Neural Networks Training, SGD and Backpropagation

Machine Learning Course - CS-433

Nov 9, 2022

Nicolas Flammarion

EPFL

Recap

NNs: Key Facts

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

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

Linear prediction (with augmented features): $y = f_{\text{Lin}}(x) = \phi(x)^\top w$

Features are given

Prediction with a NN:

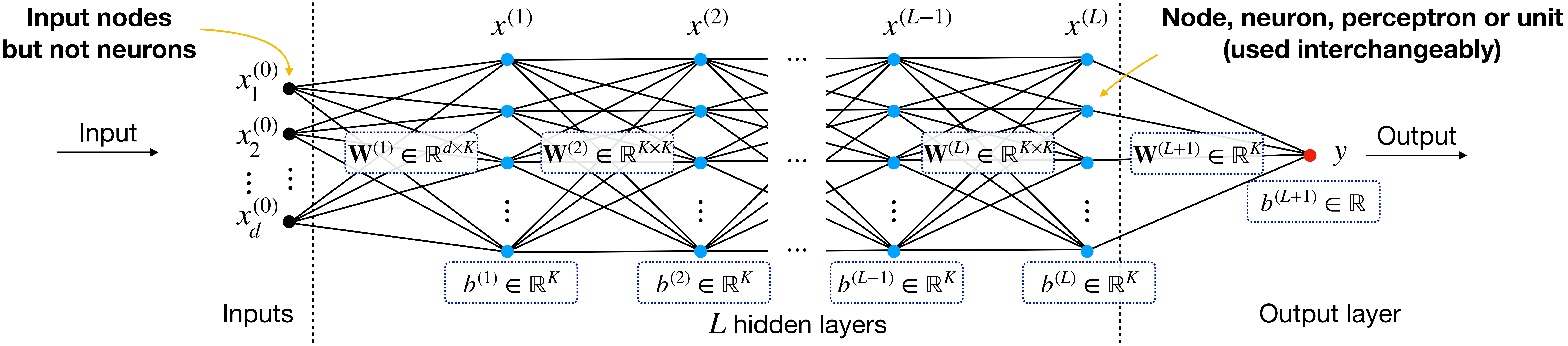
$$y = f_{\text{NN}}(x) = f(x)^\top w$$

Function implemented by the NN
parameters: weights and biases

Last layer is performing
a linear prediction

First layers transform the input
into a good representation

Fully Connected Neural Networks



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

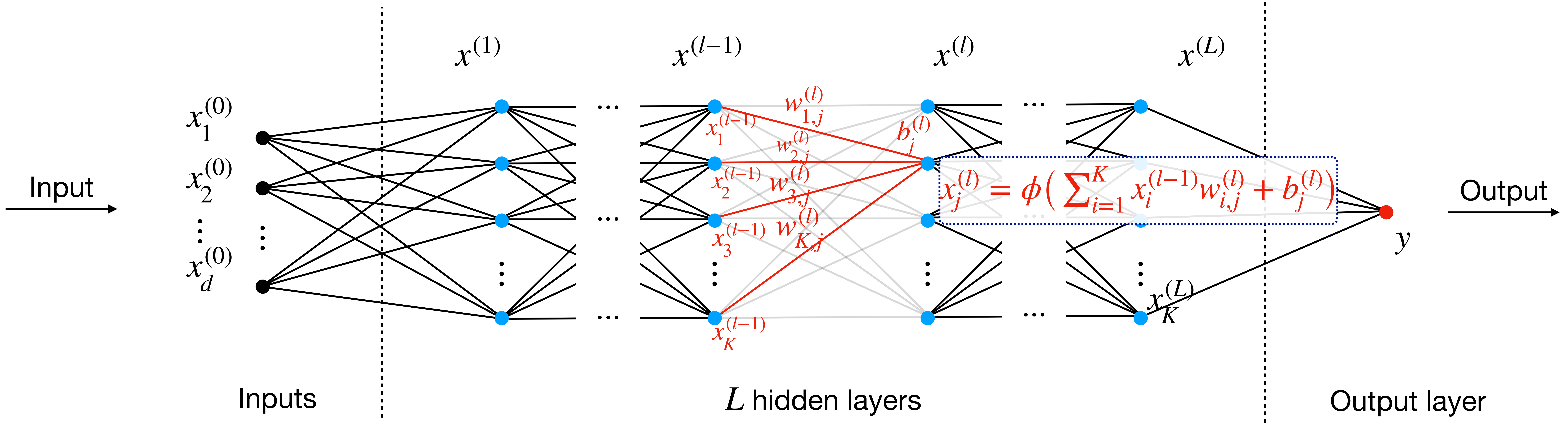
Outputs of hidden layer l given by vector: $x^{(l)} = f^{(l)}(x^{(l-1)}) := \phi\left((\mathbf{W}^{(l)})^\top x^{(l-1)} + b^{(l)}\right)$

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



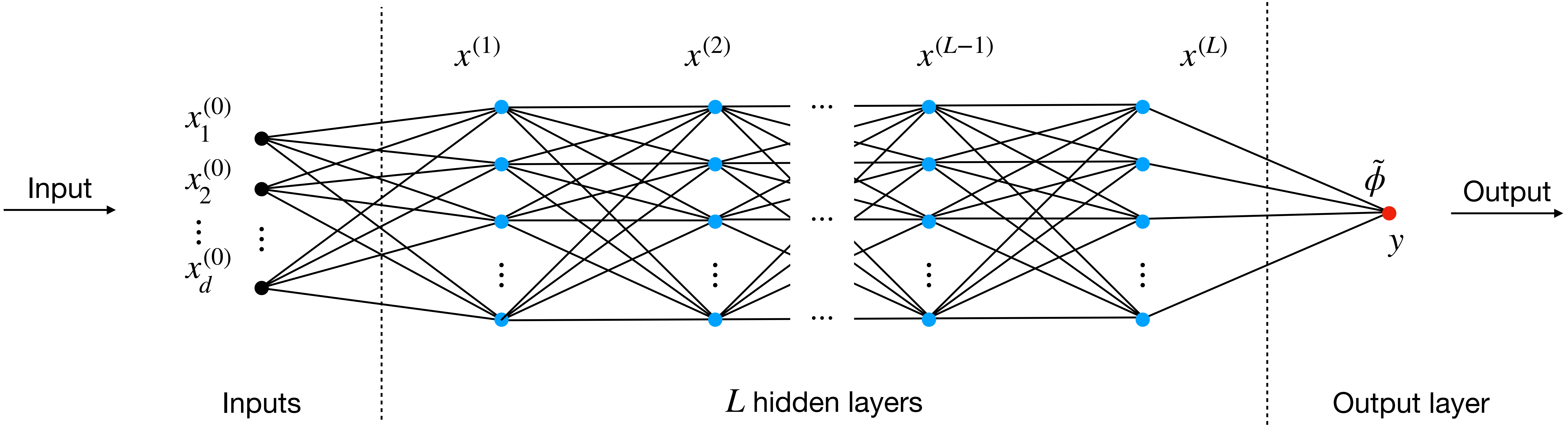
$$x_j^{(l)} = \phi\left(\sum_{i=1}^{K} x_i^{(l-1)} w_{i,j}^{(l)} + b_j^{(l)} \right)$$

Important: ϕ 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

The NN transforms the input into a more suitable representation then used to do linear predictions



Transformation of the data into
a suitable representation
It represents a function from $\mathbb{R}^d$ to $\mathbb{R}^K$

Linear prediction, e.g.,
linear regression,
logistic regression…

Representation power

- f smooth (condition on its Fourier coefficients)
- Bounded domain
- Depends on the activation function
- Average approximation in ℓ_2-norm but also point-wise approximation in ℓ_∞-norm

Today: How do we train a NN?

