Neural Networks: Basic Structure, Representation Power

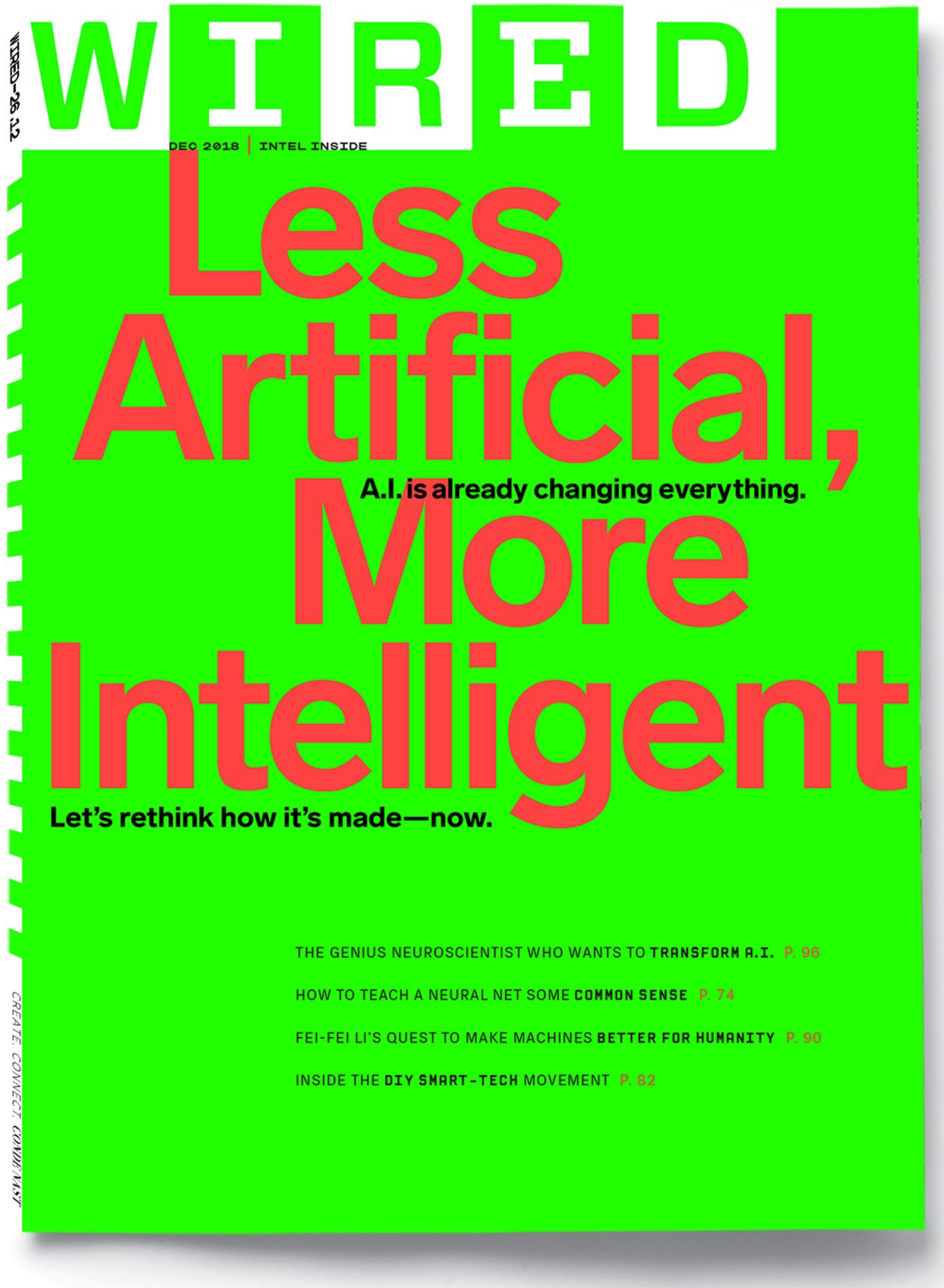
Machine Learning Course - CS-433

Nov 9, 2021

Nicolas Flammarion

EPFL

A lot of hype for NNs: perform very well in practice but are still theoretically misunderstood



Neural Networks: motivation

Supervised learning : we observe some data $S_{\text{train}} = \{x_i, y_i\}_{i=1}^n \in \mathcal{X} \times \mathcal{Y}$

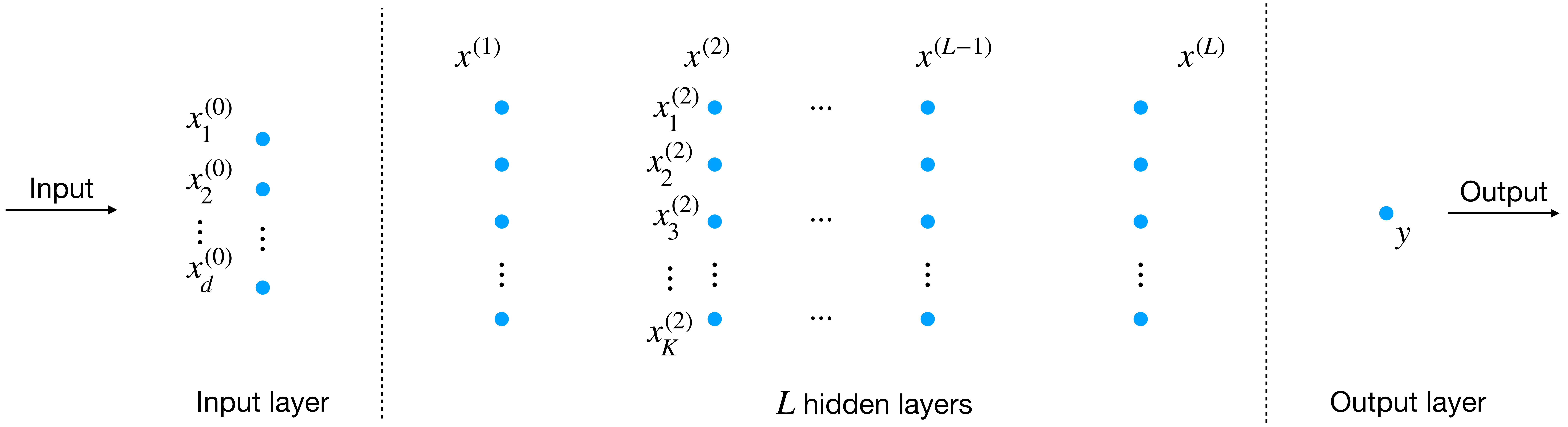
➡ given a new x, we want to predict its label y

Linear predictions: it works well only when used with **good features**

Data representation:

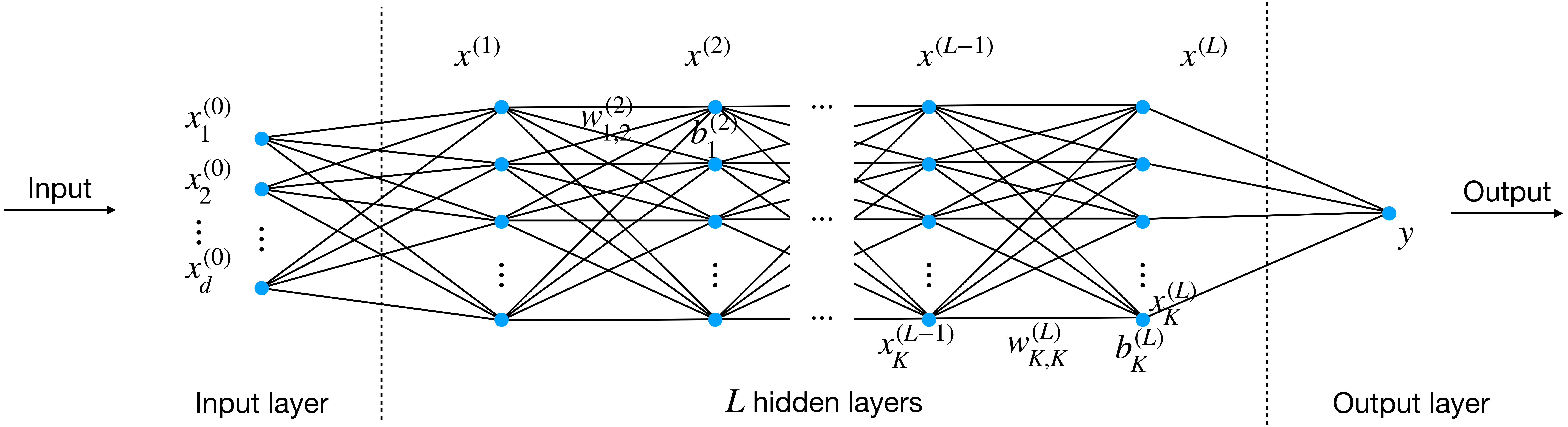
- Before: domain experts derived efficient features for a particular application
- Now: neural networks directly learn the features from the data

NNs: the basic structure



- $x_i^{(l)}$: value for the i-th node of the l-th layer
- Each layer has the same number of nodes
- Feedforward network - there is no feedback loop
- Same NN for regression and classification - only the output layer changes

Fully connected NNs: each node is connected to all the nodes in the previous layer



Parameters of the network to be learnt:

- $w_{i,j}^{(l)}$: weight associated with the edge going from node i in layer $l-1$ to node j in layer l
- $b_j^{(l)}$: bias term associated with node j in layer l

The function value at the l^{th} layer is defined recursively

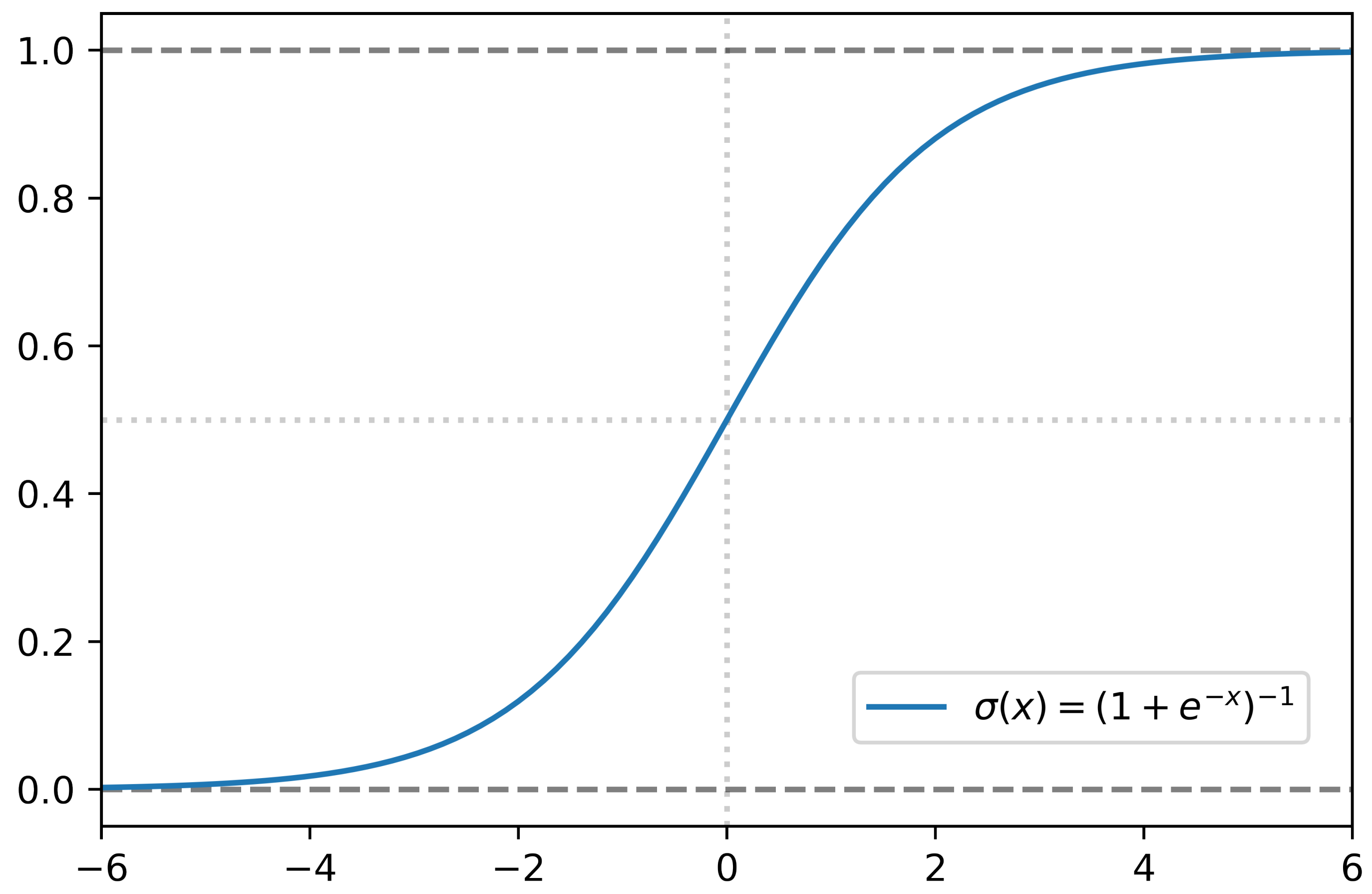
The output is given by

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

ϕ: activation function:

- Sigmoid $\frac{1}{1+e^{-x}}$
- RELU $\max\{0,x\}$

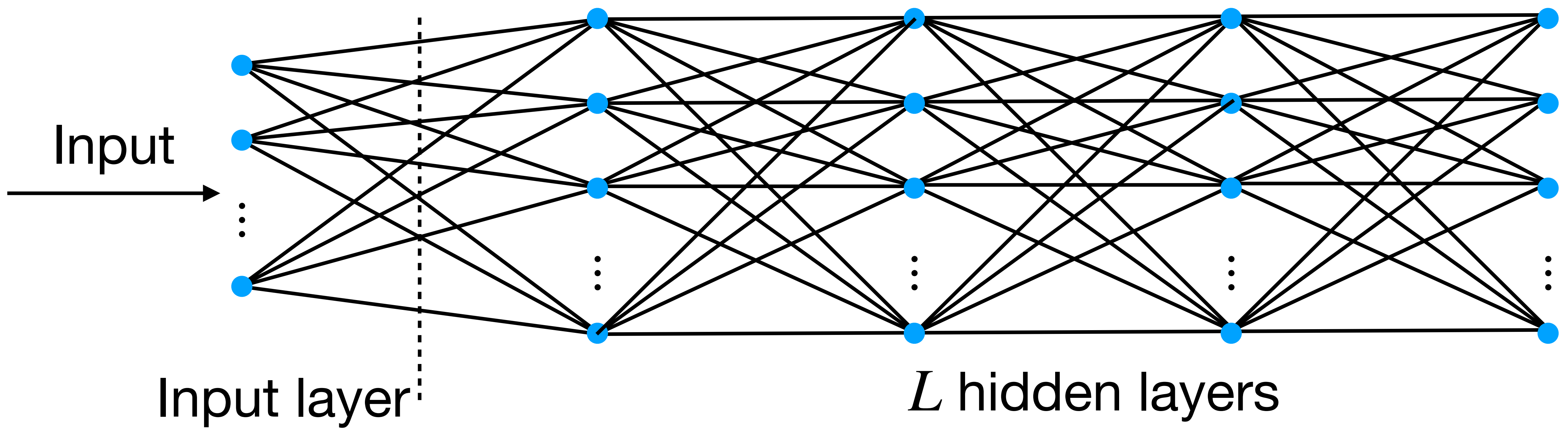
<u>Key fact</u>: ϕ is non-linear - otherwise NNs only represent linear functions



