RGL Realistic Graphics Lab
CVL Computer Vision Lab

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

EPFL CS-328: Numerical Methods for Visual Computing

2018

Neural Networks for Visual Computing

Dr. Helge Rhodin

Reading material for today's topics

Neural Networks and Deep Learning

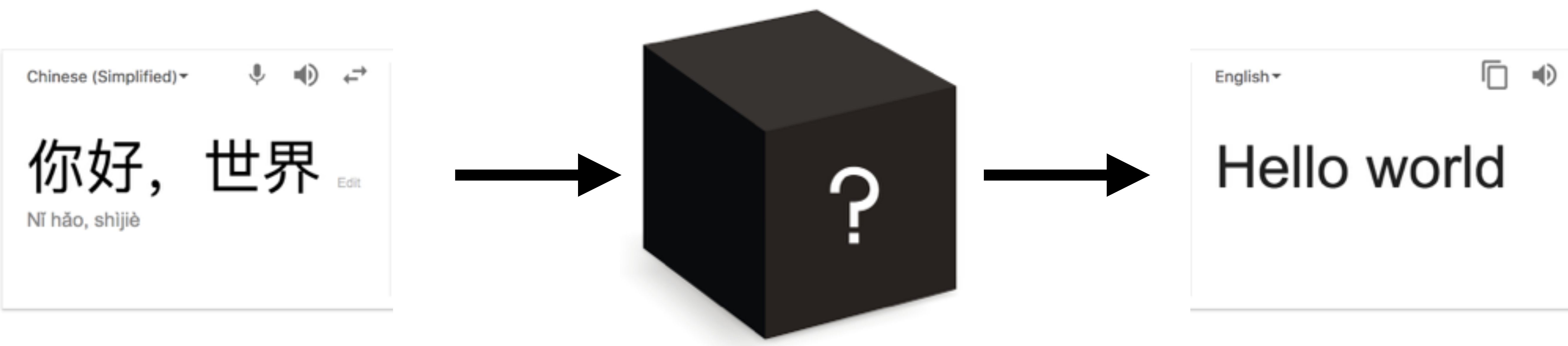
Chapter 1-2 (online book)

NeuralNetworksAndDeepLearning.com

Michael Nielsen

Neural networks in practice

Neural networks (NNS) are a widely used — a tool to learn patterns from large databases.



Natural language processing

[Google Translate]

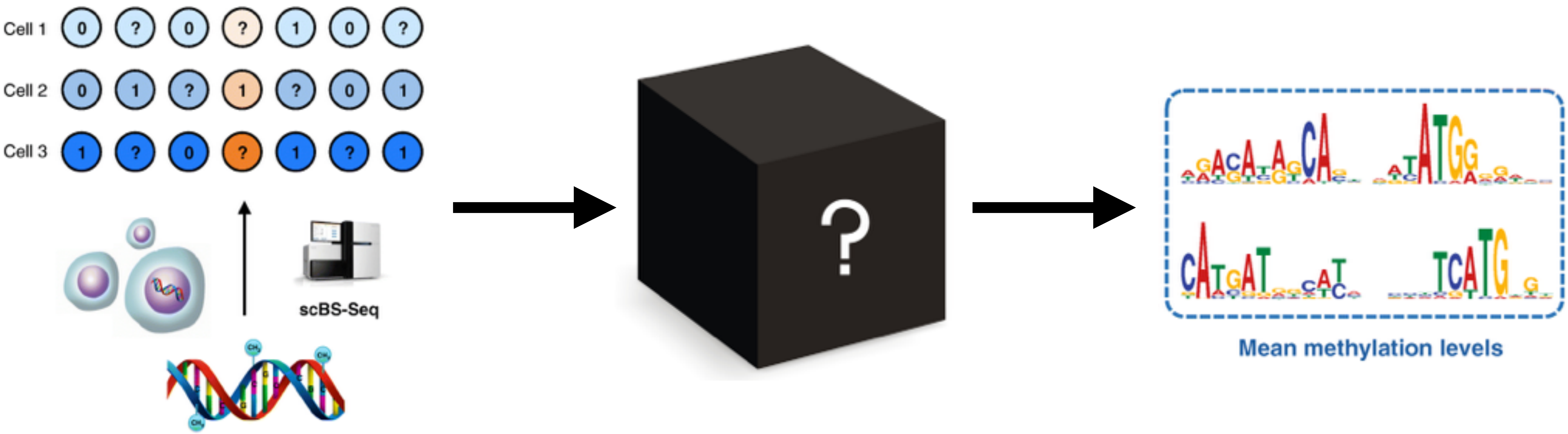
Neural networks in practice

Neural networks (NNS) are a widely used — a tool to learn patterns from large databases.

Computer vision, image segmentation

Neural networks in practice

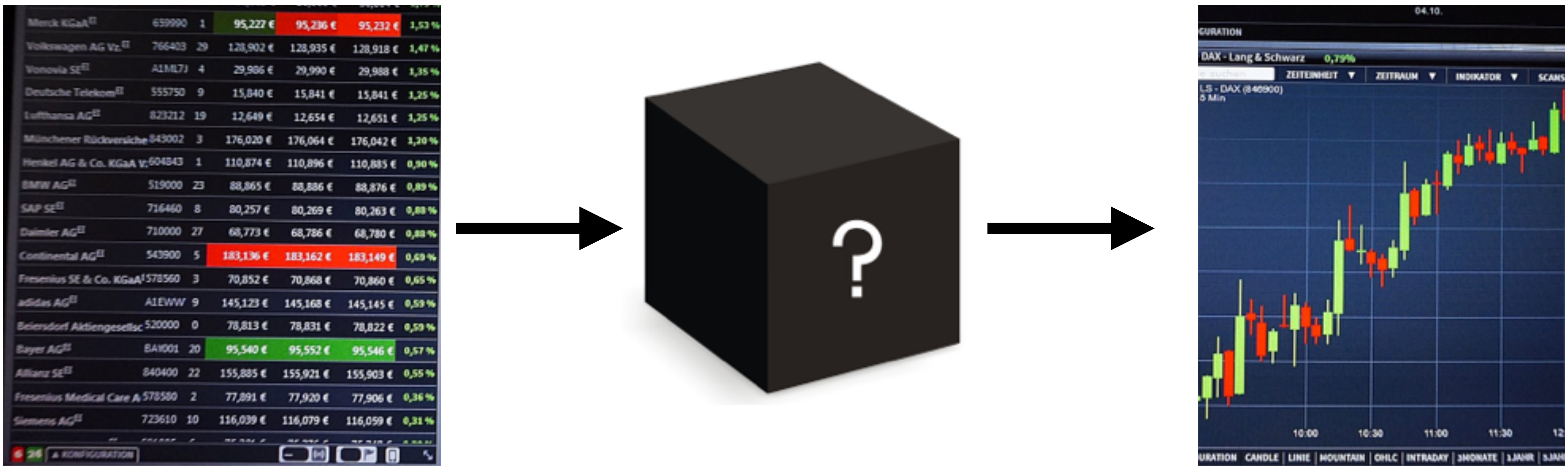
Neural networks (NNS) are a widely used — a tool to learn patterns from large databases.



Biology, DNA analysis

Neural networks in practice

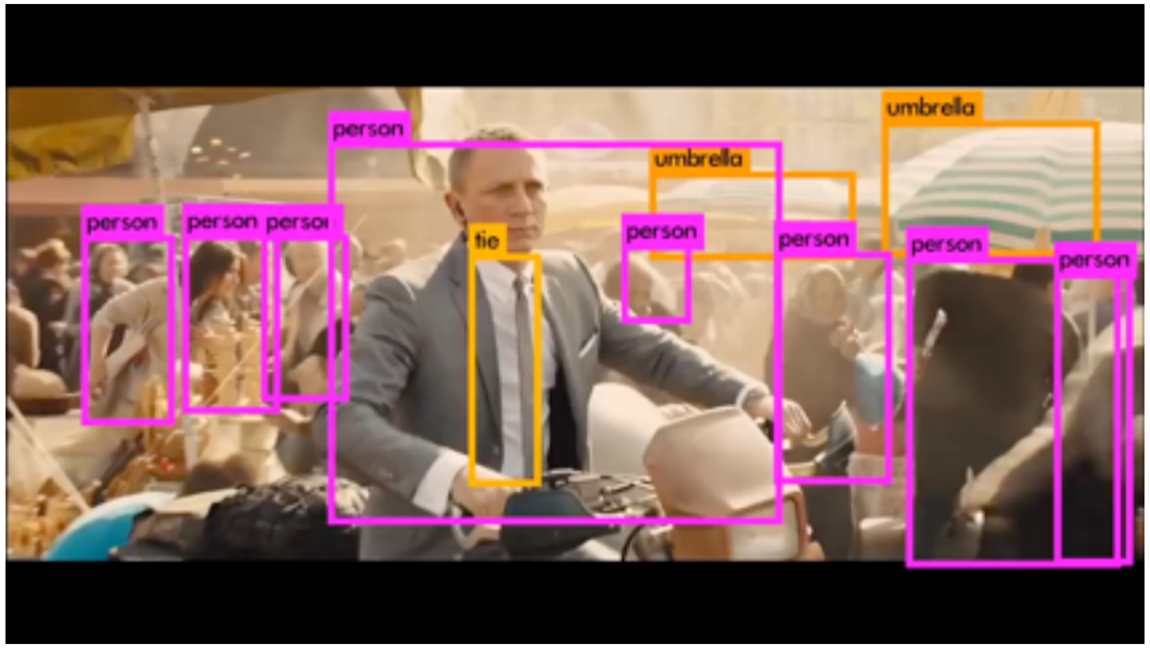
Neural networks (NNS) are a widely used — a tool to learn patterns from large databases.



Finance and risk management

Neural networks and visual computing

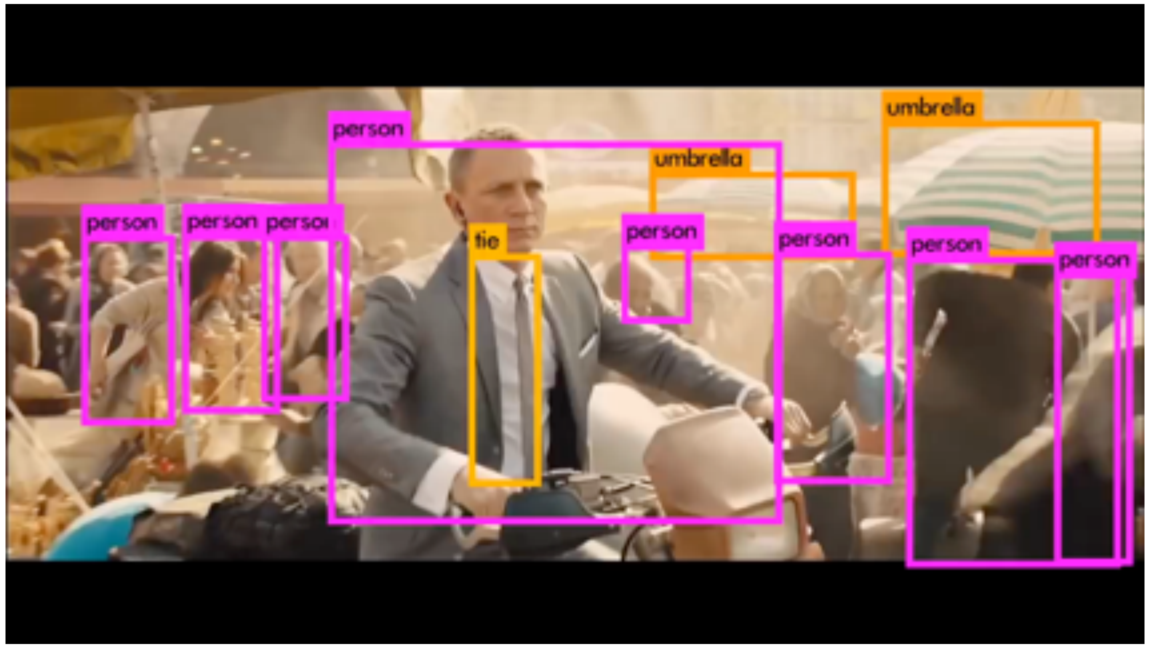
Convolutional NNs are particularly suited to extract semantic information from images.



Object and person detection [Redmon 2016]

Neural networks and visual computing

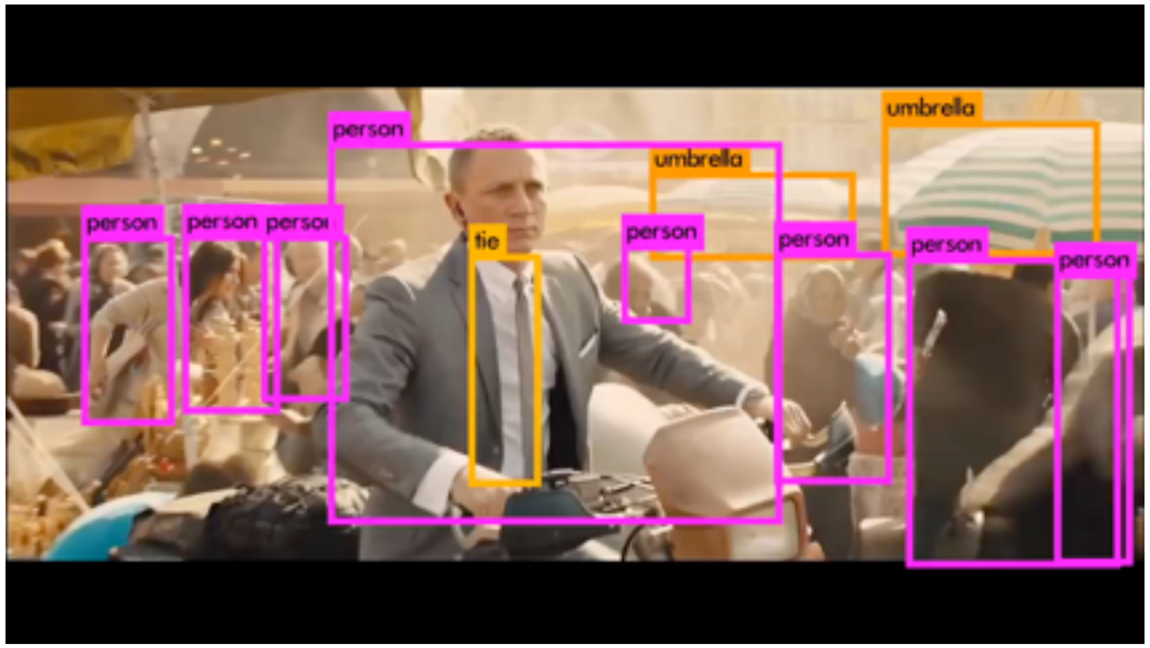
Convolutional NNs are particularly suited to extract semantic information from images.



Object and person detection [Redmon 2016]

Neural networks and visual computing

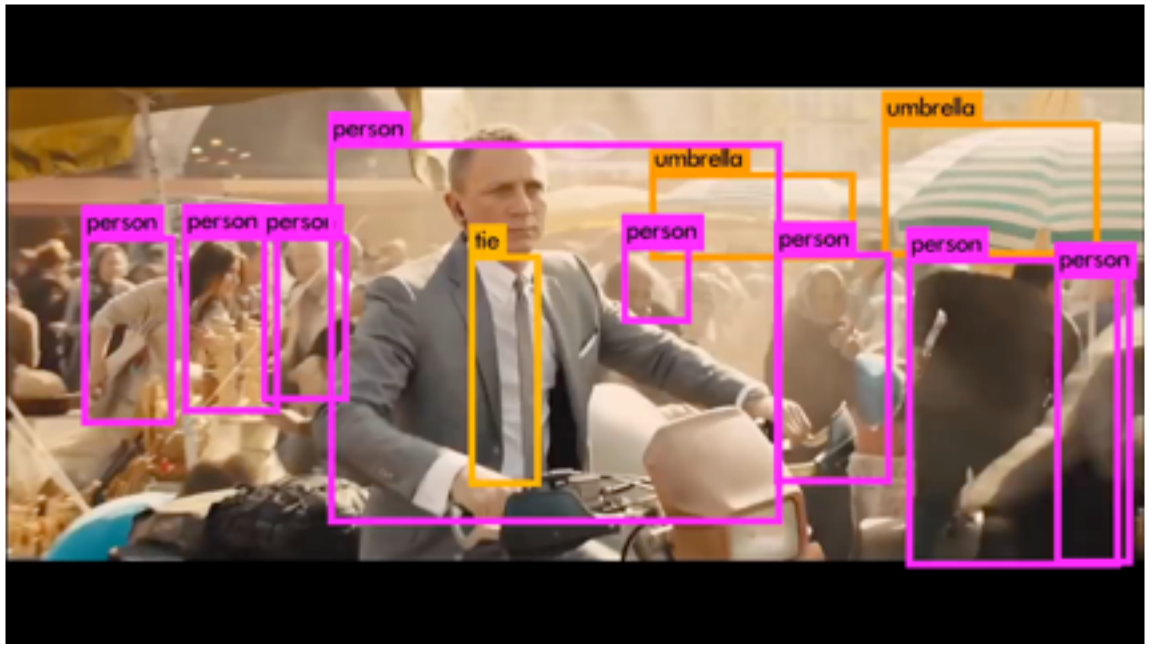
Convolutional NNs are particularly suited to extract semantic information from images.



Object and person detection [Redmon 2016]

Neural networks and visual computing

Convolutional NNs are particularly suited to extract semantic information from images.

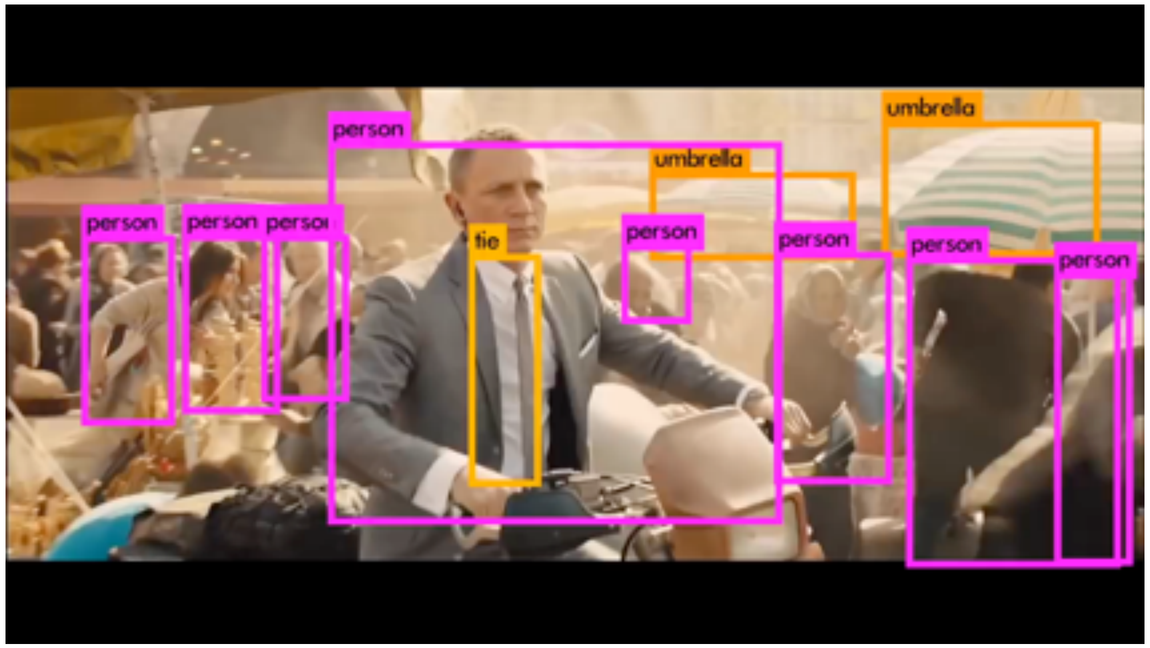


Object and person detection [Redmon 2016]

3D Human pose estimation [Mehta 2017]

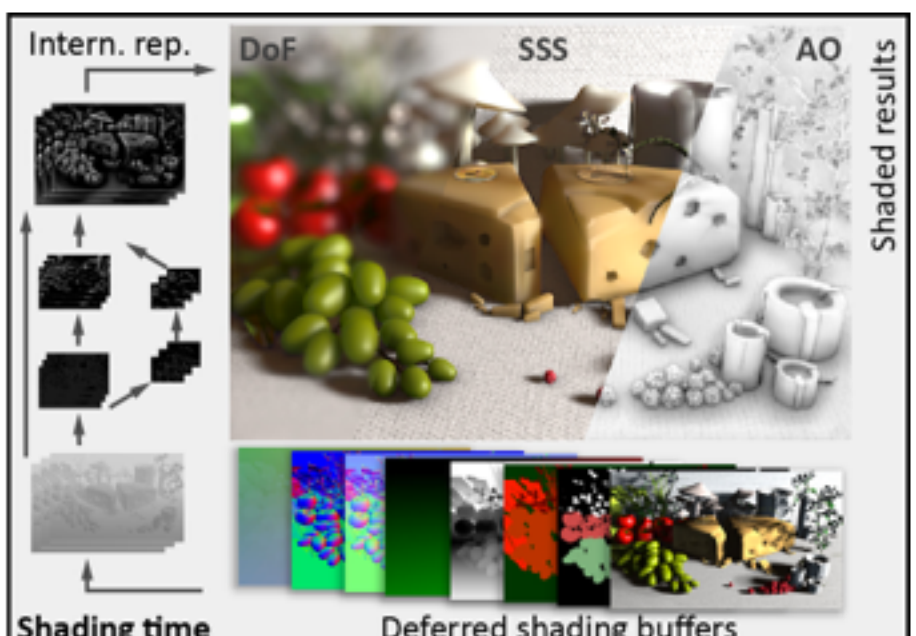
Neural networks and visual computing

Convolutional NNs are particularly suited to extract semantic information from images.



Object and person detection [Redmon 2016]

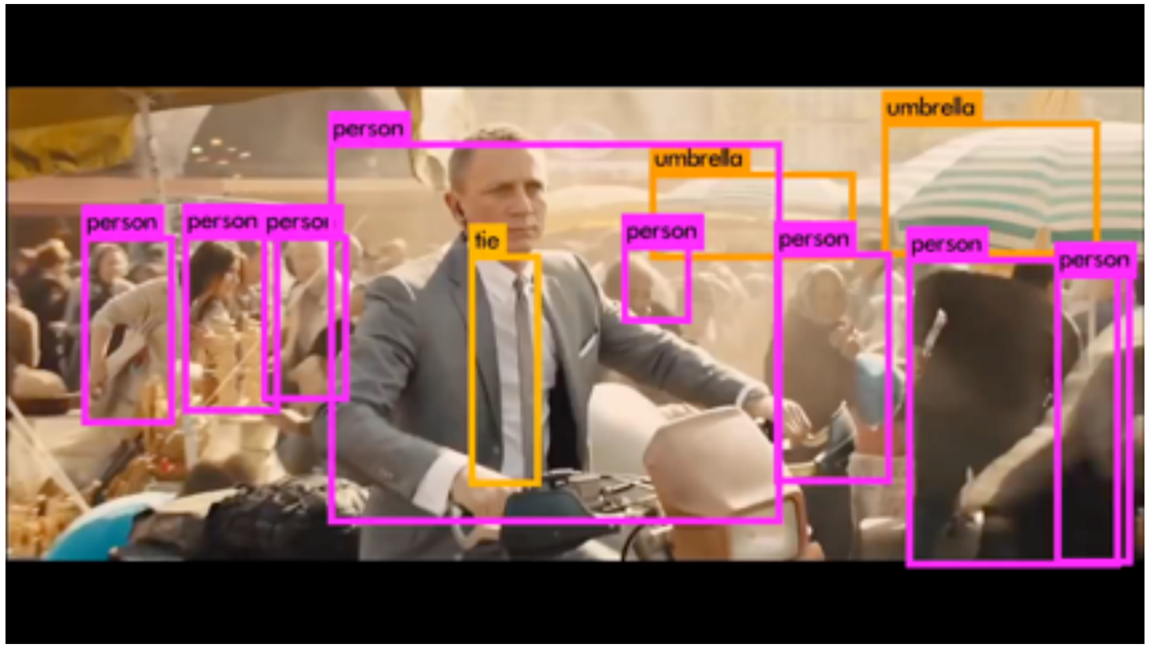
3D Human pose estimation [Mehta 2017]



Computer Graphics, rendering [Nalbach 2017]

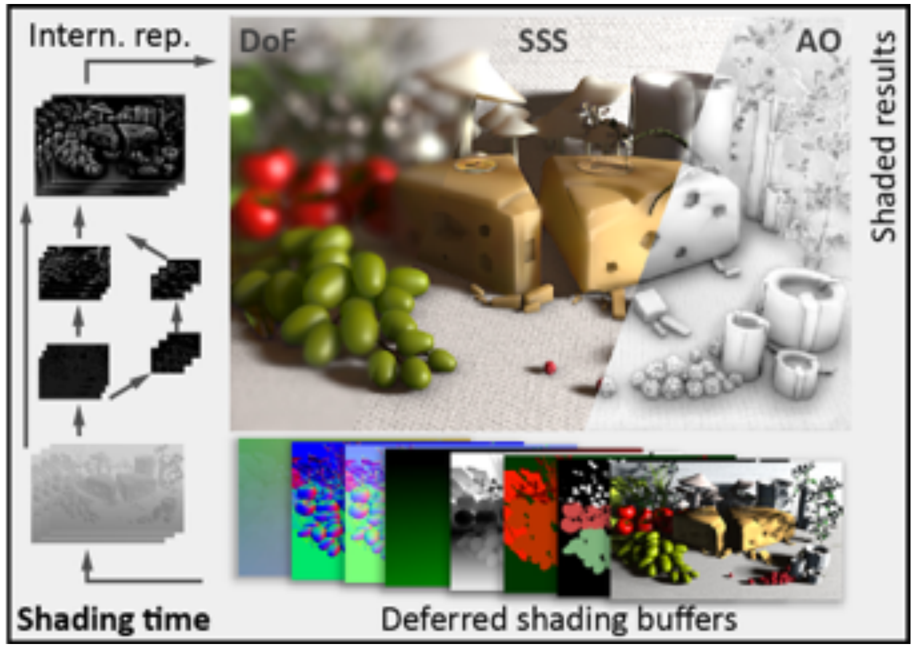
Neural networks and visual computing

Convolutional NNs are particularly suited to extract semantic information from images.

