Feedforward Neural Networks

Elena Congeduti, 13-11-2024

TU Delft

Announcement

The exam of this course is only open for students regularly enrolled in the Engineering with AI Minor programme

To use Kaggle accelerators (GPUs, TPUs), you have to verify your Kaggle account through your phone number

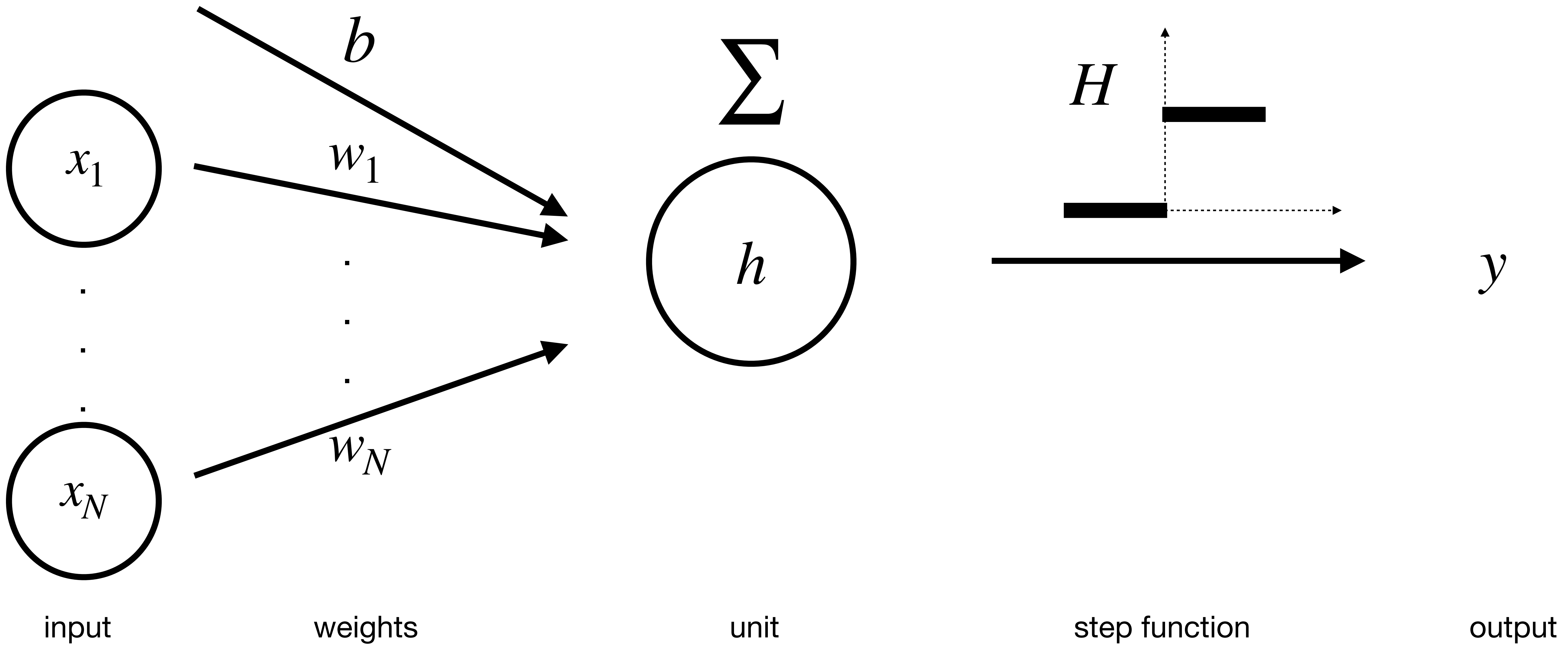
Lecture Agenda

1. Linear Perceptron

2. Feedforward Networks

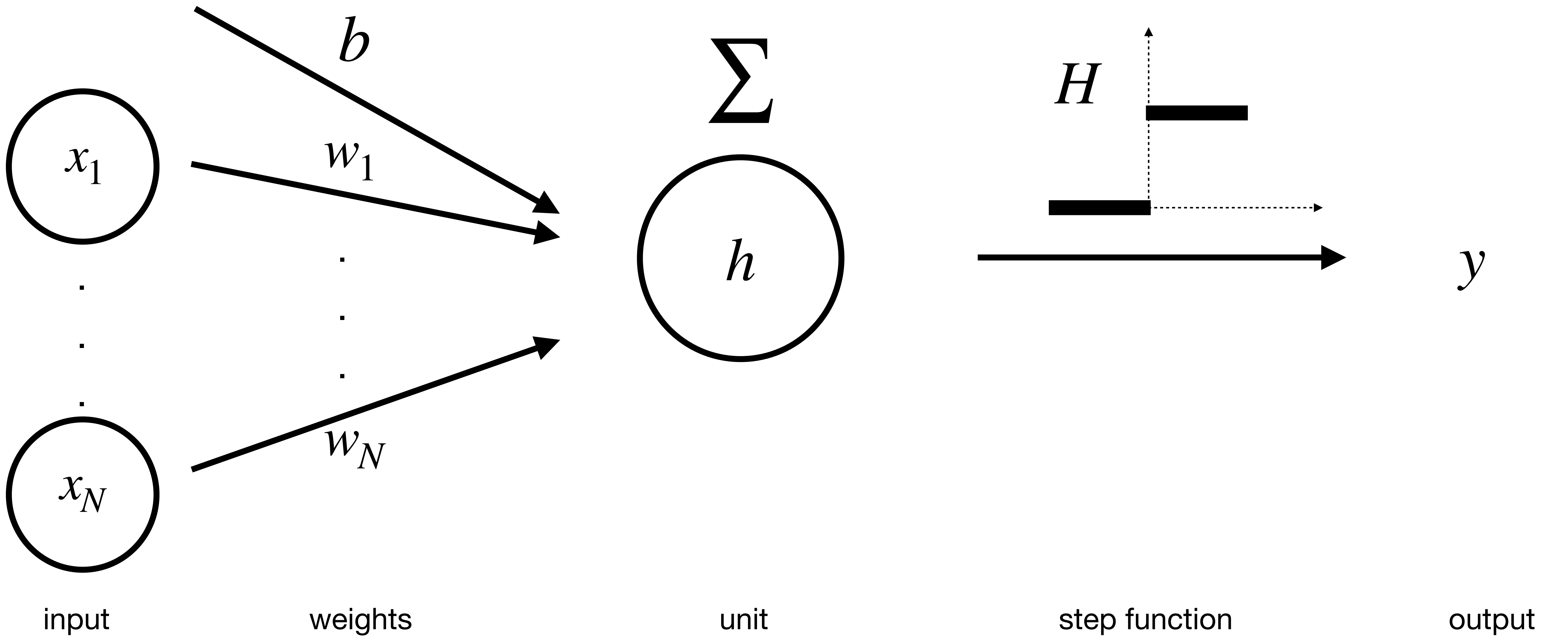
3. Training loop

Perceptron



Parametric functions to model the relation between input $x = (x_1, \ldots, x_N)$ and output y as $y = f(x; \theta)$

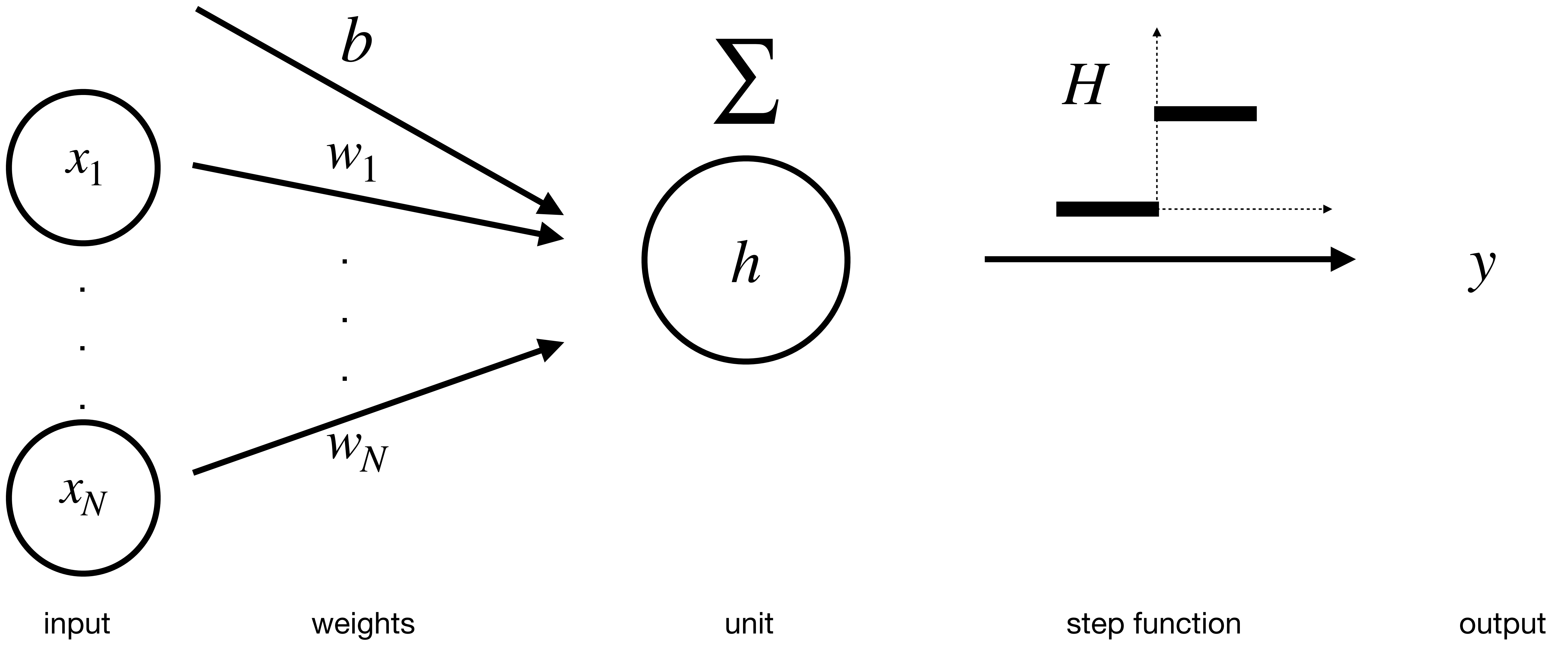
Perceptron



Parametric functions to model the relation between input $x = (x_1, \ldots, x_N)$ and output y as $y = f(x; \theta)$

- θ is the collection of parameters $\theta = \{w = (w_1, \ldots, w_N), b\}$, weights and bias
- N is the dimension of the input space or number of input features
- Hidden unit $h = x \cdot w + b = \sum_{i=1}^{N} w_i x_i + b$
- Output y is computed as $f(x; \theta) = H(h) = H(x \cdot w + b)$

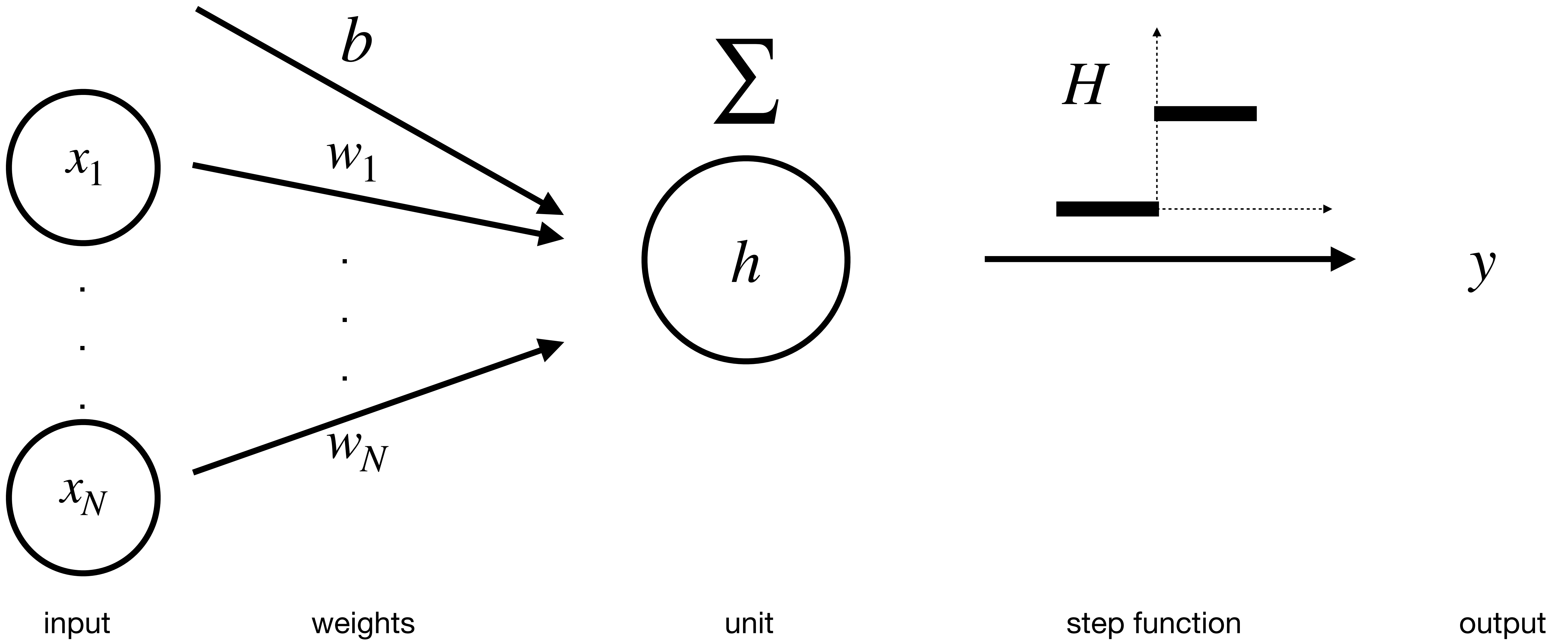
Perceptron



Parametric functions to model the relation between input $x = (x_1, \ldots, x_N)$ and output y as $y = f(x; \theta)$

1. How many weights w do we need?
2. How many bias b do we need?

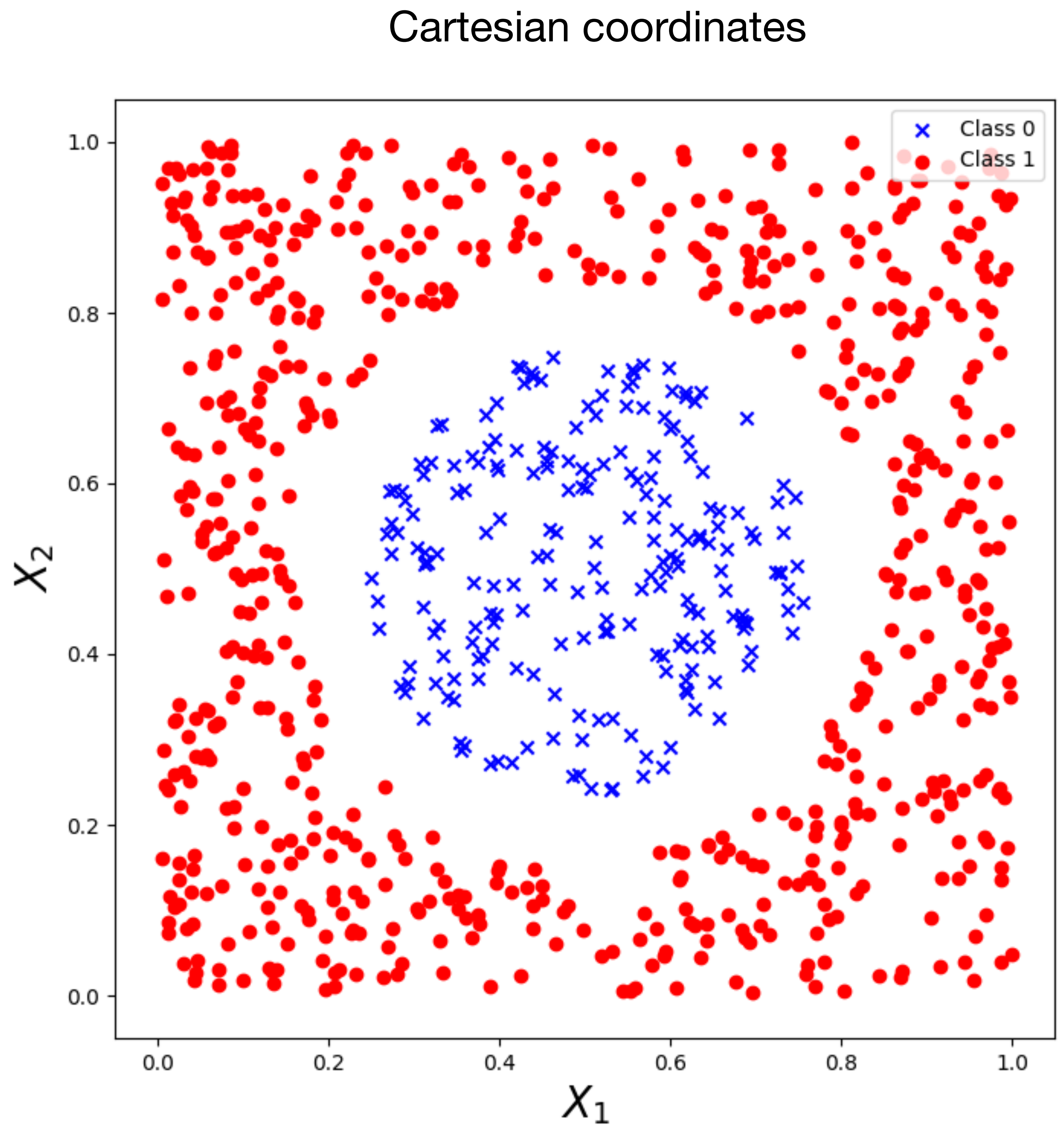
Perceptron

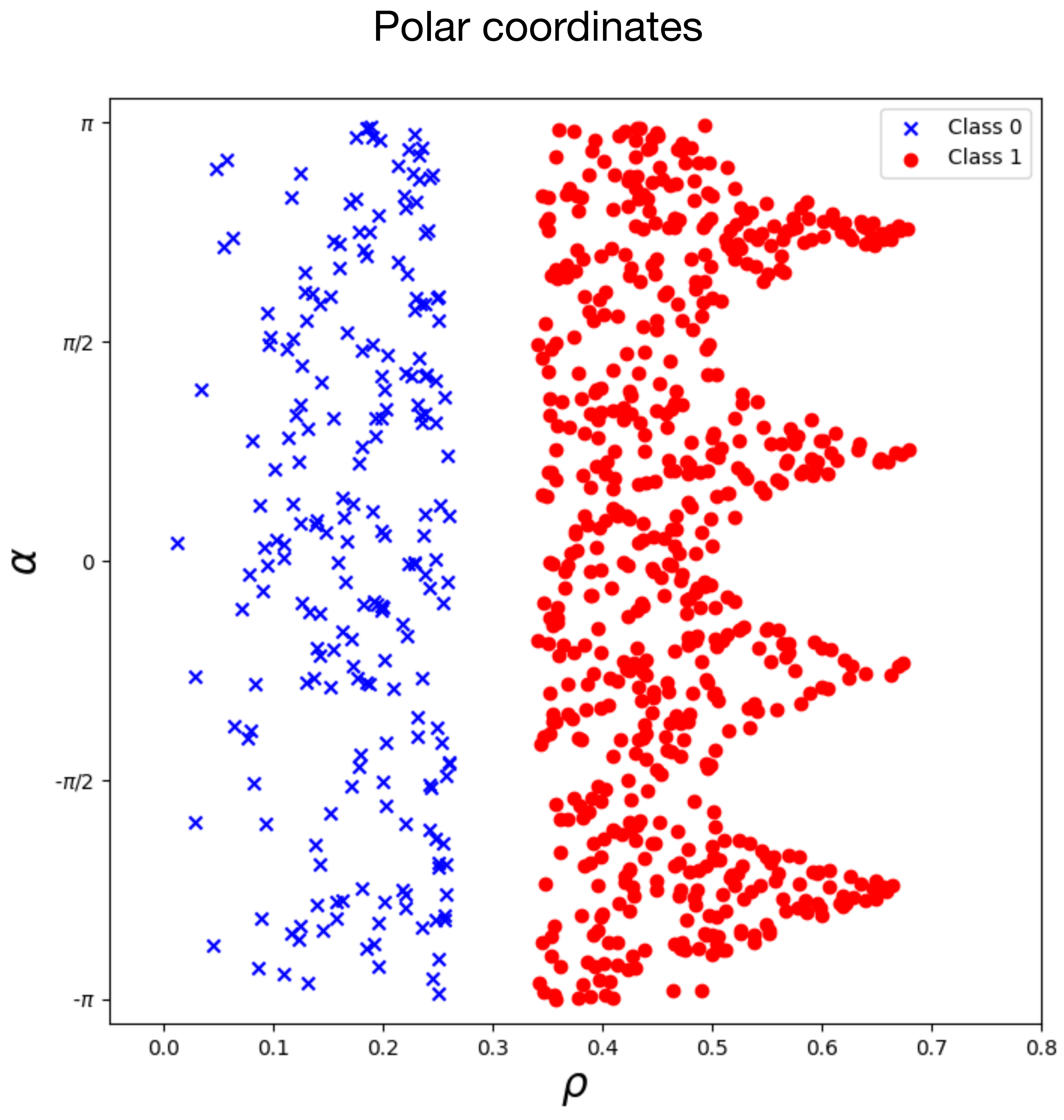


Parametric functions to model the relation between input $x = (x_1, \ldots, x_N)$ and output y as $y = f(x; \theta)$

1. How many weights w do we need? N as the input dimension
2. How many bias b do we need? 1 as the number of hidden units

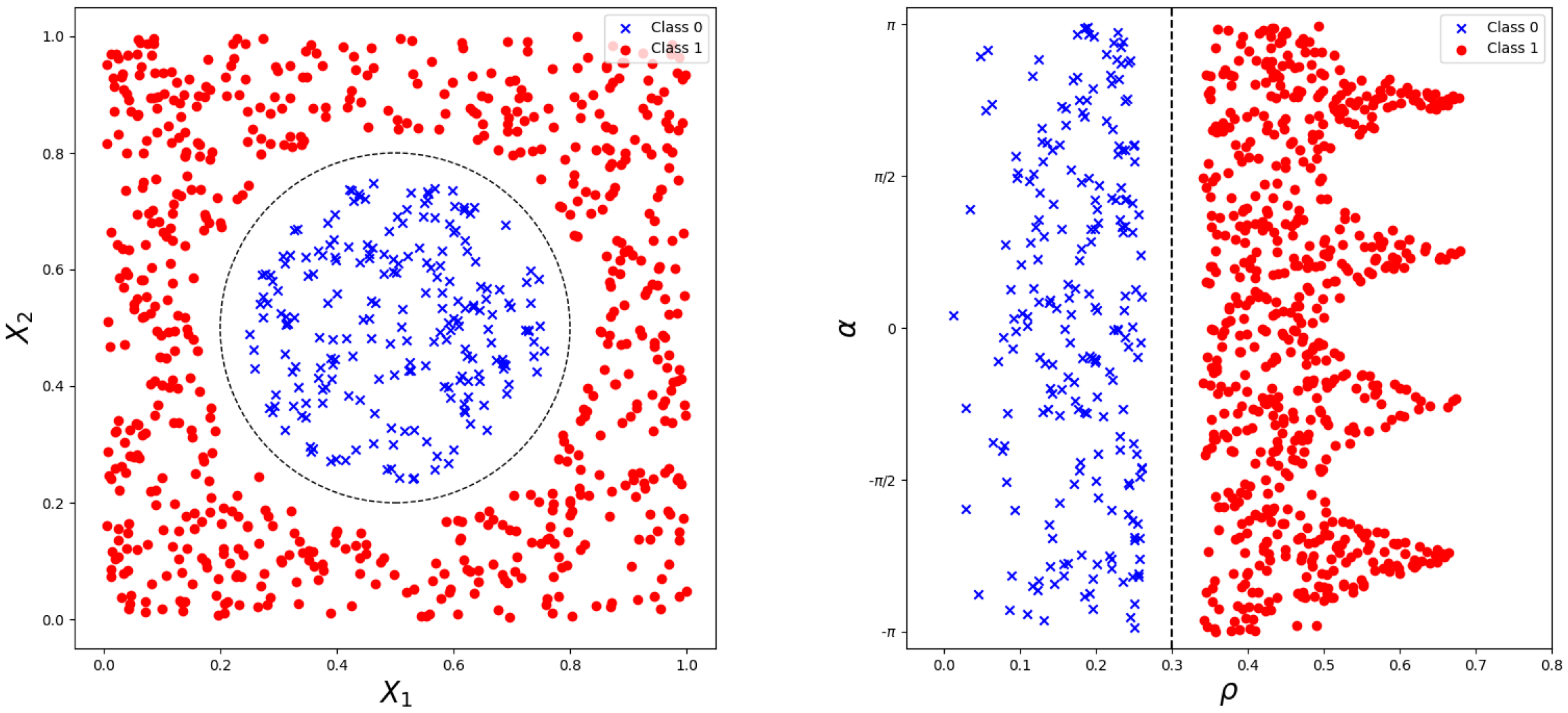
Running Example





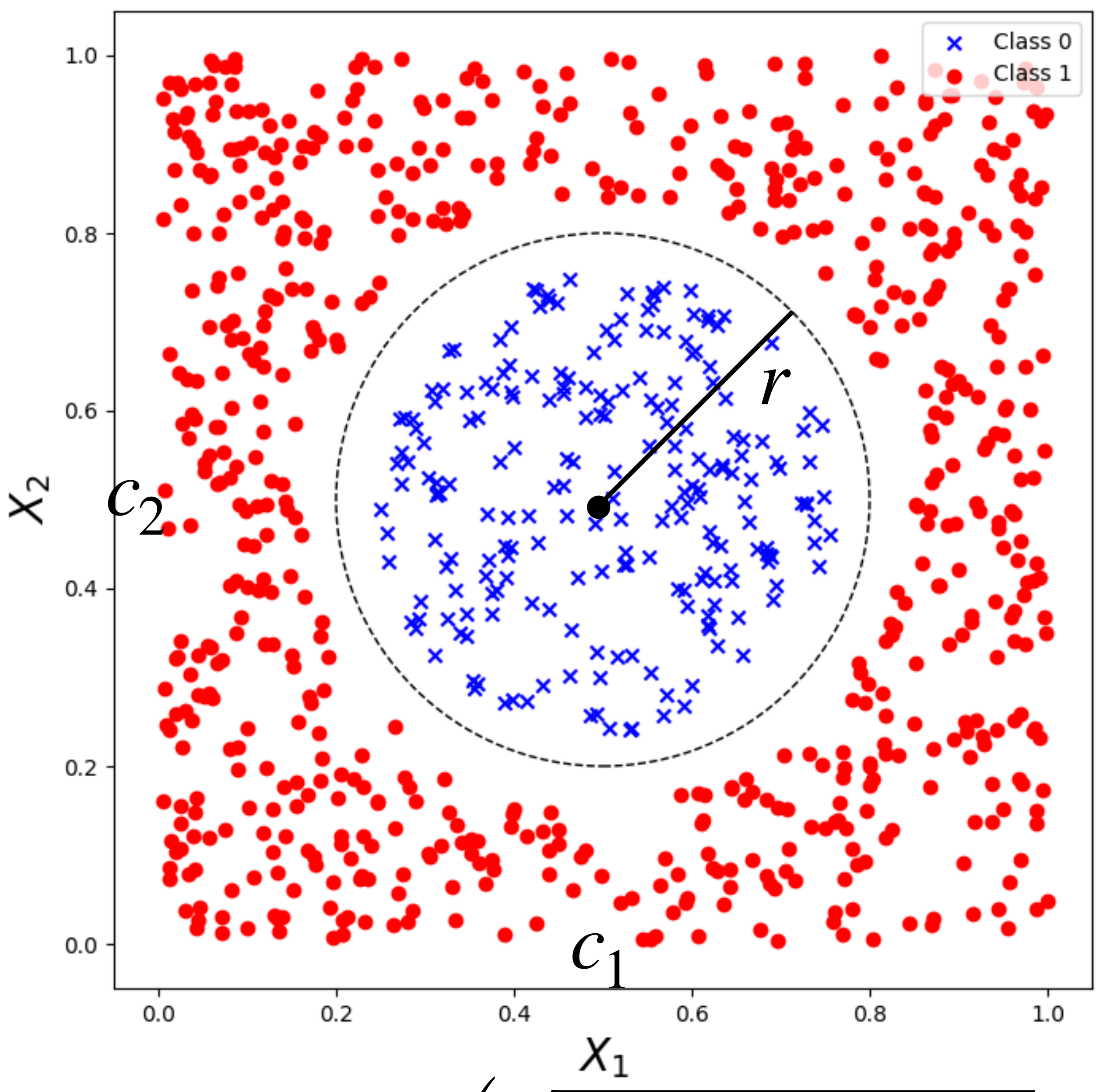
Binary classification task: distinguish between two classes regular devices (class 0) and suspicious devices (class 1)

Running Example

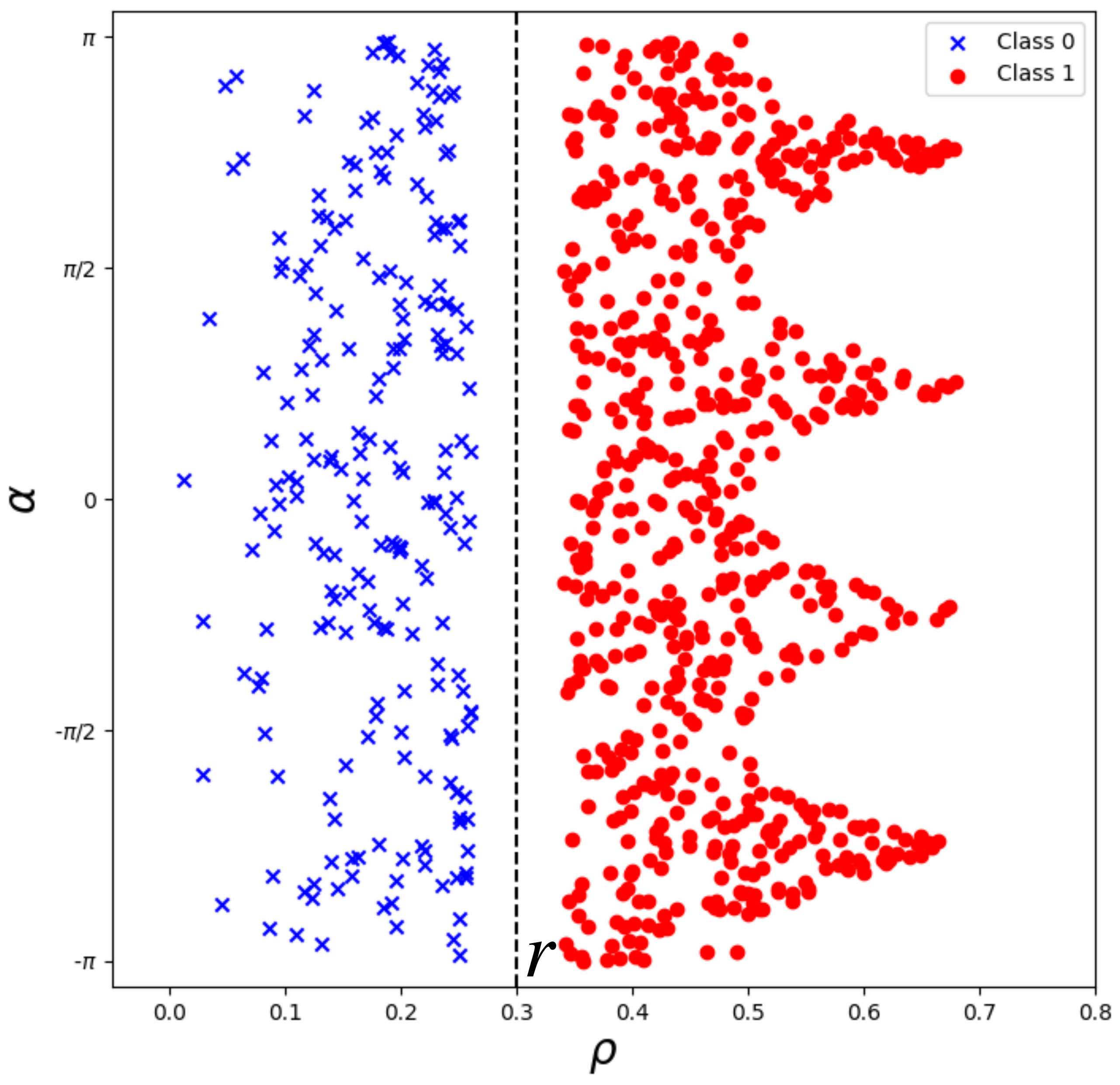


Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Running Example



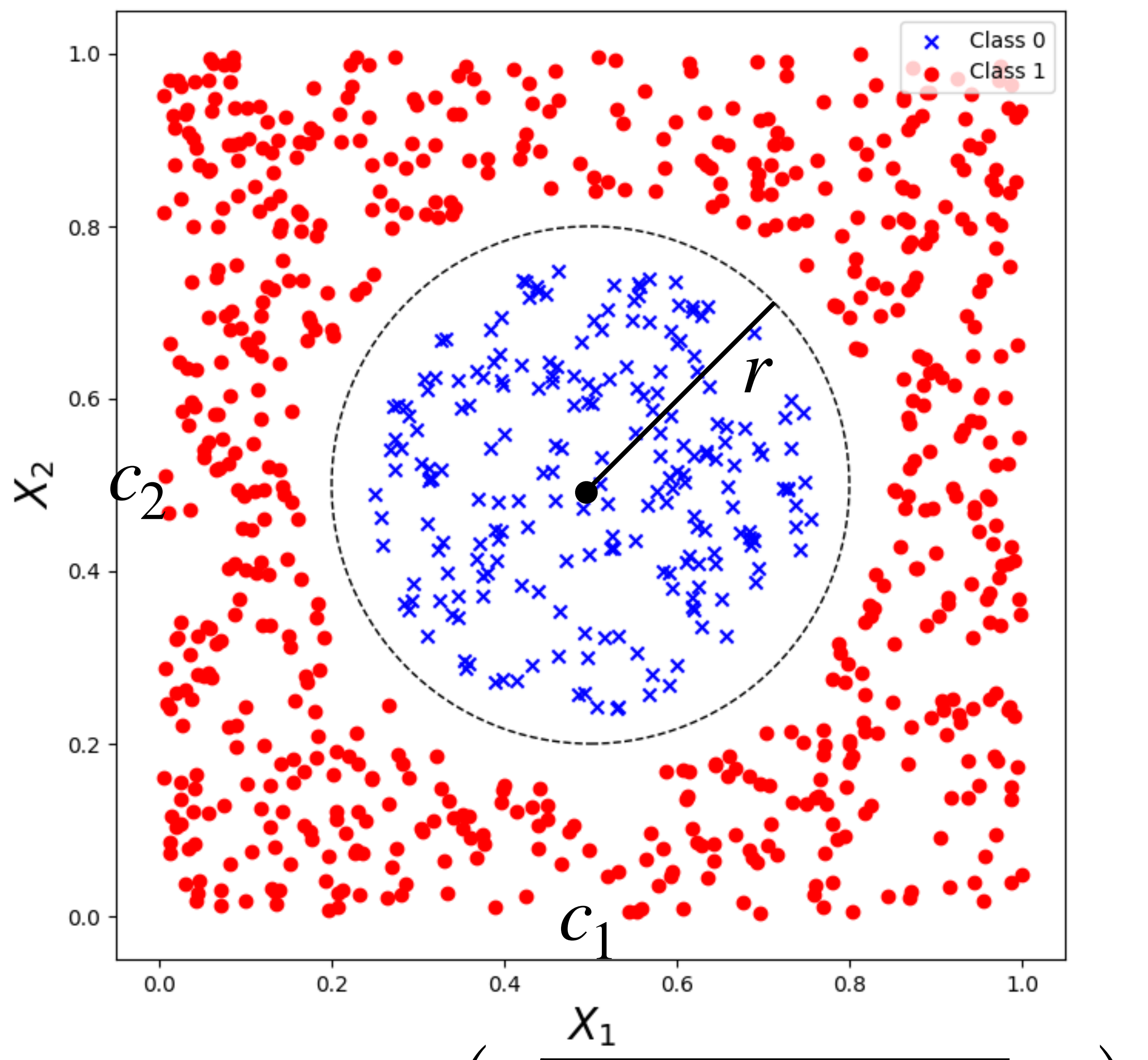
$$f^*(x_1, x_2) = H\left(\sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2} - r\right)$$



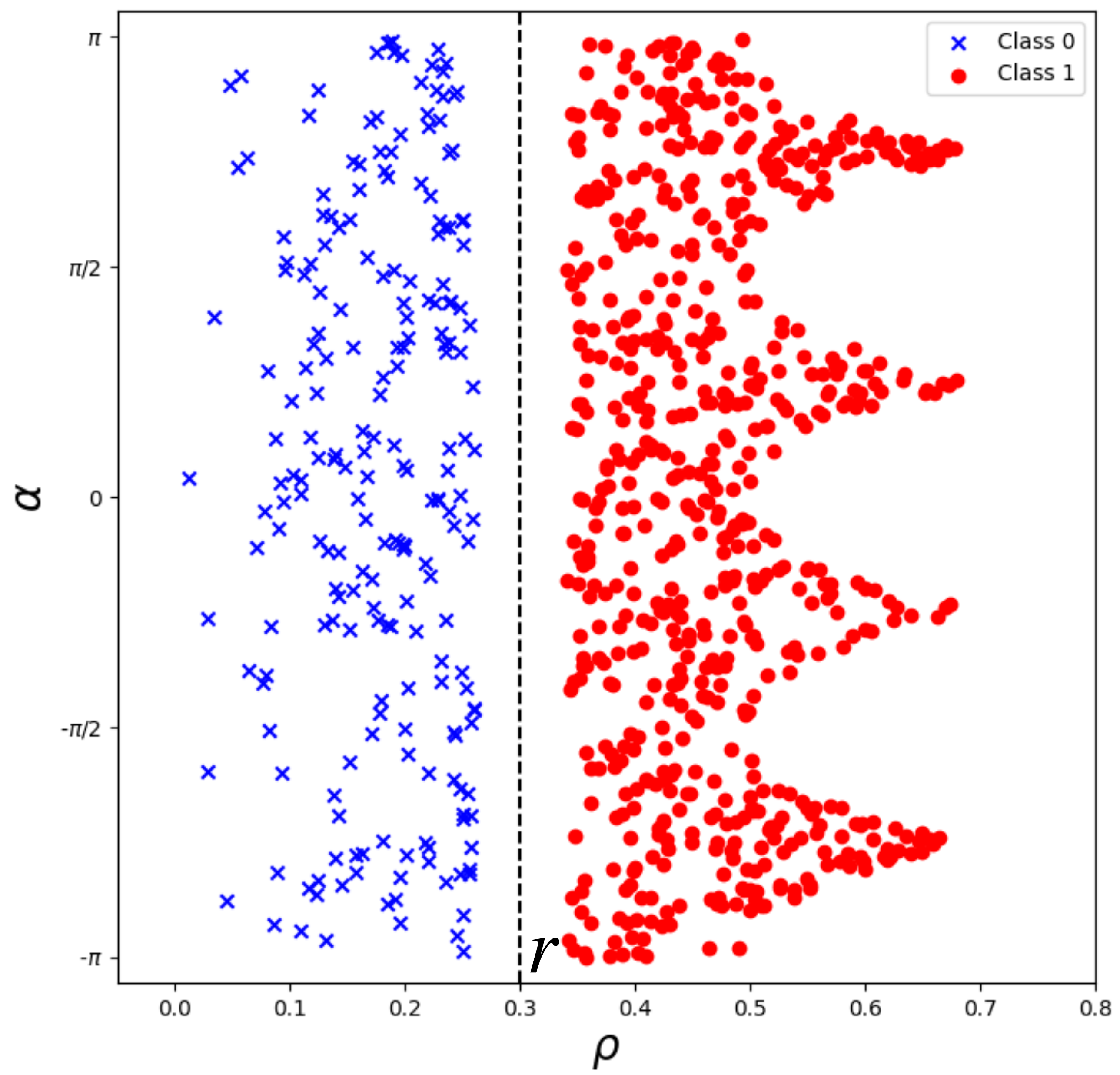
$$f^*(\rho, \alpha) = H(\rho - r)$$

Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Running Example



$$f^*(x_1, x_2) = H\left(\sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2} - r\right)$$