Training of NNs

Training loss for a regression problem with $S_{\text{train}} = \{(x_n, y_n)\}_{n=1}^{N}$:

$$L(f) = \frac{1}{2N} \sum_{n=1}^{N} \left(y_n - f(x_n)\right)^2$$

where f is the function represented by a NN with weights $\left(w_{i,j}^{(l)}\right)$ and biases $\left(b_i^{(l)}\right)$

<u>Task</u>:

$$\min_{w_{i,j}^{(l)}, b_i^{(l)}} L(f)$$

<u>Remarks</u>:

- Regularization: can be added to avoid overfitting but easy to deal with
- Non convex optimization problem

  ➡ not guaranteed to converge to a global minimum

Training of NNs with SGD

<u>SGD algorithm:</u>

Sample uniformly n, compute the gradient of $L_n = \frac{1}{2}(y_n - f(x_n))^2$ to update:

$$(w_{i,j}^{(l)})_{t+1} = (w_{i,j}^{(l)})_t - \gamma \frac{\partial}{\partial w_{i,j}^{(l)}} L_n$$

$$(b_i^{(l)})_{t+1} = (b_i^{(l)})_t - \gamma \frac{\partial}{\partial b_i^{(l)}} L_n$$

<u>In Practice:</u> Step size schedule, mini-batch, momentum, Adam

<u>Problem:</u> $O(K^2 L)$ parameters: doing chain-rules independently won't be efficient

<u>Solution:</u> Backpropagation algorithm

Compact description of output

The functions implemented by each layer can be written as:

- $x^{(1)} = f^{(1)}(x^{(0)}) := \phi\Big((\mathbf{W}^{(1)})^\top x^{(0)} + b^{(1)}\Big)$
- $x^{(2)} = f^{(2)}(x^{(1)}) := \phi\Big((\mathbf{W}^{(2)})^\top x^{(1)} + b^{(2)}\Big)$

$\vdots$
- $x^{(l)} = f^{(l)}(x^{(l-1)}) := \phi\Big((\mathbf{W}^{(l)})^\top x^{(l-1)} + b^{(l)}\Big)$

$\vdots$
- $y = f^{(L+1)}(x^{(L)}) := \tilde{\phi}\Big((\mathbf{W}^{(L+1)})^\top x^{(L)} + b^{(L+1)}\Big)$

The overall function $y = f(x^{(0)})$ is just the composition of these functions:

$$\boxed{f = f^{(L+1)} \circ f^{(L)} \circ \cdots \circ f^{(l)} \circ \cdots \circ f^{(2)} \circ f^{(1)}}$$

Cost function

Cost function:

$$L = \frac{1}{2N} \sum_{n=1}^{N} \left(y_n - f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x_n)\right)^2$$

Remarks:

- The specific form of the loss does not matter
- Function of all weight matrices and bias vectors

Individual loss for SGD:

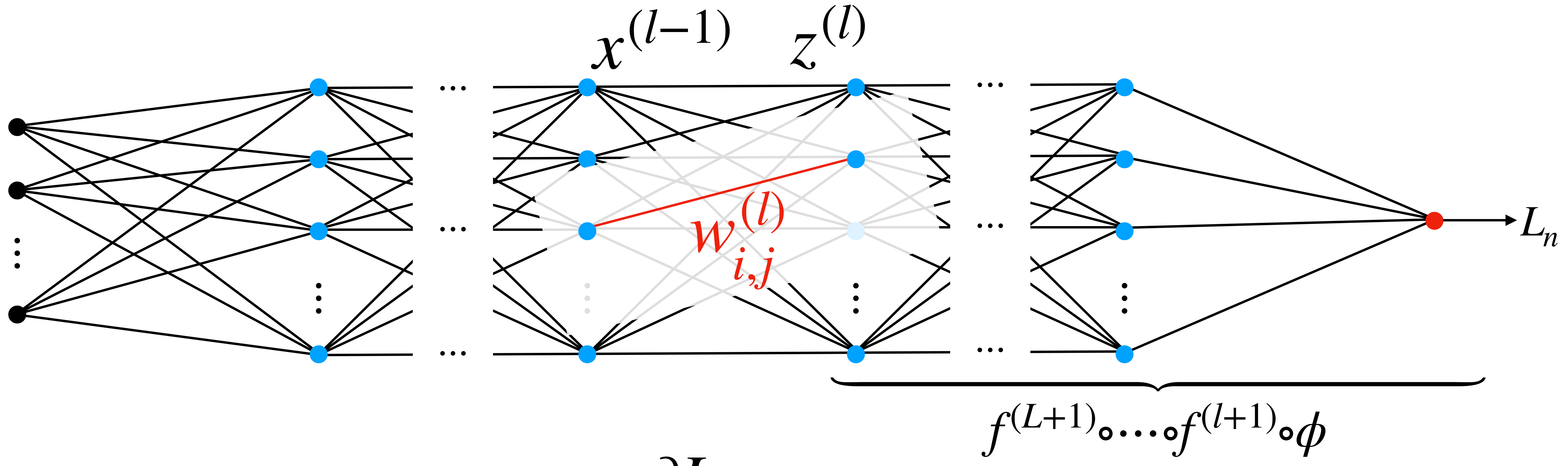
$$L_n = \frac{1}{2}\left(y_n - f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x_n)\right)^2$$

Goal: Compute for all (i, j, l)

$$\frac{\partial L_n}{\partial w_{i,j}^{(l)}} \quad \text{and} \quad \frac{\partial L_n}{\partial b_i^{(l)}}$$

Naive approach

$$L_n = \frac{1}{2}\Big(y_n - f^{(L+1)} \circ \cdots \circ f^{(l+1)} \circ \phi\big(\underbrace{(\mathbf{W}^{(l)})^\top x^{(l-1)} + b^{(l)}}_{z^{(l)}}\big)\Big)^2$$