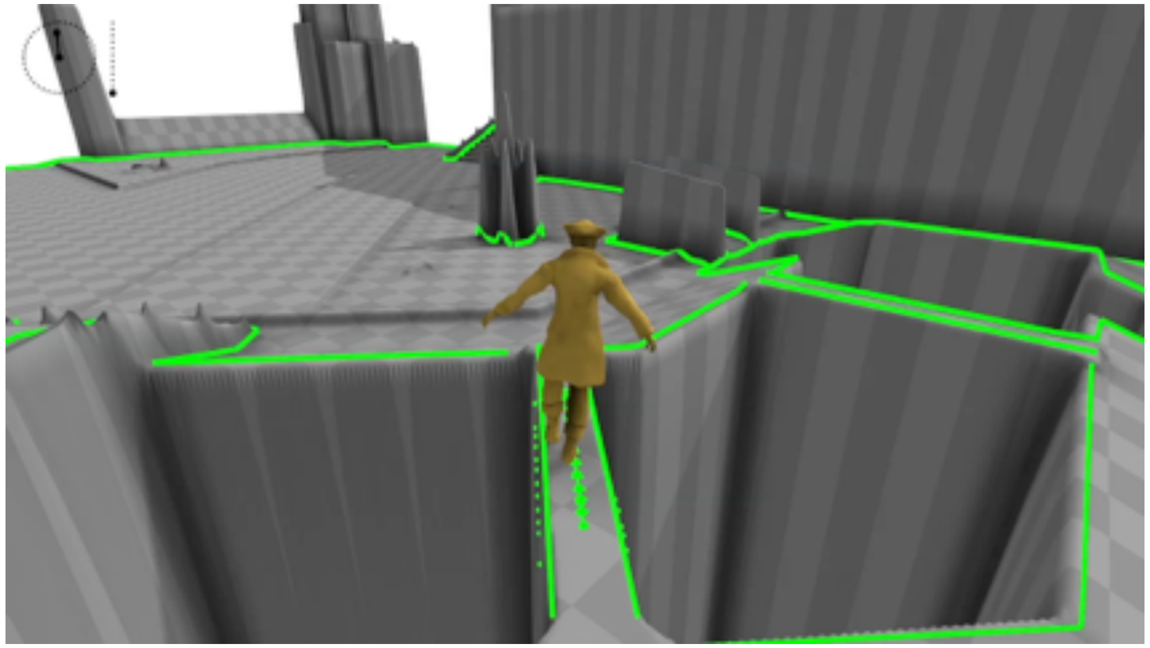


Object and person detection [Redmon 2016]

3D Human pose estimation [Mehta 2017]



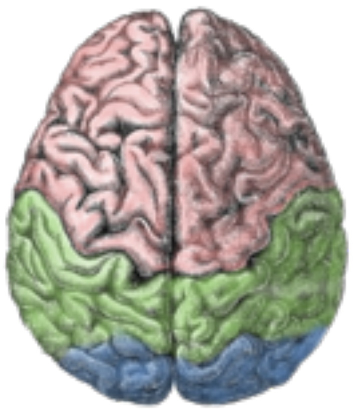
Computer Graphics, rendering [Nalbach 2017]



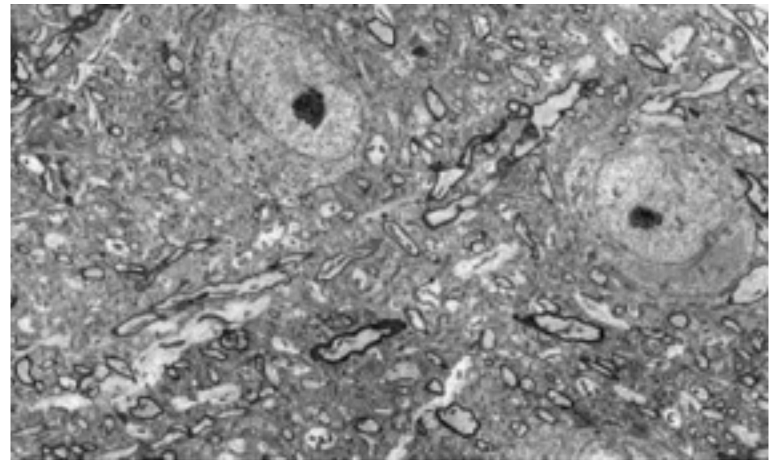
Animation, character control

Biological neural network

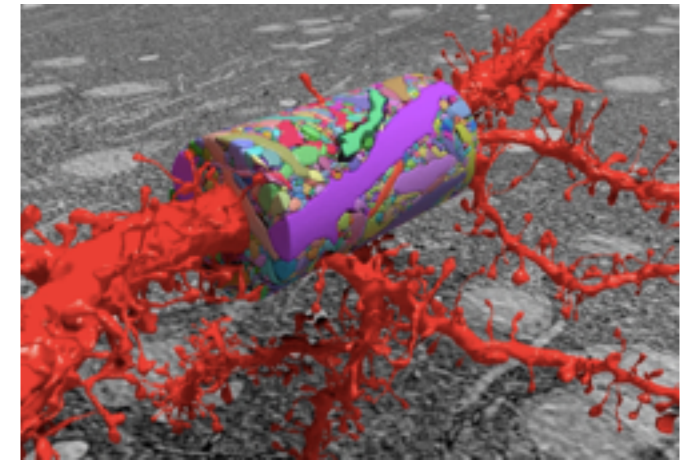
The name, but also the network structure, is inspired by our understanding of real brains.



Macroscopic scale
(human brain)



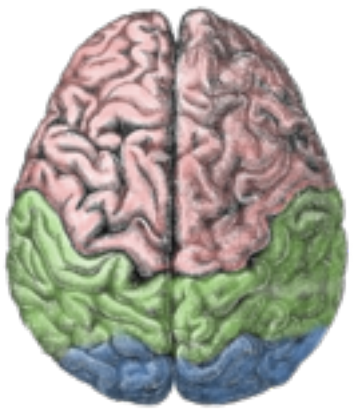
Microscopic slice
(neocortex)



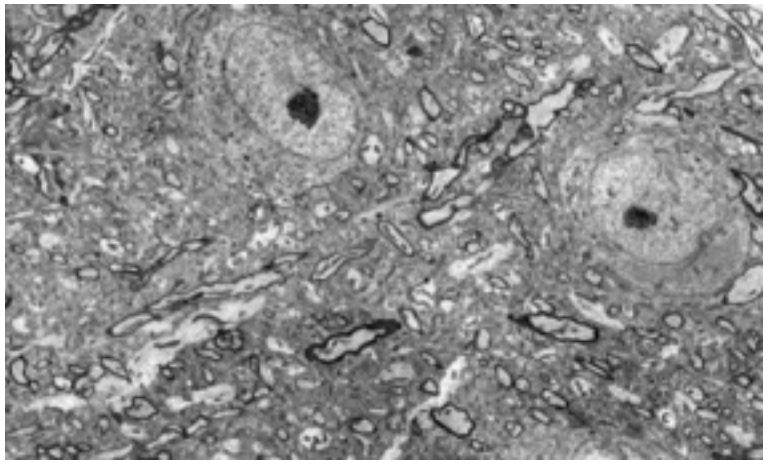
3D reconstruction
(dendrite and surrounding)

Biological neural network

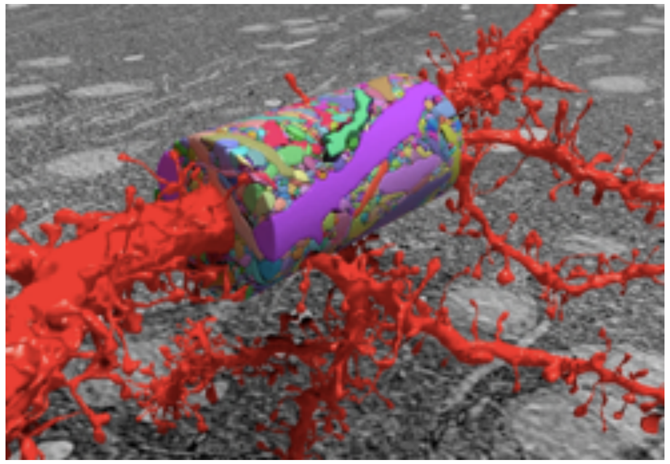
The name, but also the network structure, is inspired by our understanding of real brains.



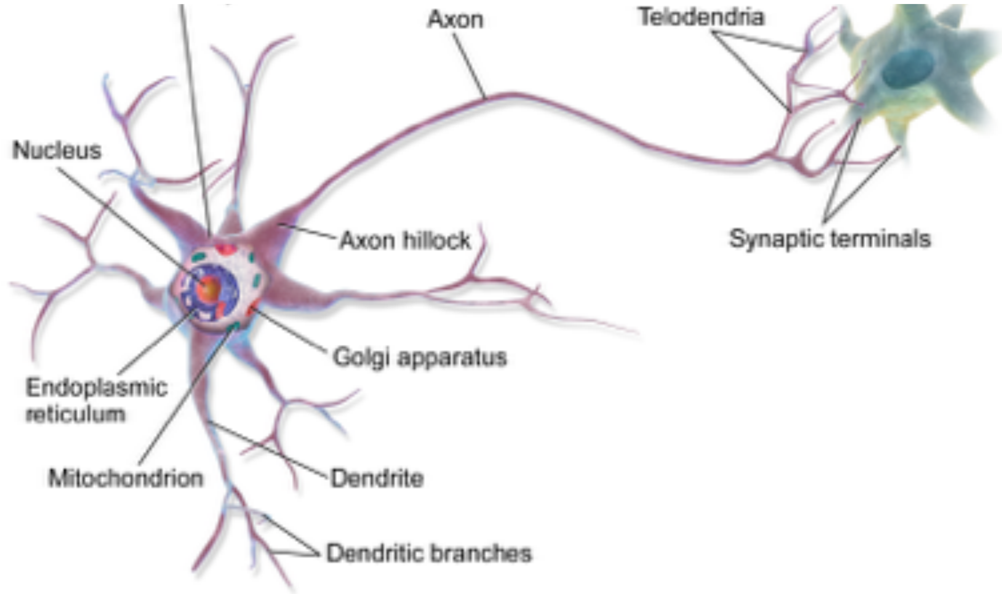
Macroscopic scale
(human brain)



Microscopic slice
(neocortex)



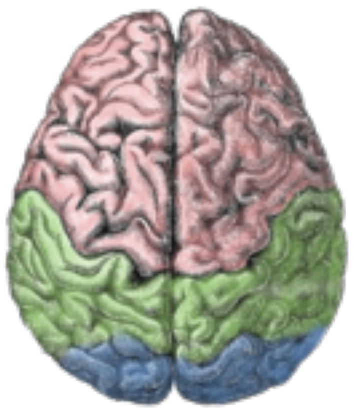
3D reconstruction
(dendrite and surrounding)



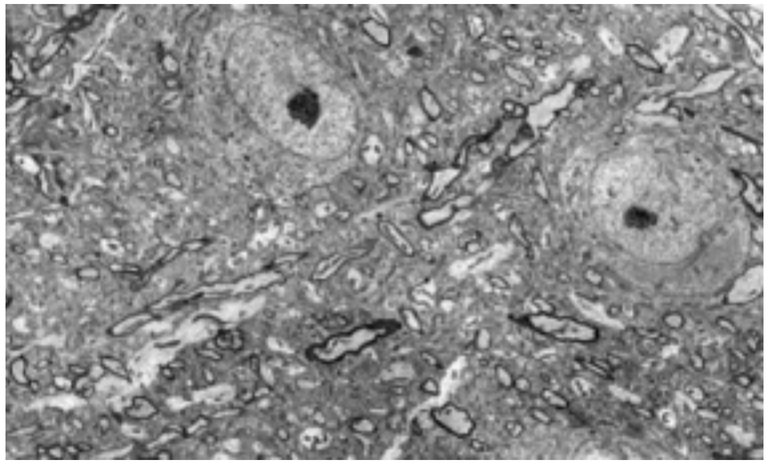
Biological model
(100 billion neurons, 100 trillion connections)

Biological neural network

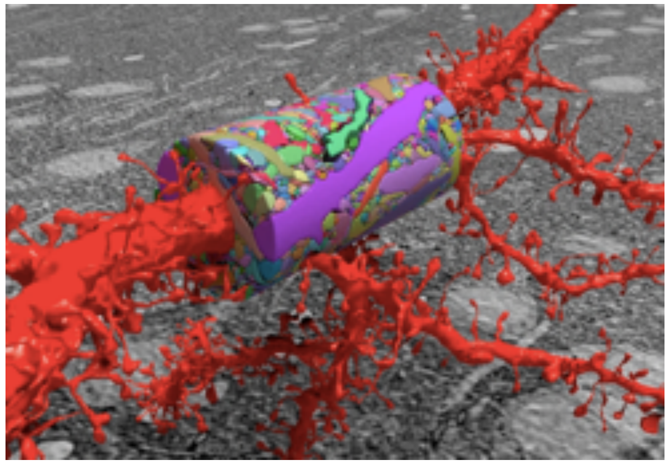
The name, but also the network structure, is inspired by our understanding of real brains.



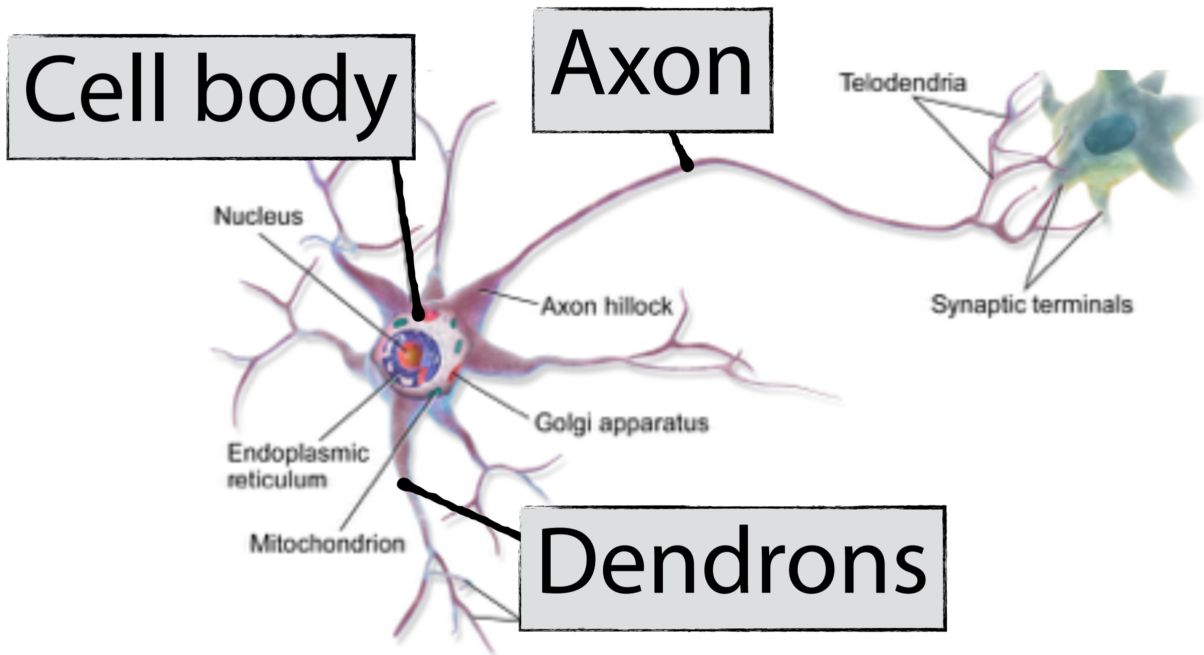
Macroscopic scale
(human brain)



Microscopic slice
(neocortex)



3D reconstruction
(dendrite and surrounding)

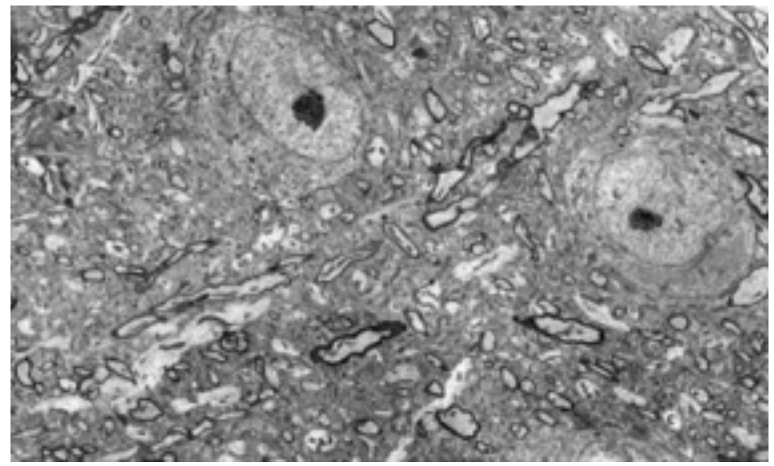


Biological model
(100 billion neurons, 100 trillion connections)

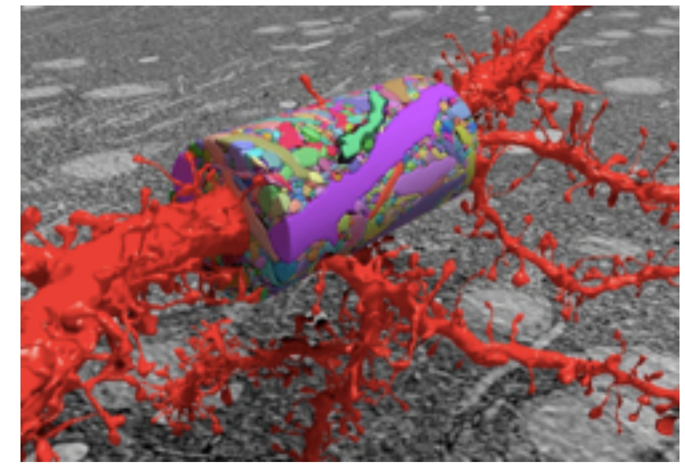
Biological neural network

The name, but also the network structure, is inspired by our understanding of real brains.

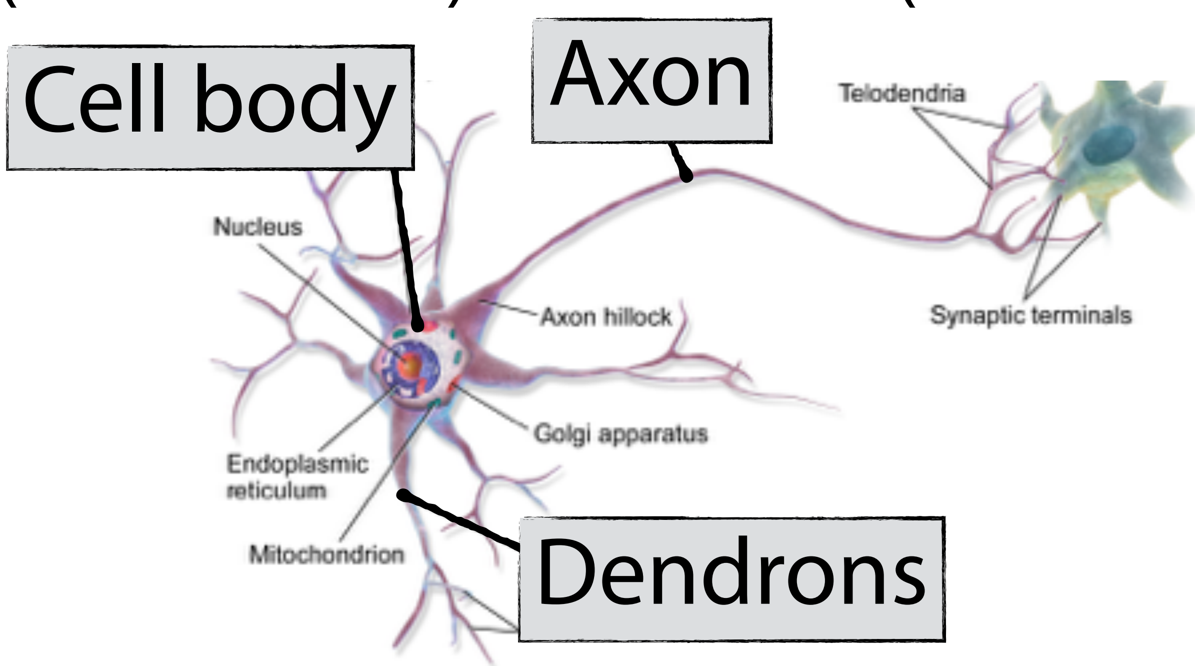
Macroscopic scale
(human brain)



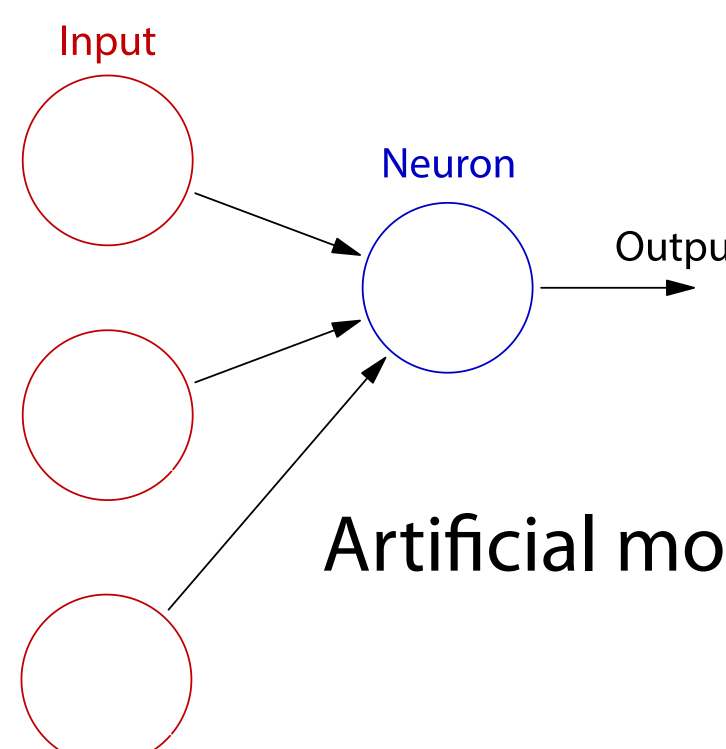
Microscopic slice
(neocortex)



3D reconstruction
(dendrite and surrounding)



Biological model
(100 billion neurons, 100 trillion connections)

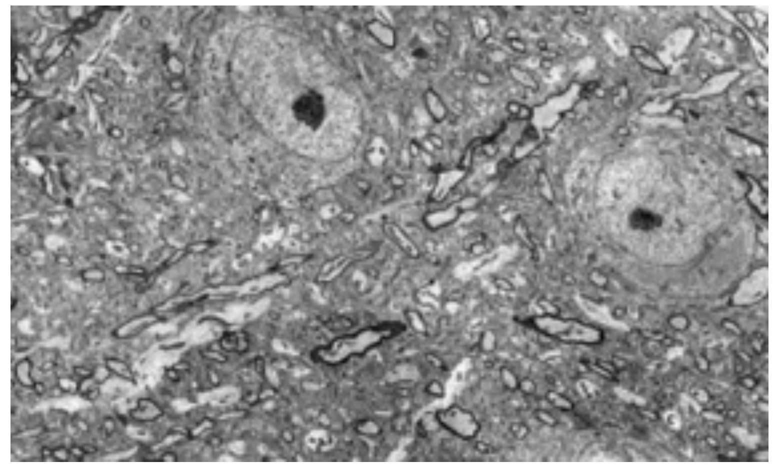


Artificial model?

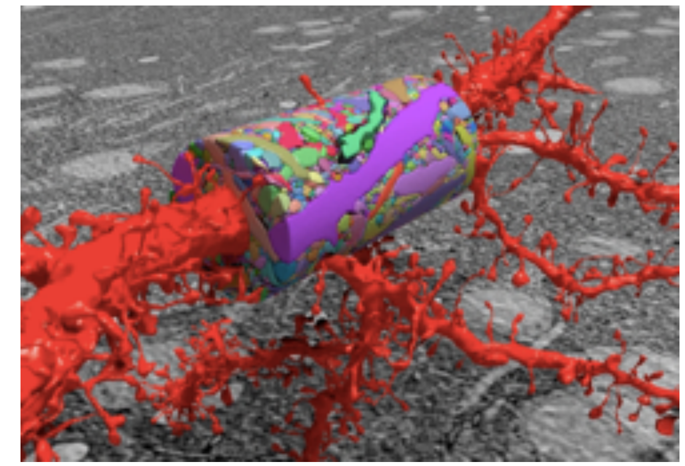
Biological neural network

The name, but also the network structure, is inspired by our understanding of real brains.

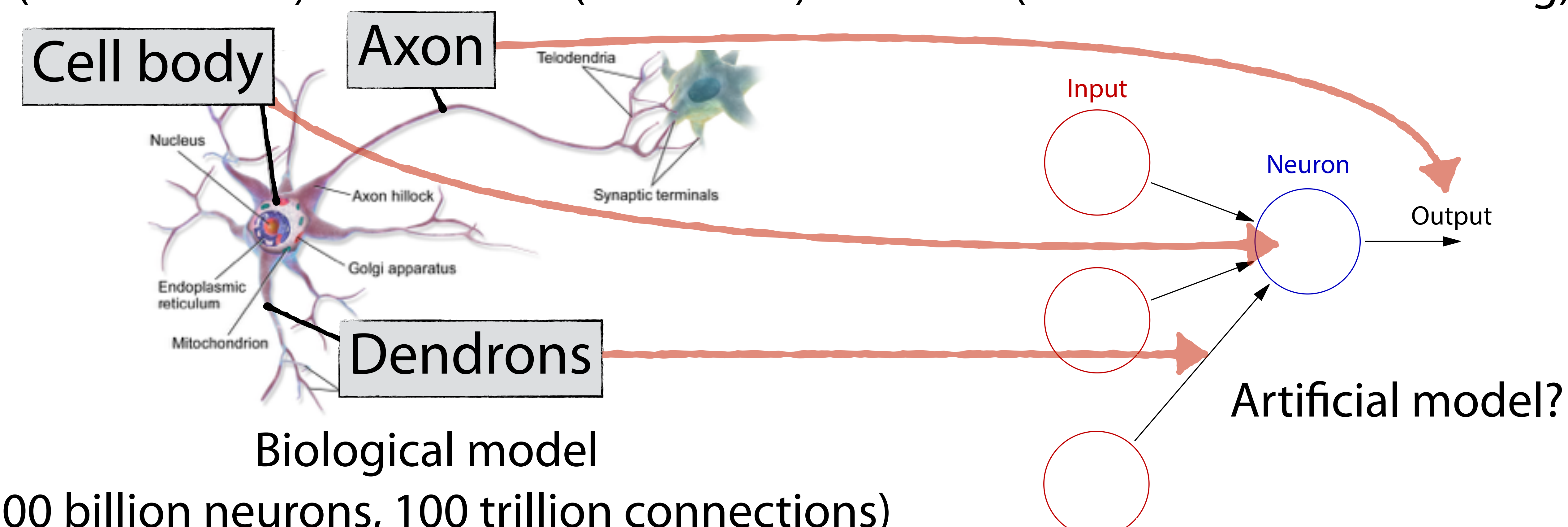
Macroscopic scale (human brain)



Microscopic slice (neocortex)



3D reconstruction (dendrite and surrounding)



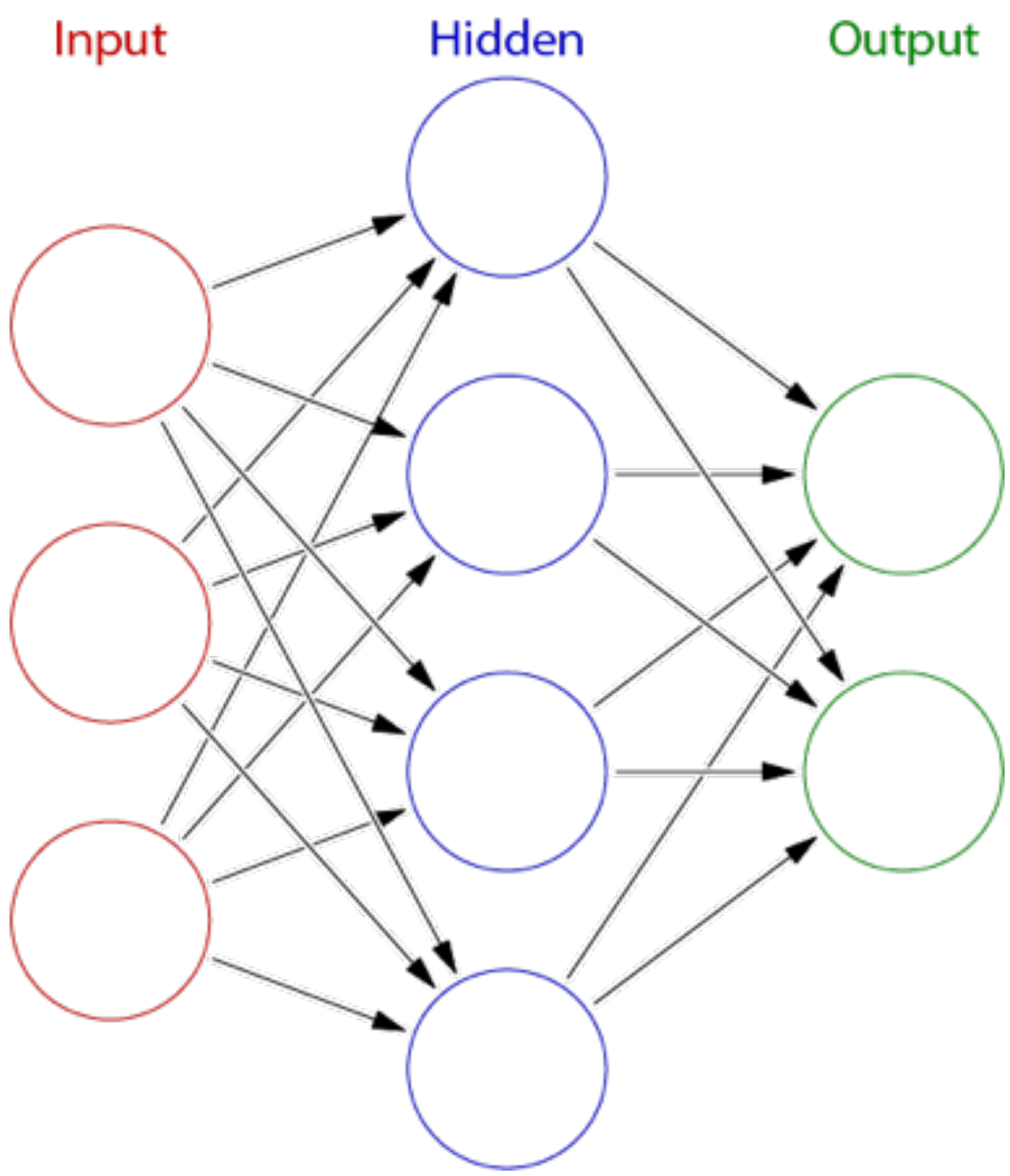
Biological model

(100 billion neurons, 100 trillion connections)

Artificial neural network

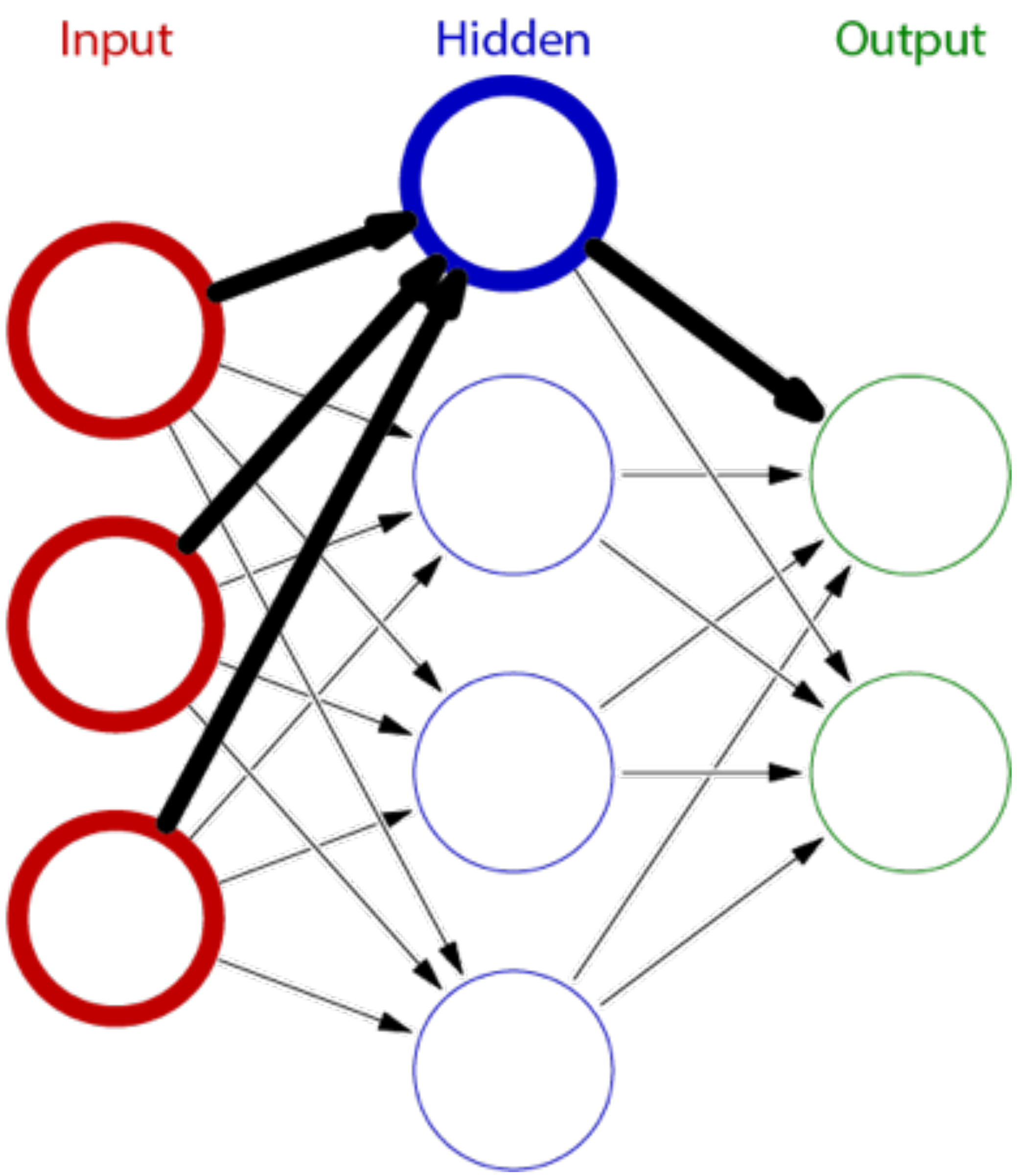
Artificial neural network — a graph

NNs are composed of simple primitives, neurons with multiple inputs and a single output.



