

# Neural Networks: Basic Structure, Representation Power

Machine Learning Course - CS-433

Nov 8, 2022

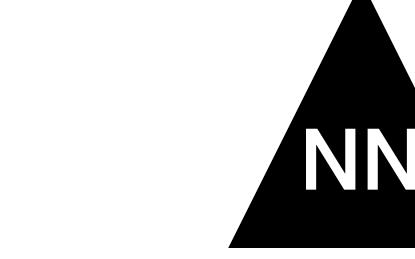
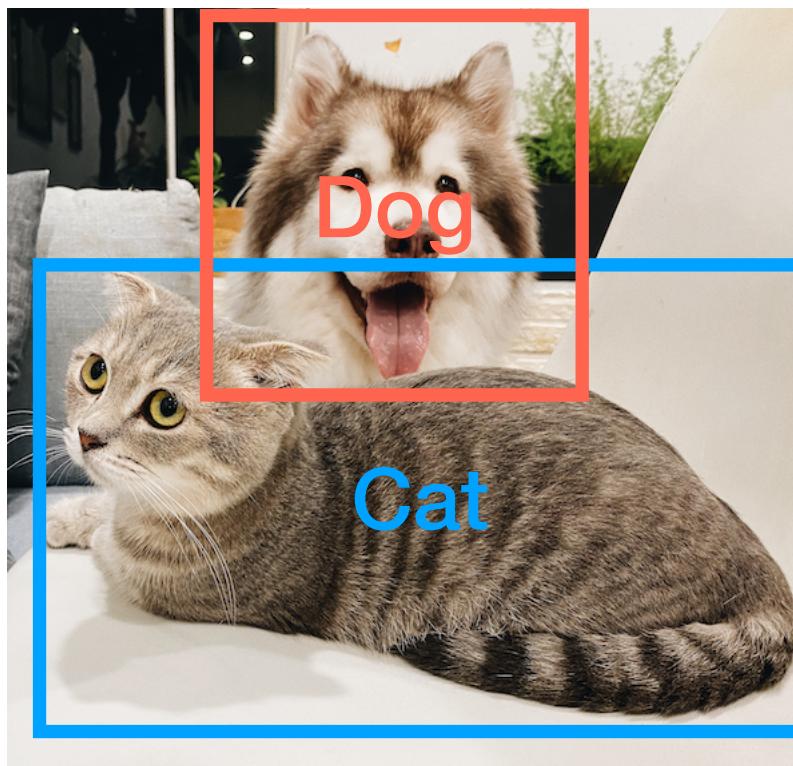
Nicolas Flammarion



# Deep Learning: ML Breakthrough

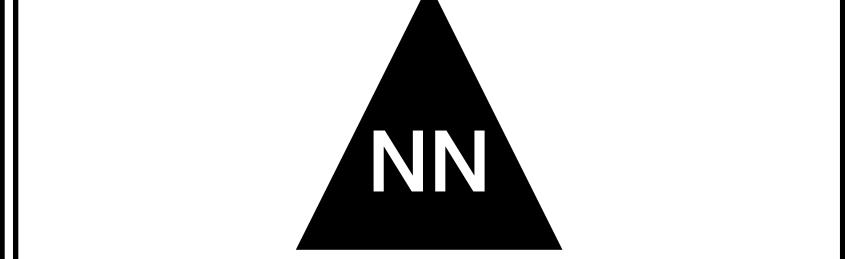
State of the art performance across a broad range of useful tasks

## Object Detection



## Translation

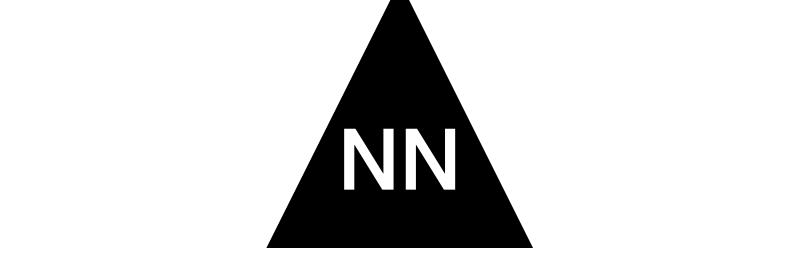
elle est très intelligente



she is very smart

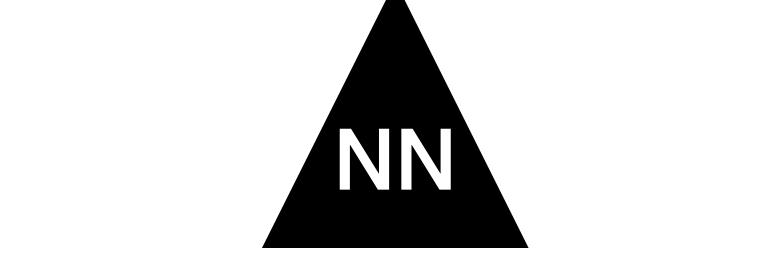
## Speech to Text

hey Siri next song



student studying machine learning

## Image Generation



student studying machine learning

## Playing Chess



# Why Neural Networks?

## Supervised learning:

- Have training data  $S_{\text{train}} = \{x_n, y_n\}_{n=1}^N \in \mathcal{X} \times \mathcal{Y}$
- Goal: Find a mapping  $\mathcal{X} \rightarrow \mathcal{Y}$  that can predict  $y$  for a new  $x$

## Classical Machine Learning:

- Relatively simple models on top of **features handcrafted by domain experts**
- Only works well when used with **good features**

## Deep Learning:

- Large neural networks that **learn features directly from the data**
- Can be viewed as a complicated feature extractor + linear prediction
- **Requires large amounts of data and compute to train**
- Quality often continues to improve with more data and larger models

# The Basic Structure of Neural Networks

# Neurons and Logistic Regression

Recap - Logistic Regression:

$$f_{LR}(x) = p(1 | x) = \sigma(x^\top w + b) = \sigma\left(\sum_{i=1}^d x_i w_i + b\right)$$

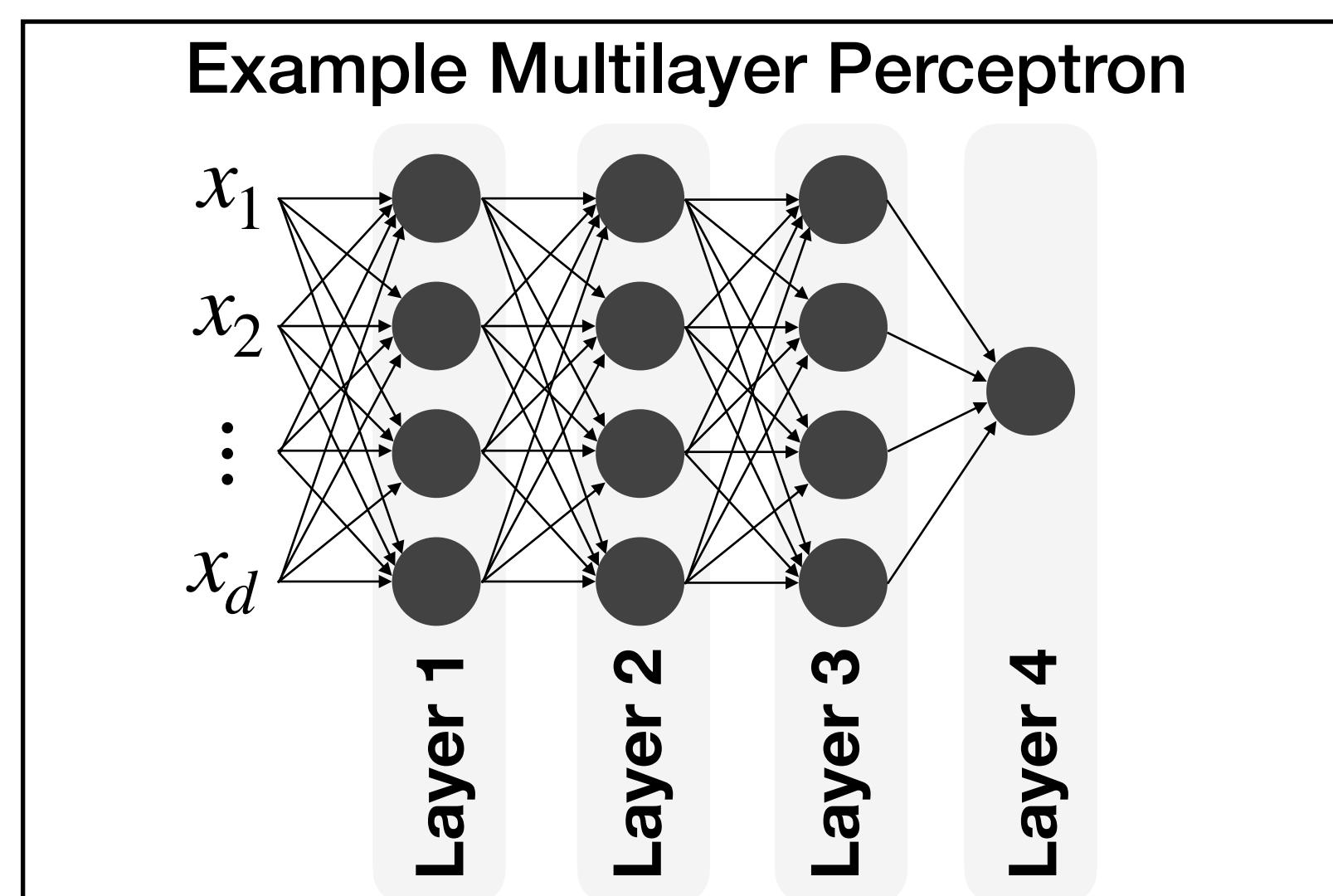
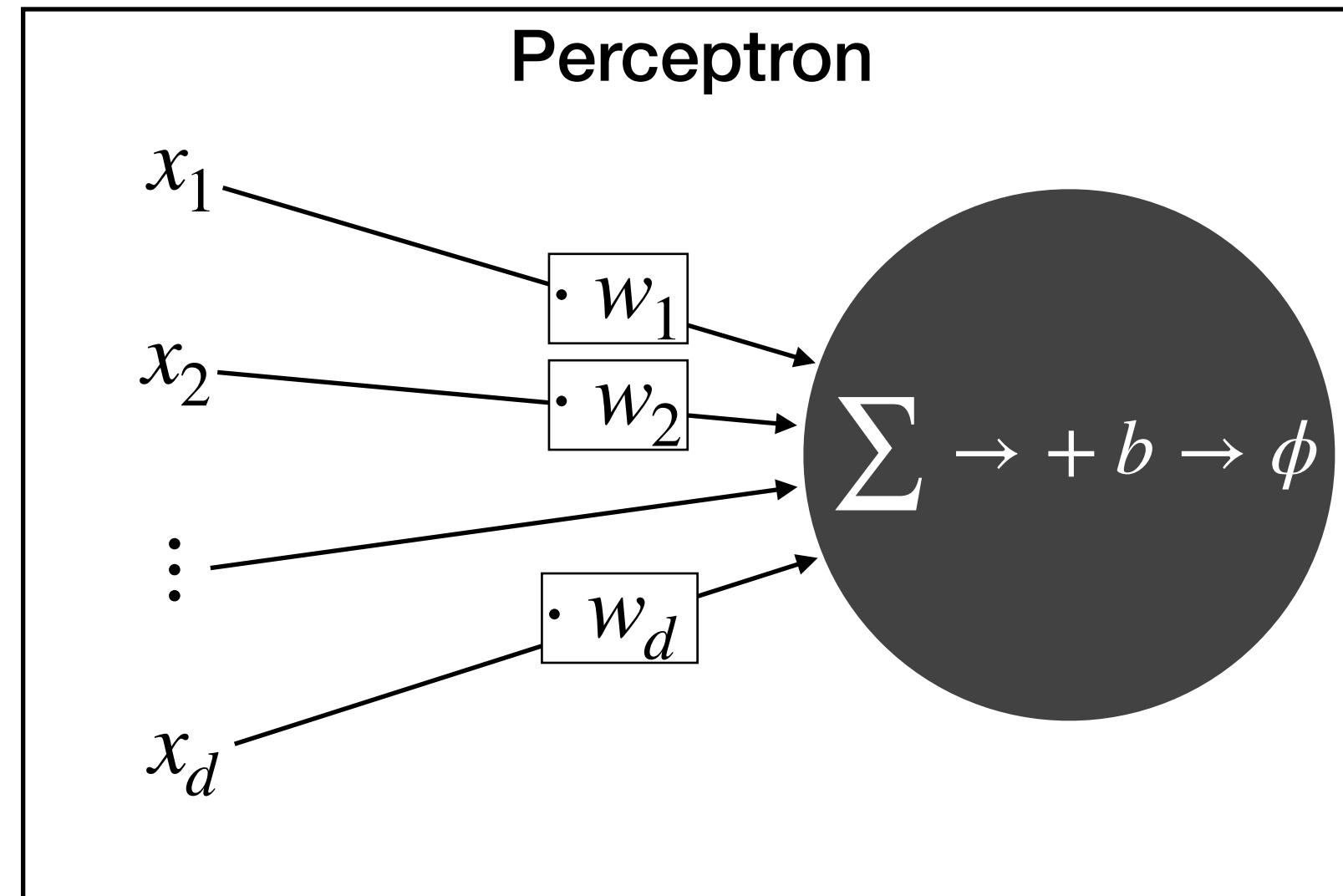
$$\sigma(z) = (1 + \exp(-z))^{-1}$$

Perceptron:

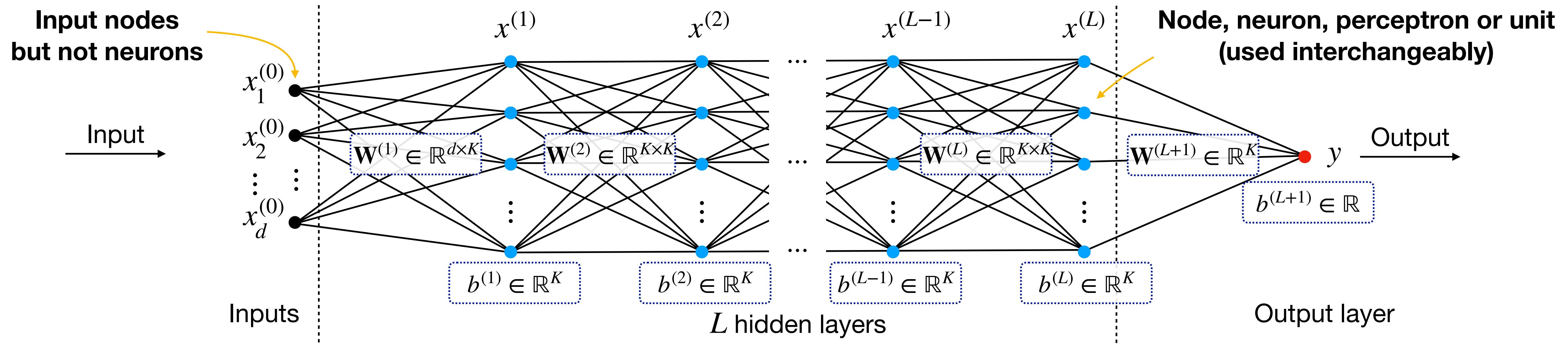
- A simple model of biological neurons
- Activation function  $\phi$
- Equivalent to  $f_{LR}$  when  $\phi = \sigma$

Multilayer Perceptron (MLP):

- Arrange perceptrons into layers
- Neurons in later layers receive inputs from prior layer
- Also known as a **fully connected neural network**



# Fully Connected Neural Networks



Assume  $L$  hidden layers with  $K$  neurons each + output layer with single node