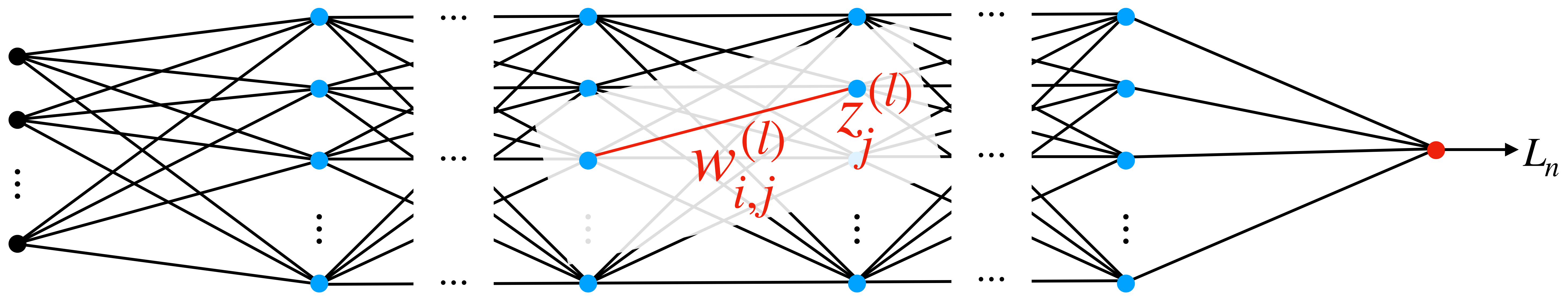


$$\frac{\partial L_n}{\partial w_{i,j}^{(l)}} \quad ?$$

Naive approach

$$L_n = \frac{1}{2}\left(y_n - f^{(L+1)} \circ \cdots \circ f^{(l+1)} \circ \phi\left(z^{(l)}\right)\right)^2$$

$$x^{(l-1)} \qquad z^{(l)} = (\mathbf{W}^{(l)})^\top x^{l-1} + b^{(l)}$$



<u>Chain rule</u>:

$$\frac{\partial L_n}{\partial w_{i,j}^{(l)}} = \sum_{k=1}^{K} \frac{\partial L_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}}$$

$$= \frac{\partial L_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}} \qquad \text{since } \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}} = 0 \text{ for } k \neq j$$

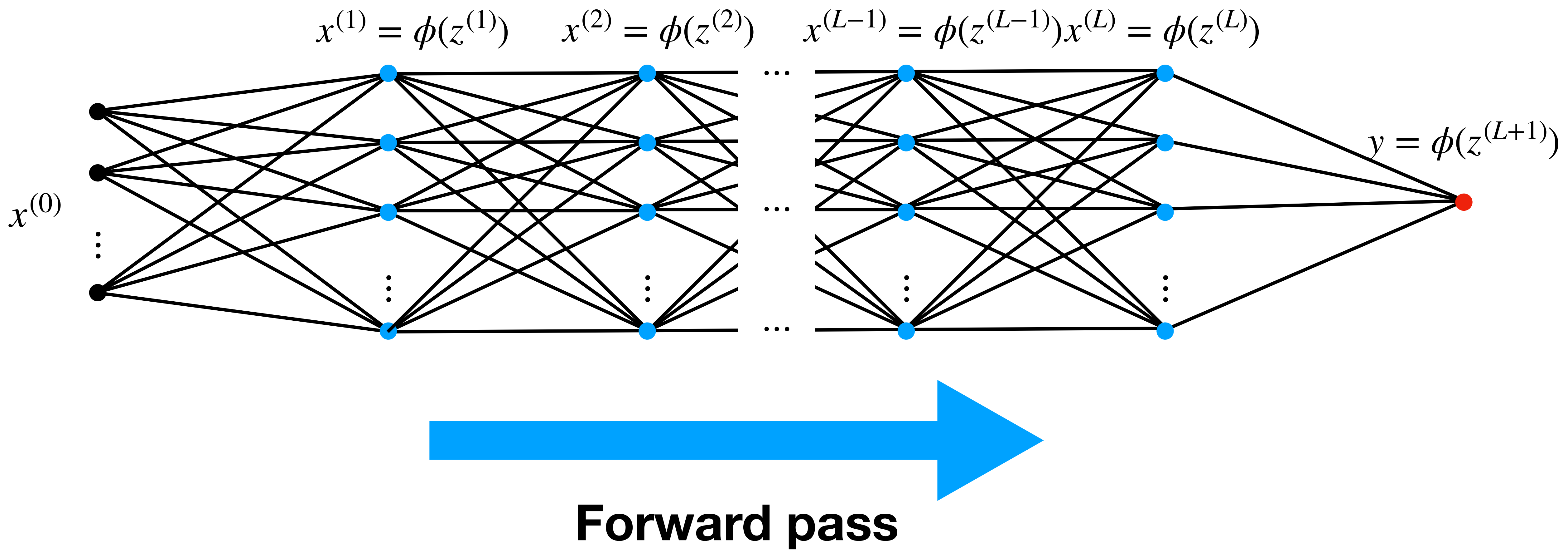
$$= \frac{\partial L_n}{\partial z_j^{(l)}} \cdot x_i^{(l-1)} \qquad \text{since } z_j^{(l)} = \sum_{k=1}^{K} w_{k,j}^{(l)} x_k^{(l-1)} + b_j^{(l)}$$

We need to compute $\frac{\partial L_n}{\partial z_j^{(l)}}$, $z^{(l)}$ and $x_i^{(l-1)}$

Forward Pass

We can compute $z^{(l)}$ and $x^{(l)}$ by a forward pass in the network:

$$x^{(0)} = x_n \in \mathbb{R}^d$$
$$z^{(l)} = (\mathbf{W}^{(l)})^\top x^{l-1} + b^{(l)}$$
$$x^{(l)} = \phi(z^{(l)})$$

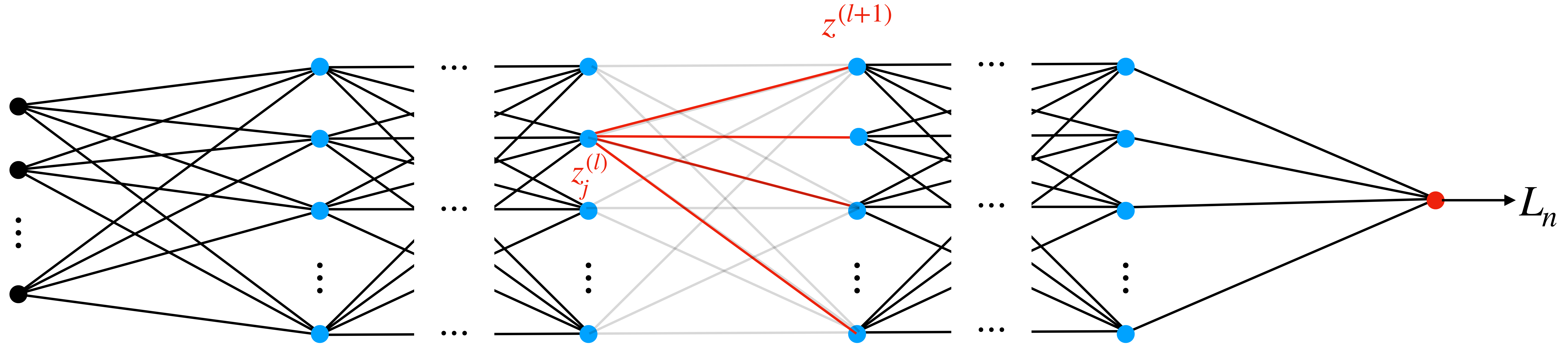


Computational complexity:

➡ one pass over the network $O(K^2L)$

Backward pass (I)

Define $\delta_j^{(l)} = \dfrac{\partial L_n}{\partial z_j^{(l)}}$



Chain rule:

$$\delta_j^{(l)} = \frac{\partial L_n}{\partial z_j^{(l)}} = \sum_k \frac{\partial L_n}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_k \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$