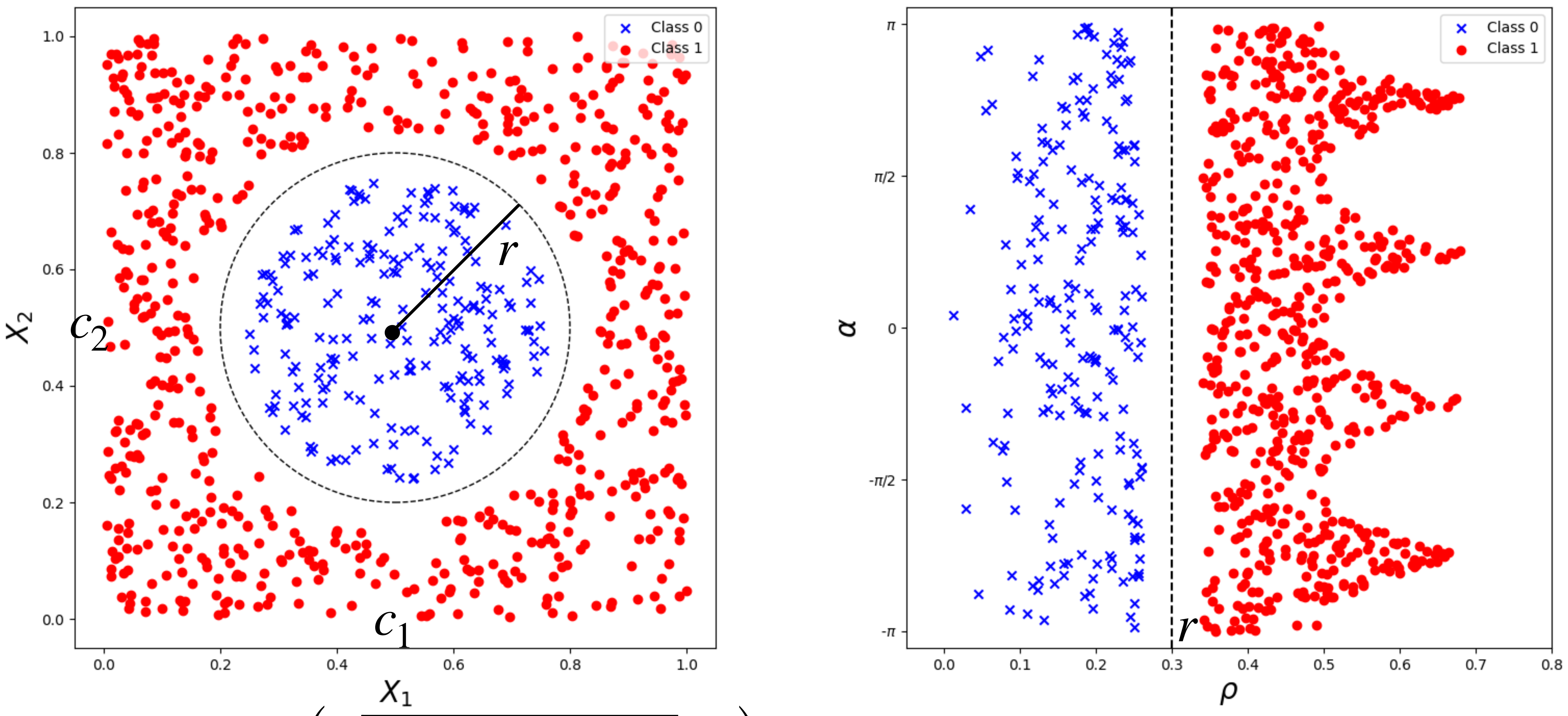


$$f^*(\rho, \alpha) = H(\rho - r)$$

Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,;\theta)$ to approximate f^* $\longrightarrow$ Find w_1, w_2, b such that $f(x_1, x_2; \theta) = H((x_1, x_2) \cdot (w_1, w_2) + b) \approx f^*(x_1, x_2)$

Running Example



$$f^*(x_1, x_2) = H\left(\sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2} - r\right)$$

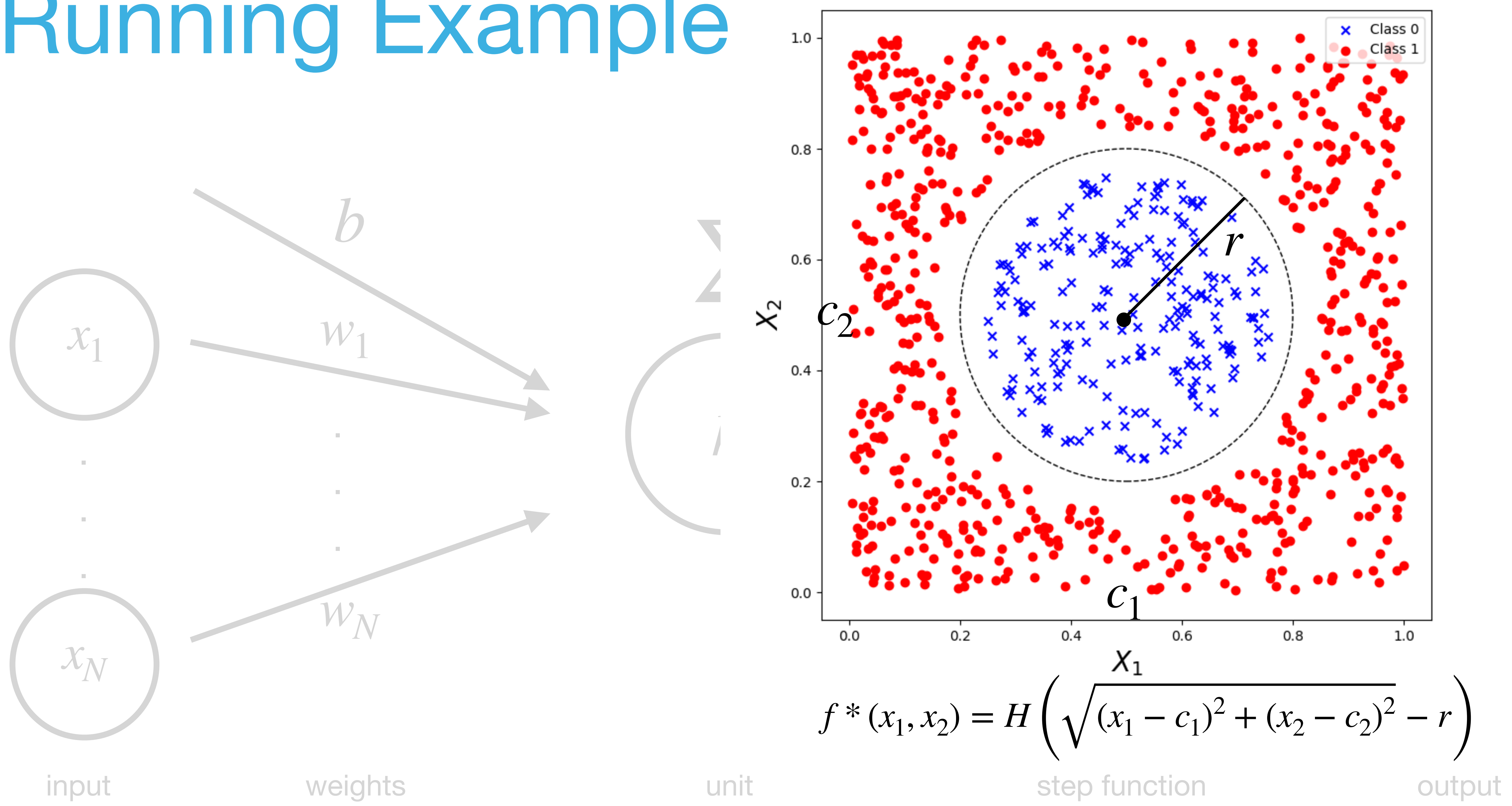
$$f^*(\rho, \alpha) = H(\rho - r)$$

Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,;\theta)$ to approximate f^* $\longrightarrow$ Find w_1, w_2, b such that $f(x_1, x_2; \theta) = H((x_1, x_2) \cdot (w_1, w_2) + b) \approx f^*(x_1, x_2)$

Can the perceptron solve sufficiently accurately this classification task?

Running Example



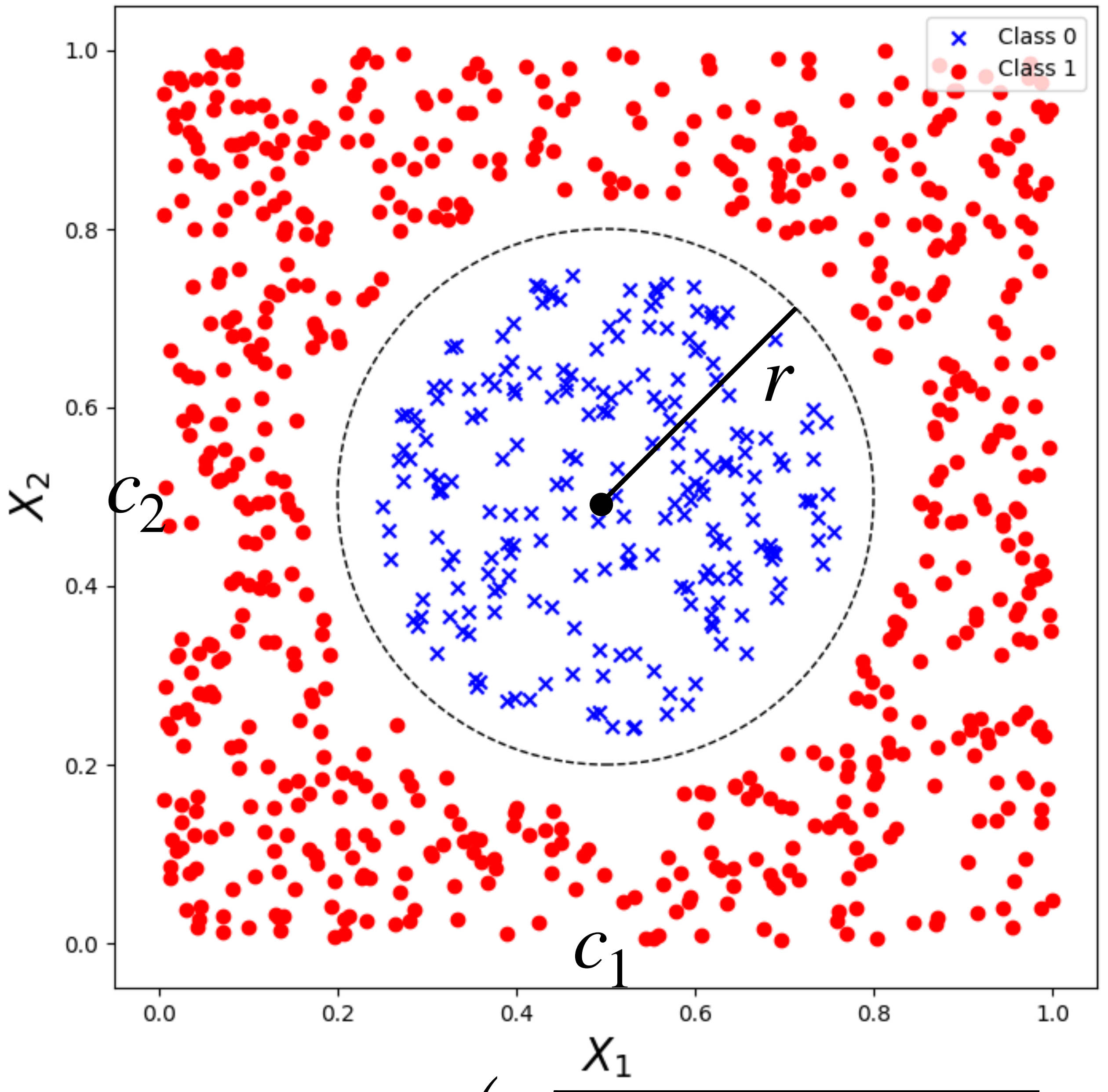
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,; \theta)$ to approximate f^* $\longrightarrow$ Find w_1, w_2, b such that $f(x_1, x_2; \theta) = H((x_1, x_2) \cdot (w_1, w_2) + b) \approx f^*(x_1, x_2)$

Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries

Running Example



$$f*(x_1, x_2) = H\left(\sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2} - r\right)$$

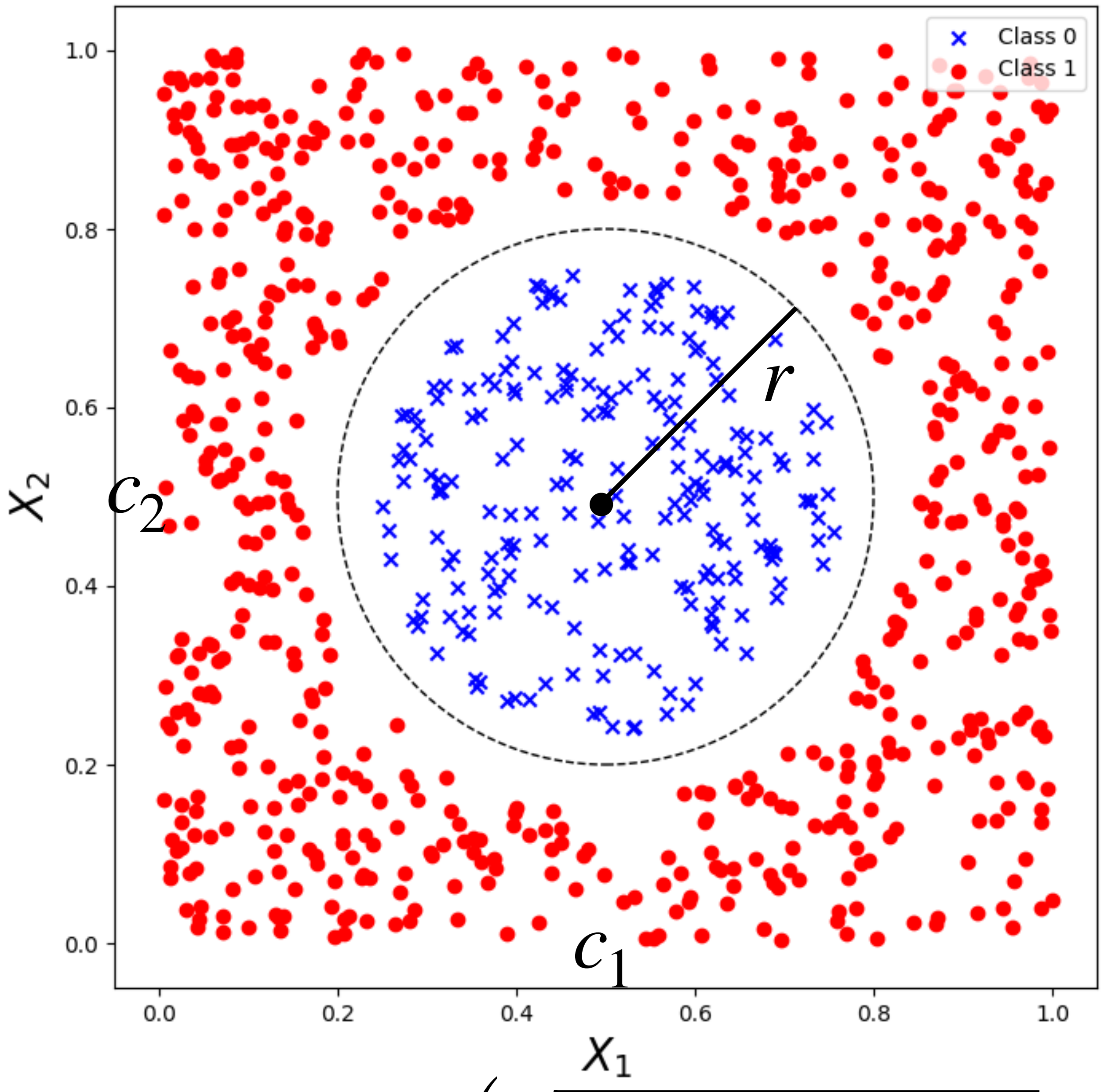
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,;\theta)$ to approximate f^* $\longrightarrow$ Find w_1, w_2, b such that $f(x_1, x_2; \theta) = \boxed{H((x_1, x_2) \cdot (w_1, w_2) + b)} \approx f^*(x_1, x_2)$

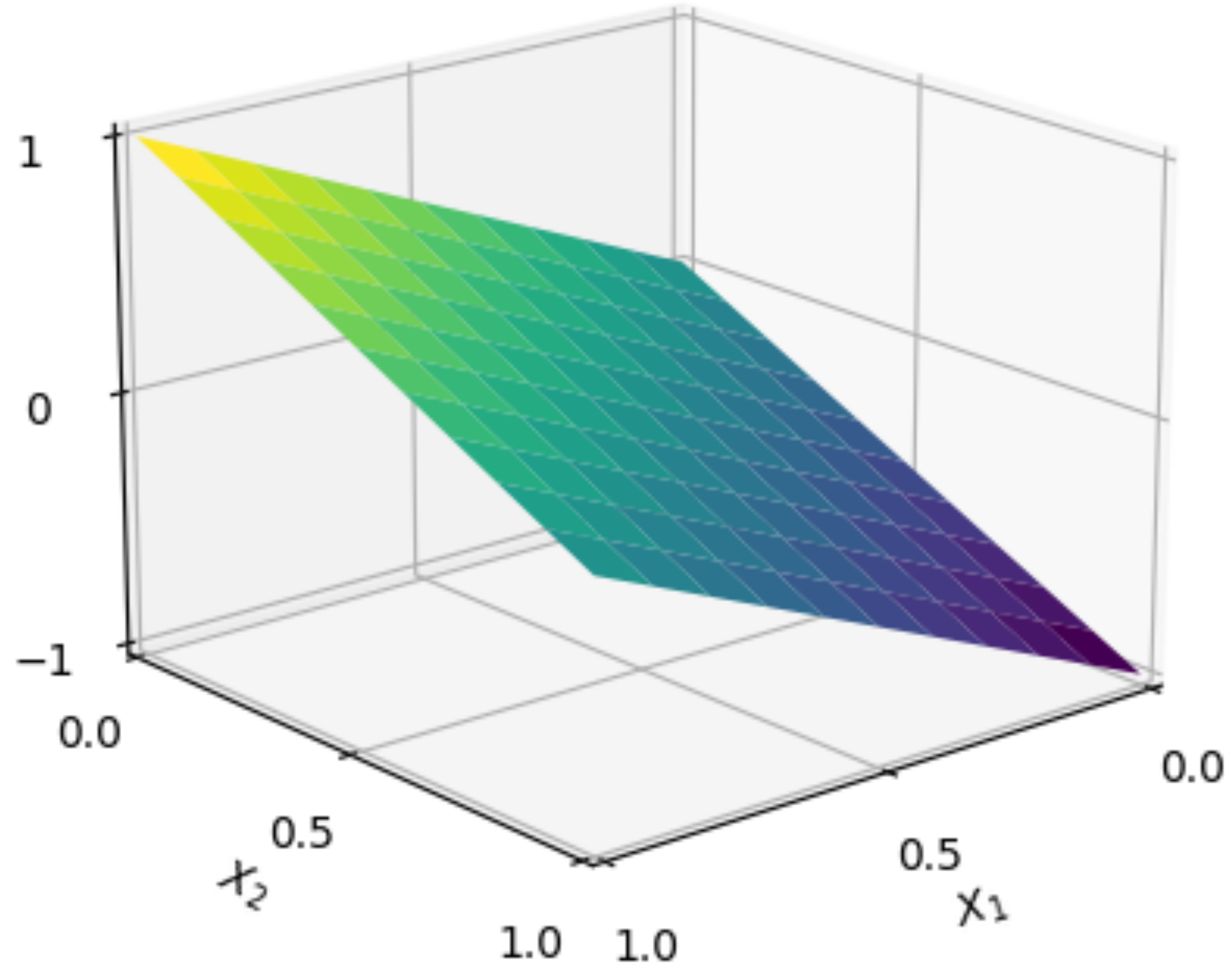
Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries

Running Example



$$f^*(x_1, x_2) = H\left(\sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2} - r\right)$$



$$(x_1, x_2) \mapsto (x_1, x_2) \cdot (w_1, w_2) + b = h$$

input weights unit step function output

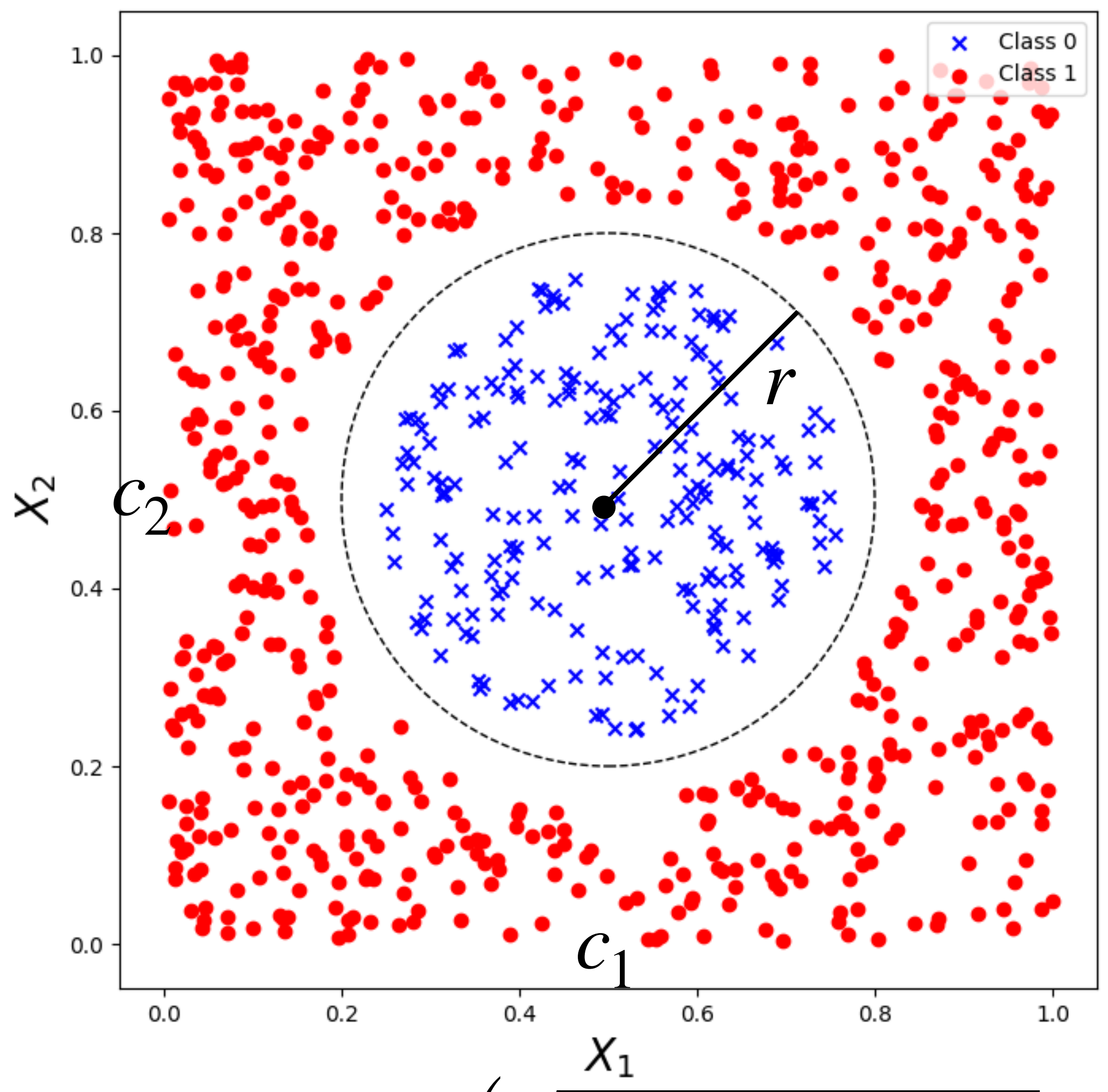
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,; \theta)$ to approximate f^* $\longrightarrow$ Find w_1, w_2, b such that $f(x_1, x_2; \theta) = H(\boxed{(x_1, x_2) \cdot (w_1, w_2) + b}) \approx f^*(x_1, x_2)$

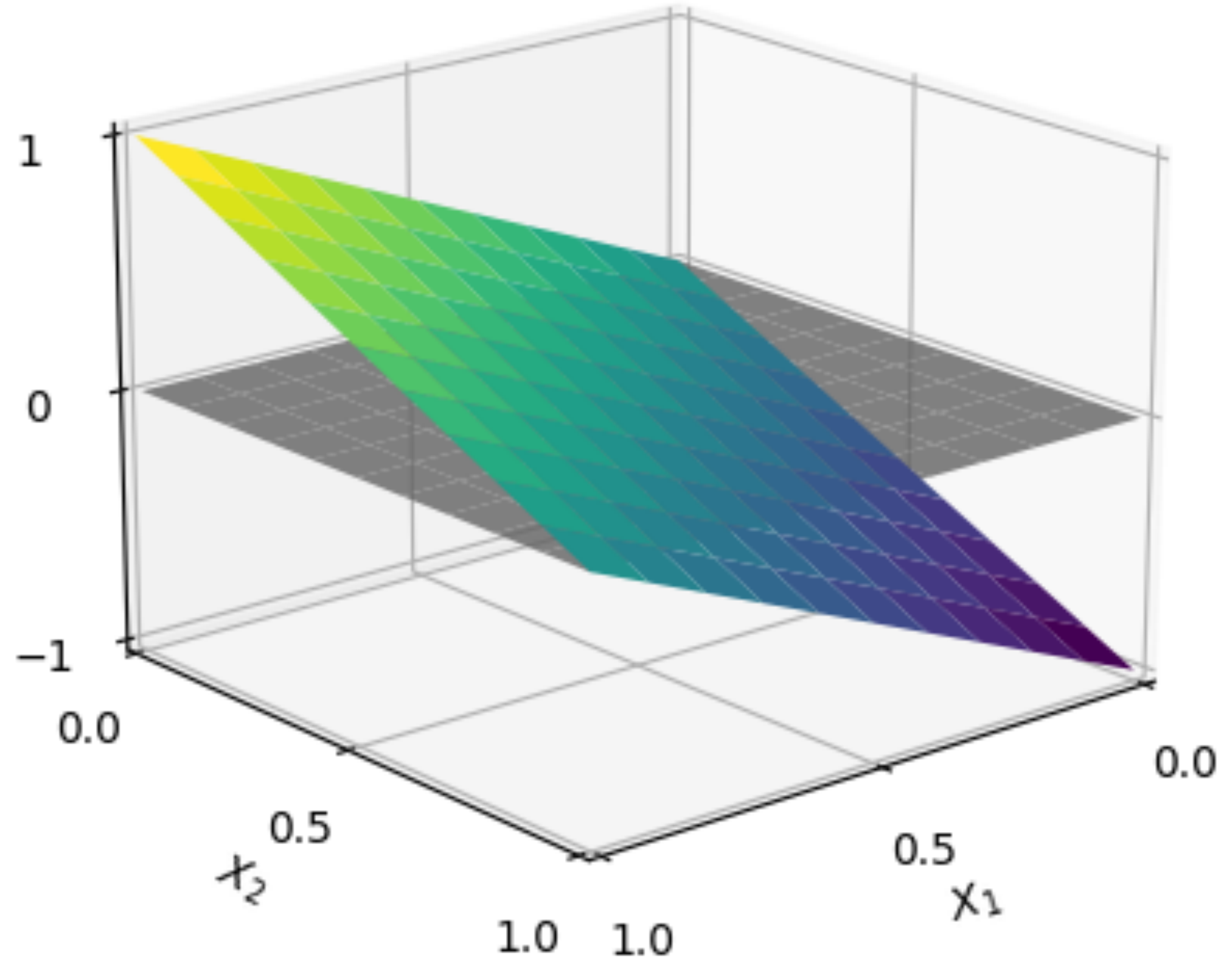
Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries

Running Example



$$f^*(x_1, x_2) = H\left(\sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2} - r\right)$$



$$(x_1, x_2) \mapsto (x_1, x_2) \cdot (w_1, w_2) + b = h$$

$$H(h) = \begin{cases} 1 & h \geq 0 \\ 0 & h < 0 \end{cases}$$

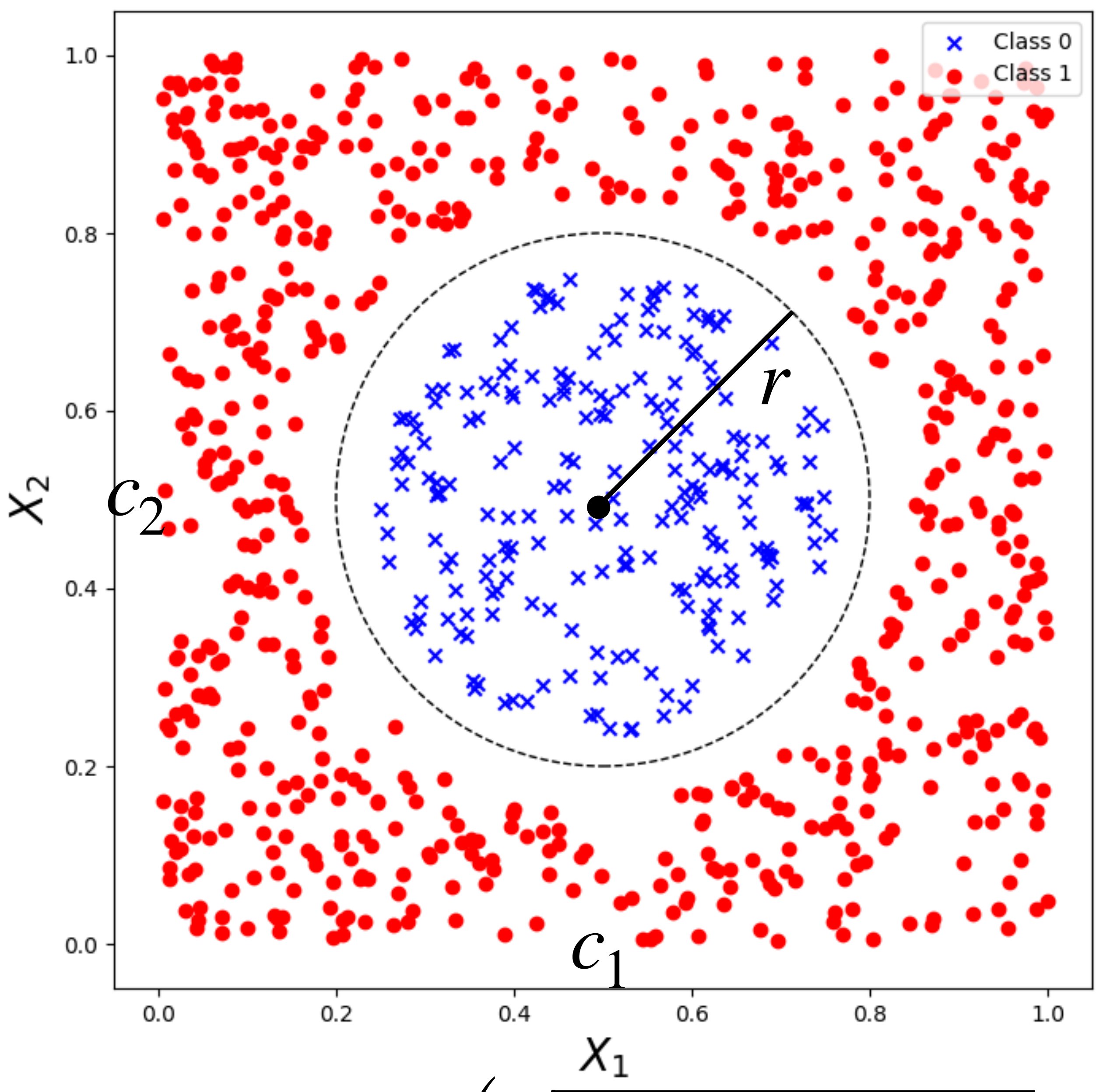
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,;\theta)$ to approximate f^* $\longrightarrow$ Find w_1, w_2, b such that $f(x_1, x_2; \theta) = H(\boxed{(x_1, x_2) \cdot (w_1, w_2) + b}) \approx f^*(x_1, x_2)$

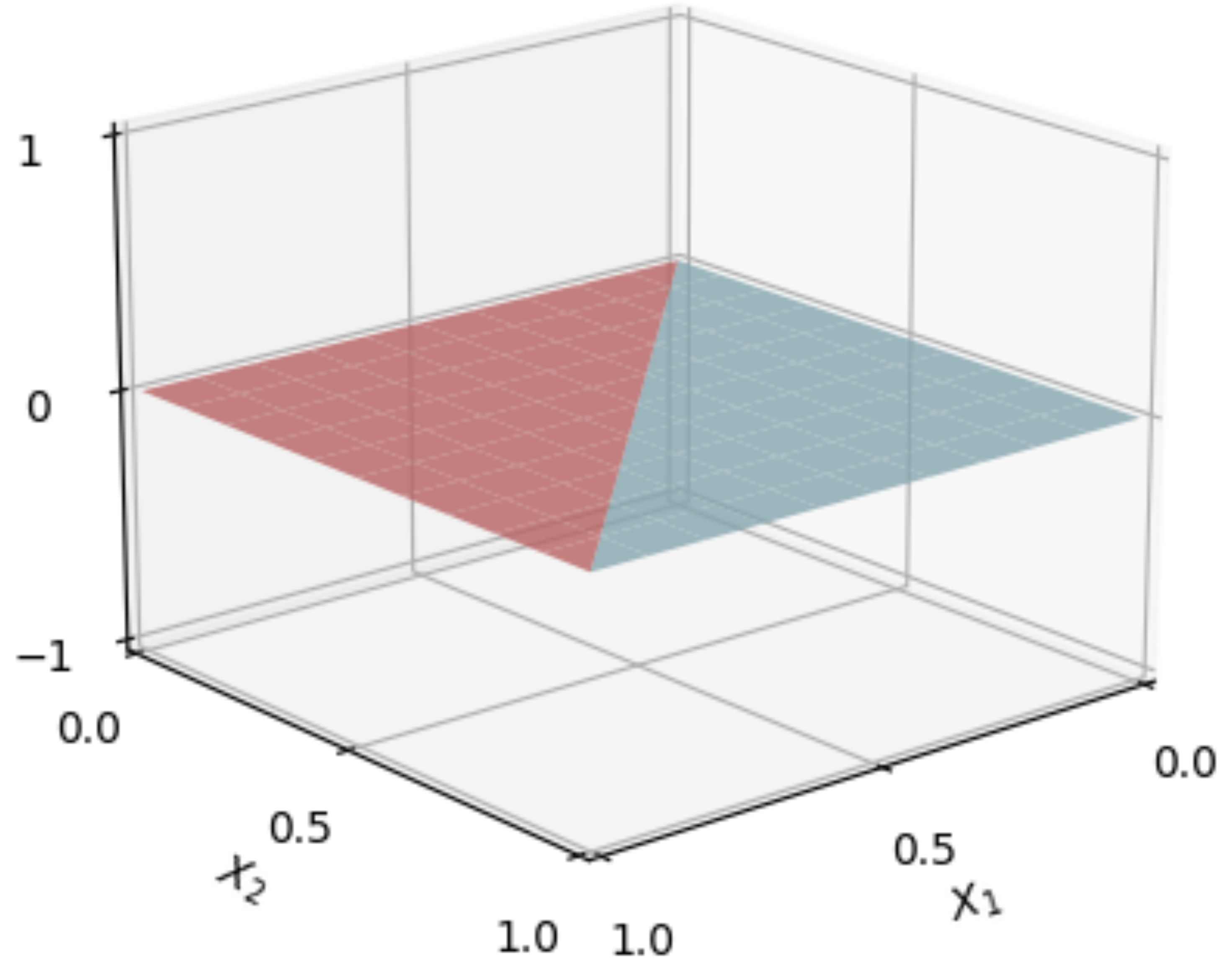
Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries

Running Example



$f^*(x_1, x_2) = H\left(\sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2} - r\right)$



$f(x_1, x_2; \theta) = H((x_1, x_2) \cdot (w_1, w_2) + b)$

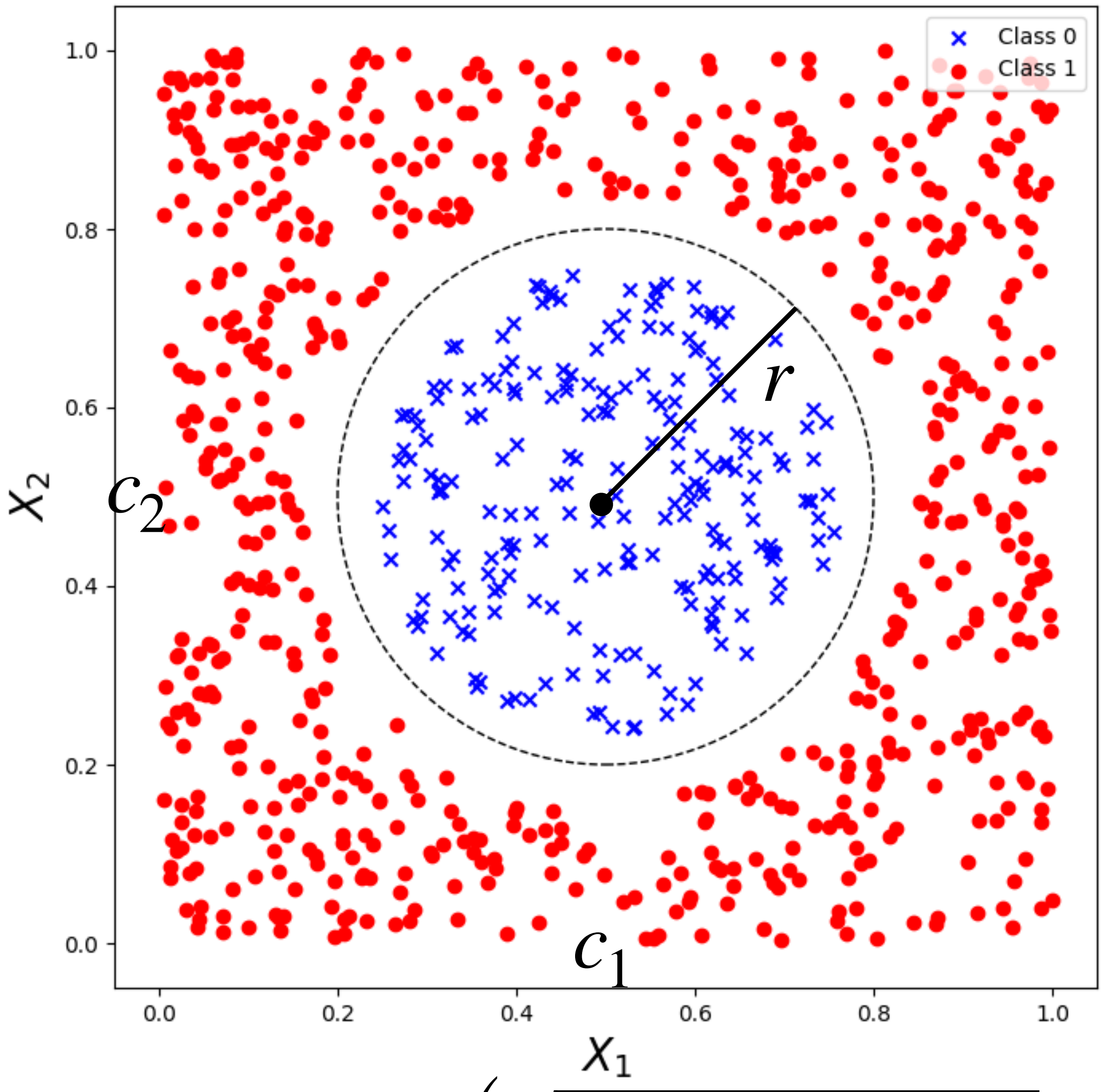
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\,\cdot\,;\theta)$ to approximate f^* $\longrightarrow$ Find w_1, w_2, b such that $f(x_1, x_2; \theta) = \boxed{H((x_1, x_2) \cdot (w_1, w_2) + b)} \approx f^*(x_1, x_2)$

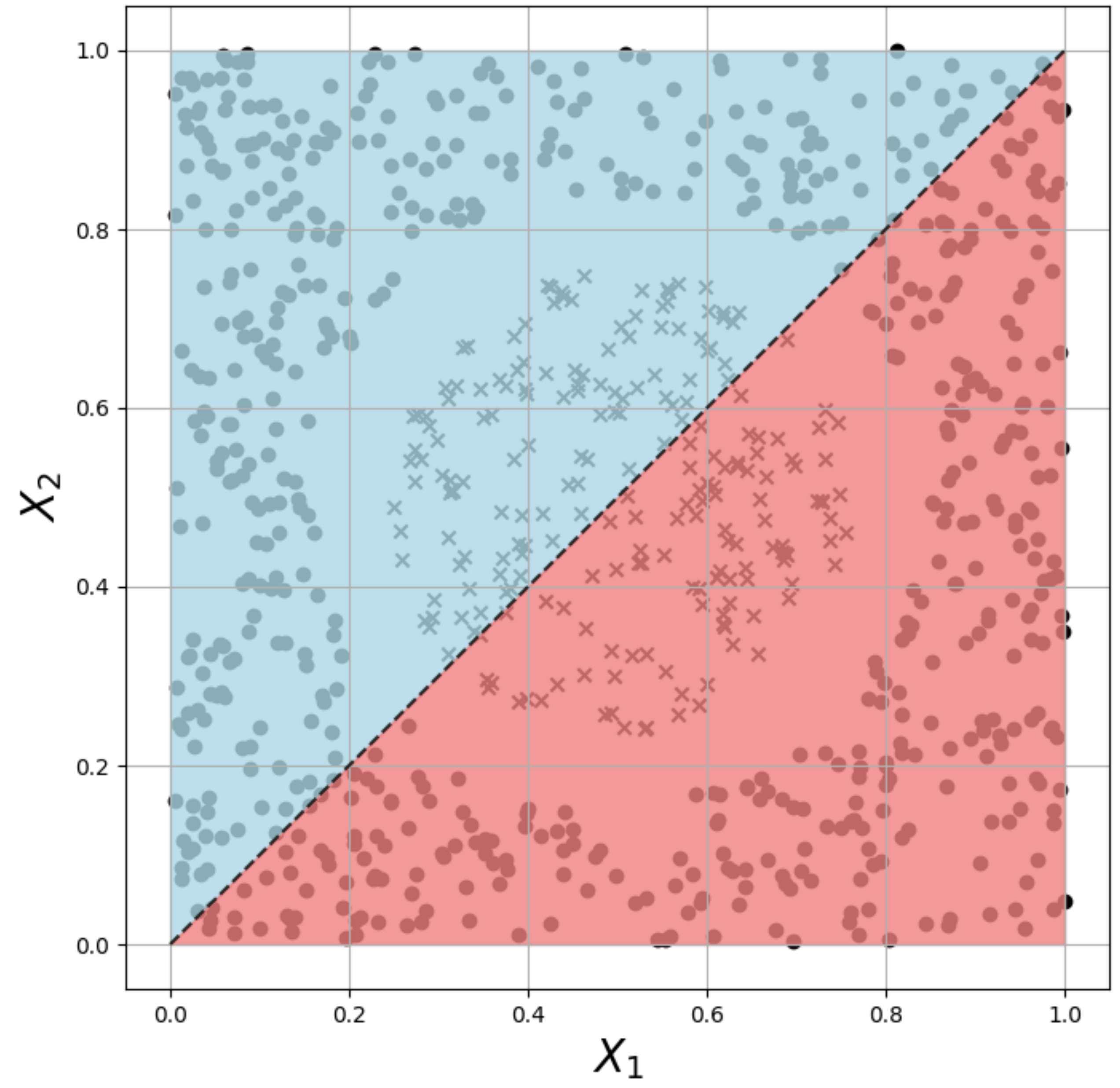
Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries

Running Example



$$f^*(x_1, x_2) = H\left(\sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2} - r\right)$$



$$f(x_1, x_2; \theta) = H((x_1, x_2) \cdot (w_1, w_2) + b)$$

NO WAY!