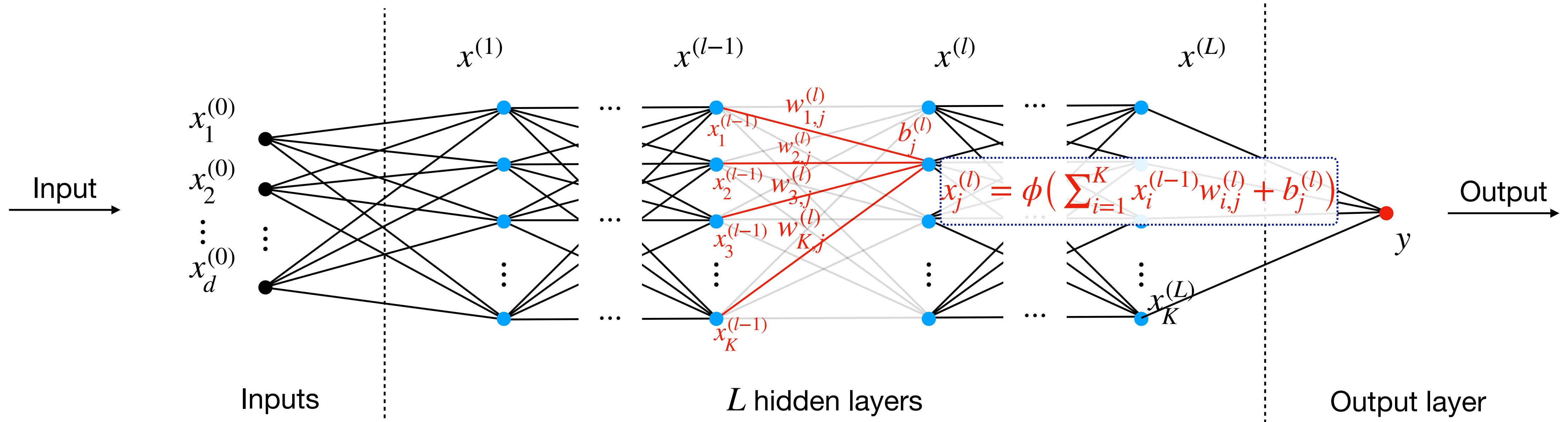
Outputs of hidden layer  $l$  given by vector:  $x^{(l)} = f^{(l)}(x^{(l-1)}) := \phi((\mathbf{W}^{(l)})^\top x^{(l-1)} + b^{(l)})$

Elementwise

**Learnable Parameters:** Weight matrices  $\mathbf{W}^{(l)}$  and bias vectors  $b^{(l)}$  for  $1 \leq l \leq L + 1$

- Each column of  $\mathbf{W}^{(l)}$  corresponds to the weights of one perceptron

# Single Neuron View



$$x_j^{(l)} = \phi\left(\sum_{i=1}^K x_i^{(l-1)} w_{i,j}^{(l)} + b_j^{(l)}\right)$$

Important:  $\phi$  is non-linear  
otherwise we can only  
represent linear functions

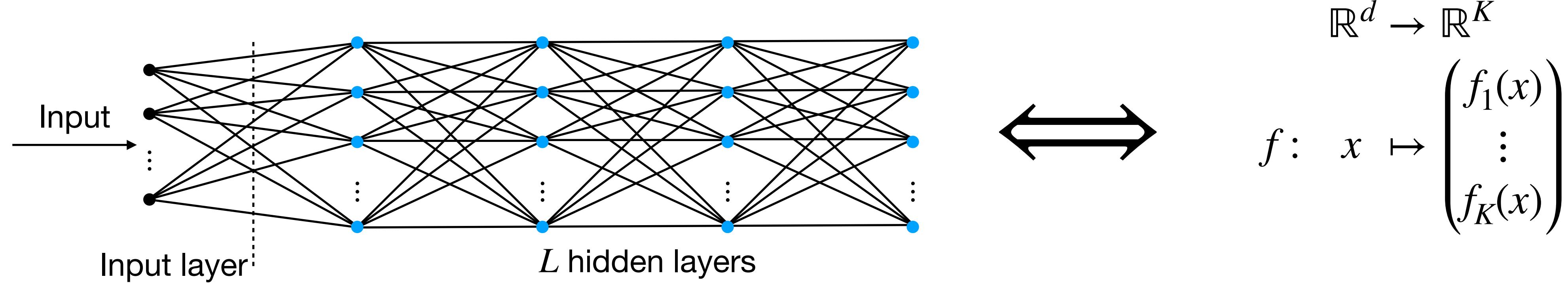
weight of the edge going from node  $i$   
in layer  $l - 1$  to node  $j$  in layer  $l$

bias term associated with  
node  $j$  in layer  $l$

# NNs extract suitable features from the input

A NN can be decomposed into a feature extractor and the output layer.

Feature extractor from  $\mathbb{R}^d$  to  $\mathbb{R}^K$ :

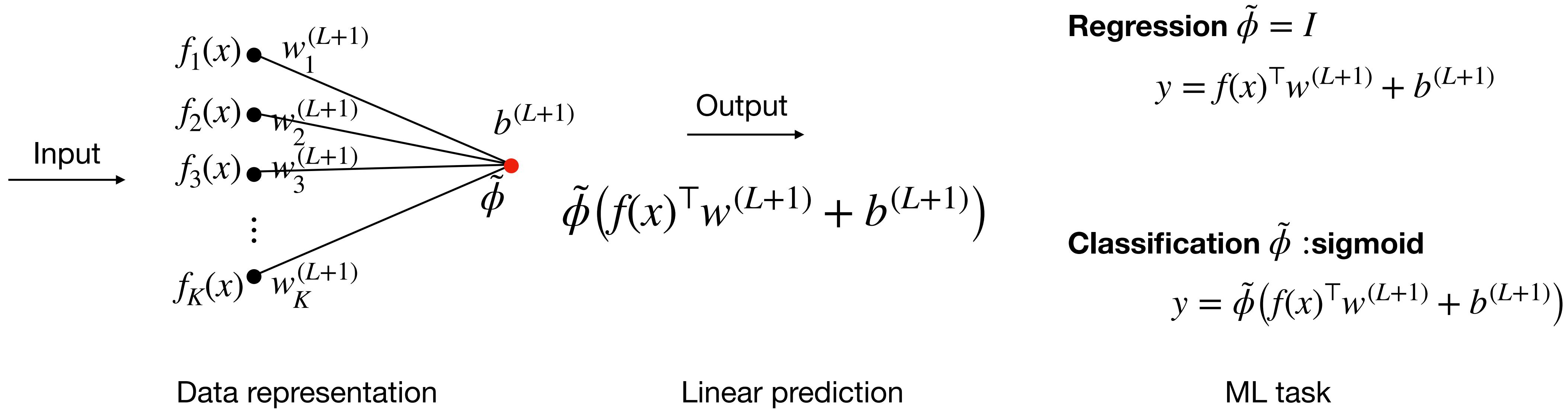


This function is defined by

- The biases  $\{b^{(l)}\}_{l \in [L]}$  and weights  $\{\mathbf{W}^{(l)}\}_{l \in [L]}$  so we learn  
    →  $O(dK + K^2L) \approx O(K^2L)$  parameters
- The activation function  $\phi$  we pick

In practice: both  $L$  and  $K$  are large - overparametrized NNs

# The last layer performs the desired ML task



A suitable representation of the data in hands, the last layer only performs a linear regression or classification step

# **Representational Power of NNs**

# Three theoretical questions in Deep Learning

- **Expressive power** of NNs: Why are the functions we are interested in so **well approximated** by NNs?
- **Success of naive optimization**: Why does **gradient descent** lead to a good local minimum?
- **Generalization miracle**: Why is there **no overfitting** with so many parameters?

# $L_2$ Approximation: Barron's result

Let  $f: \mathbb{R}^d \rightarrow \mathbb{R}$  and define  $\hat{f}(\omega) = \int_{\mathbb{R}^d} f(x) e^{-i\omega^\top x} dx$  its Fourier transform

Assumption:  $\int_{\mathbb{R}^d} |\omega| |\hat{f}(\omega)| d\omega \leq C$  (smoothness assumption)

Claim: For all  $n \geq 1$  and  $r > 0$ , it exists a function  $f_n$  of the form

$$f_n(x) = \sum_{j=1}^n c_j \phi(x^\top w_j + b_j) + c_0$$

so that

$$\int_{|x| \leq r} (f(x) - f_n(x))^2 dx \leq \frac{(2Cr)^2}{n}$$

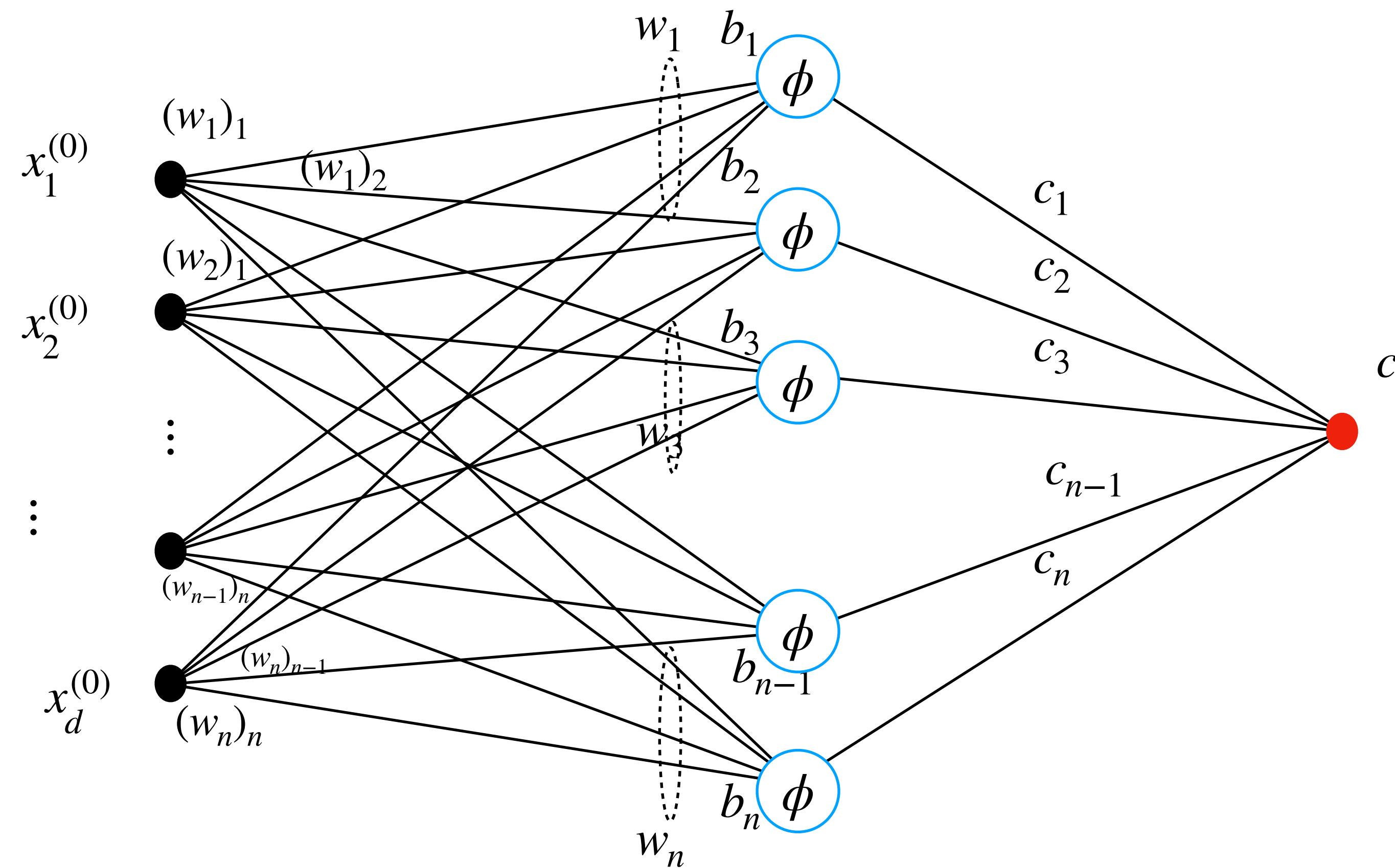
# All sufficiently smooth function can be approximated by a one-hidden-layer NN

$$\int_{|x| \leq r} (f(x) - f_n(x))^2 dx \leq \frac{(2Cr)^2}{n}$$

- The more neurons we allow, the smaller the error
- The smoother the function is (the smaller  $C$ ), the smaller the error
- The larger the domain (the larger  $r$ ), the worse the error
- Approximation in average (in  $\ell_2$ -norm)
- For any “sigmoid-like” activation function

The function  $f_n$  is a one-hidden-layer NN with  $n$  nodes

$$f_n(x) = \sum_{j=1}^n c_j \phi(x^\top w_j + b_j) + c_0 = c^\top \phi(W^\top x + b) + c_0$$