Backward pass (II)

Using $z_k^{(l+1)} = \sum_{i=1}^{K} w_{i,k}^{(l+1)} x_i^{(l)} + b_k^{(l+1)} = \sum_{i=1}^{K} w_{i,k}^{(l+1)} \phi(z_i^{(l)}) + b_k^{(l+1)}$

We obtain $\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \phi'(z_j^{(l)}) w_{j,k}^{(l+1)}$

Thus $$\delta_j^{(l)} = \sum_k \delta_k^{(l+1)} \phi'(z_j^{(l)}) w_{j,k}^{(l+1)}$$

It can be written in vector form:

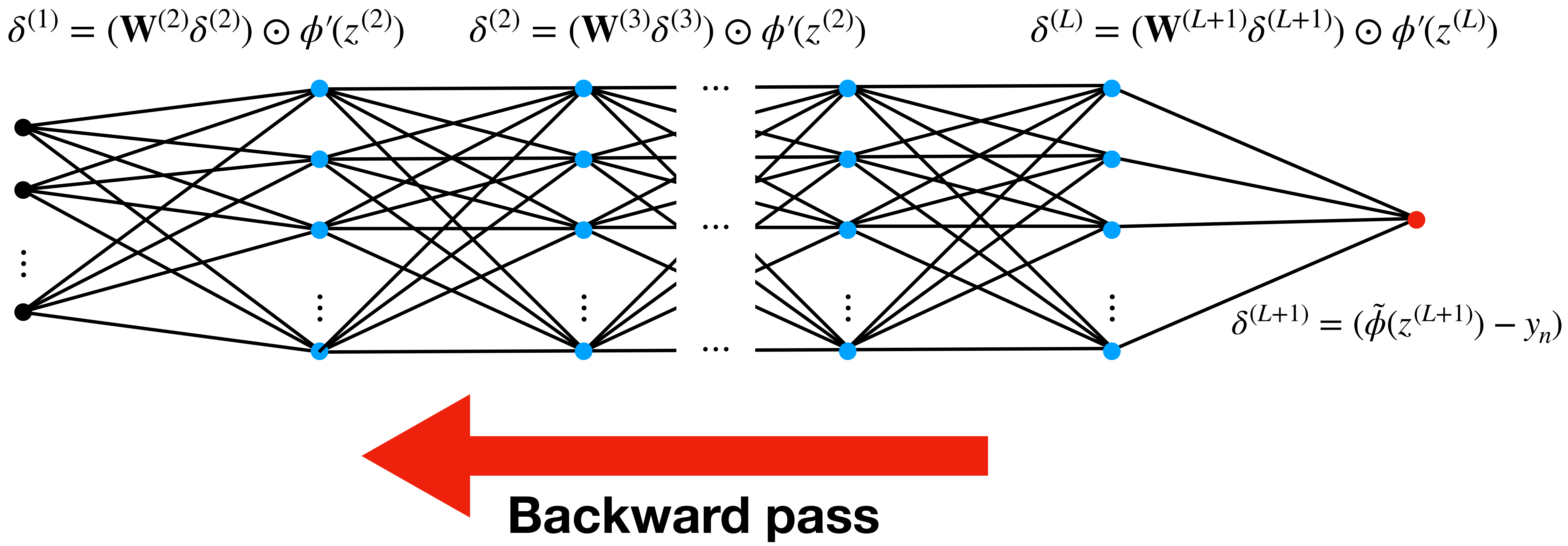
$$\delta^{(l)} = (\mathbf{W}^{(l+1)} \delta^{(l+1)}) \odot \phi'(z^{(l)})$$

$\odot$: Hadamard product, i.e., pointwise multiplication of vector

Backward pass (III)

Initialization:

$$\delta^{(L+1)} = \frac{\partial}{\partial z^{(L+1)}} \frac{1}{2}\left(y_n - \tilde{\phi}(z^{(L+1)})\right)^2$$
$$= (\tilde{\phi}(z^{(L+1)}) - y_n)\tilde{\phi}'(z^{(L+1)})$$

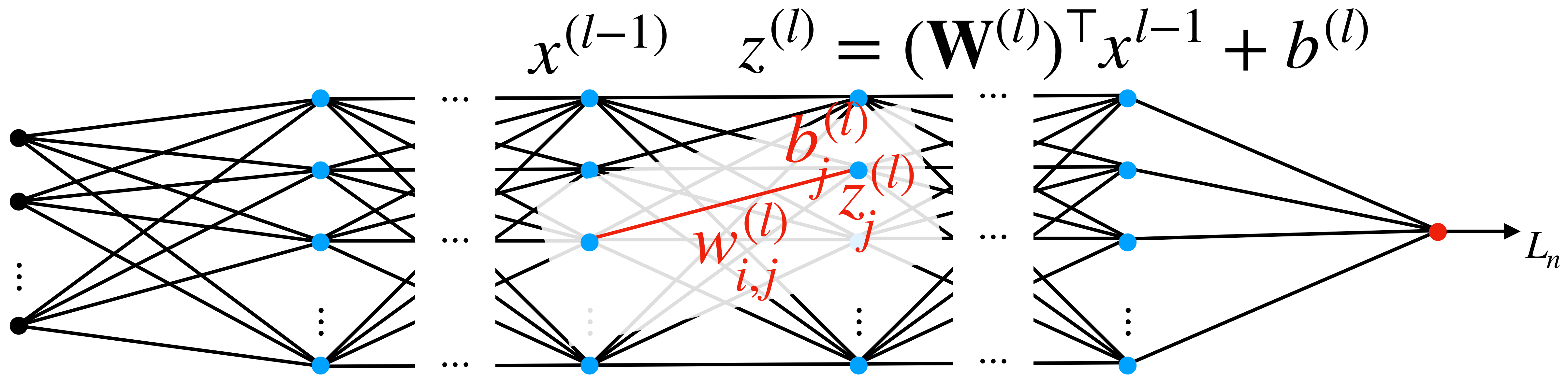


Compute all the $\delta^{(l)}$ by a backward pass in the network:

$$\delta^{(L+1)} = (\tilde{\phi}(z^{(L+1)}) - y_n)\tilde{\phi}(z^{(L+1)})$$
$$\delta^{(l)} = (\mathbf{W}^{(l+1)}\delta^{(l+1)}) \odot \phi'(z^{(l)})$$

Computational complexity: one pass over the network $O(K^2L)$

Derivatives computation



Using that $z_m^{(l)} = \sum_{k=1}^{K} w_{k,m}^{(l)} x_k^{(l-1)} + b_m^{(l)}$:

- $$\frac{\partial L_n}{\partial b_j^{(l)}} = \sum_{k=1}^{K} \frac{\partial L_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_j^{(l)}} = \frac{\partial L_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)}$$
- $$\frac{\partial L_n}{\partial w_{i,j}^{(l)}} = \sum_{k=1}^{K} \frac{\partial L_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial L_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}} = \delta_j^{(l)} \cdot x_i^{(l-1)}$$