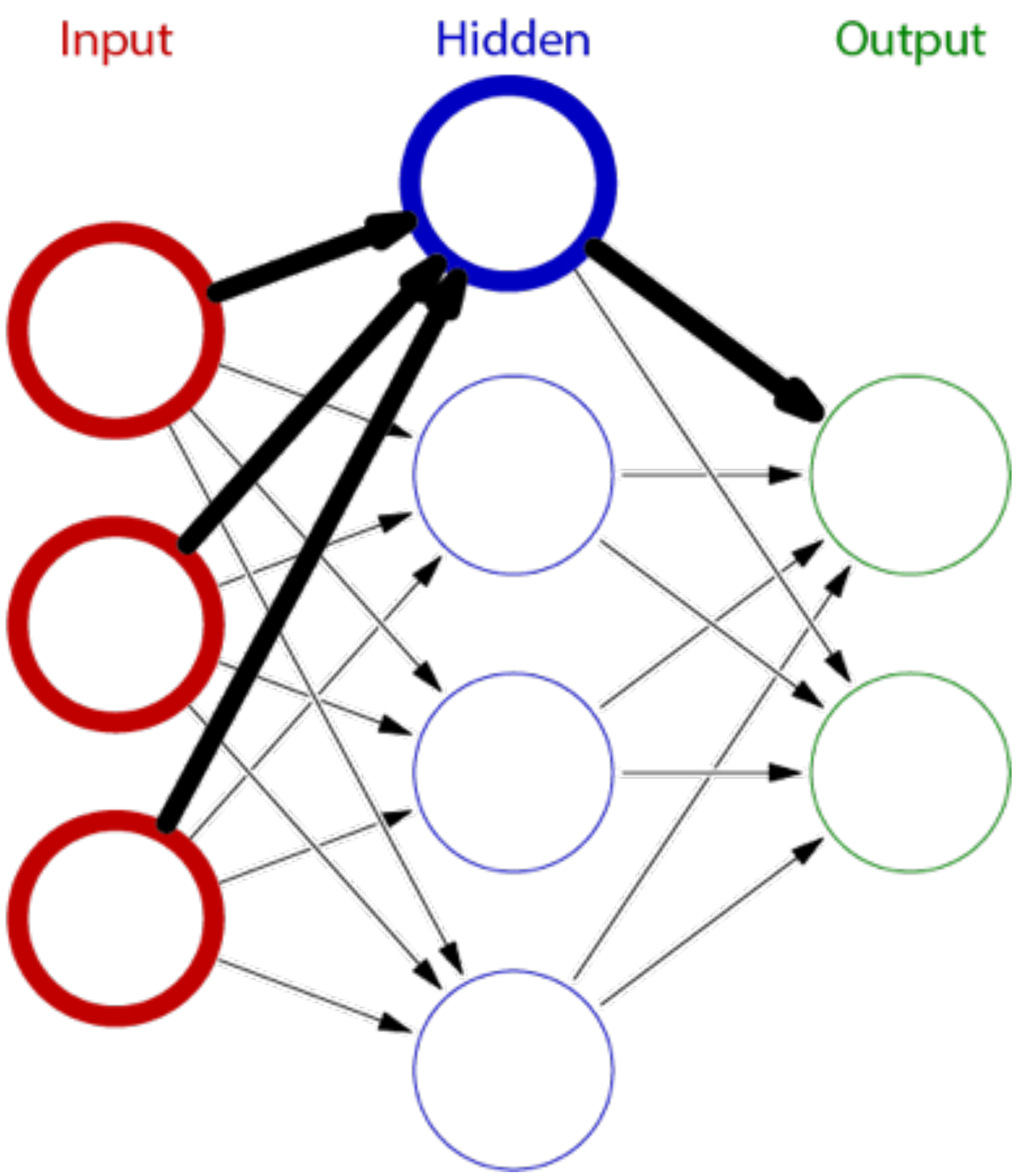
Artificial neural network — a graph

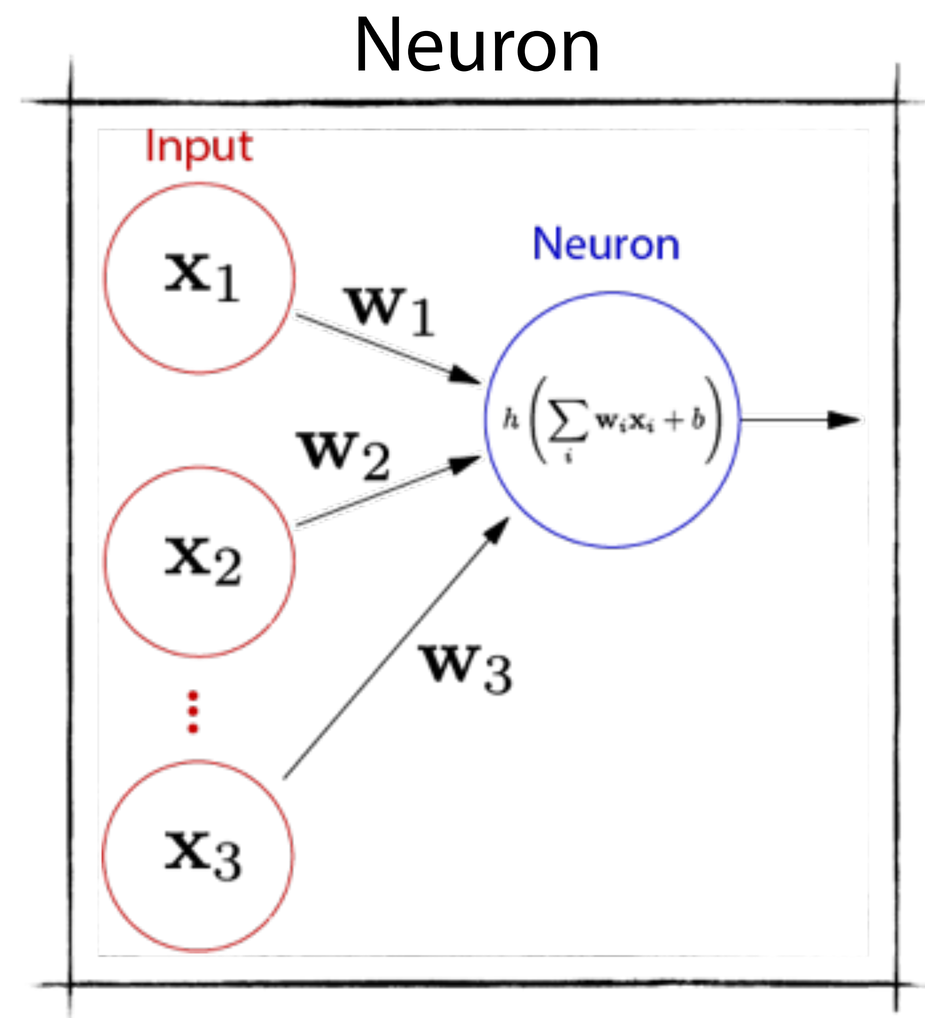
NNs are composed of simple primitives, neurons with multiple inputs and a single output.



Artificial neural network — a graph

NNs are composed of simple primitives, neurons with multiple inputs and a single output.





$\mathbf{x}_i$ input

h activation function

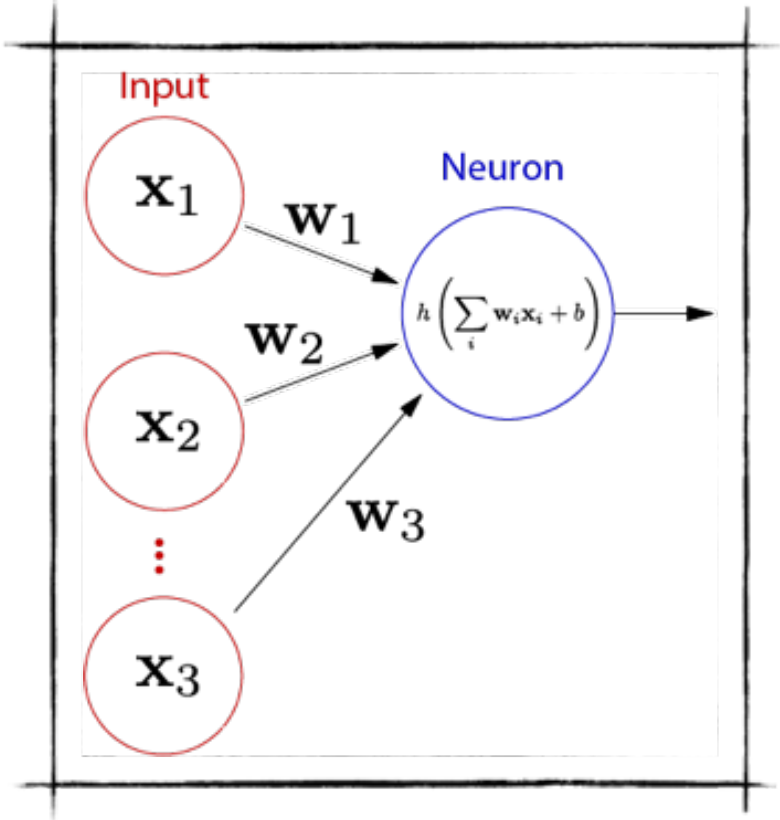
$\mathbf{w}_i$ weights

b bias

Artificial neural network — building blocks

neuron: affine map + activation function,   neuron w/o activation function = linear map

Neuron

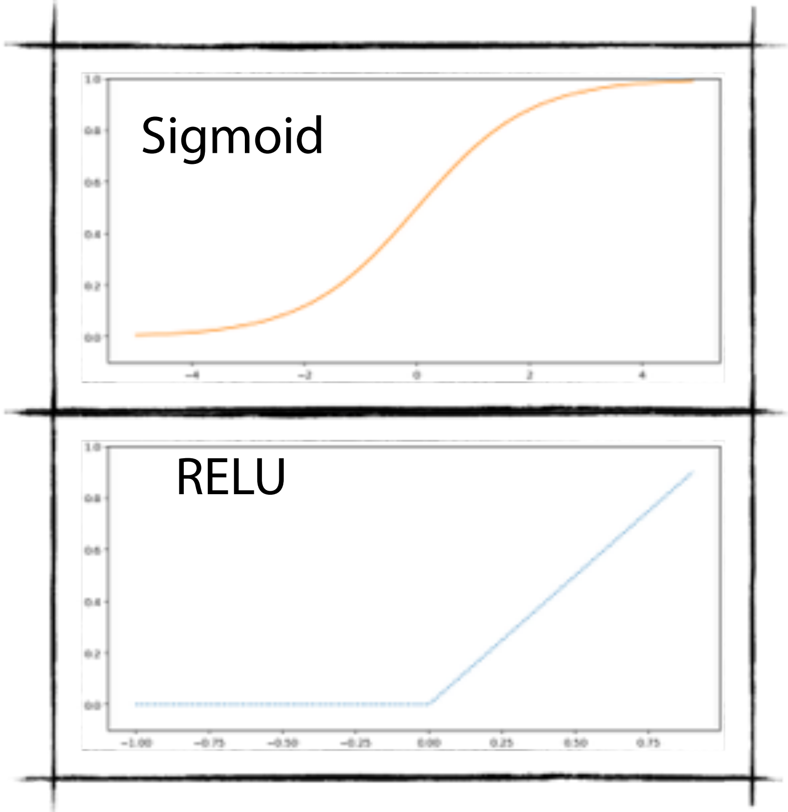


$$h\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right)$$

Artificial neural network — building blocks

neuron: affine map + activation function,   neuron w/o activation function = linear map

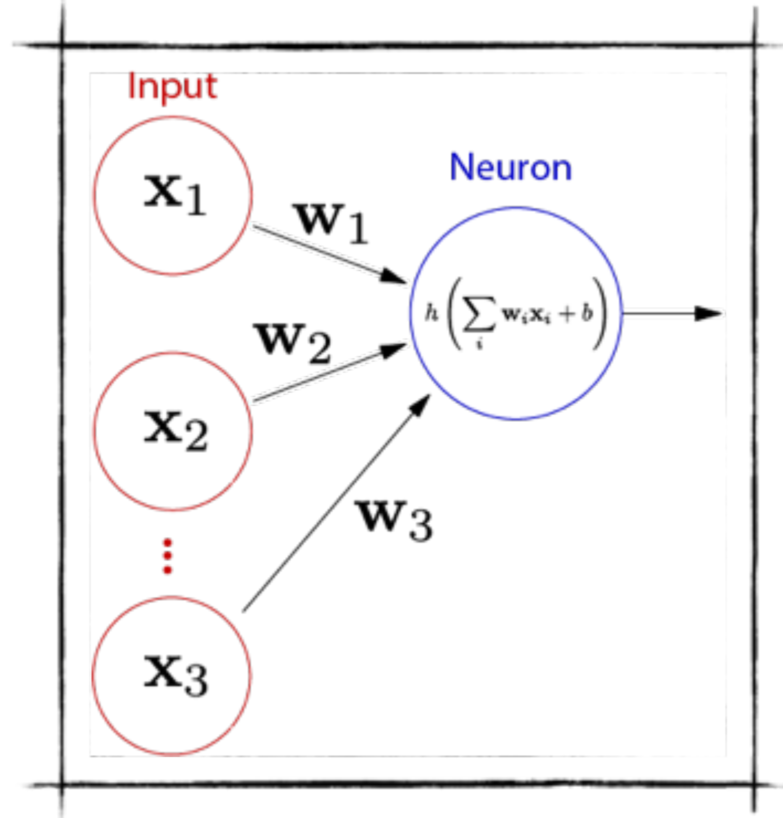
Activation function



$$\mathrm{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

$$\mathrm{relu}(x) = \max\{0, x\}$$

Neuron

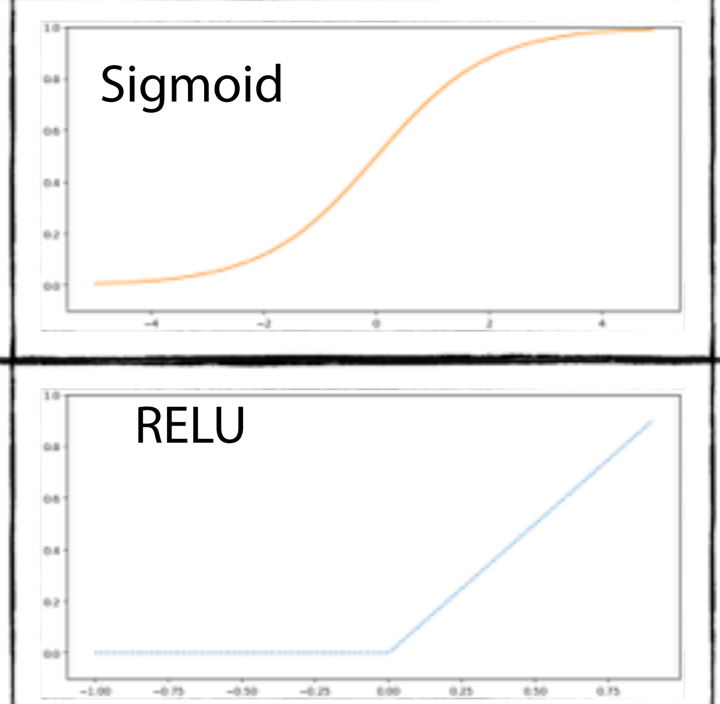


$$h\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right)$$

Artificial neural network — building blocks

neuron: affine map + activation function,   neuron w/o activation function = linear map

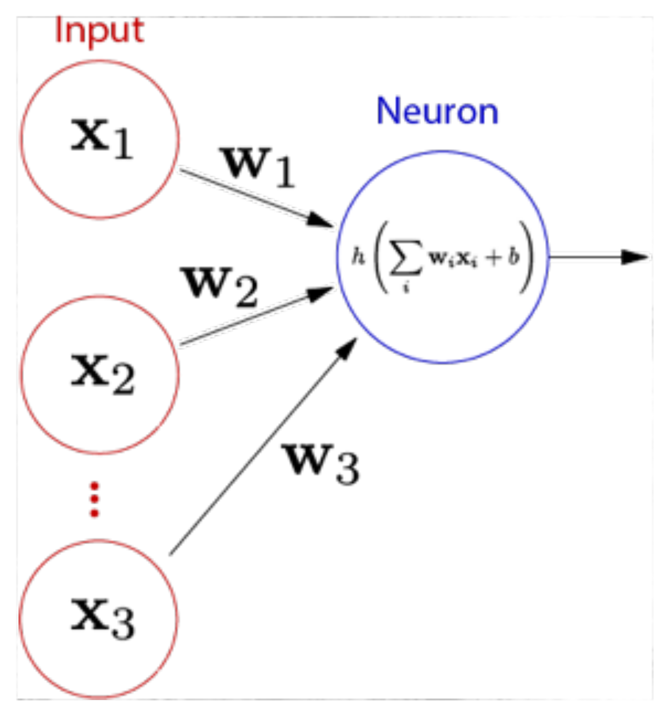
Activation function



$$\mathrm{sigmoid}(x) = \frac{1}{1+\exp(-x)}$$

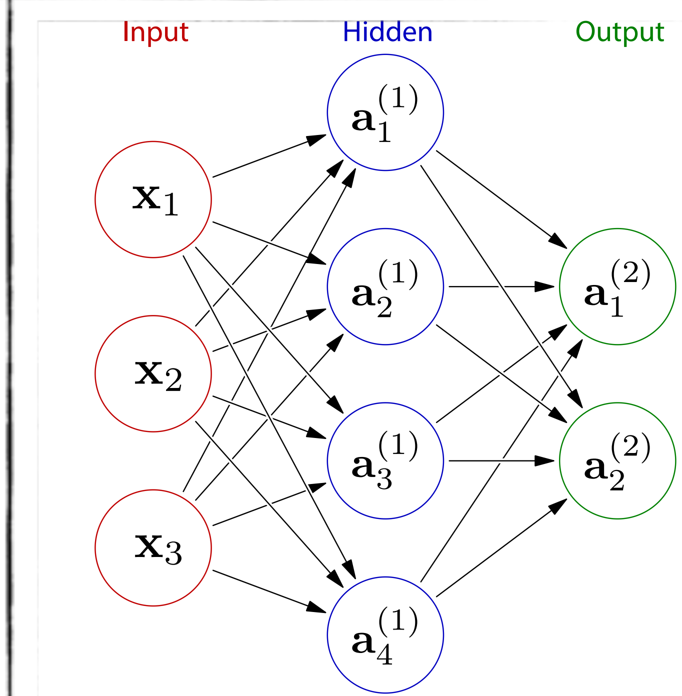
$$\mathrm{relu}(x) = \max\{0, x\}$$

Neuron



$$h\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right)$$

Fully-connected NN



$$\mathbf{a}_1^{(1)} = h\left(\mathbf{w}_{1,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{1,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{1,3}^{(1)}\mathbf{x}_3 + b_1^{(1)}\right)$$

$$\mathbf{a}_2^{(1)} = h\left(\mathbf{w}_{2,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{2,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{2,3}^{(1)}\mathbf{x}_3 + b_2^{(1)}\right)$$

$$\mathbf{a}_3^{(1)} = h\left(\mathbf{w}_{3,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{3,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{3,3}^{(1)}\mathbf{x}_3 + b_3^{(1)}\right)$$

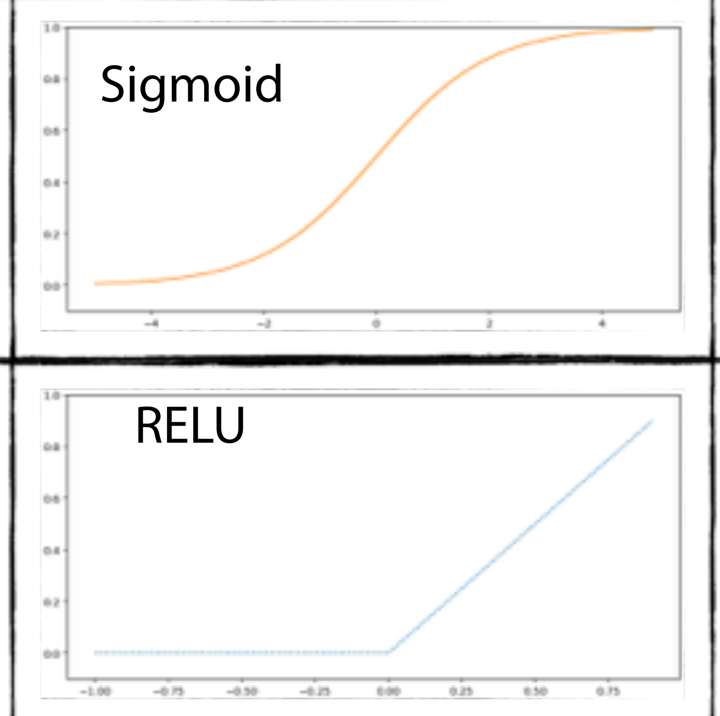
$$\mathbf{a}_1^{(2)} = \mathbf{w}_{1,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{1,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{1,3}^{(2)}\mathbf{a}_3^{(1)} + b_1^{(2)}$$

$$\mathbf{a}_2^{(2)} = \mathbf{w}_{2,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{2,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{2,3}^{(2)}\mathbf{a}_3^{(1)} + b_2^{(2)}$$

Artificial neural network — building blocks

neuron: affine map + activation function,   neuron w/o activation function = linear map

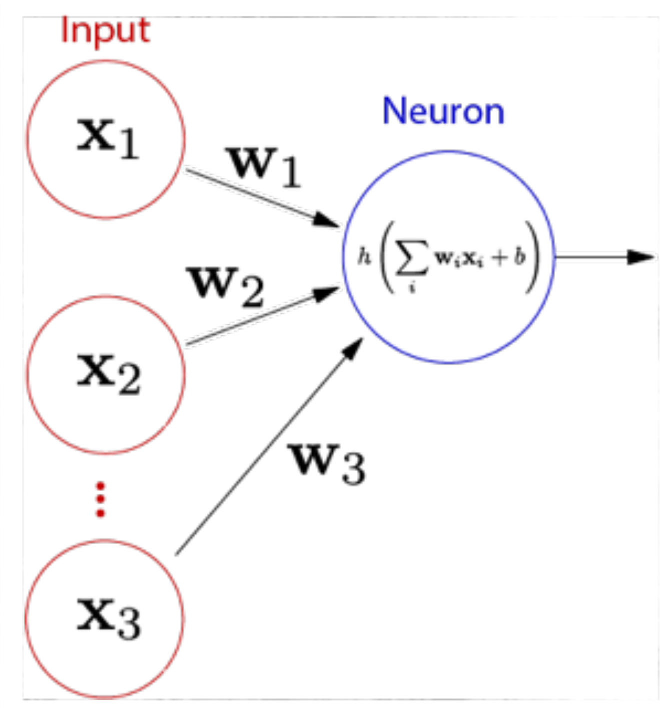
Activation function



$$\mathrm{sigmoid}(x) = \frac{1}{1+\exp(-x)}$$

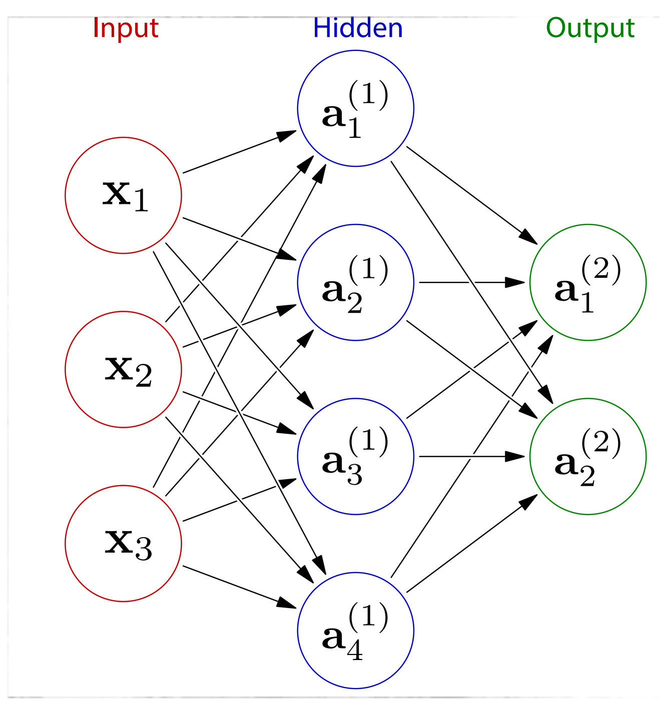
$$\mathrm{relu}(x) = \max\{0, x\}$$

Neuron



$$h\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right)$$

Fully-connected NN



$$\mathbf{a}_1^{(1)} = h\left(\mathbf{w}_{1,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{1,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{1,3}^{(1)}\mathbf{x}_3 + b_1^{(1)}\right)$$

$$\mathbf{a}_2^{(1)} = h\left(\mathbf{w}_{2,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{2,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{2,3}^{(1)}\mathbf{x}_3 + b_2^{(1)}\right)$$

$$\mathbf{a}_3^{(1)} = h\left(\mathbf{w}_{3,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{3,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{3,3}^{(1)}\mathbf{x}_3 + b_3^{(1)}\right)$$

$$\mathbf{a}_1^{(2)} = \mathbf{w}_{1,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{1,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{1,3}^{(2)}\mathbf{a}_3^{(1)} + b_1^{(2)}$$

$$\mathbf{a}_2^{(2)} = \mathbf{w}_{2,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{2,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{2,3}^{(2)}\mathbf{a}_3^{(1)} + b_2^{(2)}$$

Linear without activation function!

$$\mathbf{a}_1^{(2)} = \mathbf{w}_{1,1}^{(2)}(\mathbf{w}_{1,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{1,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{1,3}^{(1)}\mathbf{x}_3 + b_1^{(1)}) + \dots) + b_1^{(2)}$$

$$= c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + c_3\mathbf{x}_3 + c_4$$

Artificial neural network — a function

NNs are general they can be used as a black-box function that maps input to output.