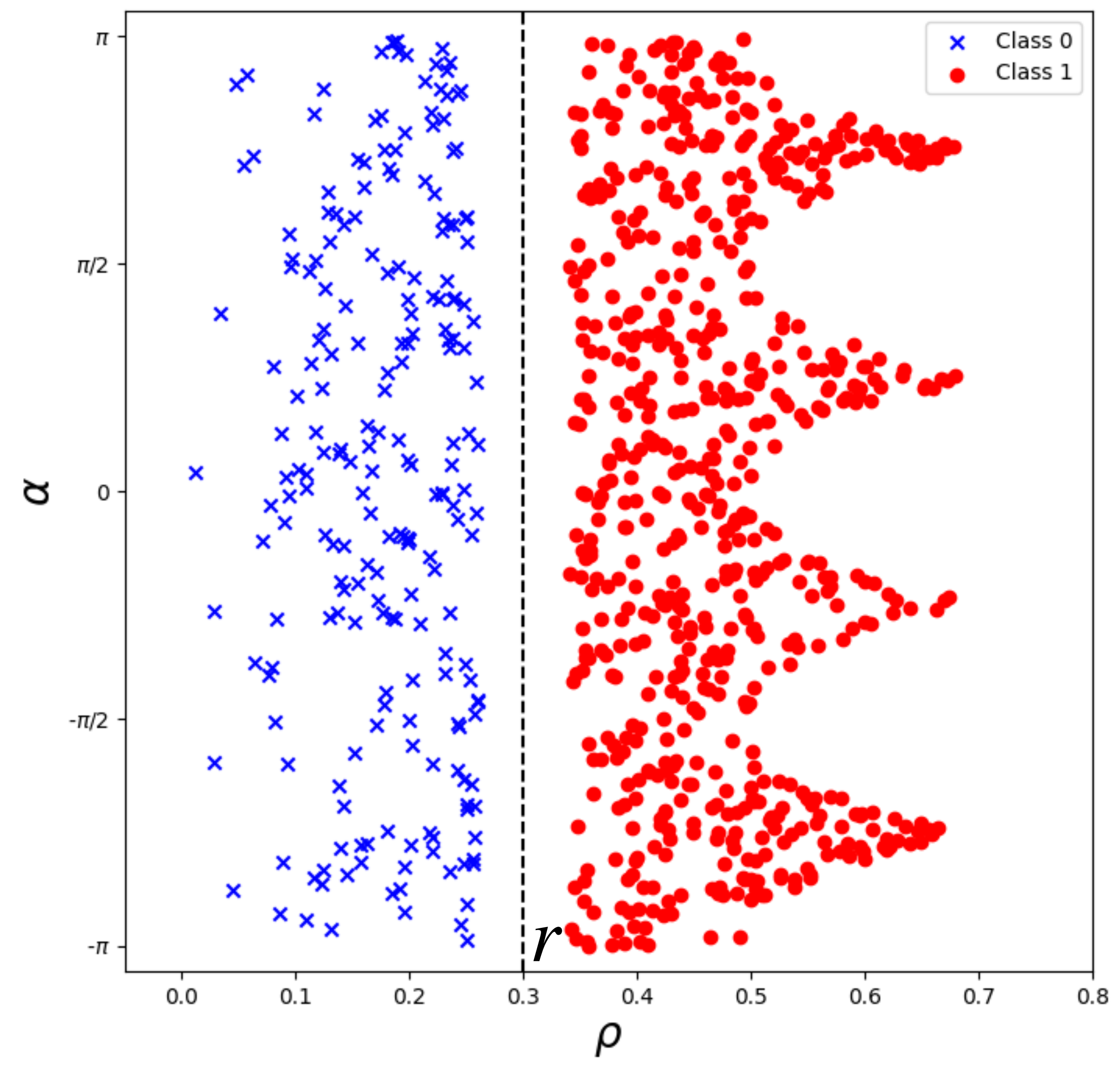
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,; \theta)$ to approximate f^* $\longrightarrow$ Find w_1, w_2, b such that $f(x_1, x_2; \theta) = H((x_1, x_2) \cdot (w_1, w_2) + b) \approx f^*(x_1, x_2)$

Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries

Running Example



$f^*(\rho, \alpha) = H(\rho - r)$

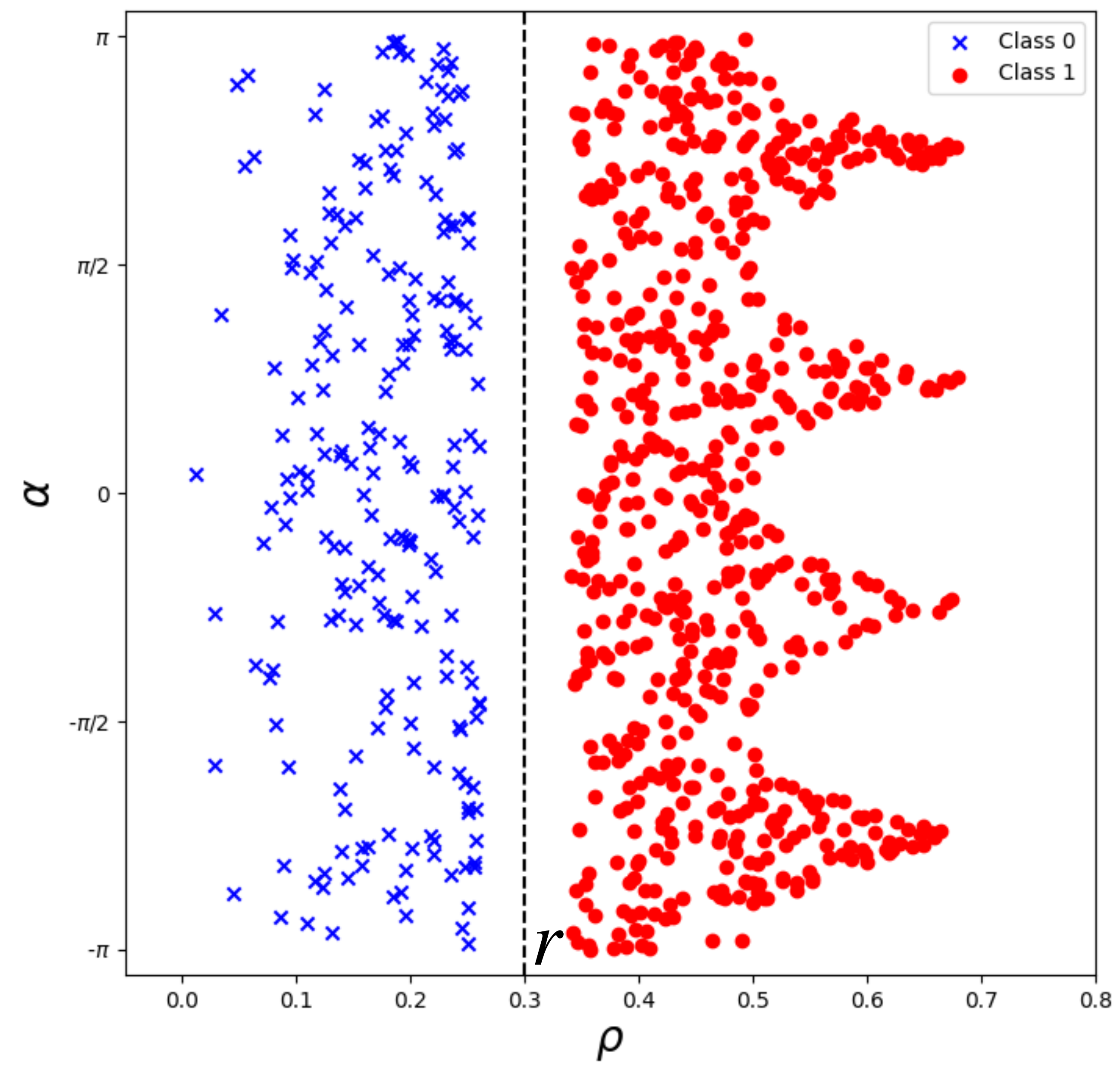
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\,\cdot\,;\theta)$ to approximate f^* $\longrightarrow$ Find w_ρ, w_α, b such that $f(\rho, \alpha; \theta) = H((\rho, \alpha) \cdot (w_\rho, w_\alpha) + b) \approx f^*(\rho, \alpha)$

Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries
- Polar coordinates?

Running Example



$f^*(\rho, \alpha) = H(\rho - r)$

$w_\rho = ?$

$w_\alpha = ?$

$b = ?$

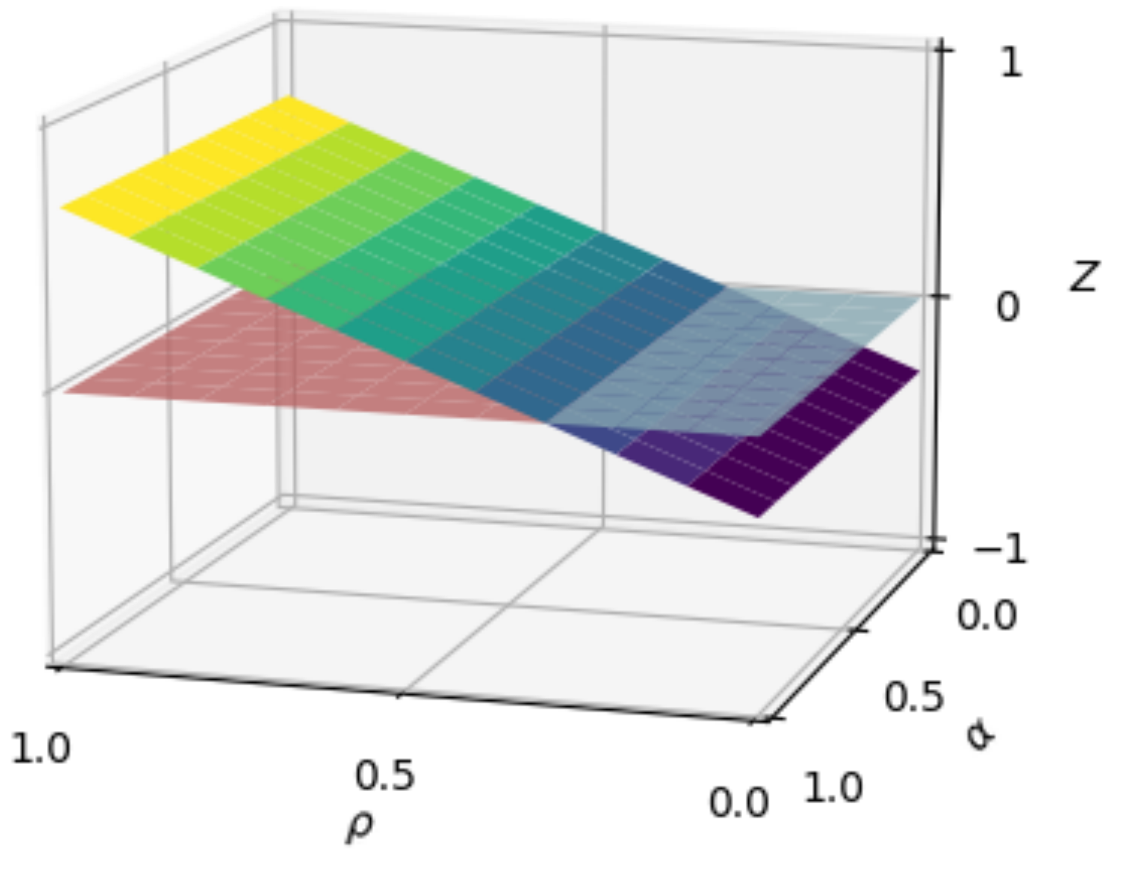
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,; \theta)$ to approximate f^* $\longrightarrow$ Find w_ρ, w_α, b such that $\boxed{f(\rho, \alpha; \theta) = H((\rho, \alpha) \cdot (w_\rho, w_\alpha) + b)} \approx f^*(\rho, \alpha)$

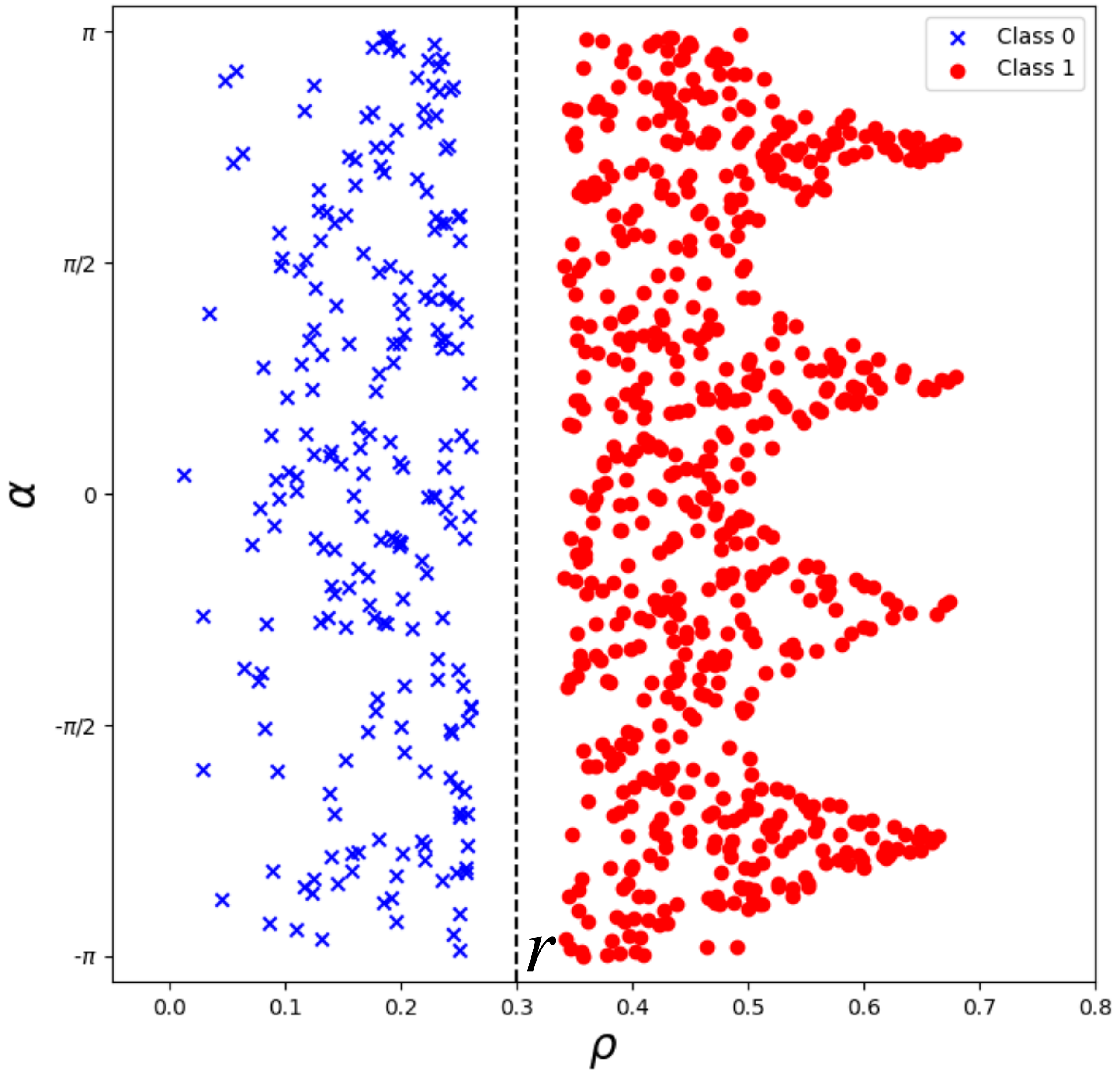
Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries
- Polar coordinates? yes, it is linearly separable. What perceptron solves this problem?

Running Example



$f(\rho, \alpha; \theta) = H(\rho w_\rho + \alpha w_\alpha + b)$



$f^*(\rho, \alpha) = H(\rho - r)$

input weights unit step function output

$w_\rho = 1$

$w_\alpha = 0$

$b = -r$

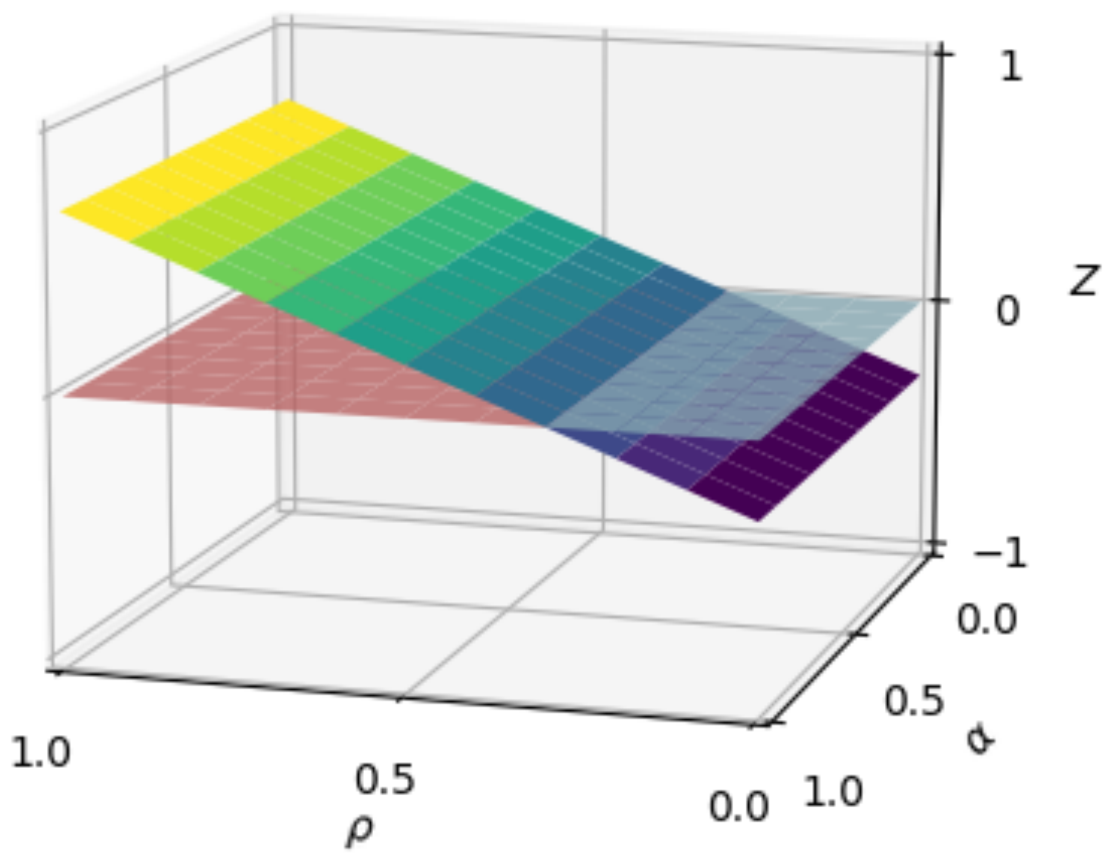
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

Learn parameters θ for $f(\cdot\,;\theta)$ to approximate f^* $\longrightarrow$ Find w_ρ, w_α, b such that $\boxed{f(\rho, \alpha; \theta) = H((\rho, \alpha) \cdot (w_\rho, w_\alpha) + b)} \approx f^*(\rho, \alpha)$

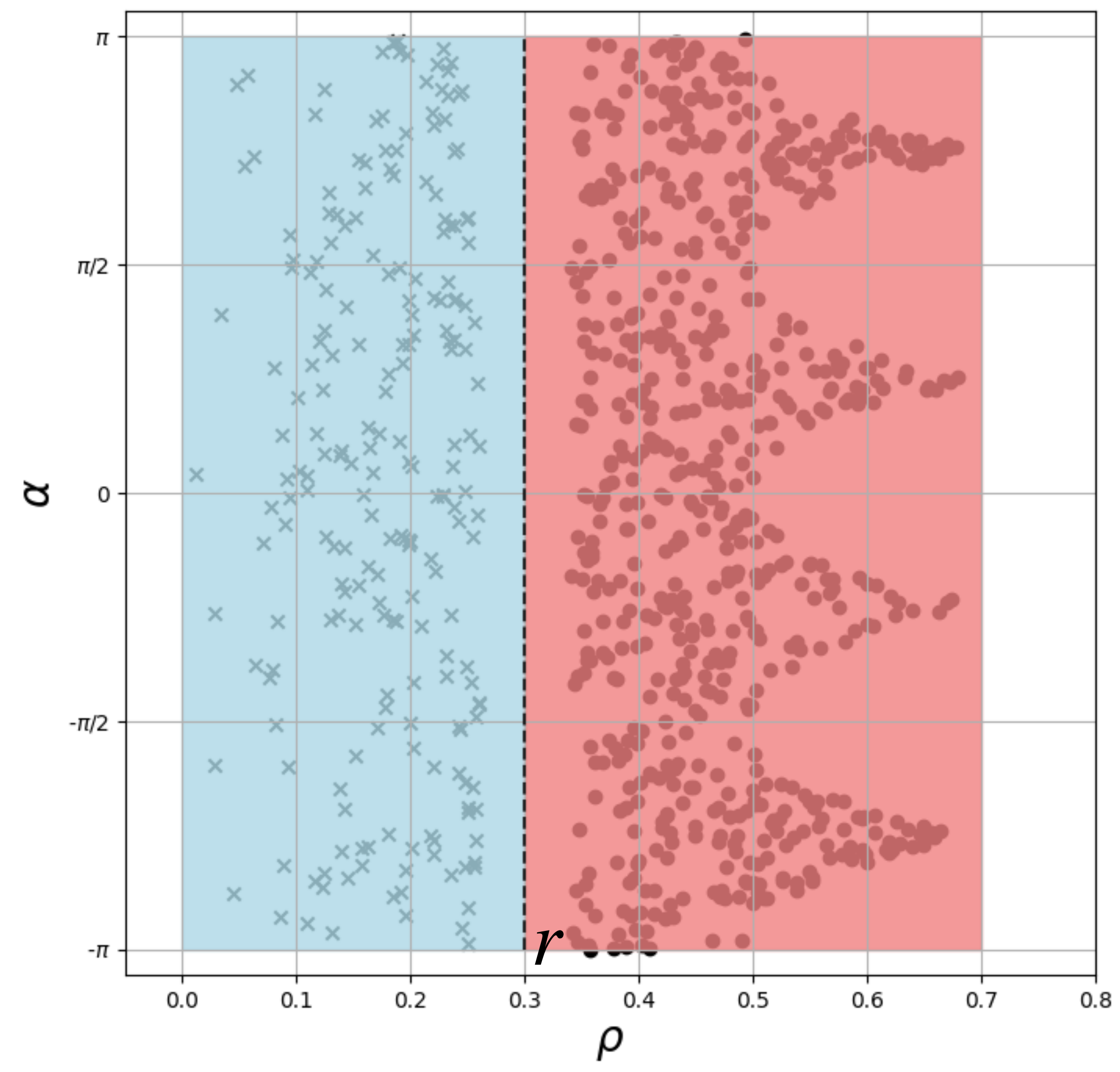
Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries
- Polar coordinates? yes, it is linearly separable. What perceptron solves this problem?

Running Example



$f(\rho, \alpha; \theta) = H(\rho w_\rho + \alpha w_\alpha + b) = H(\rho - r)$



$f^*(\rho, \alpha) = H(\rho - r)$

input weights unit step function output

$w_\rho = 1$

$w_\alpha = 0$

$b = -r$

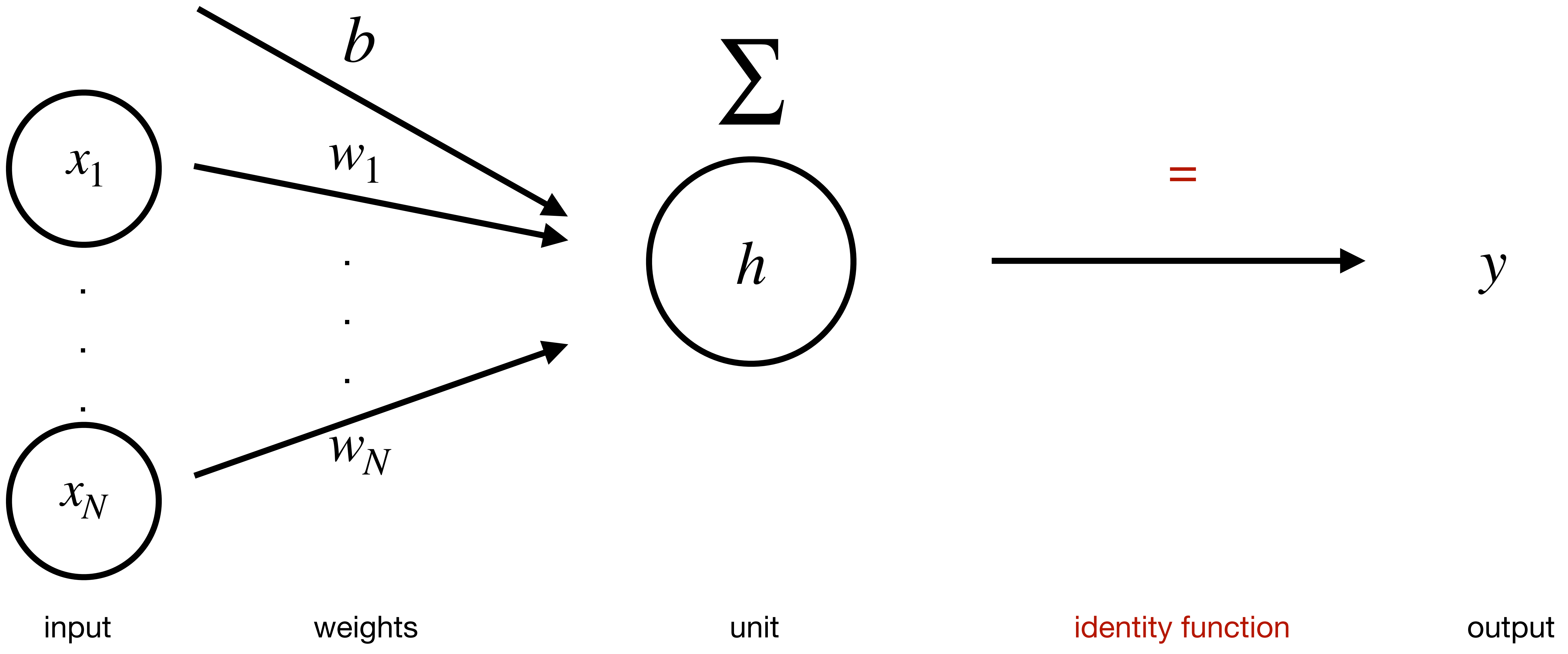
Approximate an objective function f^* from input to output space: e.g. binary classifier $f^*(x) = y \in \{0,1\}$

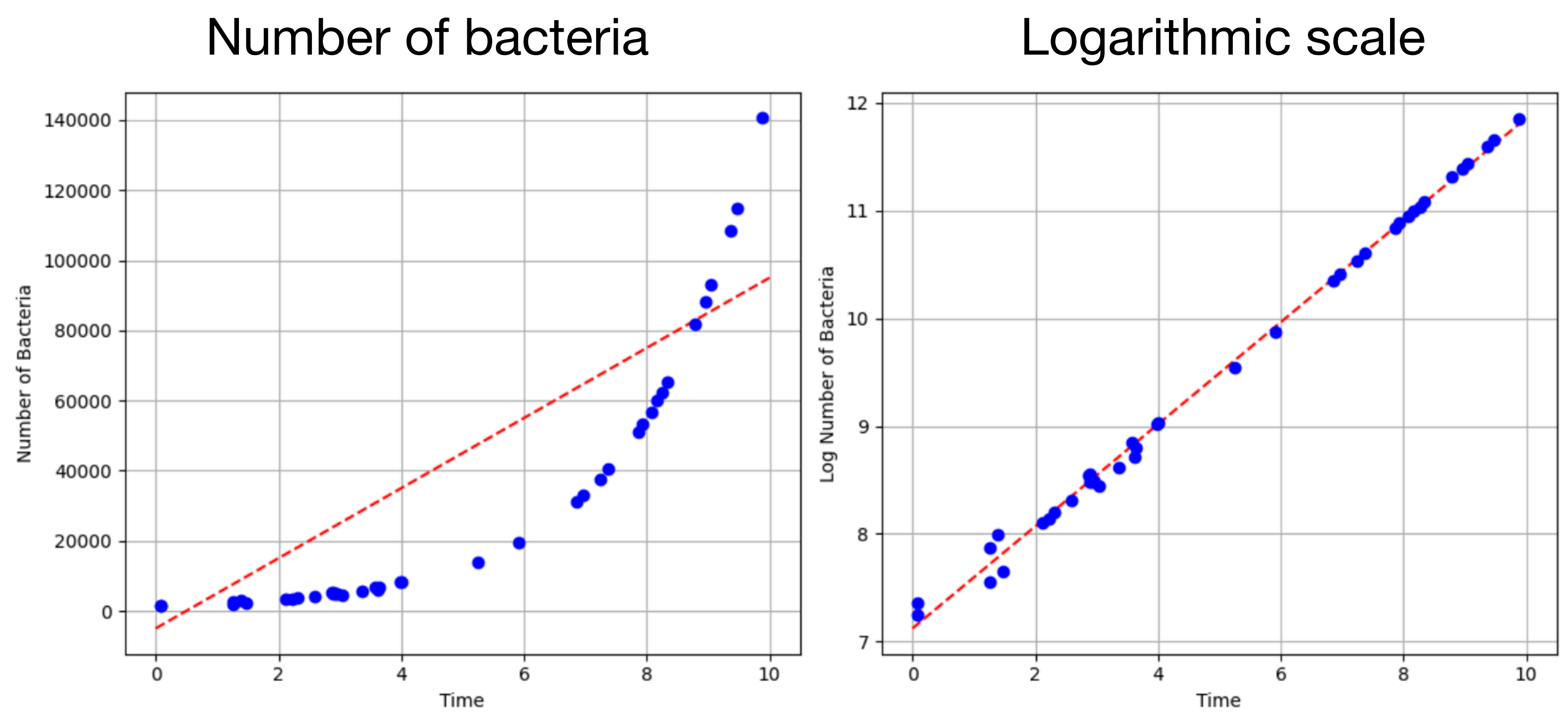
Learn parameters θ for $f(\cdot\,; \theta)$ to approximate f^* $\longrightarrow$ Find w_ρ, w_α, b such that $\boxed{f(\rho, \alpha; \theta) = H((\rho, \alpha) \cdot (w_\rho, w_\alpha) + b) = f^*(\rho, \alpha)}$

Can the perceptron solve sufficiently accurately this classification task?

- Cartesian coordinates: no, it is not a linearly separable problem and perceptrons only model linear decision boundaries
- Polar coordinates? yes, it is linearly separable. What perceptron solves this problem?

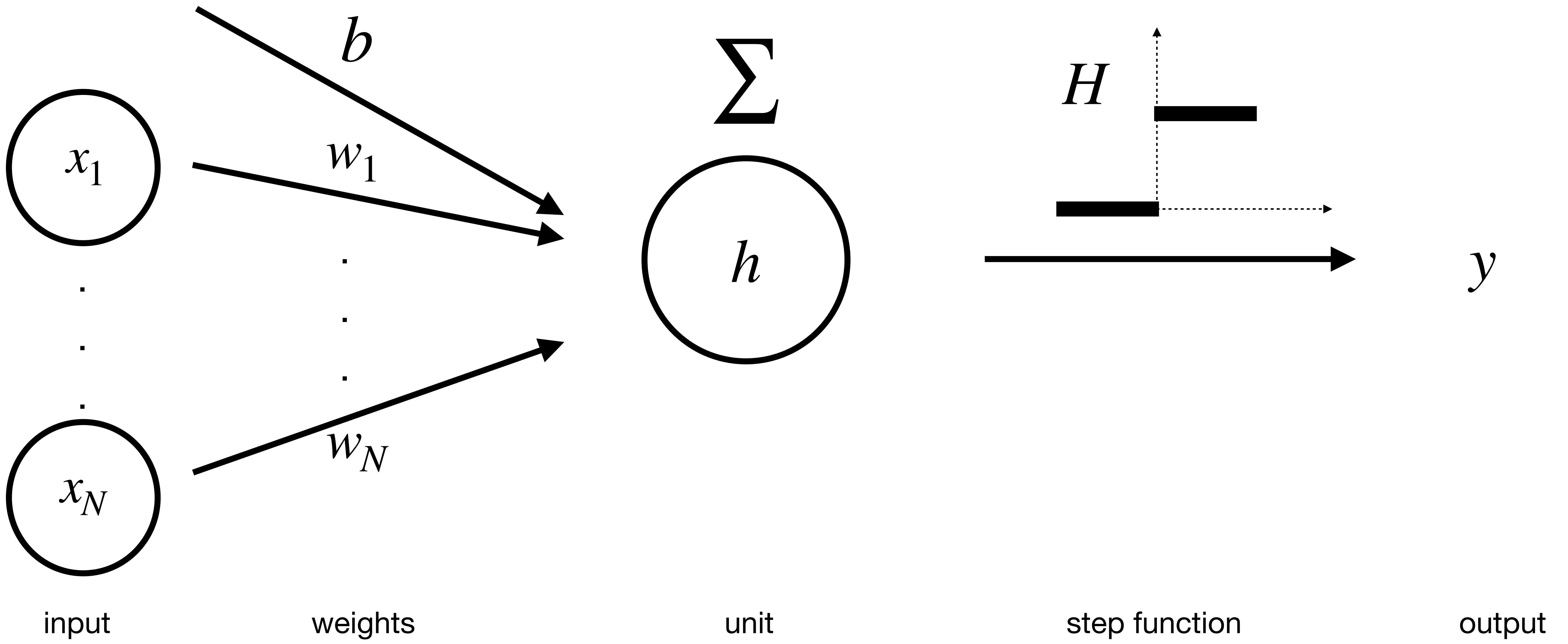
Perceptron for Regression





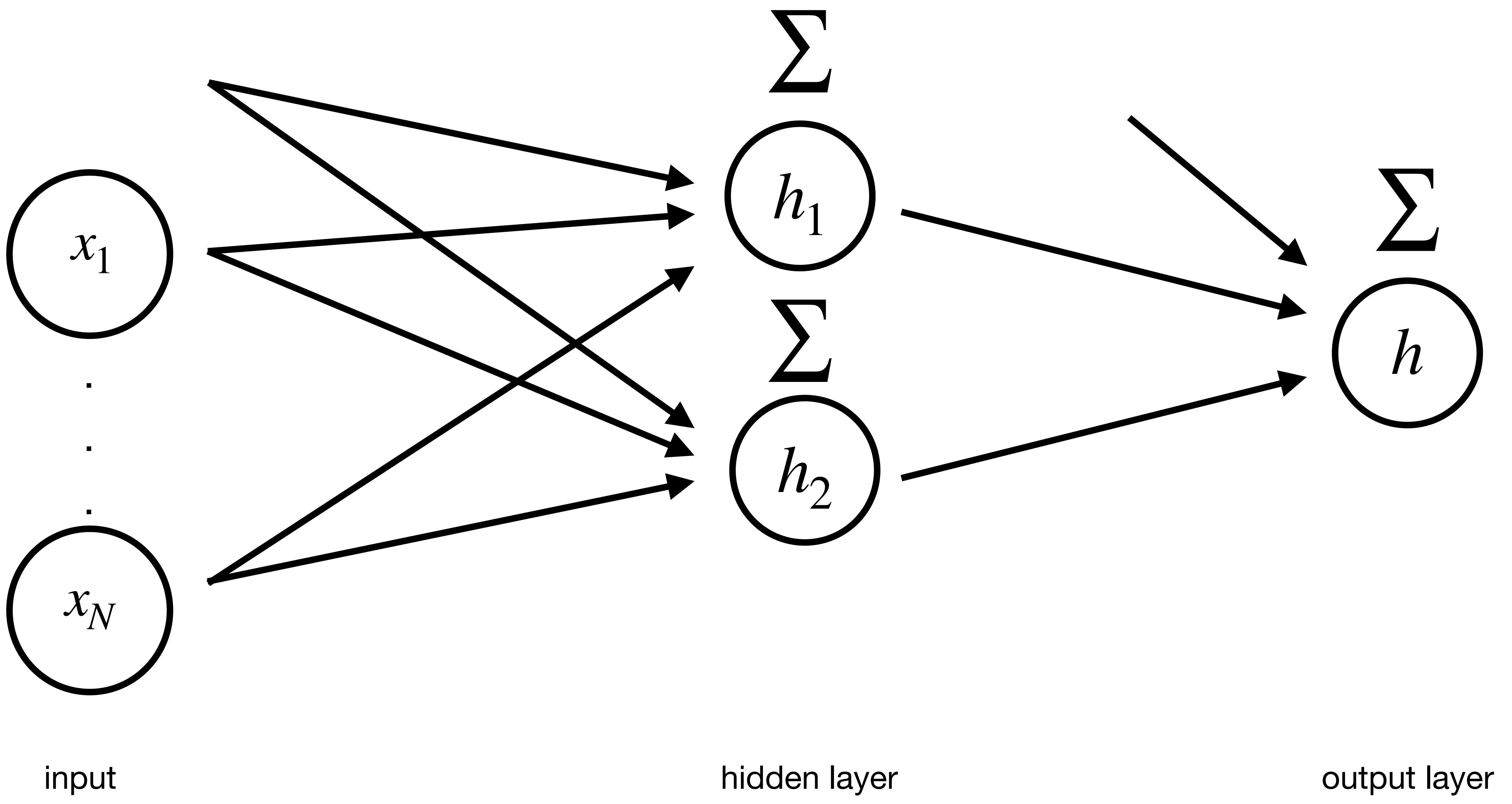
For regression problems it models input/output relation as a line.