Backpropagation algorithm

Forward pass:

$$x^{(0)} = x_n \in \mathbb{R}^d$$
$$z^{(l)} = (\mathbf{W}^{(l)})^\top x^{l-1} + b^{(l)}$$
$$x^{(l)} = \phi(z^{(l)})$$

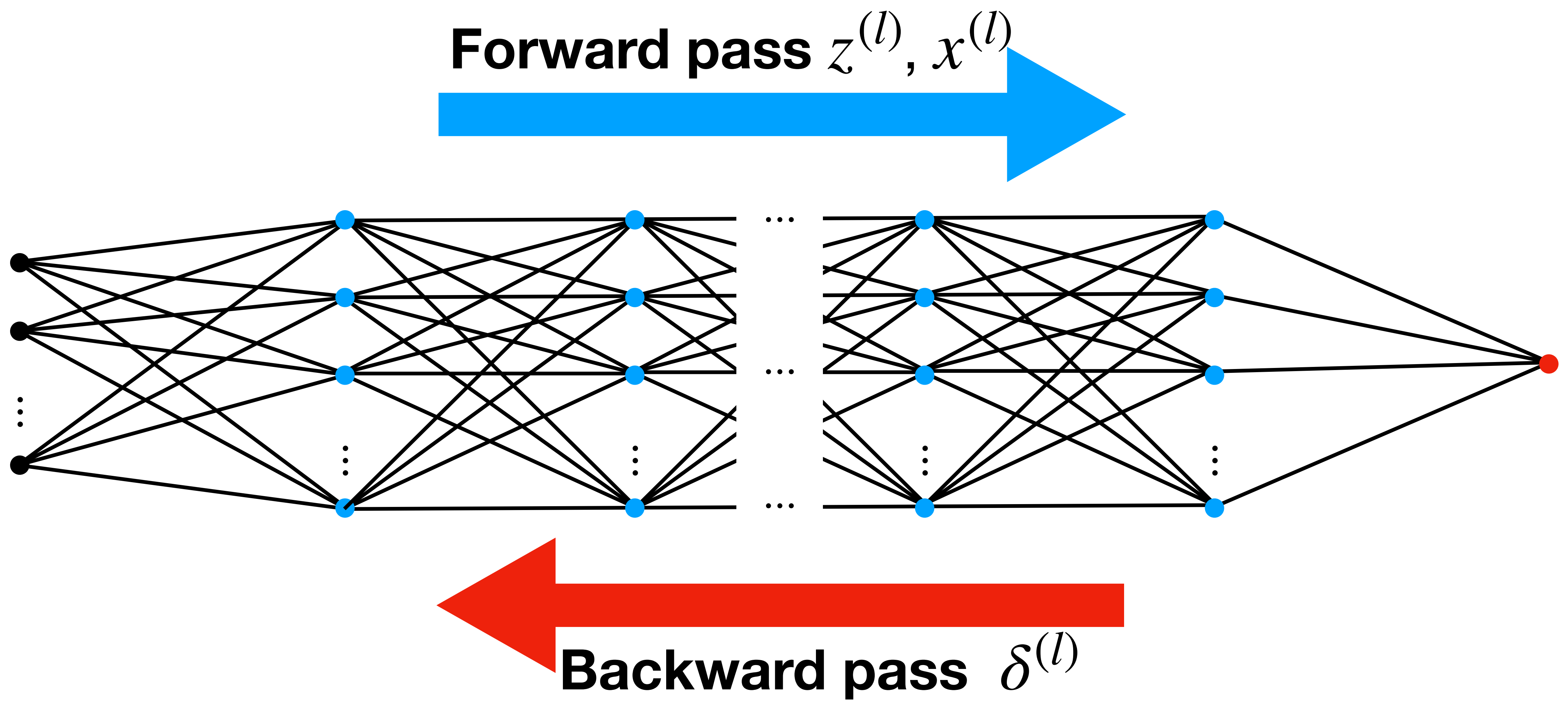
Backward pass:

$$\delta^{(L+1)} = (\tilde{\phi}(z^{(L+1)}) - y_n)\tilde{\phi}'(z^{(L+1)})$$

$$\delta^{(l)} = (\mathbf{W}^{(l+1)}\delta^{(l+1)}) \odot \phi'(z^{(l)})$$

Compute the derivatives:

$$\frac{\partial L_n}{\partial w_{i,j}^{(l)}} = \delta_j^{(l)} x_i^{(l-1)}$$
$$\frac{\partial L_n}{\partial b_j^{(l)}} = \delta_j^{(l)}$$



Overall Complexity: $O(K^2 L)$

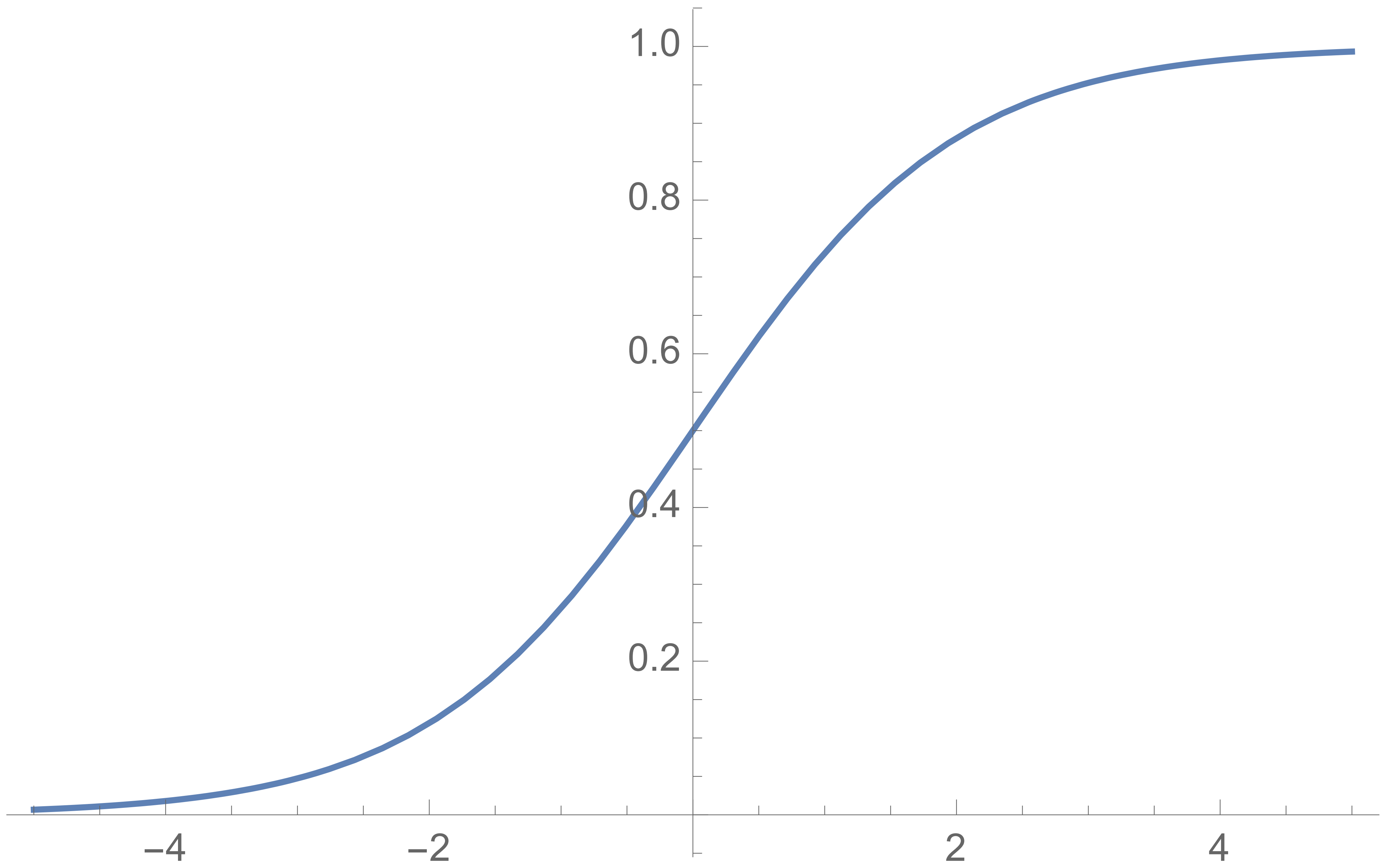
Neural Networks
Popular Activation Functions