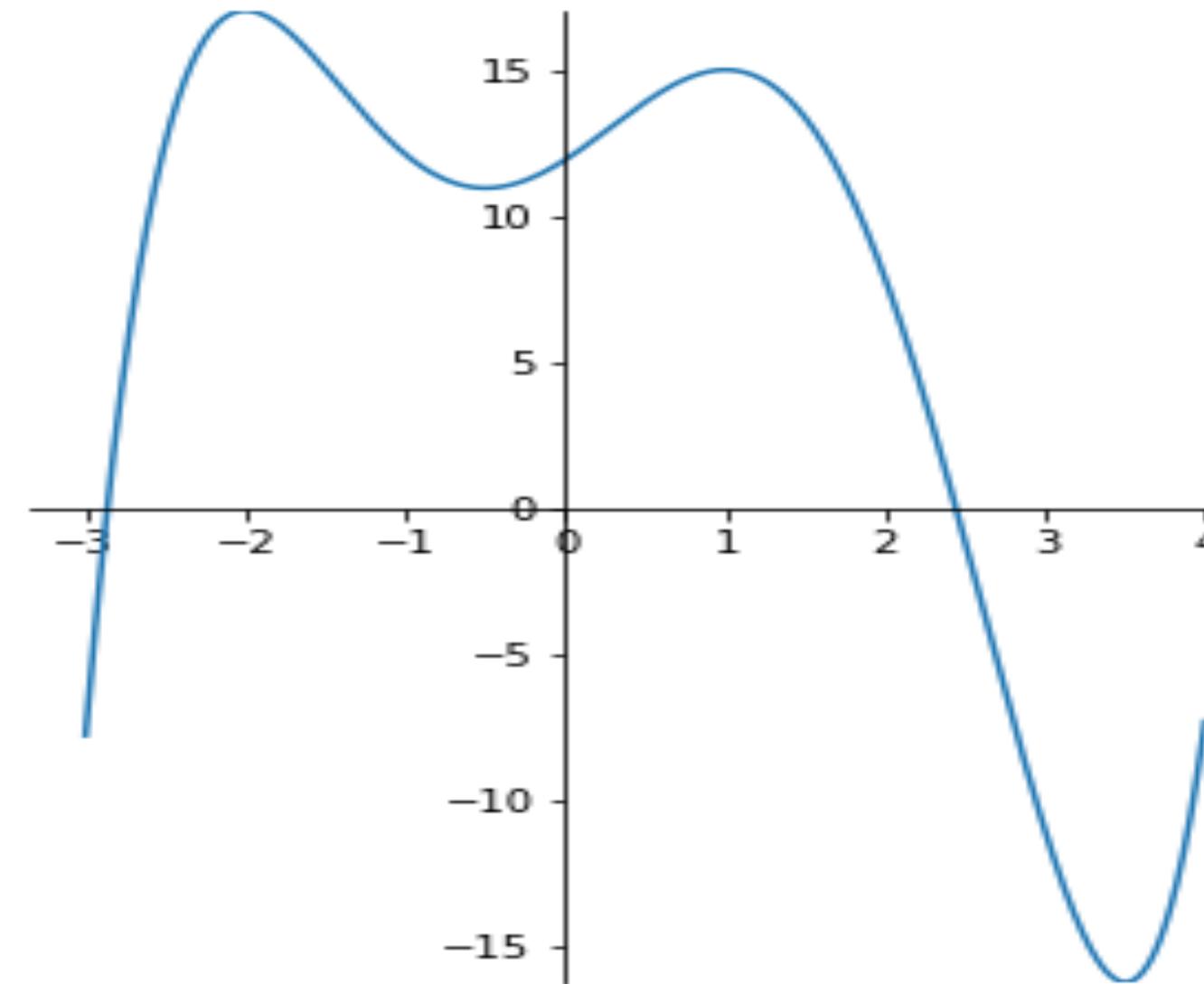


# $L_1$ Approximation: Proof by picture

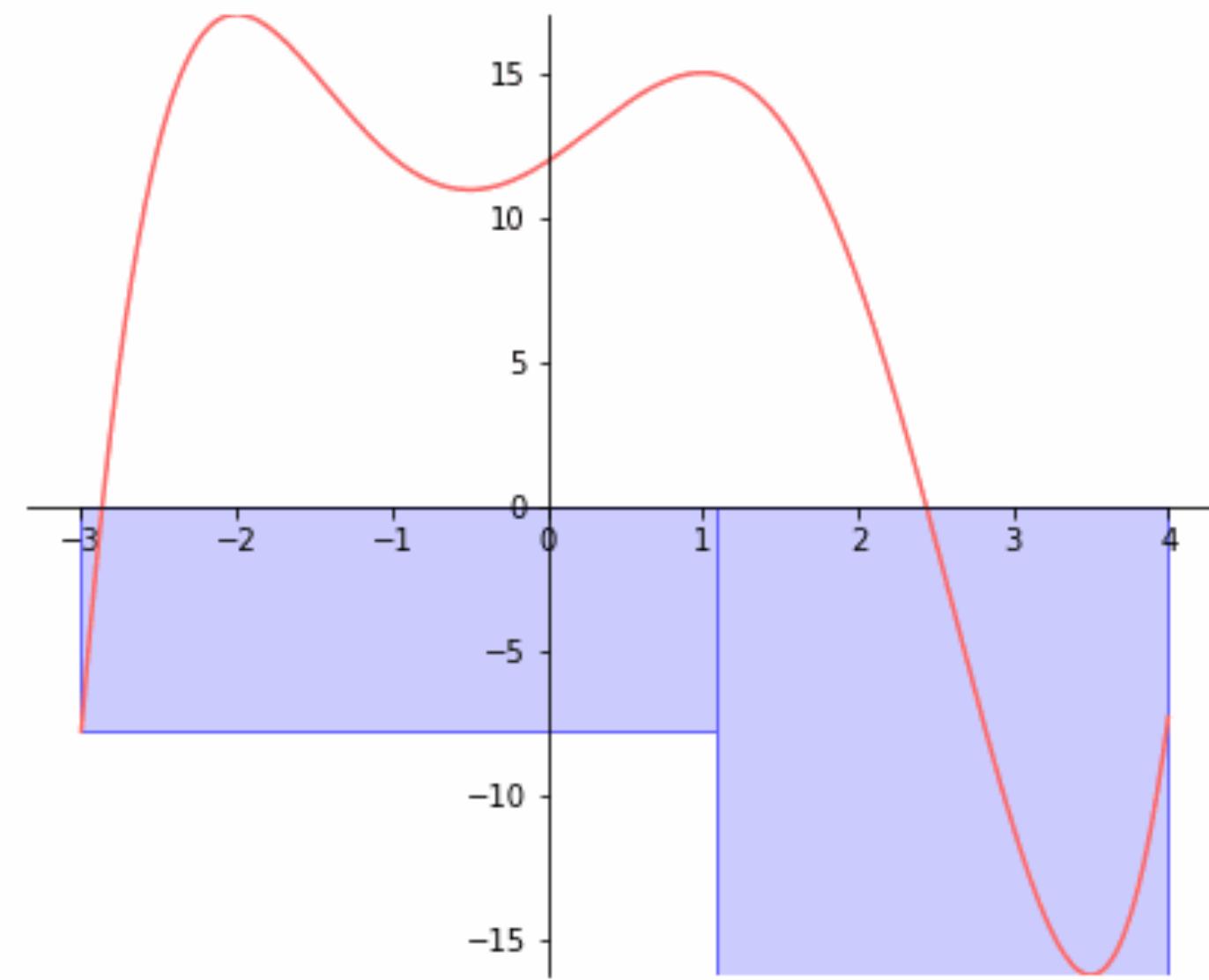
Simple and intuitive explanation of a slightly different result:

“A NN with sigmoid activation and at most two hidden layers can approximate well a smooth function in average, i.e, in  $\ell_1$ -norm”

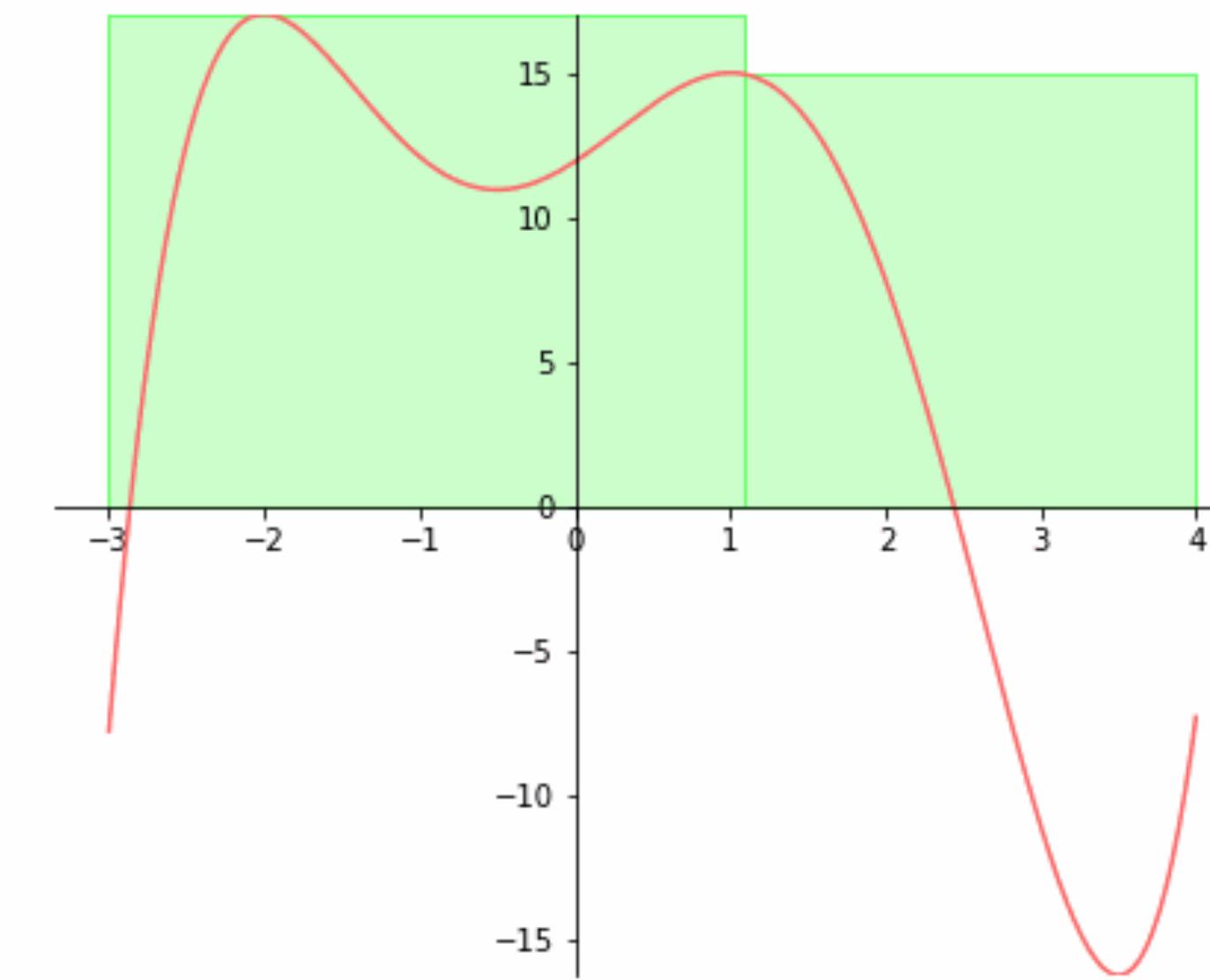
Consider a function  $f: \mathbb{R} \rightarrow \mathbb{R}$  on a bounded domain



# Approximation of the function by a sum of rectangles



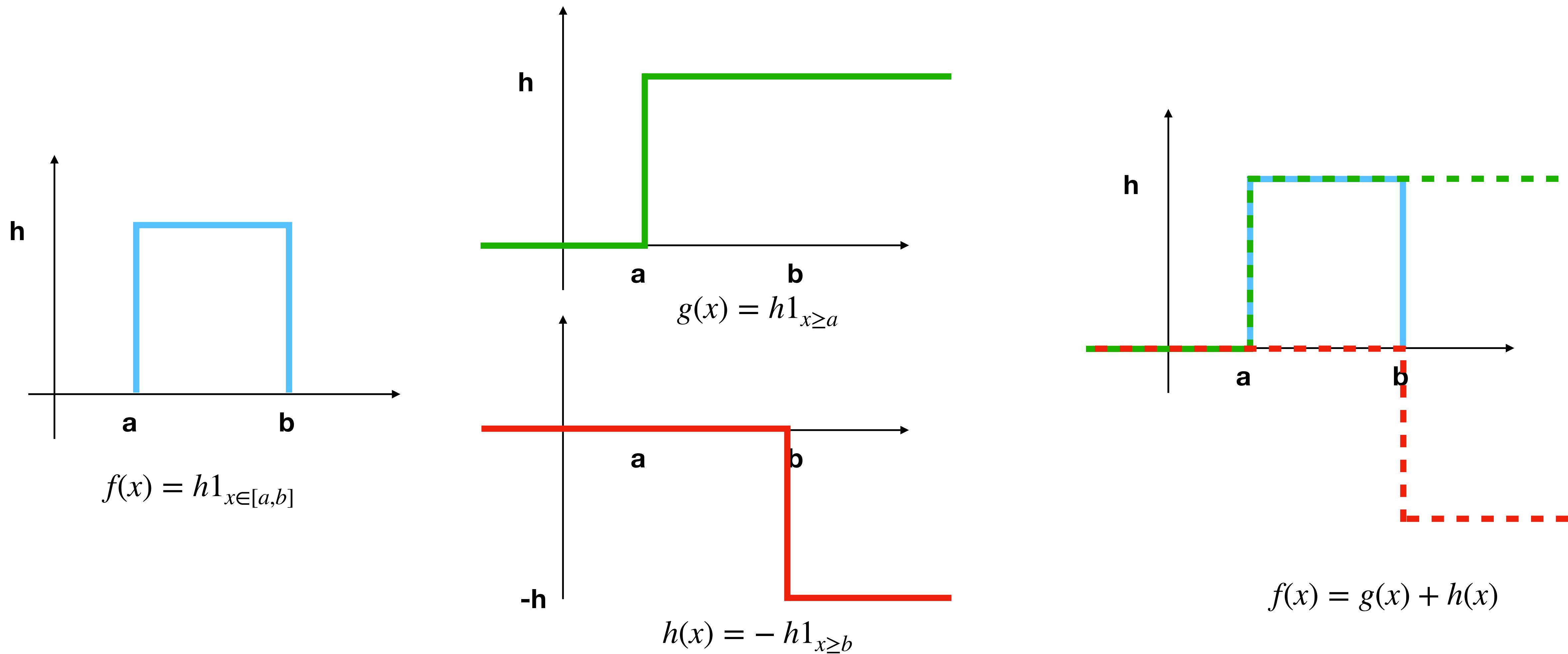
From below



From above

The function is Riemann integrable - can be approximated arbitrarily closely by “lower” and “upper” sums of rectangle

# A rectangle is equal to the sum of two step functions



# Approximate a step function with a sigmoid

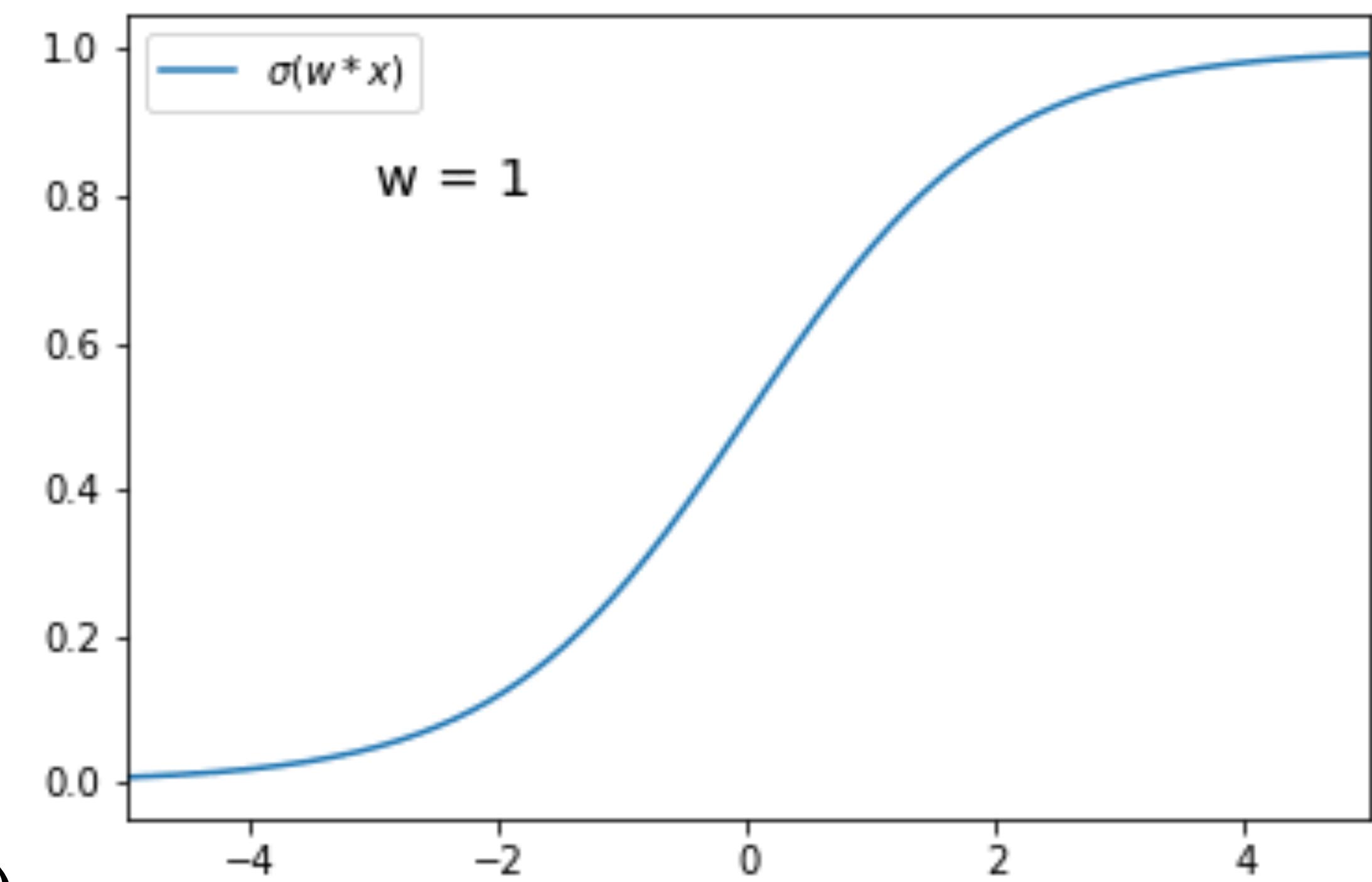
$$\tilde{\phi}(x) = \phi(w(x - b))$$

By setting:

- $b$ : where the transition happens
- $w$ : makes the transition steeper

Derivative:  $\tilde{\phi}'(b) = w/4$

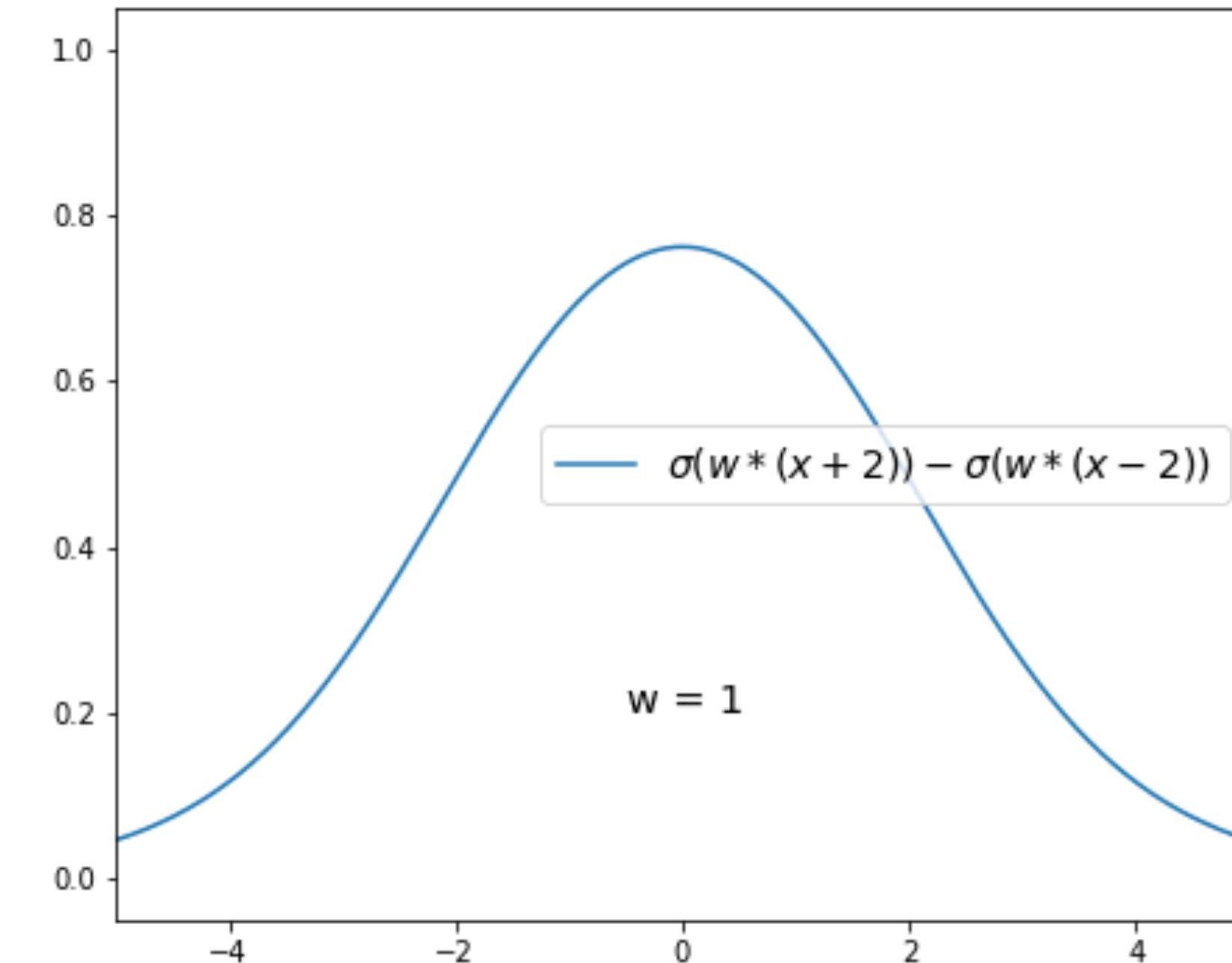
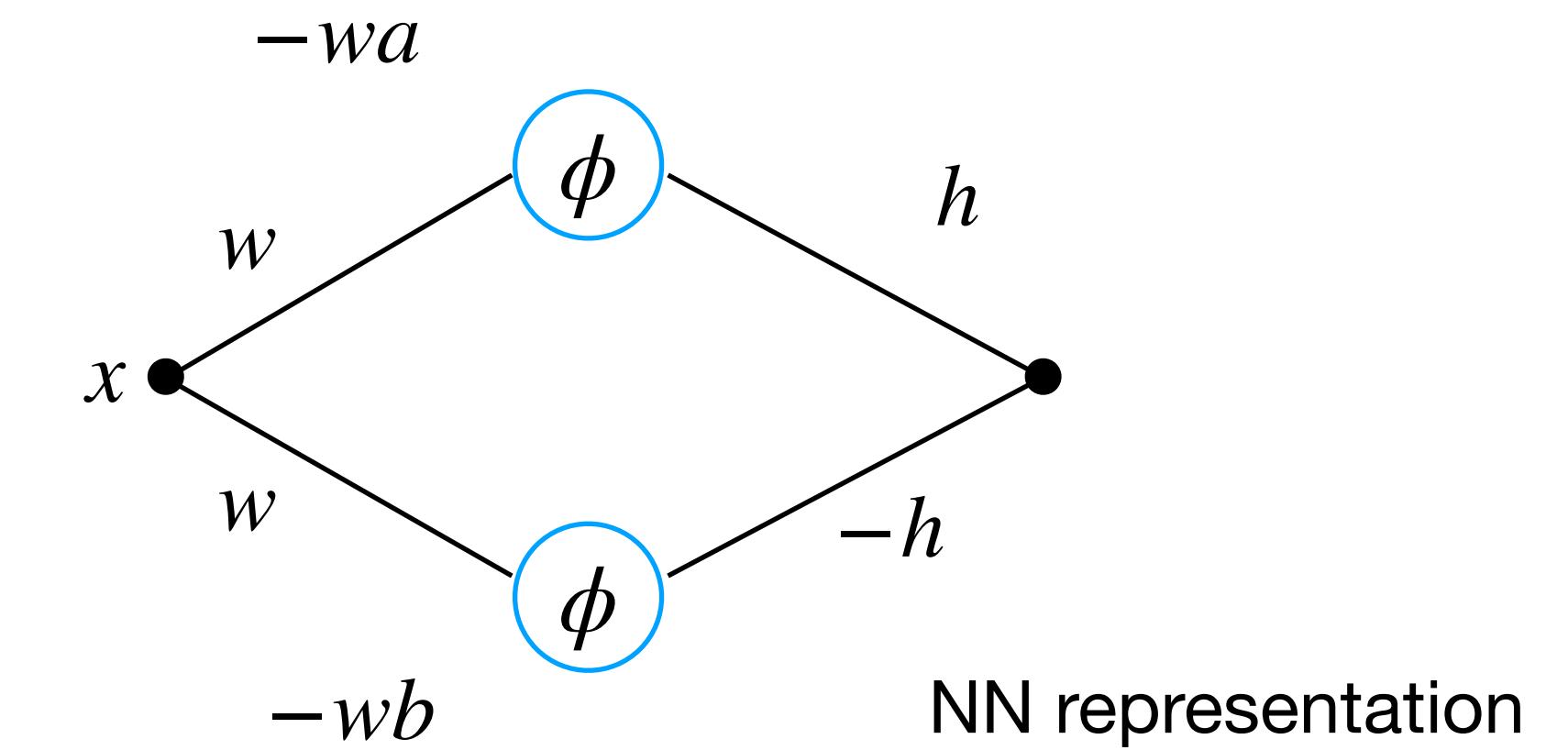
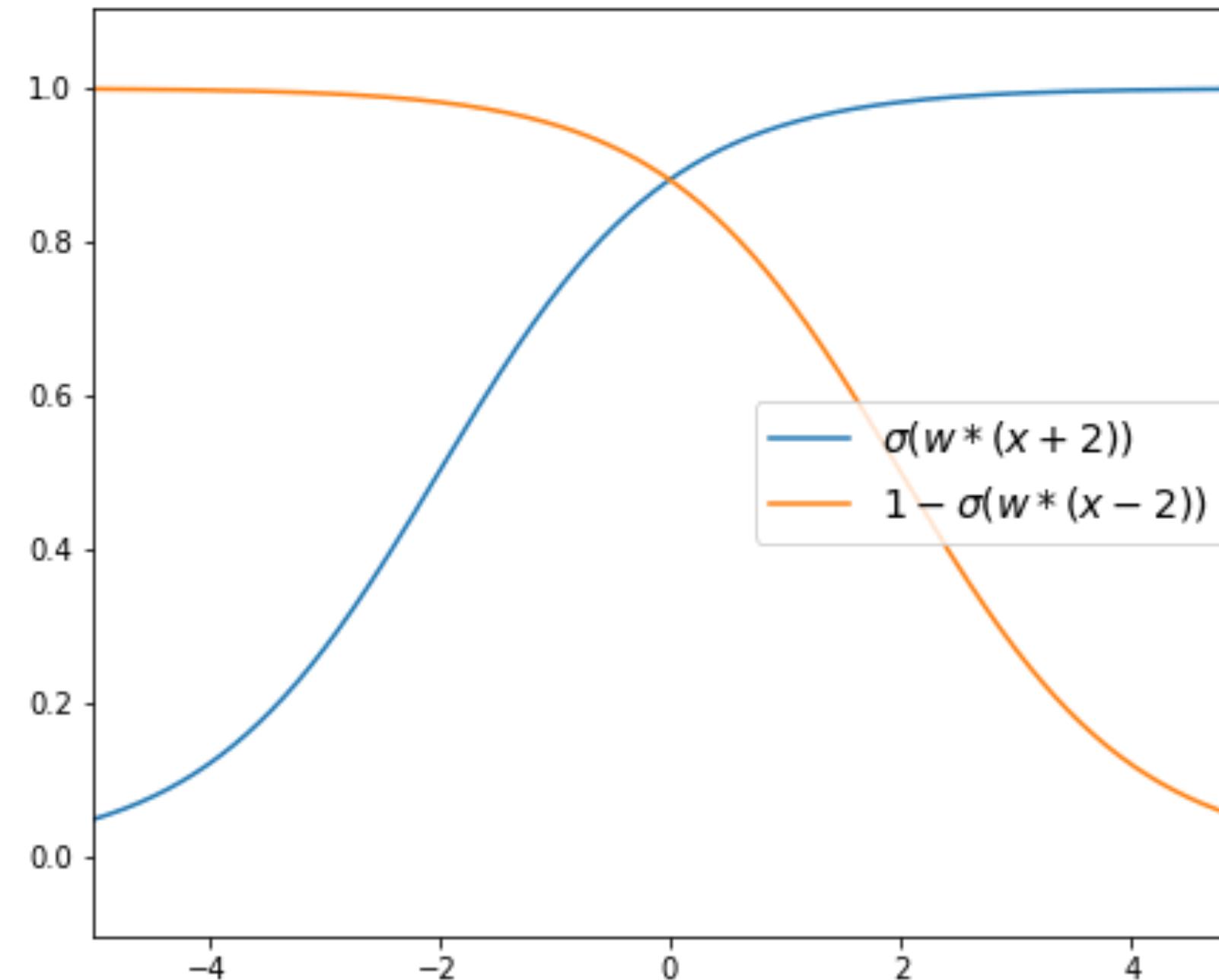
→ The width of the transition is  $O(4/w)$



# Approximation of the rectangle

$$h(\phi(w(x - a)) - \phi(w(x - b)))$$

$$\begin{aligned}a &= -2 \\b &= 2 \\h &= 1\end{aligned}$$

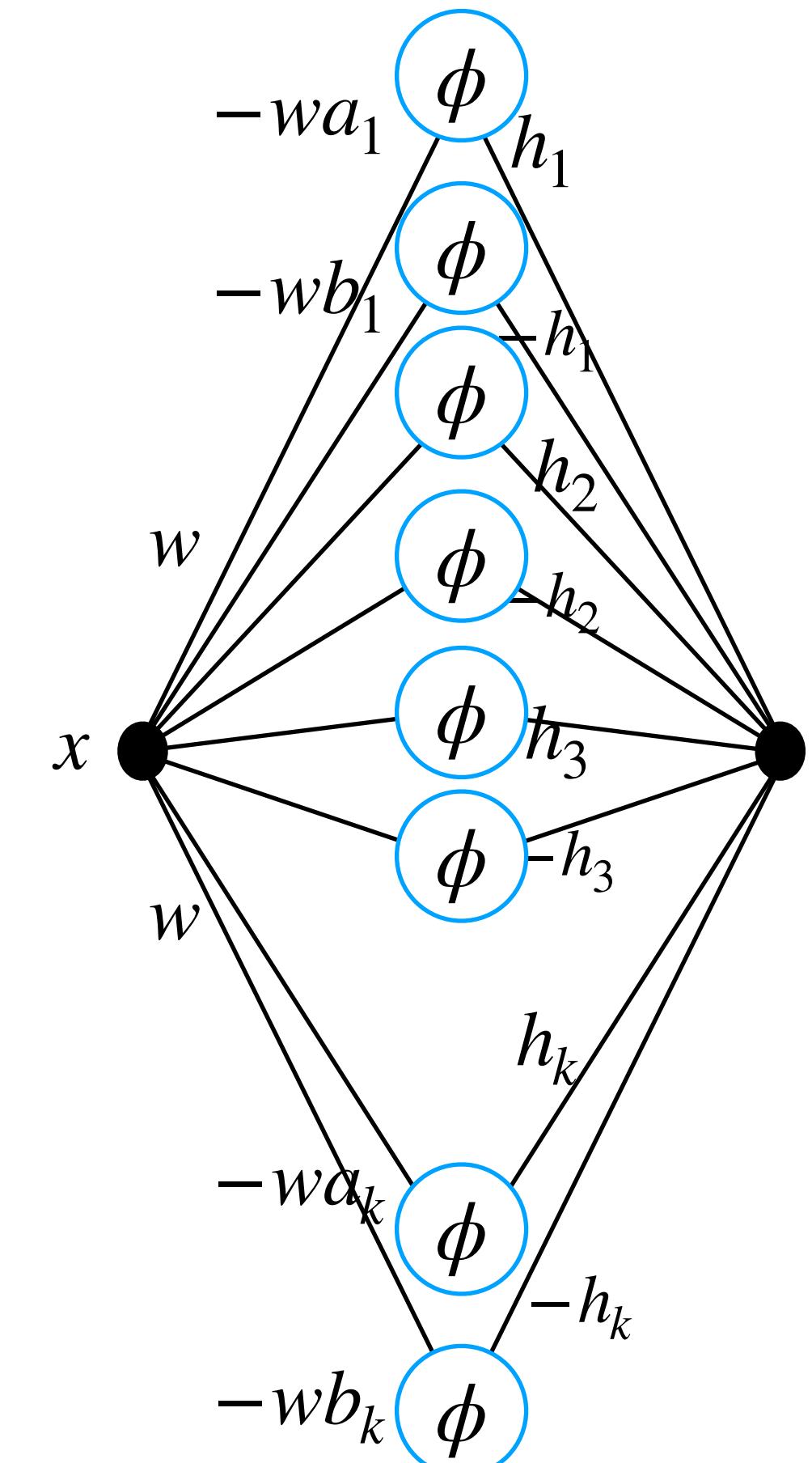


# Conclusion in the 1D case

1. Approximate the function in the Riemann sense by a sum of  $k$  rectangles
  2. Approximate each rectangle, by means of two nodes in the hidden layer of a NN
  3. Compute the sum (with appropriate sign) of all the hidden layers at the output node
    - NN with one hidden layer containing  $2k$  nodes for a Riemann sum with  $k$  rectangles

## Remarks:

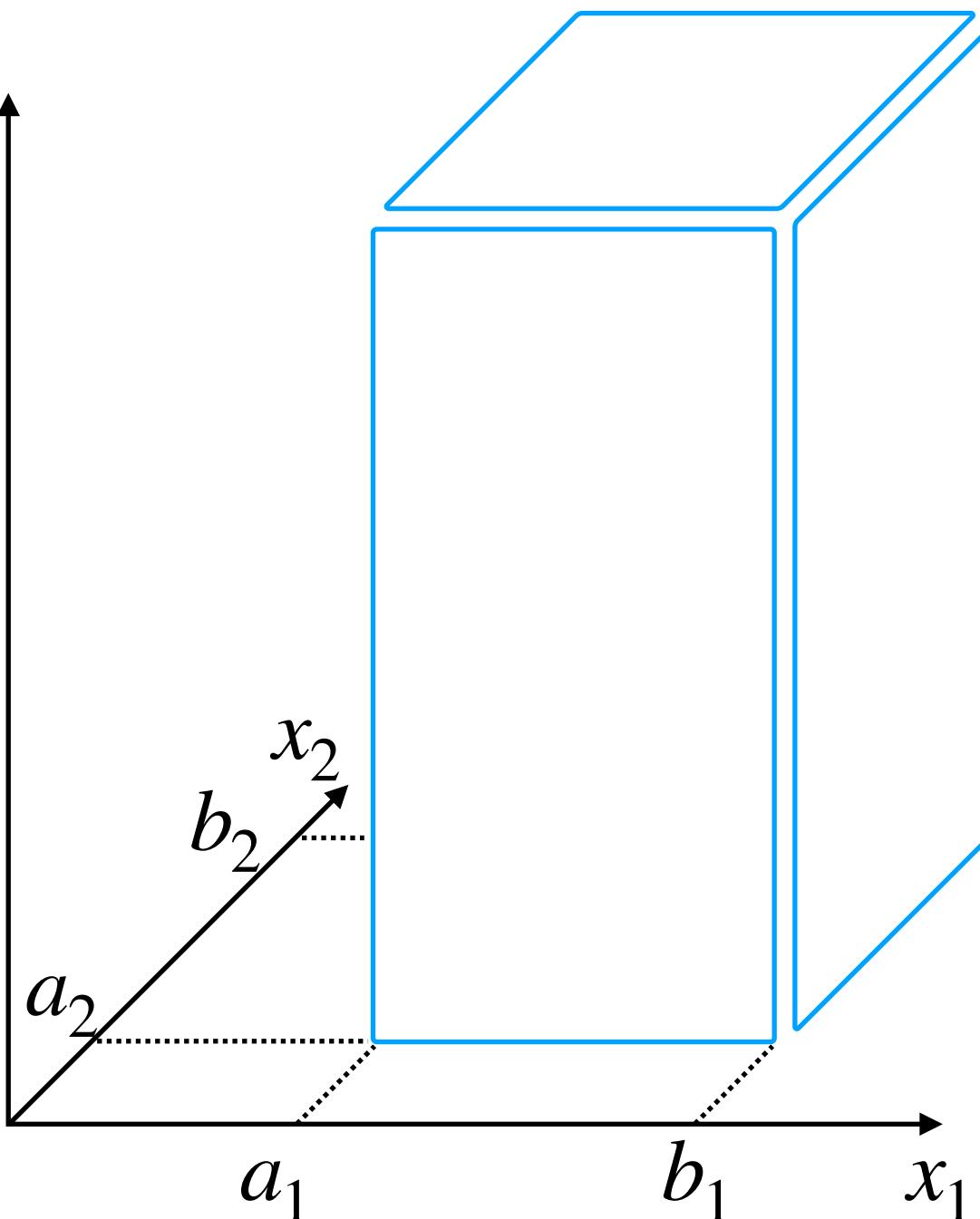
- Same intuition hold for any sigmoid like function
  - Only intuition, not quantitative
  - Need the weights  $w$  to be large