Artificial neural network — a function

NNs are general they can be used as a black-box function that maps input to output.

Examples

Artificial neural network — a function

NNs are general they can be used as a black-box function that maps input to output.

Input domain →  → Output domain

Examples

Hello world $\xrightarrow{f(\mathbf{x}, \theta)}$ 你好，世界

Artificial neural network — a function

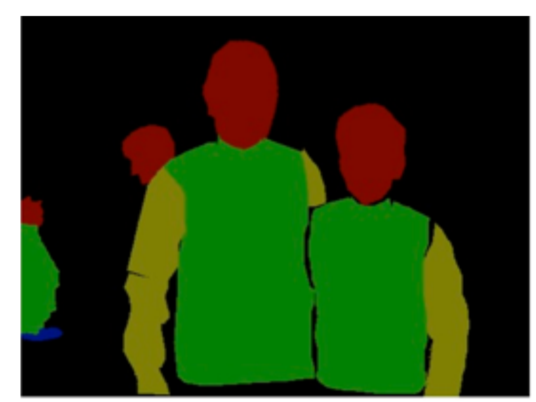
NNs are general they can be used as a black-box function that maps input to output.

Input domain →  → Output domain

Examples

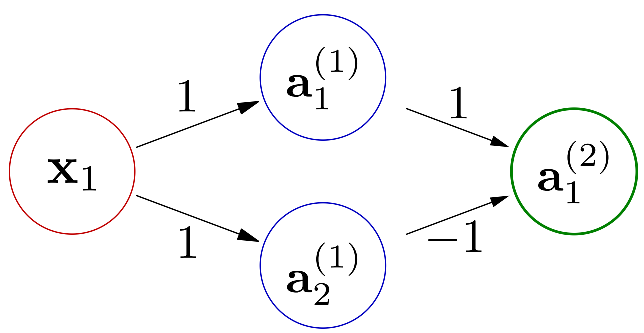
$$f(\mathbf{x}, \theta)$$

Hello world → 你好，世界

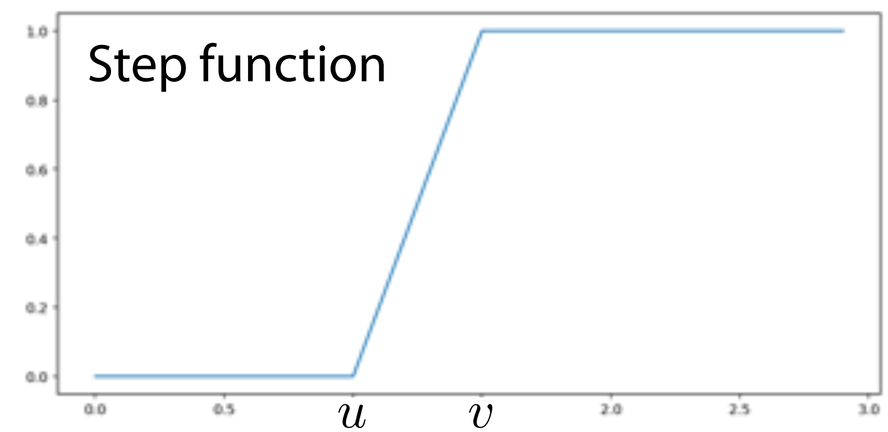
 → 

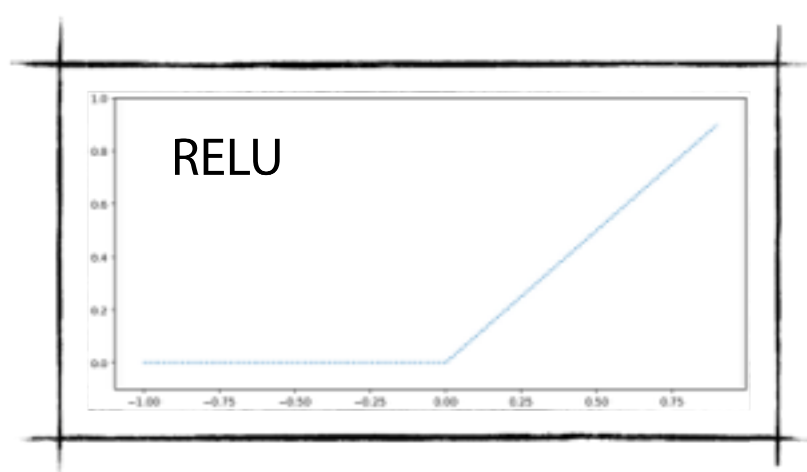
Approximation power — universal

Four neurons can form a box-function, multiple boxes can approximate continuous functions.



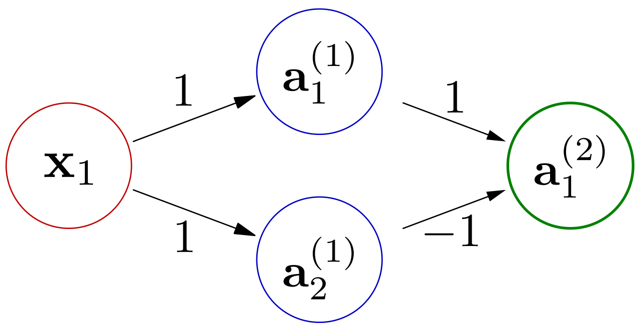
$$\mathbf{a}_1^{(1)} = \texttt{relu}(x - u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x - v)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)}$$



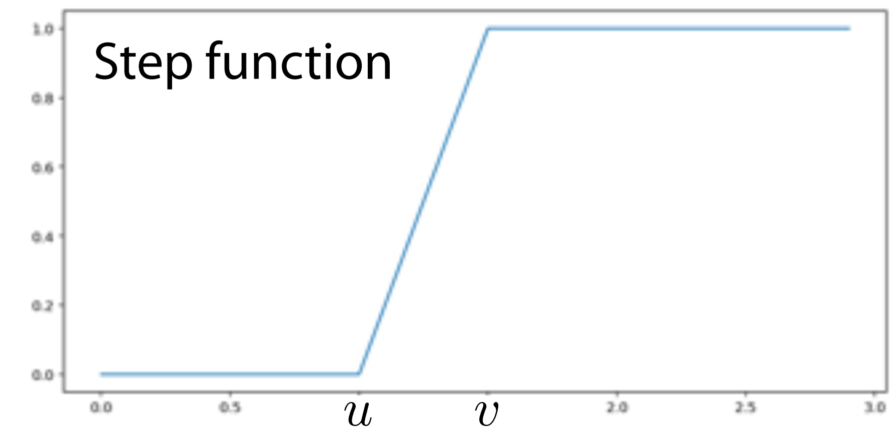


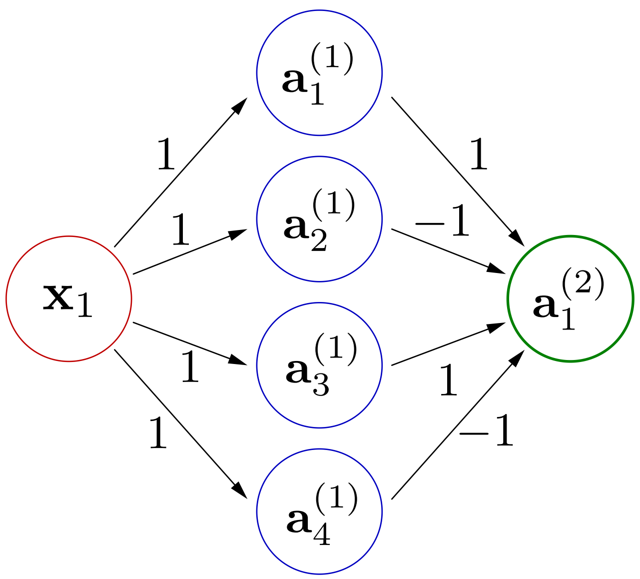
Approximation power — universal

Four neurons can form a box-function, multiple boxes can approximate continuous functions.

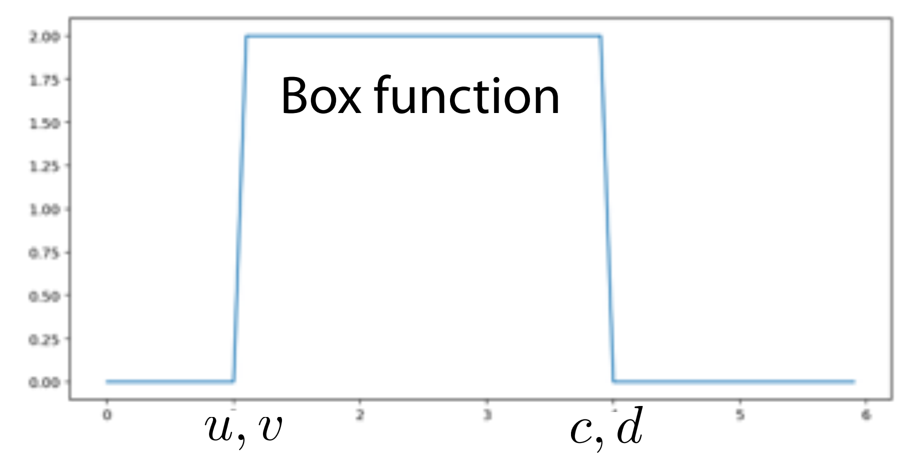


$$\mathbf{a}_1^{(1)} = \mathtt{relu}(x-u)$$
$$\mathbf{a}_2^{(1)} = \mathtt{relu}(x-v)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)}$$



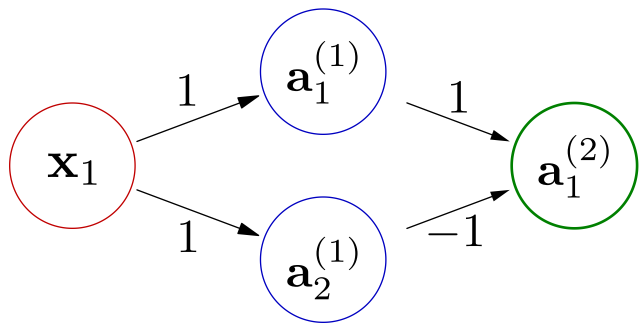


$$\mathbf{a}_1^{(1)} = \mathtt{relu}(x-u)$$
$$\mathbf{a}_2^{(1)} = \mathtt{relu}(x-v)$$
$$\mathbf{a}_3^{(1)} = \mathtt{relu}(x-c)$$
$$\mathbf{a}_4^{(1)} = \mathtt{relu}(x-d)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)} - (\mathbf{a}_3^{(1)} - \mathbf{a}_4^{(1)})$$

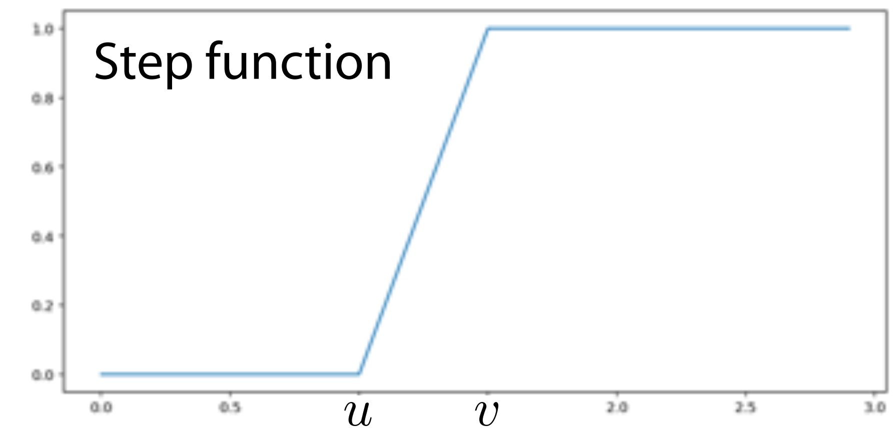


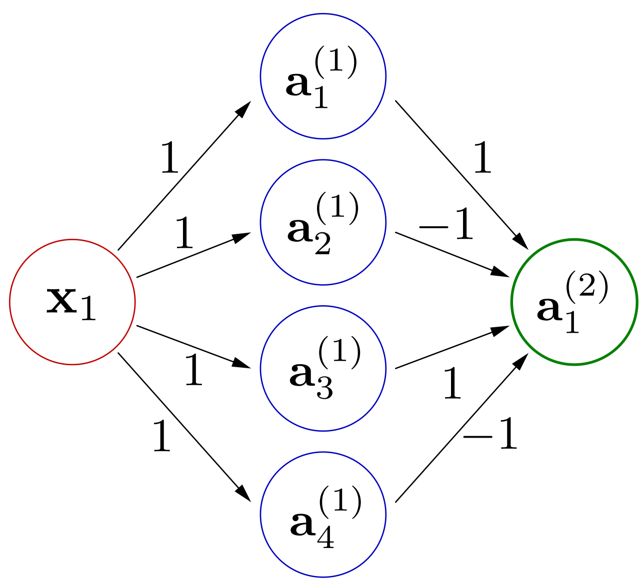
Approximation power — universal

Four neurons can form a box-function, multiple boxes can approximate continuous functions.

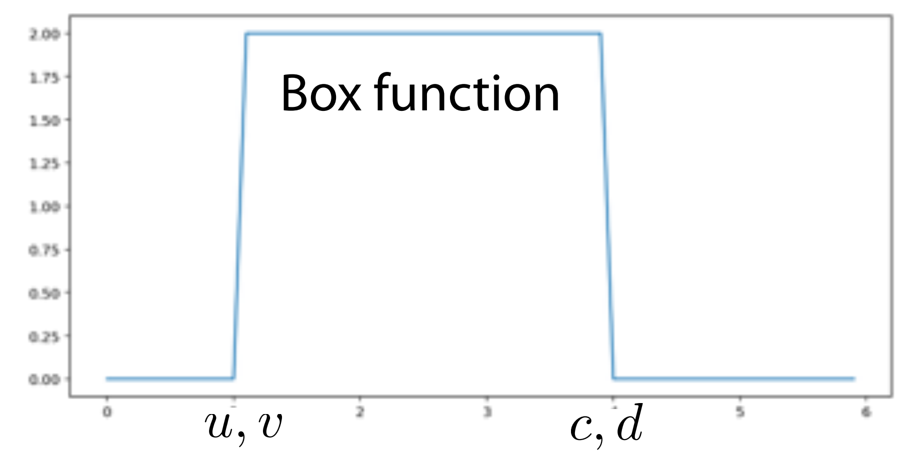


$$\mathbf{a}_1^{(1)} = \texttt{relu}(x - u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x - v)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)}$$

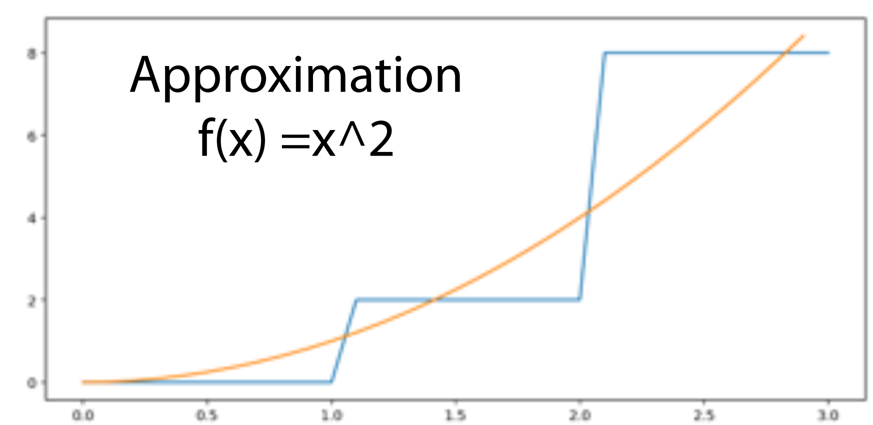




$$\mathbf{a}_1^{(1)} = \texttt{relu}(x - u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x - v)$$
$$\mathbf{a}_3^{(1)} = \texttt{relu}(x - c)$$
$$\mathbf{a}_4^{(1)} = \texttt{relu}(x - d)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)} - (\mathbf{a}_3^{(1)} - \mathbf{a}_4^{(1)})$$



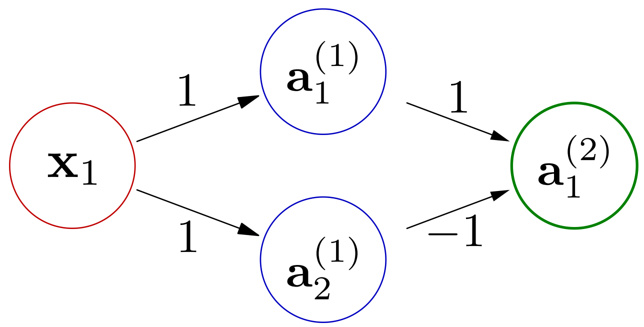
⋮



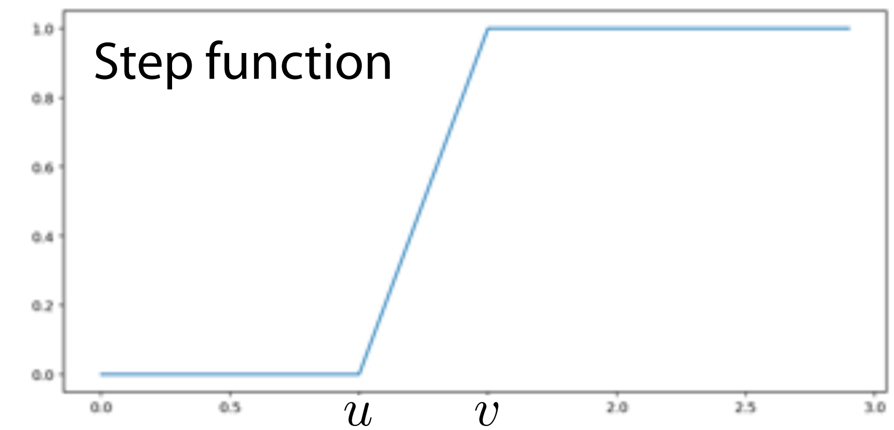
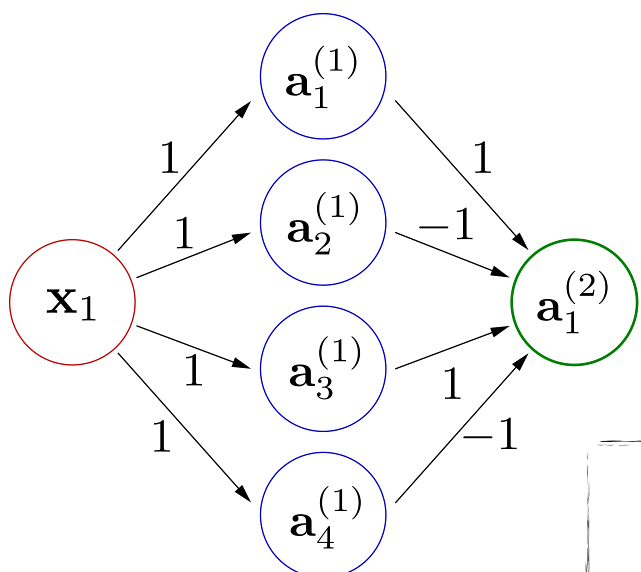
Mathematical prove in [Hornik et al., 1989; Cybenko, 1989]

Approximation power — universal

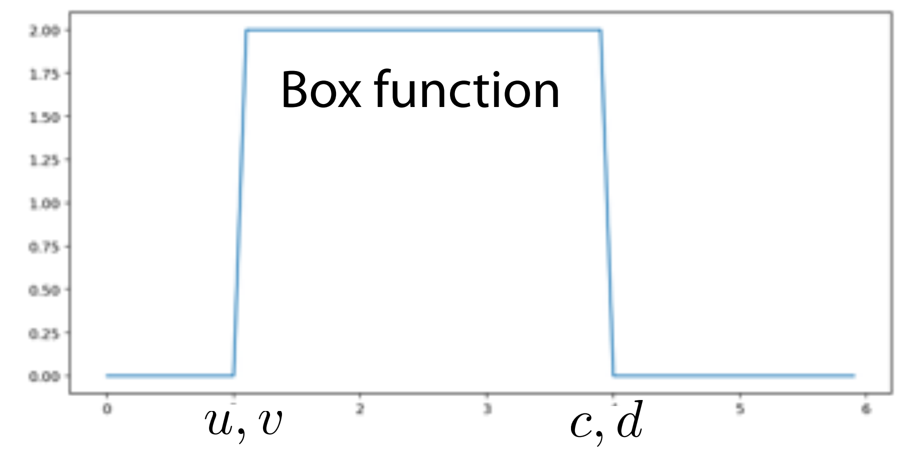
Four neurons can form a box-function, multiple boxes can approximate continuous functions.



$$\mathbf{a}_1^{(1)} = \texttt{relu}(x - u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x - v)$$
$$\mathbf{a}_1^{(2)} = \mathbf{a}_1^{(1)} - \mathbf{a}_2^{(1)}$$

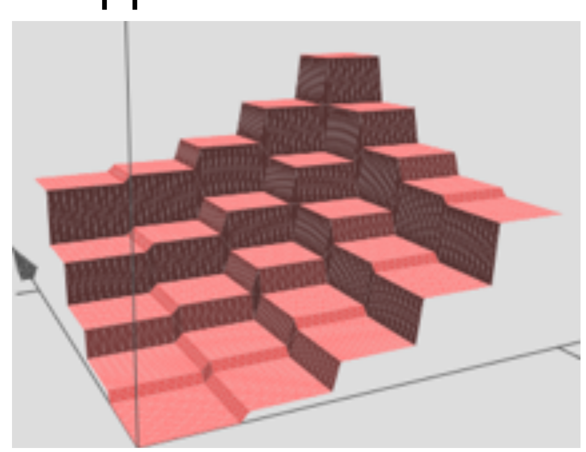



$$\mathbf{a}_1^{(1)} = \texttt{relu}(x - u)$$
$$\mathbf{a}_2^{(1)} = \texttt{relu}(x - v)$$
$$\mathbf{a}_3^{(1)} = \texttt{relu}(x - c)$$
$$\mathbf{a}_4^{(1)} = \texttt{relu}(x - d)$$

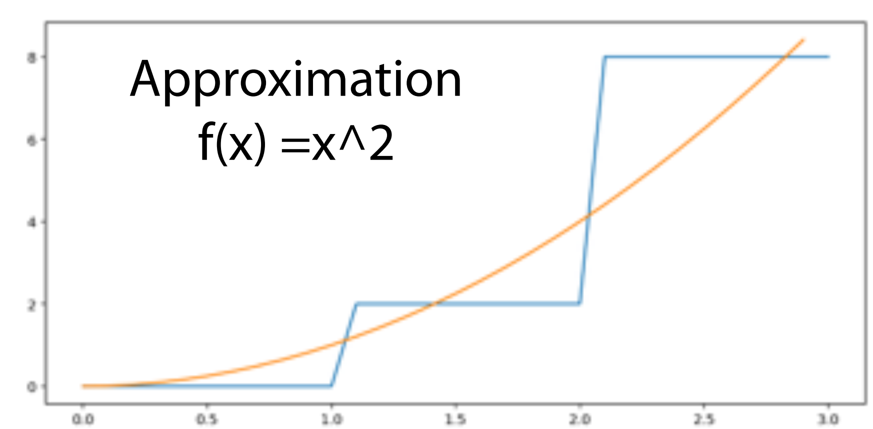


⋮

Approximation in 2D



[neuralnetworksanddeeplearning.com]

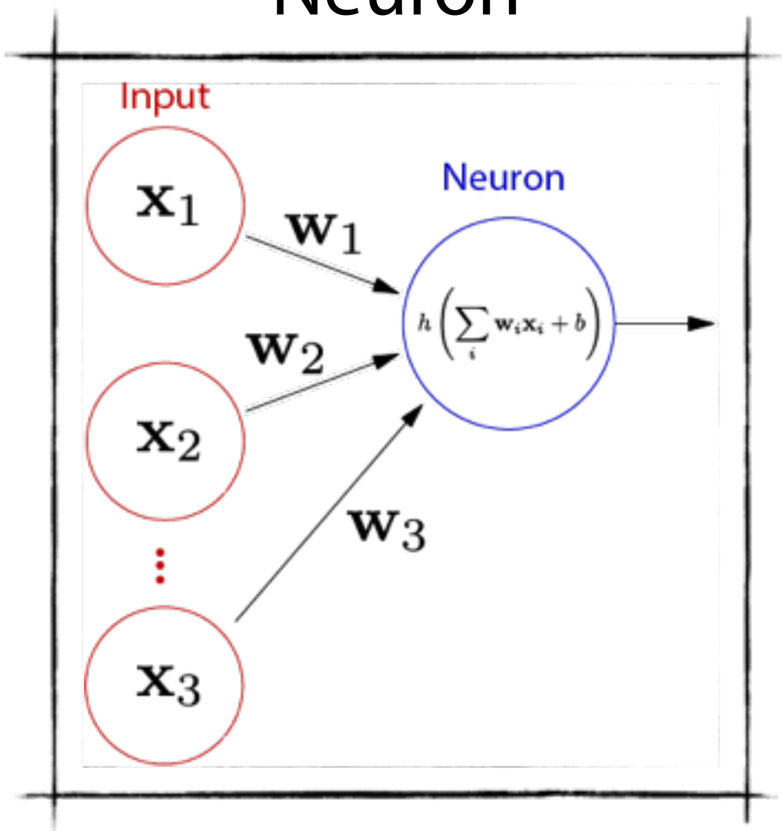


Mathematical prove in [Hornik et al., 1989; Cybenko, 1989]

Artificial neural network — structure

It is common and efficient to group neurons in layers and to use matrix-vector notation.

Neuron

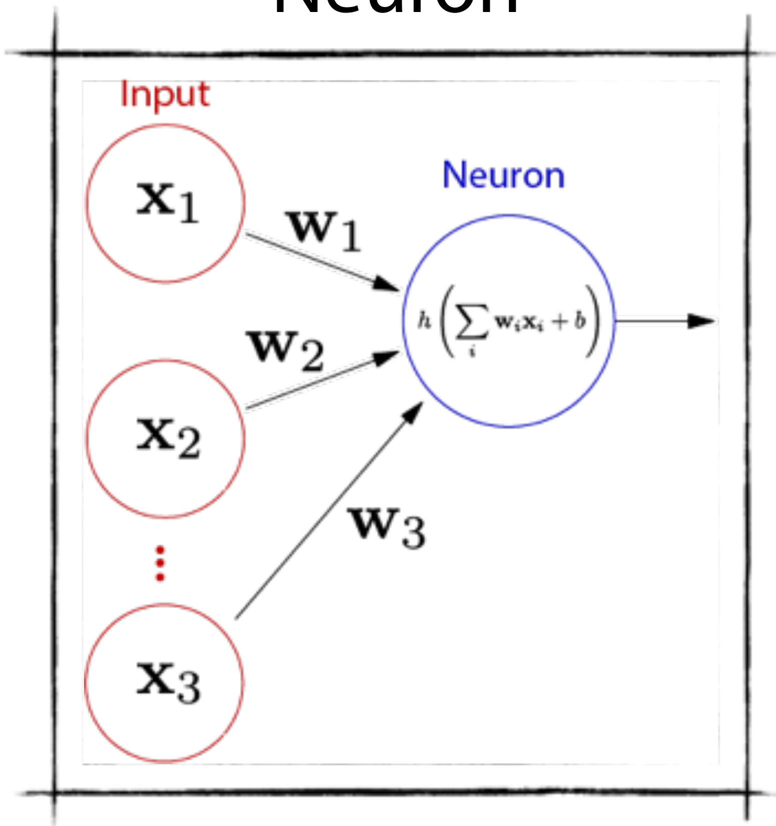


$$h\left(\sum_i \mathbf{w}_i\mathbf{x}_i + b\right)$$

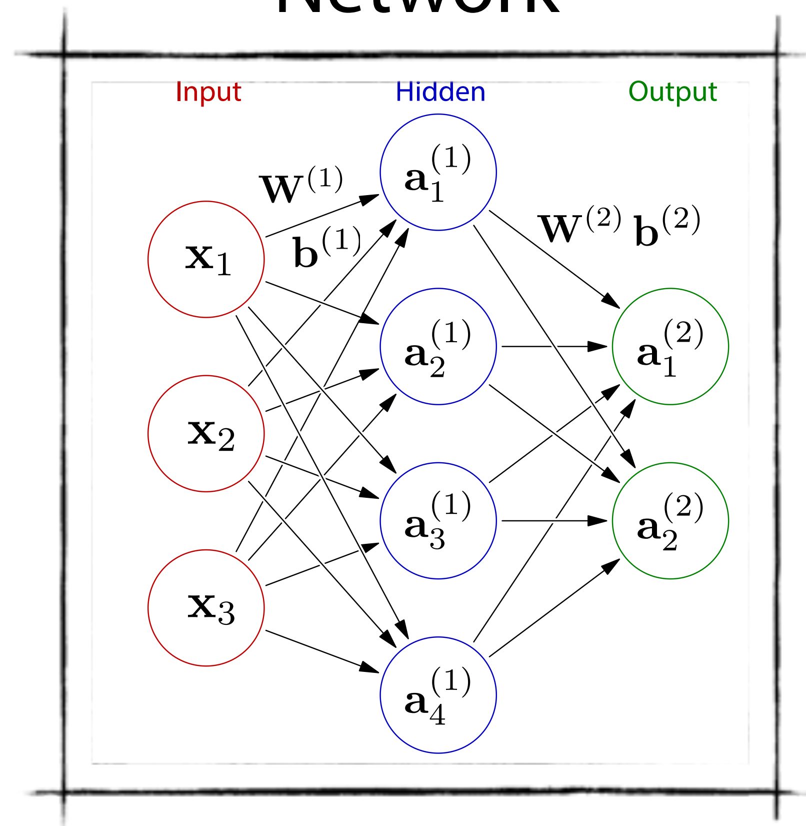
Artificial neural network — structure

It is common and efficient to group neurons in layers and to use matrix-vector notation.