Beyond Linear Perceptron



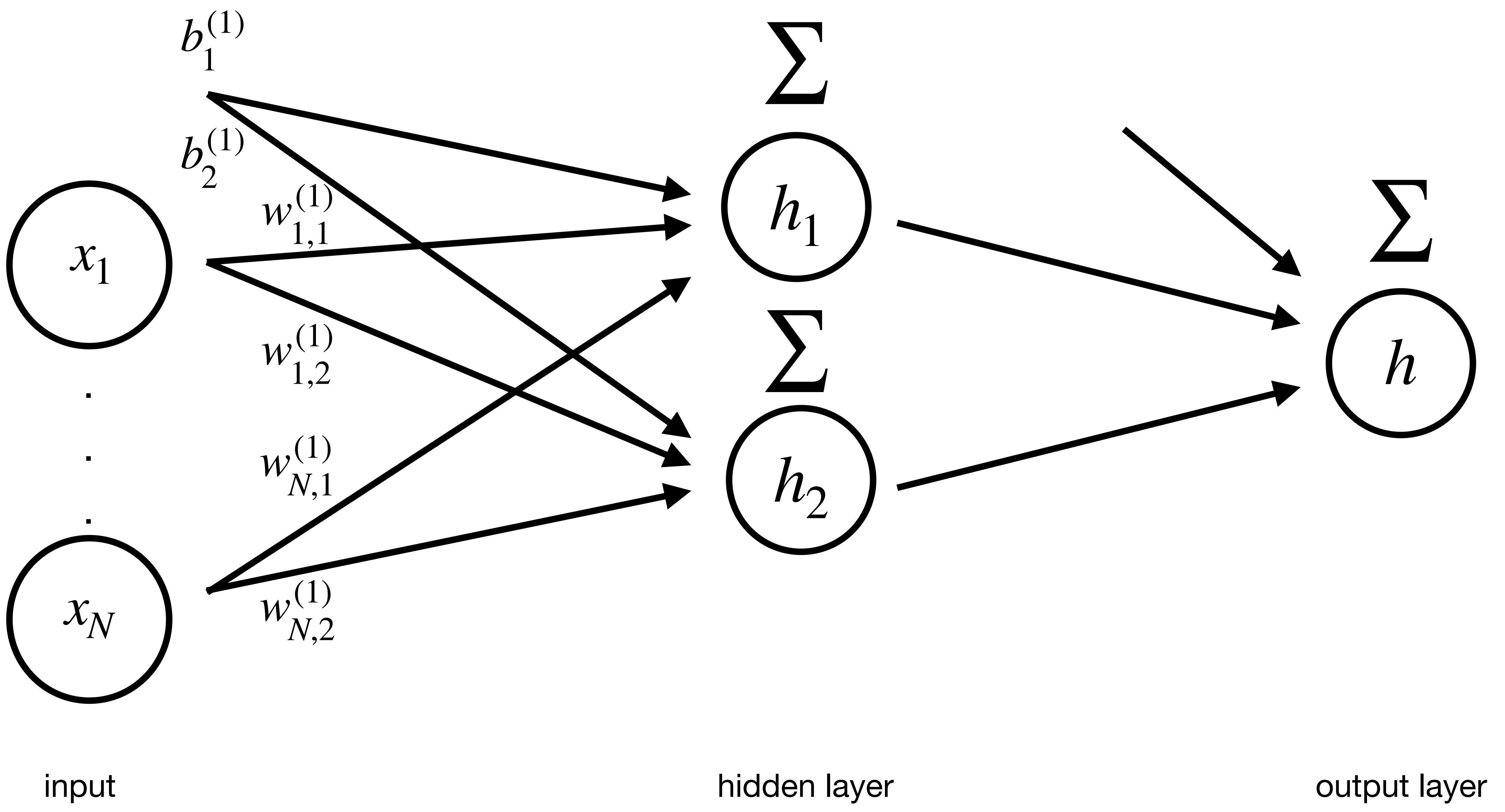
Extend linear models to represent a broader family of target functions f^*

Beyond Linear Perceptron



Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

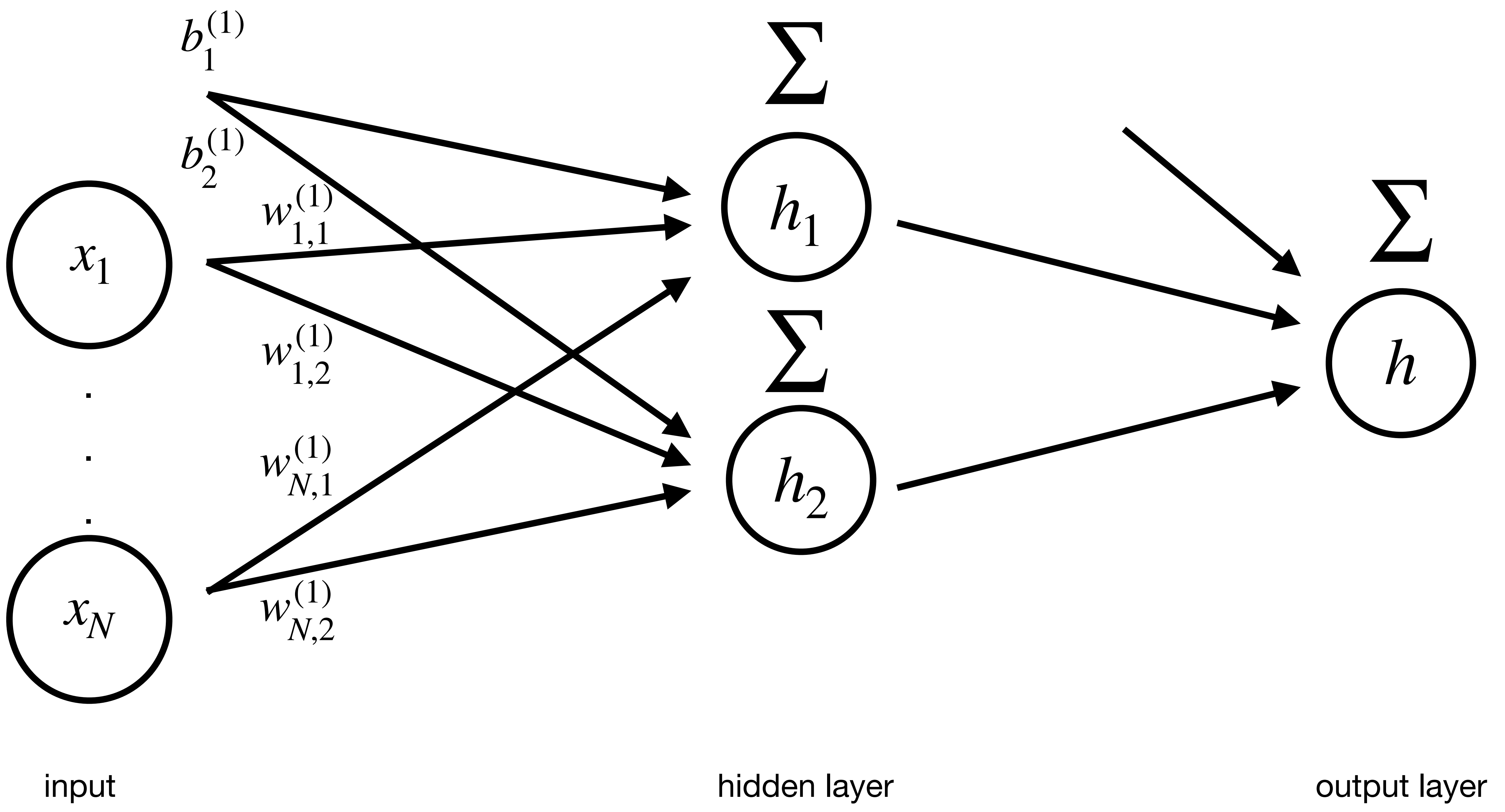
Beyond Linear Perceptron



Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ \cdots & \cdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix}$$

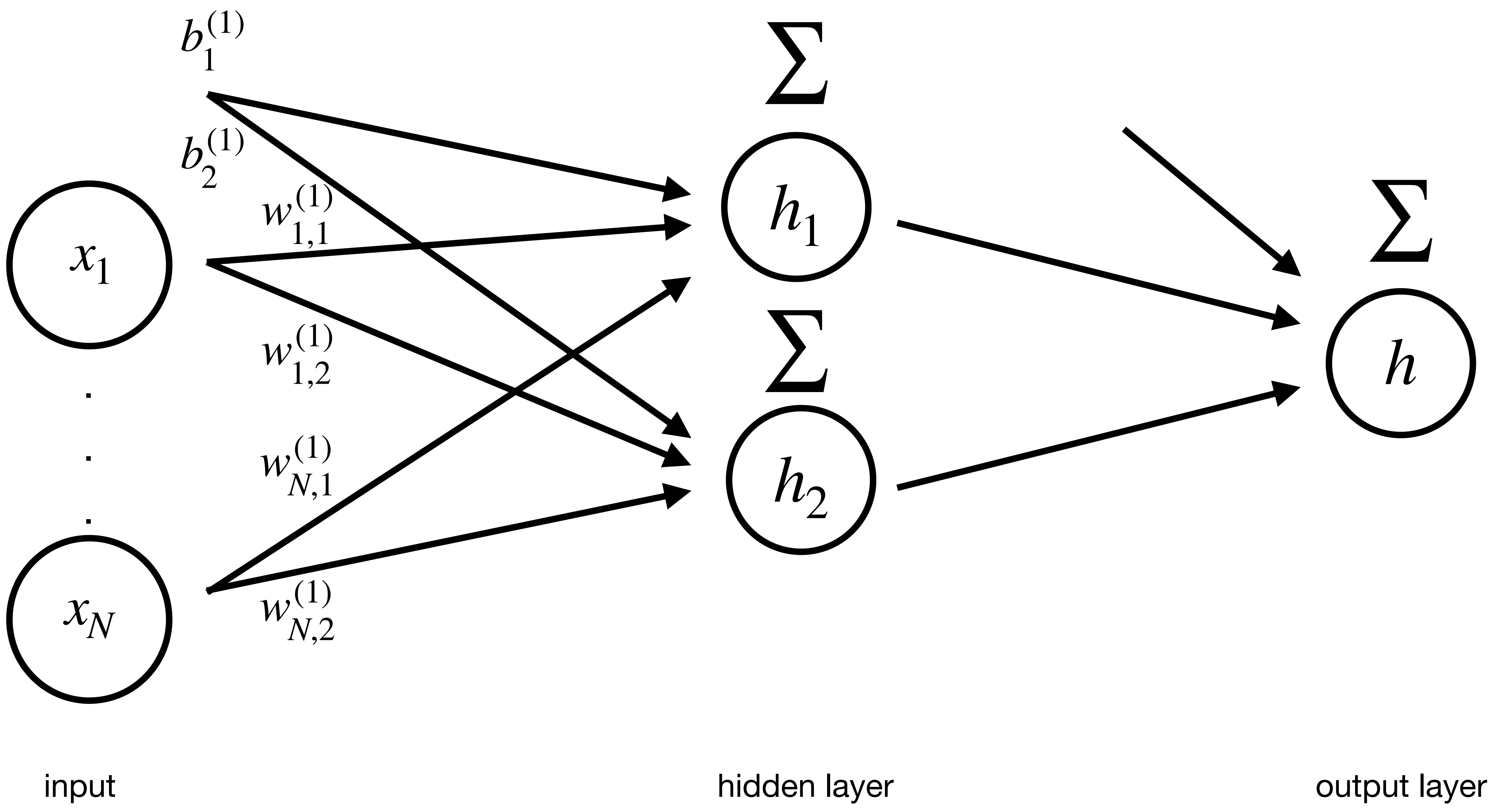
Beyond Linear Perceptron



Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ \cdots & \cdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix} \quad \longrightarrow \quad h^{(1)}(x) = x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)} \in \mathbb{R}^?$$

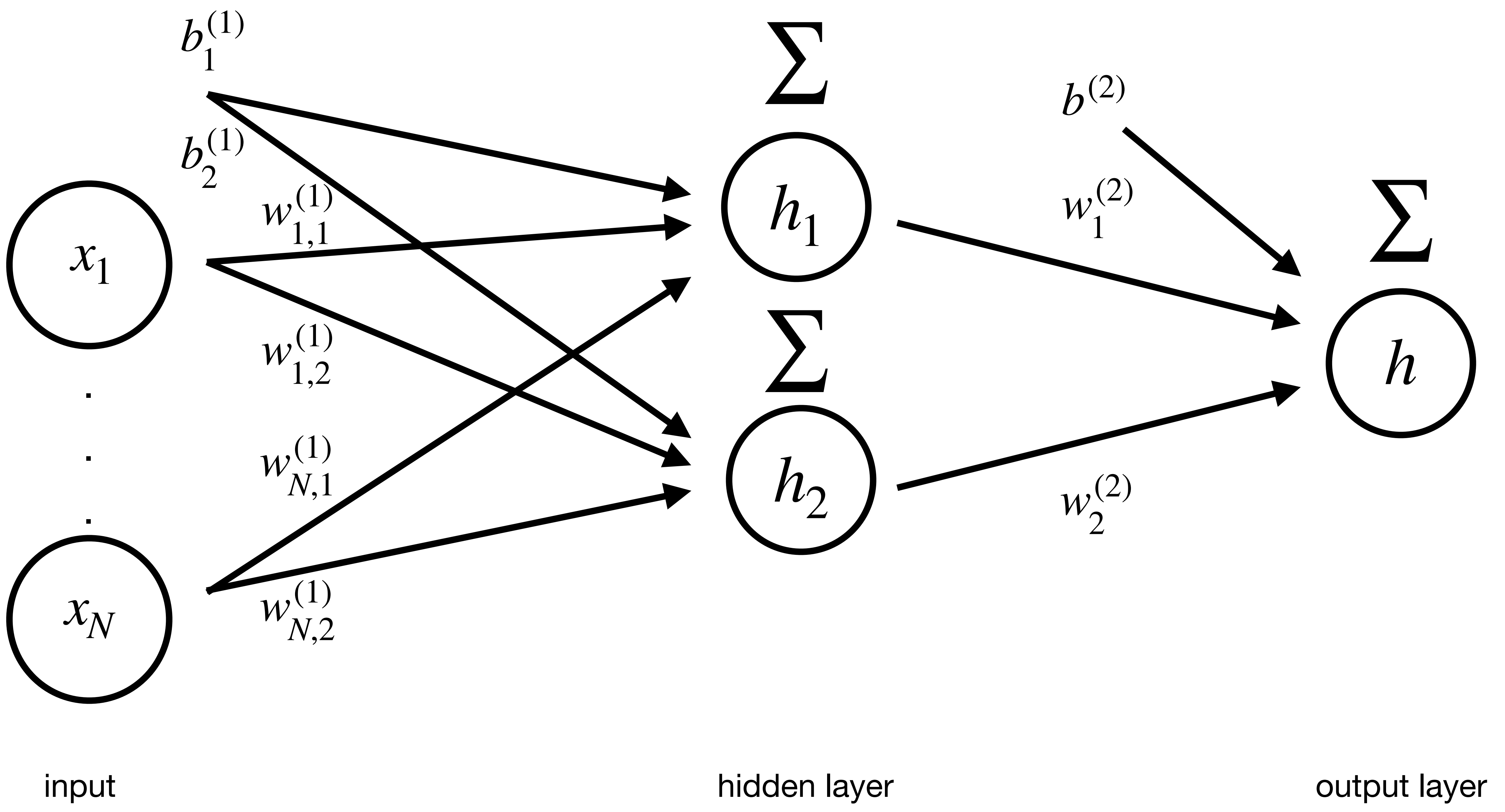
Beyond Linear Perceptron



Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ \cdots & \cdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix} \longrightarrow \quad h^{(1)}(x) = x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)} \in \mathbb{R}^2$$

Beyond Linear Perceptron

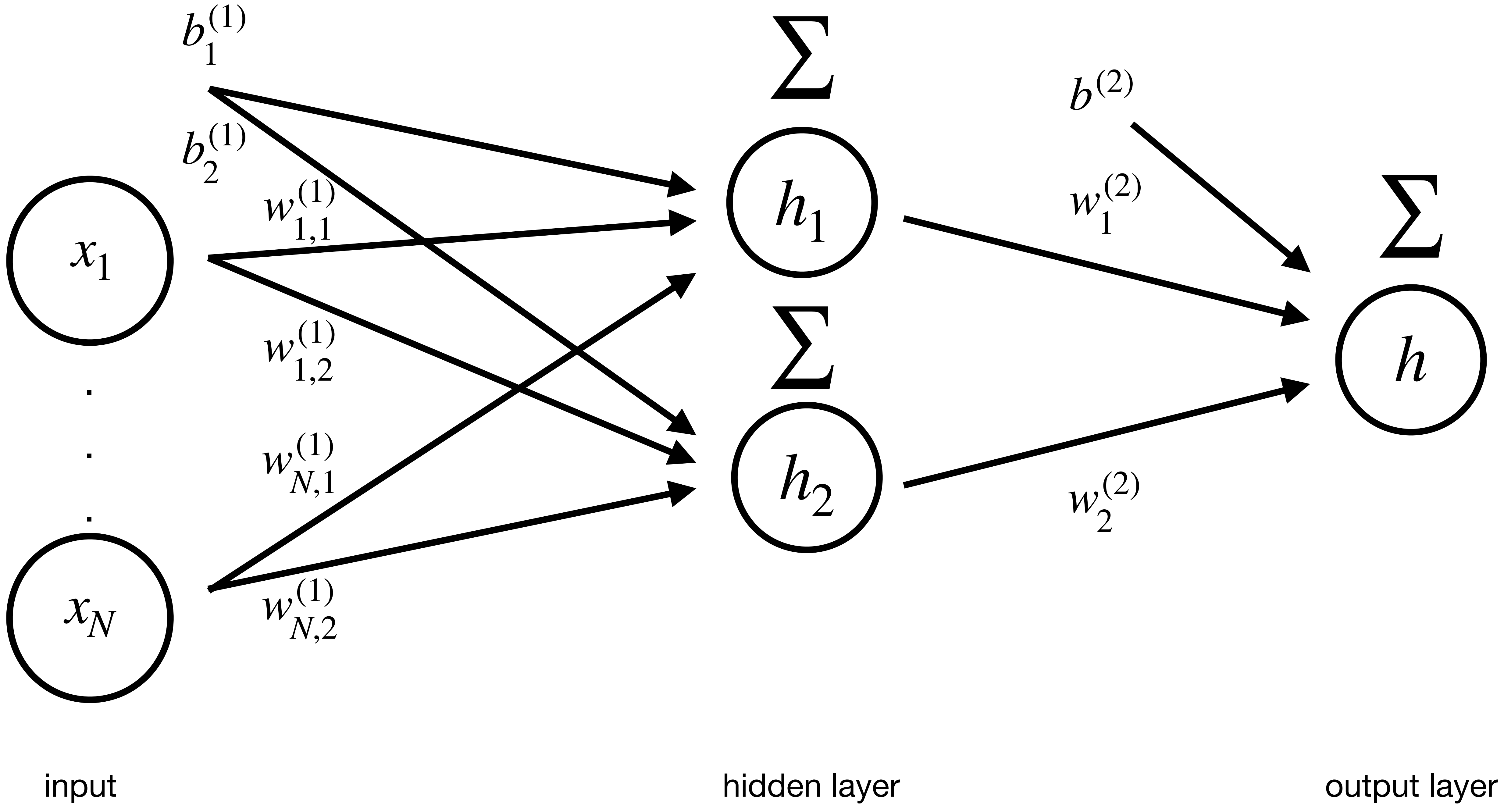


Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ \cdots & \cdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix} \longrightarrow h^{(1)}(x) = x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)} = (h_1, h_2) \in \mathbb{R}^2$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} w_1^{(2)} \\ w_2^{(2)} \end{pmatrix} \quad \mathbf{b}^{(2)} = \begin{pmatrix} b^{(2)} \end{pmatrix} \longrightarrow h^{(2)}((h_1, h_2)) = (h_1, h_2) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)} = h \in \mathbb{R}^1$$

Beyond Linear Perceptron



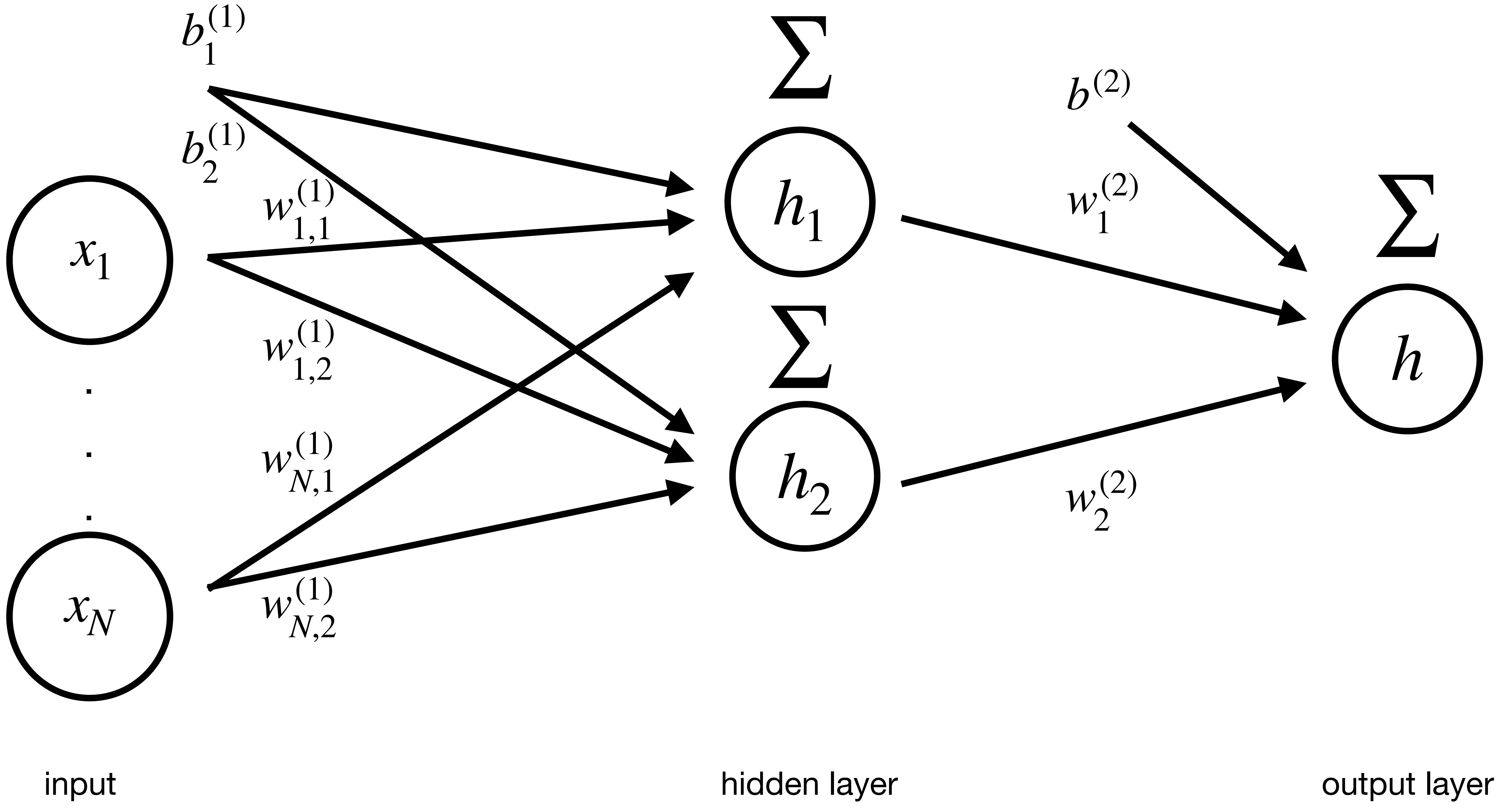
Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ \cdots & \cdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix} \longrightarrow h^{(1)}(x) = x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)} = (h_1, h_2) \in \mathbb{R}^2$$

$$f(x;\theta) = ?$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} w_1^{(2)} \\ w_2^{(2)} \end{pmatrix} \quad \mathbf{b}^{(2)} = \begin{pmatrix} b^{(2)} \end{pmatrix} \longrightarrow h^{(2)}((h_1, h_2)) = (h_1, h_2) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)} = h \in \mathbb{R}^1$$

Beyond Linear Perceptron



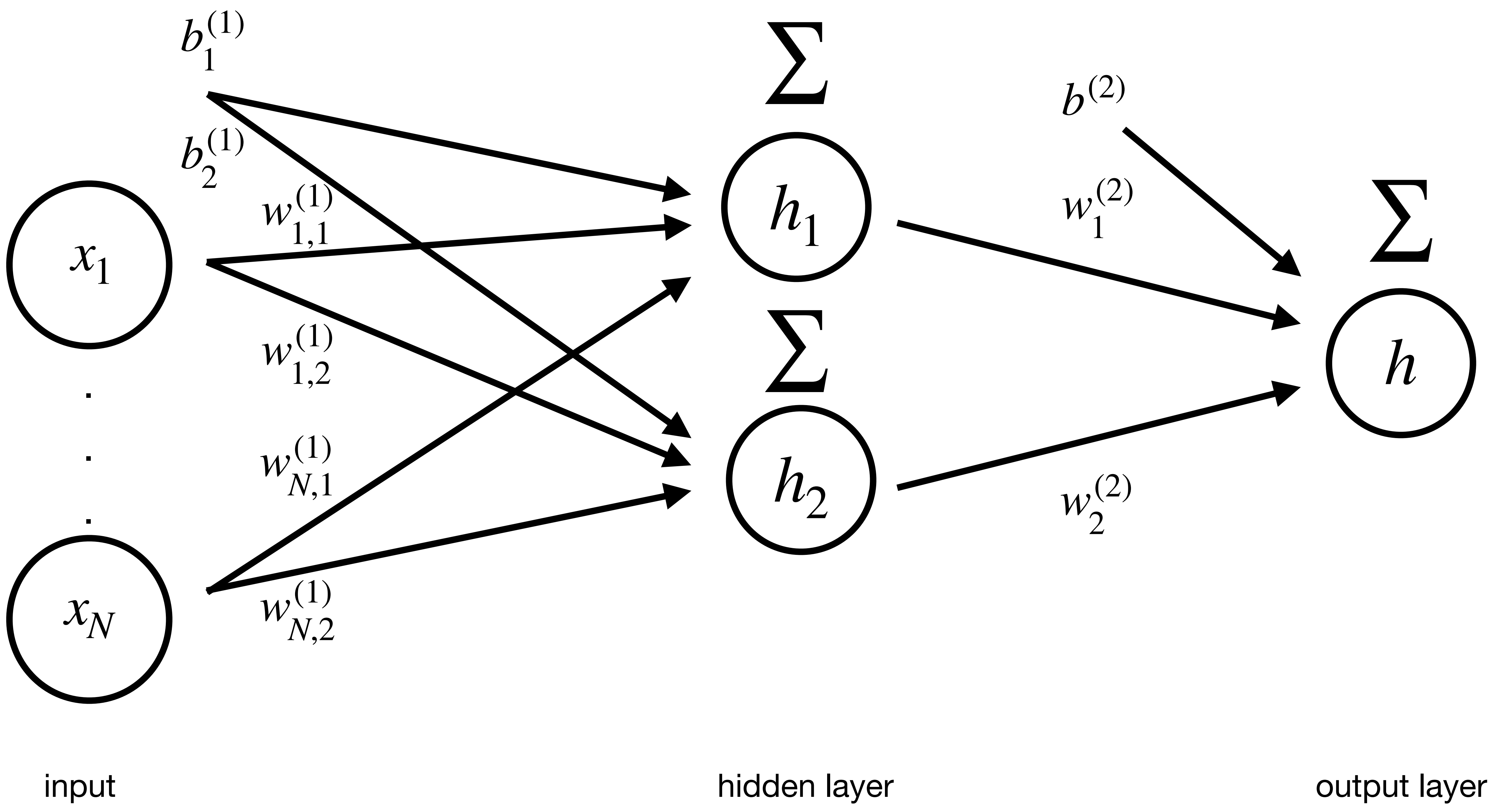
Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ \cdots & \cdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix} \longrightarrow h^{(1)}(x) = x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)} = (h_1, h_2) \in \mathbb{R}^2$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} w_1^{(2)} \\ w_2^{(2)} \end{pmatrix} \quad \mathbf{b}^{(2)} = \begin{pmatrix} b^{(2)} \end{pmatrix} \longrightarrow h^{(2)}((h_1, h_2)) = (h_1, h_2) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)} = h \in \mathbb{R}^1$$

$$f(x;\theta) = \overbrace{\big(\overbrace{x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)}}^{h^{(1)}}\big) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)}}^{h^{(2)}}$$

Beyond Linear Perceptron



Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

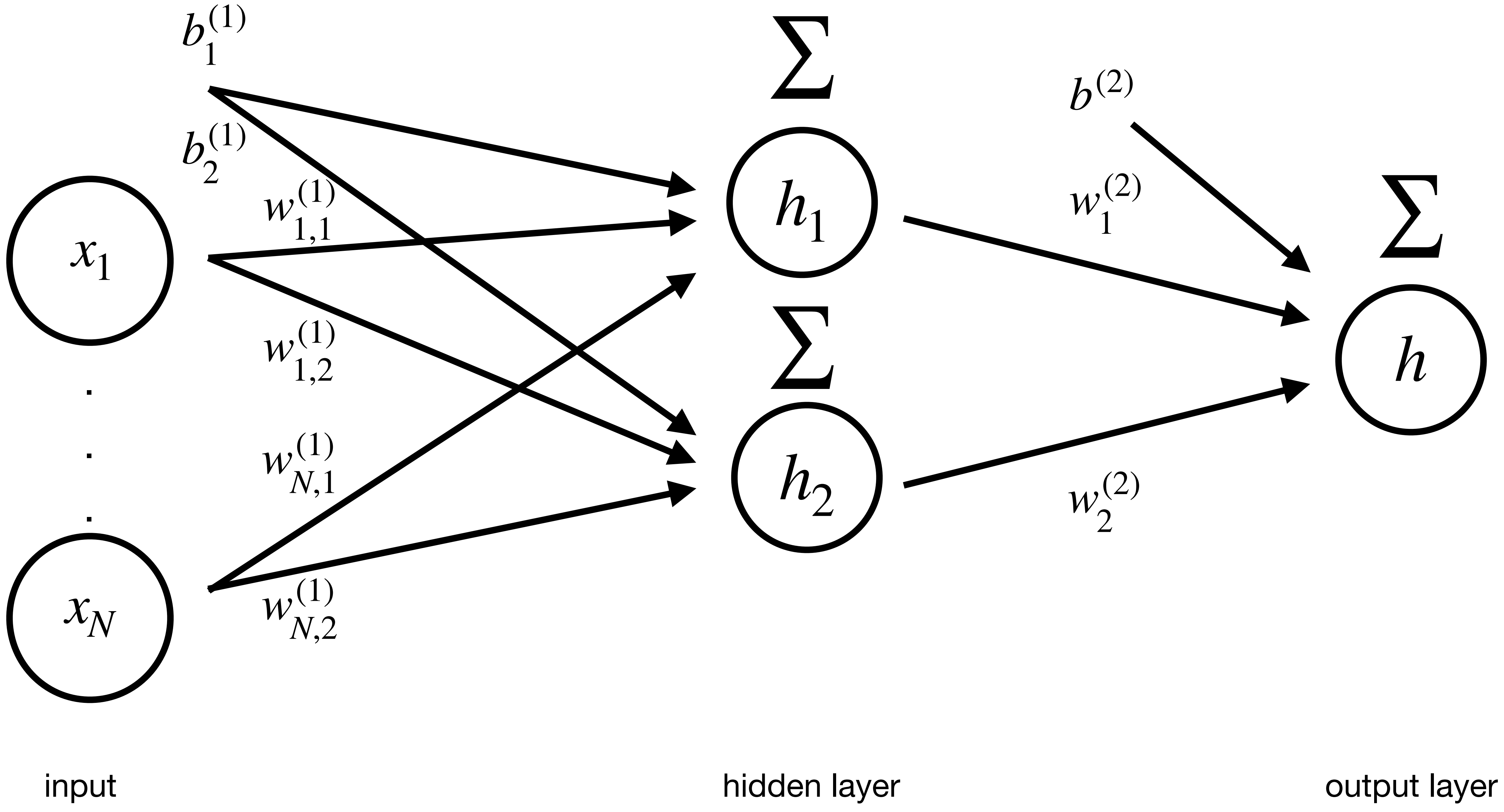
$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ \cdots & \cdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix} \longrightarrow h^{(1)}(x) = x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)} = (h_1, h_2) \in \mathbb{R}^2$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} w_1^{(2)} \\ w_2^{(2)} \end{pmatrix} \quad \mathbf{b}^{(2)} = \begin{pmatrix} b^{(2)} \end{pmatrix} \longrightarrow h^{(2)}((h_1, h_2)) = (h_1, h_2) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)} = h \in \mathbb{R}^1$$

$$f(x;\theta) = \overbrace{\overbrace{\left(x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right)}^{h^{(1)}} \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)}}^{h^{(2)}}$$

What is the difference with the linear perceptron?

Beyond Linear Perceptron



Network as composition of hidden layers: $f(x;\theta) = h^{(2)}(h^{(1)}(x))$

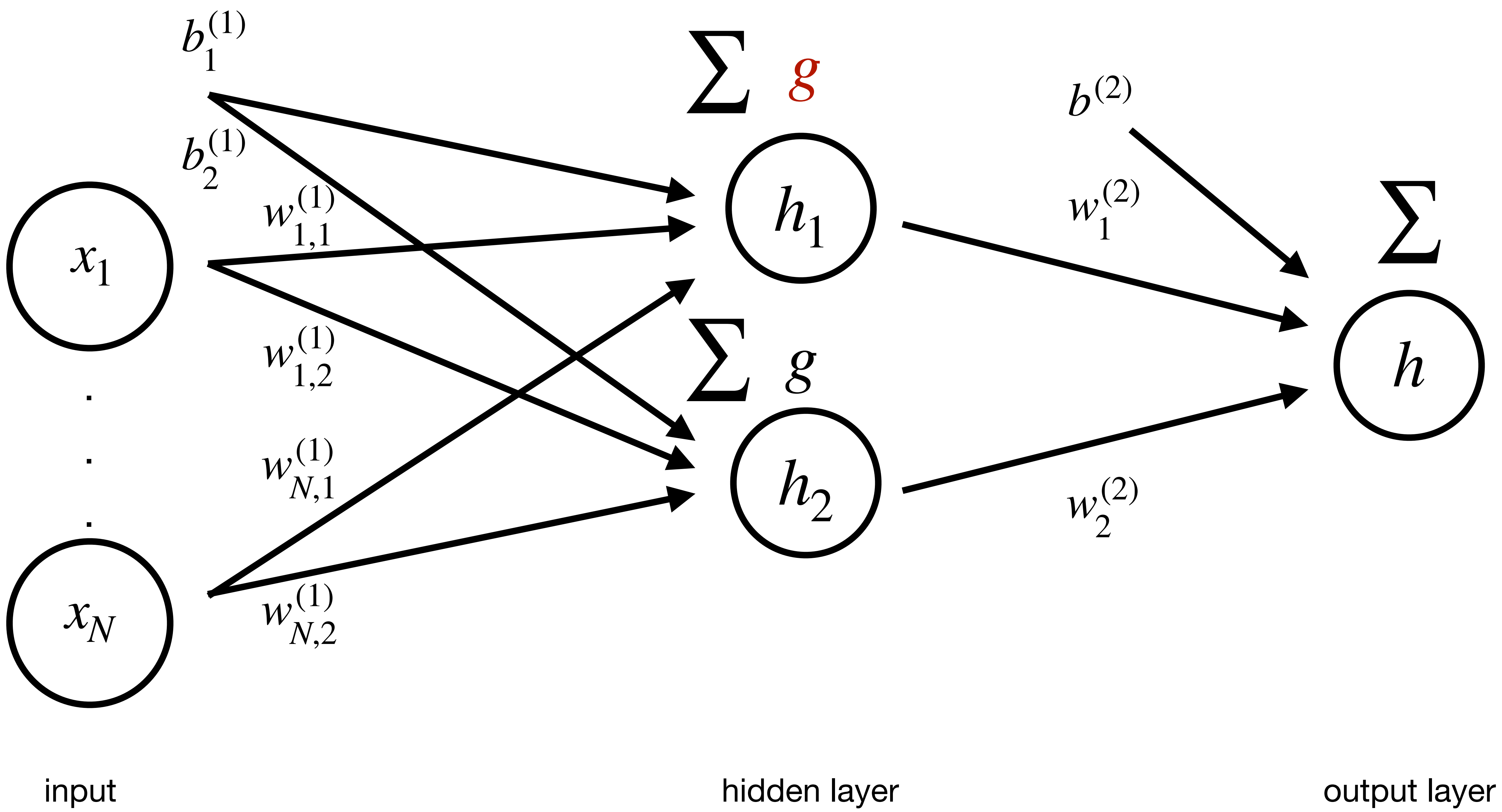
$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ \cdots & \cdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} \end{pmatrix} \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} \end{pmatrix} \longrightarrow h^{(1)}(x) = x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)} = (h_1, h_2) \in \mathbb{R}^2$$

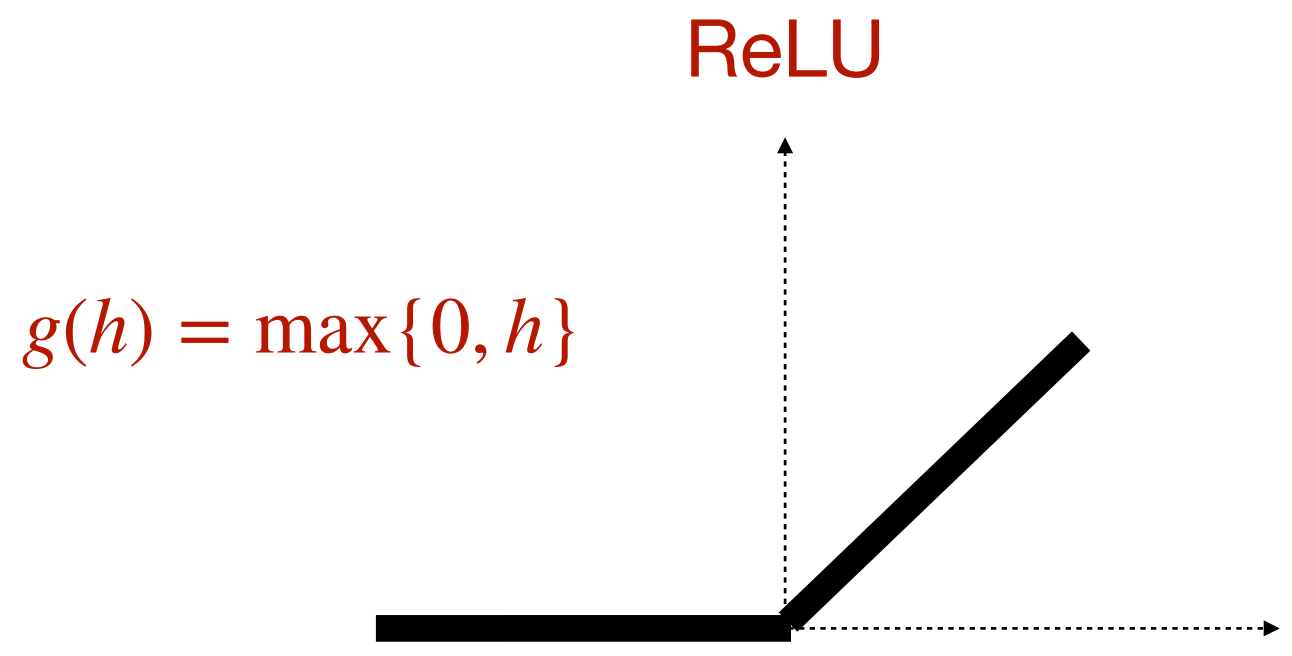
$$\mathbf{W}^{(2)} = \begin{pmatrix} w_1^{(2)} \\ w_2^{(2)} \end{pmatrix} \quad \mathbf{b}^{(2)} = \begin{pmatrix} b^{(2)} \end{pmatrix} \longrightarrow h^{(2)}((h_1, h_2)) = (h_1, h_2) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)} = h \in \mathbb{R}^1$$

$$\begin{aligned} f(x;\theta) &= \left(x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)} \\ &= x \cdot \underbrace{\mathbf{W}^{(1)}\mathbf{W}^{(2)}}_{\mathbf{W}} + \underbrace{\mathbf{b}^{(1)} \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)}}_{\mathbf{b}} \end{aligned}$$