Machine Learning Course - CS-433
Nov 9, 2022
Nicolas Flammarion

EPFL

The sigmoid

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

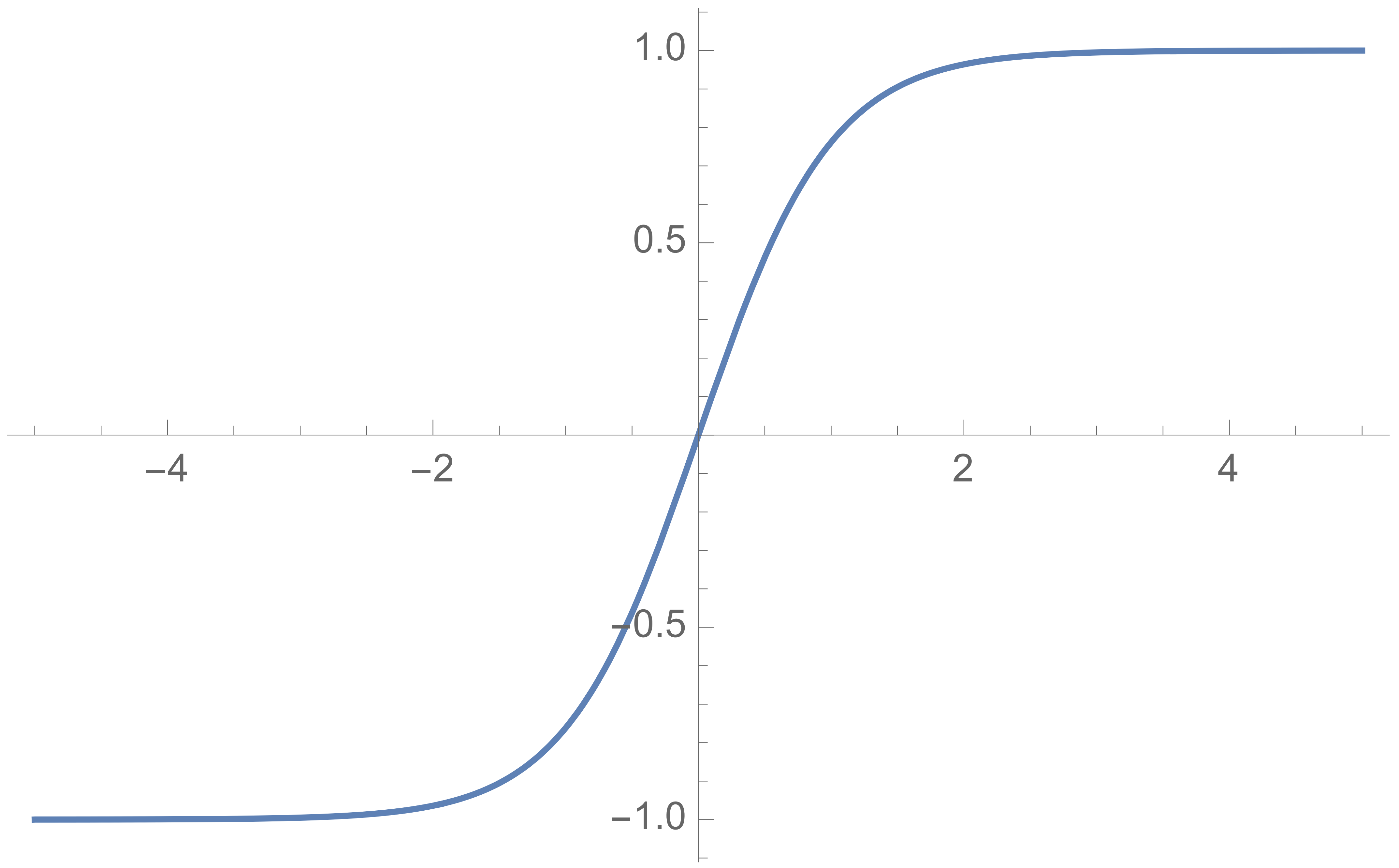


- Pro: Smooth everywhere

- Cons: $|\sigma'(x)| \lll 1$ for $|x| \ggg 1$ - problem of vanishing gradient

Hyperbolic Tangent

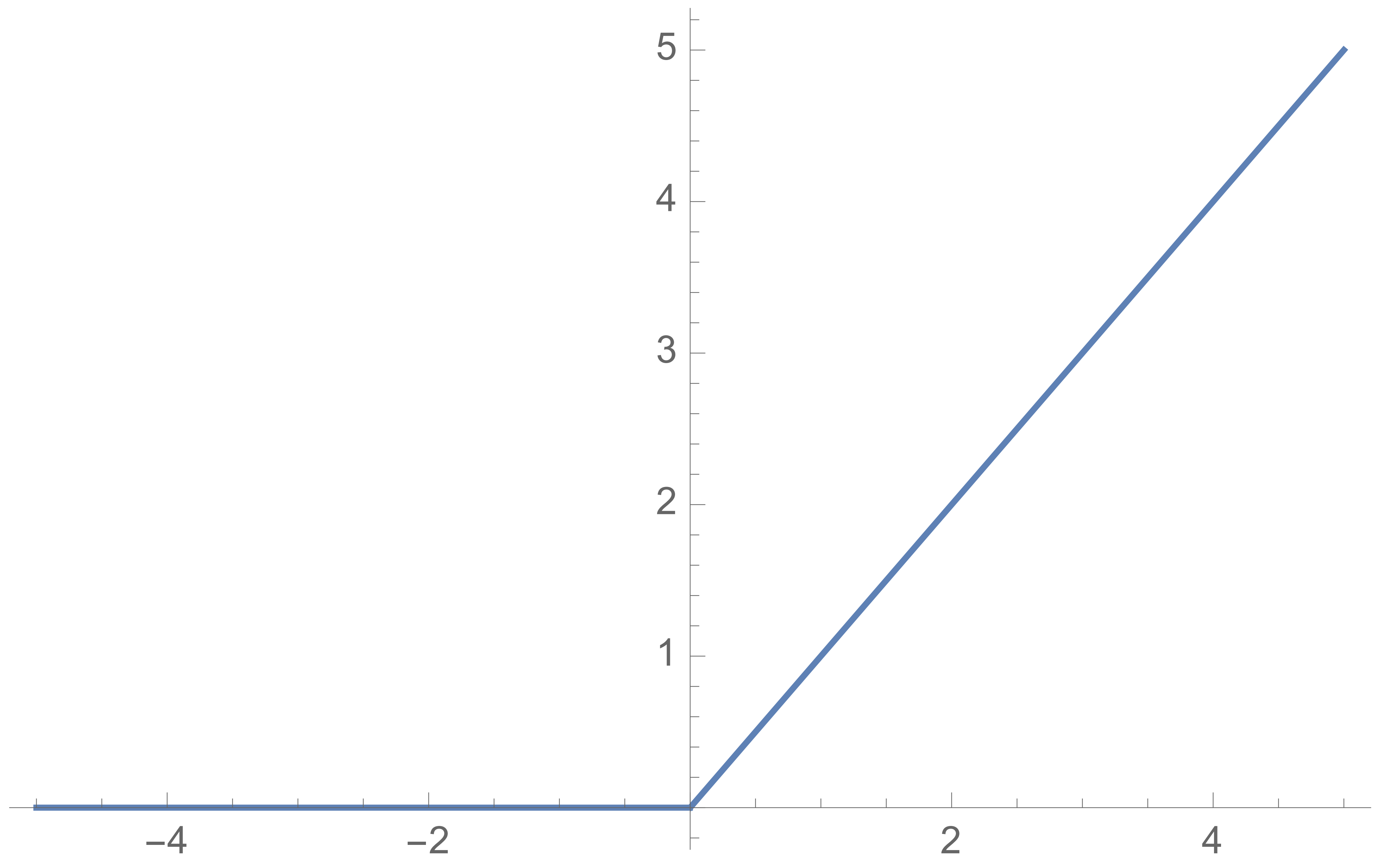