Sigmoid function

The NN transforms the input into a more suitable representation

The NN can be decomposed in two. The first part represents a function from $\mathbb{R}^d$ to $\mathbb{R}^K$



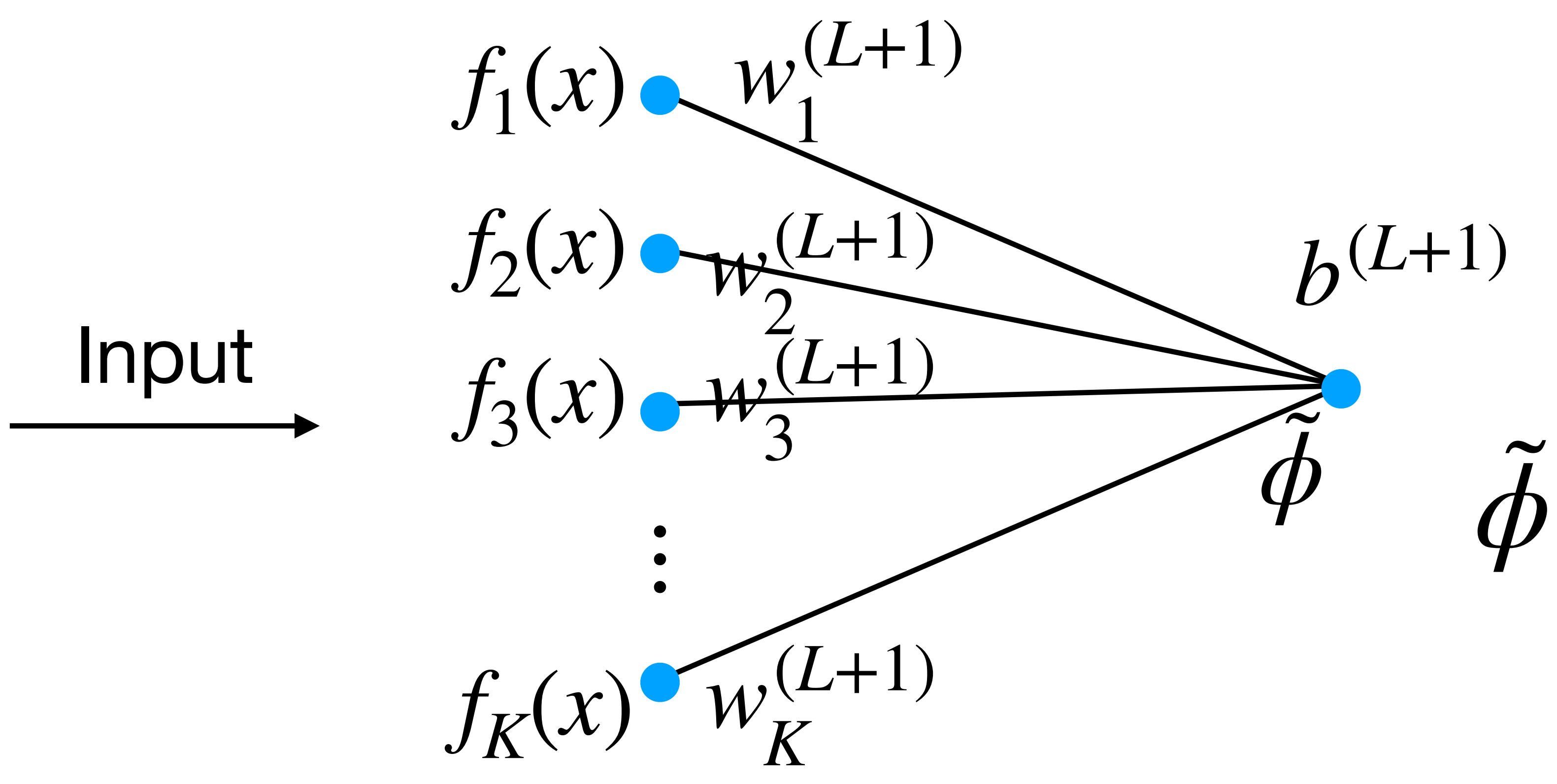
$$\Longleftrightarrow \qquad \begin{aligned} &\mathbb{R}^d \to \mathbb{R}^K \\ f: \quad &x \mapsto \begin{pmatrix} f_1(x) \\ \vdots \\ f_K(x) \end{pmatrix} \end{aligned}$$

This function is defined by

- The biases $\{b_i^{(l)}\}_{i \in [K], l \in [L]}$ and weights $\{w_{i,j}^{(l)}\}_{i,j \in [K], l \in [L]}$ we learn
  - ➡ $O(K^2 L)$ parameters
- The activation function ϕ we pick

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

The last layer performs the desired ML task



Output

$$\tilde{\phi}\left(f(x)^\top w^{(L+1)} + b^{(L+1)}\right)$$

Regression $\tilde{\phi} = I$

$$y = f(x)^\top w^{(L+1)} + b^{(L+1)}$$

Classification $\tilde{\phi}$ **:sigmoid**

$$y = \tilde{\phi}\left(f(x)^\top w^{(L+1)} + b^{(L+1)}\right)$$

Data representation

Linear prediction

ML task

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

The three main challenges of deep learning

- **Expressive power** of NNs: why are the function we are interested in so **well approximated** by NNs?

- **Success of naive optimisation**: why does **gradient descent** lead to a good local minimum?

- **Generalization miracle**: why is there **no overfitting** with so many parameters?

Barron’s Approximation result

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

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

<u>Claim</u>: For all $n \geq 1$, 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

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

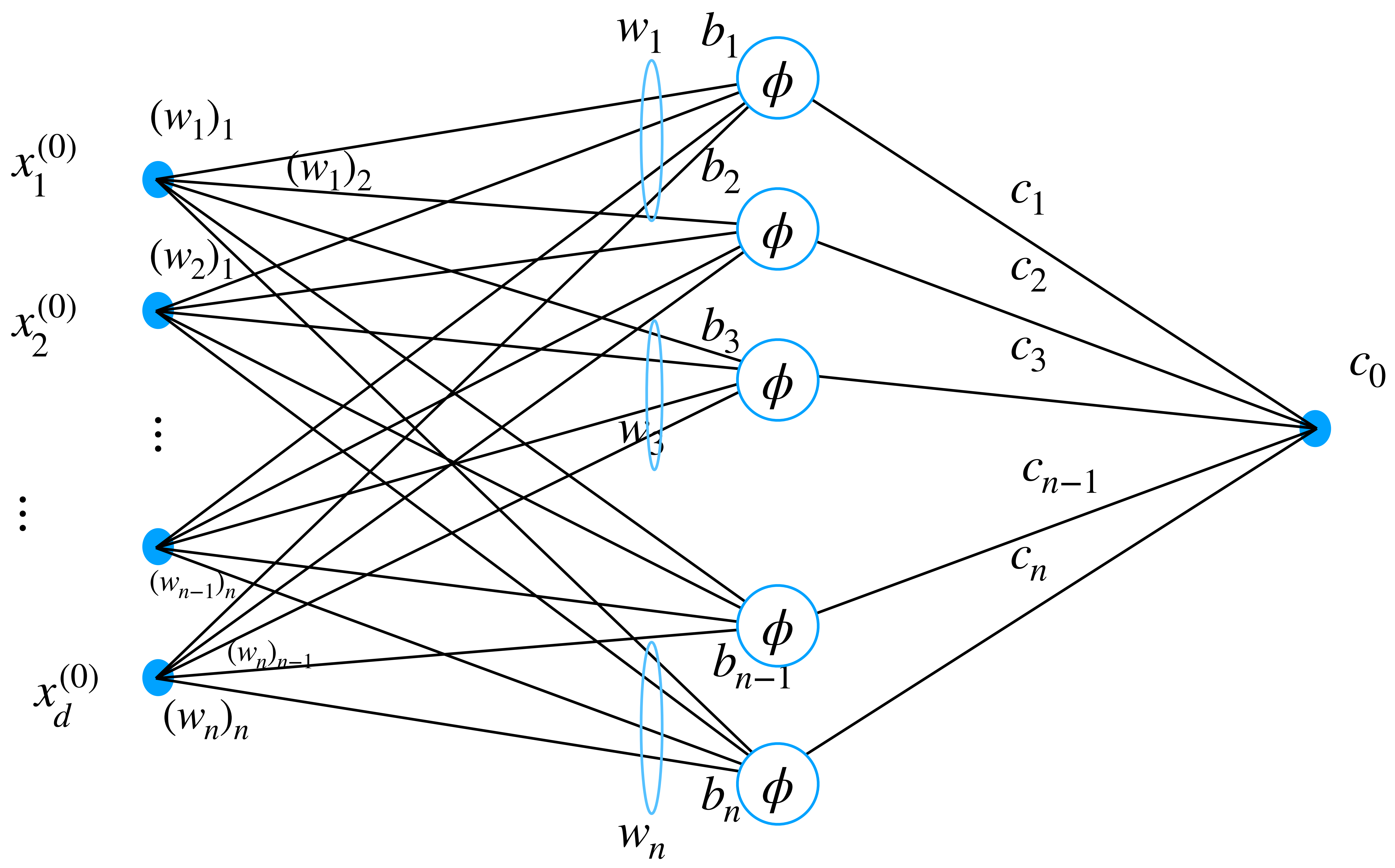
All sufficiently smooth function can be approximated by a one-hidden-layer NN

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

- 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 ℓ_2-norm)
- For any “sigmoid-like” activation function

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

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

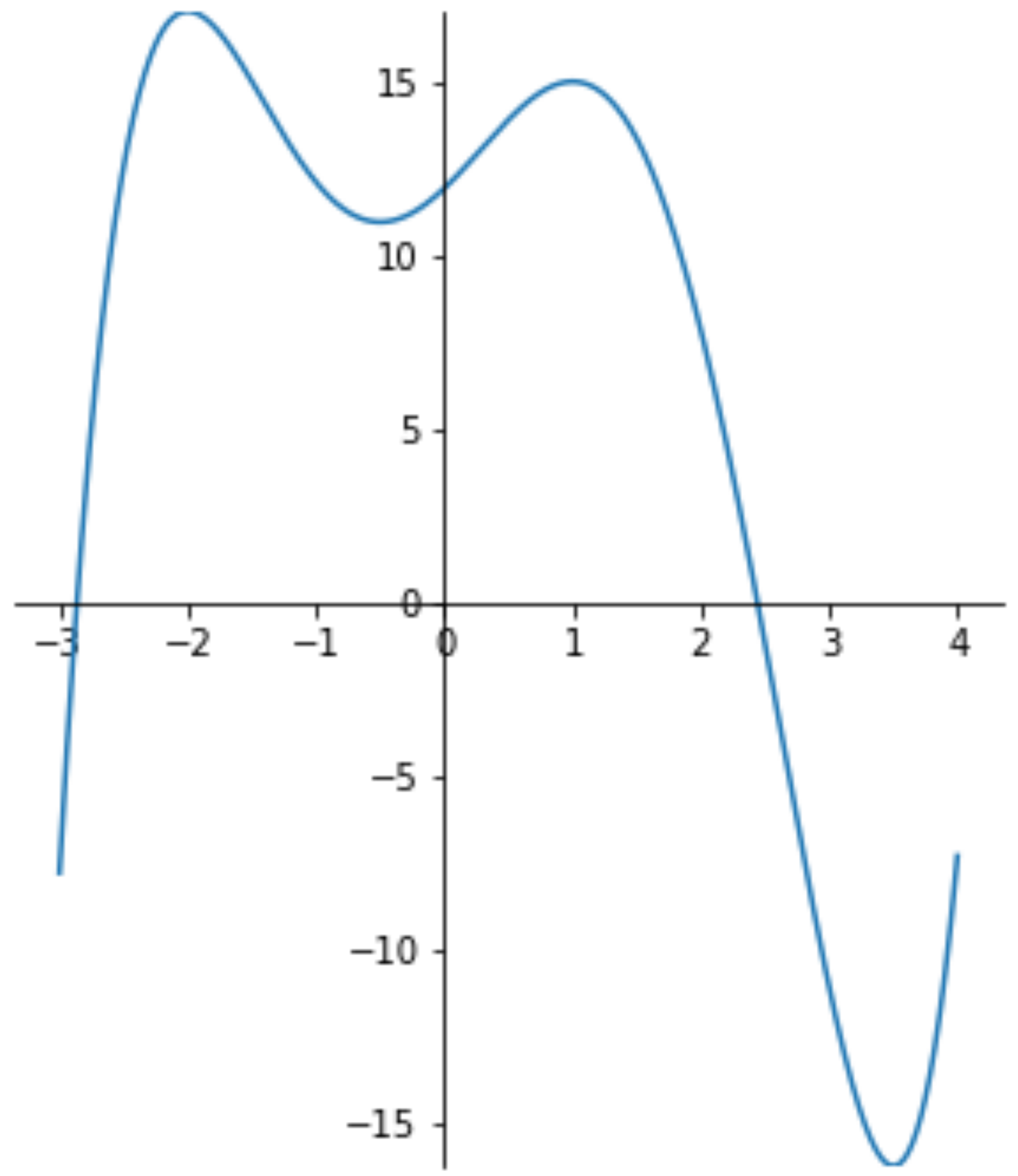


Proof by picture

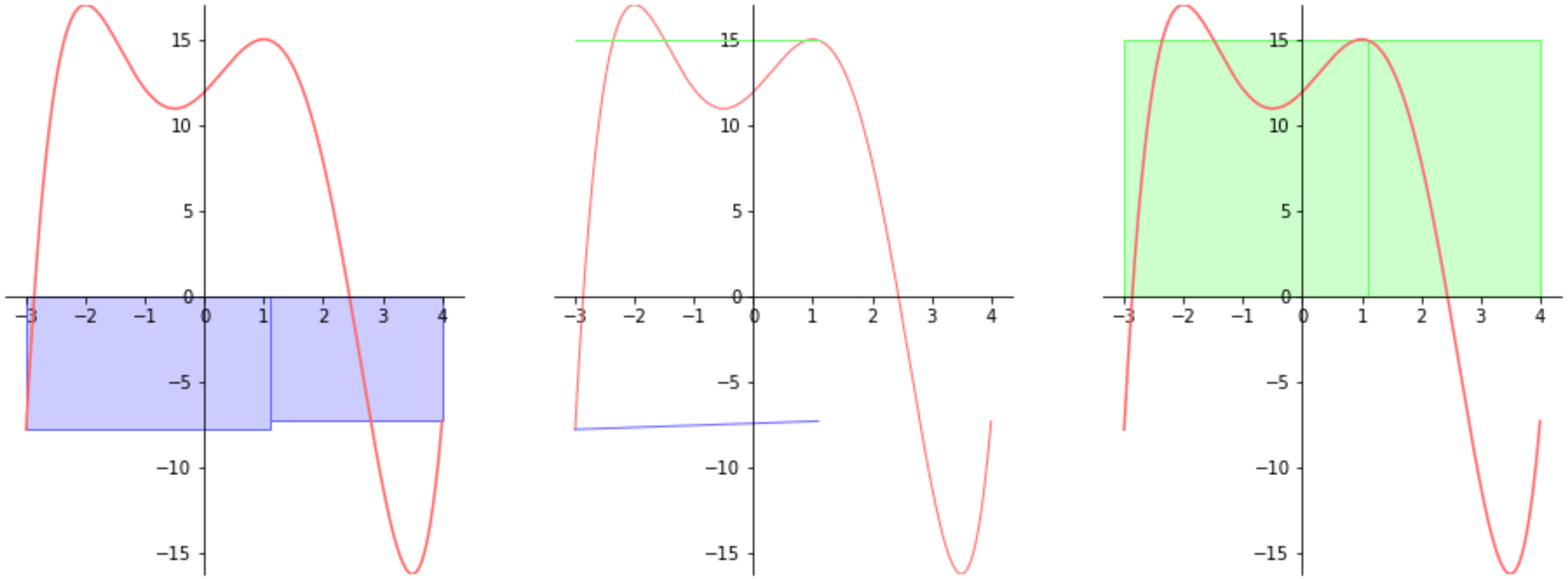
Simple and intuitive explanation of a slightly different result:

"A NN with sigmoid activation and at most two hidden layers can approximate well a smooth function in average, i.e, in ℓ_1-norm"

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



Approximation of the function by a sum of rectangles

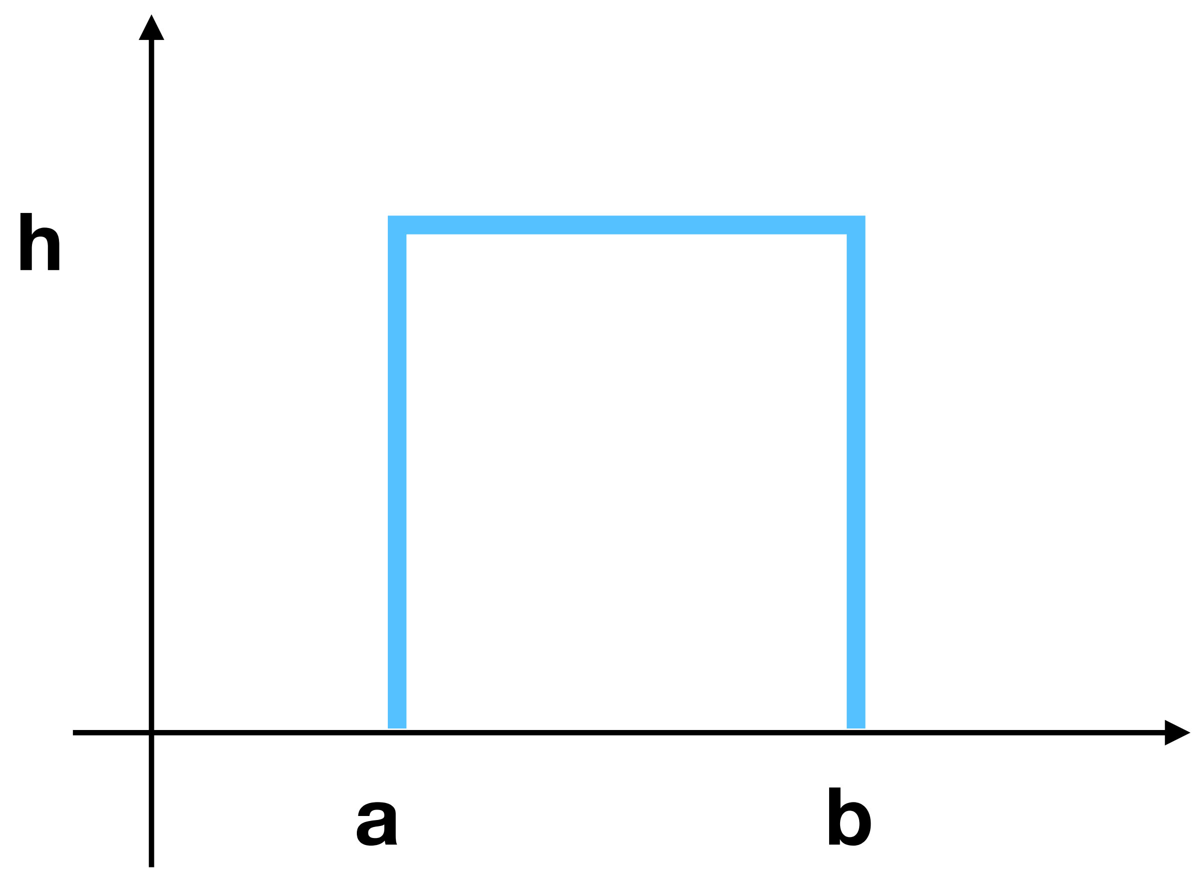


From below

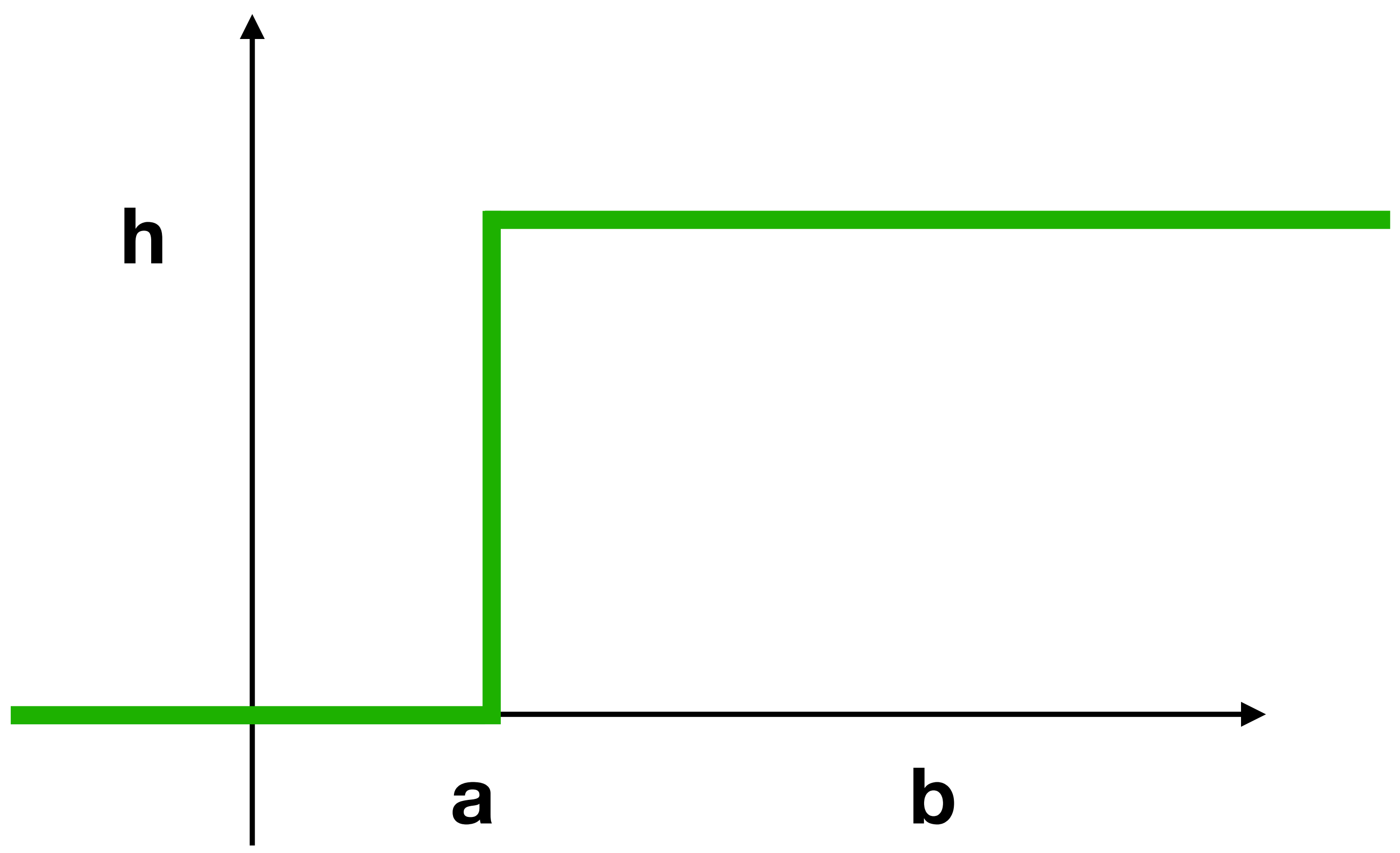
From above

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

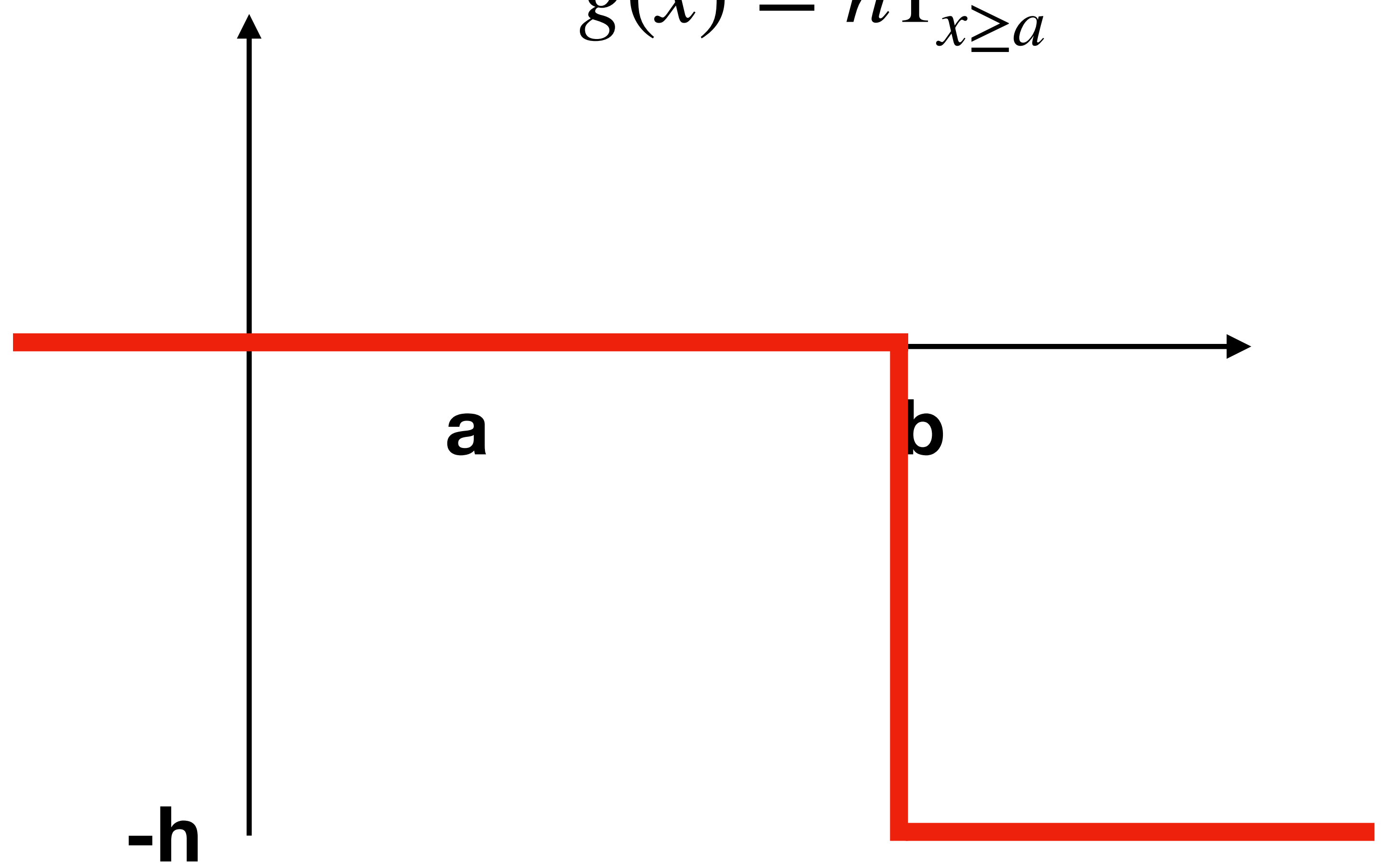
A rectangle is equal to the sum of two step functions



