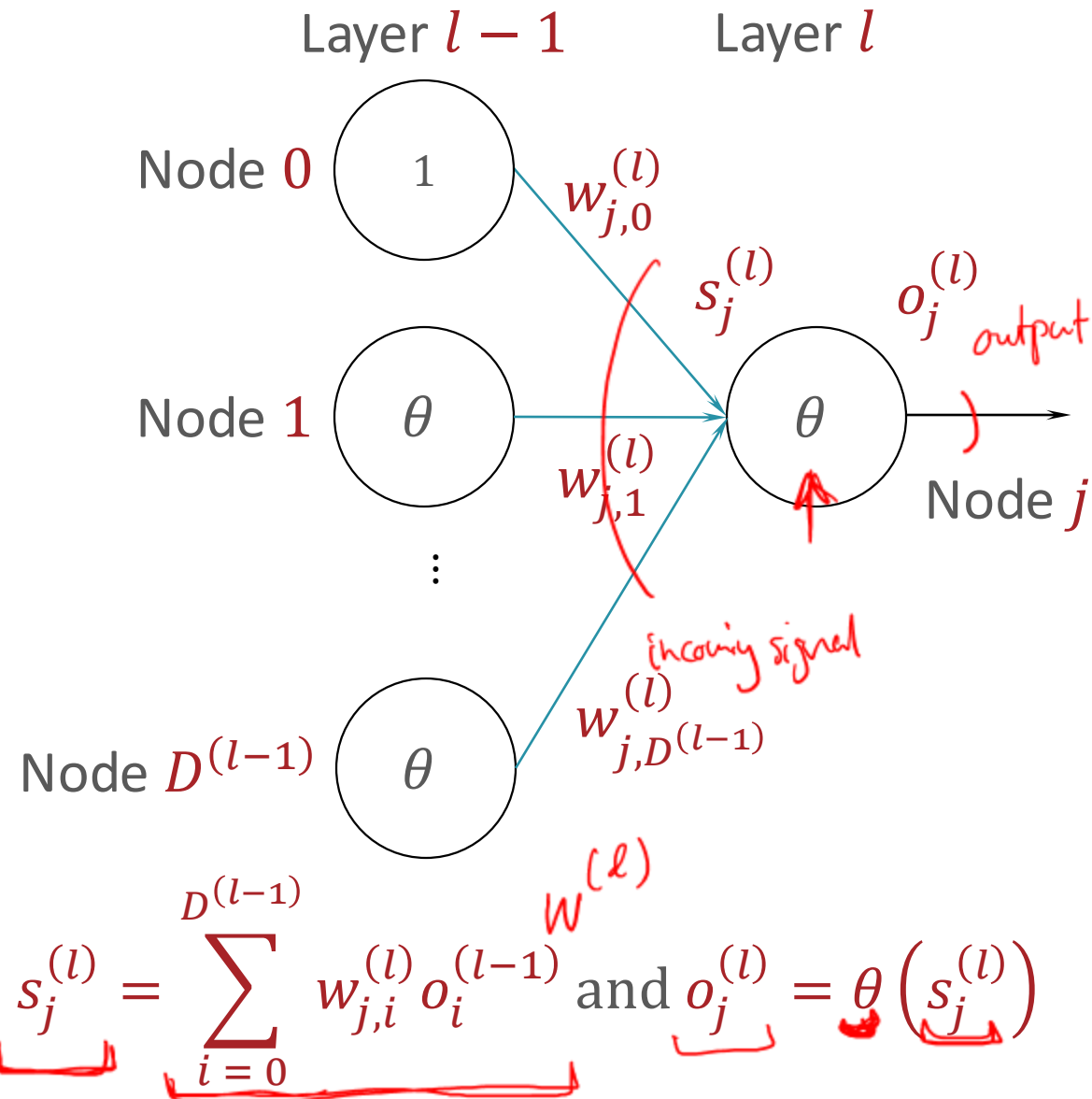
- How many hidden layers are necessary to simulate logistic regression using a NN? 0