Neuron



Network



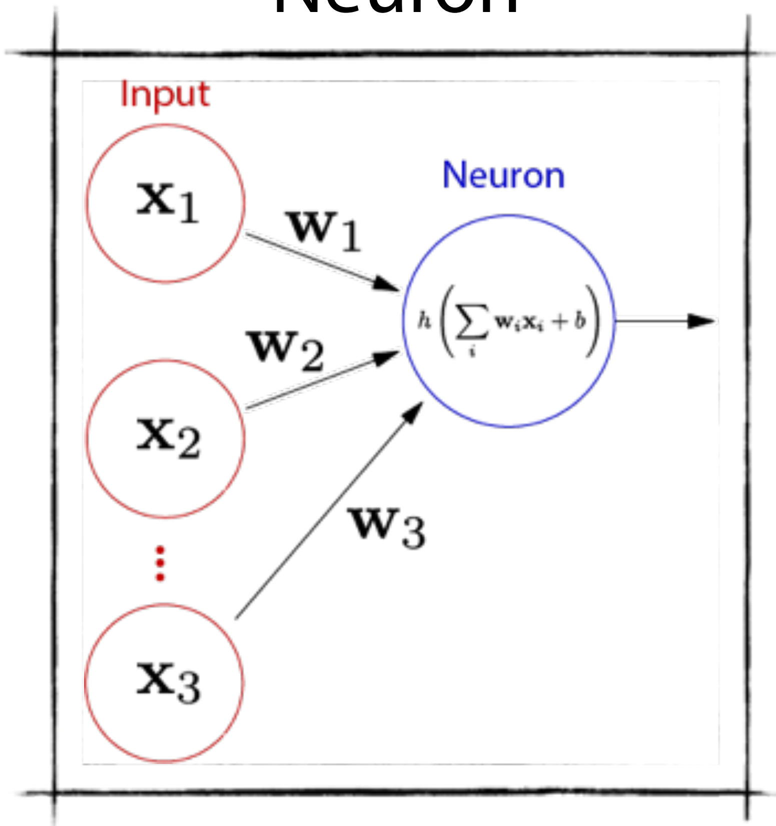
$$h\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right)$$

$$\begin{aligned}
\mathbf{a}_1^{(1)} &= h\left(\mathbf{w}_{1,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{1,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{1,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_1^{(1)}\right) \\
\mathbf{a}_2^{(1)} &= h\left(\mathbf{w}_{2,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{2,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{2,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_2^{(1)}\right) \\
\mathbf{a}_3^{(1)} &= h\left(\mathbf{w}_{3,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{3,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{3,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_3^{(1)}\right) \\
\mathbf{a}_1^{(2)} &= \quad \mathbf{w}_{1,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{1,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{1,3}^{(2)}\mathbf{a}_3^{(1)} + \mathbf{b}_1^{(2)} \\
\mathbf{a}_2^{(2)} &= \quad \mathbf{w}_{2,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{2,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{2,3}^{(2)}\mathbf{a}_3^{(1)} + \mathbf{b}_2^{(2)}
\end{aligned}$$

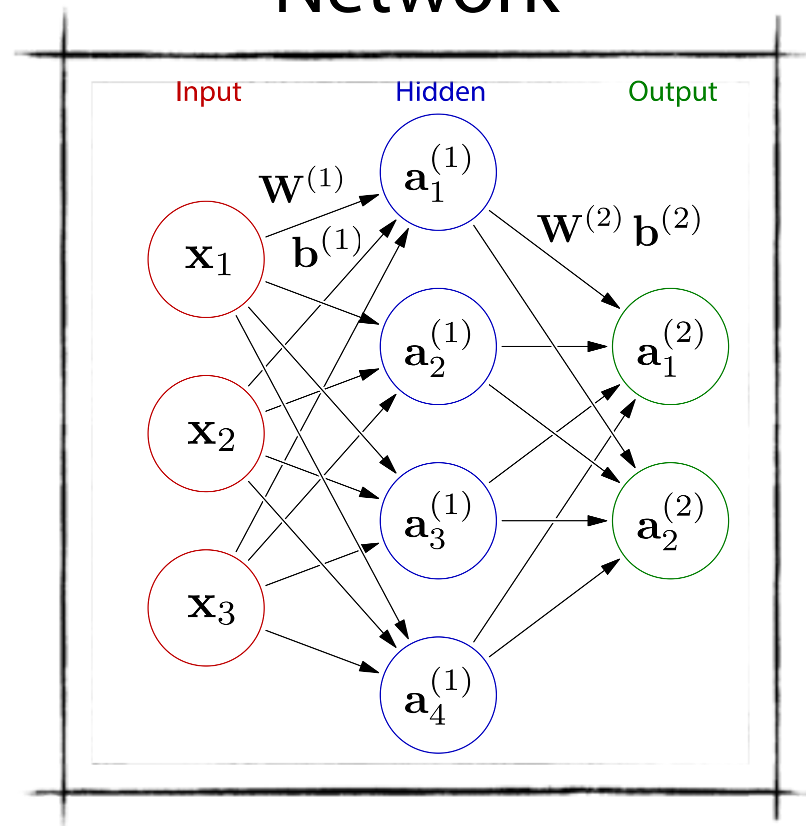
Artificial neural network — structure

It is common and efficient to group neurons in layers and to use matrix-vector notation.

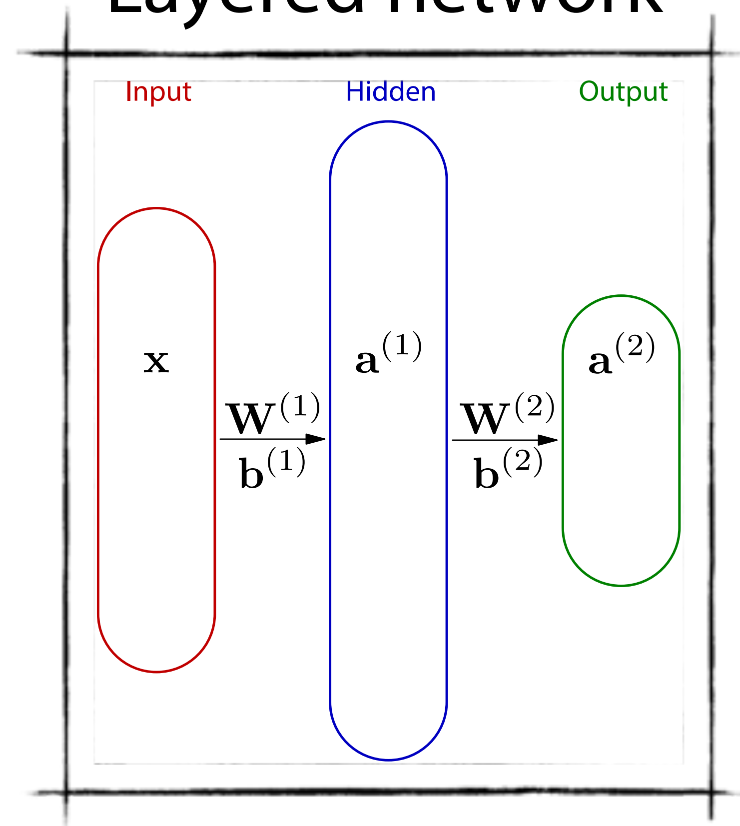
Neuron



Network



Layered network



$$h\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right)$$

$$\mathbf{a}_1^{(1)} = h\left(\mathbf{w}_{1,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{1,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{1,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_1^{(1)}\right)$$

$$\mathbf{a}_2^{(1)} = h\left(\mathbf{w}_{2,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{2,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{2,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_2^{(1)}\right)$$

$$\mathbf{a}_3^{(1)} = h\left(\mathbf{w}_{3,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{3,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{3,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_3^{(1)}\right)$$

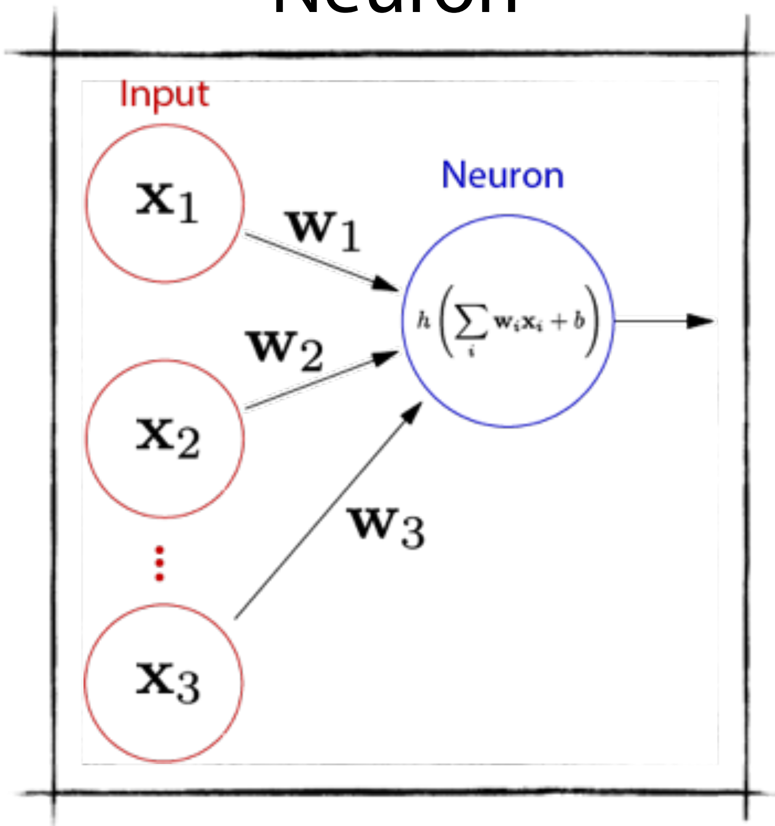
$$\mathbf{a}_1^{(2)} = \mathbf{w}_{1,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{1,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{1,3}^{(2)}\mathbf{a}_3^{(1)} + \mathbf{b}_1^{(2)}$$

$$\mathbf{a}_2^{(2)} = \mathbf{w}_{2,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{2,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{2,3}^{(2)}\mathbf{a}_3^{(1)} + \mathbf{b}_2^{(2)}$$

Artificial neural network — structure

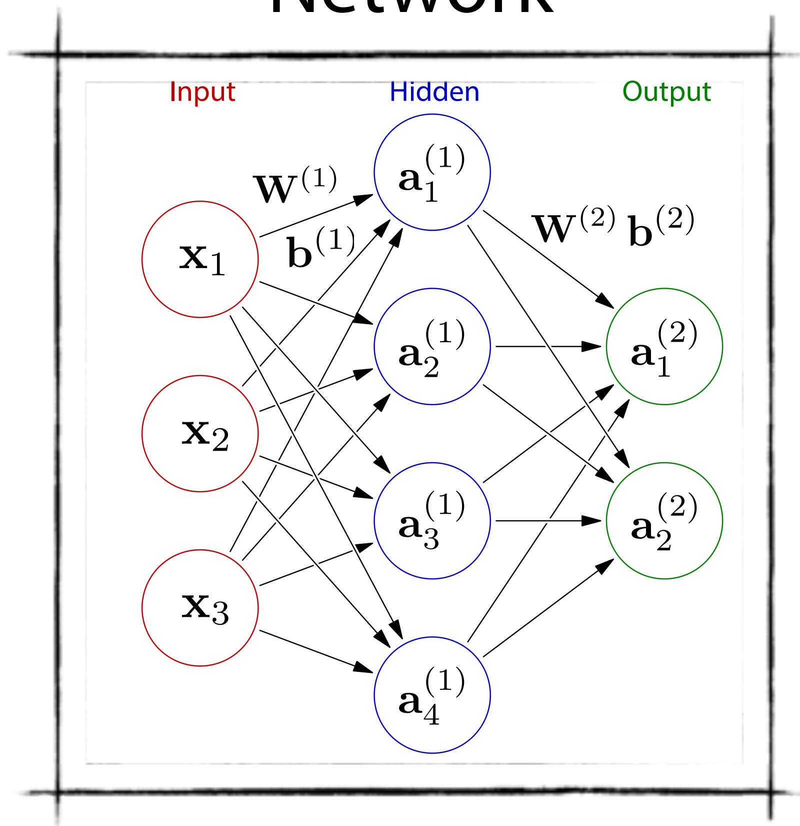
It is common and efficient to group neurons in layers and to use matrix-vector notation.

Neuron



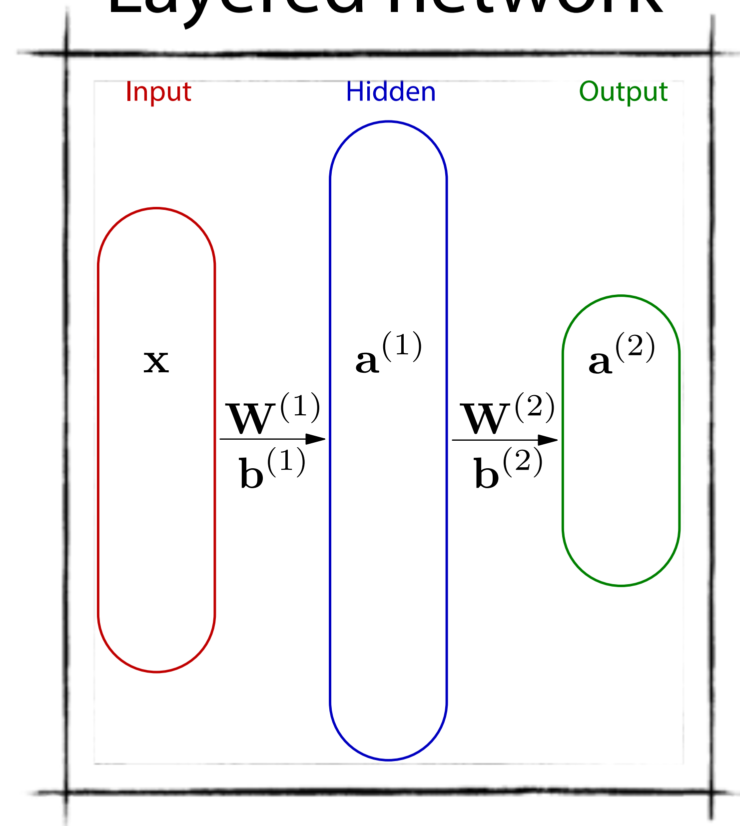
$$h\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right)$$

Network



$$\mathbf{a}_1^{(1)} = h\left(\mathbf{w}_{1,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{1,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{1,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_1^{(1)}\right)$$
$$\mathbf{a}_2^{(1)} = h\left(\mathbf{w}_{2,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{2,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{2,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_2^{(1)}\right)$$
$$\mathbf{a}_3^{(1)} = h\left(\mathbf{w}_{3,1}^{(1)}\mathbf{x}_1 + \mathbf{w}_{3,2}^{(1)}\mathbf{x}_2 + \mathbf{w}_{3,3}^{(1)}\mathbf{x}_3 + \mathbf{b}_3^{(1)}\right)$$
$$\mathbf{a}_1^{(2)} = \mathbf{w}_{1,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{1,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{1,3}^{(2)}\mathbf{a}_3^{(1)} + \mathbf{b}_1^{(2)}$$
$$\mathbf{a}_2^{(2)} = \mathbf{w}_{2,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{2,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{2,3}^{(2)}\mathbf{a}_3^{(1)} + \mathbf{b}_2^{(2)}$$

Layered network



$$\mathbf{a}^{(1)} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{a}^{(2)} = \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}$$

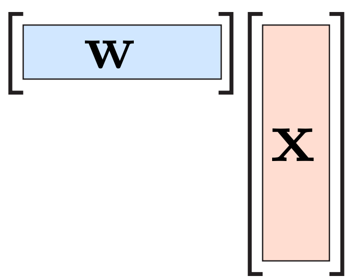
Linear and affine transformations

An affine map is a linear map plus an offset = a linear map with an augmented vector.

Linear

$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i$$

$$= \mathbf{w} \cdot \mathbf{x}$$



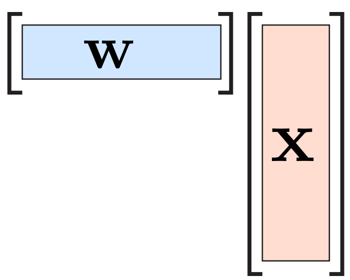
Linear and affine transformations

An affine map is a linear map plus an offset = a linear map with an augmented vector.

Linear

$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i$$

$$= \mathbf{w} \cdot \mathbf{x}$$

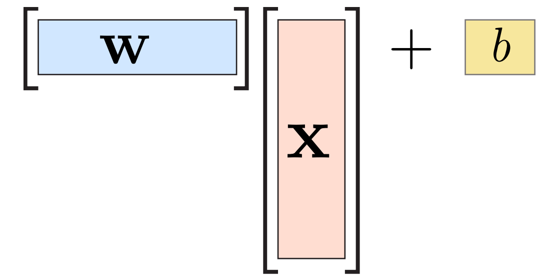


Affine

$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i + b$$

$$= \mathbf{w} \cdot \mathbf{x} + b$$

$$= \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

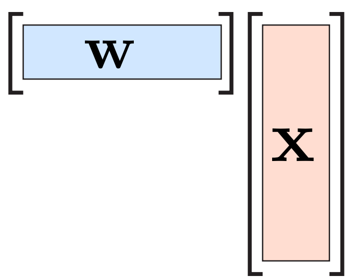


Linear and affine transformations

An affine map is a linear map plus an offset = a linear map with an augmented vector.

Linear

$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i$$
$$= \mathbf{w} \cdot \mathbf{x}$$

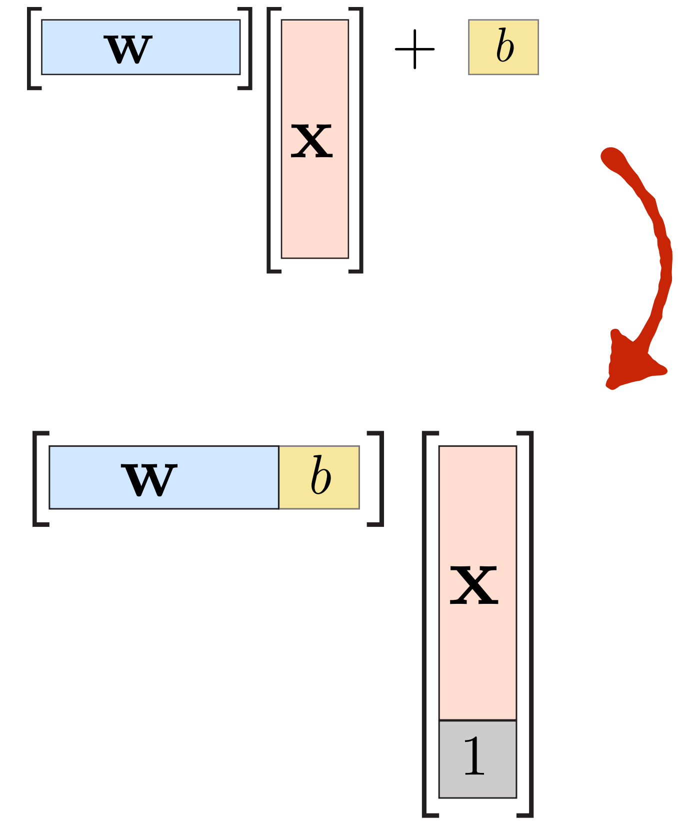


Affine

$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i + b$$
$$= \mathbf{w} \cdot \mathbf{x} + b$$
$$= \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

with $\tilde{\mathbf{w}} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n, b)$

and $\tilde{\mathbf{x}} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, 1)$

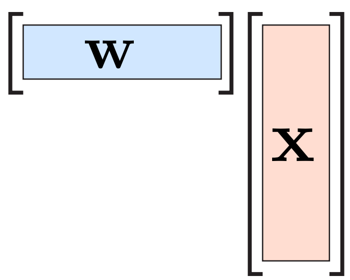


Linear and affine transformations

An affine map is a linear map plus an offset = a linear map with an augmented vector.

Linear

$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i$$
$$= \mathbf{w} \cdot \mathbf{x}$$

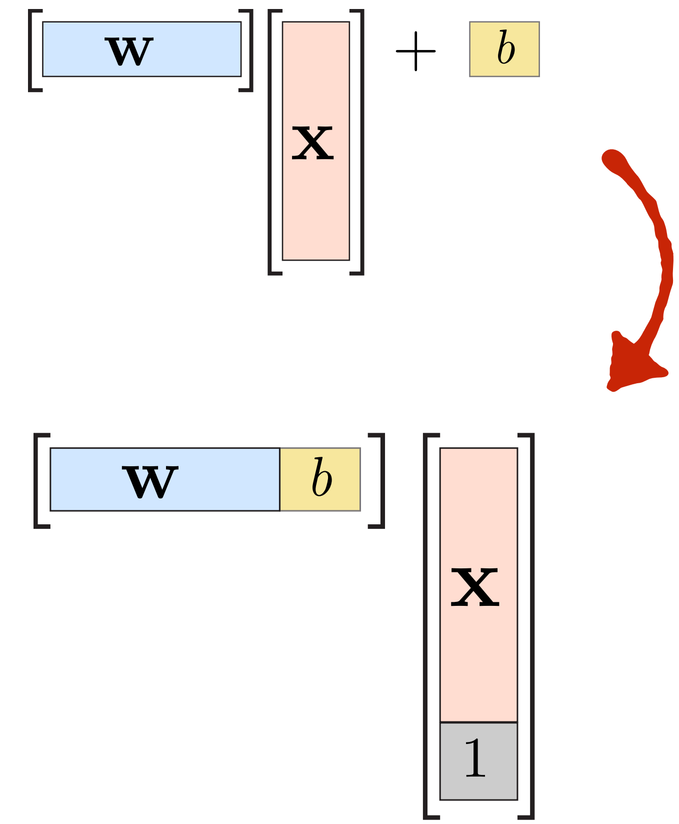


Affine

$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i + b$$
$$= \mathbf{w} \cdot \mathbf{x} + b$$
$$= \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

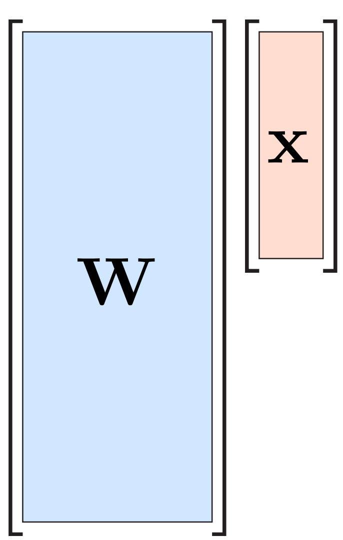
with $\tilde{\mathbf{w}} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n, b)$

and $\tilde{\mathbf{x}} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, 1)$



Multidimensional

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x}$$

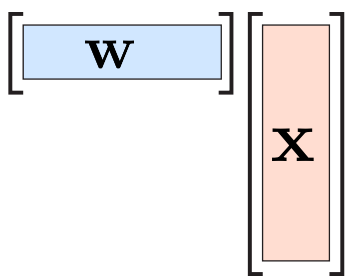


Linear and affine transformations

An affine map is a linear map plus an offset = a linear map with an augmented vector.

Linear

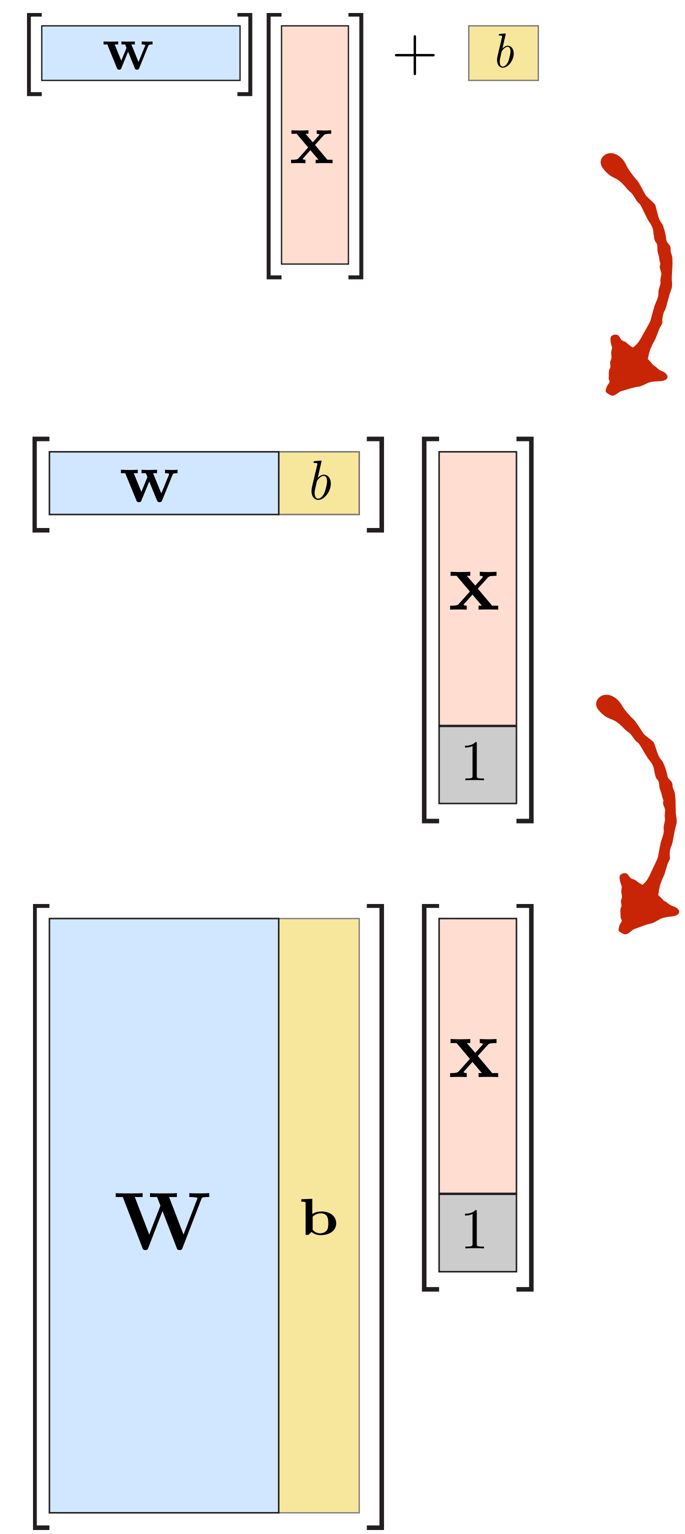
$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i$$
$$= \mathbf{w} \cdot \mathbf{x}$$



Affine

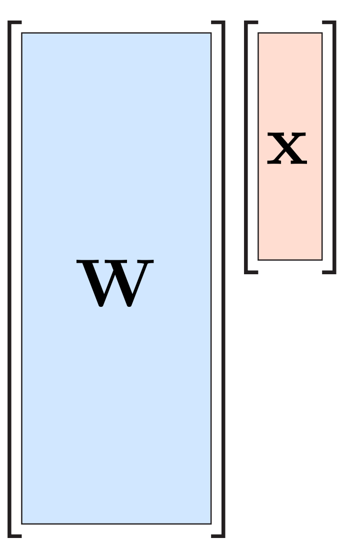
$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i + b$$
$$= \mathbf{w} \cdot \mathbf{x} + b$$
$$= \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

with $\tilde{\mathbf{w}} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n, b)$
and $\tilde{\mathbf{x}} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, 1)$



Multidimensional

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x}$$



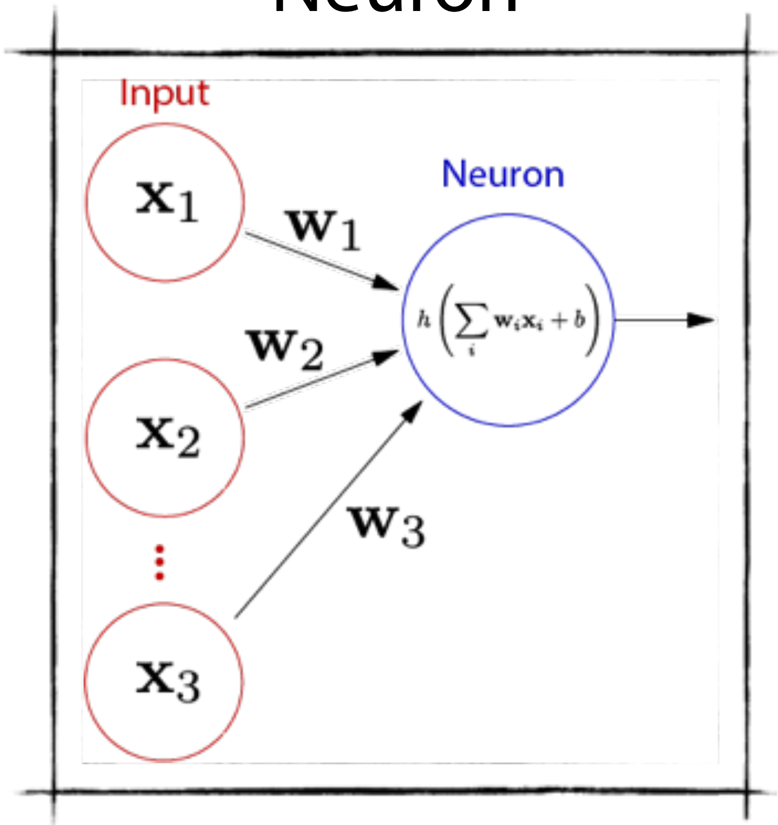
$$f(\mathbf{x}) = \tilde{\mathbf{W}} \cdot \tilde{\mathbf{x}}$$

$$\text{with } \tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{w}_{1,1} & \mathbf{w}_{1,2} & \cdots & \mathbf{w}_{1,n} & b_1 \\ \mathbf{w}_{2,1} & \mathbf{w}_{2,2} & \cdots & \mathbf{w}_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \end{pmatrix}$$

Artificial neural network — matrix notation

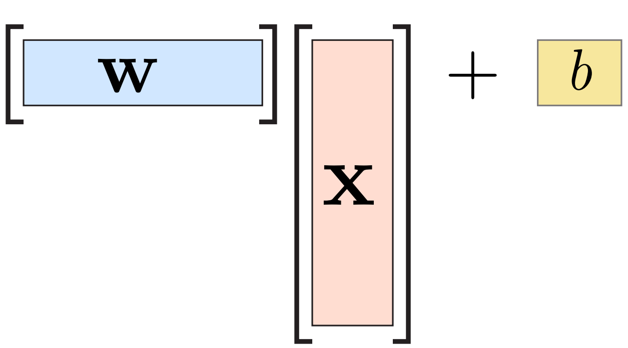
Layered NNs are simply a chain of matrix multiplications and activation functions.