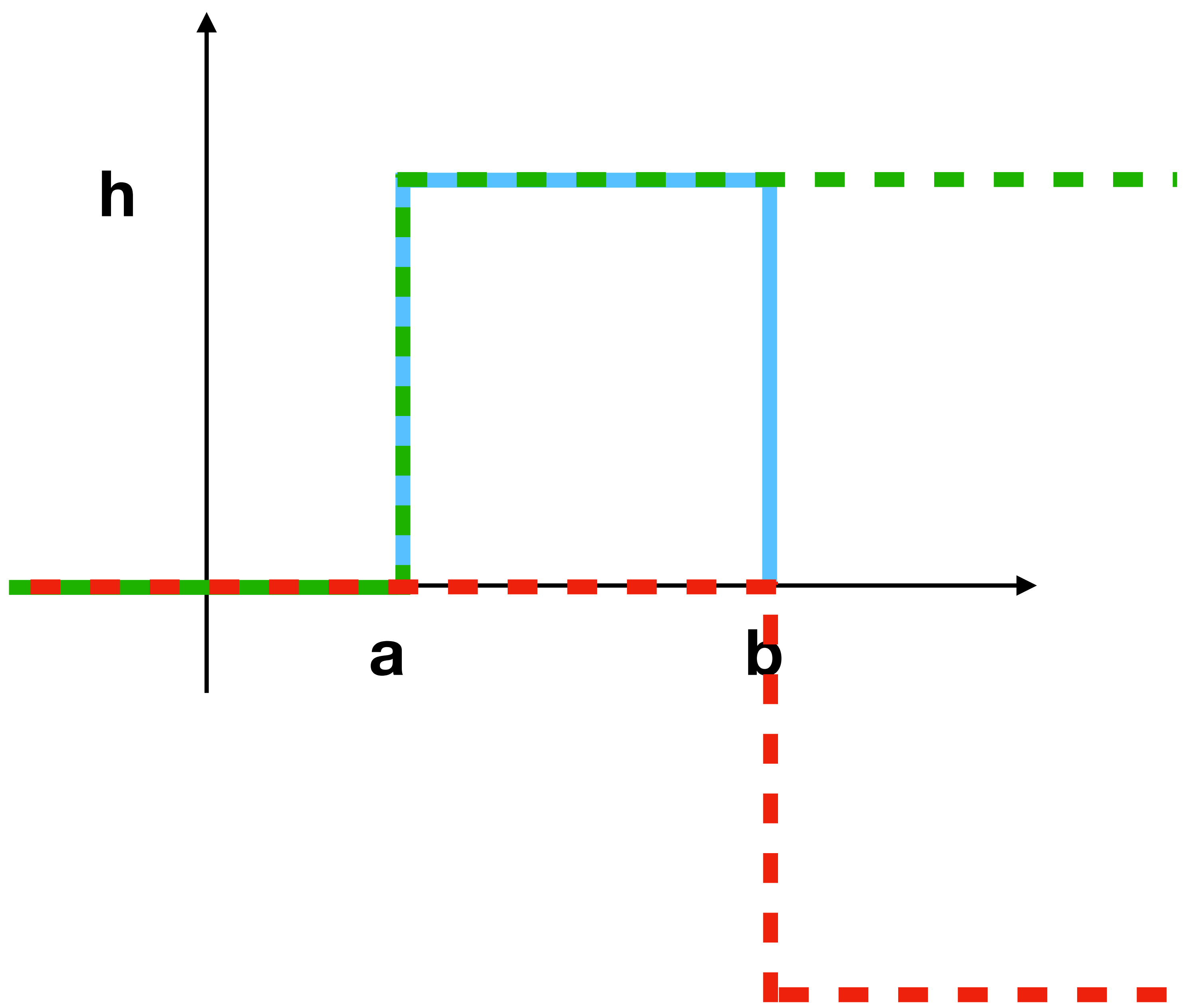
$f(x) = h1_{x\in[a,b]}$



$g(x) = h1_{x\geq a}$



$h(x) = -h1_{x\geq b}$



$f(x) = g(x) + h(x)$

Approximate a step function with a sigmoid

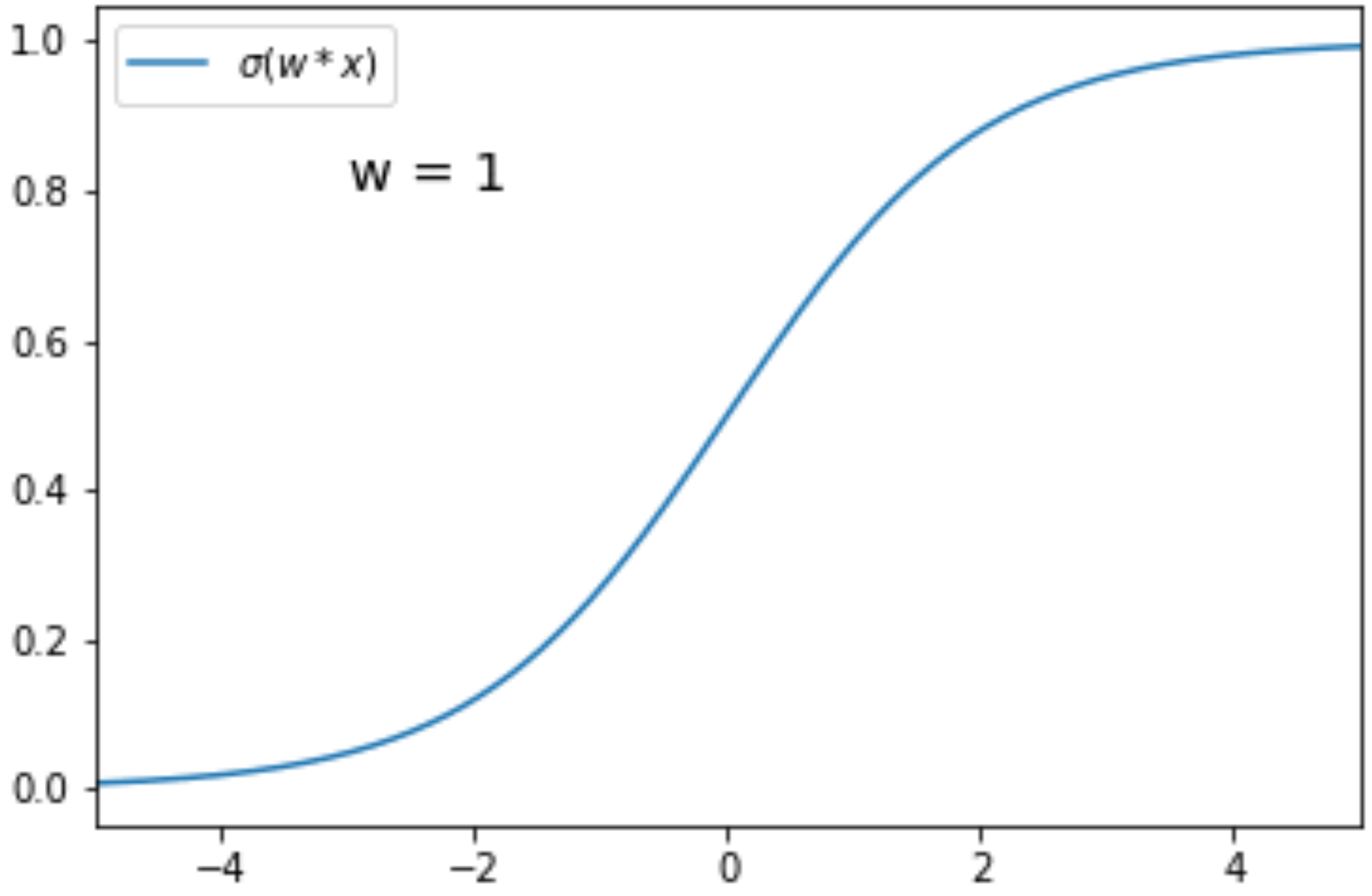
$$\tilde{\phi}(x) = \phi\big(w(x - b)\big)$$

By setting:

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

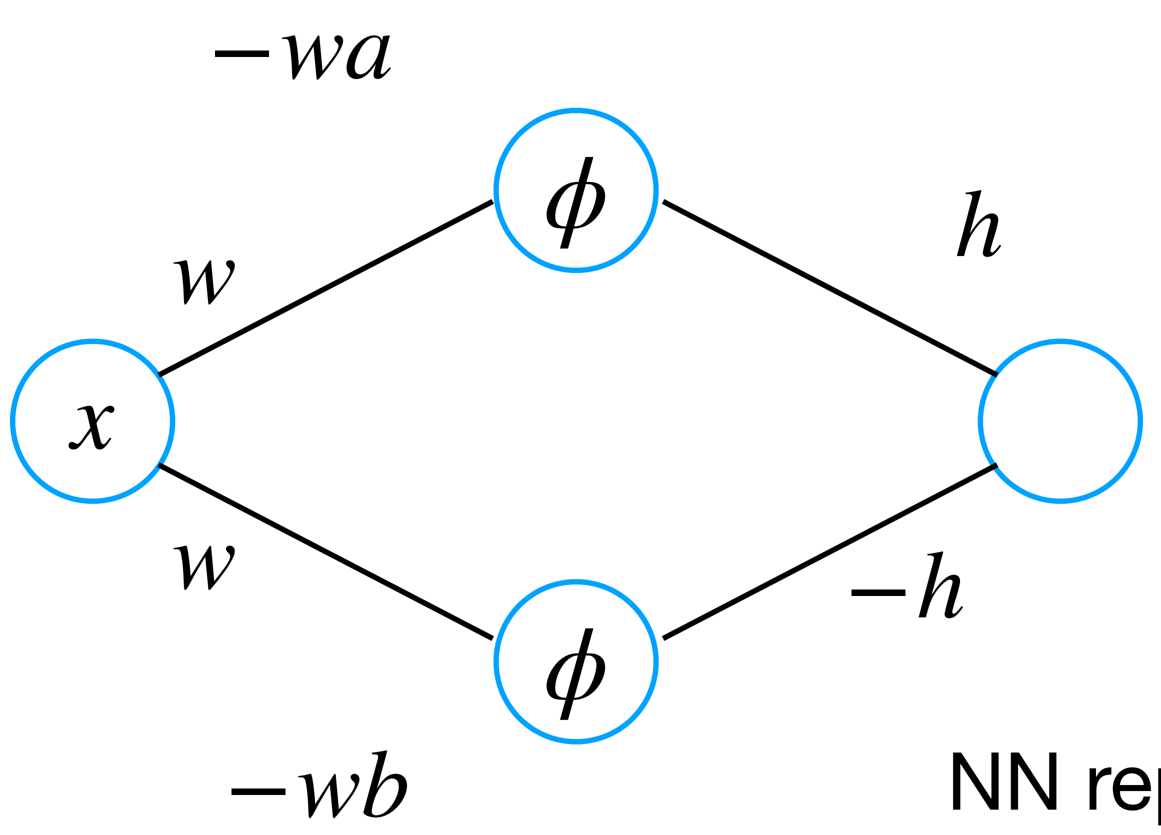
Derivative: $\tilde{\phi}'(b) = w/4$

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

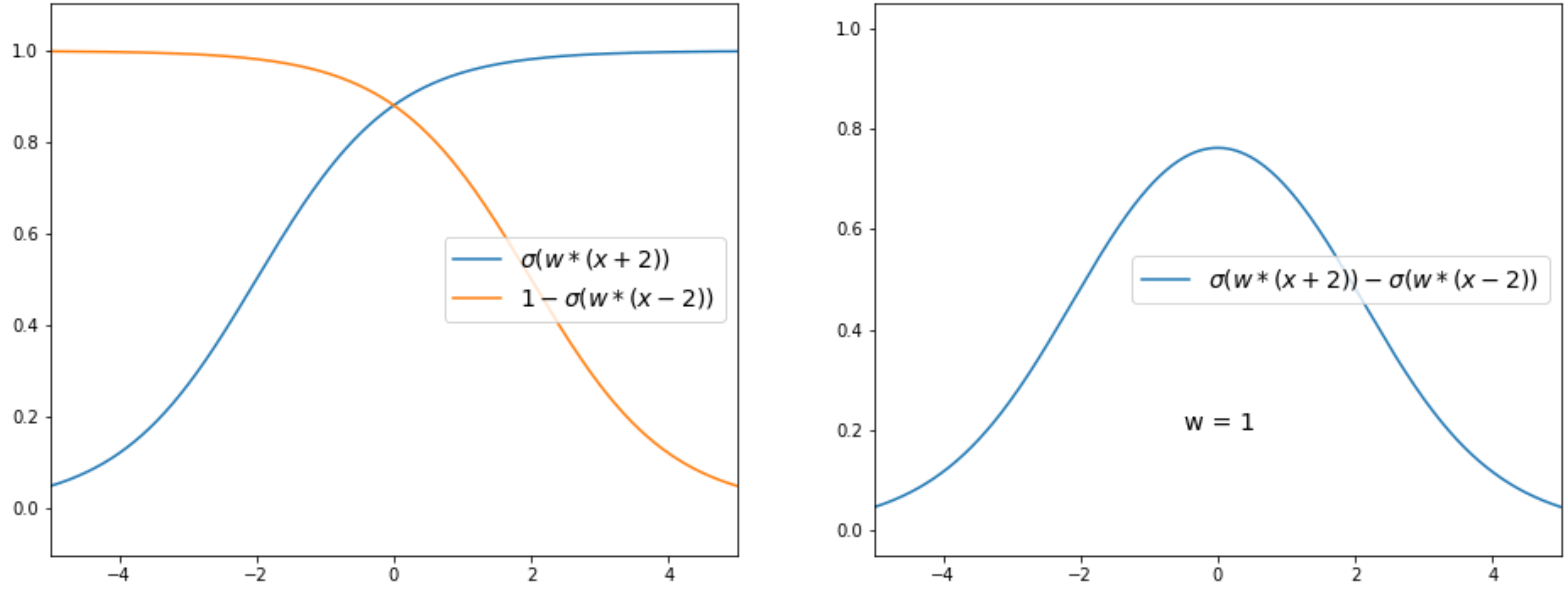


Approximation of the rectangle

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



NN representation



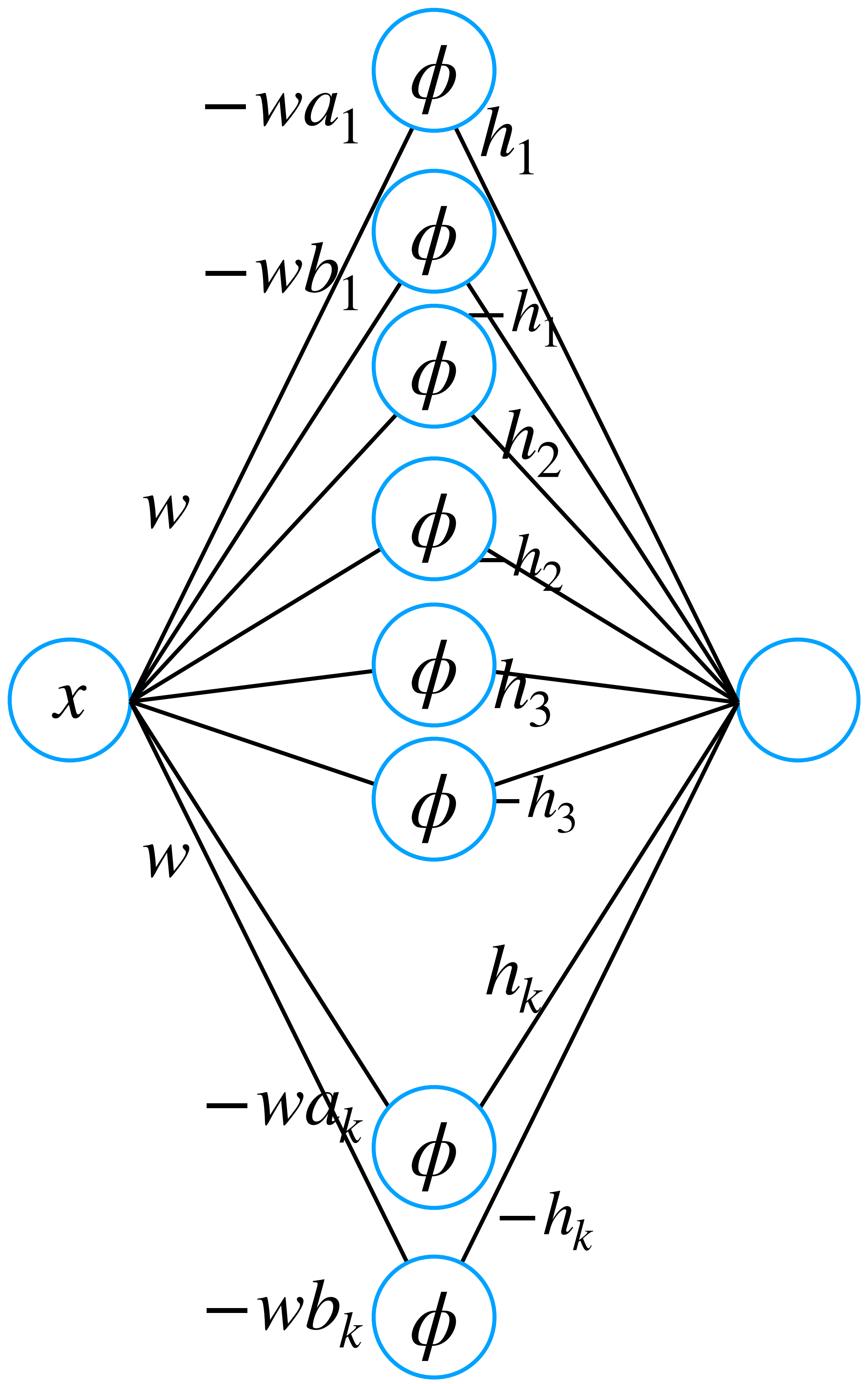
Conclusion in the 1D case

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

<u>Rmk:</u>

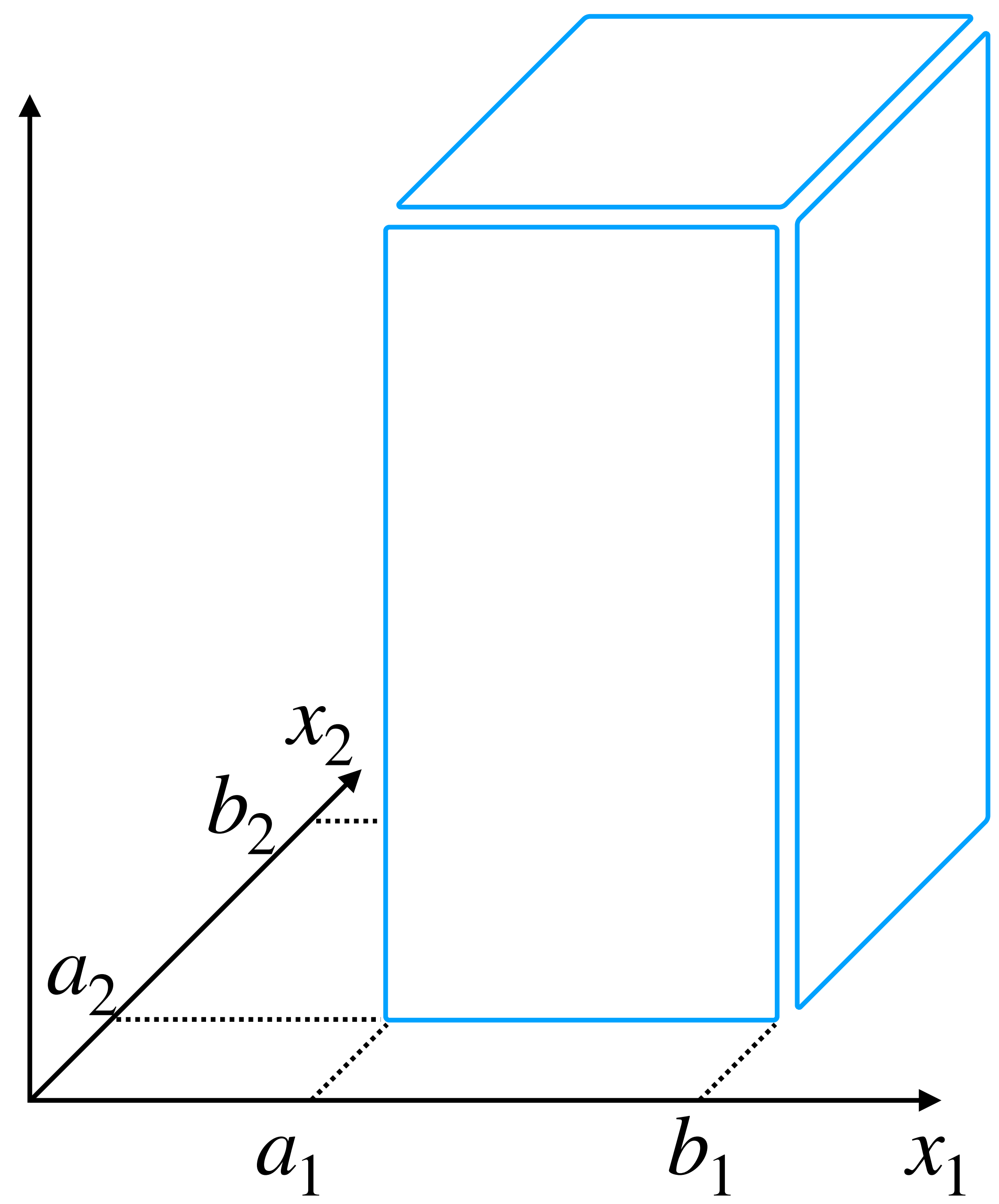
- Same intuition hold for any sigmoid like function
- Only intuition, not quantitative
- Need the weights w to be large