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



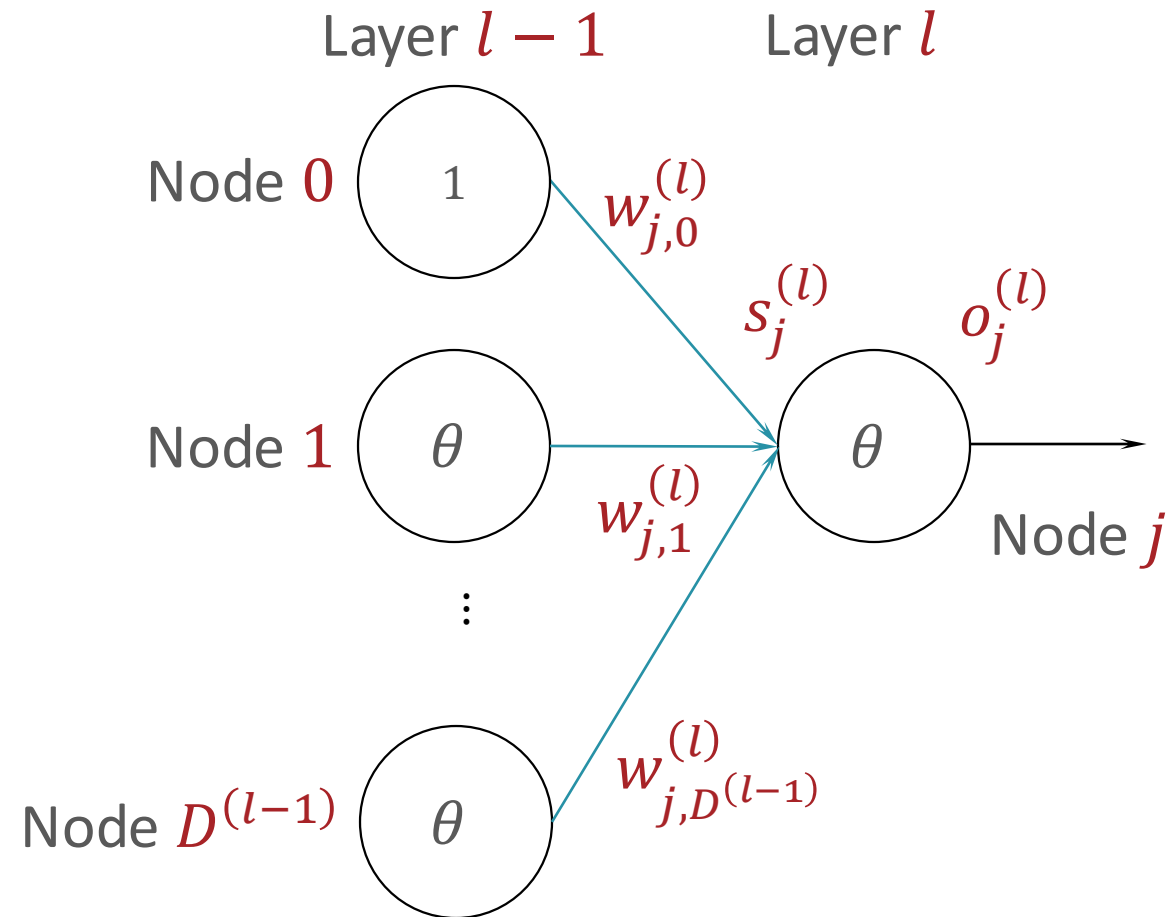
Signal and Outputs

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



Signal and Outputs

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



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

Forward Propagation for Making Predictions

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

Stochastic 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 $i \in \text{shuffle}(\{1, \dots, N\})$
 - For $l = 1, \dots, L$
 - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell^{(i)}(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)})$
 - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta^{(0)} G^{(l)}$
 - Increment t : $t = t + 1$
- Output: $W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}$

Two questions:

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

2. How on earth do we take these gradients?

- 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 $i \in \text{shuffle}(\{1, \dots, N\})$
 - For $l = 1, \dots, L$
 - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell^{(i)}(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)})$ (???)
 - 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!)