Neuron



$$h(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

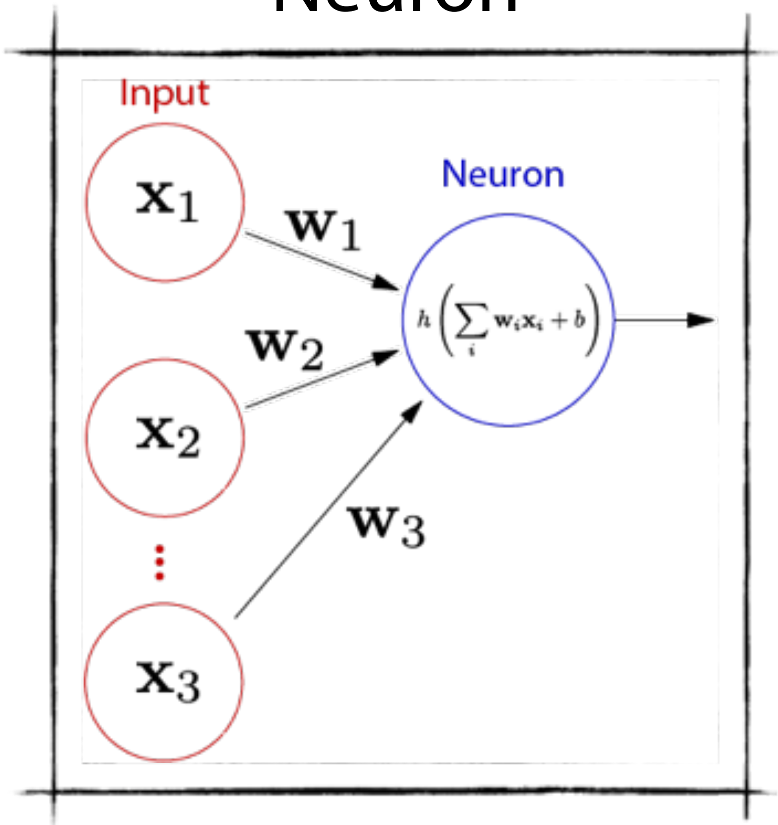
Matrix representation



Artificial neural network — matrix notation

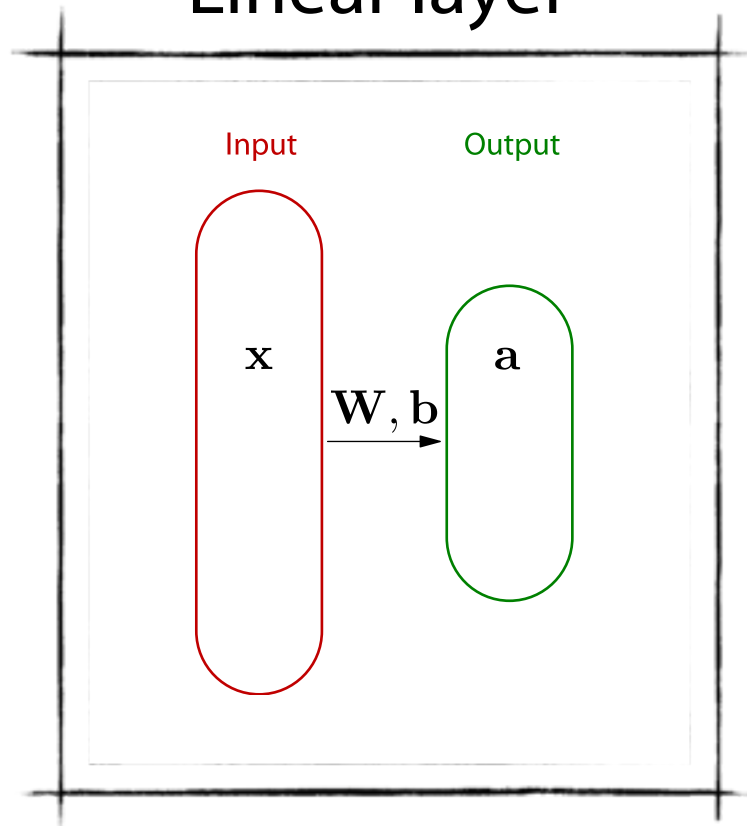
Layered NNs are simply a chain of matrix multiplications and activation functions.

Neuron



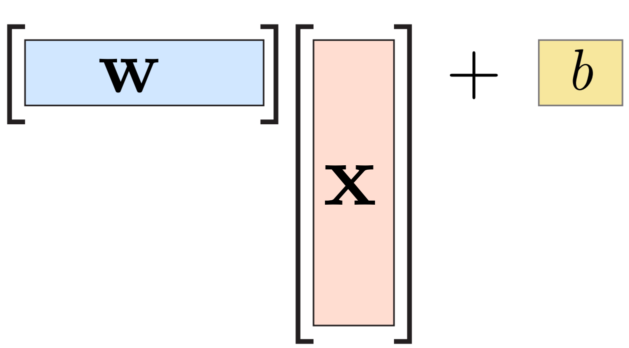
$$h(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

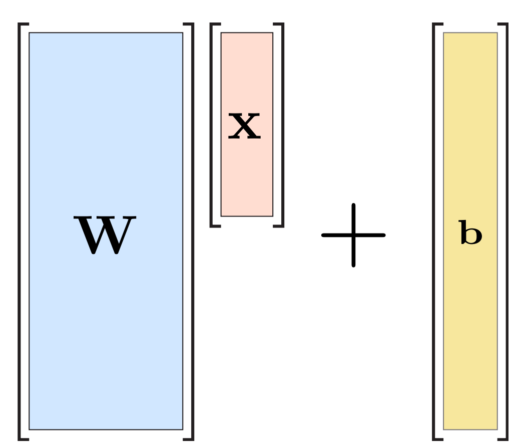
Linear layer



$$\begin{aligned}\mathbf{a} &= h(\mathbf{W}\mathbf{x} + \mathbf{b}) \\ &= h(\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b}))\end{aligned}$$

Matrix representation

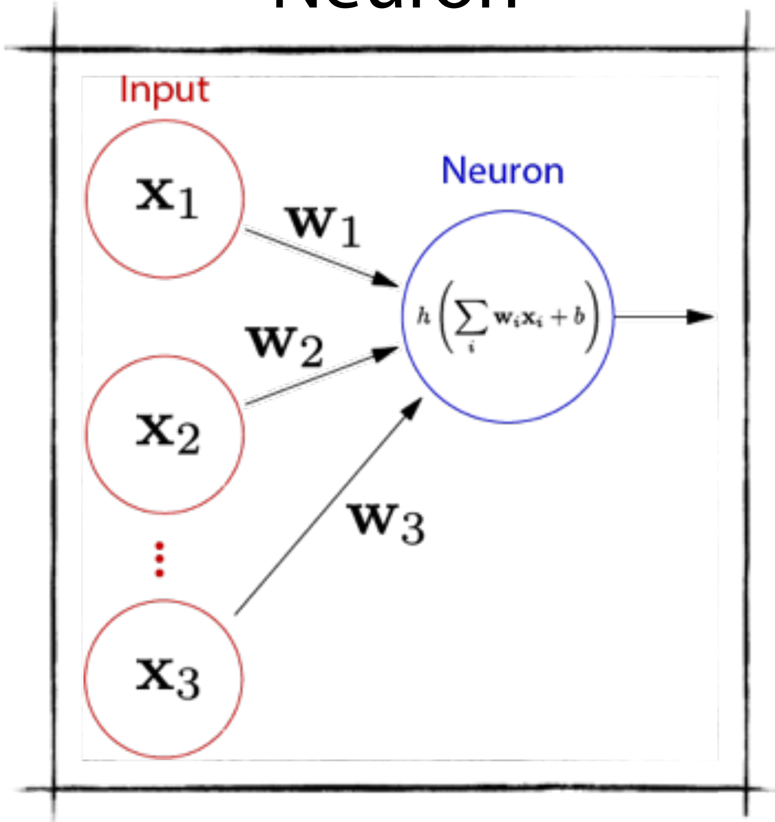




Artificial neural network — matrix notation

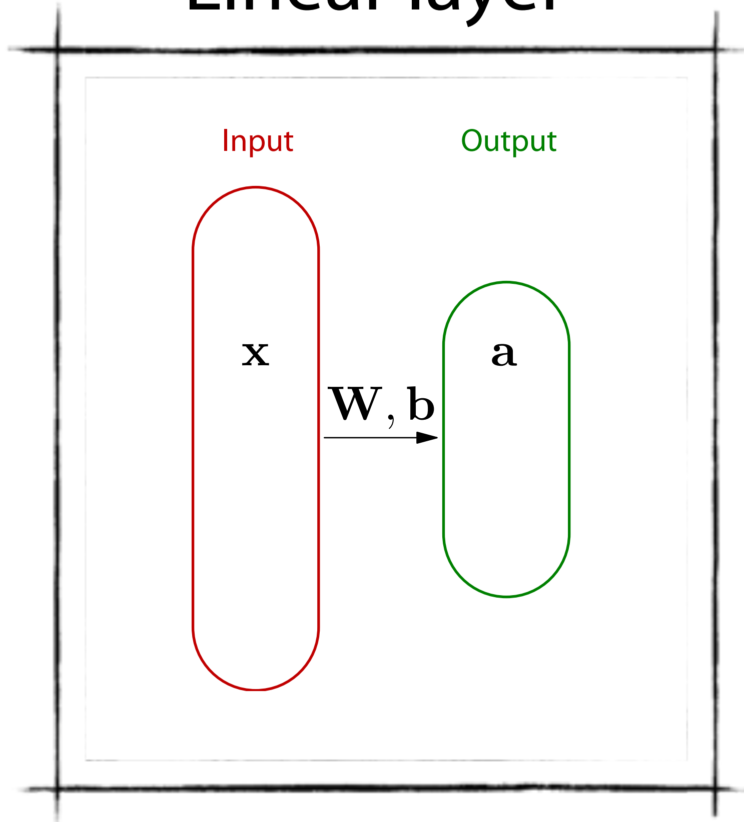
Layered NNs are simply a chain of matrix multiplications and activation functions.

Neuron



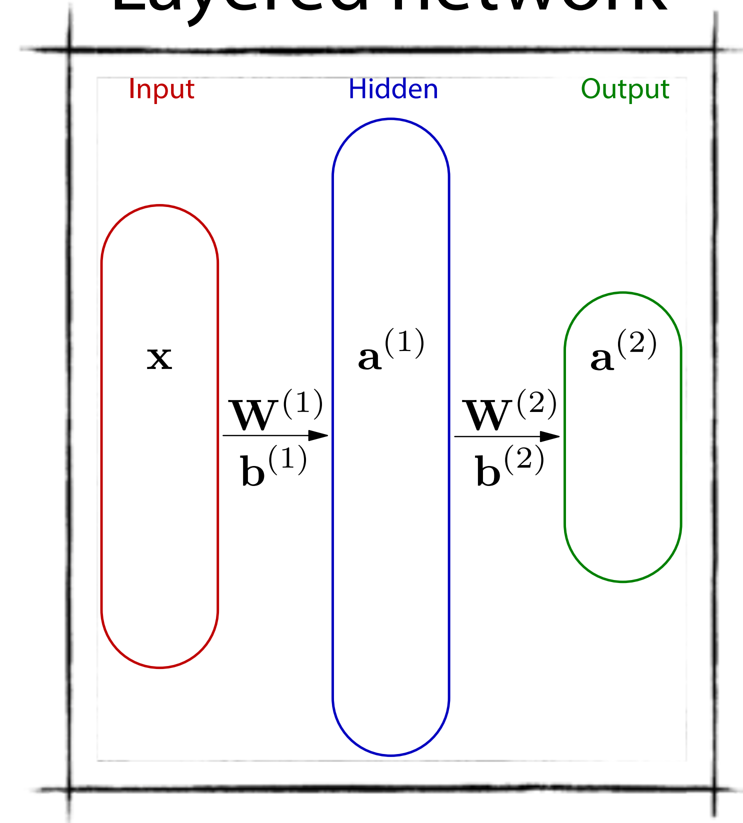
$$h(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

Linear layer



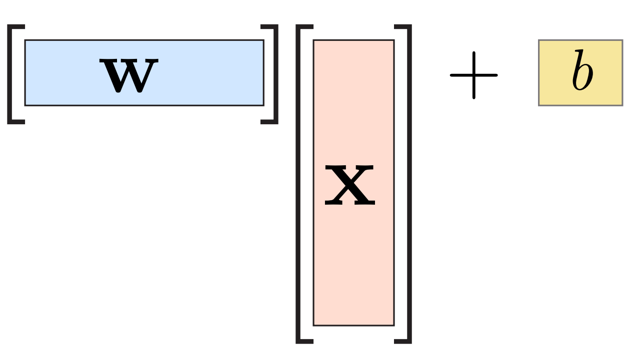
$$\mathbf{a} = h(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$= h(\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b}))$$

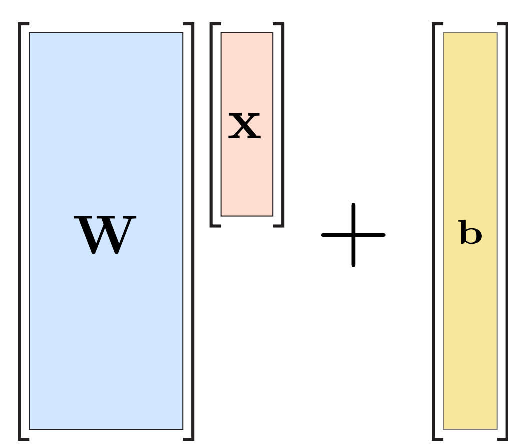
Layered network

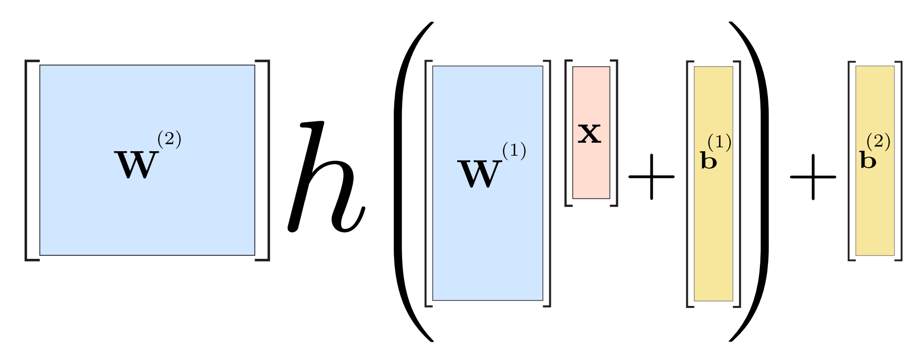


$$\mathbf{a}^{(1)} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{a}^{(2)} = \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}$$

Matrix representation



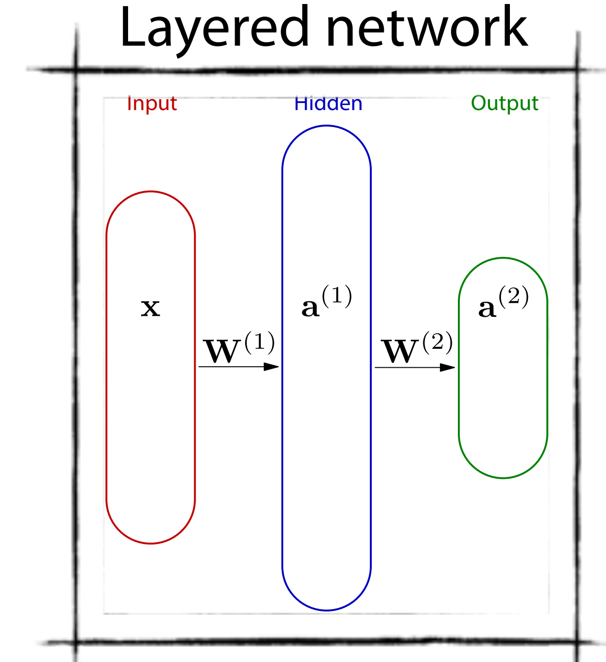




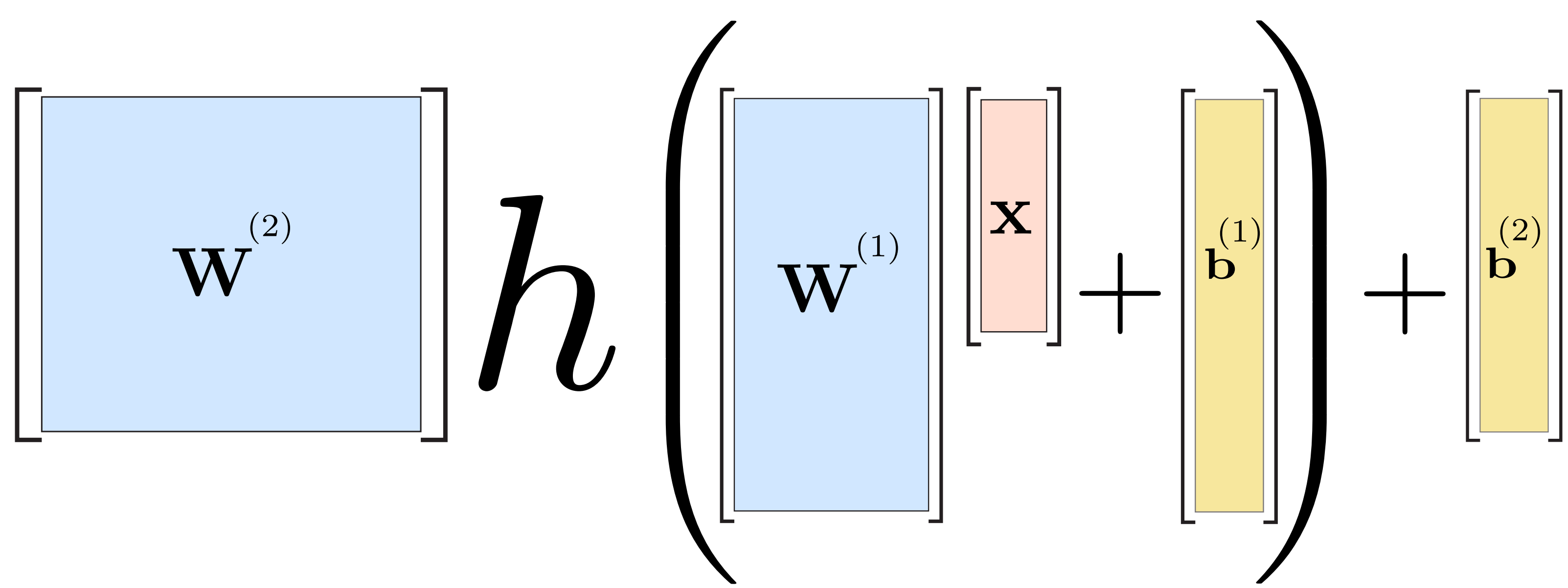
Artificial neural network — matrix notation

Layered NNs are simply a chain of matrix multiplications and activation functions.

Layered network



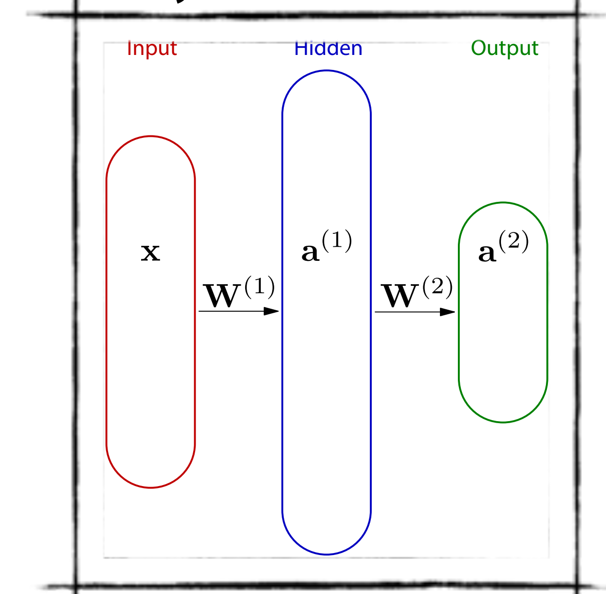
$$\mathbf{a}^{(1)} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{a}^{(2)} = h(\mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)})$$



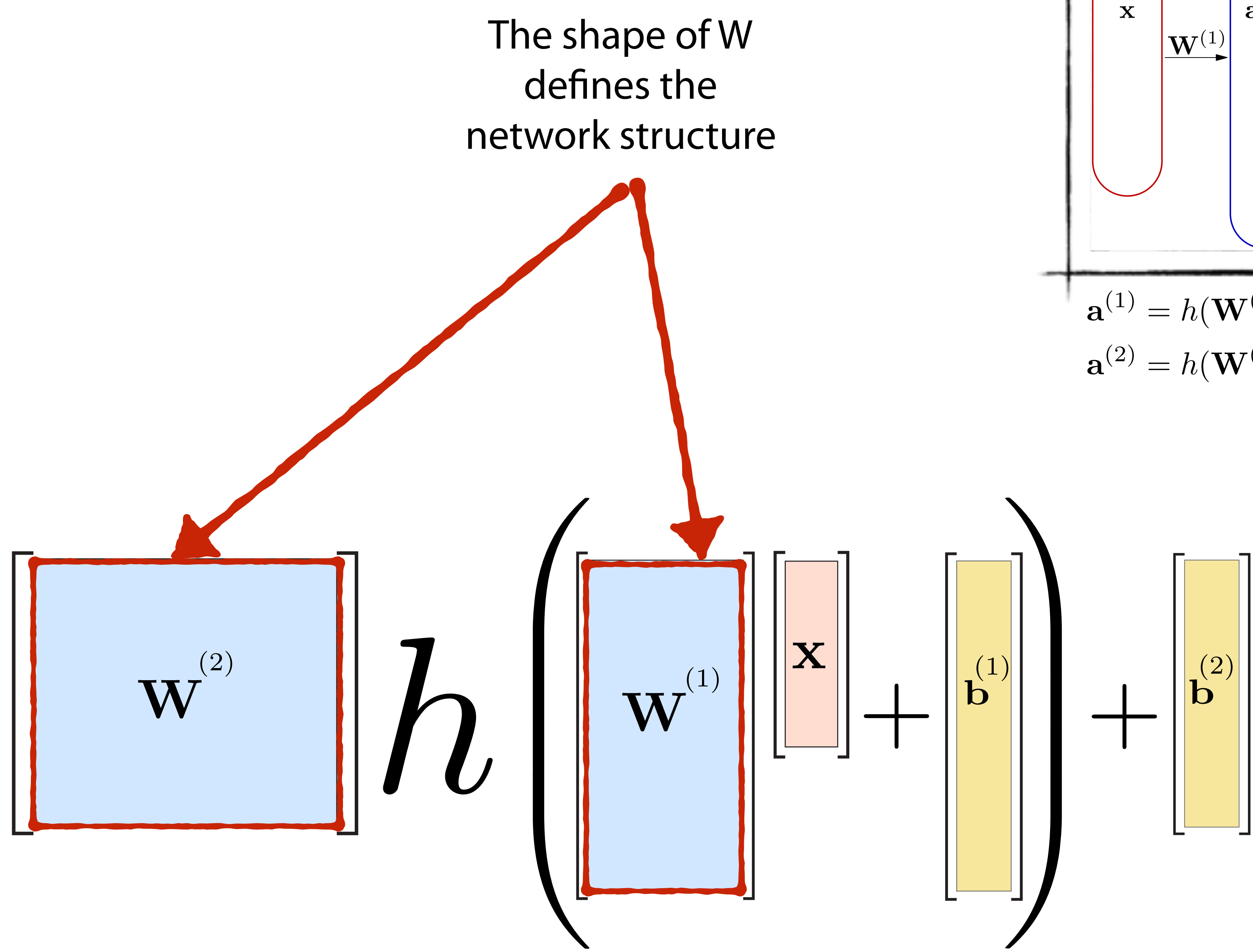
Artificial neural network — matrix notation

Layered NNs are simply a chain of matrix multiplications and activation functions.