None! It is still a linear perceptron

Feedforward Neural Network



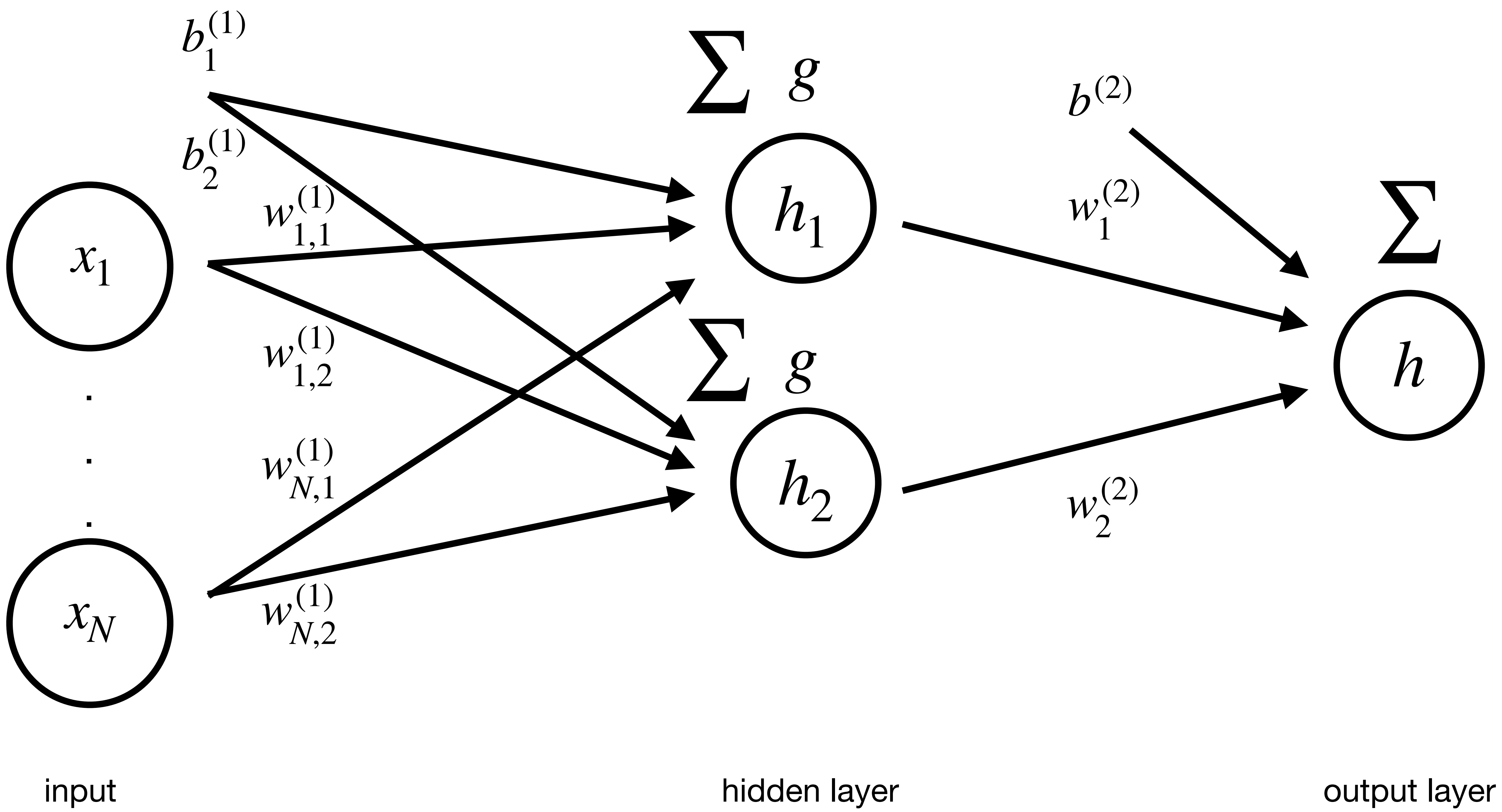


Apply non-linear **activation functions** g to each unit output

$$f(x;\theta) = g\left(x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)*}$$

g is applied component-wise

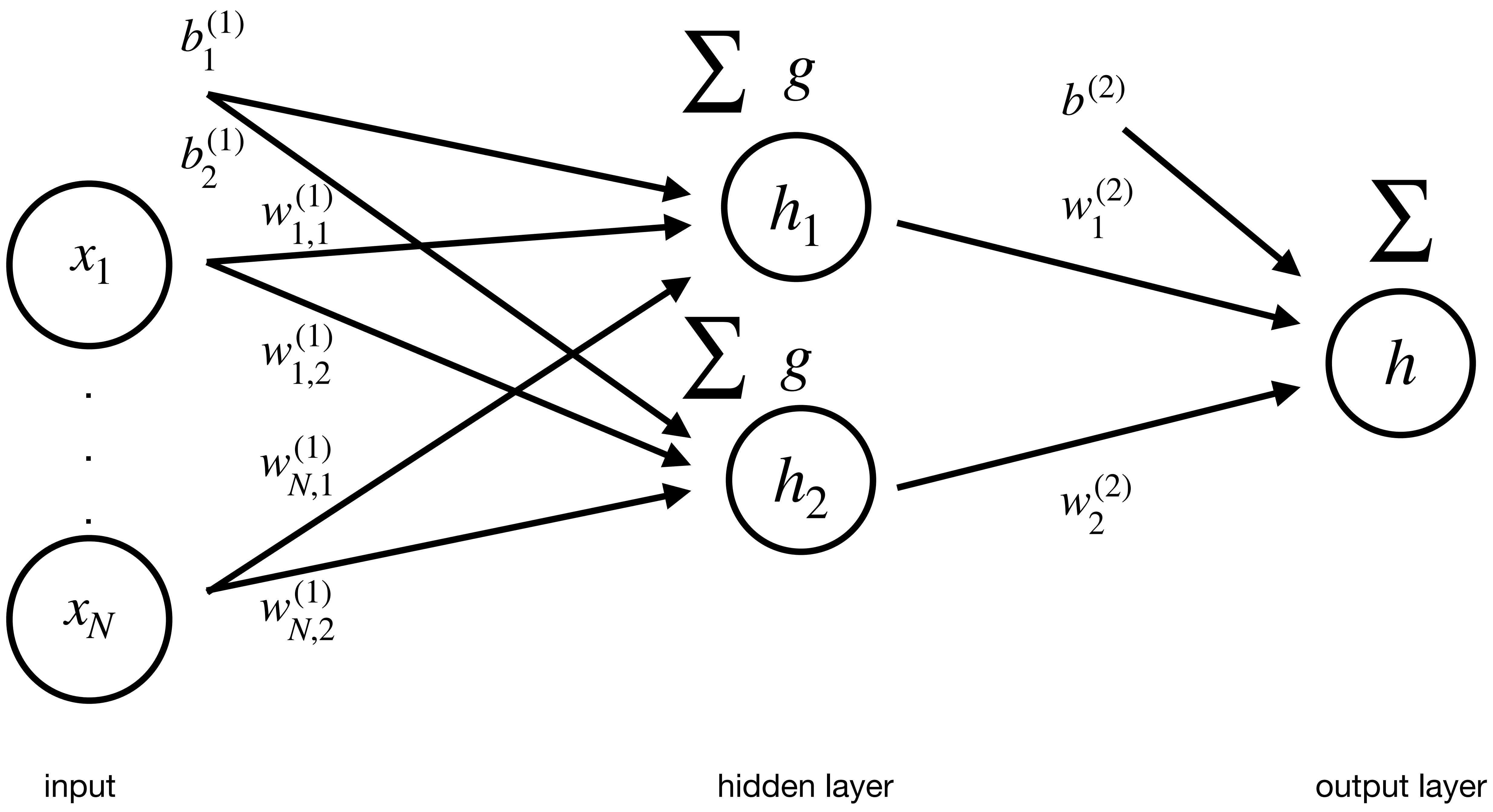
Feedforward Neural Network



Notation and terminology:

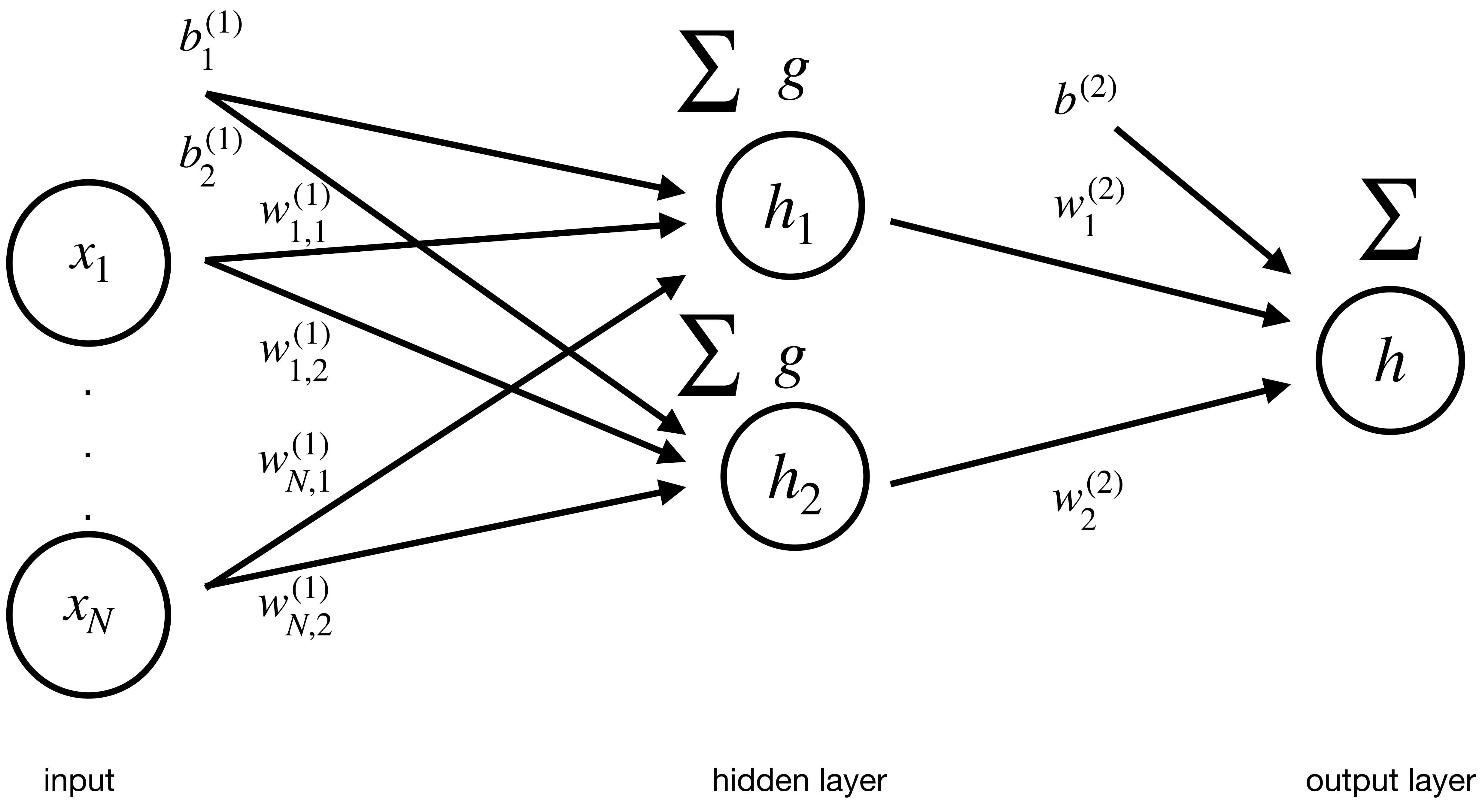
- Depth K [#hidden + output layer] = 2
- $h^{(i)}$ layer i, $h^{(K)}$ output layer
- Layer width $D_1, \ldots, D_K$ [#of hidden units per layer] = 2,1

Feedforward Neural Network



How many learnable parameters or what is the network size?

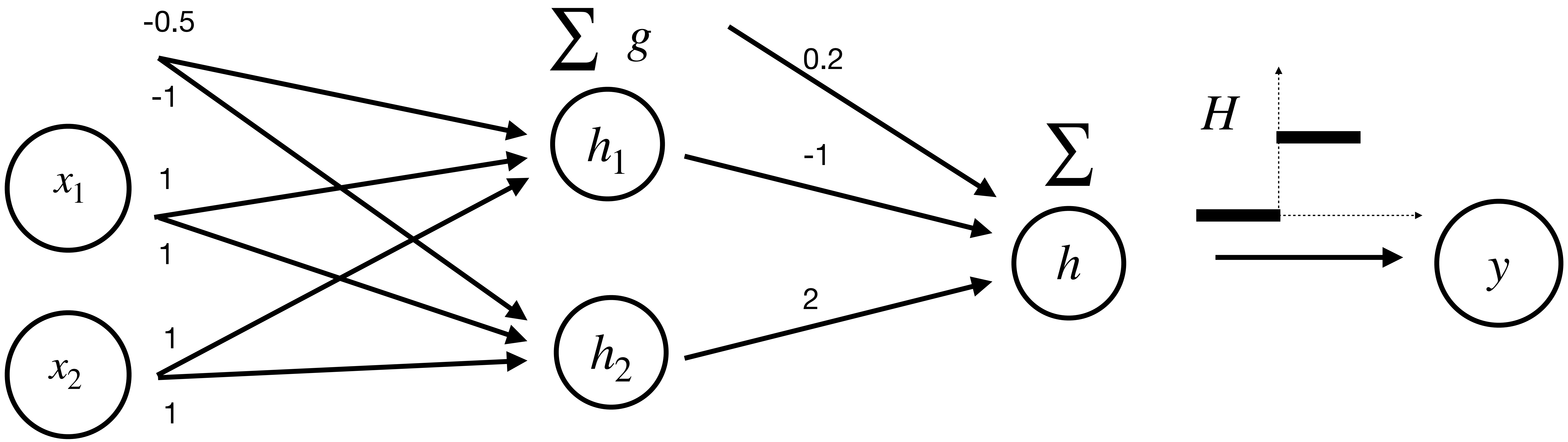
Feedforward Neural Network



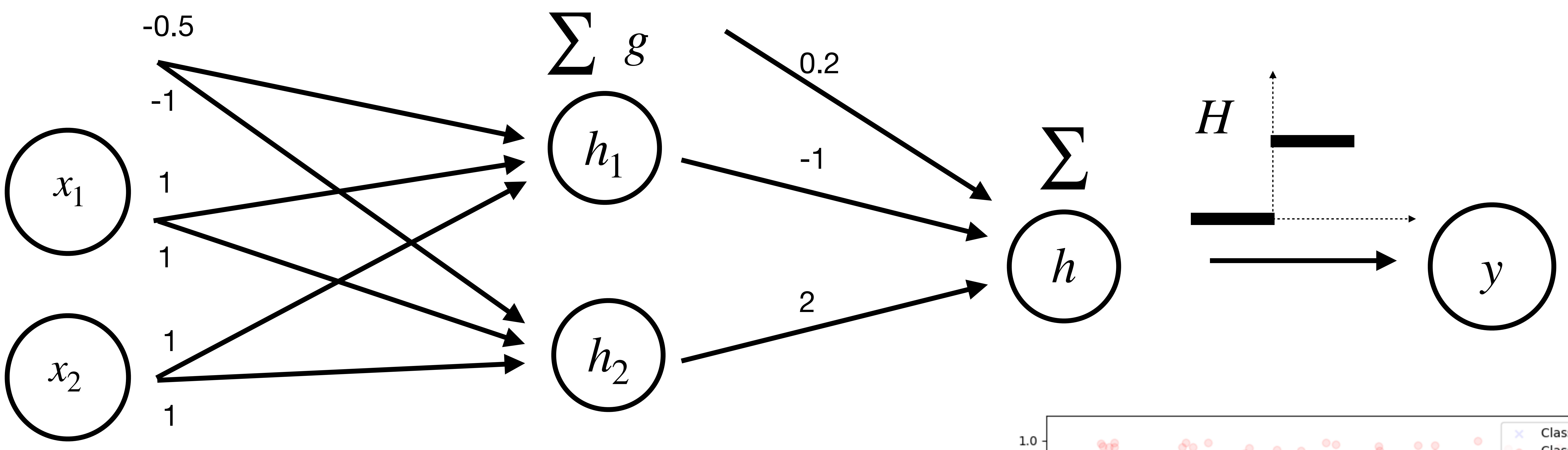
How many learnable parameters or what is the network size? $(2N+2)+(2+1)=2N+5$

*Can you find a general expression for $N, K, D_1, \ldots, D_K$?

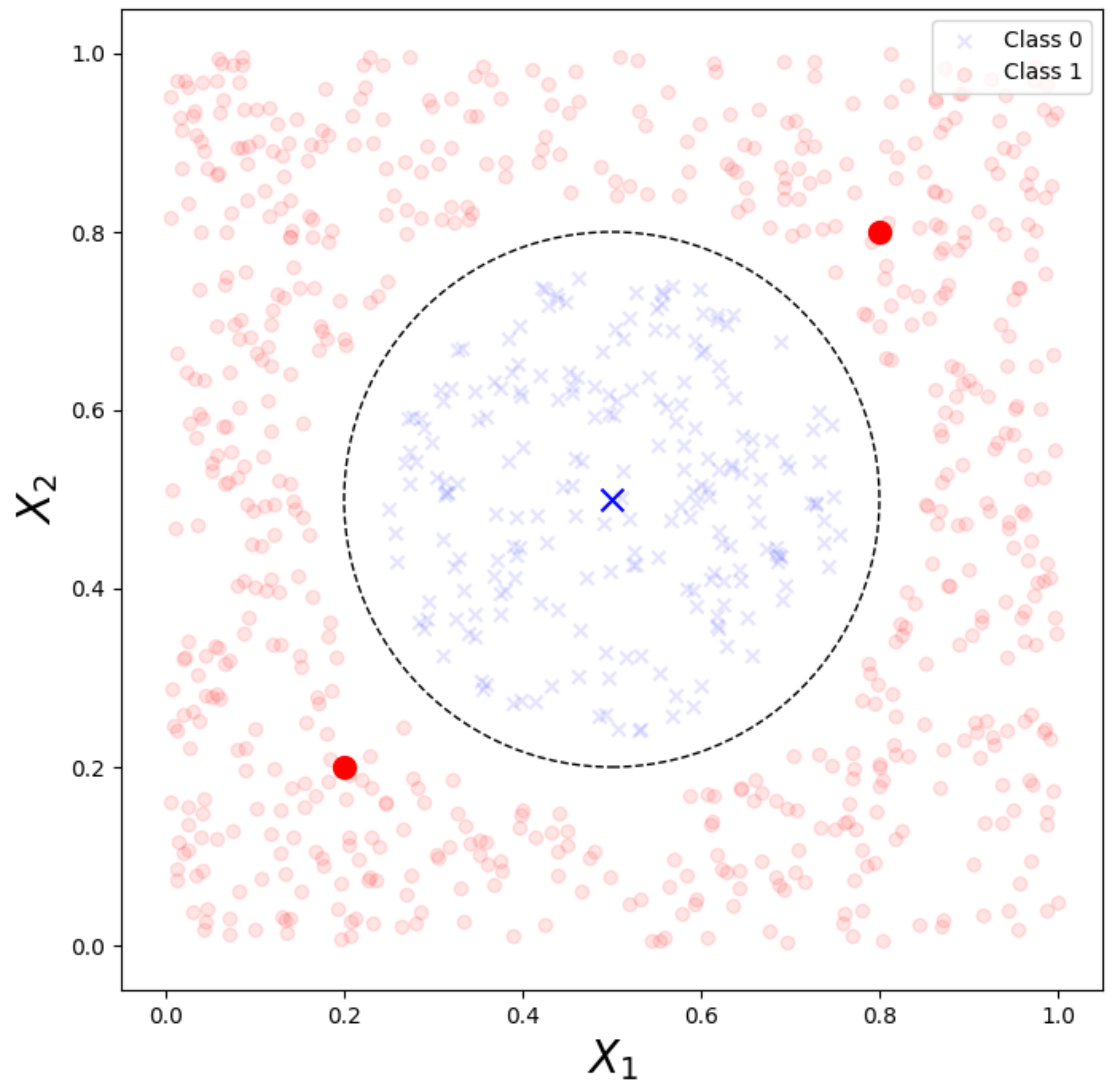
Forward Pass



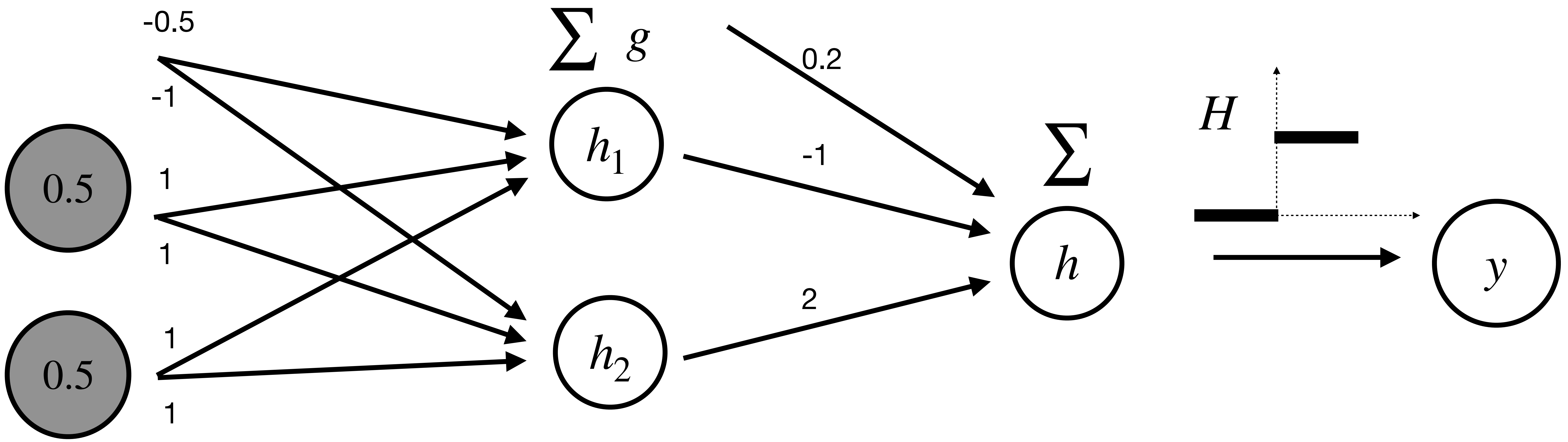
Forward Pass: Running Example



Input	Output	Class
(0.5, 0.5)		0
(0.2, 0.2)		1
(0.8, 0.8)		1



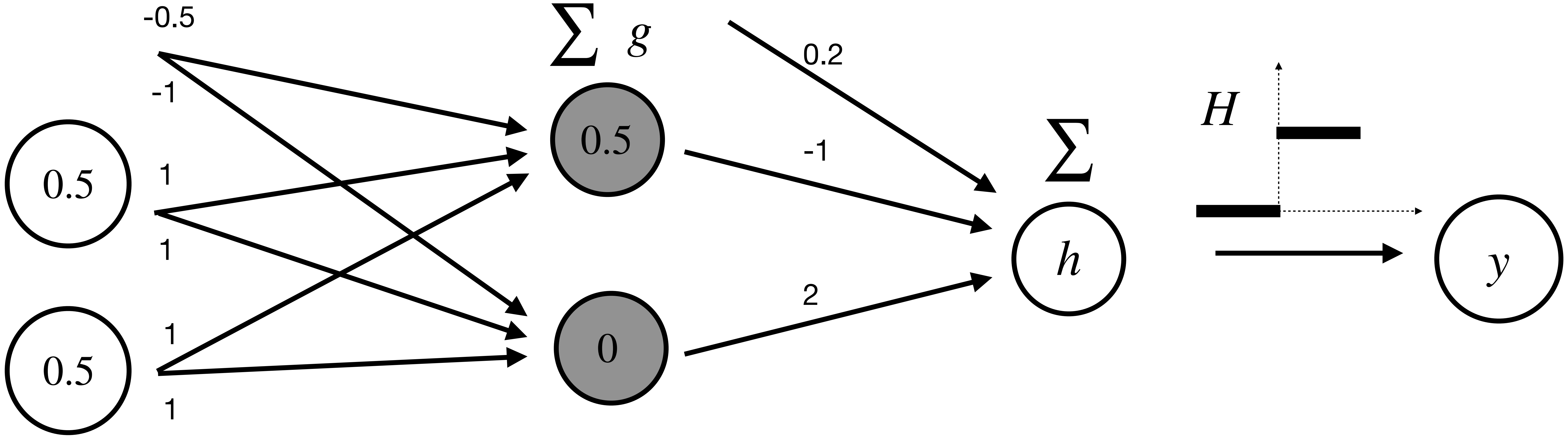
Forward Pass: Running Example



Input	Output
(0.5, 0.5)	
(0.2, 0.2)	
(0.8, 0.8)	

Class
0
1
1

Forward Pass: Running Example

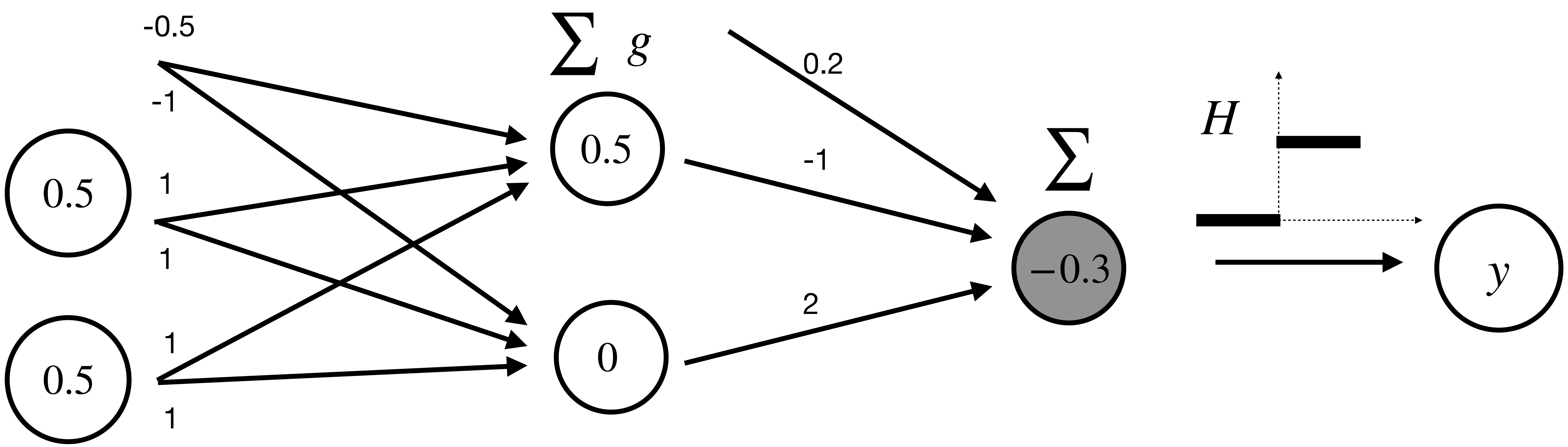


Input	Output
(0.5, 0.5)	
(0.2, 0.2)	
(0.8, 0.8)	

Class
0
1
1

$$h_1 = g(0.5 + 0.5 - 0.5) = 0.5$$
$$h_2 = g(0.5 + 0.5 - 1) = 0$$

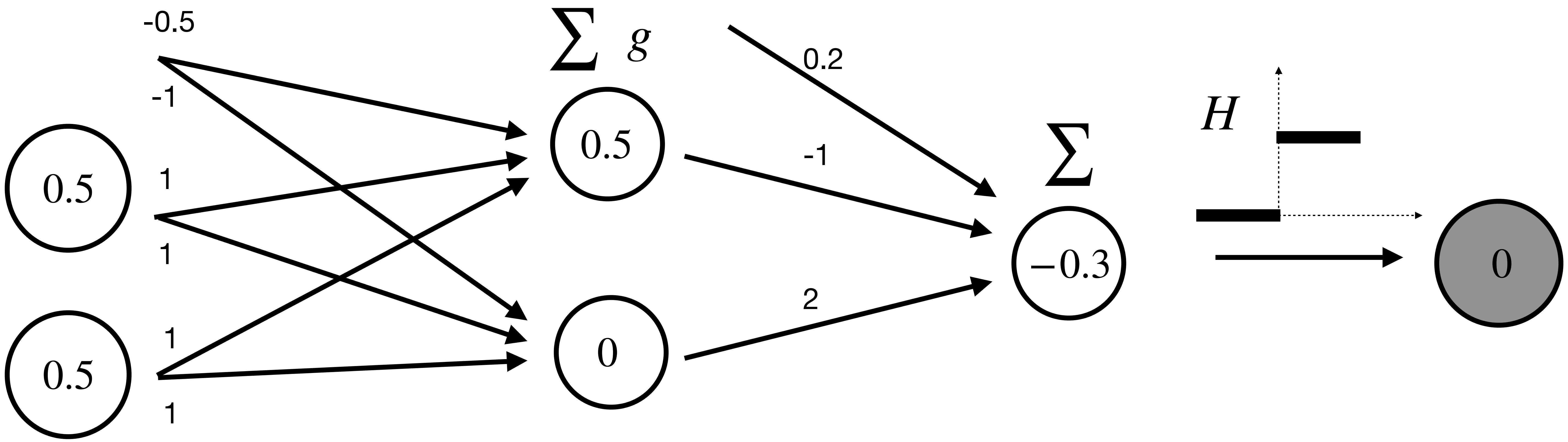
Forward Pass: Running Example



Input	Output	Class
(0.5, 0.5)		0
(0.2, 0.2)		1
(0.8, 0.8)		1

$$h_1 = g(0.5 + 0.5 - 0.5) = 0.5$$
$$h_2 = g(0.5 + 0.5 - 1) = 0$$
$$h = -0.5 + 2 \times 0 + 0.2 = -0.3$$

Forward Pass: Running Example

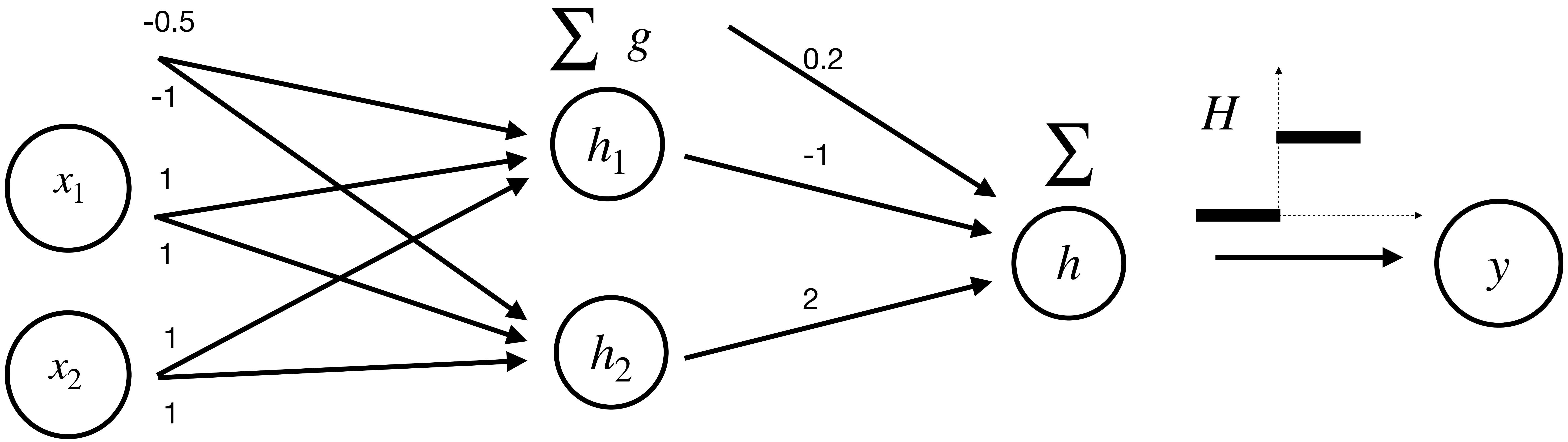


Input	Output
(0.5, 0.5)	**0**
(0.2, 0.2)	
(0.8, 0.8)	

Class
0
1
1

$$h_1 = g(0.5 + 0.5 - 0.5) = 0.5$$
$$h_2 = g(0.5 + 0.5 - 1) = 0$$
$$h = -0.5 + 2 \times 0 + 0.2 = -0.3$$
$$\hat{y} = 0$$

Forward Pass: Running Example

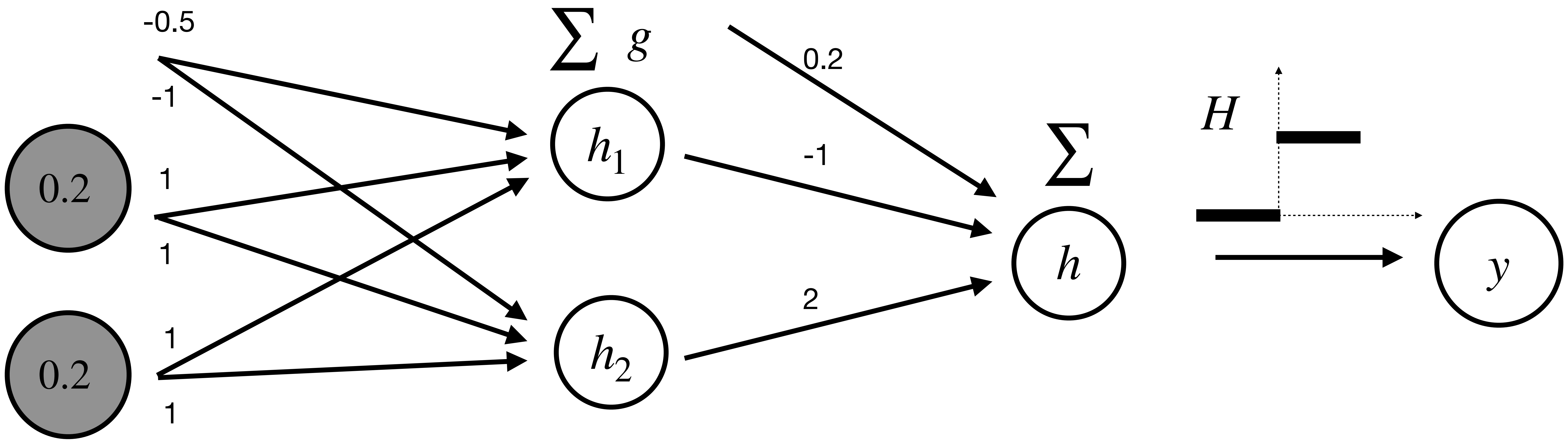


Input	Output
(0.5, 0.5)	0
(0.2, 0.2)	**?**
(0.8, 0.8)	**?**

Class
0
1
1

Take 2 minutes to compute the forward pass by yourself

Forward Pass: Running Example

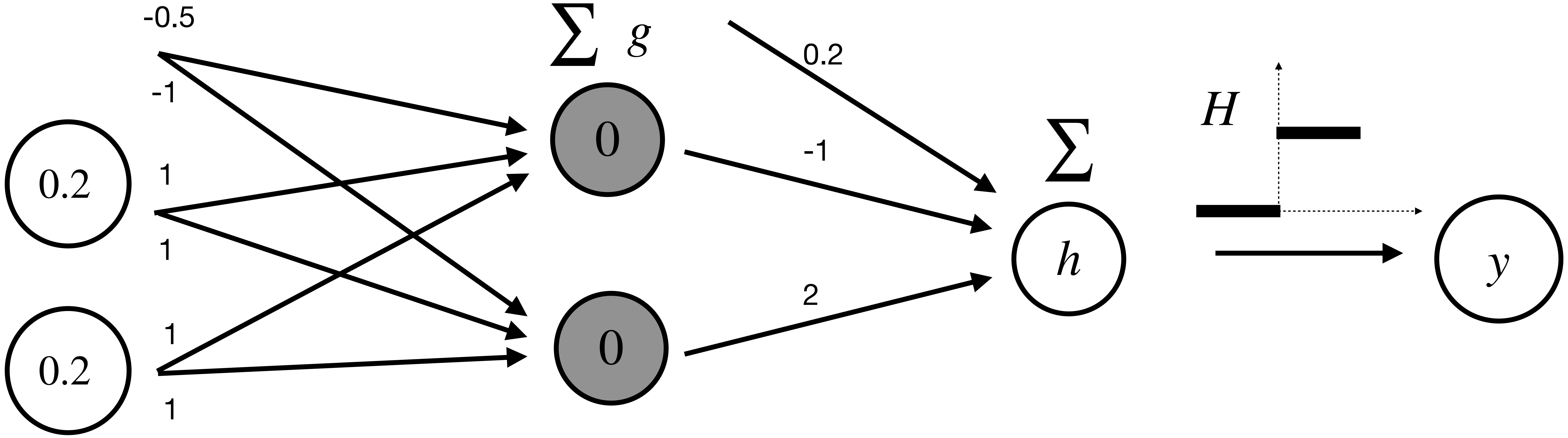


Input	Output
(0.5, 0.5)	0
(0.2, 0.2)	**?**
(0.8, 0.8)	?

Class
0
1
1

$$h = -g(0.2 + 0.2 - 0.5) + 2\,g(0.2 + 0.2 - 1) + 0.2$$

Forward Pass: Running Example

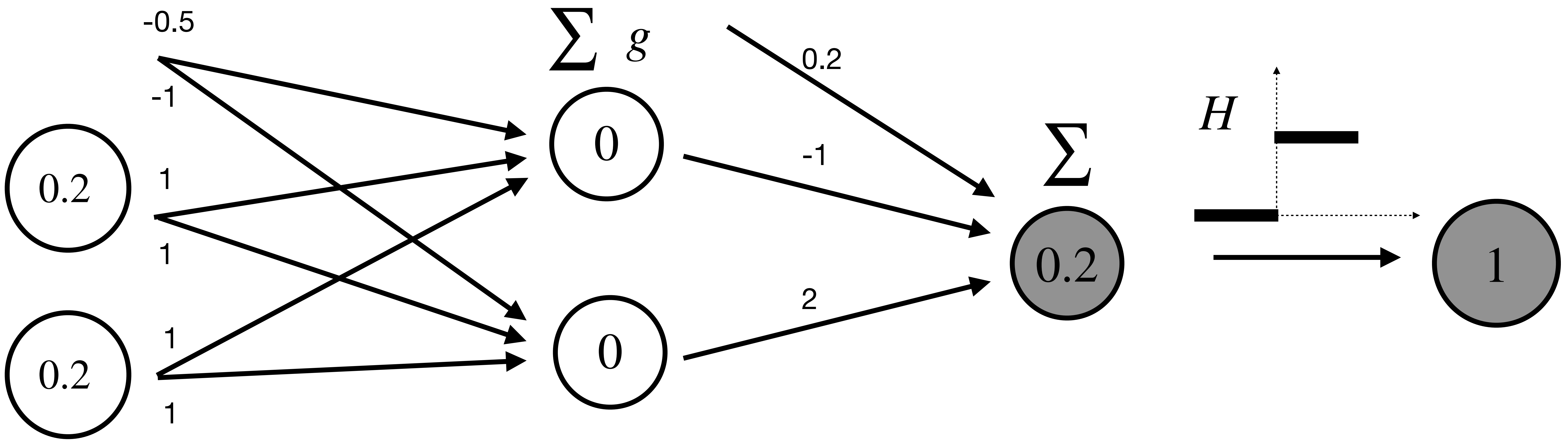


Input	Output
(0.5, 0.5)	0
(0.2, 0.2)	**?**
(0.8, 0.8)	?

Class
0
1
1

$$h = -\overbrace{g(0.2 + 0.2 - 0.5)}^{<0} + 2\,\overbrace{g(0.2 + 0.2 - 1)}^{<0} + 0.2$$

Forward Pass: Running Example

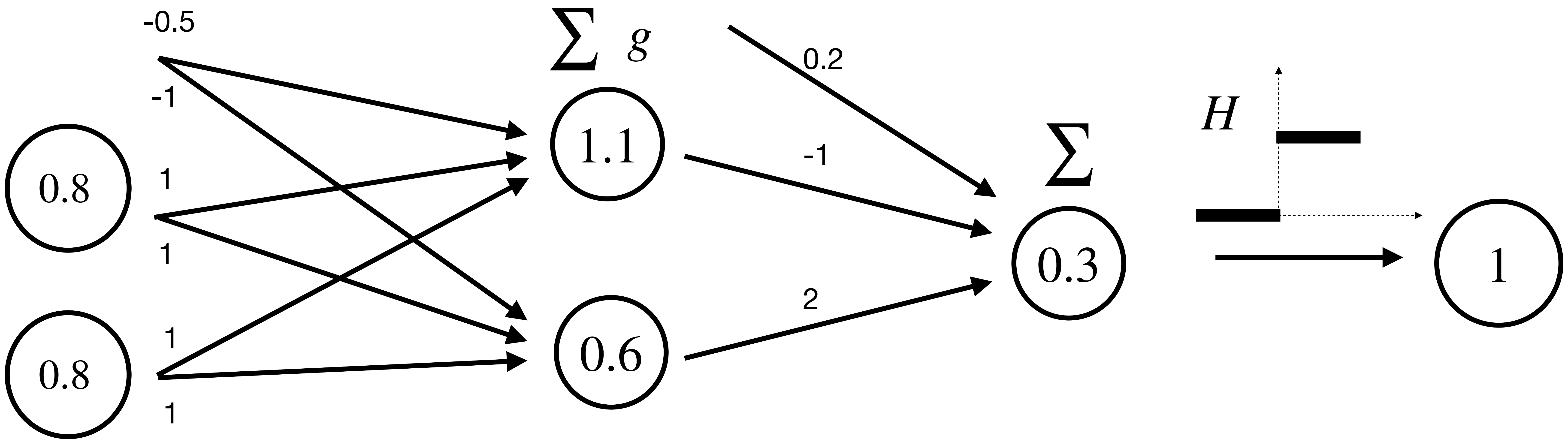


Input	Output
(0.5, 0.5)	0
(0.2, 0.2)	**1**
(0.8, 0.8)	?