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

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

$$\text{cross-entropy loss} = - [y \log(p) + (1-y) \log(1-p)]$$
$$\begin{cases} p = 0.9/0.1 & -\log(0.9) \\ y = 1 \end{cases} \quad \begin{cases} p = 0.1/0.9 & -\log(0.9) \\ y = 0 & -\log(0.1) \end{cases}$$

Loss Functions for Neural Networks

- Binary classification - cross-entropy loss

$$D = \{(x_i, y_i)\}_{i=1}^N$$

$$\hat{y}_i = h_{\mathbf{w}}(x_i)$$

$$\text{Likelihood of data} = \prod_{i=1}^N P(y^i | x^i, \mathbf{w}) = \sum \log P(y^i | x^i, \mathbf{w})$$

Loss Functions for Neural Networks

- Binary classification - cross-entropy loss

- Assume $P(Y = 1 | \mathbf{x}, W^{(1)}, \dots, W^{(L)}) = h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x})$

$$\ell^{(i)}(W^{(1)}, \dots, W^{(L)}) = - \log P(y^{(i)} | \mathbf{x}^{(i)}, W^{(1)}, \dots, W^{(L)})$$

$$= - \log \left(\underbrace{h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(i)})}_{\substack{\uparrow \\ P(Y=1|\mathbf{x}, W^{(1)}, \dots, W^{(L)})}}^{y^{(i)}} \left(\underbrace{1 - h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(i)})}_{P(Y=0|\mathbf{x}, W^{(1)}, \dots, W^{(L)})} \right)^{1-y^{(i)}} \right)$$

$$= - \underbrace{y^{(i)}}_{\substack{\uparrow \\ P(Y=1|\mathbf{x}, W^{(1)}, \dots, W^{(L)})}} \log \left(h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(i)}) \right)$$

$$- \underbrace{(1 - y^{(i)})}_{\substack{\uparrow \\ P(Y=0|\mathbf{x}, W^{(1)}, \dots, W^{(L)})}} \log \left(1 - h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(i)}) \right)$$

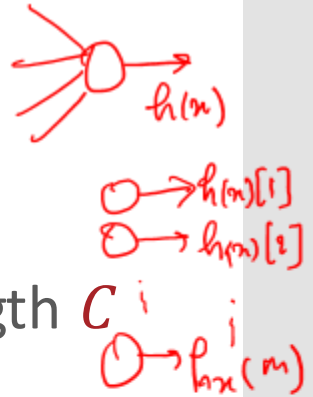
Loss Functions for Neural Networks

- Multi-class classification - also the cross-entropy loss!