# Larger dimension: $d = 2$

Same idea:

- Approximate the function by 2D rectangles
- Approximate a 2D rectangle by sigmoids

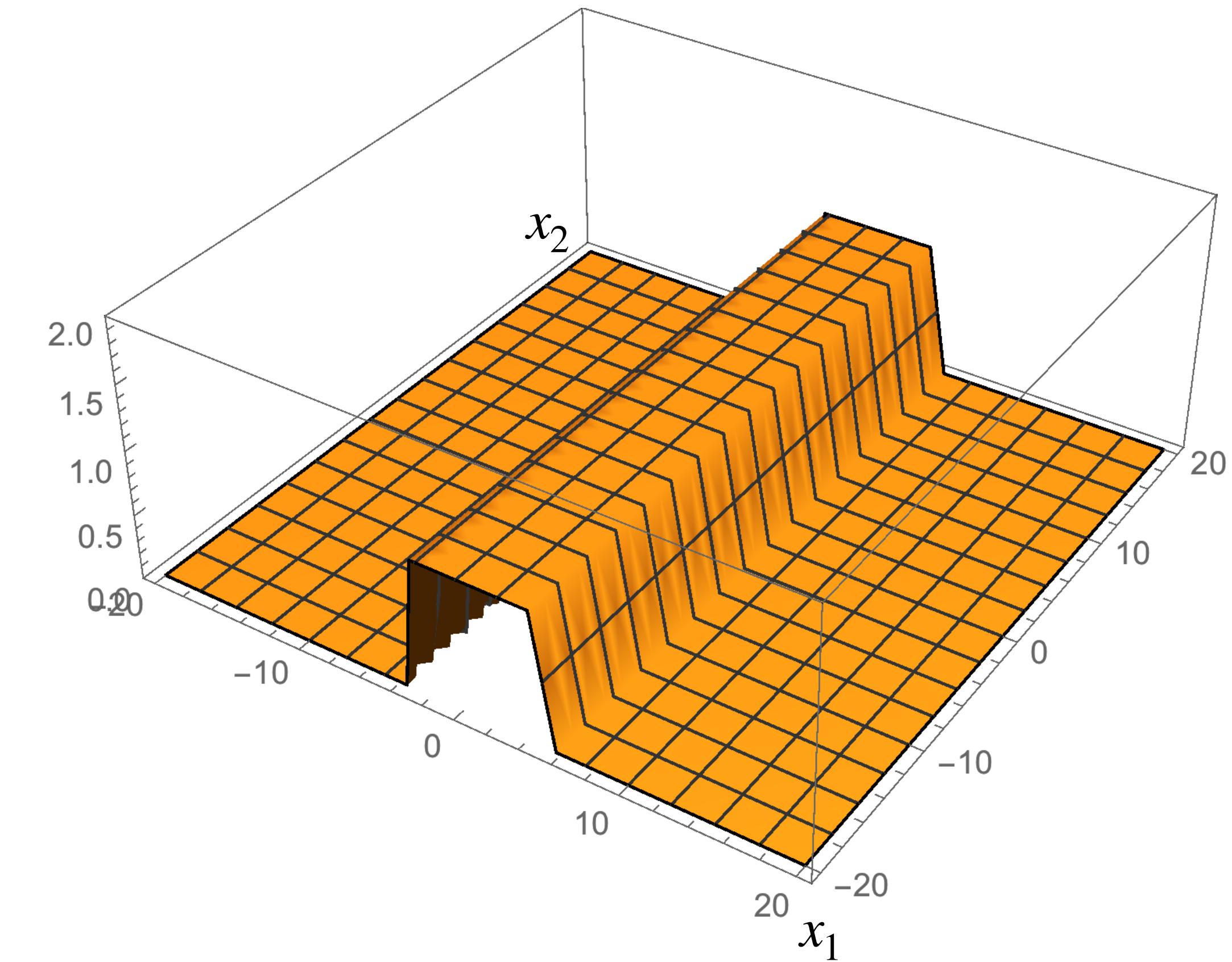
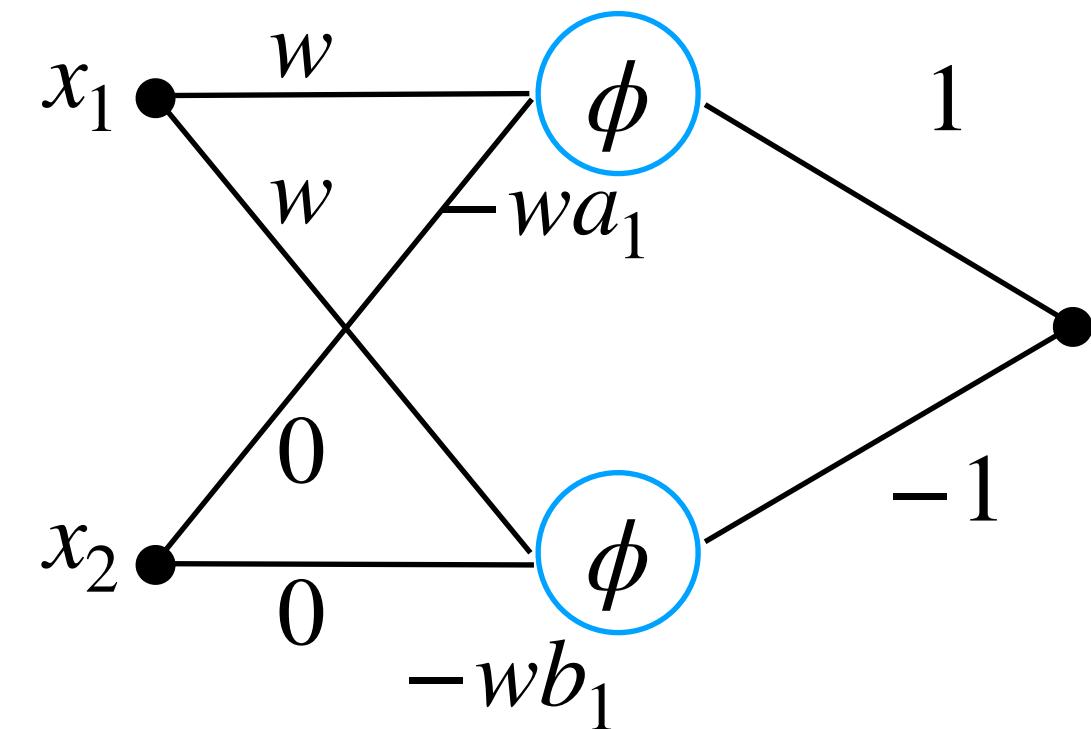


# Two sigmoids can approximate an infinite rectangle

$$(x_1, x_2) \mapsto \phi(w(x_1 - a_1)) - \phi(w(x_1 - b_1))$$

Rectangle:

- going from  $a_1$  to  $b_1$  in the  $x_1$  direction
- unbounded in the  $x_2$  direction

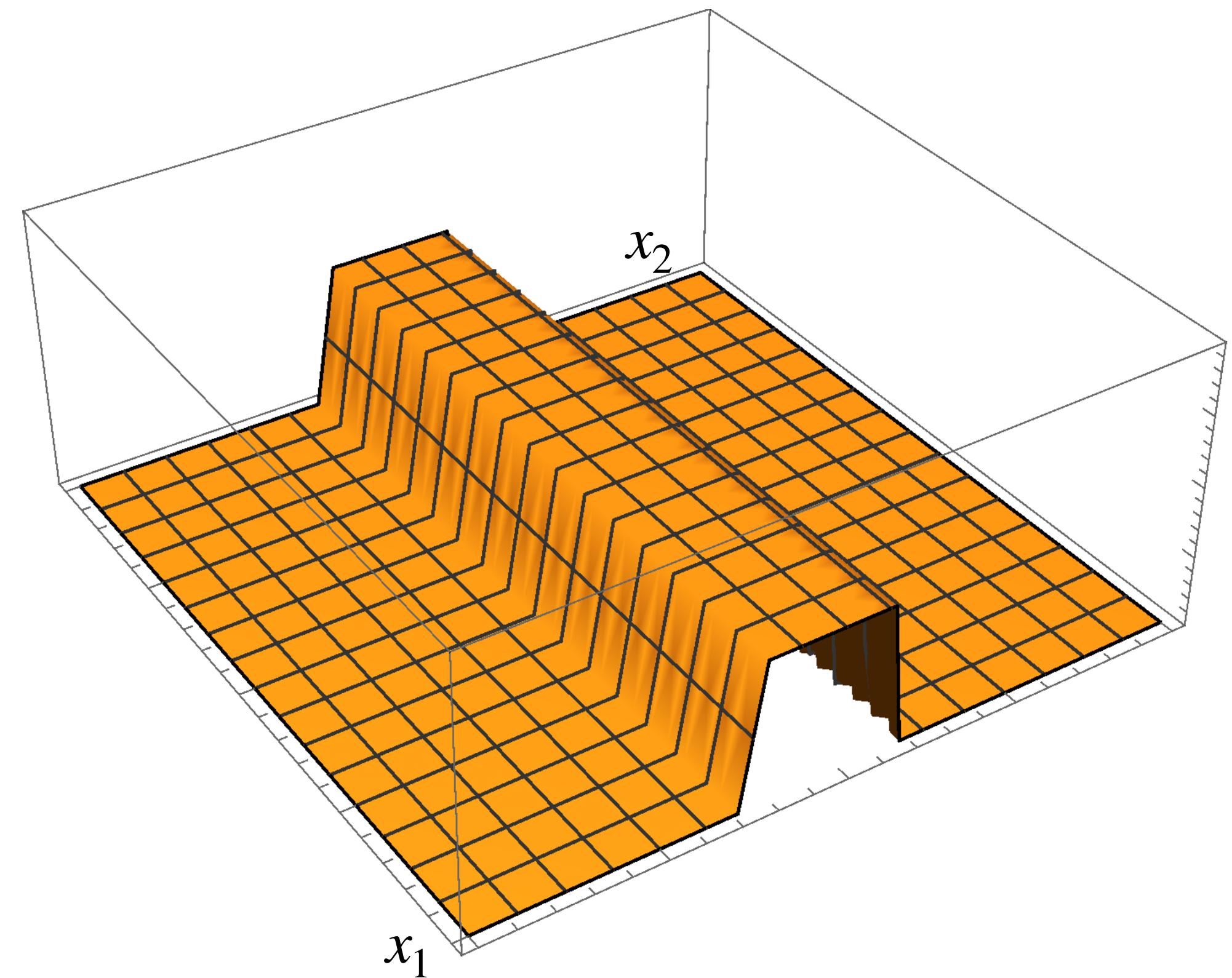
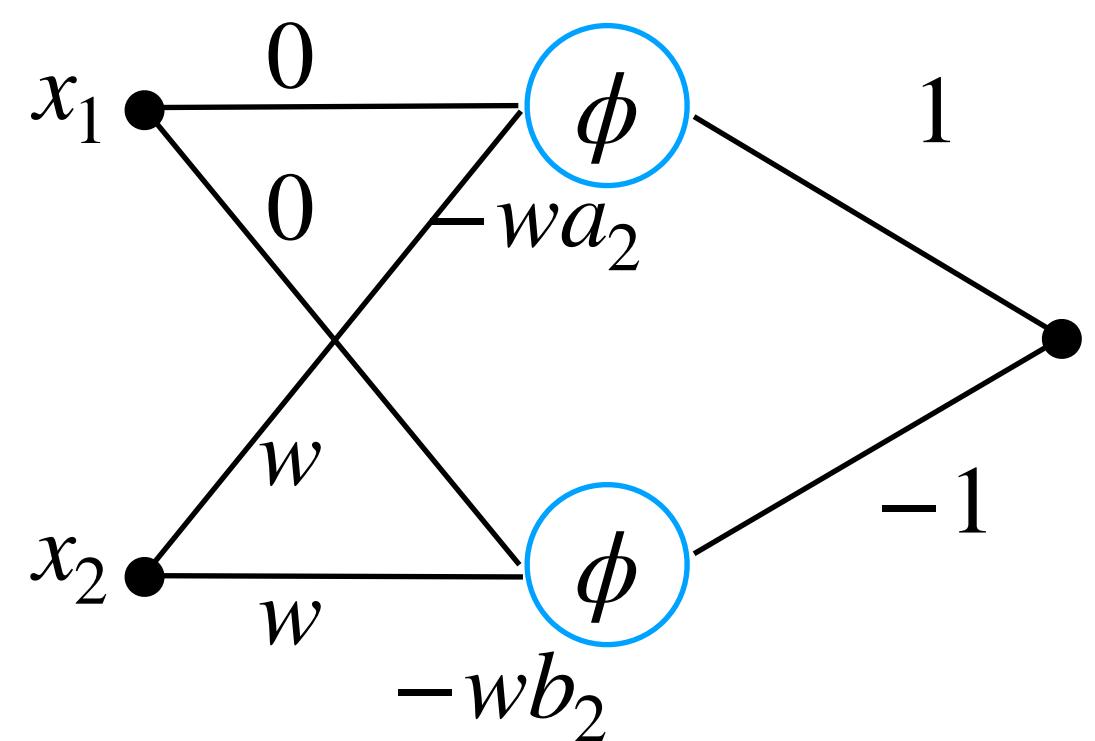


# Two sigmoids can approximate an infinite rectangle

$$(x_1, x_2) \mapsto \phi(w(x_2 - a_2)) - \phi(w(x_2 - b_2))$$

Rectangle:

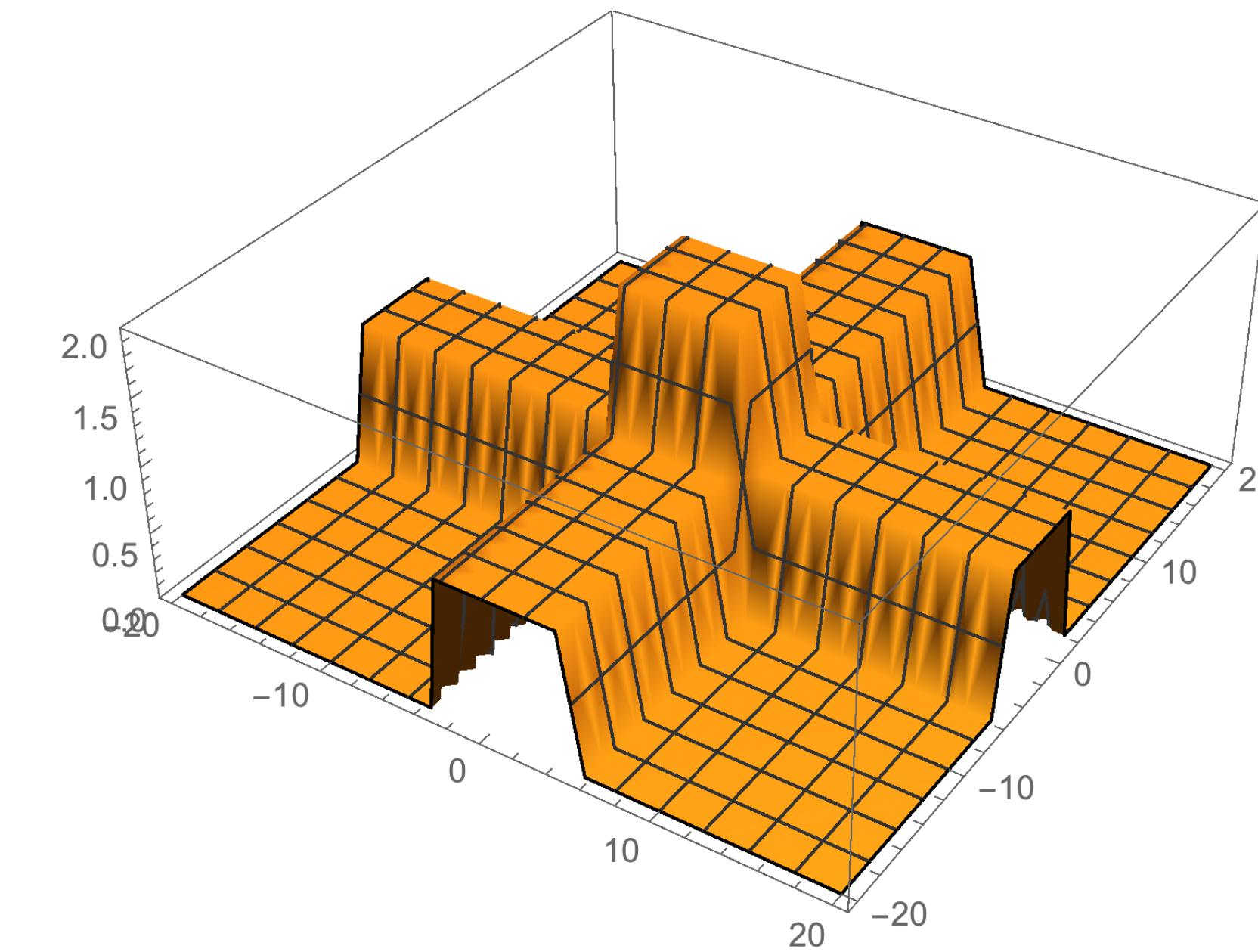
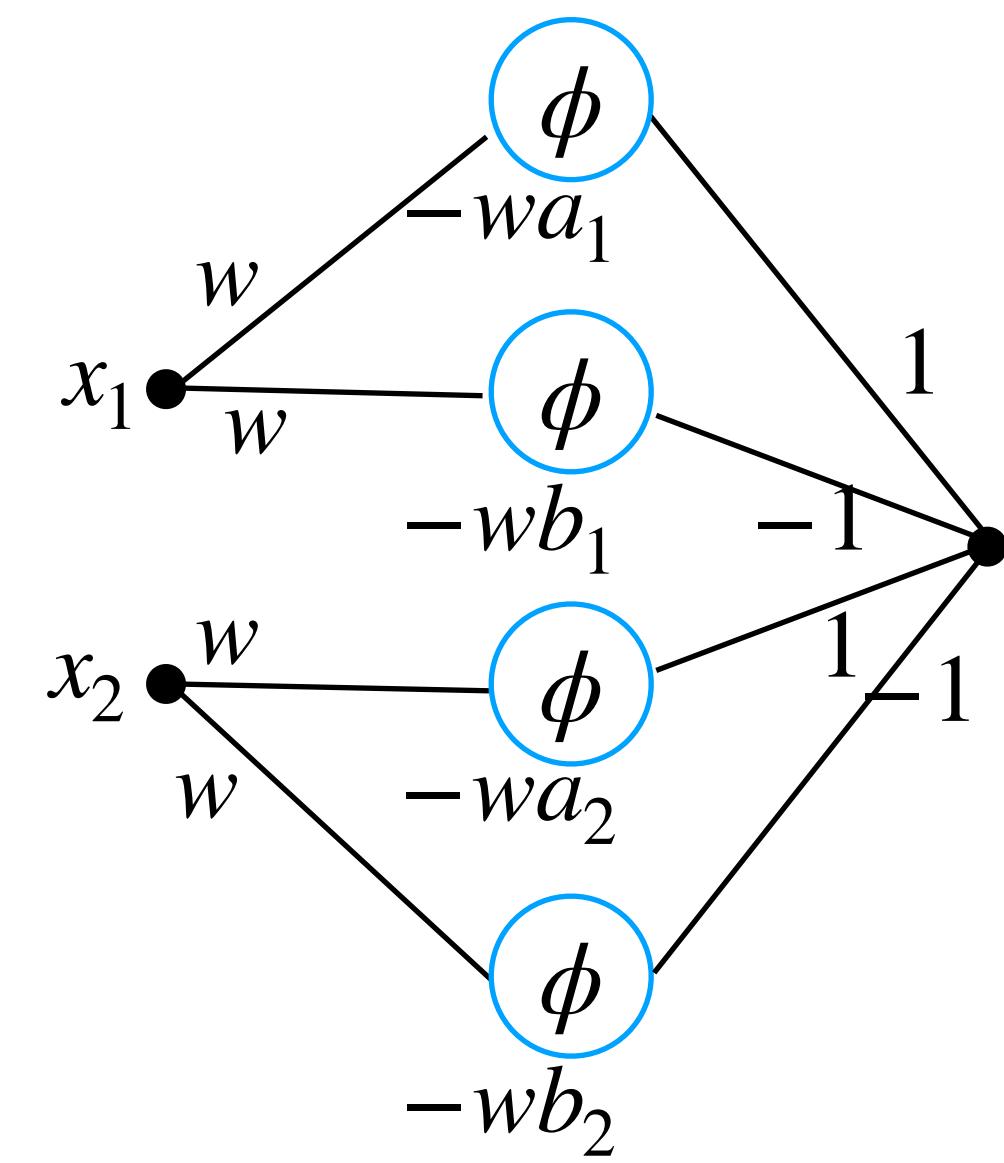
- going from  $a_2$  to  $b_2$  in the  $x_2$  direction
- unbounded in the  $x_1$  direction



# Four sigmoids can approximate a cross

$$(x_1, x_2) \mapsto \phi(w(x_1 - a_1)) - \phi(w(x_1 - b_1)) + \phi(w(x_2 - a_2)) - \phi(w(x_2 - b_2))$$

→ close to what we want with the exception of the two infinite “arms”



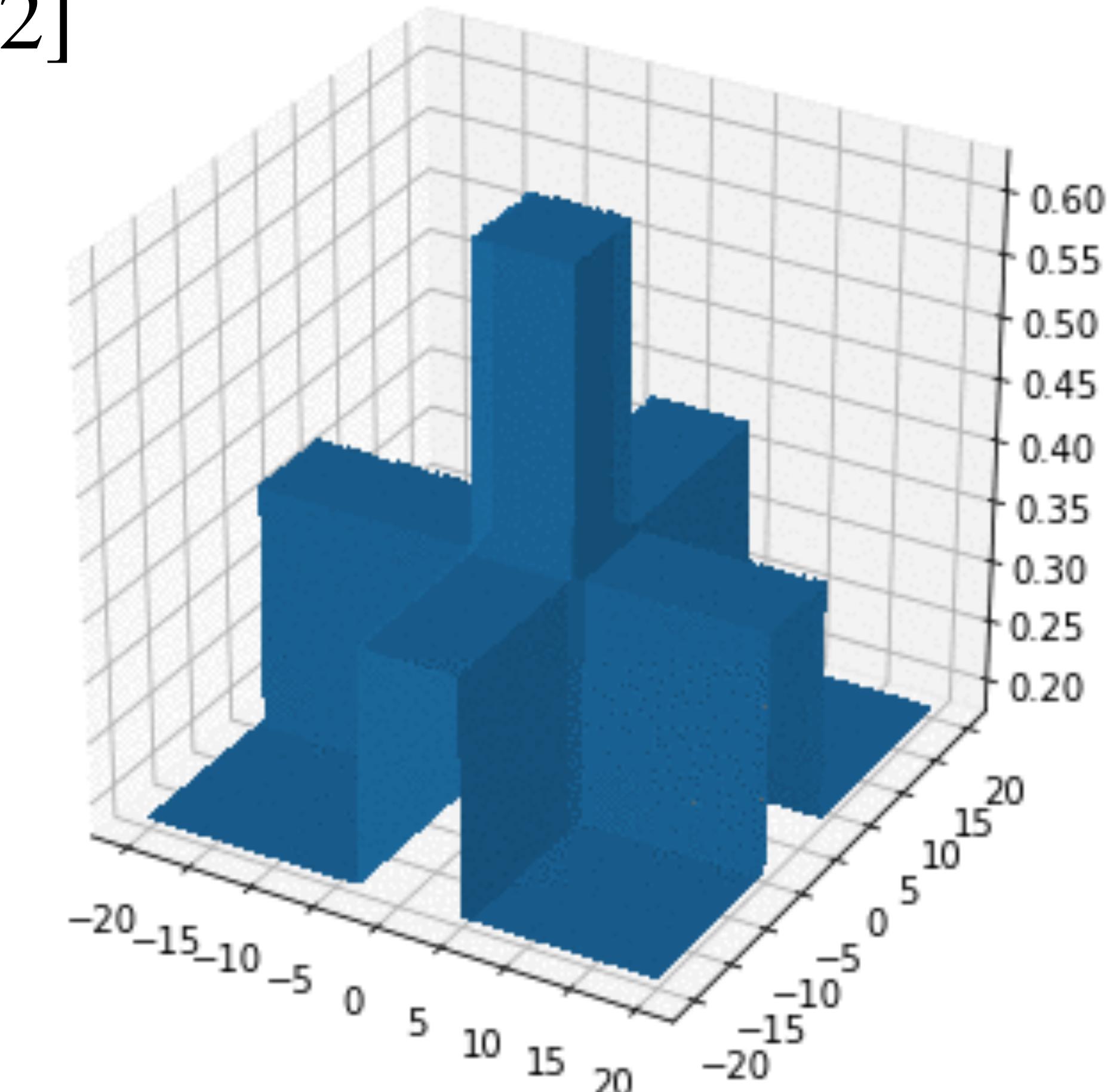
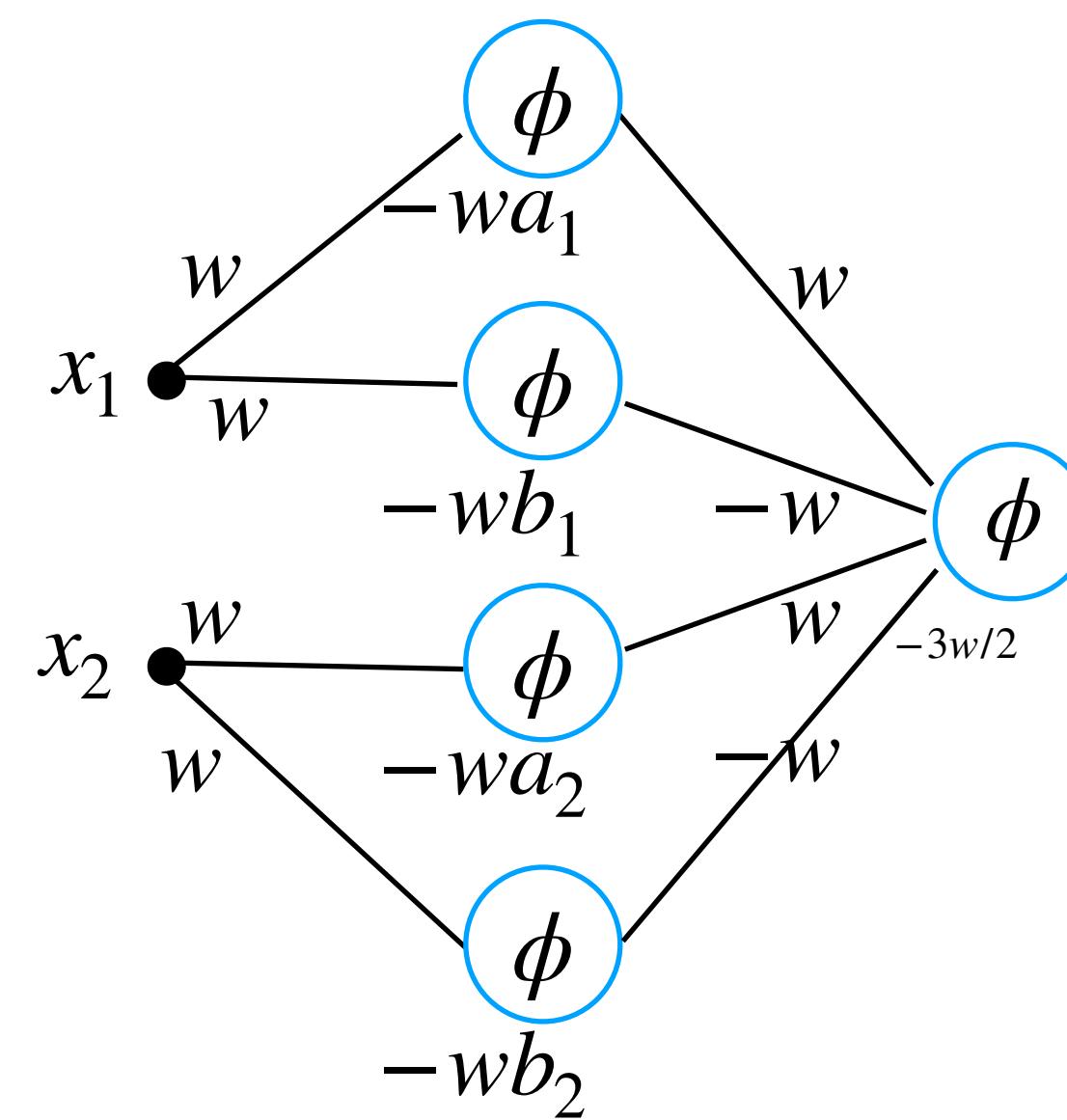
How to get rid of the crossed arms?

# The sigmoid can threshold the unwanted infinite arms

Threshold the function value would remove the arms

It is equivalent to compose it with  $1_{y \geq c}$  for  $c \in (1,2]$

- approximate  $1_{y \geq c}$  by a sigmoid with large weight  $w$  and appropriate bias (e.g.,  $3w/2$ )



# Point-wise approximations

Def: piecewise linear (PWL) function:

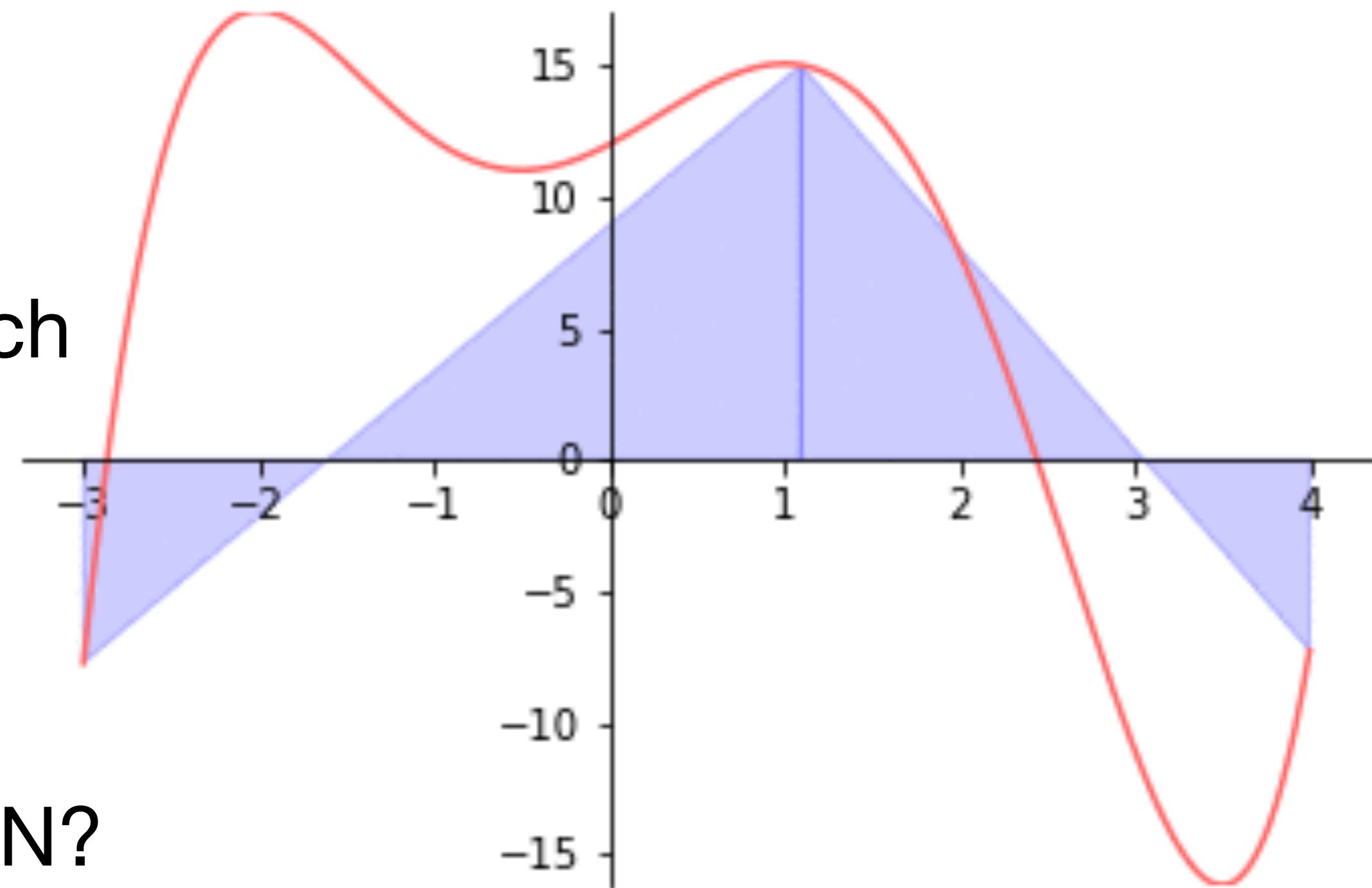
$$q(x) = \sum_{i=1}^m (a_i x + b_i) \mathbf{1}_{r_{i-1} \leq x < r_i} \text{ with } a_i r_i + b_i = a_{i+1} r_i + b_{i+1}$$