Class
0
1
1

$h = 0.2$ $\hat{y} = 1$

Forward Pass: Running Example

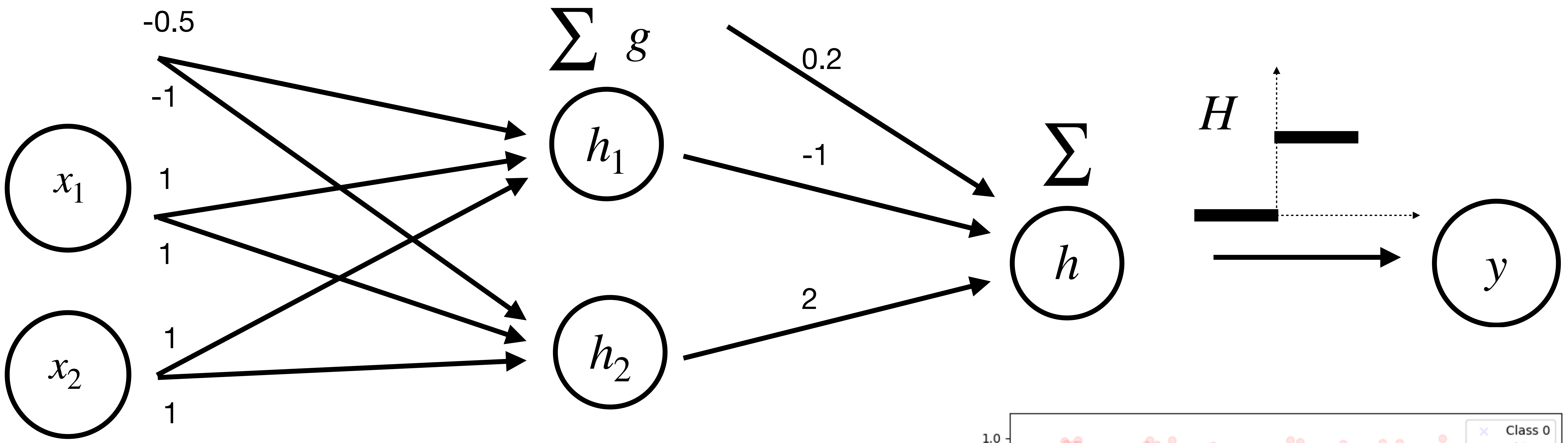


Input	Output
(0.5, 0.5)	0
(0.2, 0.2)	1
(0.8, 0.8)	**1**

Class
0
1
1

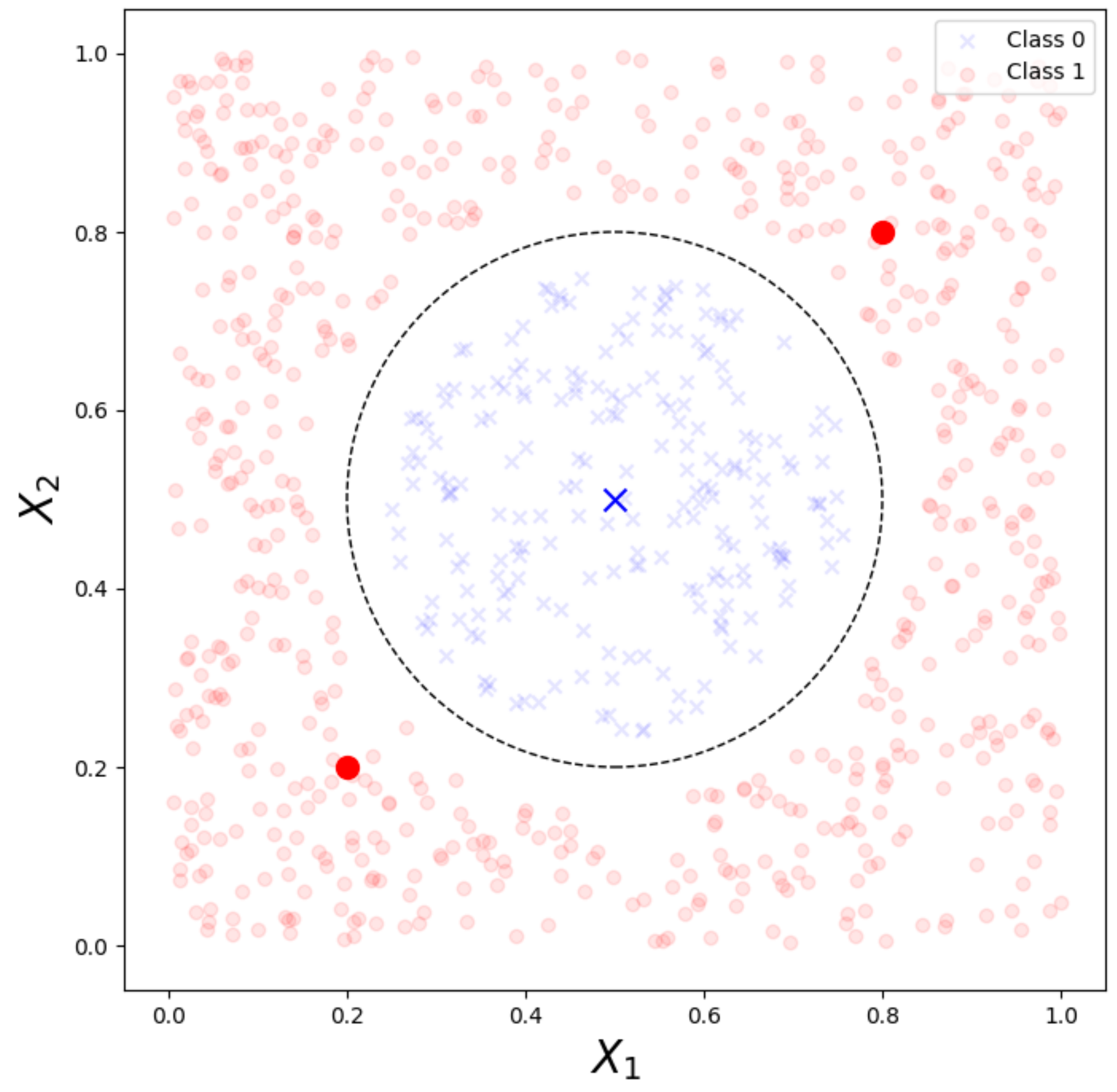
$h = 0.3$ $\hat{y} = 1$

Forward Pass: Running Example



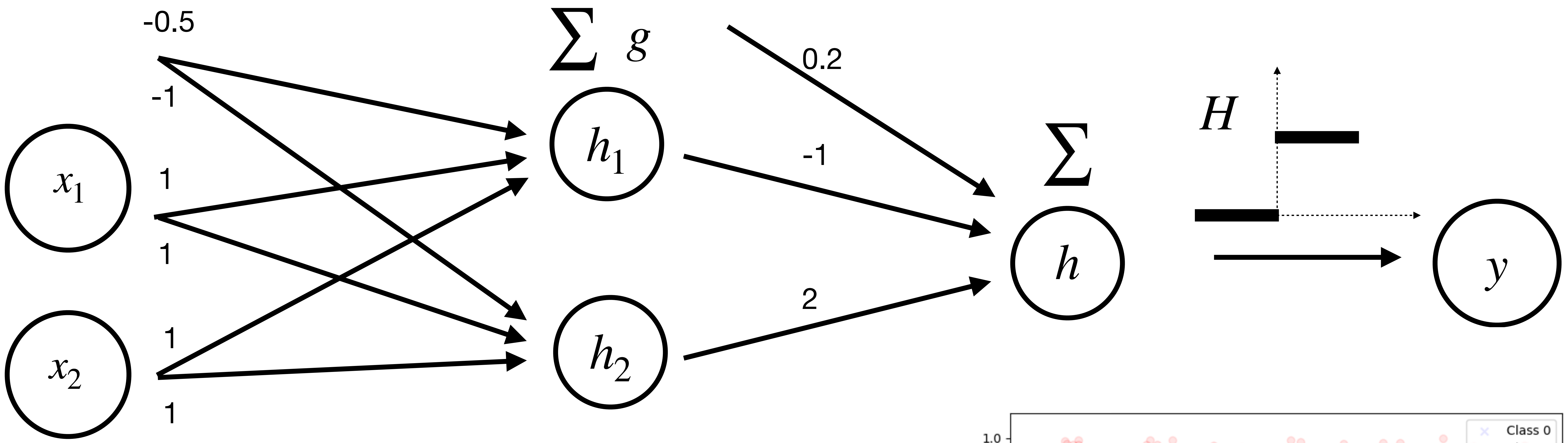
Input	Output
(0.5, 0.5)	0
(0.2, 0.2)	1
(0.8, 0.8)	1

Class
0
1
1

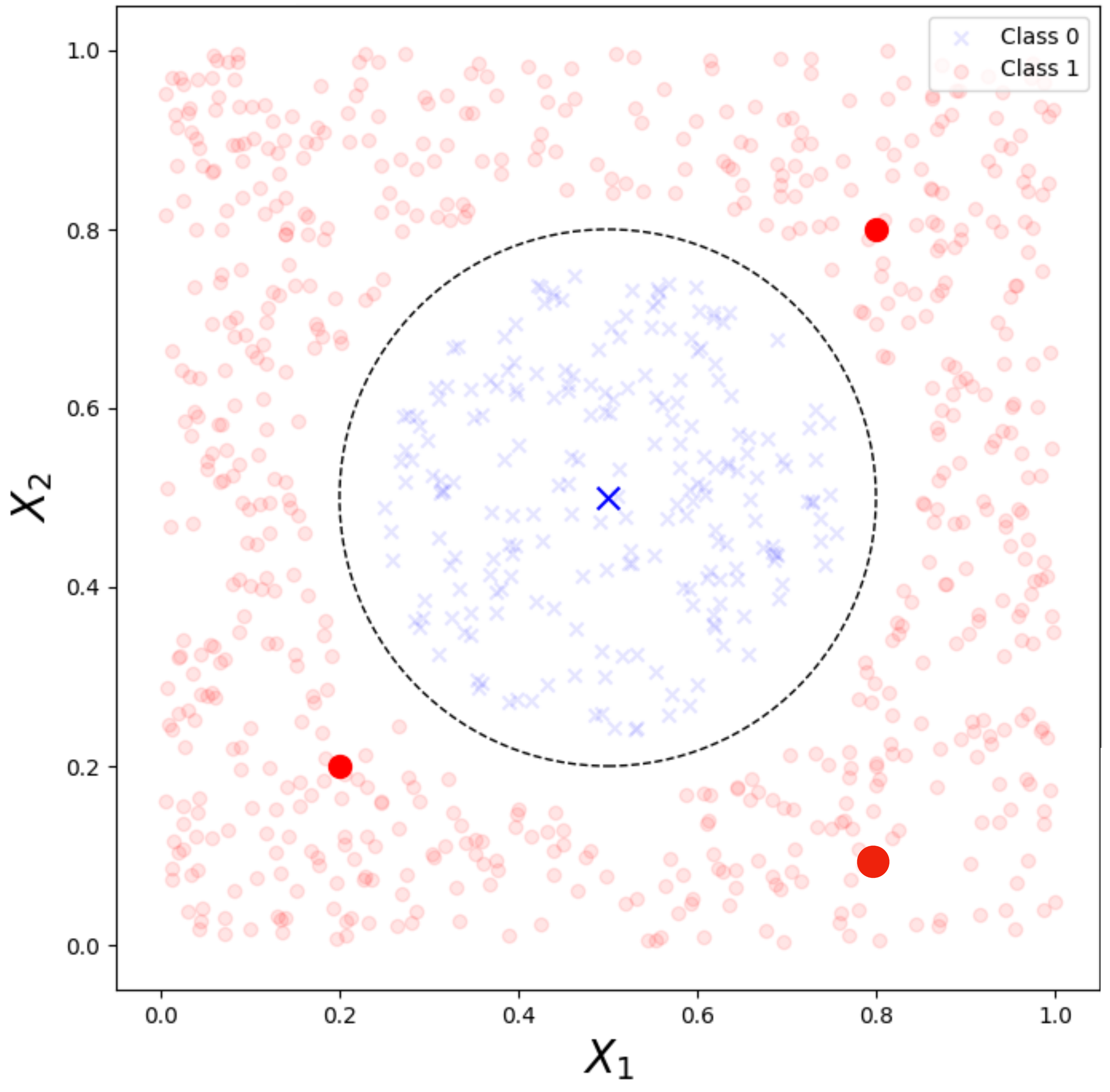


Better than linear perceptron! However…

Forward Pass: Running Example

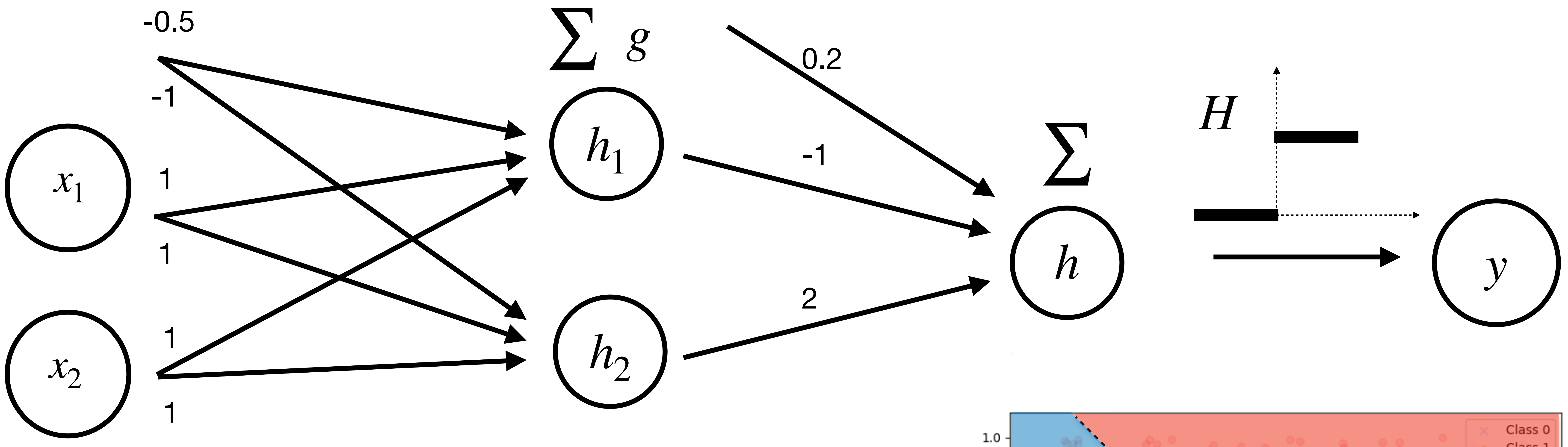


Input	Output	Class
(0.5, 0.5)	0	0
(0.2, 0.2)	1	1
(0.8, 0.8)	1	1
(0.8, 0.1)	0	1

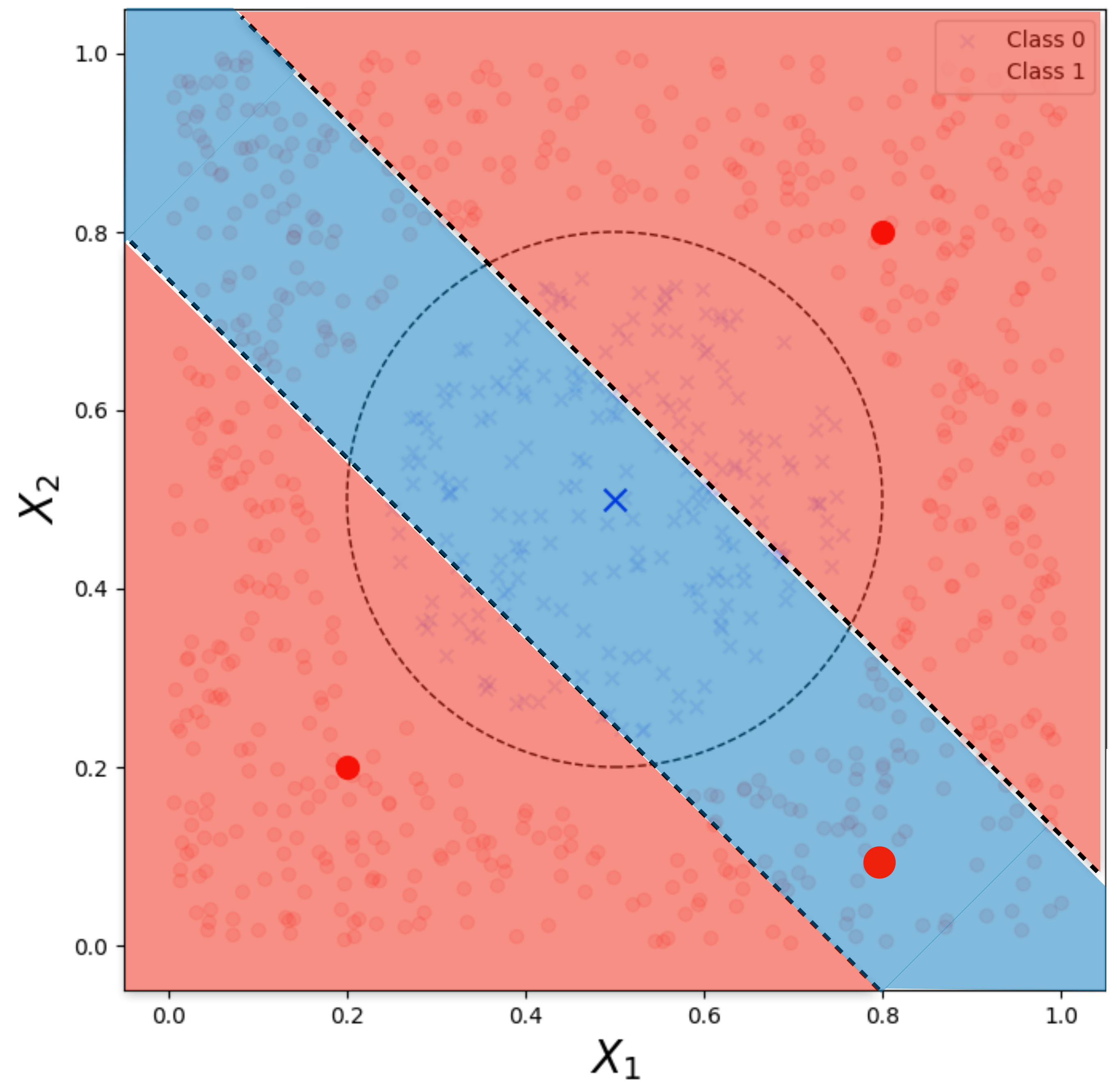


Better than linear perceptron! However…

Forward Pass: Running Example

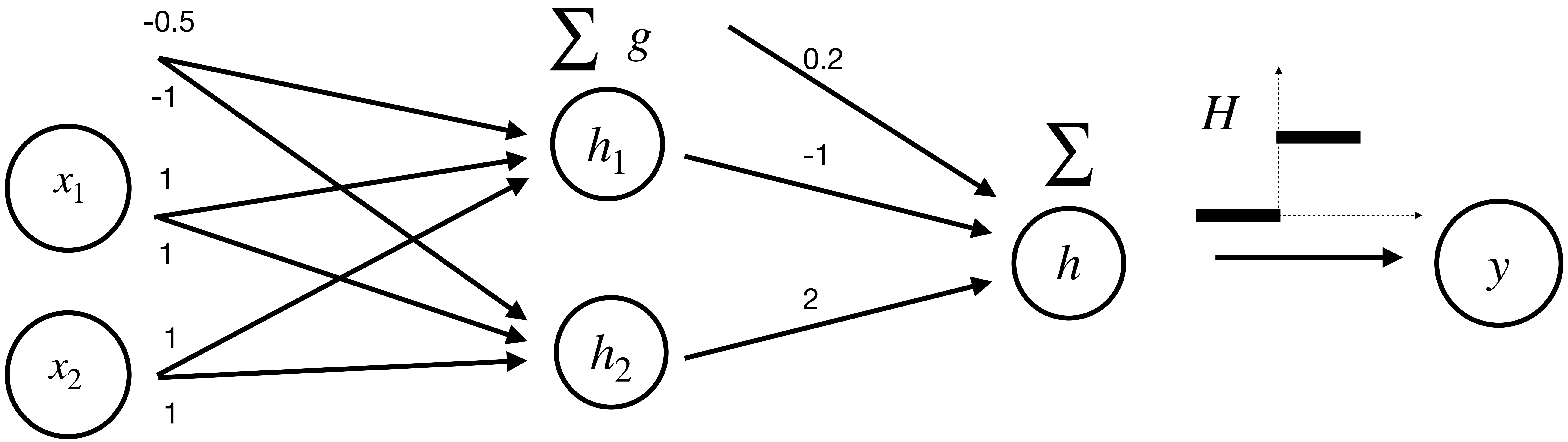


Input	Output	Class
(0.5, 0.5)	0	0
(0.2, 0.2)	1	1
(0.8, 0.8)	1	1
(0.8, 0.1)	0	1



Better than linear perceptron! However…

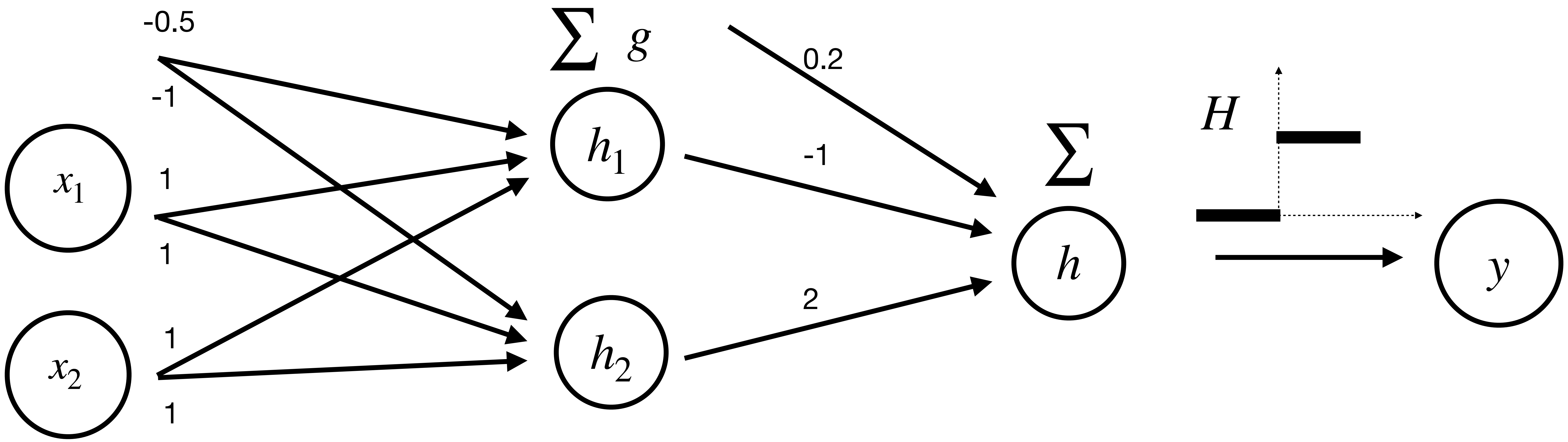
Batch Operations



Input	Output
(0.5, 0.5)	0
(0.2, 0.2)	1
(0.8, 0.8)	1
(0.8, 0.1)	0

$$
x = \begin{array}{c c}
\overbrace{\hphantom{x_1^{(1)} \quad x_2^{(1)}}}^{\text{input dimension}} & \\
\begin{pmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ x_1^{(4)} & x_2^{(4)} \end{pmatrix} & \left.\vphantom{\begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}}\right] \text{\#sample in the batch}
\end{array}
$$

Matrix Notation



$$f(x;\theta) = g\left(x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

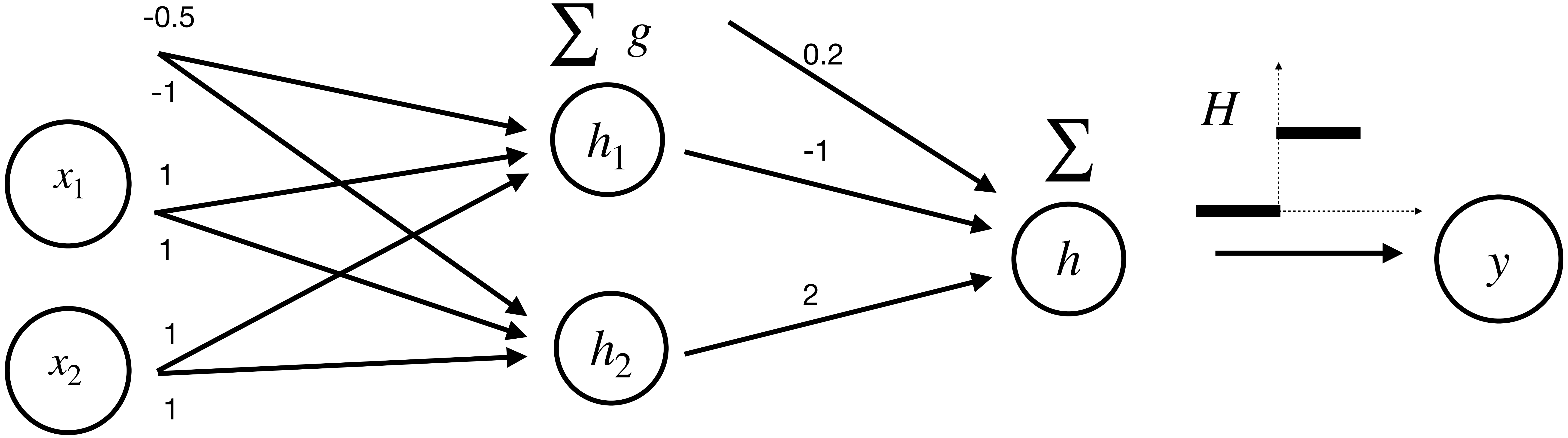
$$x = \begin{array}{cl} \text{input dimension} & \\ \begin{pmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ x_1^{(4)} & x_2^{(4)} \end{pmatrix} & \text{\#sample in the batch} \end{array}$$

$\mathbf{W}^{(1)} = ?$ $\qquad$ $\mathbf{b}^{(1)} = ?$

$\mathbf{W}^{(2)} = ?$ $\qquad$ $\mathbf{b}^{(2)} = ?$

Take 2 minutes to think

Matrix Notation



broadcasting

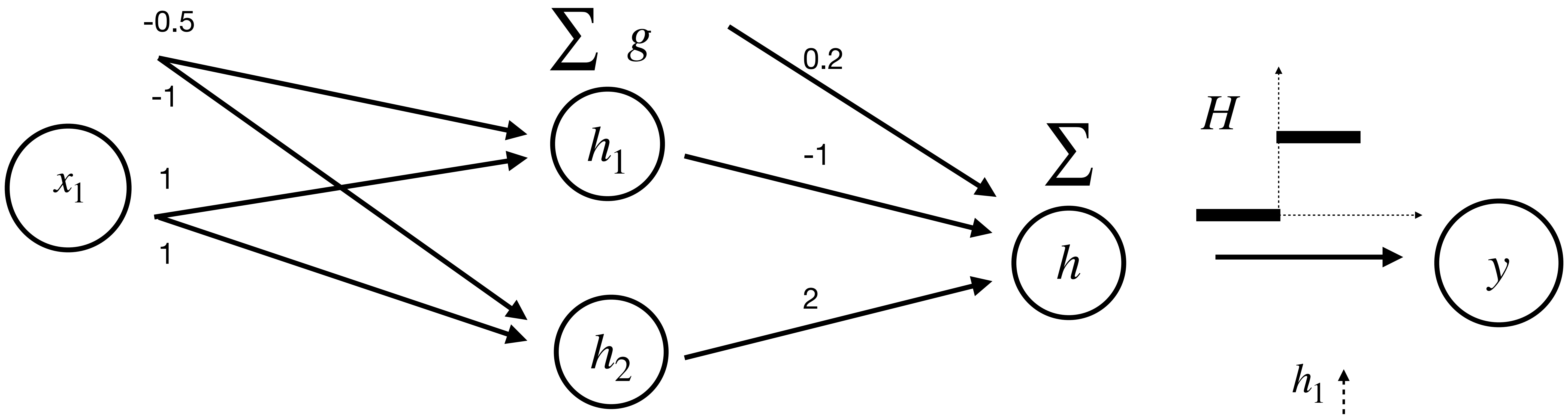
$$f(x;\theta) = g\left(x \cdot \mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right) \cdot \mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

$$\mathbf{W}^{(1)} = \begin{pmatrix} 1 1 \\ 1 1 \end{pmatrix} \qquad \mathbf{b}^{(1)} = (-0.5 \ \ -1)$$

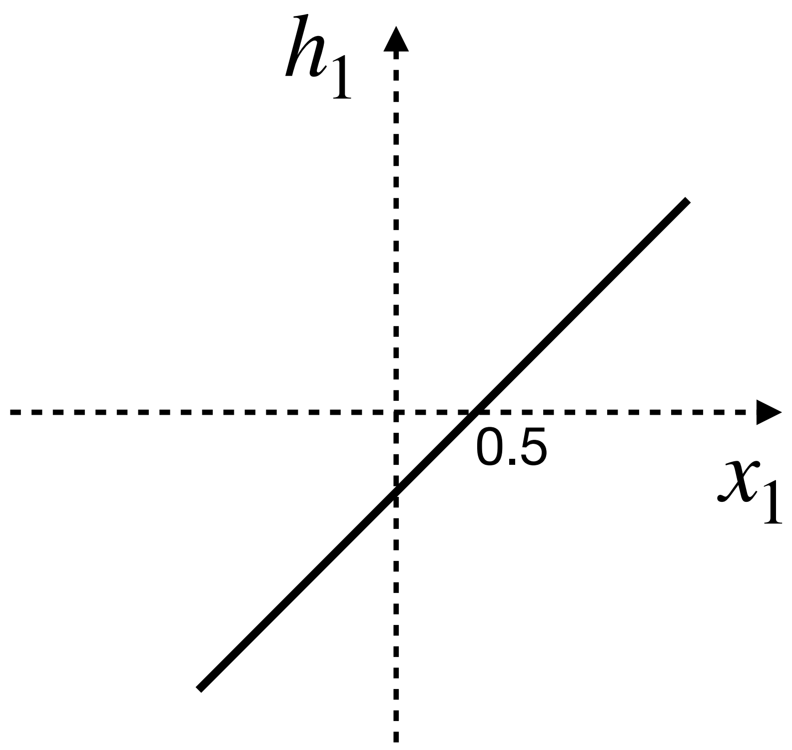
$$\mathbf{W}^{(2)} = \begin{pmatrix} -1 \\ 2 \end{pmatrix} \qquad \mathbf{b}^{(2)} = (0.2)$$

$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$

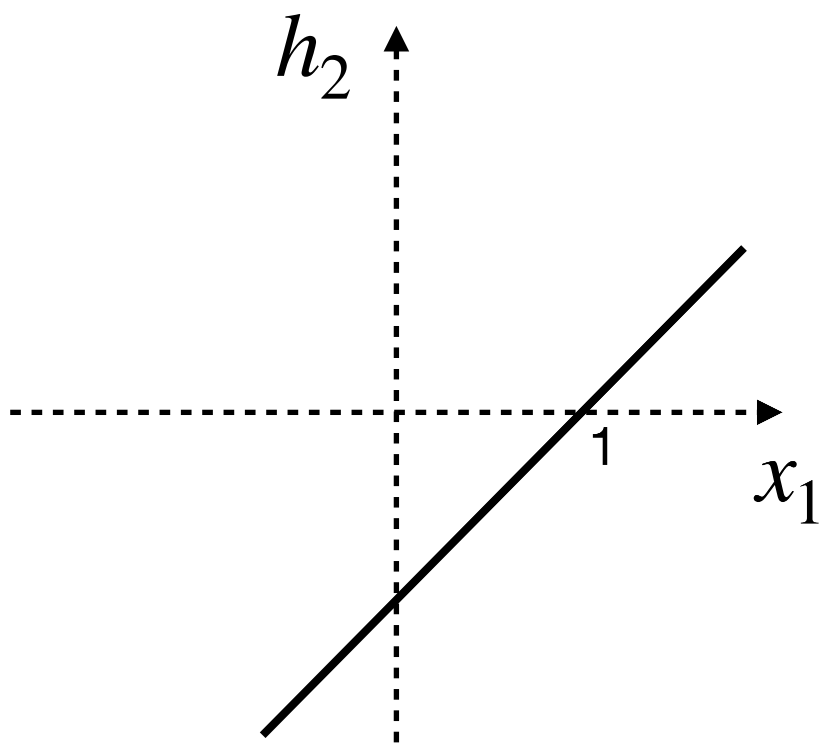
Network's Representation (1D)



Which family of functions are represented?

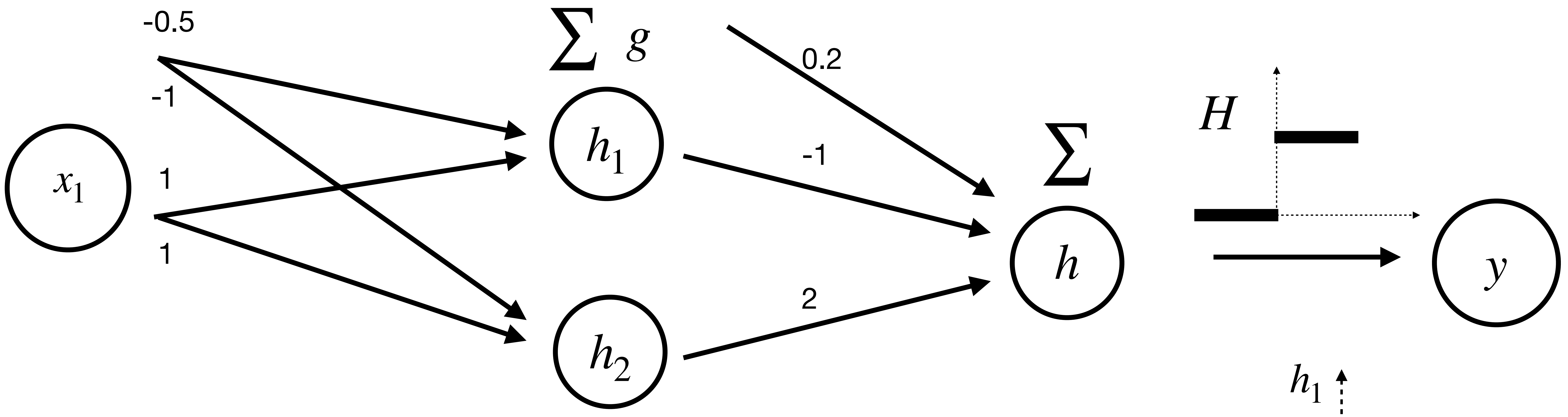


$h_1(x_1) = \max\{\mathbf{x_1 - 0.5}, 0\}$

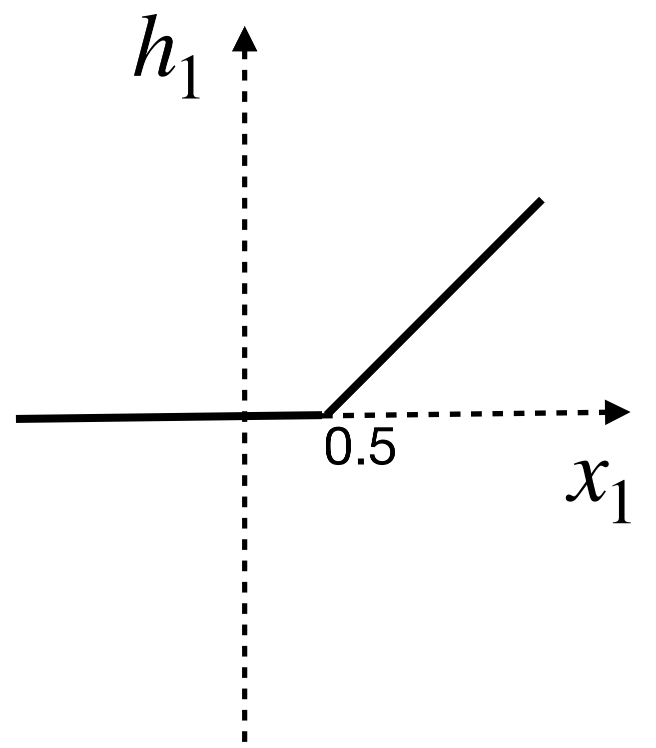


$h_2(x_1) = \max\{\mathbf{x_1 - 1}, 0\}$

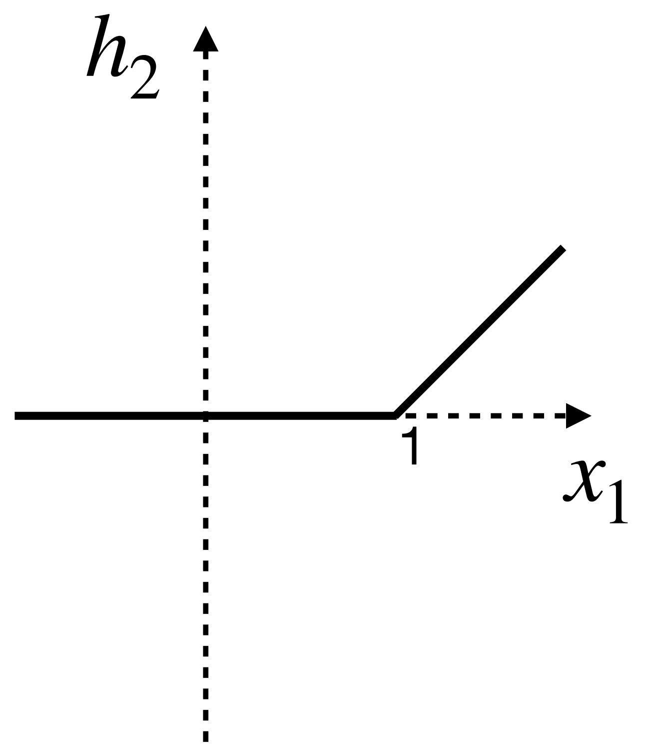
Network's Representation (1D)



Which family of functions are represented?

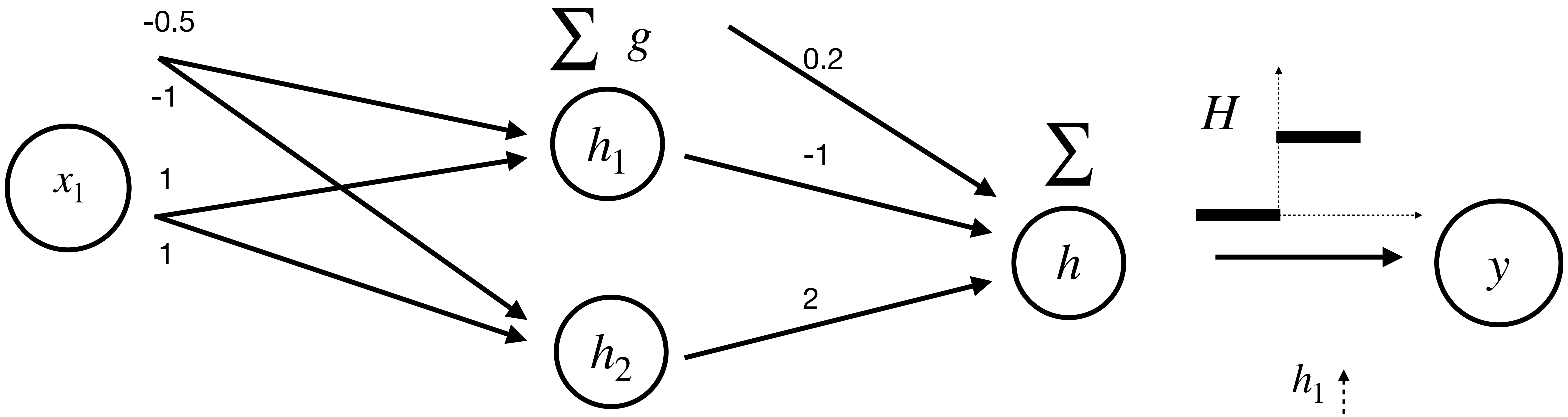


$h_1(x_1) = \max\{x_1 - 0.5, 0\}$

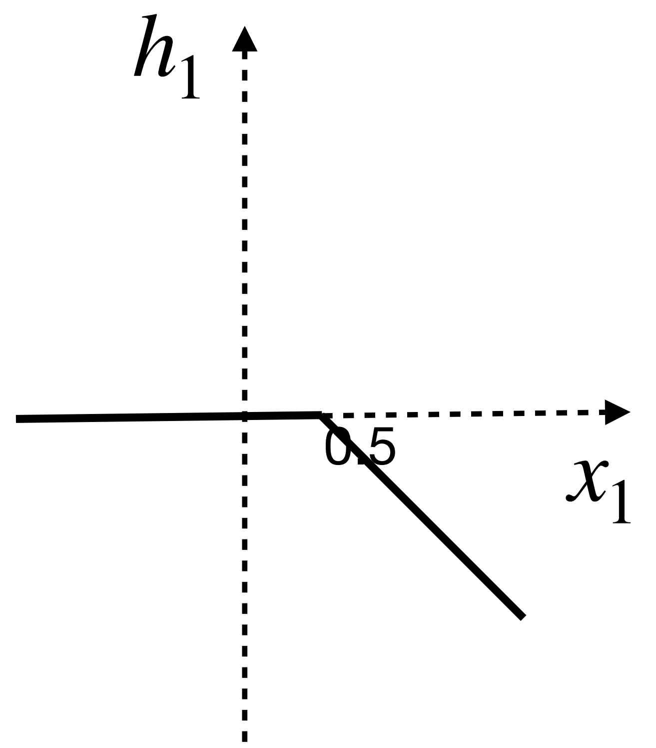


$h_2(x_1) = \max\{x_1 - 1, 0\}$

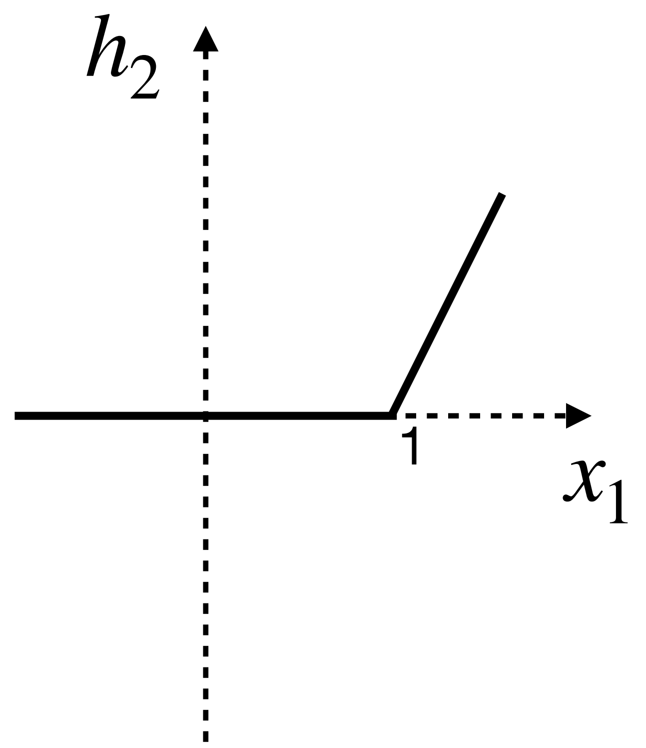
Network's Representation (1D)



Which family of functions are represented?