Layered network

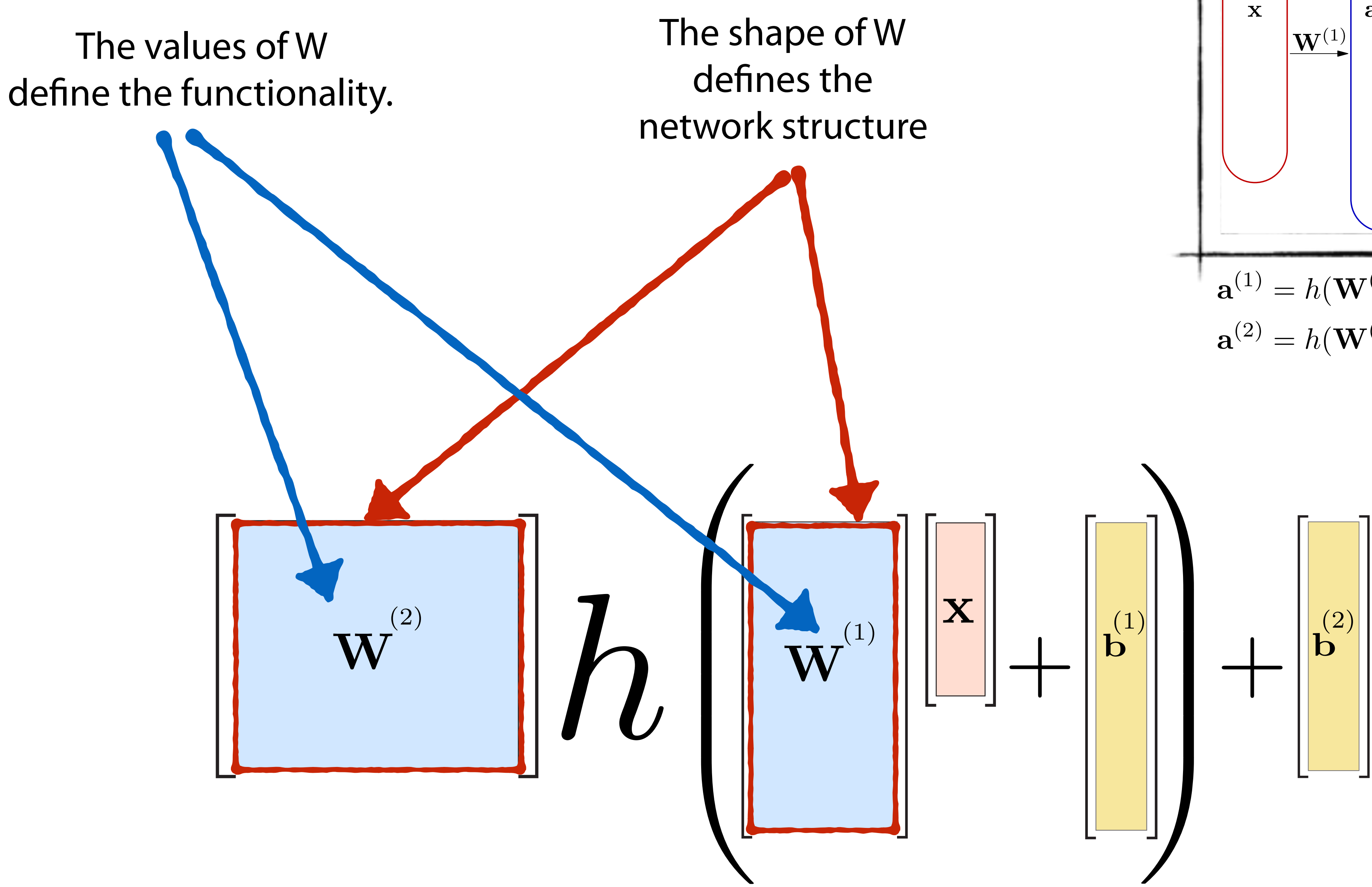


$$\mathbf{a}^{(1)} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{a}^{(2)} = h(\mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)})$$

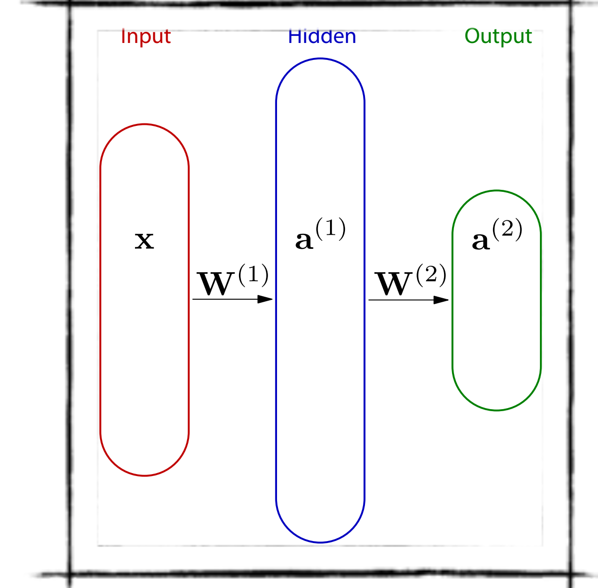


Artificial neural network — matrix notation

Layered NNs are simply a chain of matrix multiplications and activation functions.



Layered network

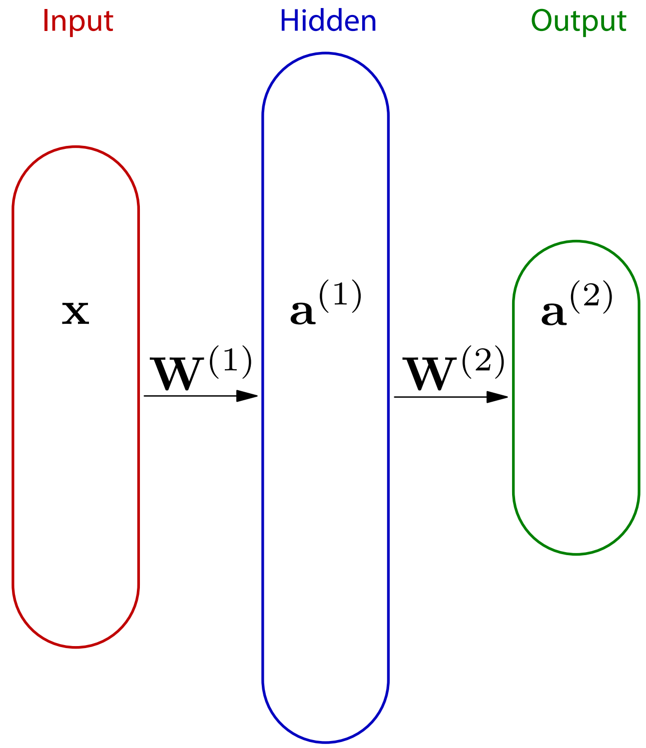


$$\mathbf{a}^{(1)} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{a}^{(2)} = h(\mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)})$$

Deep Learning

Chaining many (more than 1) hidden layers yields a deep neural network.

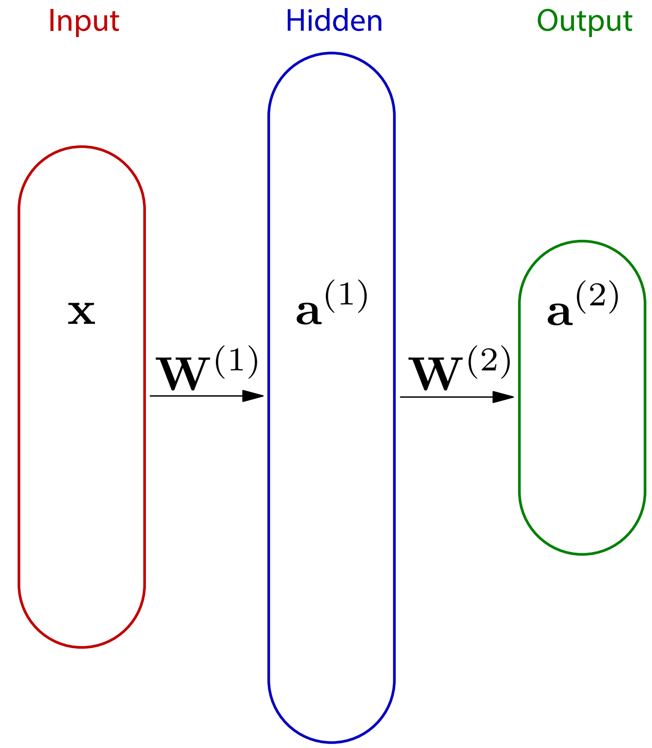
Shallow Learning



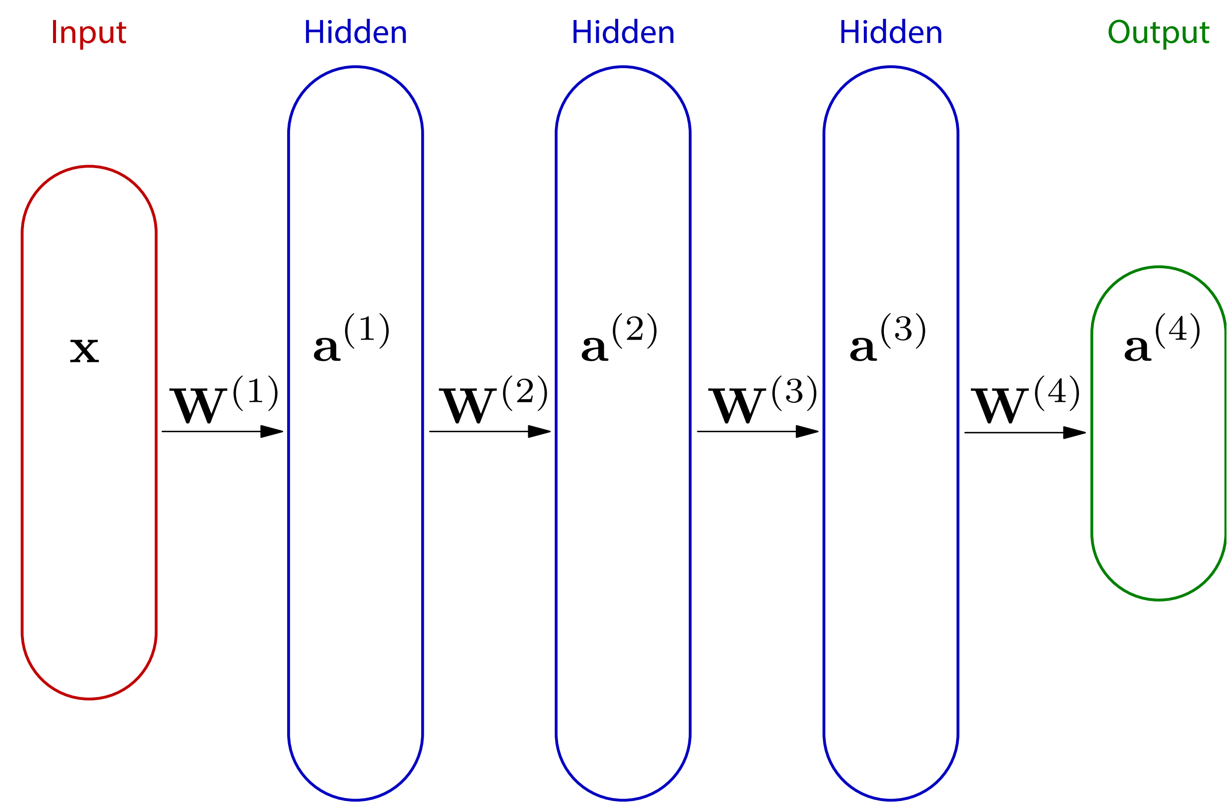
Deep Learning

Chaining many (more than 1) hidden layers yields a deep neural network.

Shallow Learning



Deep Learning



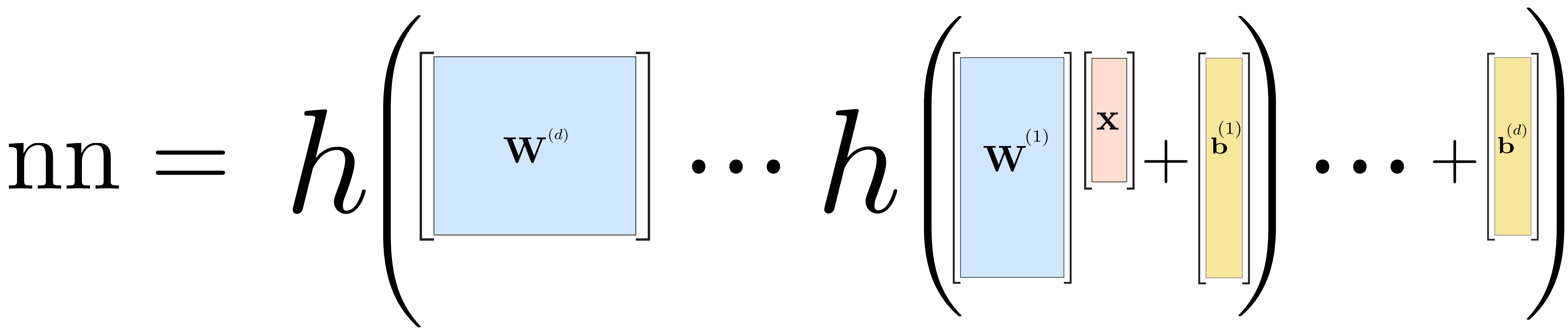
Deep Learning

Chaining many (more than 1) hidden layers yields a deep neural network.

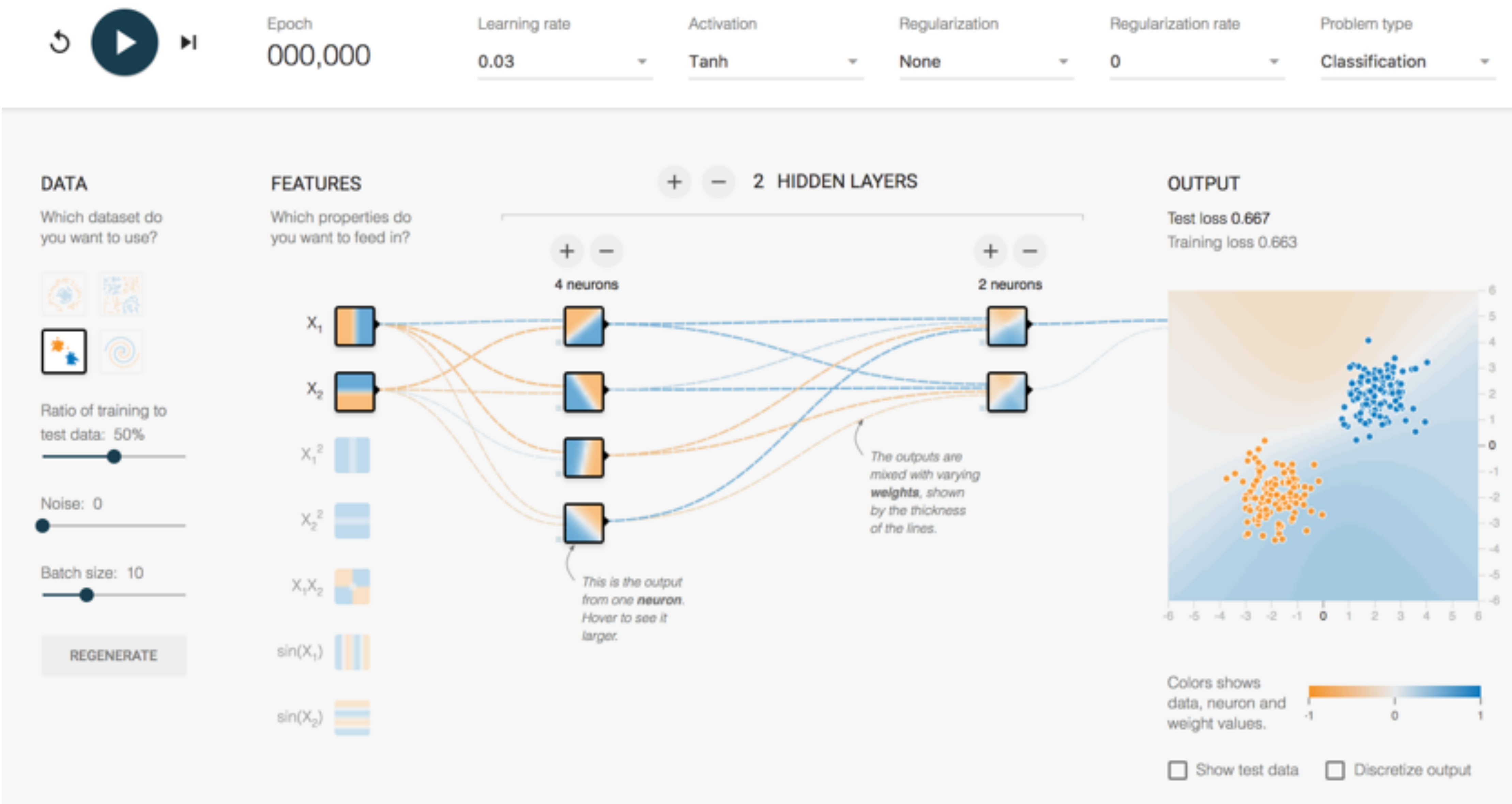
Functional representation

$$\text{nn} = \text{linear}(\cdots h(\text{linear}(\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}))\cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(d)})$$

Matrix representation



Neural network playground (classification)



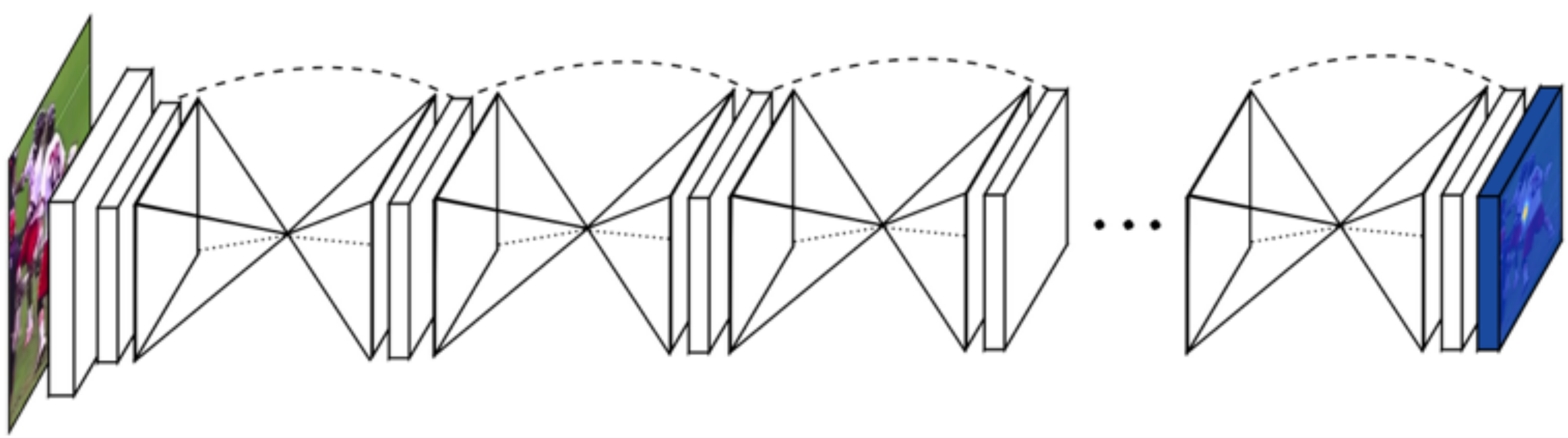
http://playground.tensorflow.org

Deeeeeeep Learning

Very deep networks of hundreds of layers can be formed but require special architectures.

- Residual network, more than 100 layers

- Stacked hourglass network, task-dependent architectures



FaceApp

Very difficult tasks can be modeled. Such as changing the age or gender of a photo.

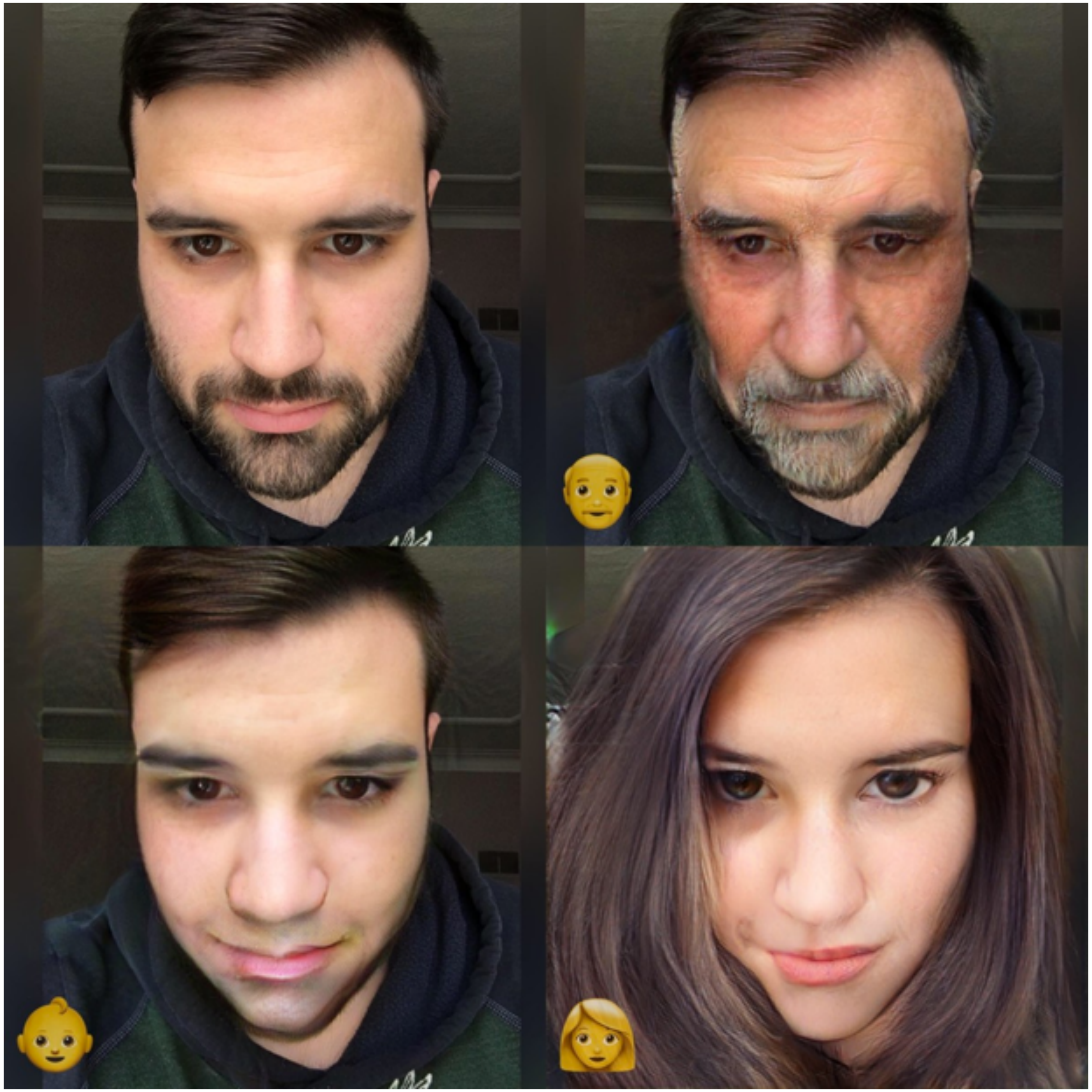


Image to image translation

[Everybody dance now. Chan et al. 2018]

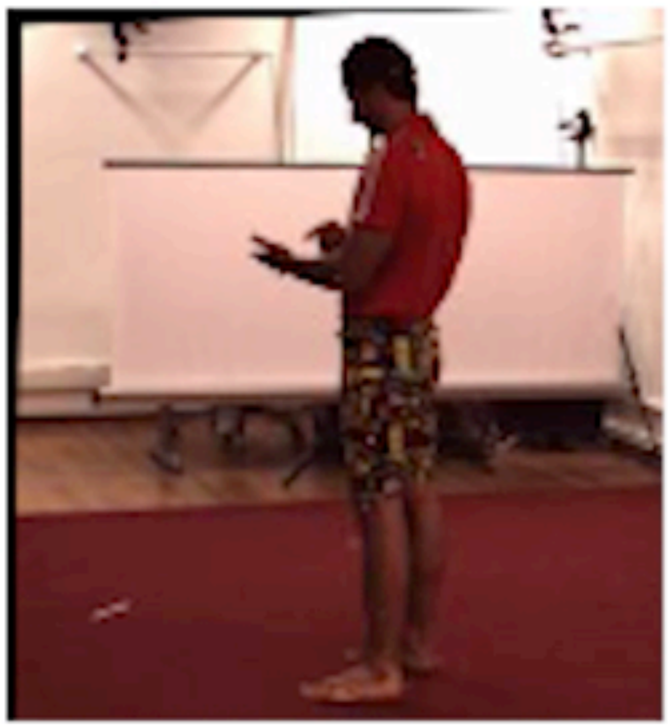
Image to image translation

[Everybody dance now. Chan et al. 2018]

My research on human motion capture

[Rhodin et al. 2018]

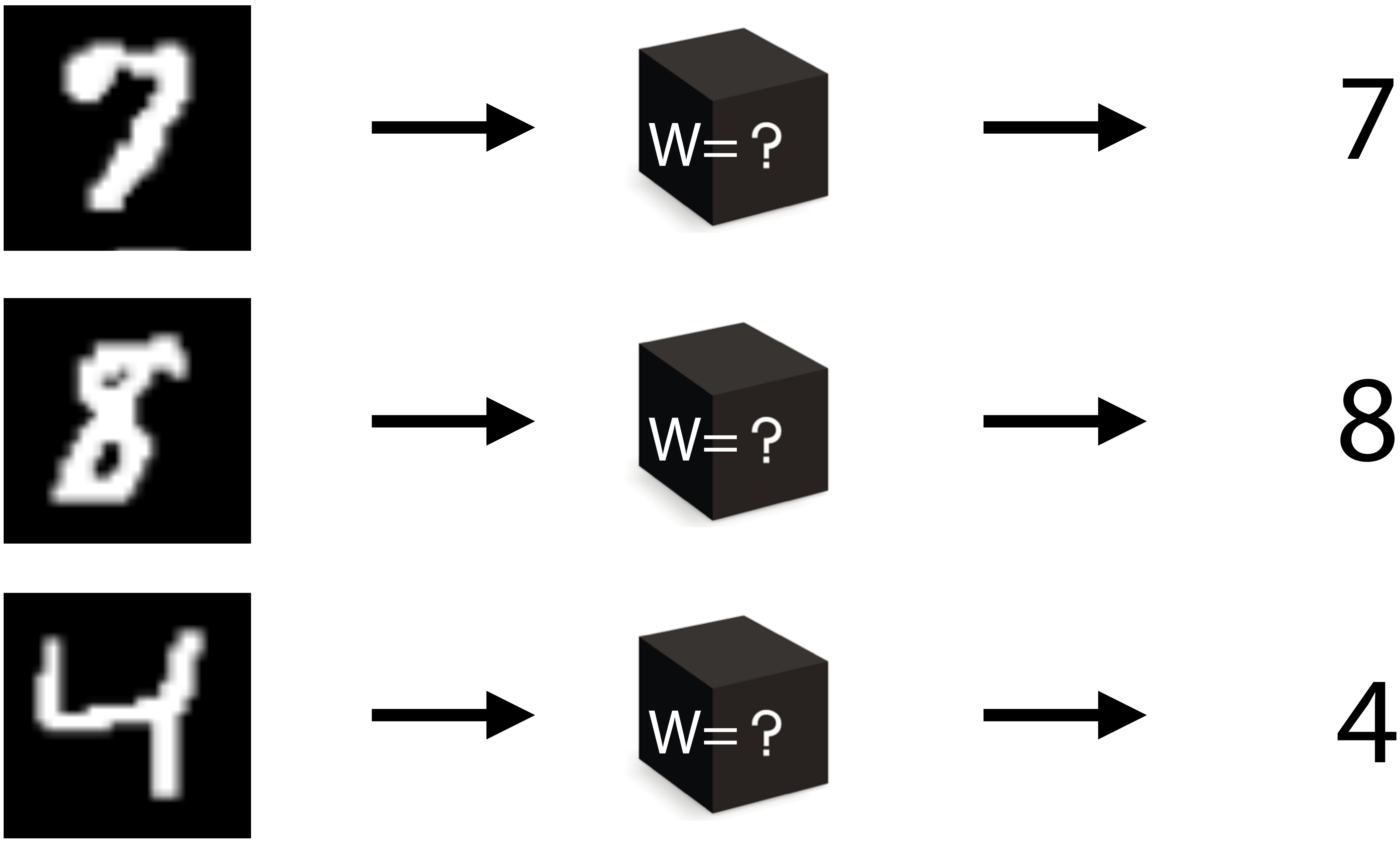
My research on human motion capture



[Rhodin et al. 2018]

Learning from examples

The vast amount of NN parameters can't be defined by hand. It is learned from data.



Big data and deep learning

Training neural networks require huge amounts of data — coined big data.

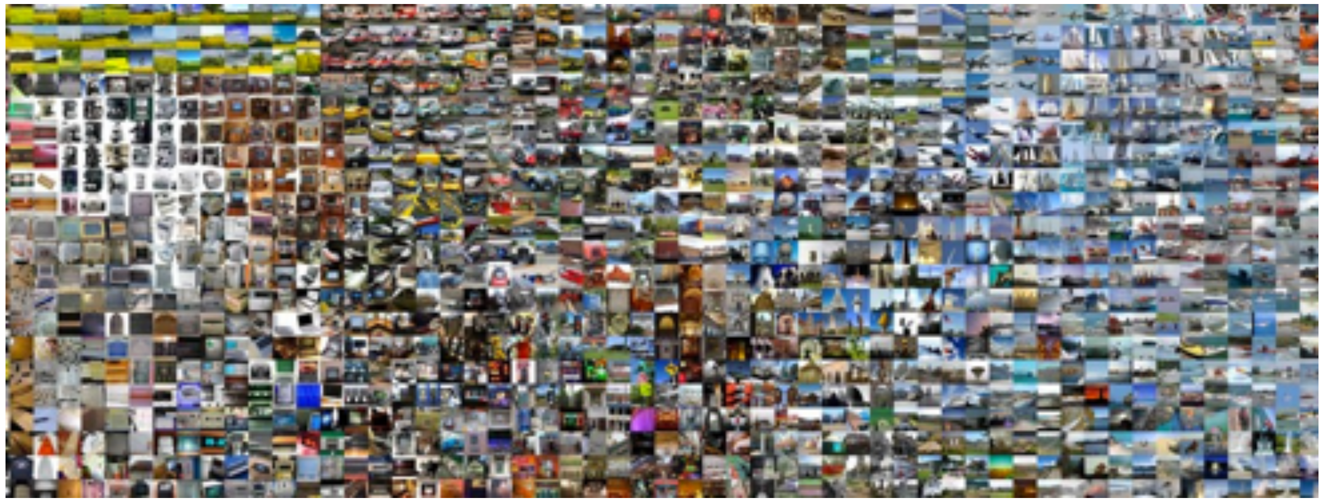


Image net
(14 million image-class pairs)

Human3.6M
(3.6 million image-3D pose pairs)

Big data and deep learning

Training neural networks require huge amounts of data — coined big data.