$\ell_\infty$ -approximation result (Shektman, 1982):

Let  $f$  be a continuous function on  $[0,1]$ . For all  $\varepsilon > 0$ , it exists a piecewise linear function  $q$  such that:

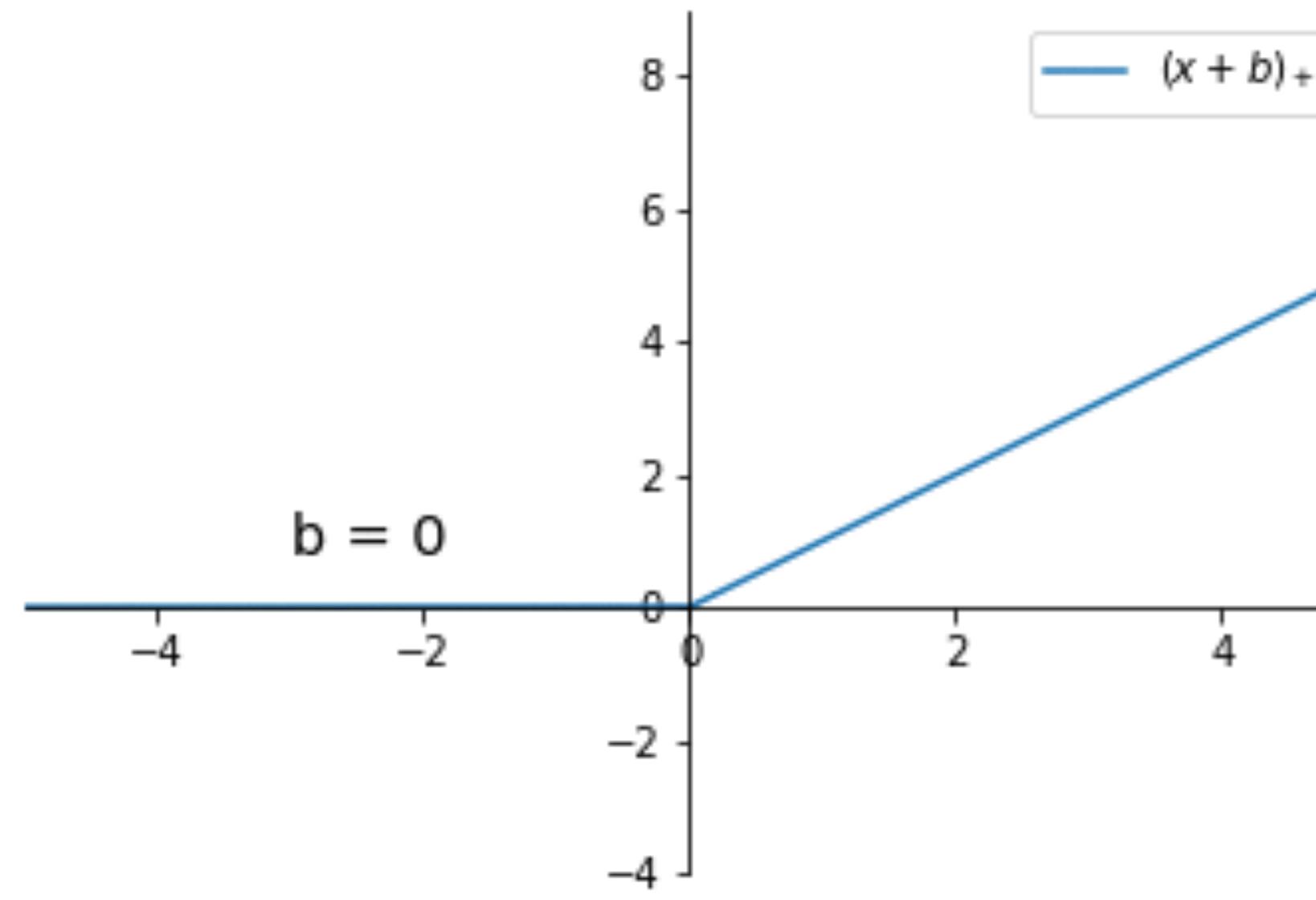
$$\sup_{x \in [0,1]} |f(x) - q(x)| \leq \varepsilon$$

→ How to approximate PWL functions with a NN?

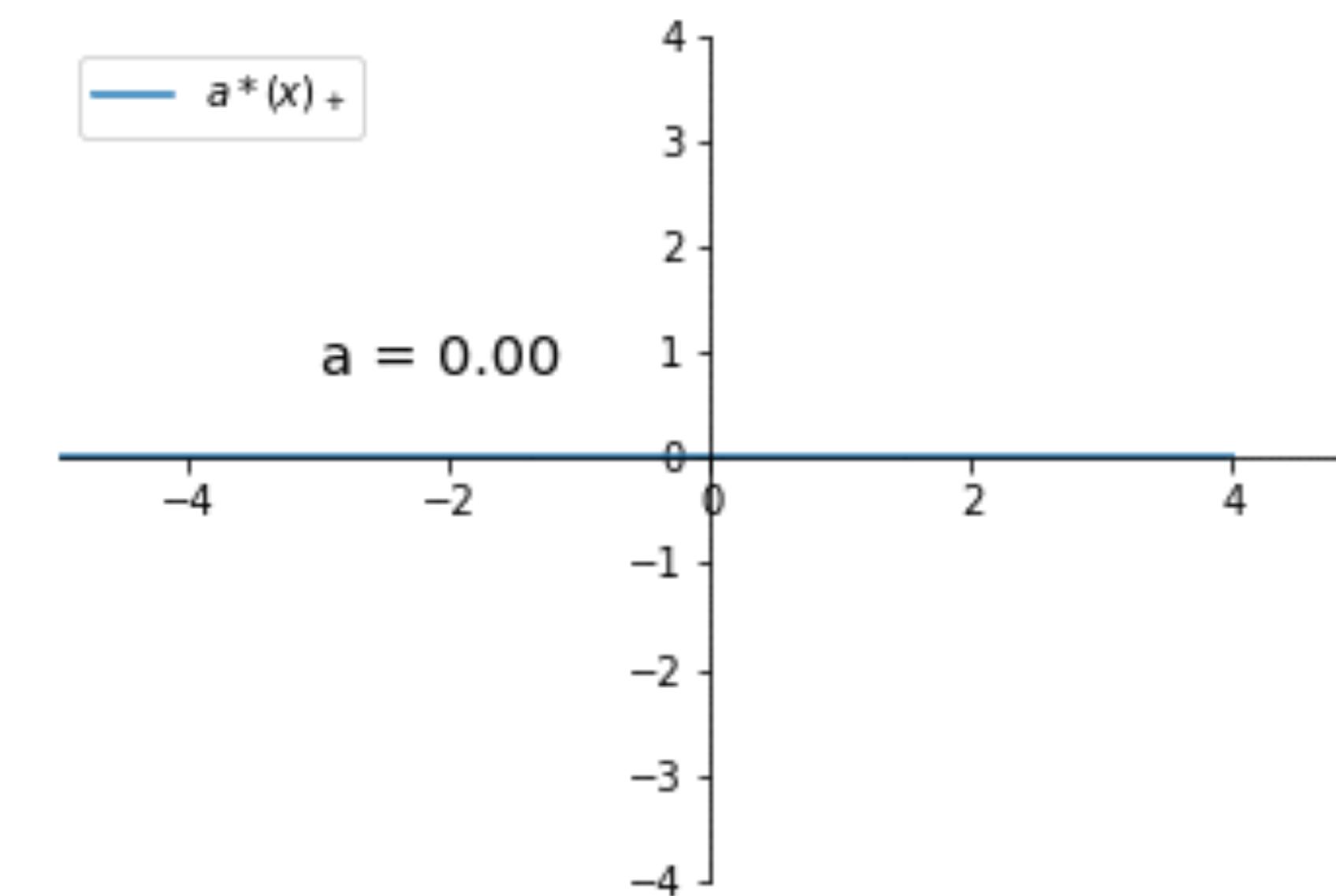


# Rectified linear unit - RELU

$$(x)_+ = \max \{0, x\}$$



The bias  $b$  determines where the kink is



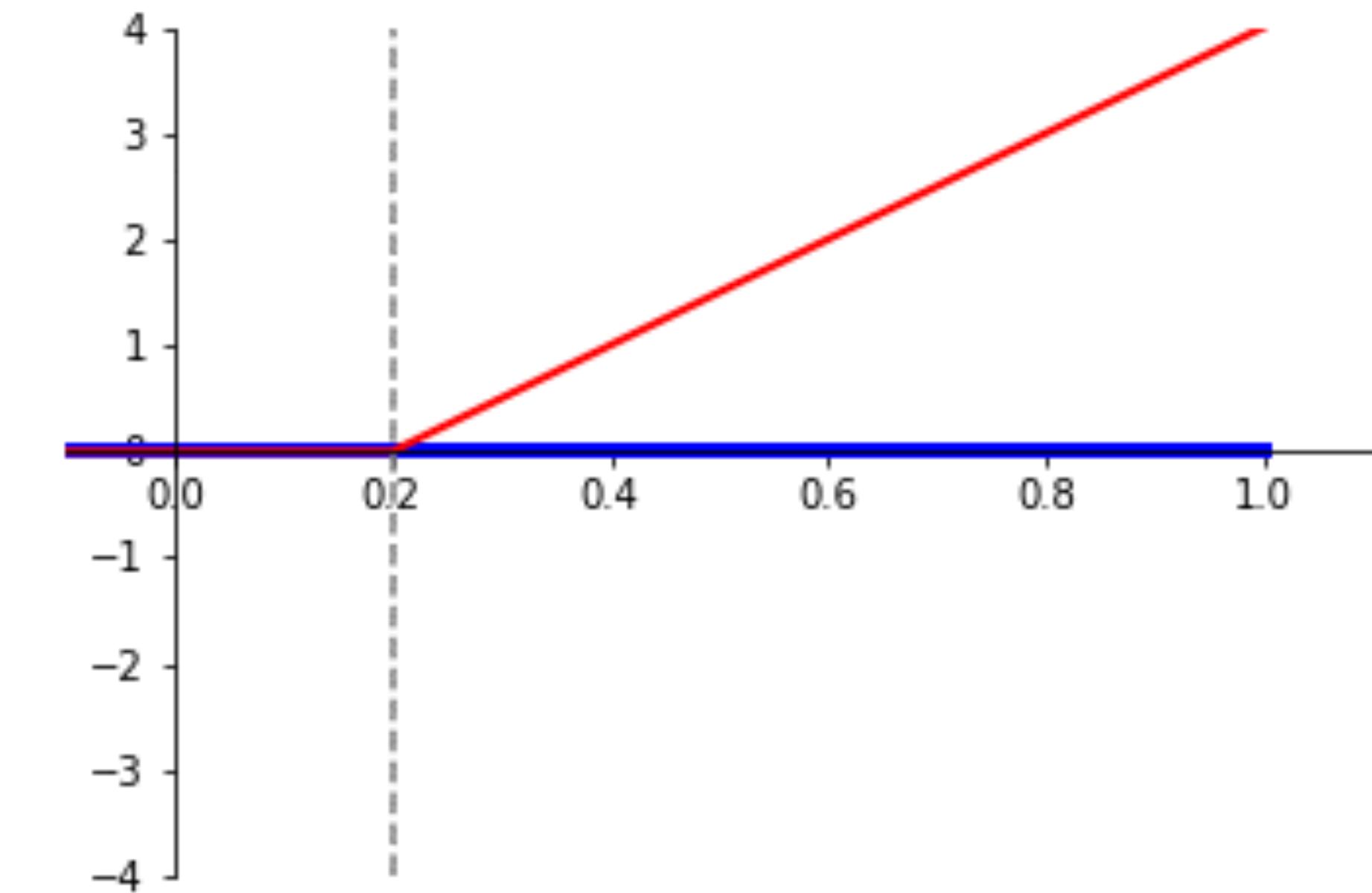
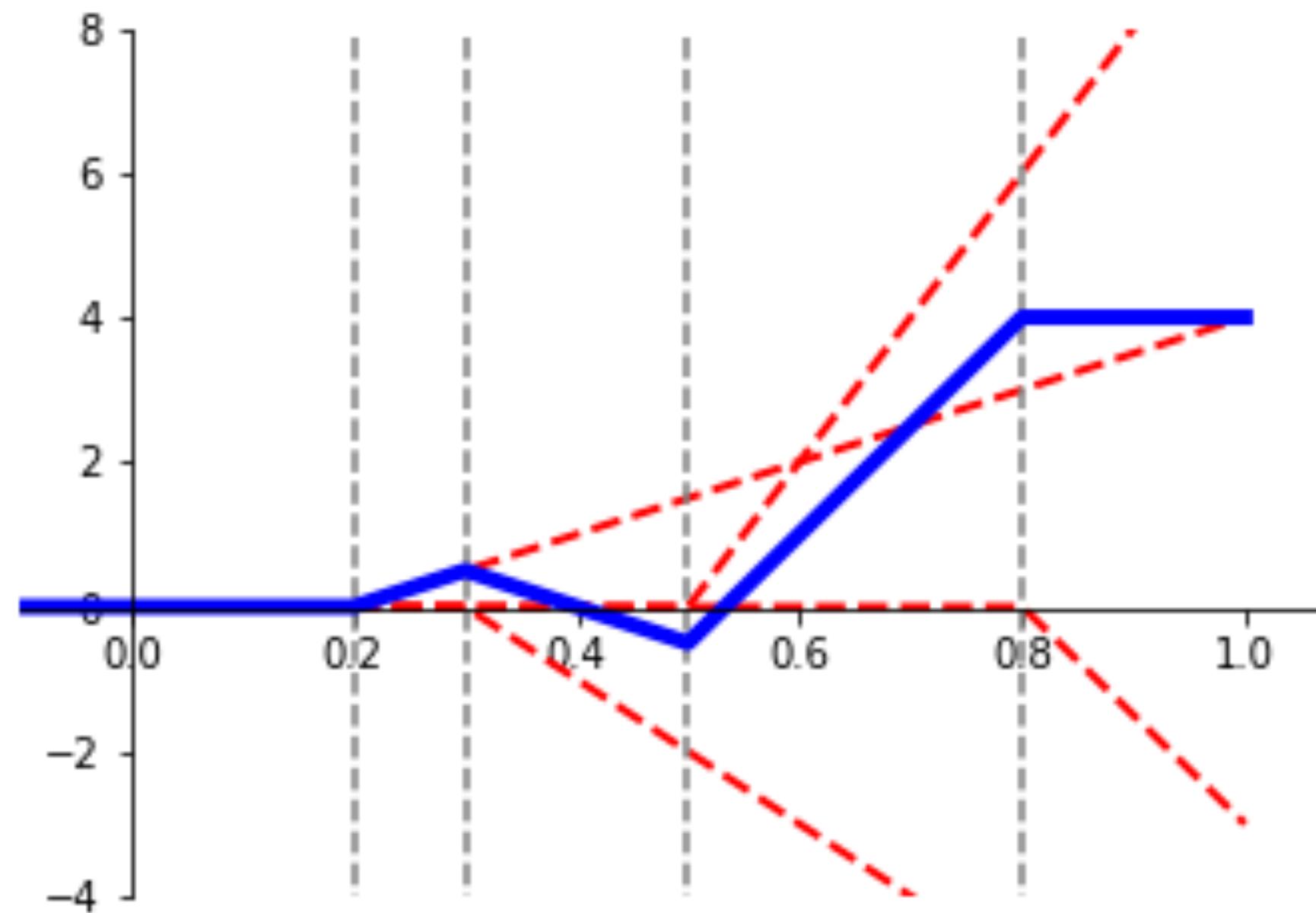
The weight  $a$  determines the slope

# Linear combinations of RELUs and PWL functions

$\sum_{i=1}^m \tilde{a}_i(x - \tilde{b}_i)_+$  is a piecewise linear function

How do we get a new segment with slope  $a$  starting at  $r > \max_i(\tilde{b}_i)$ ?