- Express the label as a one-hot or one-of- C vector e.g.,

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

¹
²
³
^m



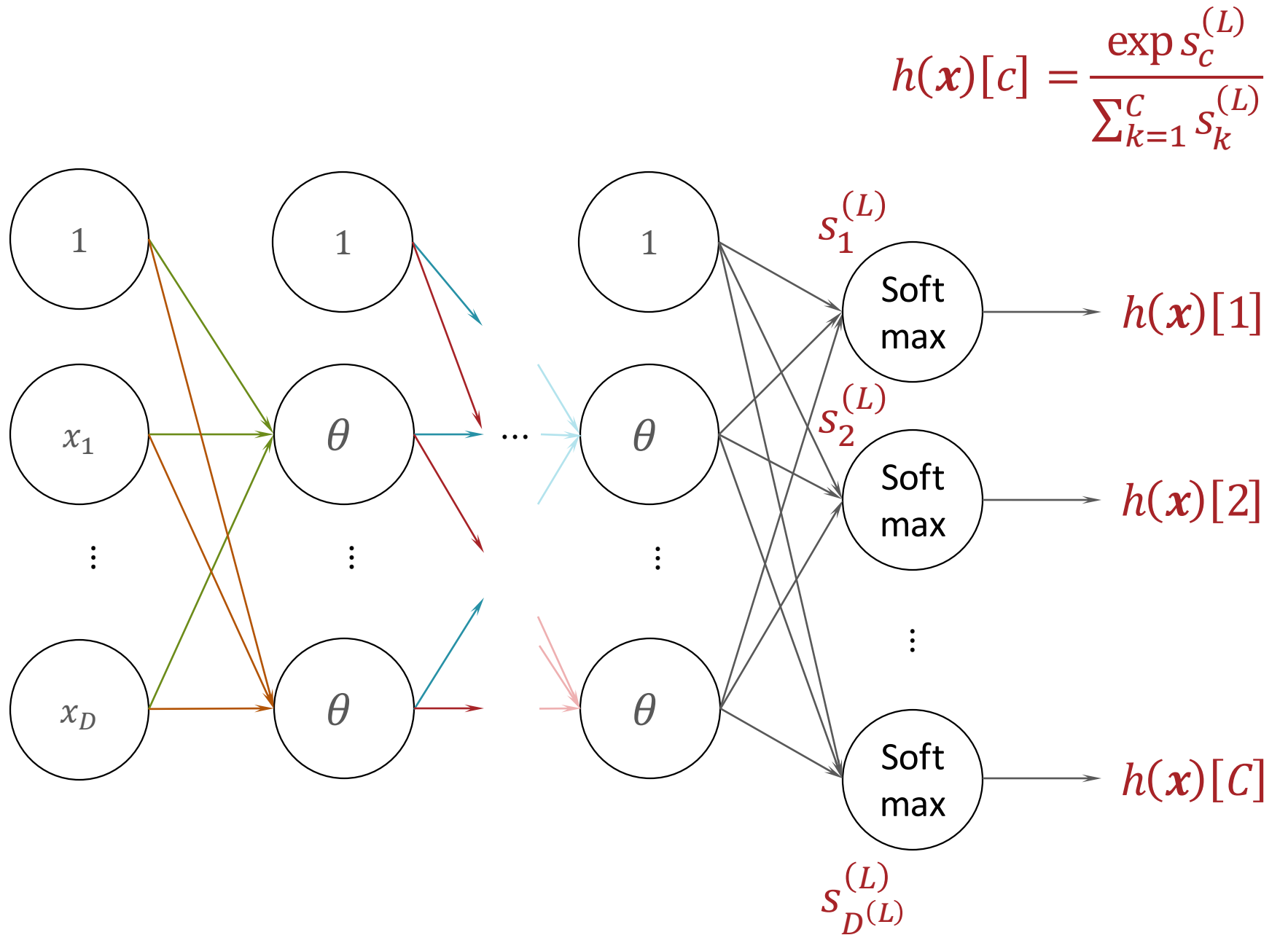
- Assume the neural network output is also a vector of length C

$$P(y[c] = 1 | \mathbf{x}, W^{(1)}, \dots, W^{(L)}) = \underbrace{h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x})}_{\text{red underline}} \underbrace{[c]}_{\text{red box}}$$

- Then the cross-entropy loss is

$$\begin{aligned} \ell^{(i)}(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}) &= -\log P(y^{(i)} | \mathbf{x}^{(i)}, W^{(1)}, \dots, W^{(L)}) \\ &= -\sum_{c=1}^C \underbrace{y[c]}_{\text{red underline}} \underbrace{\log h_{W^{(1)}, \dots, W^{(L)}}(\mathbf{x}^{(n)})}_{\text{red underline}} \underbrace{[c]}_{\text{red underline}} \end{aligned}$$

Multi-dimensional Outputs



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