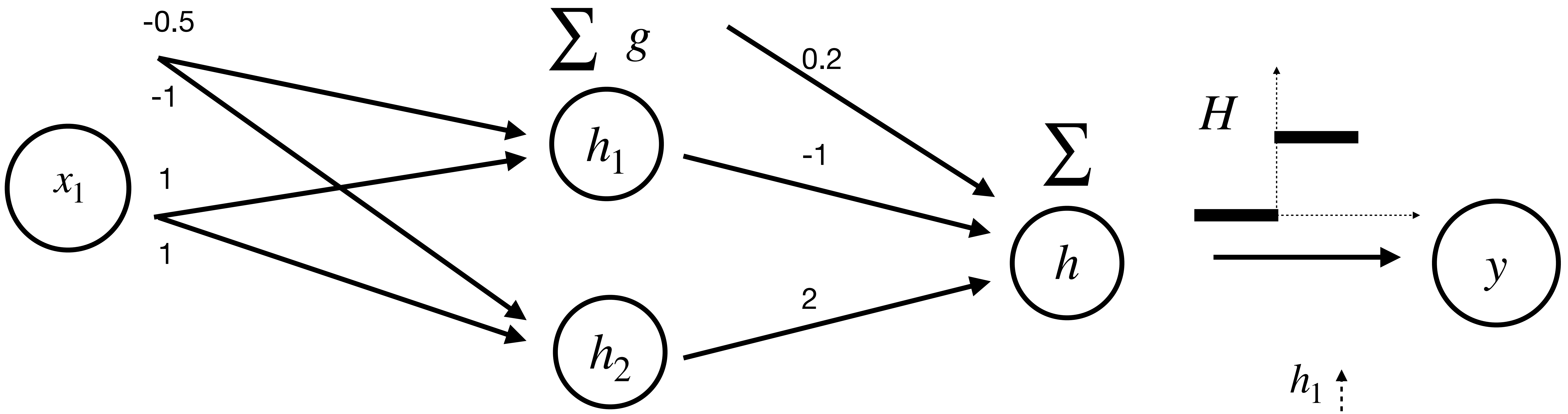


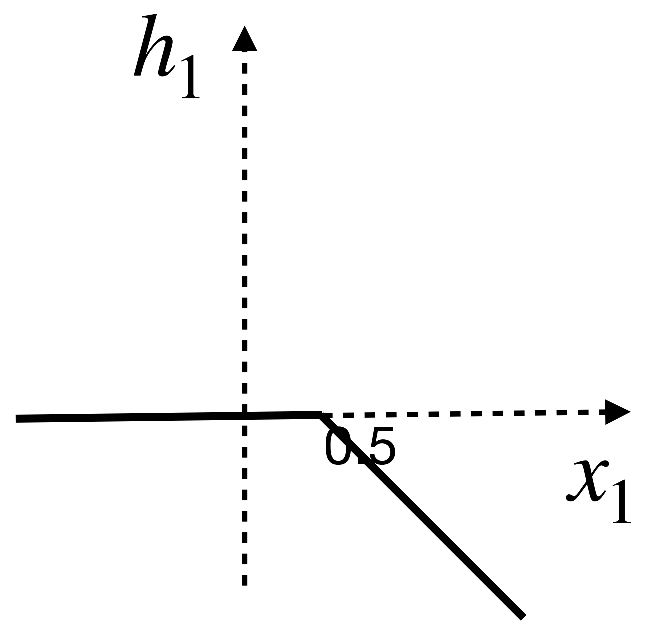
$-h_1(x_1)$

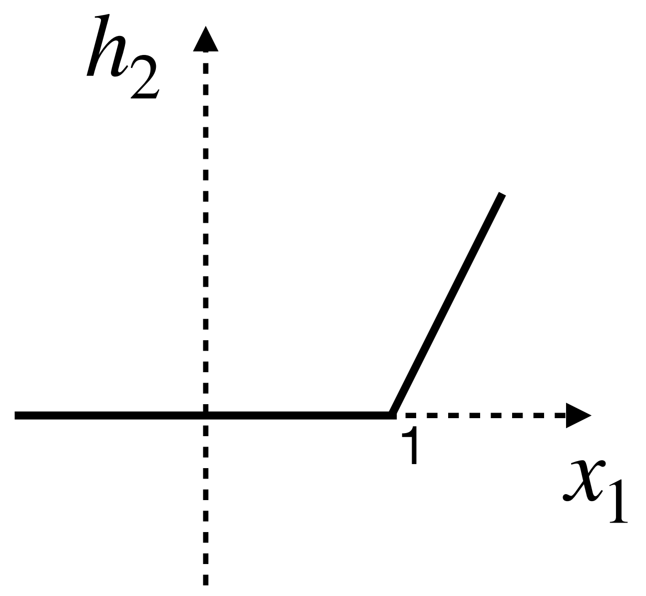


$2h_2(x_1)$

Network's Representation (1D)

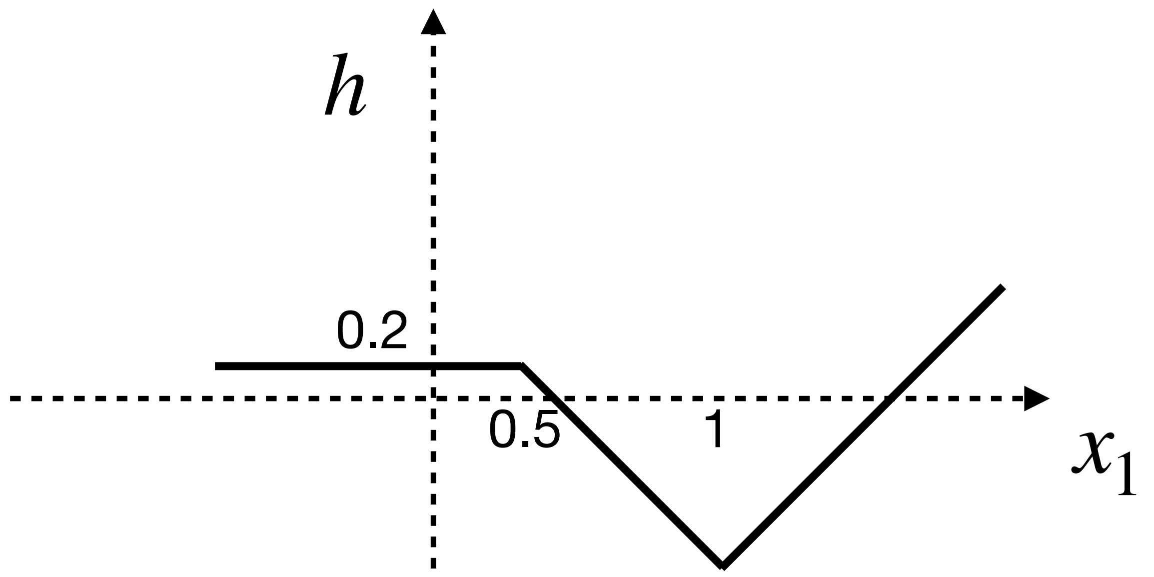






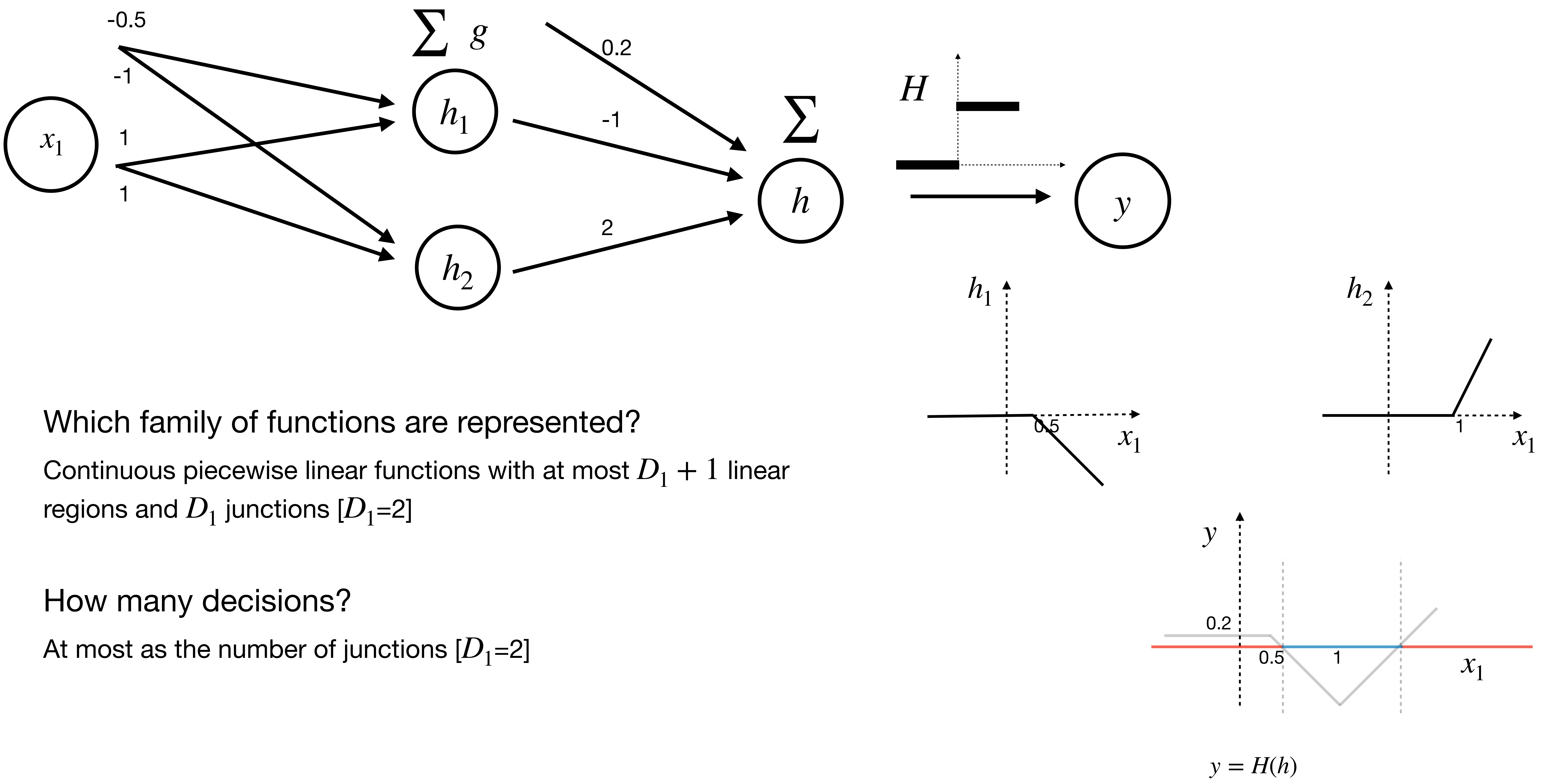
Which family of functions are represented?

Continuous piecewise linear functions with at most $D_1 + 1$ linear regions and D_1 junctions [D_1=2]



$h = -h_1 + 2h_2 + 0.2$

Network's Representation (1D)



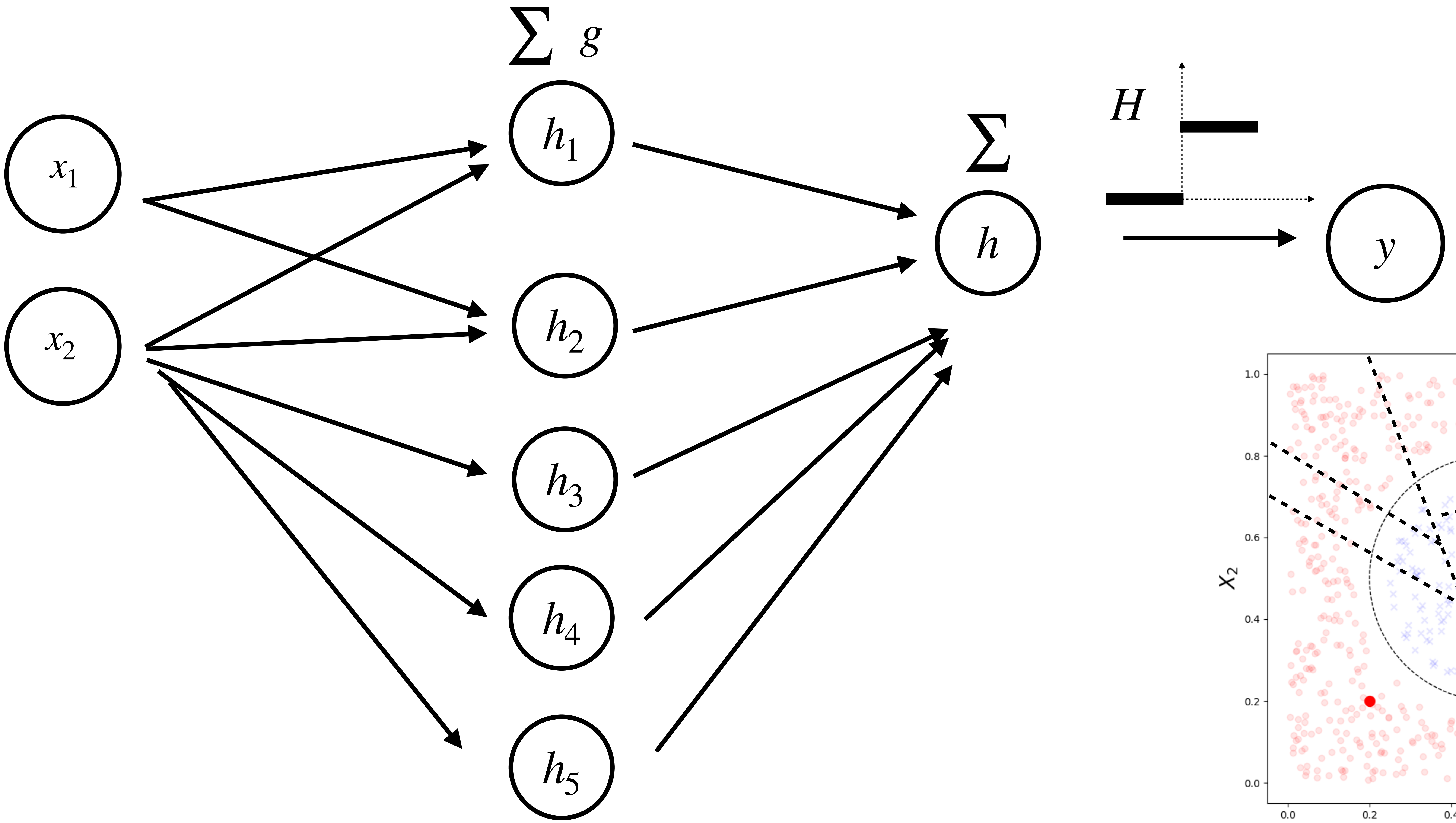
Which family of functions are represented?

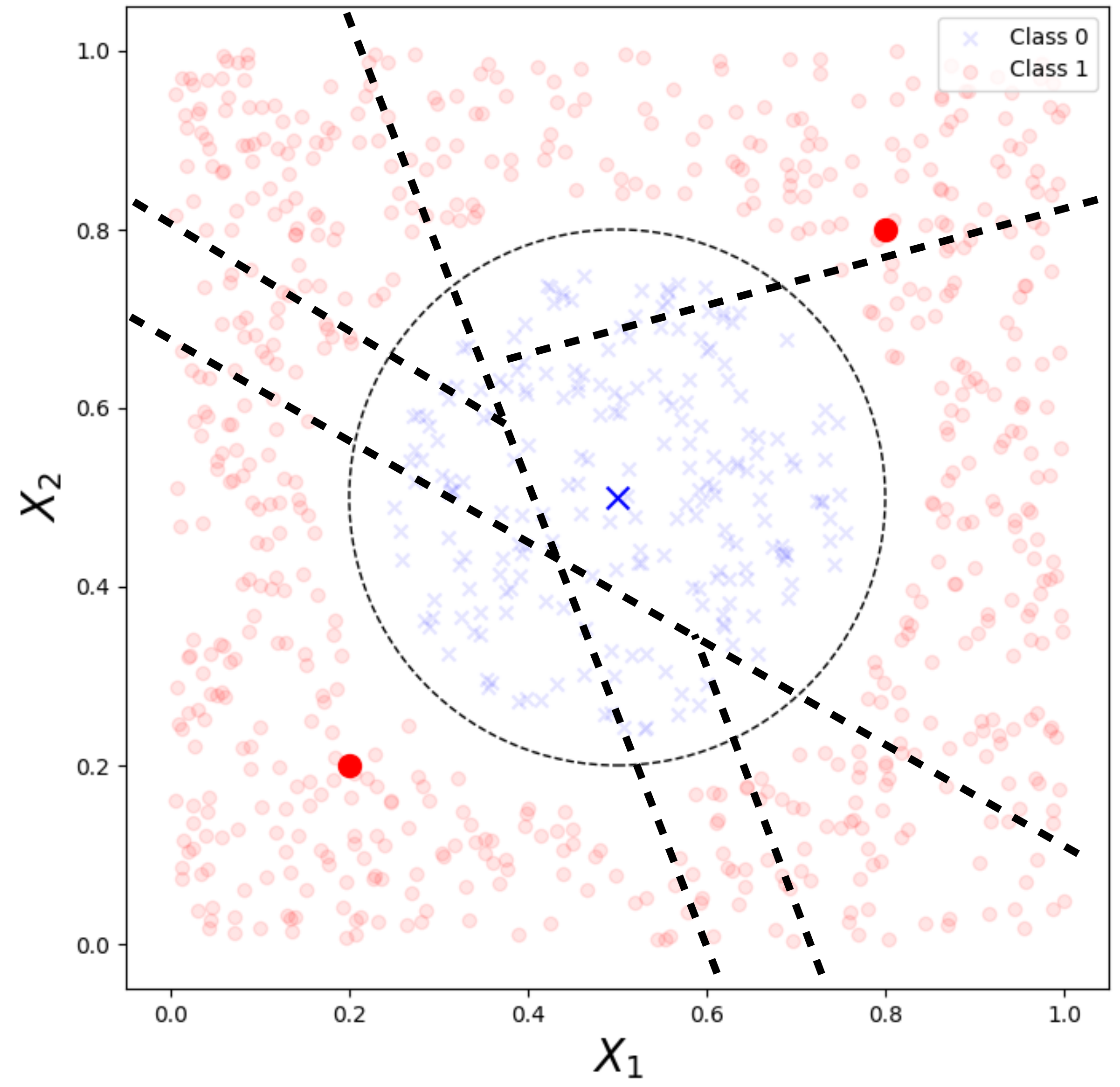
Continuous piecewise linear functions with at most $D_1 + 1$ linear regions and D_1 junctions [D_1=2]

How many decisions?

At most as the number of junctions [D_1=2]

Network's Representation



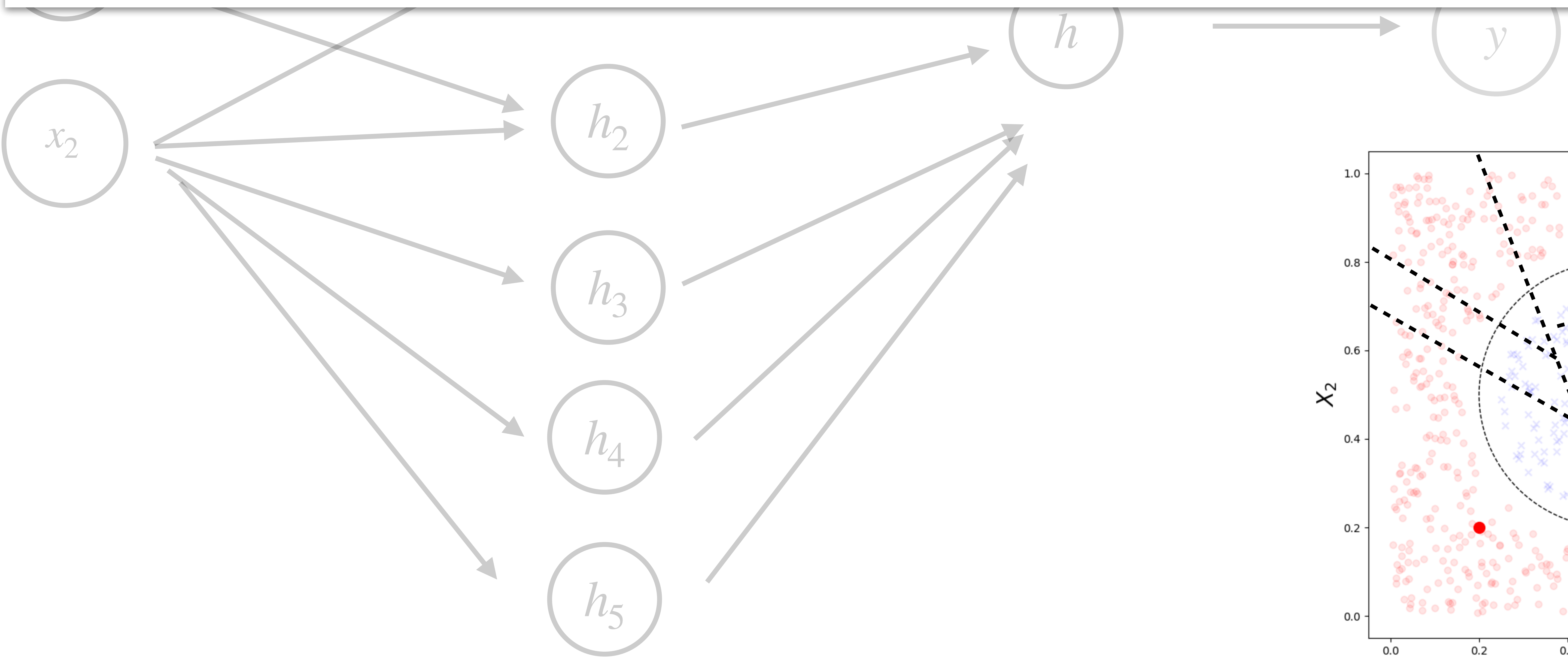


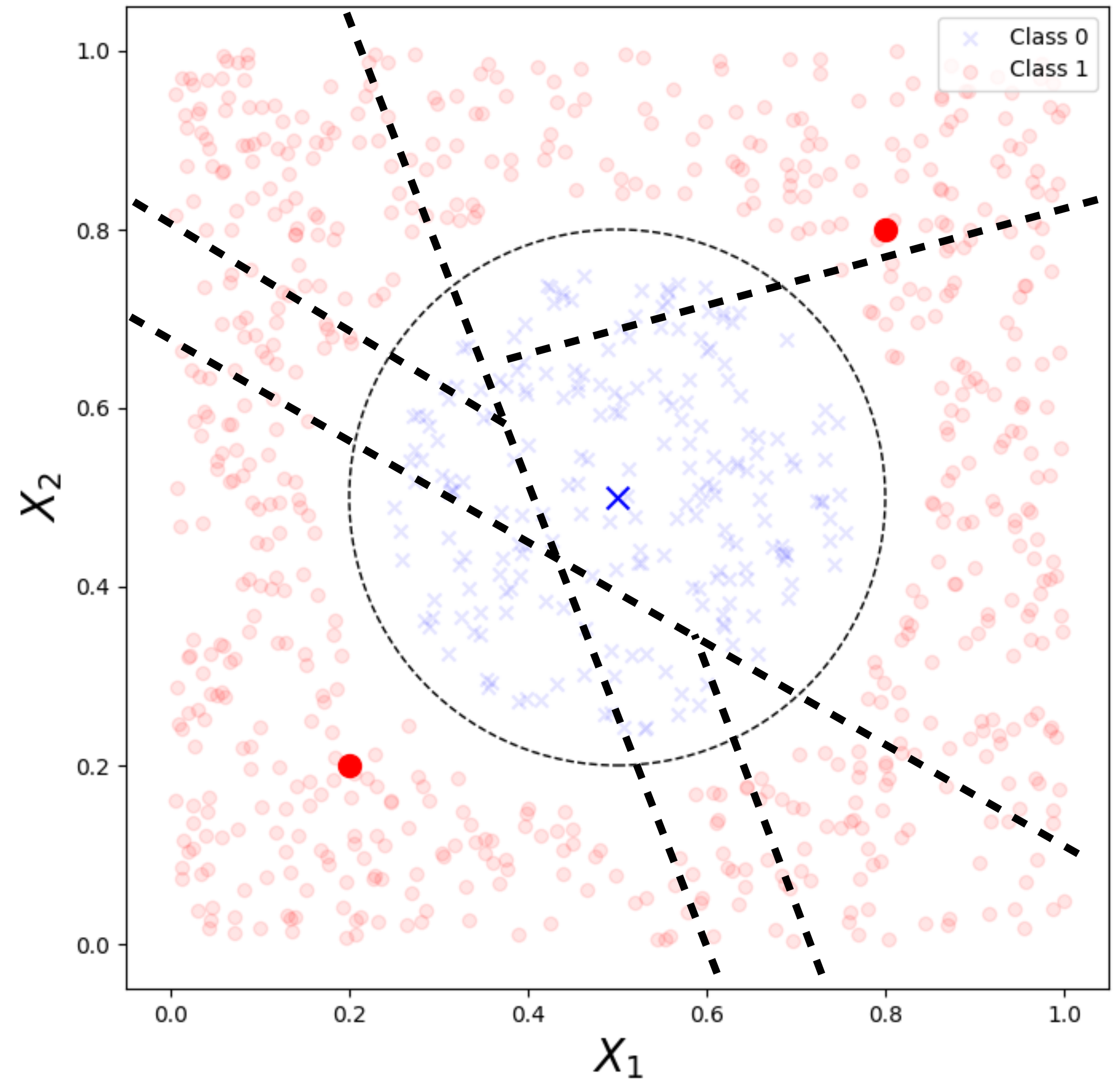
Adding more hidden units allows the model to approximate more complex functions

Shallow Networks

Universal Approximation Theorem

*A one hidden layer network, a **shallow network**, with enough hidden units and an activation function can approximate arbitrarily closely any continuous function on an N dimensional input space.*





Adding more hidden units allows the model to approximate more complex functions

Wider or Deeper Networks (Optional)

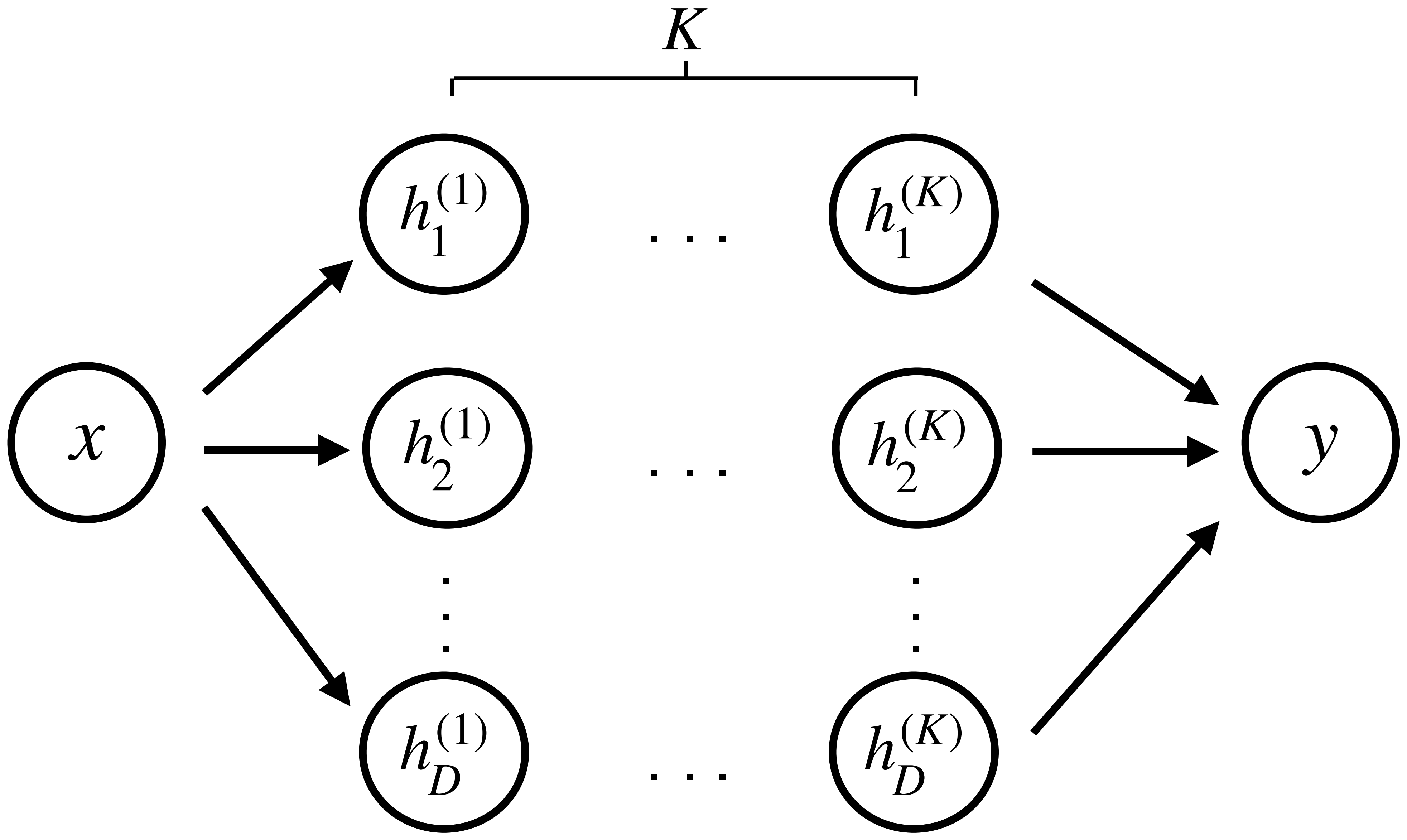
Universal Approximation Theorem

A one hidden layer network, a ***shallow network****, with enough hidden units and an activation function can approximate arbitrarily closely any continuous function on an N dimensional input space.*

(1d input/output) A network with K hidden layers each consisting of D hidden units each:

- Size: $3D + 1 + (K-1)D(D+1)$ parameters
- Representation capacity: $(D+1)^K$ linear regions

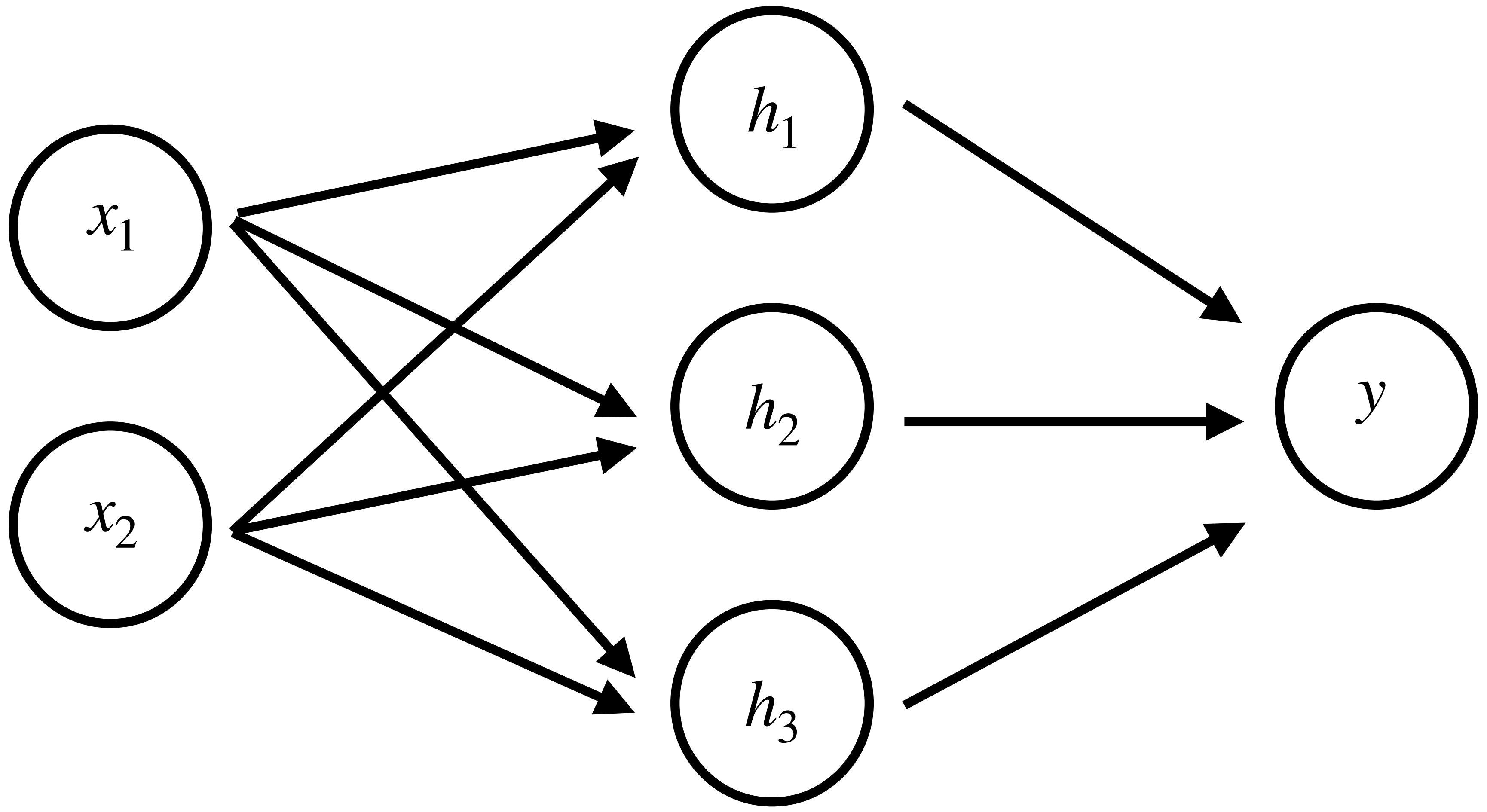
Number of parameters grows linearly in the depth K while the number of different regions grows exponentially in K.



Training steps

Iterate until convergence

1. Forward pass: for batch x compute output $\hat{y} = f(x; \theta)$
2. Evaluate: compare the $\hat{y}$ with the class label y
3. Backward pass: update the parameters θ

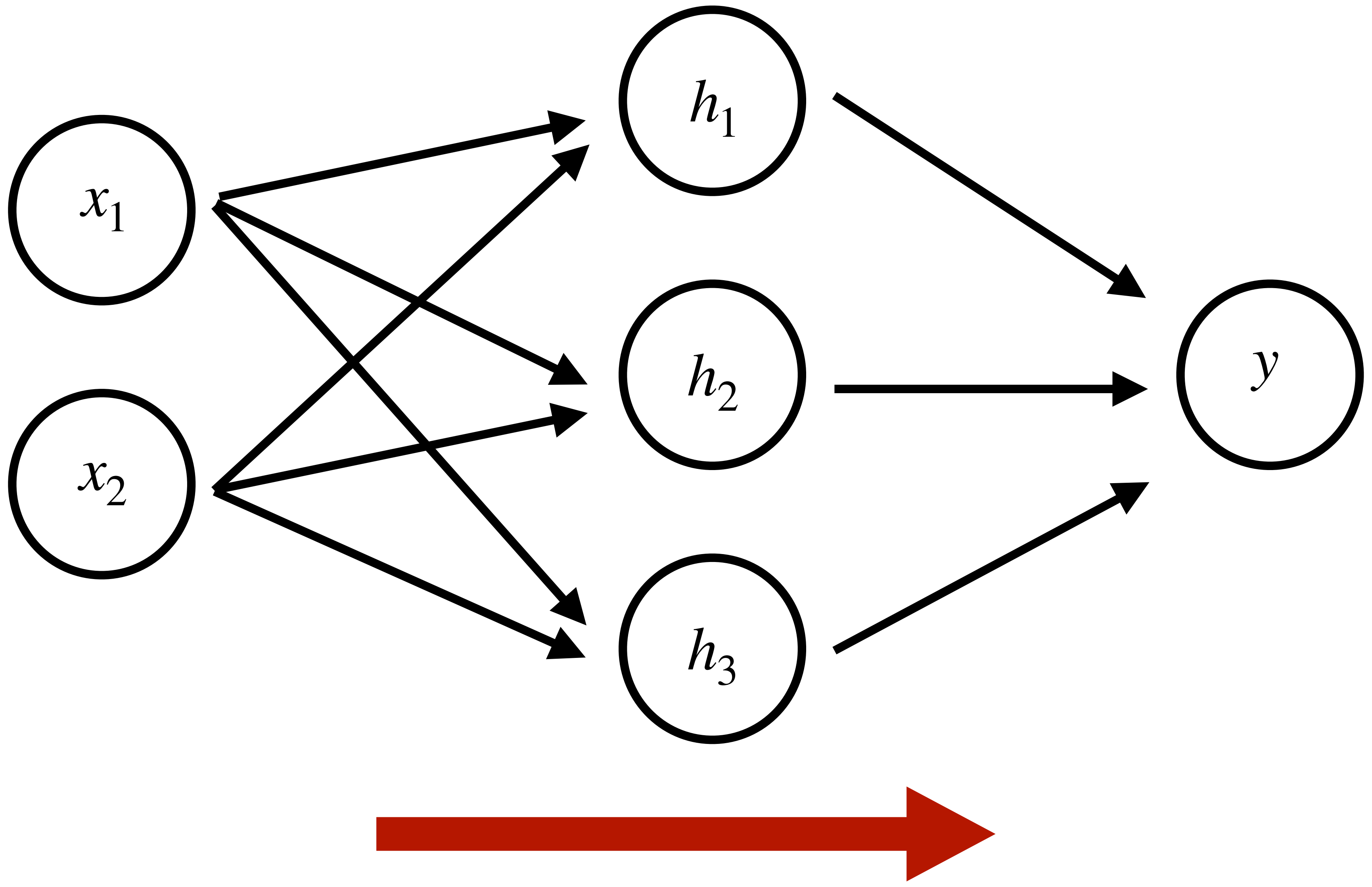


Training steps

Iterate until convergence

1. **Forward pass: for batch x compute output $\hat{y} = f(x; \theta)$**
2. Evaluate: compare the $\hat{y}$ with the class label y
3. Backward pass: update the parameters θ

$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$

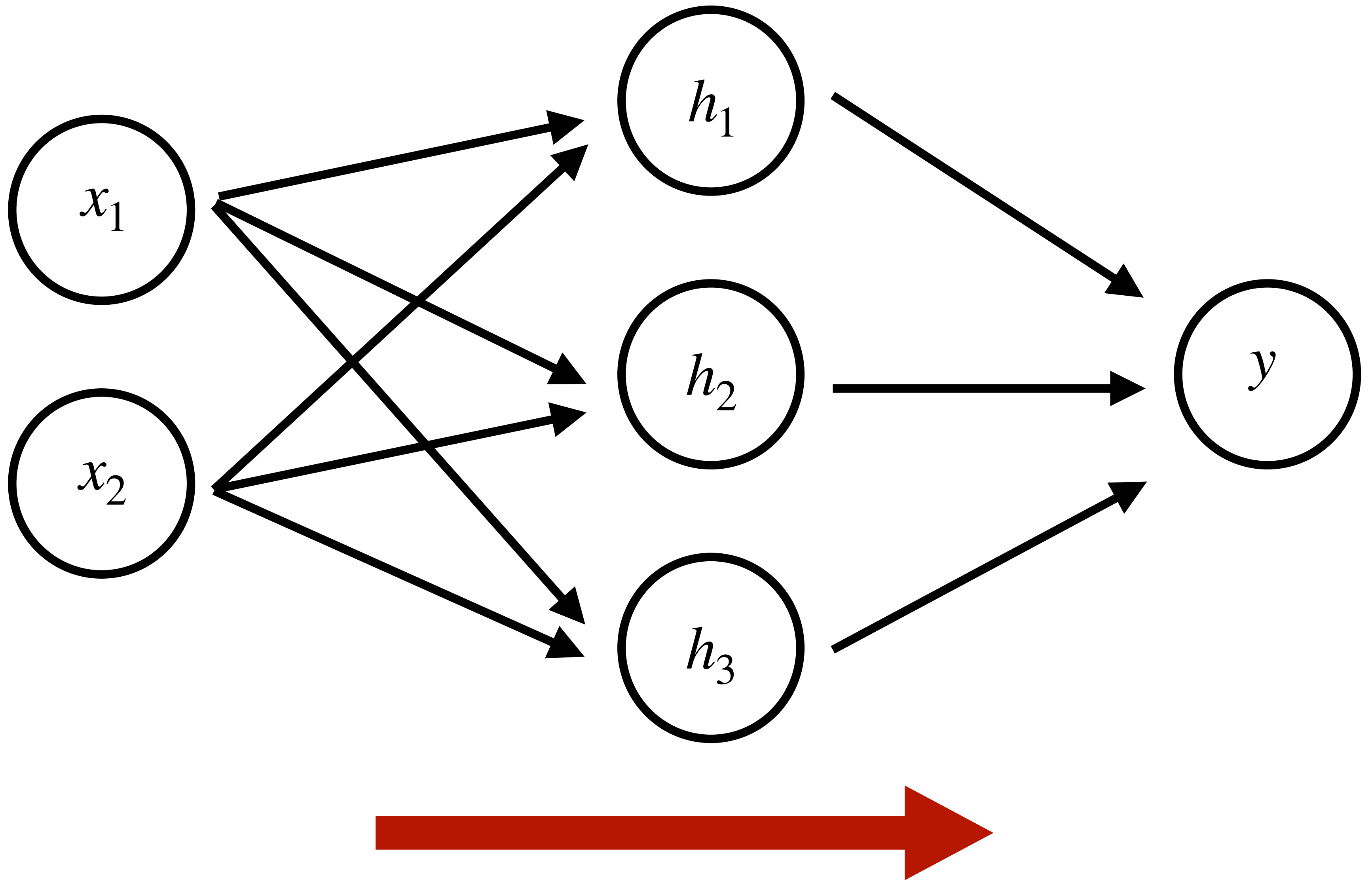


Training steps

Iterate until convergence

1. **Forward pass: for batch x compute output $\hat{y} = f(x; \theta)$**
2. Evaluate: compare the $\hat{y}$ with the class label y
3. Backward pass: update the parameters θ