Larger dimension: $d = 2$

Same idea:

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

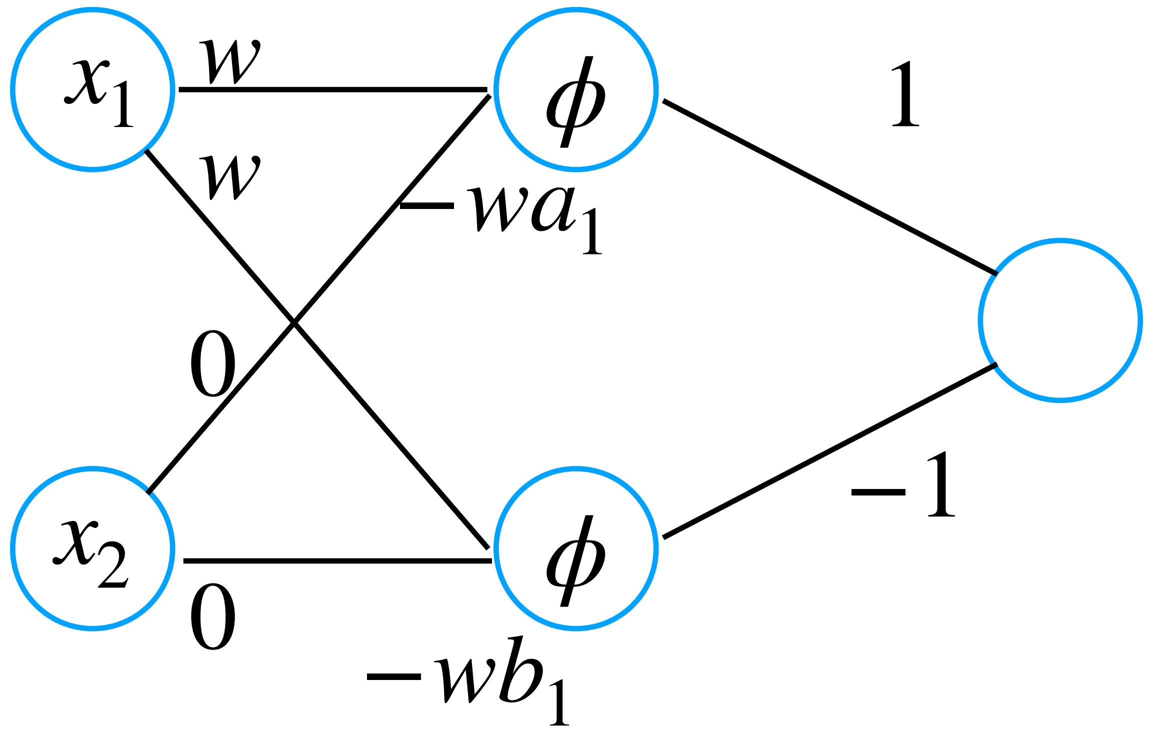


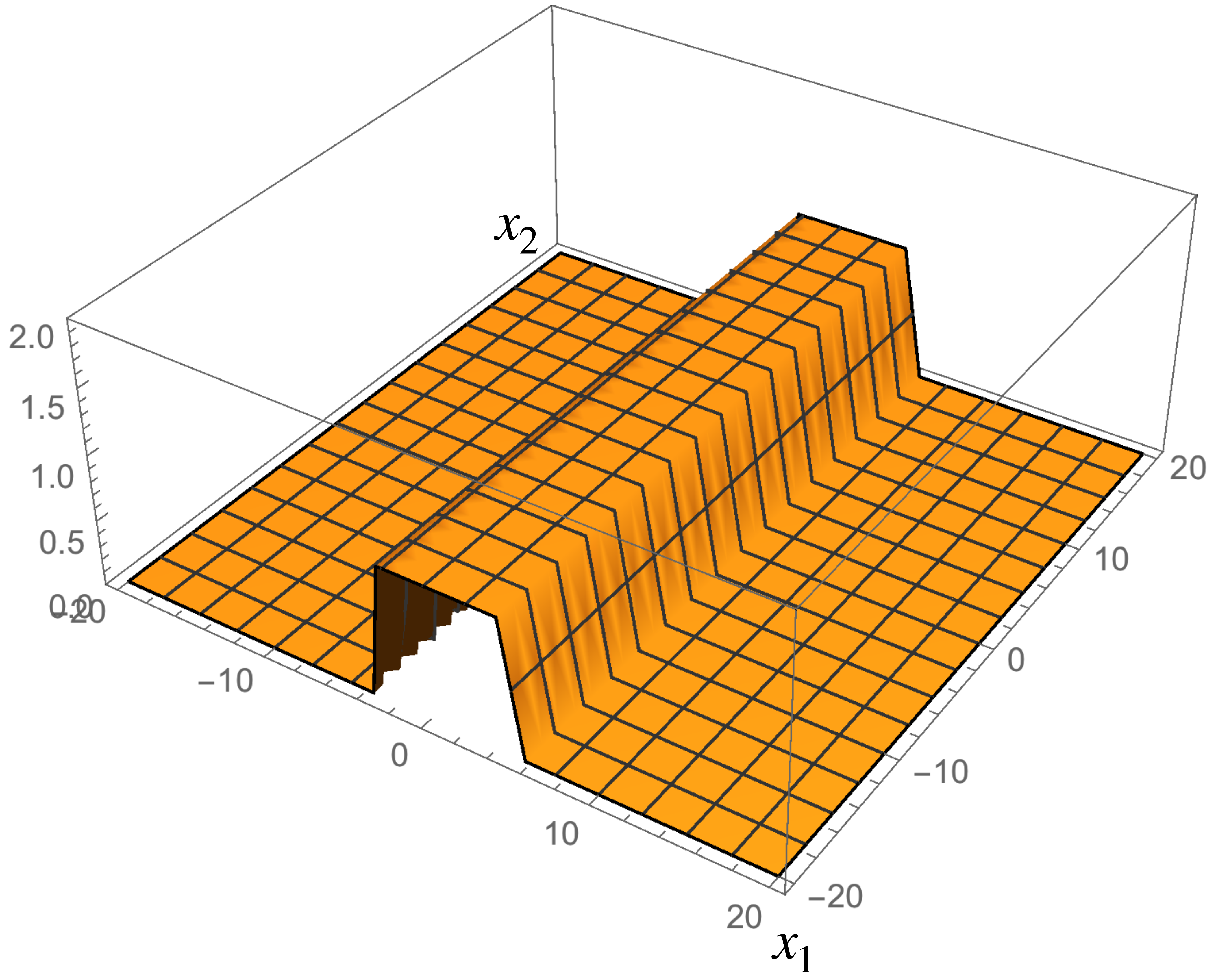
Two sigmoids can approximate an infinite rectangle

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

Rectangle:

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



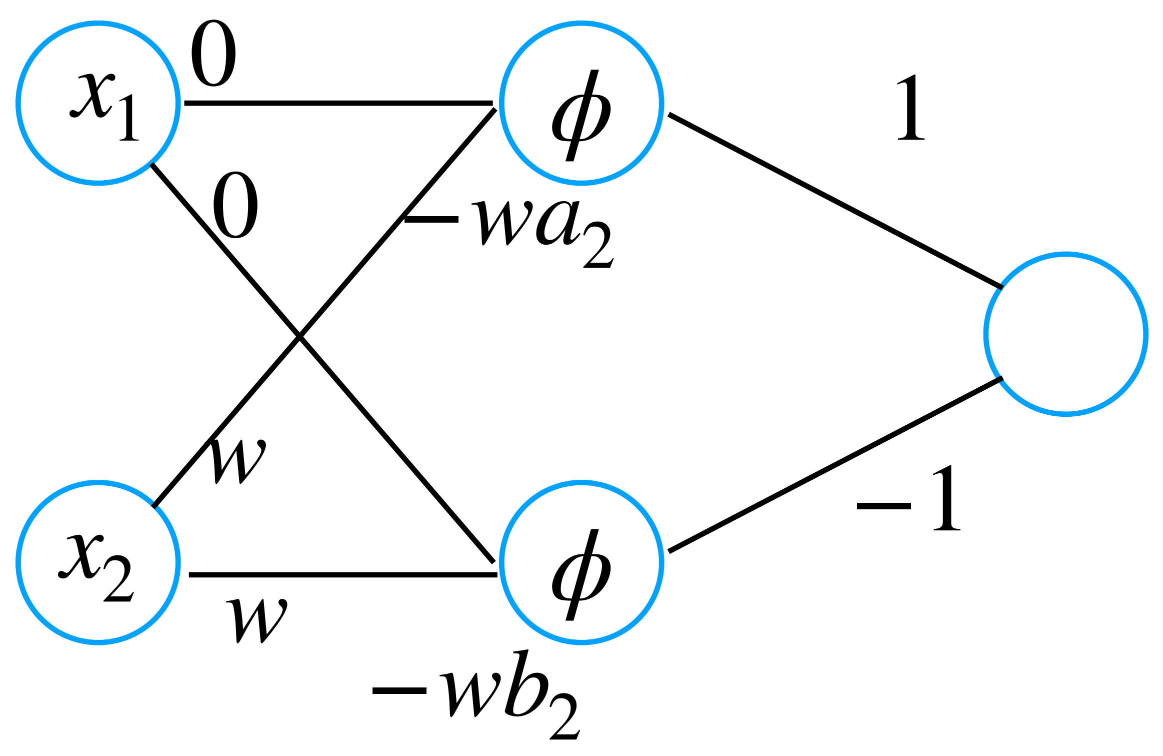


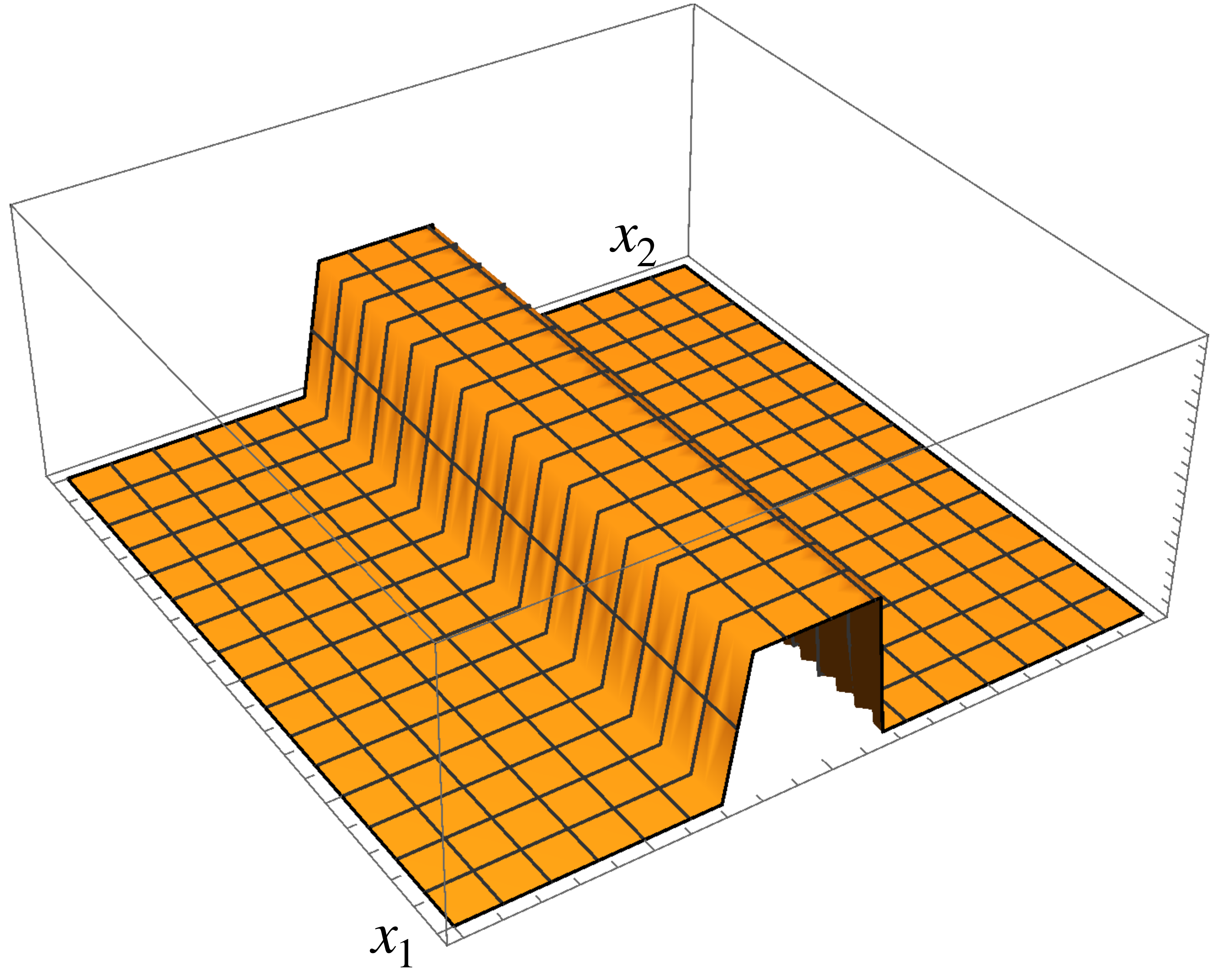
Two sigmoids can approximate an infinite rectangle

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

Rectangle:

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

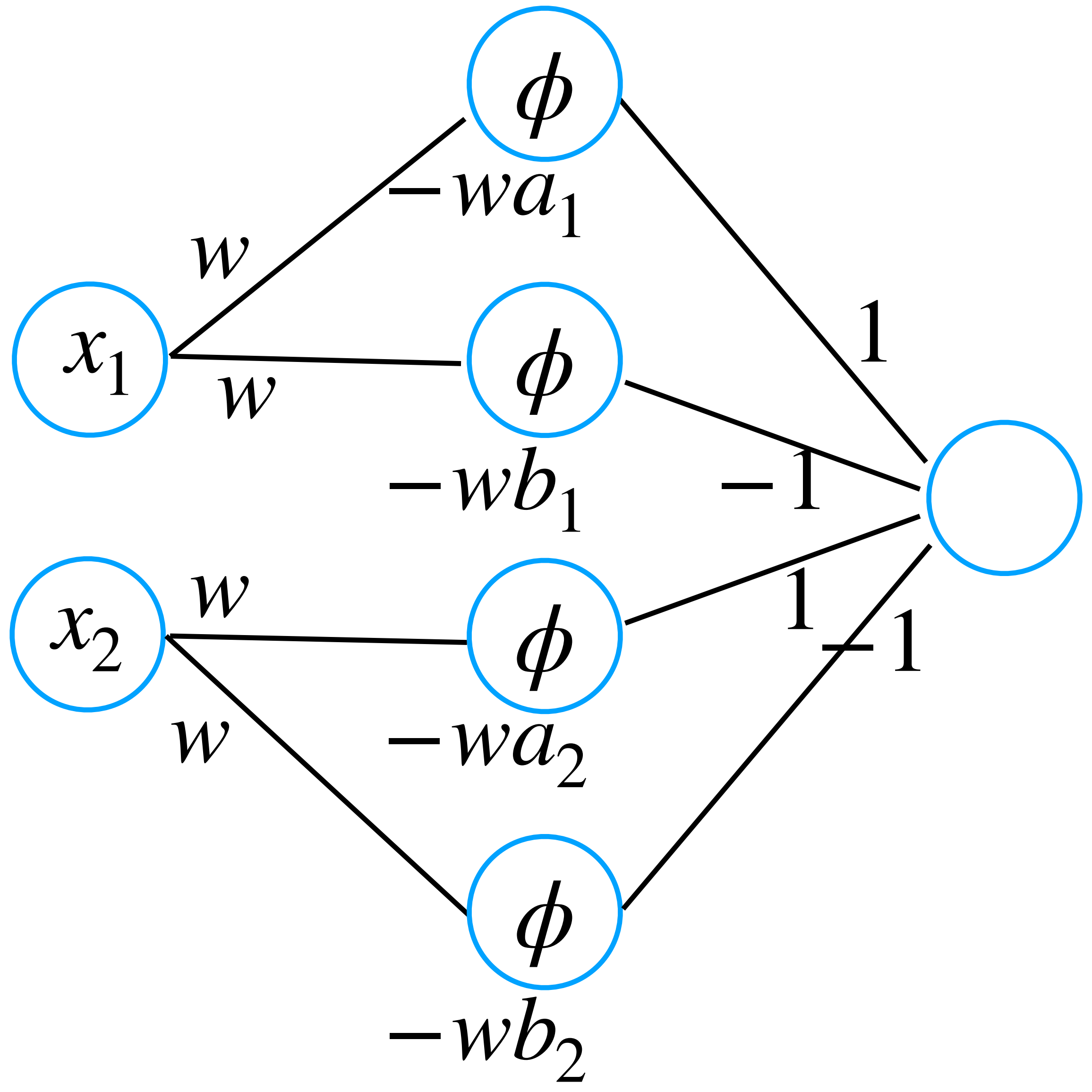


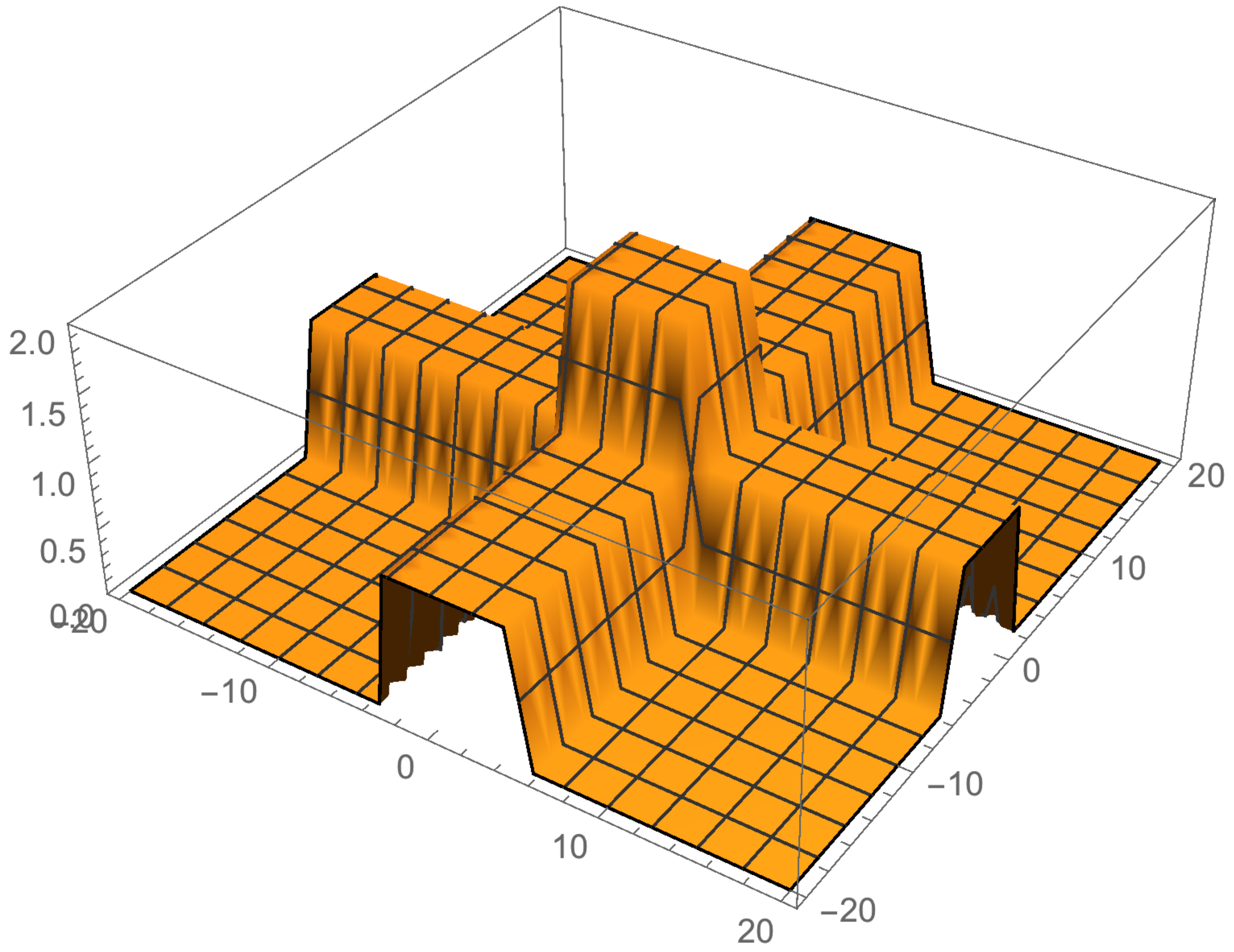


Four sigmoids can approximate a cross

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

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





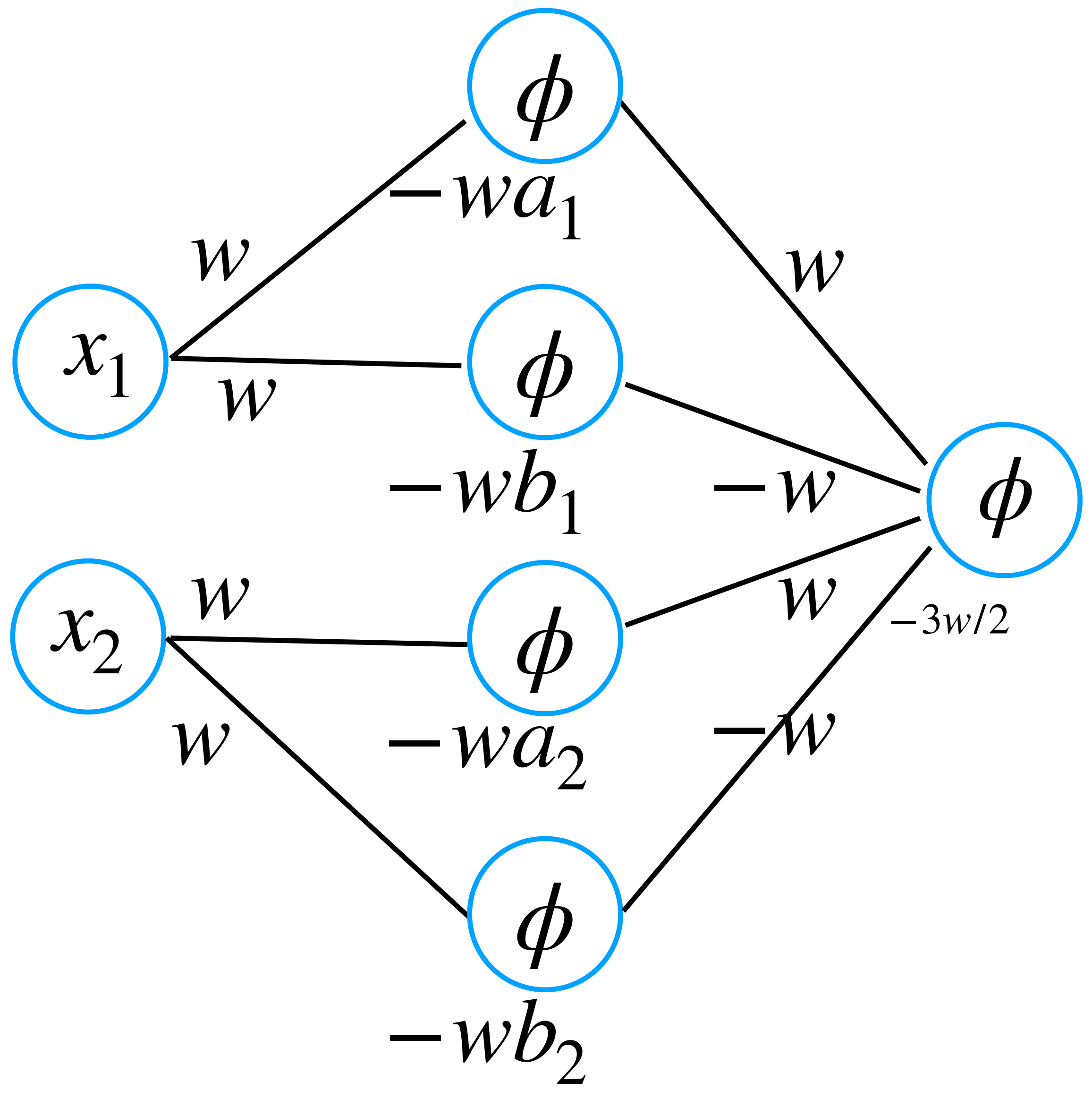
How to get rid of the crossed arms?

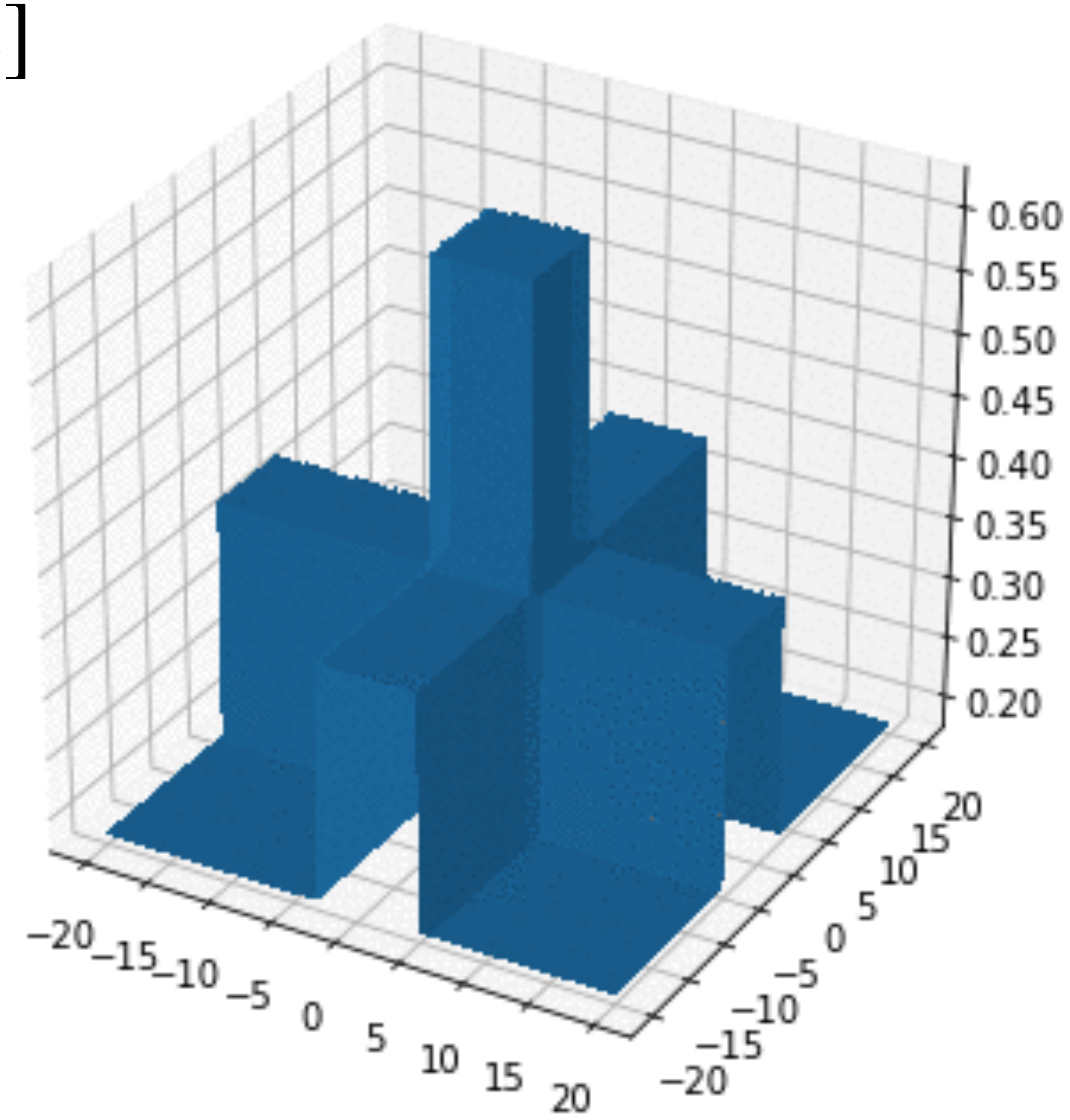
The sigmoid can threshold the unwanted infinite arms

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

Let f be a continuous function on $[0,1]$

Different ℓ_∞-approximation results exist:

- Polynomials: Stone Weierstrass theorem:

$\forall \varepsilon > 0,\ \exists p \in \mathbb{R}[X],$

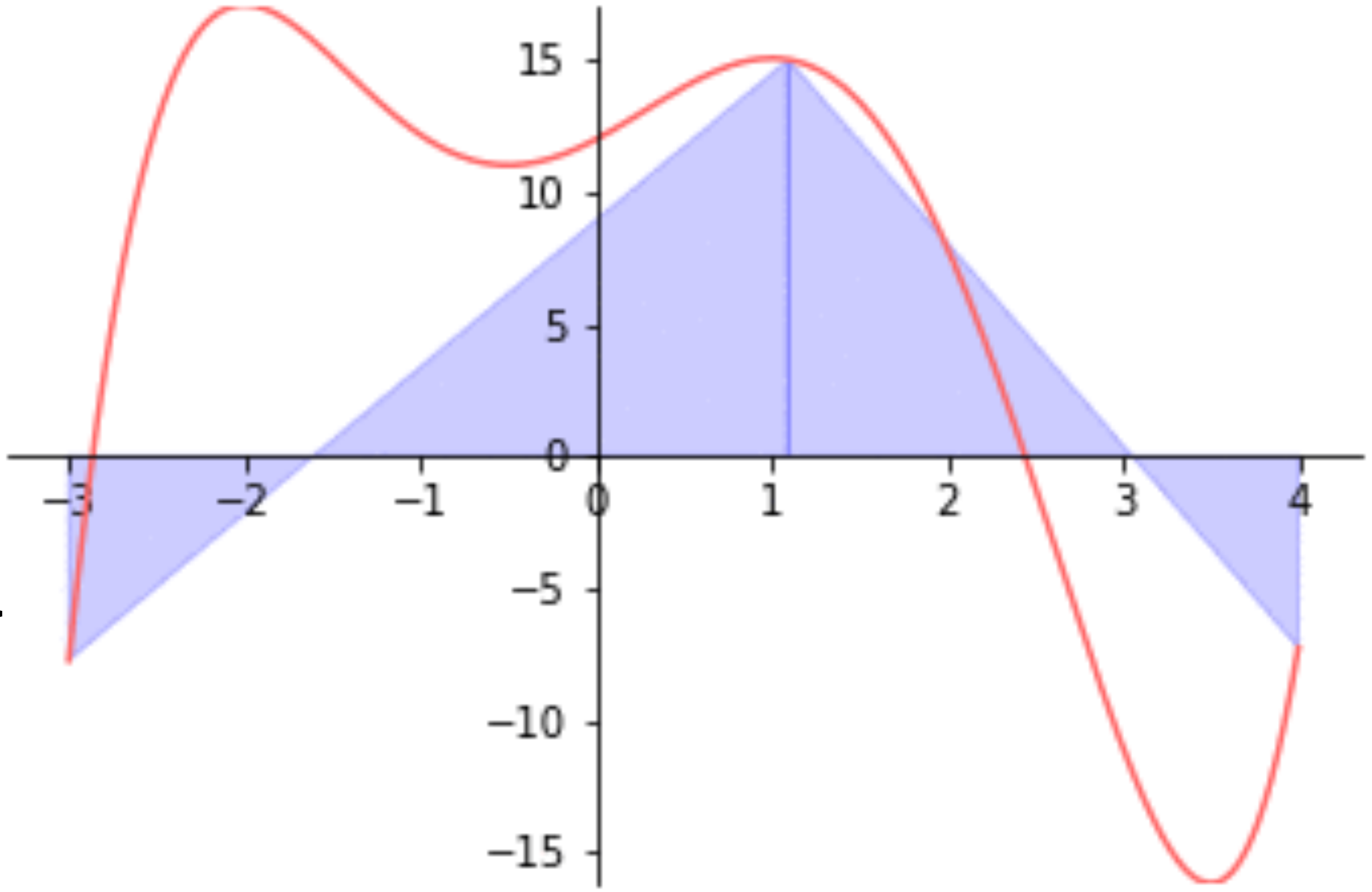
$$\sup_{x\in[0,1]} |f(x) - p(x)| \leq \varepsilon$$

- Piecewise linear function (Shektman, 1982}

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

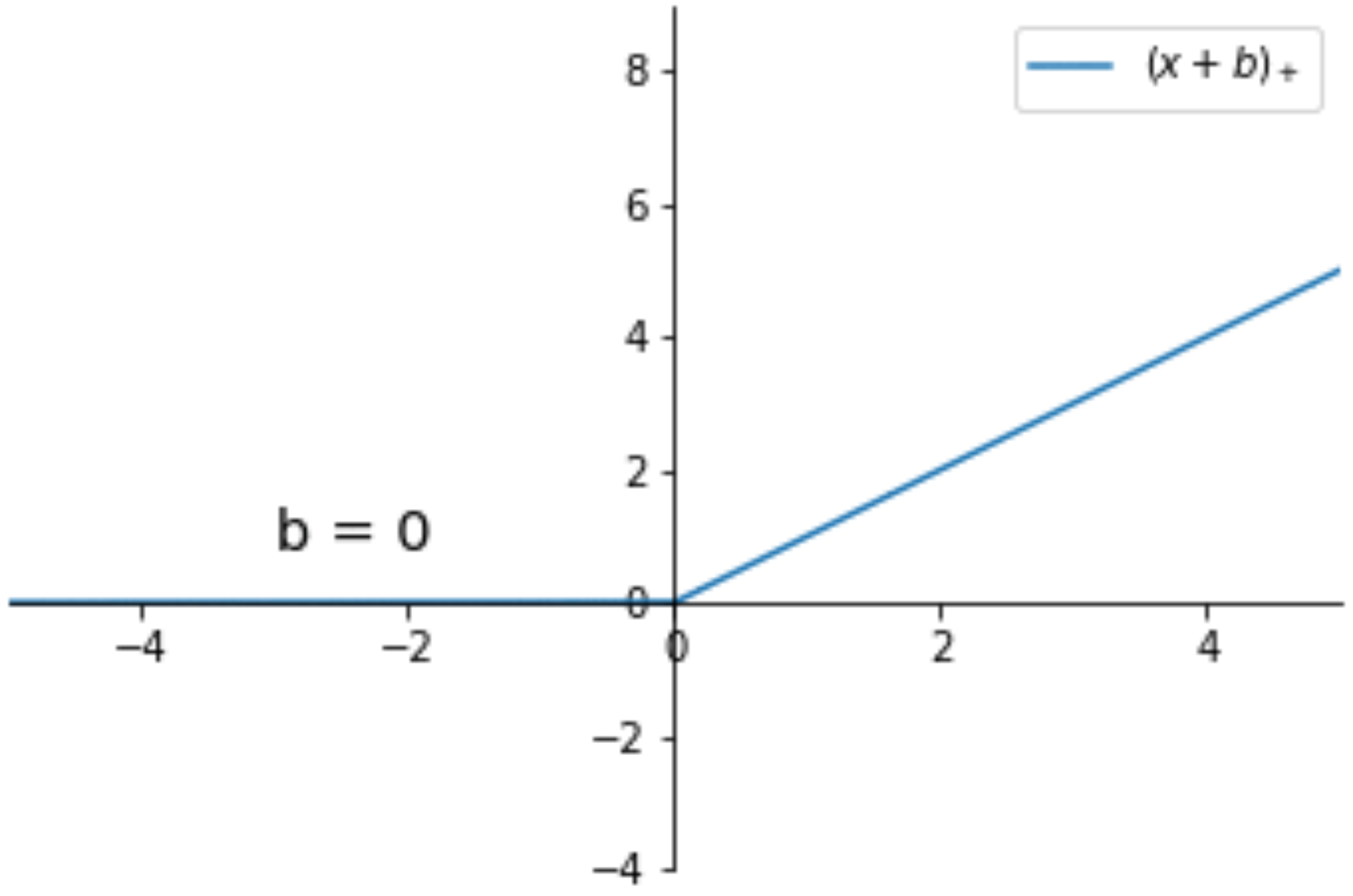
with $a_i r_i + b_i = a_{i+1} r_i + b_{i+1}$

➡ How to approximate such functions with a NN?