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$$



- Related to the sigmoid but balanced
- Vanishing gradient problem

Rectified linear unit - RELU

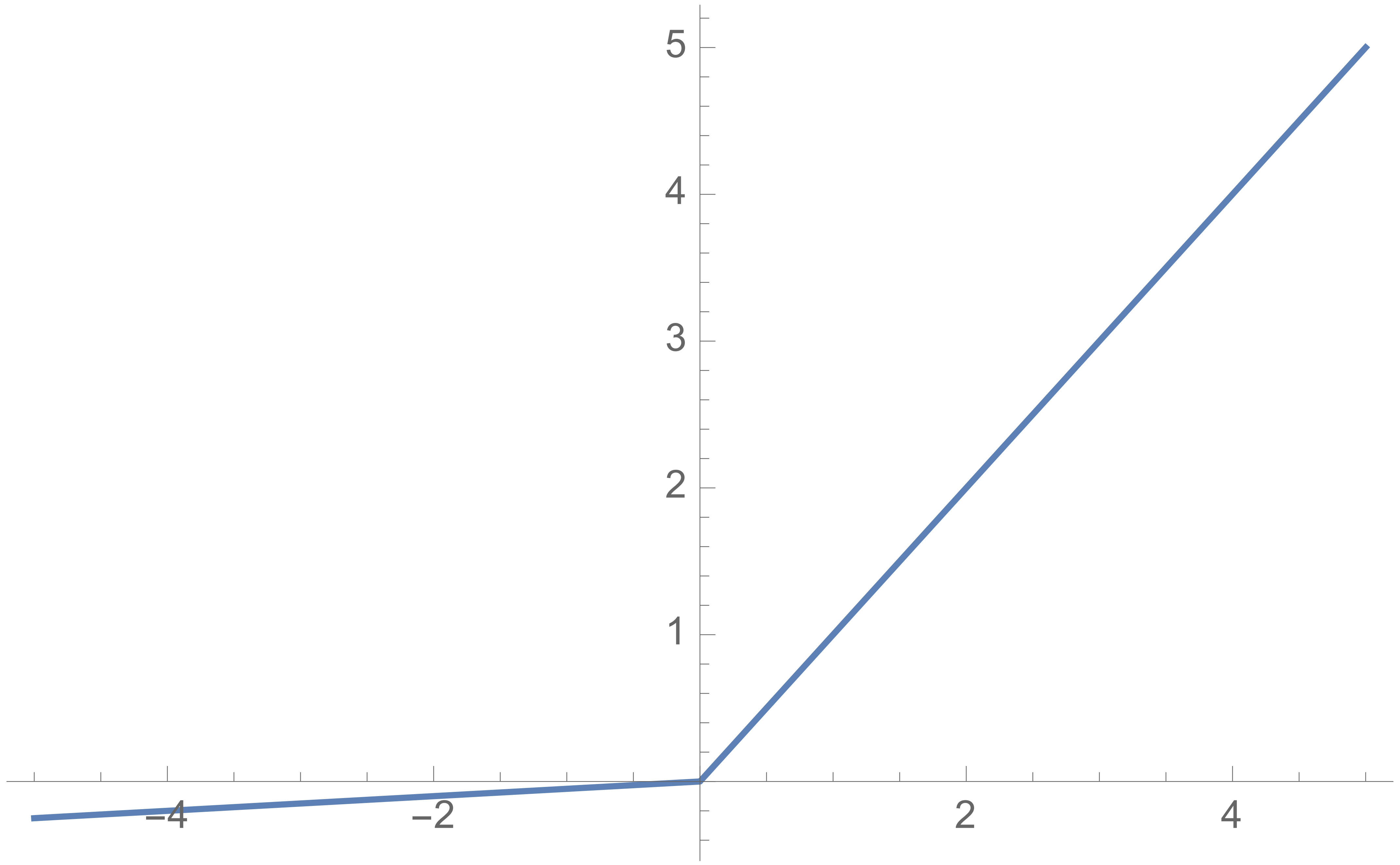
$$(x)_+ = \max\{0, x\}$$



- Pro: no vanishing gradient for $x \geq 0$
- Cons: no differentiable in 0 and the derivative is 0 for negative value