$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$



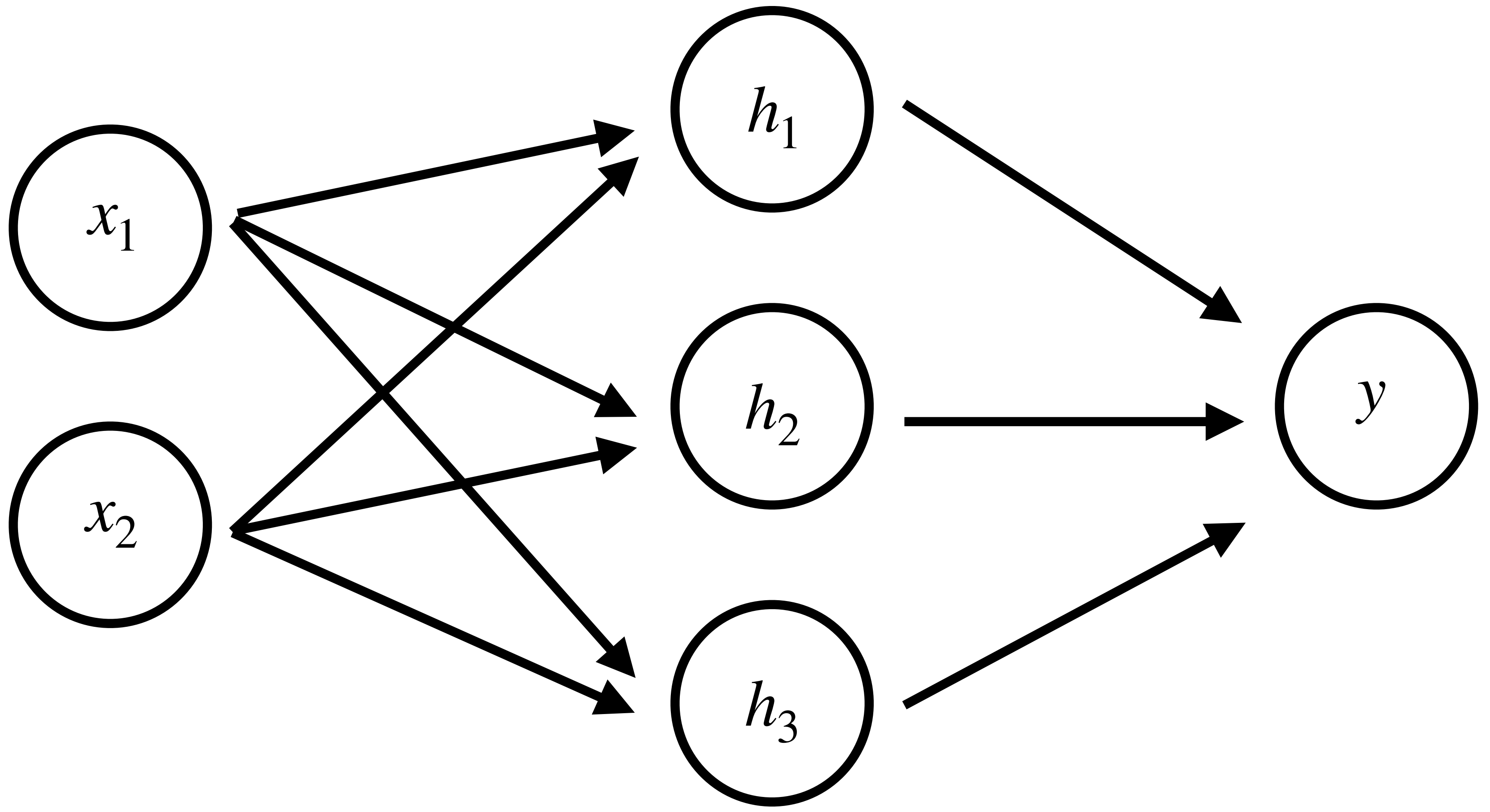
$$\hat{y} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Training steps

Iterate until convergence

1. Forward pass: for batch x compute output $\hat{y} = f(x;\theta)$
2. **Evaluate: compare the $\hat{y}$ with the class label y**
3. Backward pass: update the parameters θ

$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$



$$\hat{y} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \quad ? \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Need to be comparable

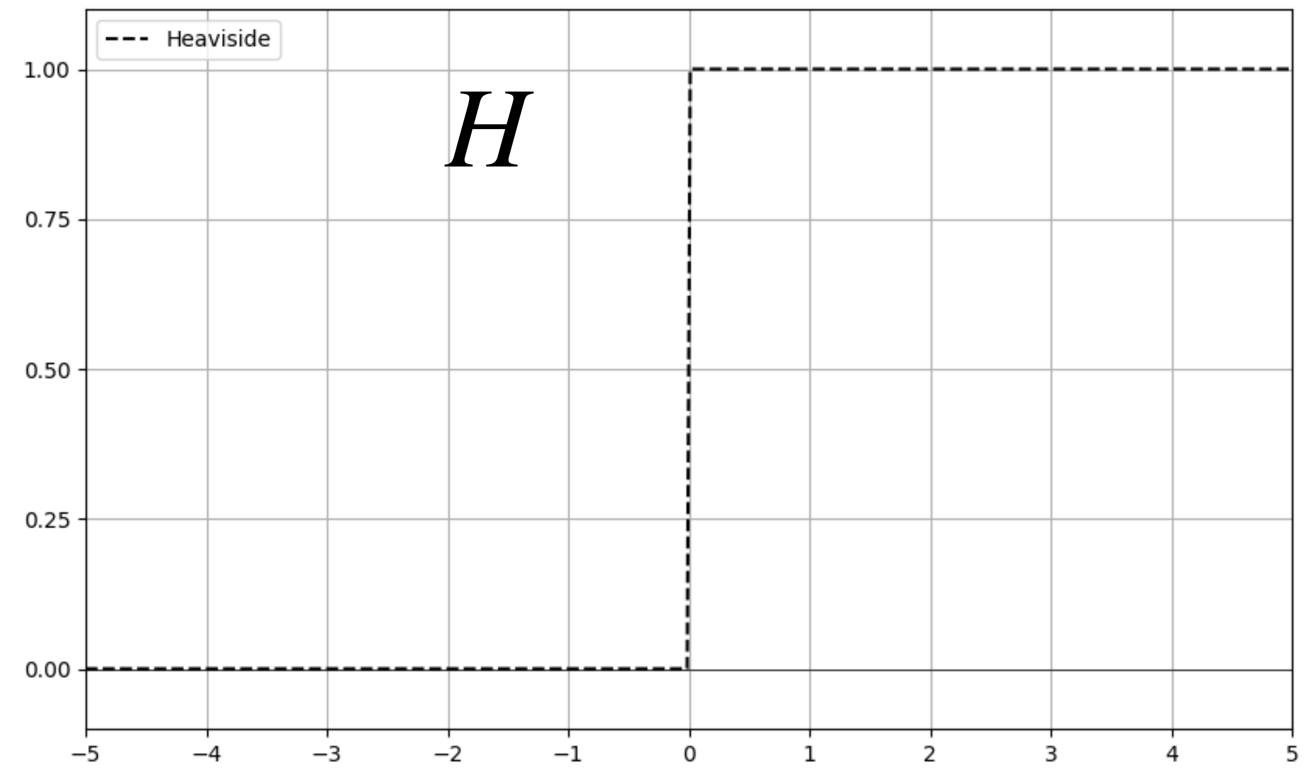
Output layer

Last layer dimension compatible with output

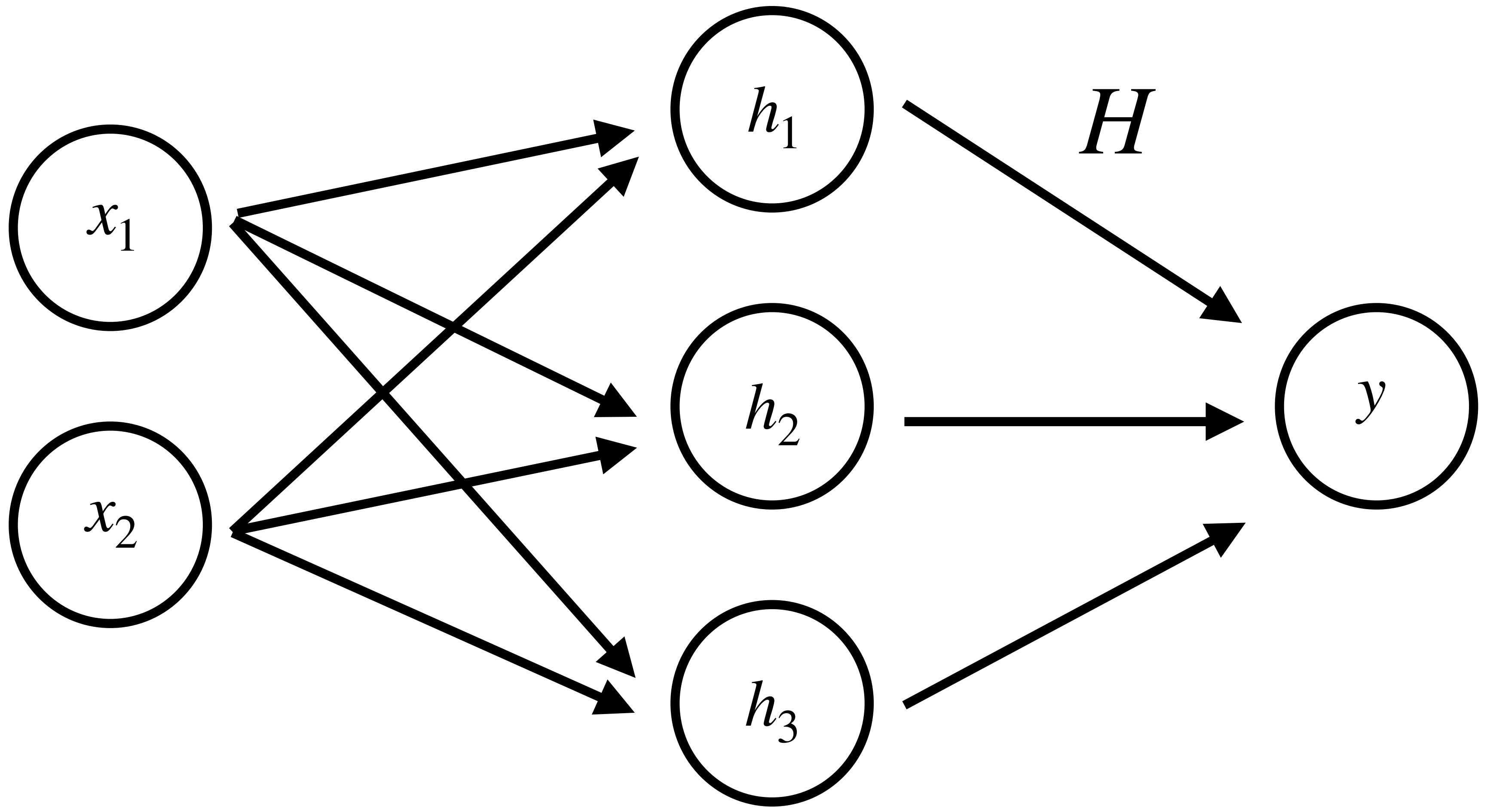
Last layer activation function

- Values aligned with the labels
- ‘Usable’ gradient

Heaviside function



$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$

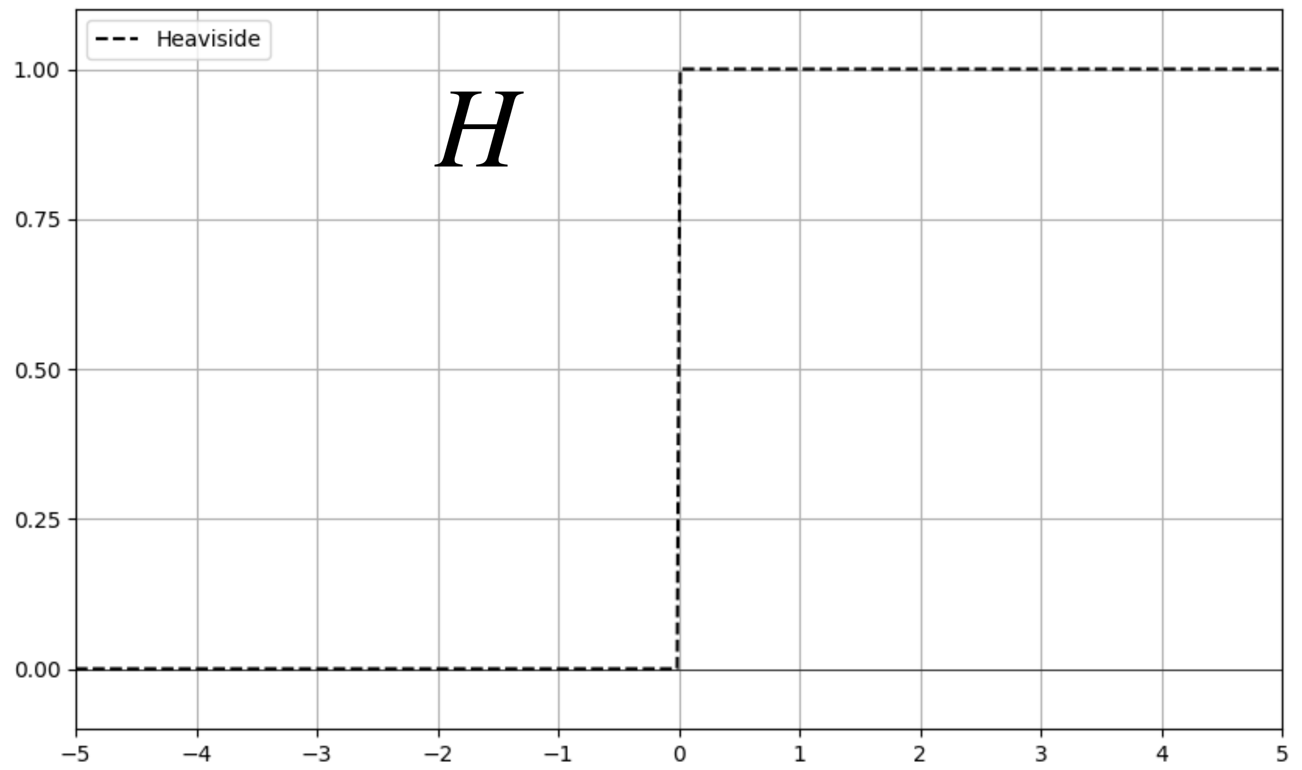


$$\hat{y} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \quad ? \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Output layer

Last layer dimension compatible with output

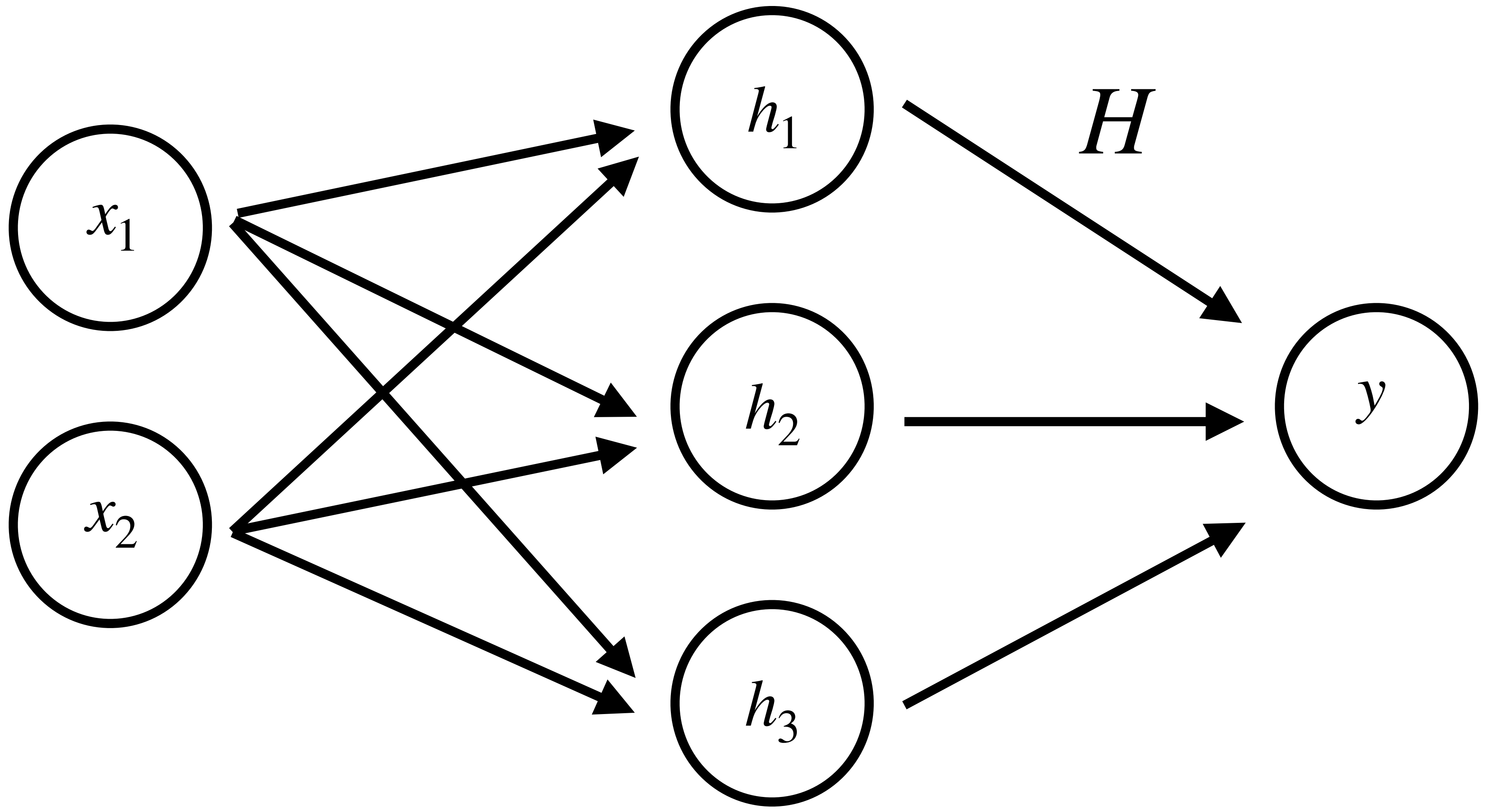
Heaviside function



Last layer activation function

- Values aligned with the labels ⟶ H maps to classes 0, 1
- ~~‘Usable’ gradient~~ ⟶ The derivative is constantly 0

$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$



$$\hat{y} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \quad ? \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Output layer

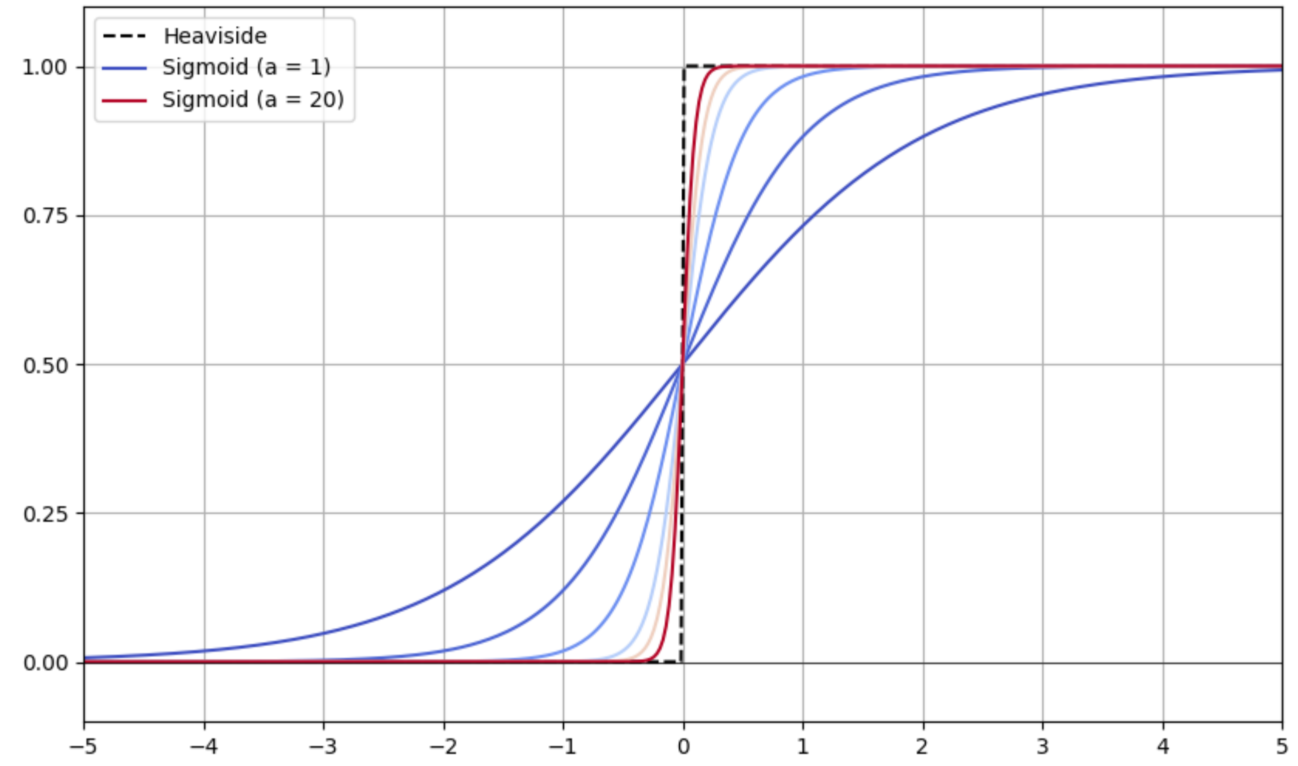
Last layer dimension compatible with output

Last layer activation function

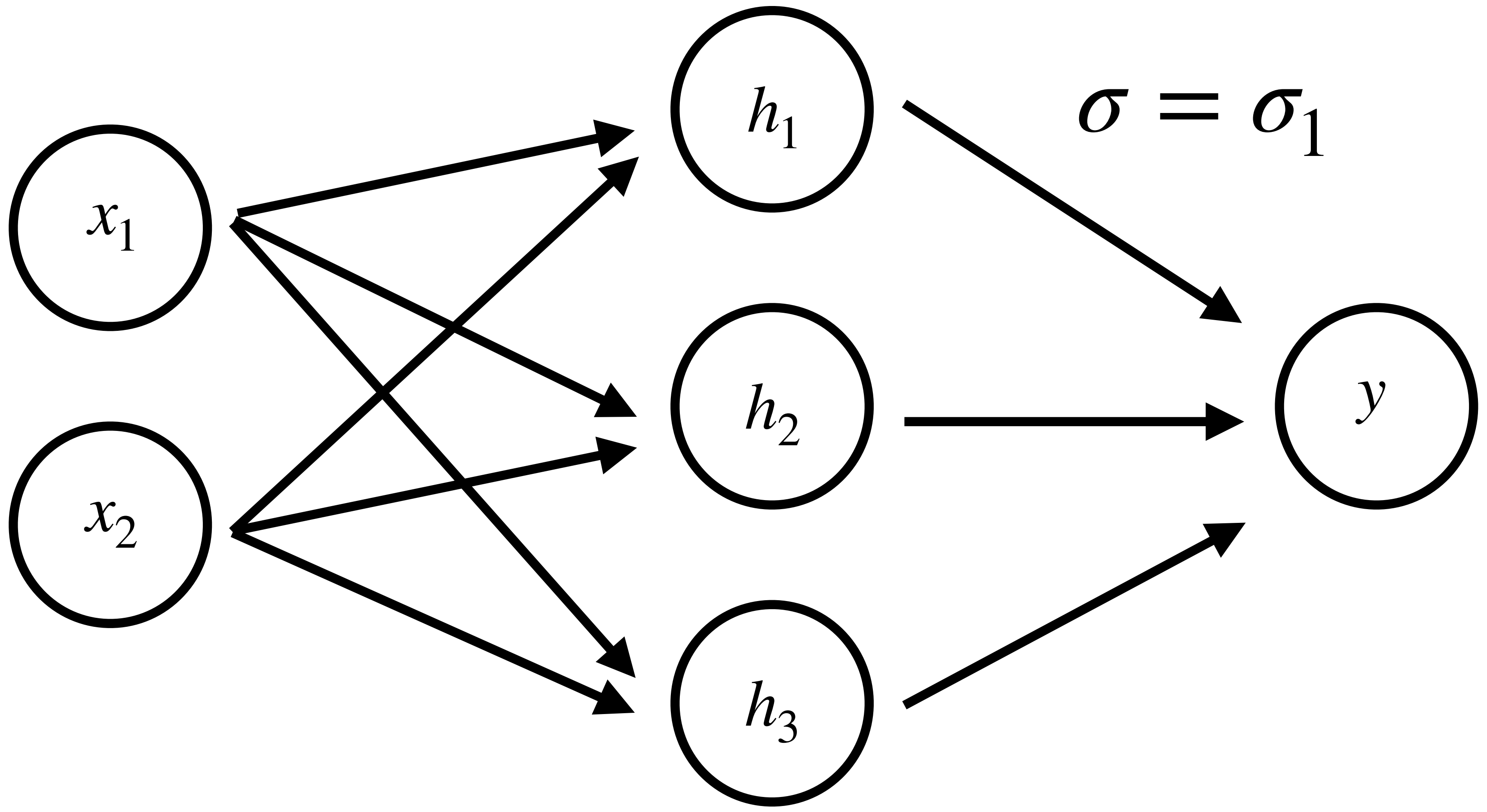
- Values aligned with the labels → ?
- ‘Usable’ gradient → ?

Sigmoid function

$$\sigma_a(x) = \frac{1}{1 + e^{-ax}}$$



$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$



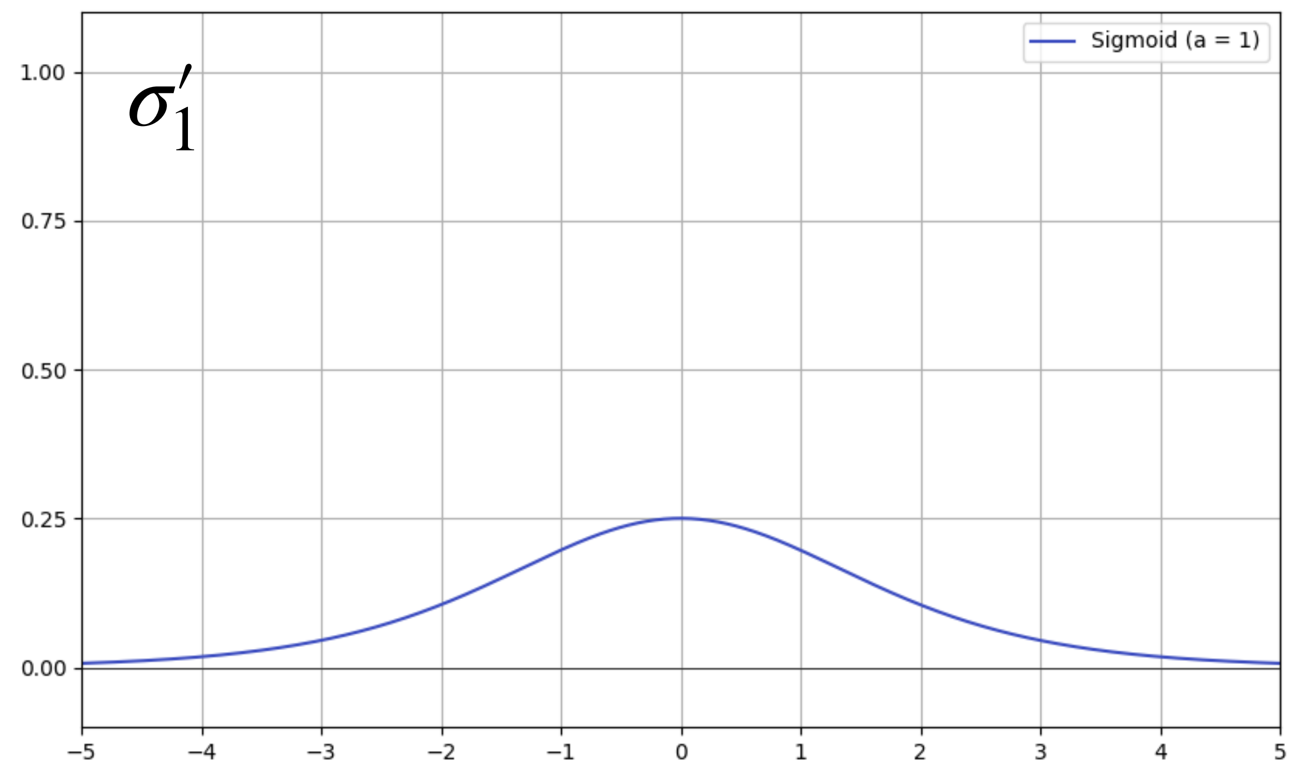
$$\hat{y} = \begin{pmatrix} 0.1 \\ 0.9 \\ 0.8 \\ 0.3 \end{pmatrix} ? \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Output layer

Last layer dimension compatible with output

Sigmoid function

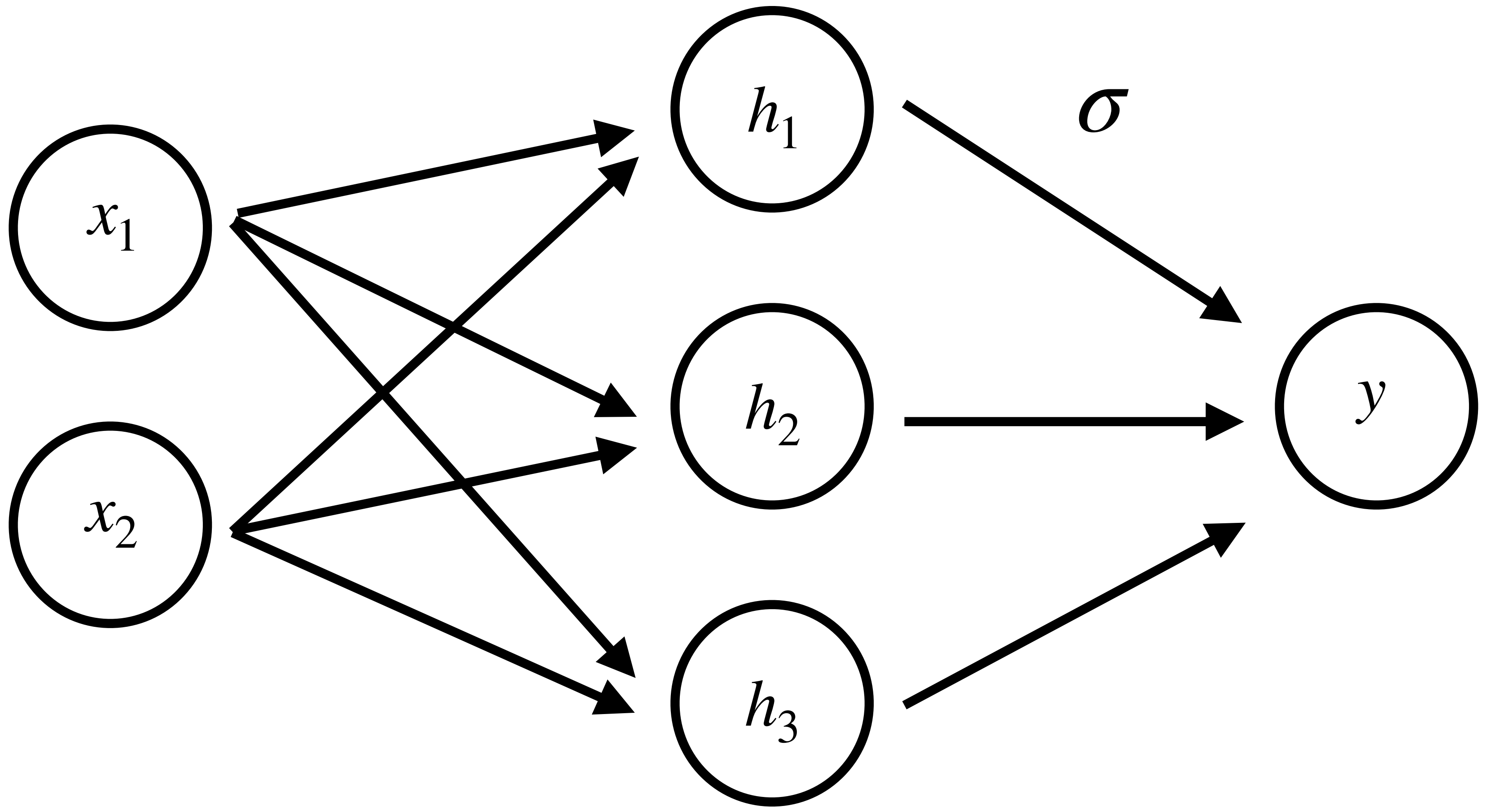
$$\sigma'_a = a\sigma_a(1 - \sigma_a)$$



Last layer activation function

- Values aligned with the labels ⟶ Ok, it maps probabilities in [0,1]
- ‘Usable’ gradient ⟶ yes, continuous derivate

$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$



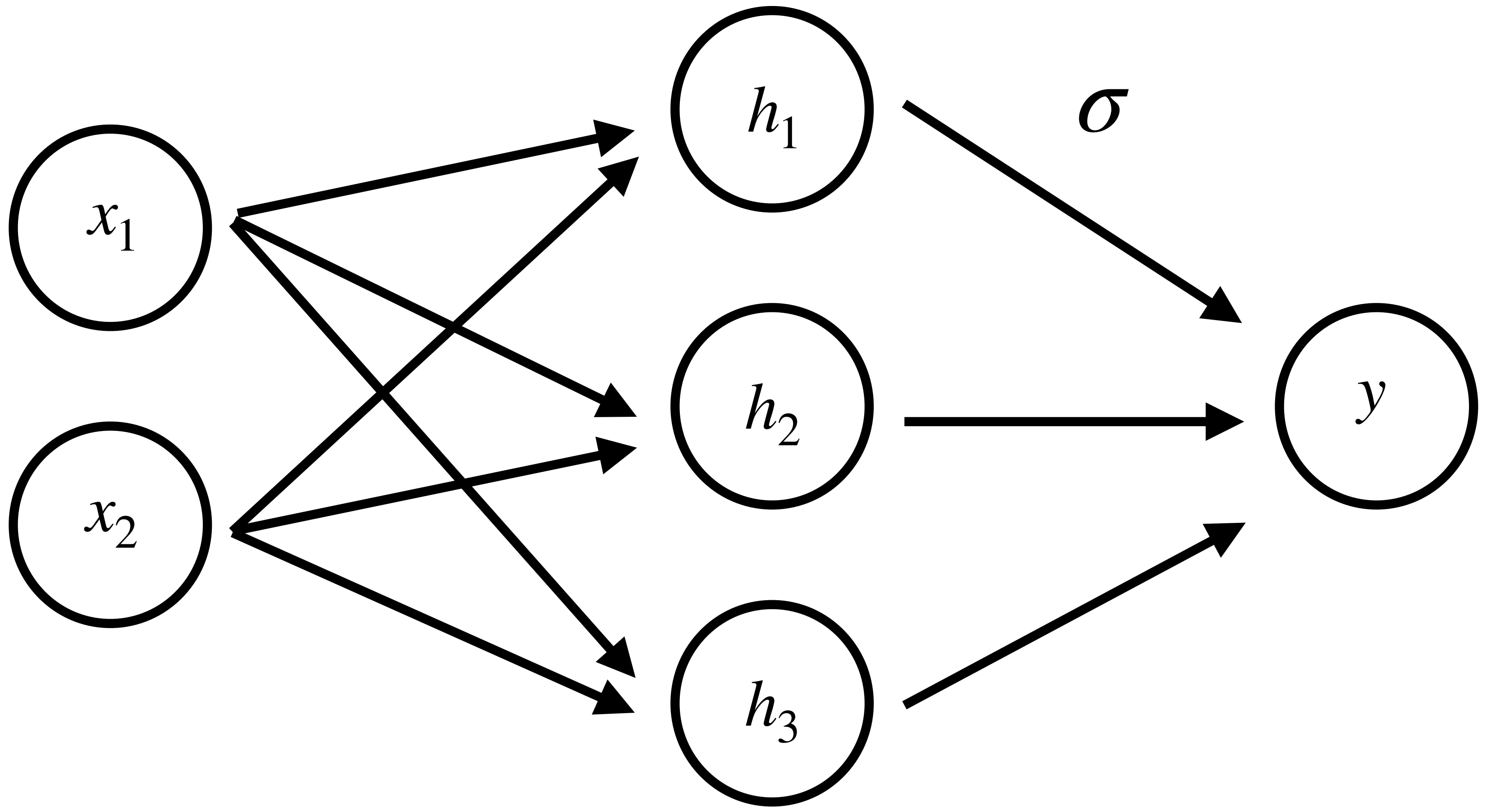
$$\hat{y} = \begin{pmatrix} 0.1 \\ 0.9 \\ 0.8 \\ 0.3 \end{pmatrix} ? \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Training steps

Iterate until convergence

1. Forward pass: for batch x compute output $\hat{y} = f(x;\theta)$
2. **Evaluate: compare the $\hat{y}$ with the class label y**
3. Backward pass: update the parameters θ

$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$



$$\hat{y} = \begin{pmatrix} 0.1 \\ 0.9 \\ 0.8 \\ 0.3 \end{pmatrix} ? \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

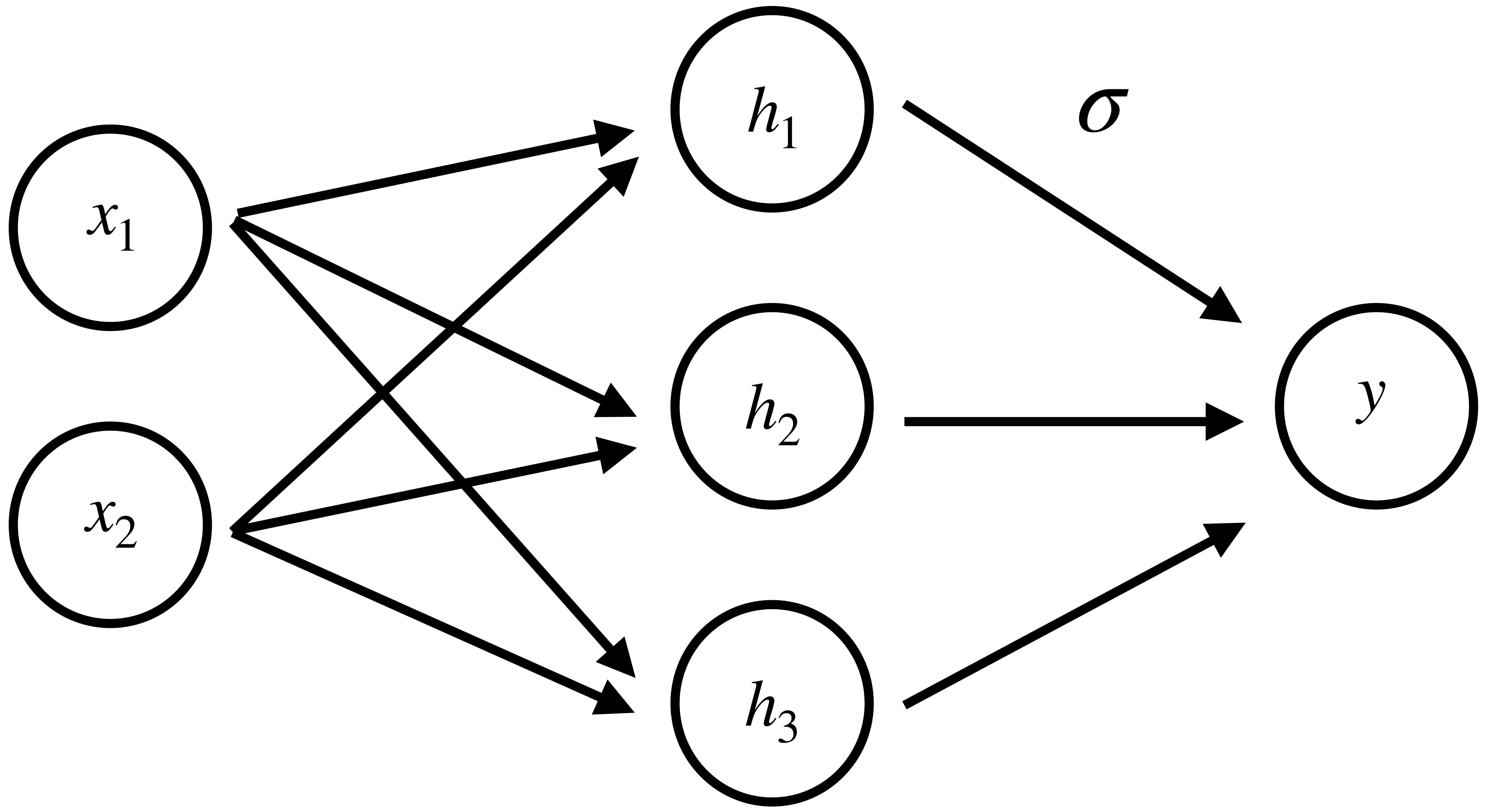
How to compare?

Training steps

Iterate until convergence

1. Forward pass: for batch x compute output $\hat{y} = f(x; \theta)$
2. **Evaluate: compare the $\hat{y}$ with the class label y**
3. Backward pass: update the parameters θ

$$x = \begin{pmatrix} 0.5 & 0.5 \\ 0.2 & 0.2 \\ 0.8 & 0.8 \\ 0.8 & 0.1 \end{pmatrix}$$



$$\hat{y} = \begin{pmatrix} 0.1 \\ 0.9 \\ 0.8 \\ 0.3 \end{pmatrix} ? \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

How to compare?

A scalar objective/cost/loss/error function $L(\hat{y}, y; \theta)$.

Summary

Topics

1. Linear perceptrons
2. Deep feedforward networks
3. Network’s representation
4. Training loop

Reading material

- *Understanding Deep Learning* - Chapter 3, 4