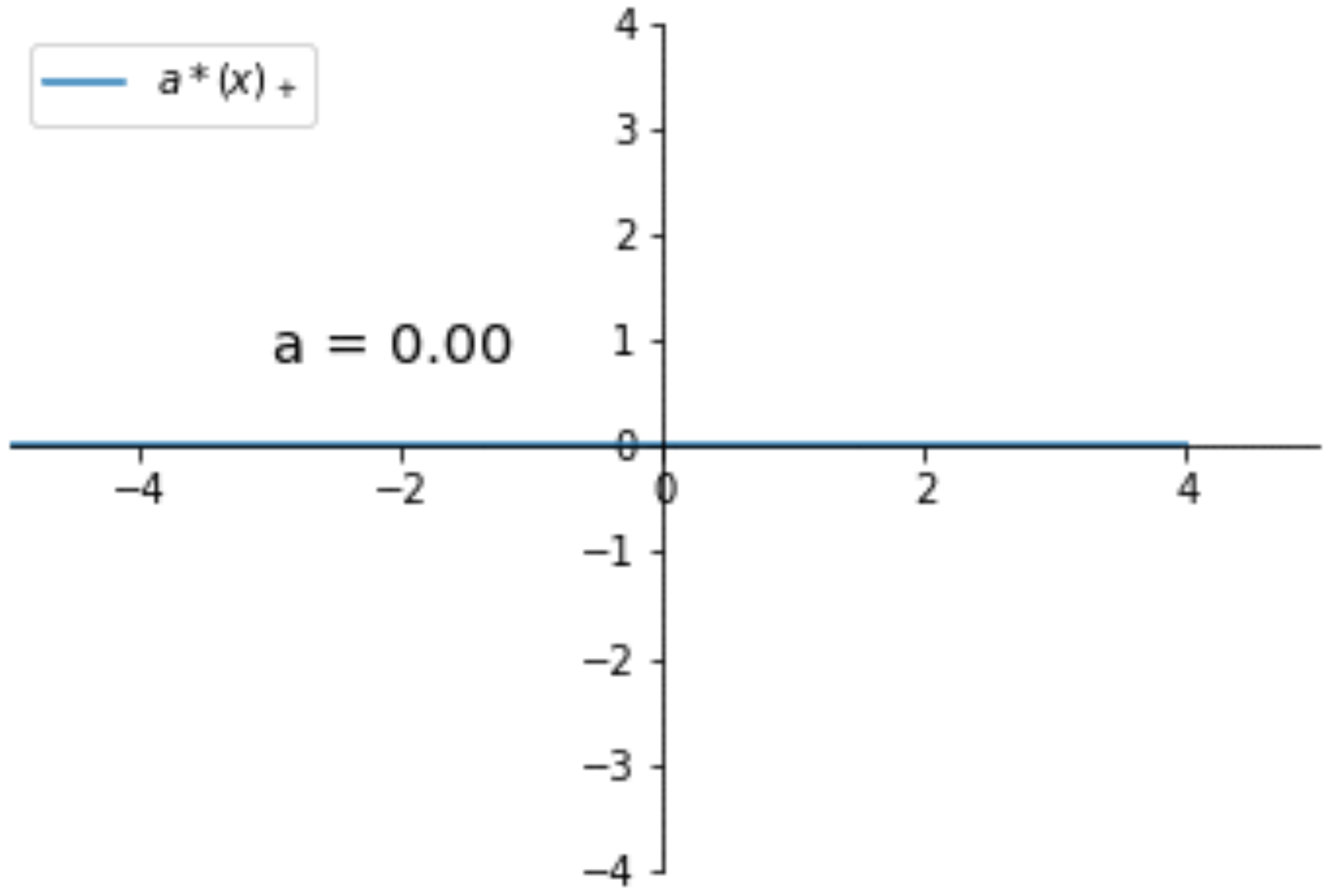


Rectified linear unit - RELU

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



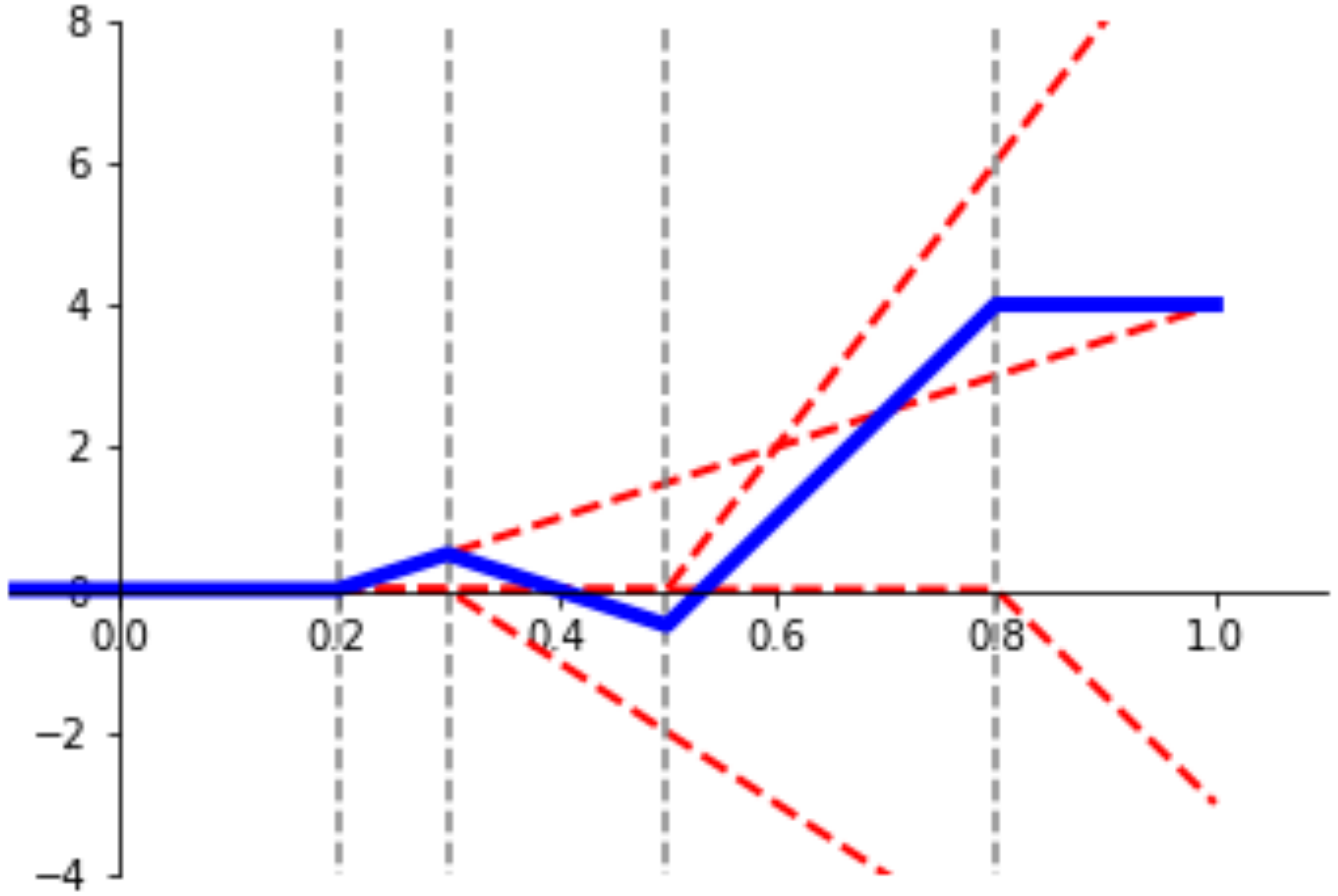
The bias b determines where the kink is

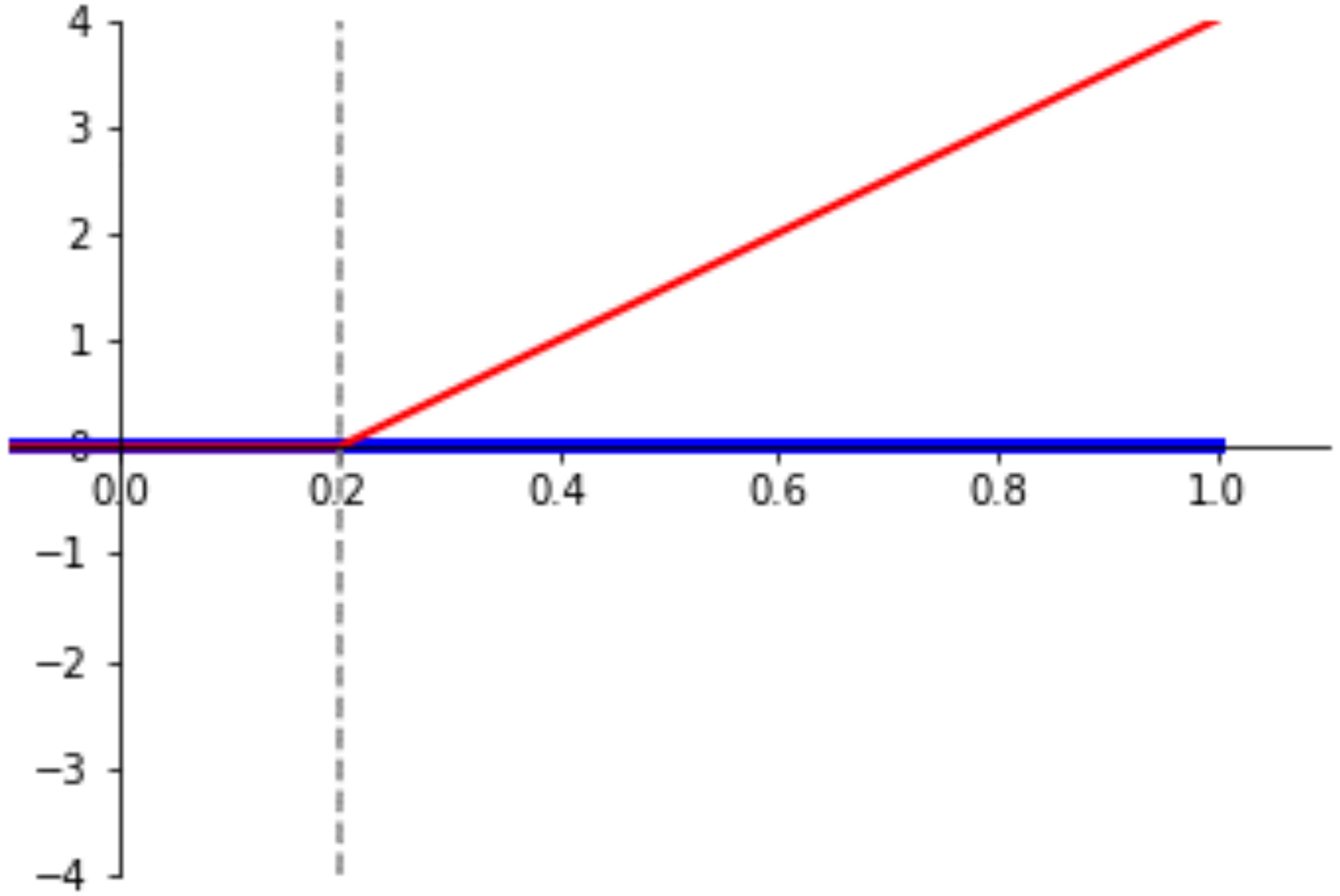


The weight a determines the slope

Linear combinations of RELUs are piecewise linear functions

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





Piecewise linear functions can be written as combination of RELU

<u>Claim 1</u>: q can be rewritten as

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

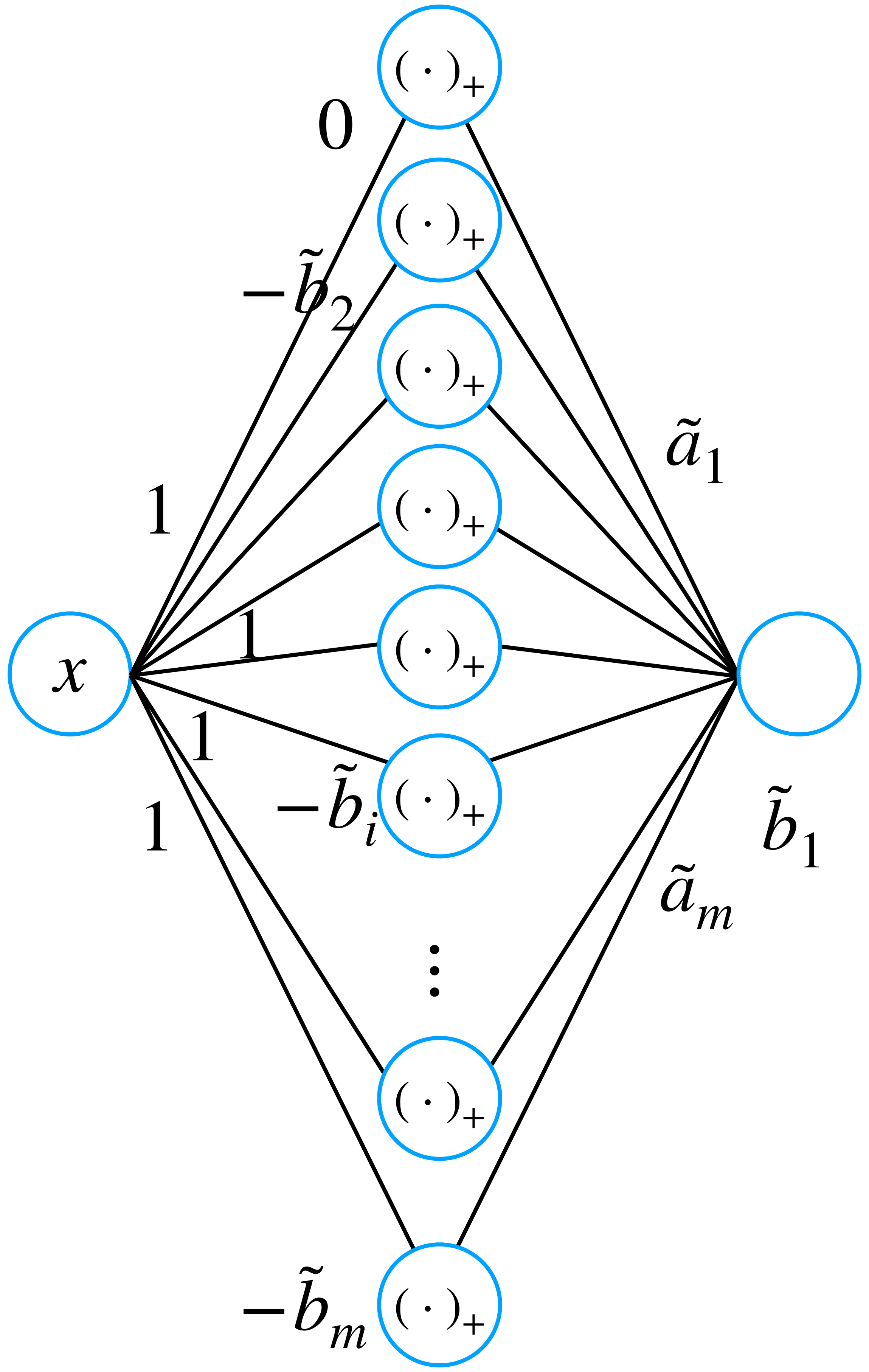
where $\tilde{a}_1 = a_1, \tilde{b}_1 = b_1,\ a_i = \sum_{j=1}^{i} \tilde{a}_i$ and $\tilde{b}_i = r_{i-1}$

<u>Claim 2</u>: 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

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

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

- 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 $x \in [r_1, r_2]$, $r(x) = \tilde{a}_1 x + \tilde{b}_1 + (a_2 - a_1)(x - r_1)_+$

  $$= a_1 x + b_1 + (a_2 - a_1)(x - r_1) = a_2 x + b_1 - (a_2 - a_1) r_1$$

  $r'(x) = a_2$ and $r(r_1) = q(r_1)$ as shown above

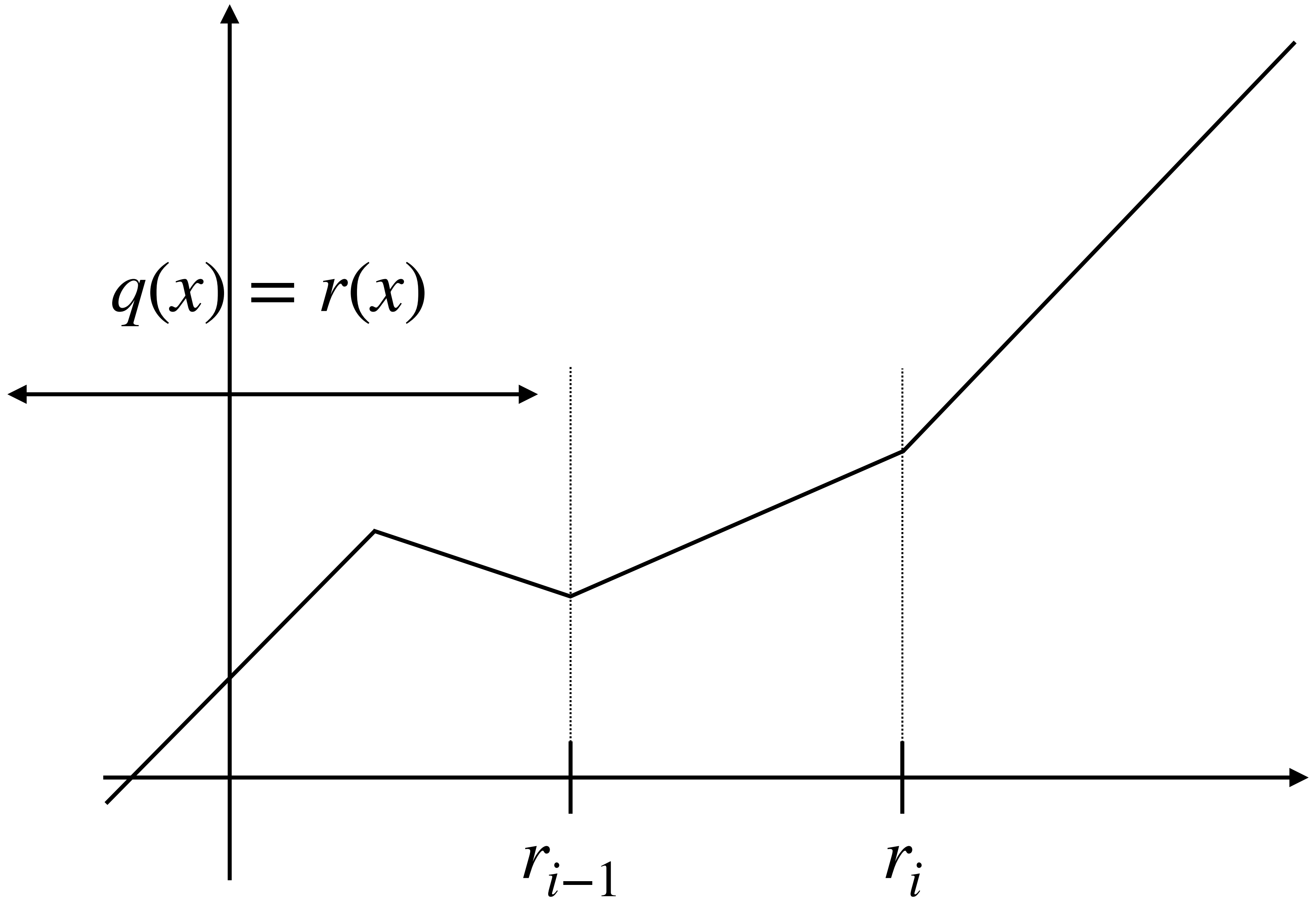
  $\implies r(x) = q(x)$ for $x \in [r_1, r_2]$

Proof by induction

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

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

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



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

$\implies r(x) = q(x)$ for $x \in [r_{i-1}, r_1]$

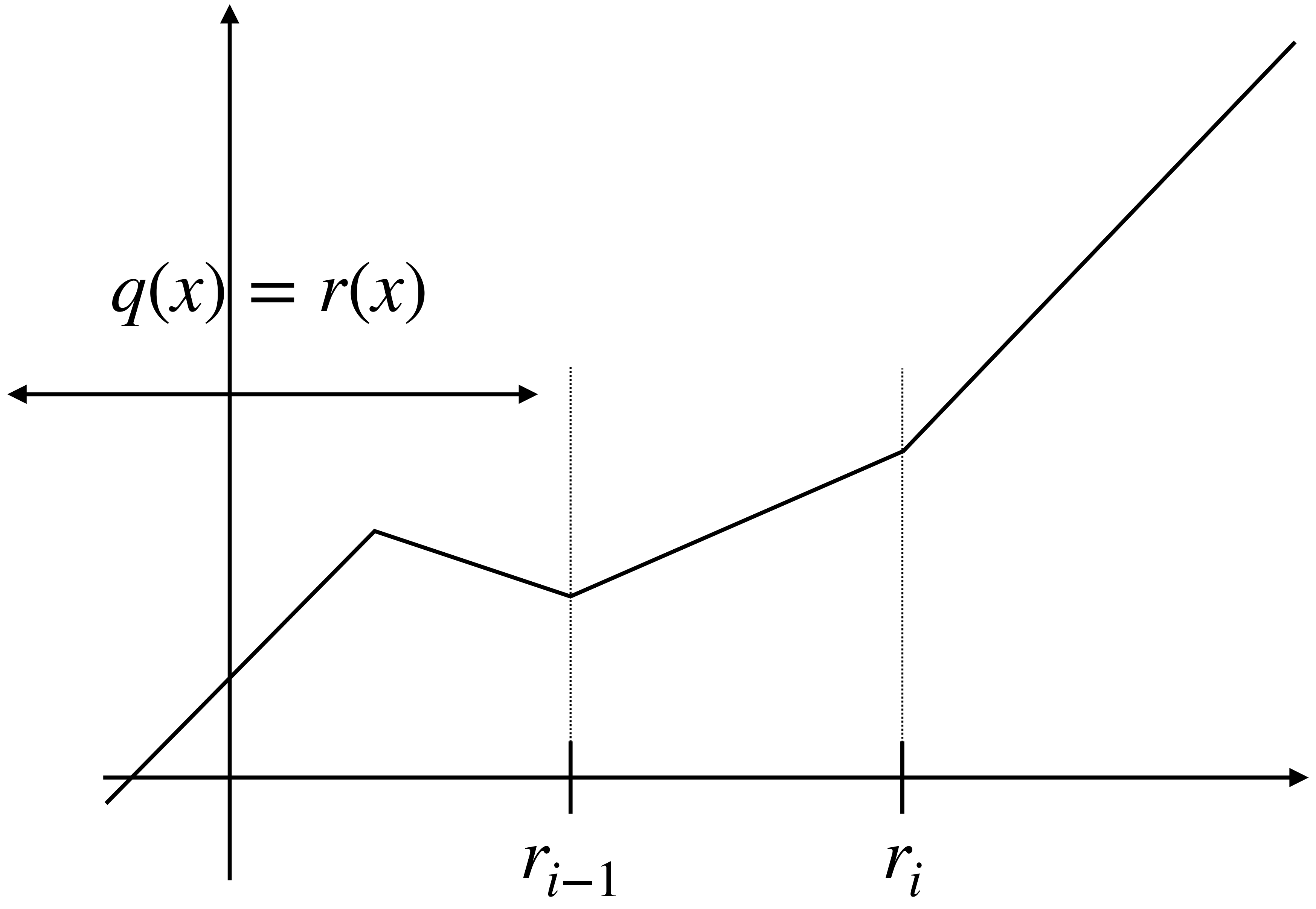
Why: two affine functions with the same starting point and the same slope are equal

Proof by induction - bis

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

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

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



Thus

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

$\implies r(x) = q(x)$ for $x \in [r_{i-1}, r_1]$

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