Intuition: Get the kink at  $r$  by setting  $\tilde{b}_{i+1} = r$  and slope by additionally canceling existing slope i.e.  $\tilde{a}_{i+1} = a - \sum_i \tilde{a}_i$



# Formal: Piecewise linear functions can be written as combination of RELU

Claim 1: Any PWL  $q$  can be rewritten as

$$q(x) = \tilde{a}_1 x + \tilde{b}_1 + \sum_{i=2}^m \tilde{a}_i (x - \tilde{b}_i)_+$$

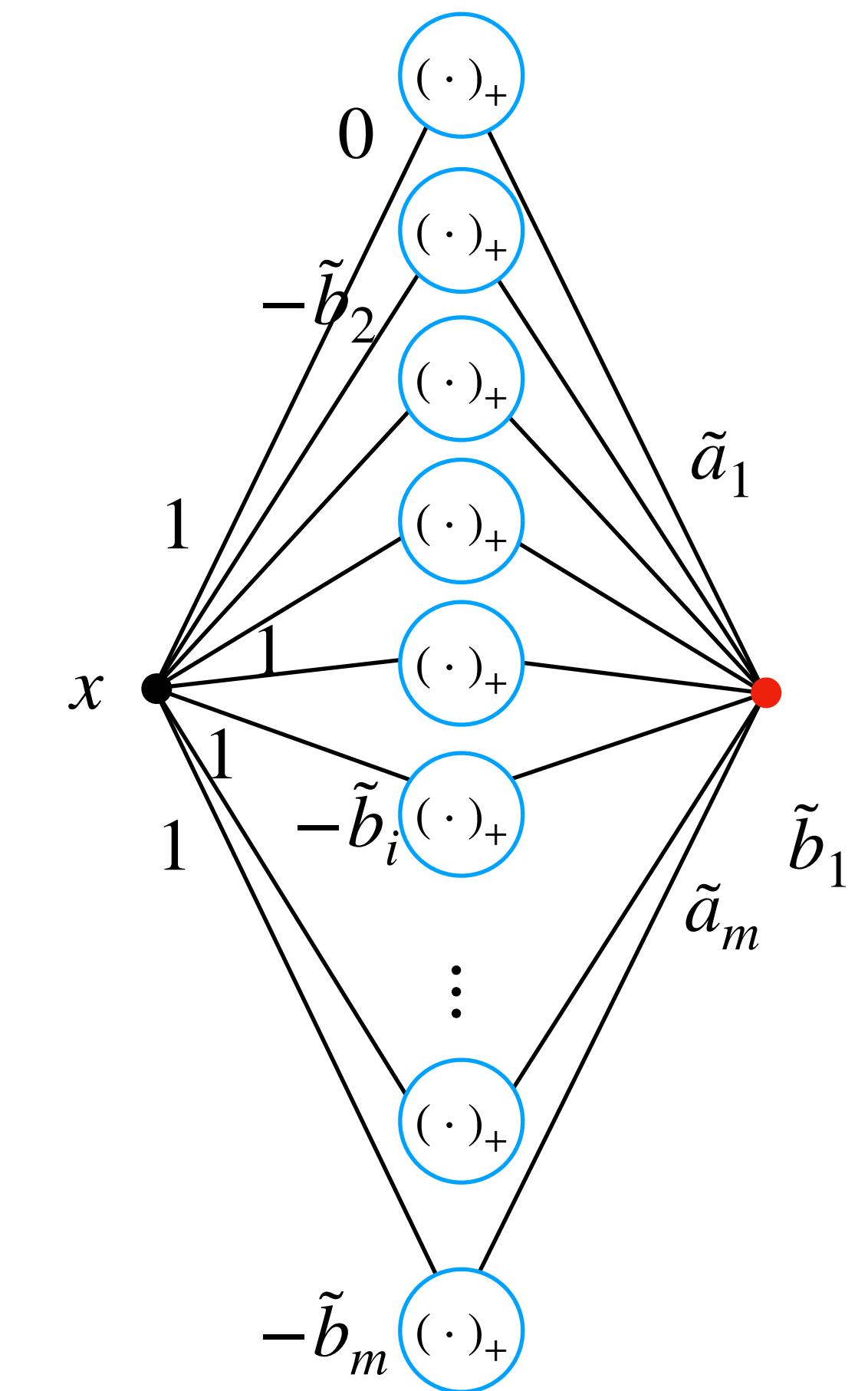
where  $\tilde{a}_1 = a_1$ ,  $\tilde{b}_1 = b_1$ ,  $a_i = \sum_{j=1}^i \tilde{a}_j$  and  $\tilde{b}_i = r_{i-1}$

Claim 2:  $q$  can be implemented as a one-hidden-layer NN with RELU activation. Each term corresponds to one node:

- Bias  $-\tilde{b}_i$
- Output weight  $\tilde{a}_i$

The term  $\tilde{a}_1 x + \tilde{b}_1$  also corresponds to one node:

- Bias  $\tilde{b}_1$ : bias of the output node
- Term  $\tilde{a}_1 x = \tilde{a}_1 (x)_+$  since  $x \in [0,1]$



# Proof of the equivalent formulation

$$q(x) = \sum_{i=1}^m (a_i x + b_i) 1_{r_{i-1} \leq x < r_i} \quad r(x) = \tilde{a}_1 x + \tilde{b}_1 + \sum_{i=2}^m \tilde{a}_i (x - \tilde{b}_i)_+$$

$$\tilde{a}_1 = a_1, \tilde{b}_1 = b_1 \text{ and } a_i = \sum_{j=1}^i \tilde{a}_j \text{ and } \tilde{b}_i = r_{i-1}$$

- For  $x \in [0, r_1]$   
 $(\tilde{a}_1, \tilde{b}_1) = (a_1, b_1) \implies q(x) = a_1 x + b_1 = \tilde{a}_1 x + \tilde{b}_1 = r(x)$  because  $\tilde{b}_2 = r_1$
- For  $x \in [r_1, r_2]$ ,  
$$\begin{aligned} r(x) &= \tilde{a}_1 x + \tilde{b}_1 + (a_2 - a_1)(x - r_1)_+ \\ &= a_1 x + b_1 + (a_2 - a_1)(x - r_1) = a_2 x + b_1 - (a_2 - a_1)r_1 \end{aligned}$$
  
 $r'(x) = a_2$  and  $r(r_1) = q(r_1)$  as shown above  
 $\implies r(x) = q(x)$  for  $x \in [r_1, r_2]$

# Proof by induction

Let's assume that  $r(x) = q(x)$  for  $x \in [0, r_{i-1}]$

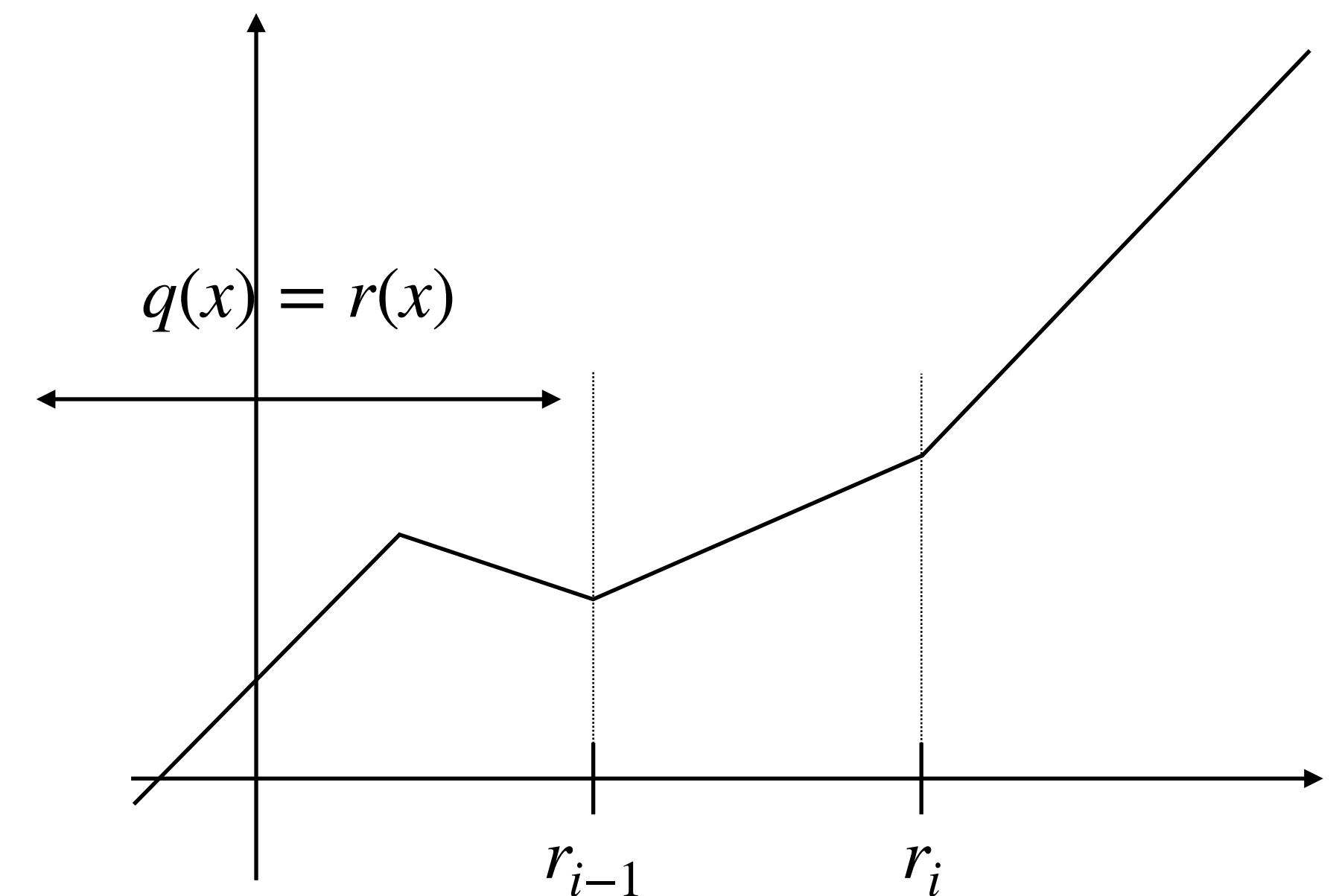
For  $x \in [r_{i-1}, r_i]$

$$\begin{aligned} r(x) &= \tilde{a}_1 x + \tilde{b}_1 + \sum_{j=2}^m \tilde{a}_j (x - \tilde{b}_j)_+ \\ &= \tilde{a}_1 x + \tilde{b}_1 + \sum_{j=2}^i \tilde{a}_j (x - \tilde{b}_j) \\ &= \sum_{j=1}^i \tilde{a}_j x + \tilde{b}_1 - \sum_{j=2}^i \tilde{a}_j \tilde{b}_j \end{aligned}$$

Thus

- $r'(x) = \sum_{j=1}^i \tilde{a}_j = a_i$  good slope
- $r(r_{i-1}) = q(r_{i-1})$  good starting point

$$\implies r(x) = q(x) \text{ for } x \in [r_{i-1}, r_i]$$



Why: two affine functions with the same starting point and the same slope are equal

# Proof by induction - bis

Let's assume that  $r(x) = q(x)$  for  $x \in [0, r_{i-1}]$

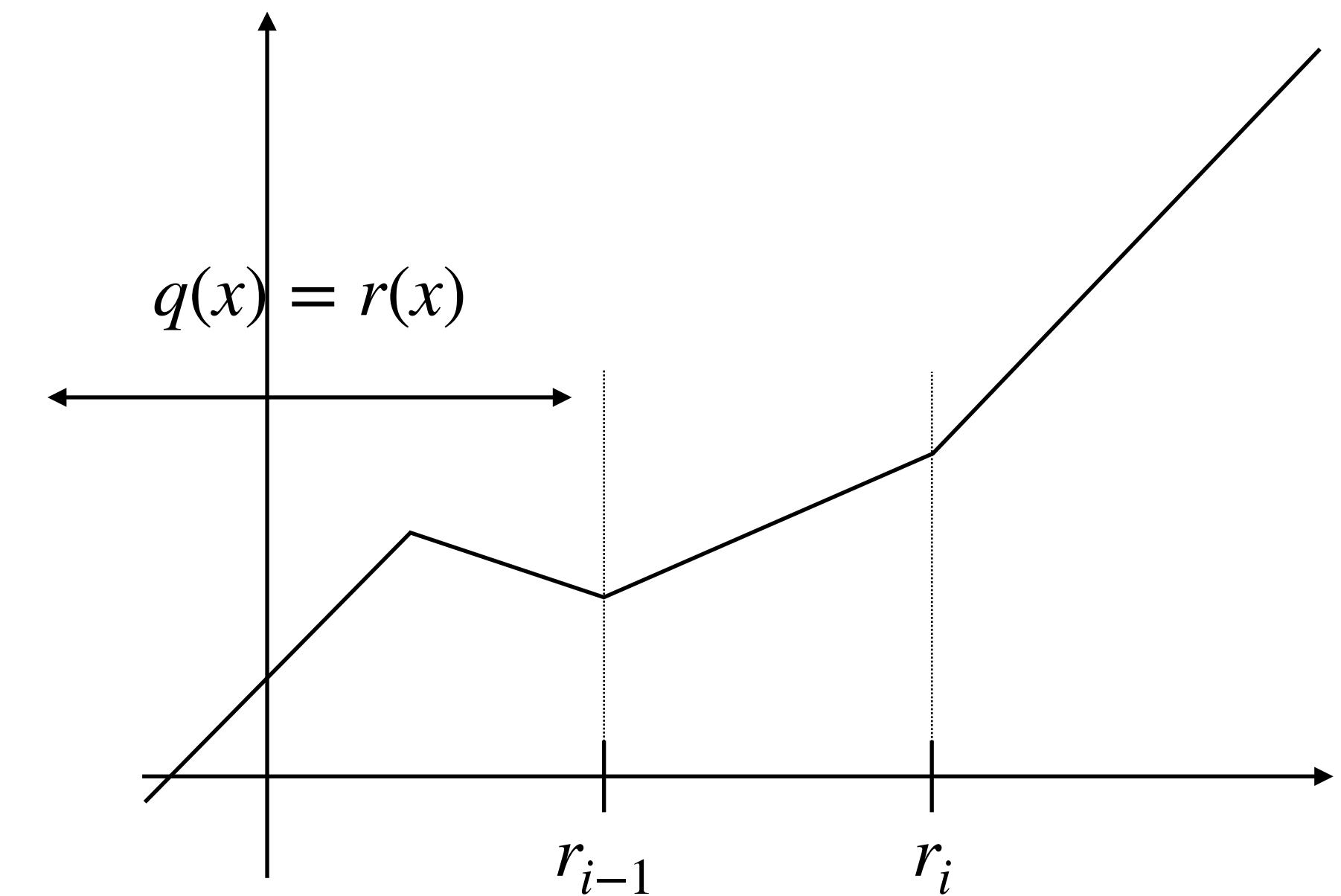
For  $x \in [r_{i-1}, r_i]$

$$\begin{aligned} r(x) &= \tilde{a}_1 x + \tilde{b}_1 + \sum_{j=2}^m \tilde{a}_j (x - \tilde{b}_j)_+ \\ &= \underbrace{\tilde{a}_1 x + \tilde{b}_1 + \sum_{j=2}^{i-1} \tilde{a}_j (x - \tilde{b}_j)_+}_{= q(x), x \in [r_{i-2}, r_{i-1}] \text{ by induction}} + \tilde{a}_i (x - \tilde{b}_i) \\ &= a_{i-1} x + b_{i-1} + \tilde{a}_i (x - \tilde{b}_i) \end{aligned}$$

Thus

- $r'(x) = a_{i-1} + \tilde{a}_i = a_i$  good slope
- $r(r_{i-1}) = q(r_{i-1})$  good starting point

$$\implies r(x) = q(x) \text{ for } x \in [r_{i-1}, r_i]$$



Why: two affine functions with the same starting point and the same slope are equal

# Conclusion: representation power

- $f$  smooth (condition on its Fourier coefficients)
- Bounded domain
- Condition on the activation function
- Average approximation in  $\ell_2$ -norm but also point-wise approximation in  $\ell_\infty$ -norm