Leaky RELU - LRELU

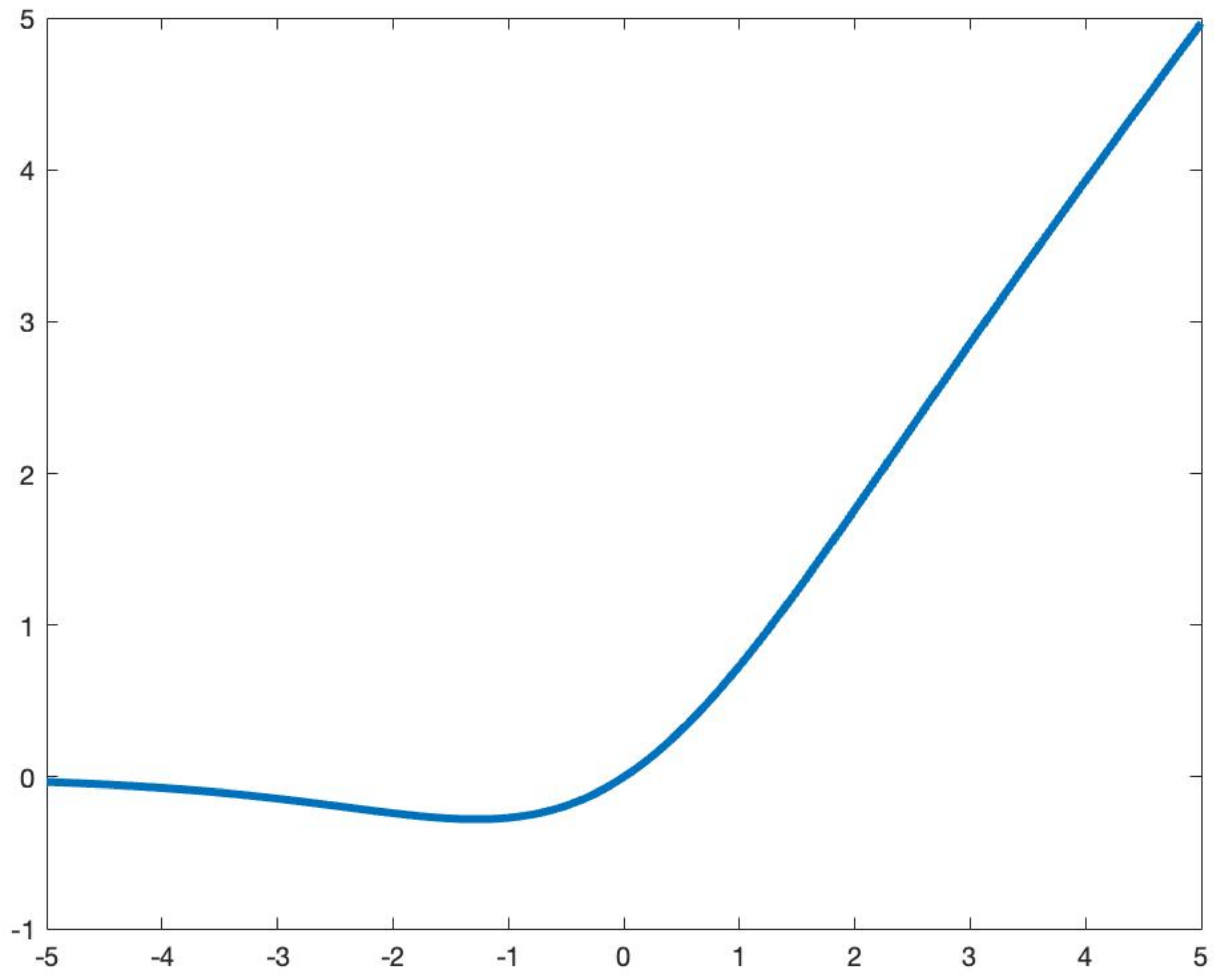
$$f(x) = \max\{\alpha x, x\}$$



- Correction of the 0 gradient of the RELU

Swish

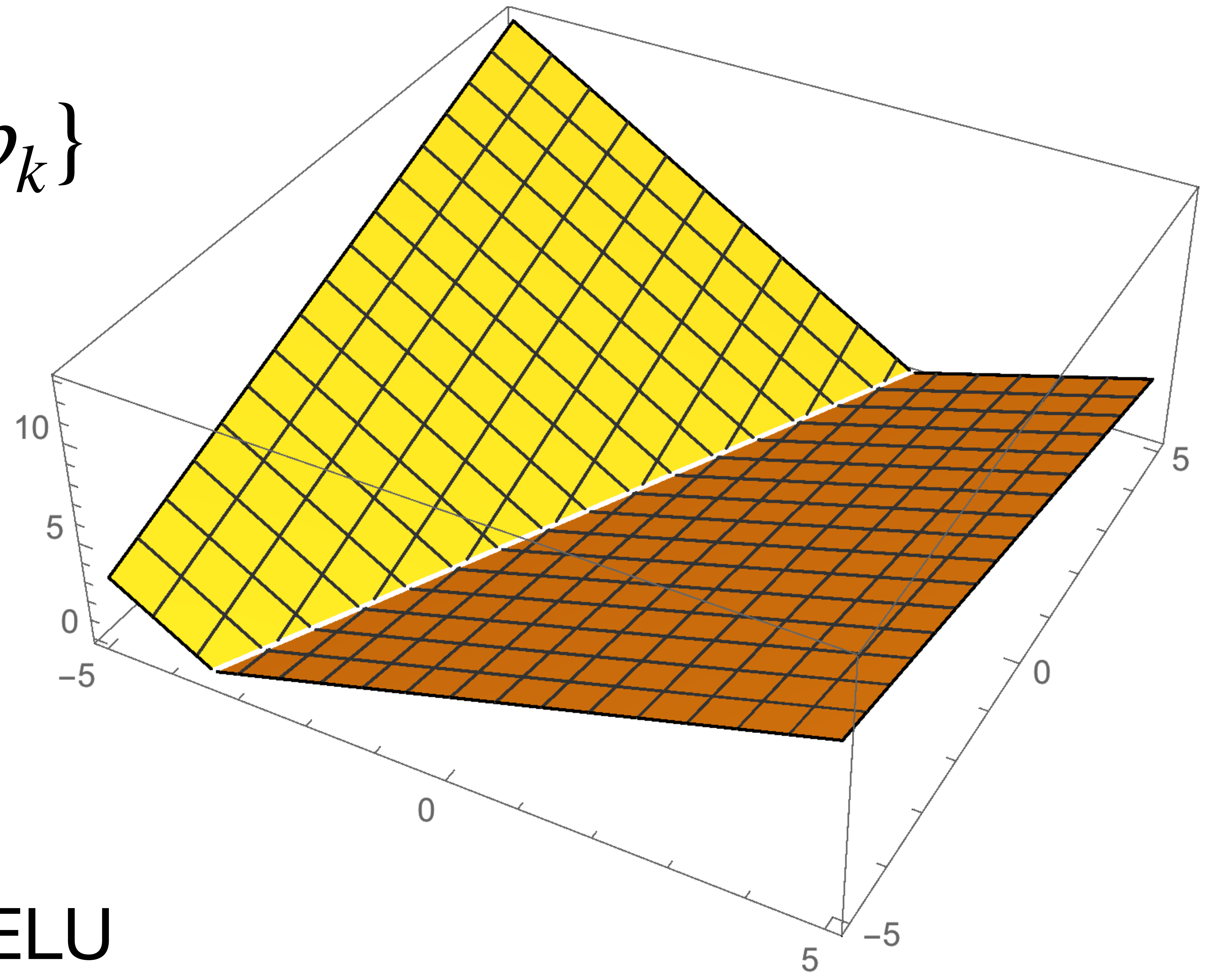
$$f(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}$$



- Correction of the 0 gradient of the RELU
- Smooth everywhere

Maxout

$$f(x) = \max\{x^\top w_1 + b_1, \cdots, x^\top w_k + b_k\}$$



- Generalization of the RELU and the LRELU