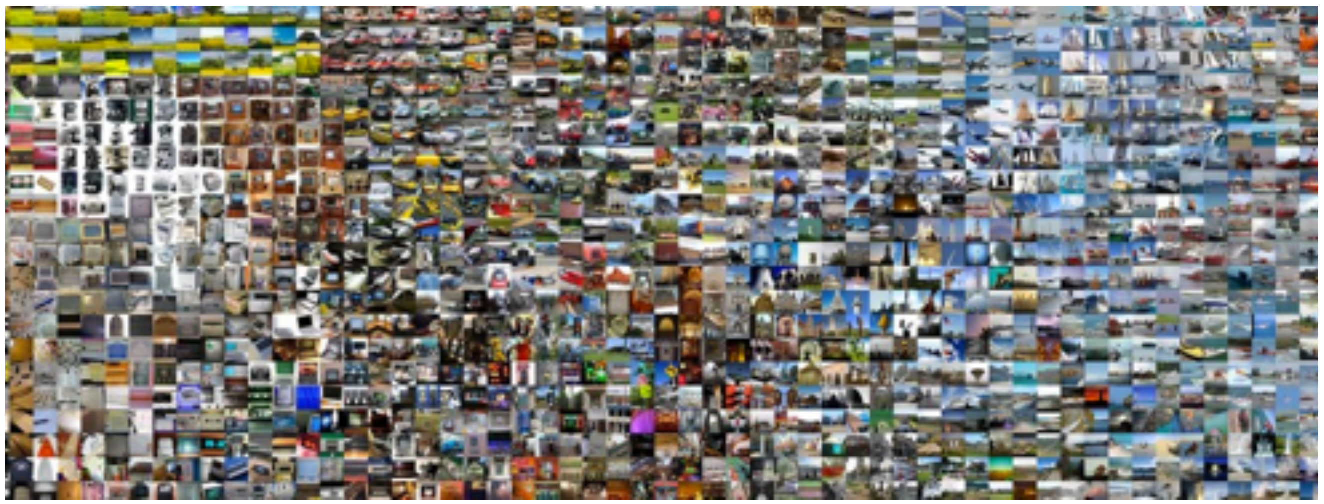
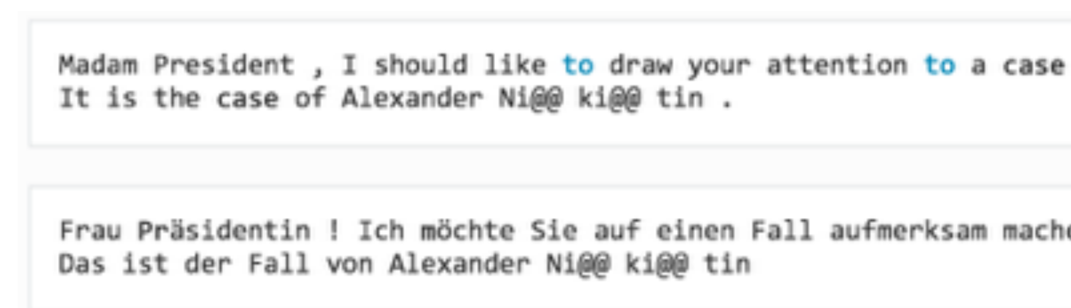


Image net
(14 million image-class pairs)

Human3.6M
(3.6 million image-3D pose pairs)

MPII human pose
(40 thousand image-2D pose pairs)



Europarl translation dataset
(60 million words per language)

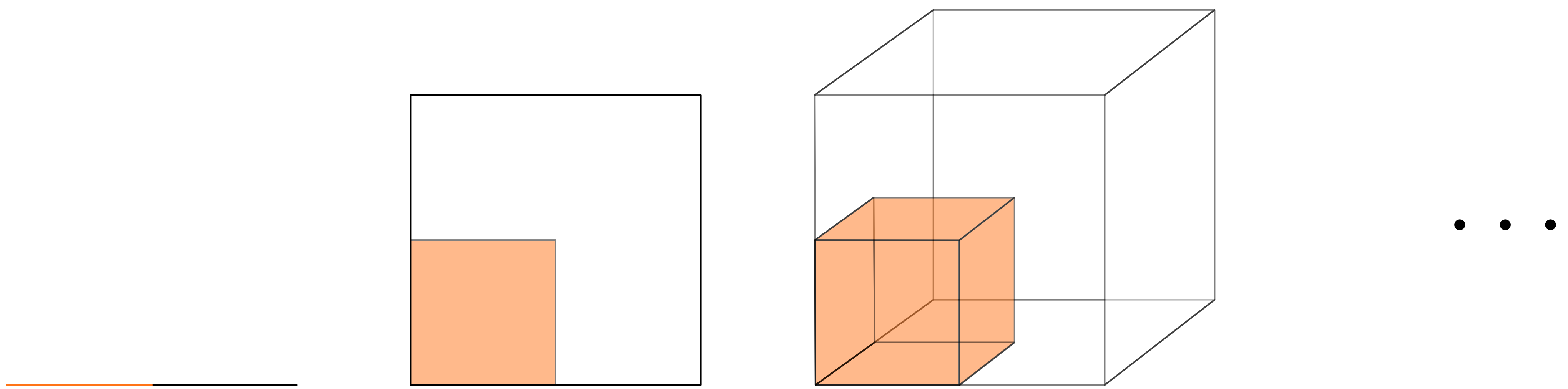
MNIST database of handwritten digits

It contains 70 000 digit examples and is one of the most well-known and used test beds.

Sampling?

In high dimensions, the course of dimensionality prevents regular sampling. Splitting each dimension in half causes exponentially many cells. It is impossible to create large enough datasets that samples all these dimensions.

- Sampling, course of dimensionality

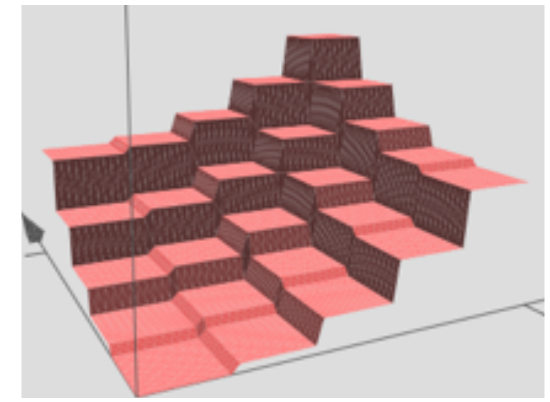


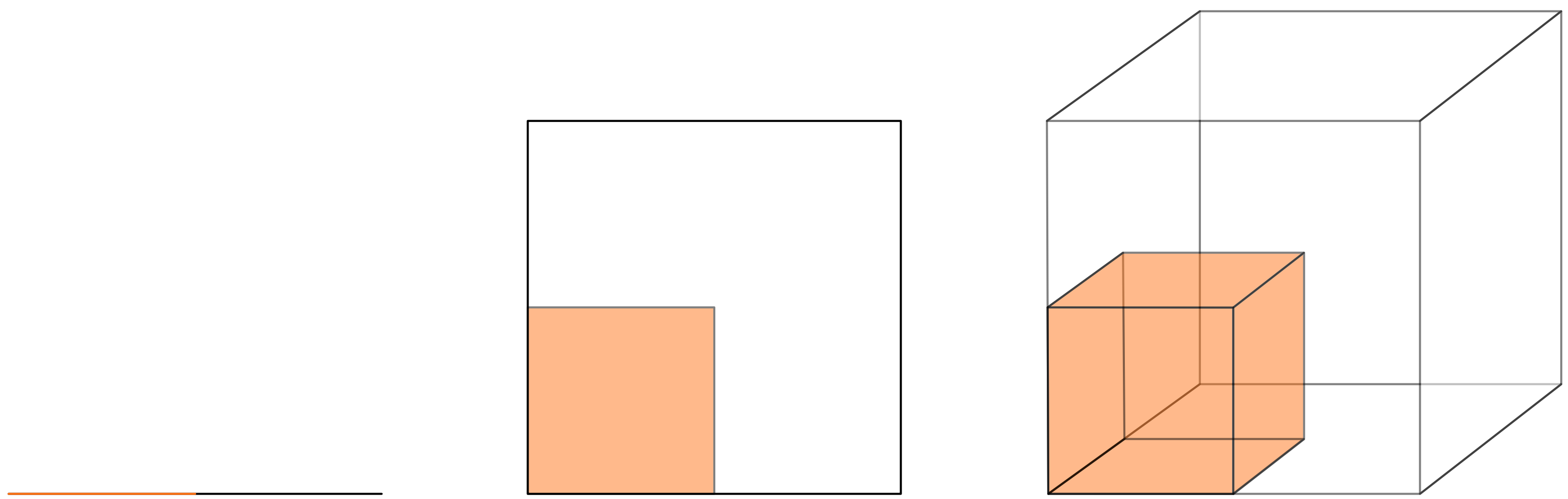
1D	2D	3D	784D
2 cells	4 cells	8 cells	2^{784} cells

Sampling?

In high dimensions, the course of dimensionality prevents regular sampling. Splitting each dimension in half causes exponentially many cells. It is impossible to create large enough datasets that samples all these dimensions.

- Sampling, course of dimensionality





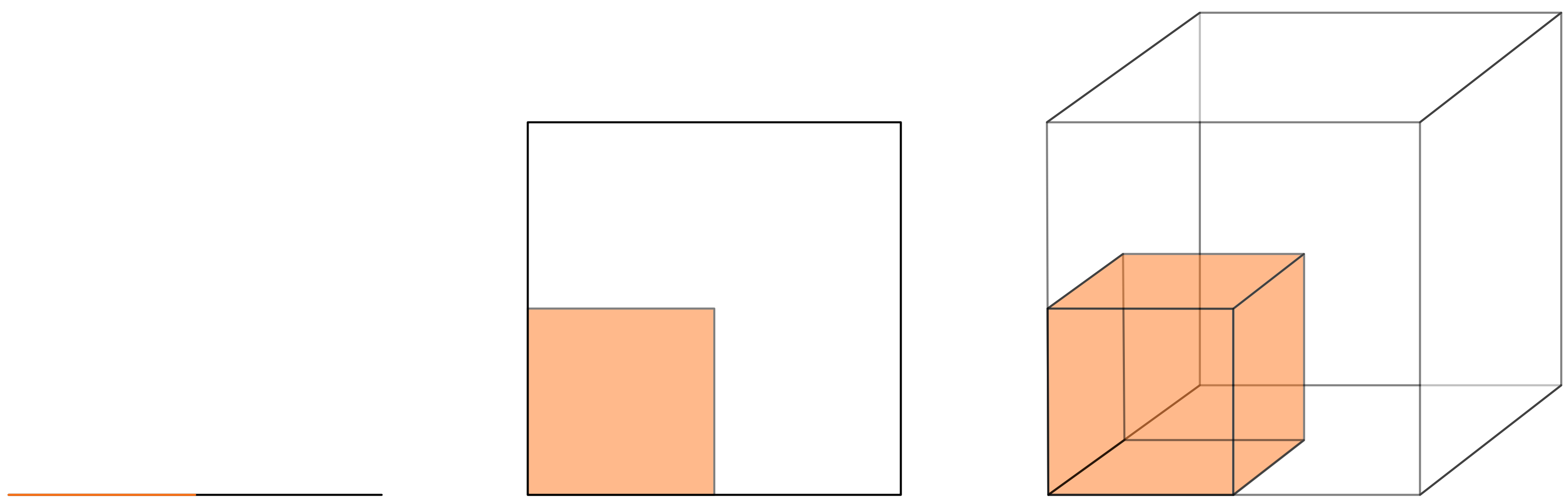
. . .

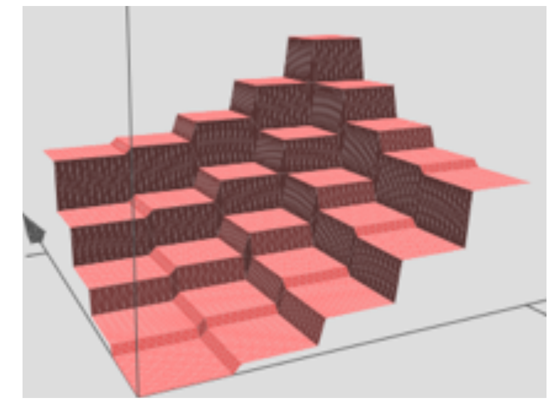
1D	2D	3D	784D
2 cells	4 cells	8 cells	2^{784} cells

Sampling?

In high dimensions, the course of dimensionality prevents regular sampling. Splitting each dimension in half causes exponentially many cells. It is impossible to create large enough datasets that samples all these dimensions.

- Sampling, course of dimensionality





. . .

1D	2D	3D	784D
2 cells	4 cells	8 cells	2^{784} cells

- Non-convex problem
    - Newton's method, BFGS, or gradient decent solver?

Neural network — a parametric function

NN: A parametric function, but with more parameters & higher complexity than seen before.

$\mathrm{nn} =$ Input domain →  → Output domain

	Function	Parameters
Linear regression	$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$	$\mathbf{w} \in \mathbb{R}^n$
Polynomial regression	$f(x) = \mathbf{w} \cdot (1, x, x^2, \ldots, x^{k-1})$	$\mathbf{w} \in \mathbb{R}^k$
Neural Networks	$\mathrm{linear}(\cdots h(\mathrm{linear}(\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)})) \cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(d)})$	$\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \cdots, \mathbf{b}^{(d)}\}$

$\mathbf{W}^{(i)} \in \mathbb{R}^{m_i \times m_{i-1}}$

$\mathbf{b}^{(i)} \in \mathbb{R}^{m_i}$

Neural network — a parametric function

NN: A parametric function, but with more parameters & higher complexity than seen before.

$\text{nn} =$ Input domain →  → Output domain

	Function	Parameters
Linear regression	$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$	$\mathbf{w} \in \mathbb{R}^n$
Polynomial regression	$f(x) = \mathbf{w} \cdot (1, x, x^2, \ldots, x^{k-1})$	$\mathbf{w} \in \mathbb{R}^k$
Neural Networks	$\text{linear}(\cdots h(\text{linear}(\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)})) \cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(d)})$	$\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \cdots, \mathbf{b}^{(d)}\}$

increase

$$\mathbf{W}^{(i)} \in \mathbb{R}^{m_i \times m_{i-1}}$$
$$\mathbf{b}^{(i)} \in \mathbb{R}^{m_i}$$

Neural network — a parametric function

NN: A parametric function, but with more parameters & higher complexity than seen before.

$\text{nn} =$ Input domain

Output domain

	Function	Parameters	
Linear regression	$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$	$\mathbf{w} \in \mathbb{R}^n$	increase
Polynomial regression	$f(x) = \mathbf{w} \cdot (1, x, x^2, \ldots, x^{k-1})$	$\mathbf{w} \in \mathbb{R}^k$	increase
Neural Networks	$\text{linear}(\cdots h(\text{linear}(\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)})) \cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(d)})$	$\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \cdots, \mathbf{b}^{(d)}\}$	

$$\mathbf{W}^{(i)} \in \mathbb{R}^{m_i \times m_{i-1}}$$
$$\mathbf{b}^{(i)} \in \mathbb{R}^{m_i}$$

An optimization problem

Optimize the parameters such that the predicted values are as close as possible to the labels.

$$\arg\min_{\theta} E(D, \theta) \qquad \theta = \{\mathbf{W}^{(i)}, \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \cdots, \mathbf{b}^{(d)}\}$$

An optimization problem

Optimize the parameters such that the predicted values are as close as possible to the labels.

$$\arg\min_{\theta} E(D, \theta) \qquad \theta = \{\mathbf{W}^{(i)}, \mathbf{W}^{(2)}, \cdots, \mathbf{W}^{(d)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \cdots, \mathbf{b}^{(d)}\}$$

Akin to optimization problems in previous lectures

Linear least squares

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$$

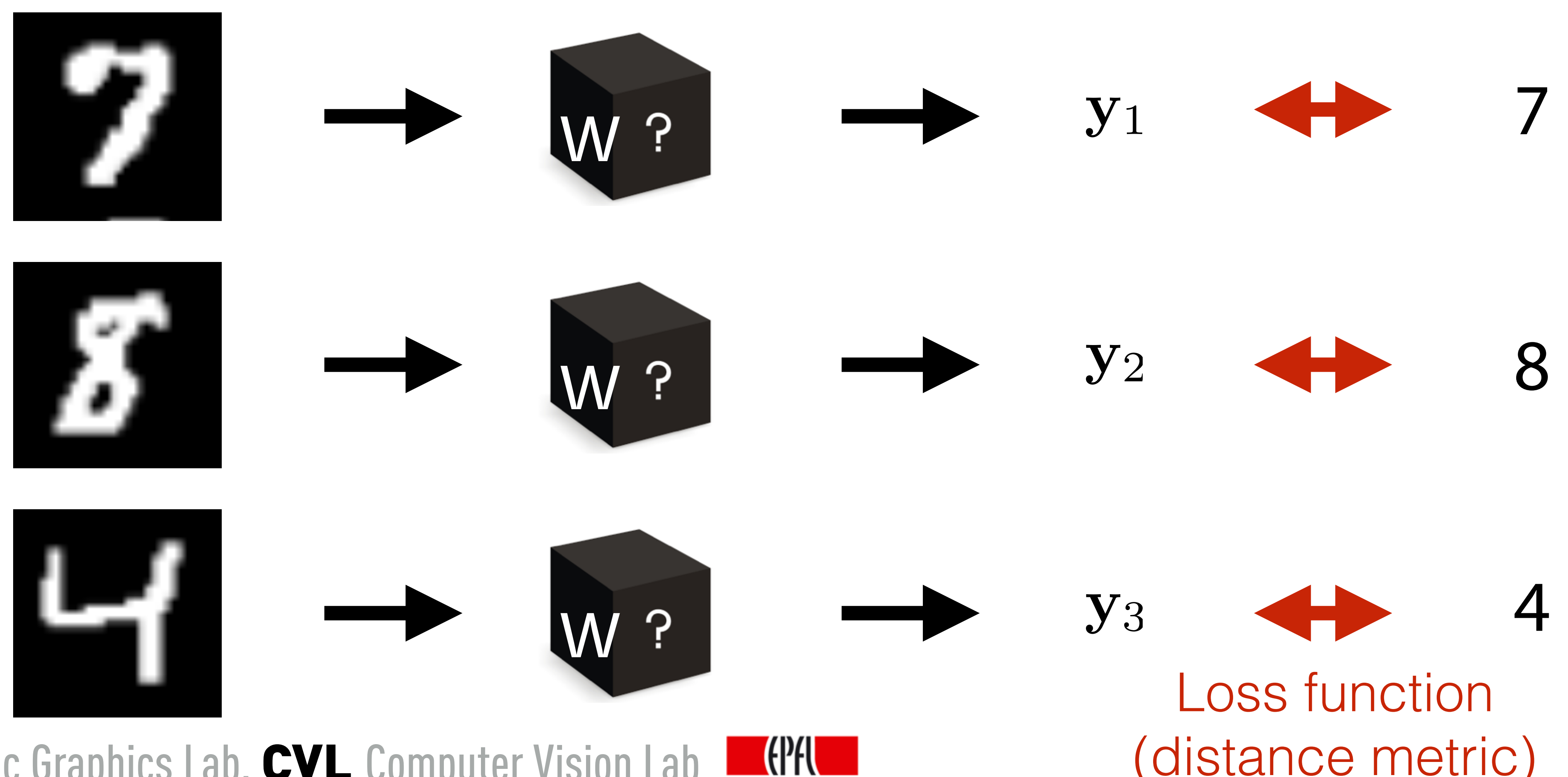
Non-linear least squares

$$\min_{\mathbf{x}} \|\mathbf{f}(\mathbf{x}) - \mathbf{b}\|^2$$

An optimization problem

Optimize the parameters such that the predicted values are as close as possible to the labels.

$$\arg\min_{\theta} E(D, \theta) \qquad \theta = \{\mathbf{w}^{(i)}, \mathbf{w}^{(2)}, \cdots$$



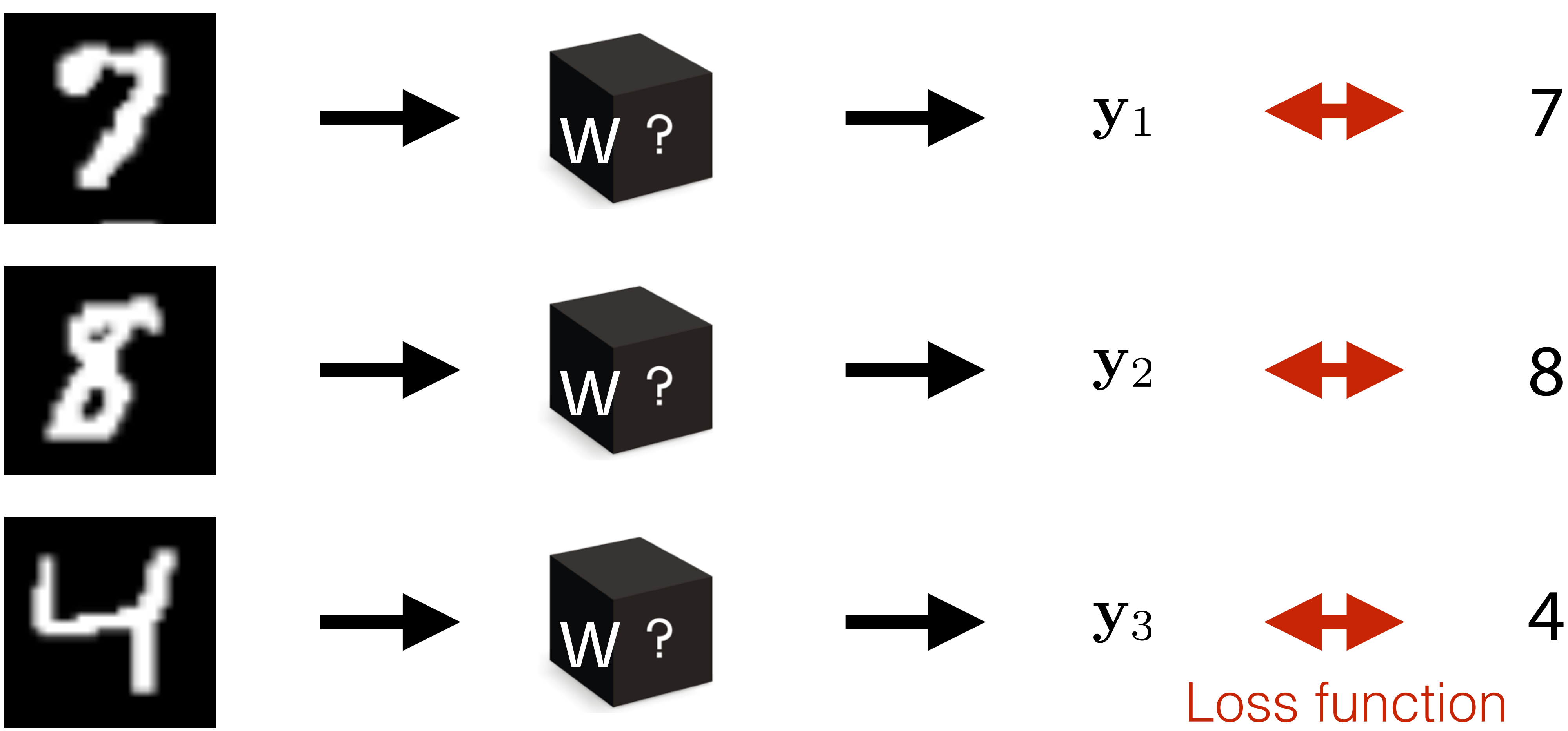
An optimization problem

Optimize the parameters such that the predicted values are as close as possible to the labels.

Parameter

$$\arg\min_{\theta} E(D, \theta)$$

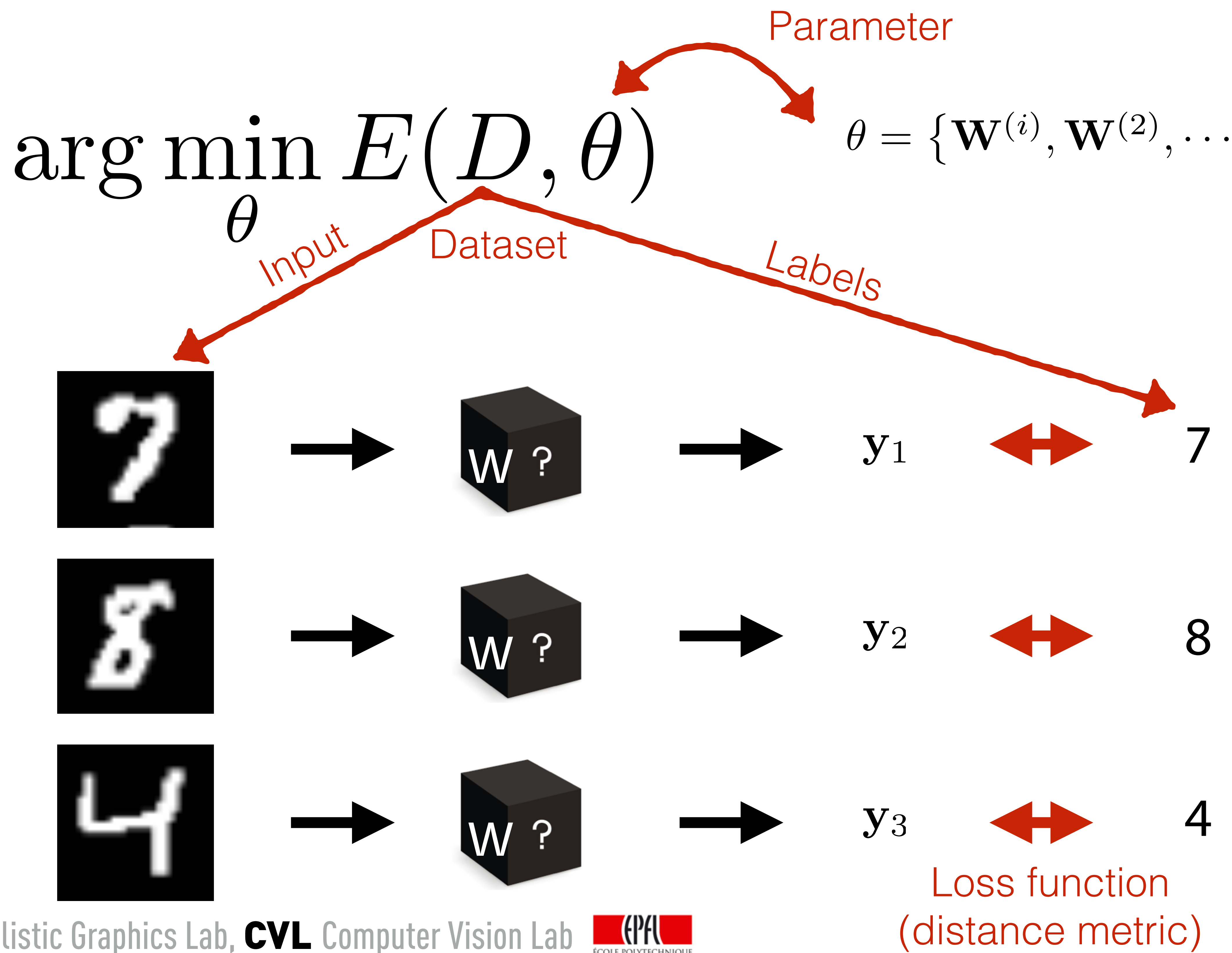
$$\theta = \{\mathbf{w}^{(i)}, \mathbf{w}^{(2)}, \cdots$$



Loss function
(distance metric)

An optimization problem

Optimize the parameters such that the predicted values are as close as possible to the labels.



Objective and loss function

The difference between prediction and label — a compromise of tractability and realism.

$$\arg\min_{\theta} E(D, \theta)$$

Objective and loss function

The difference between prediction and label — a compromise of tractability and realism.

General form

$$\arg\min_{\theta} E(D, \theta)$$

Objective and loss function

The difference between prediction and label — a compromise of tractability and realism.

General form

$$\arg\min_{\theta} E(D, \theta)$$

Separable sum

$$E(D, \theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} l(\mathrm{nn}(\mathbf{x}^{(i)}, \theta), y^{(i)})$$

Objective and loss function

The difference between prediction and label — a compromise of tractability and realism.

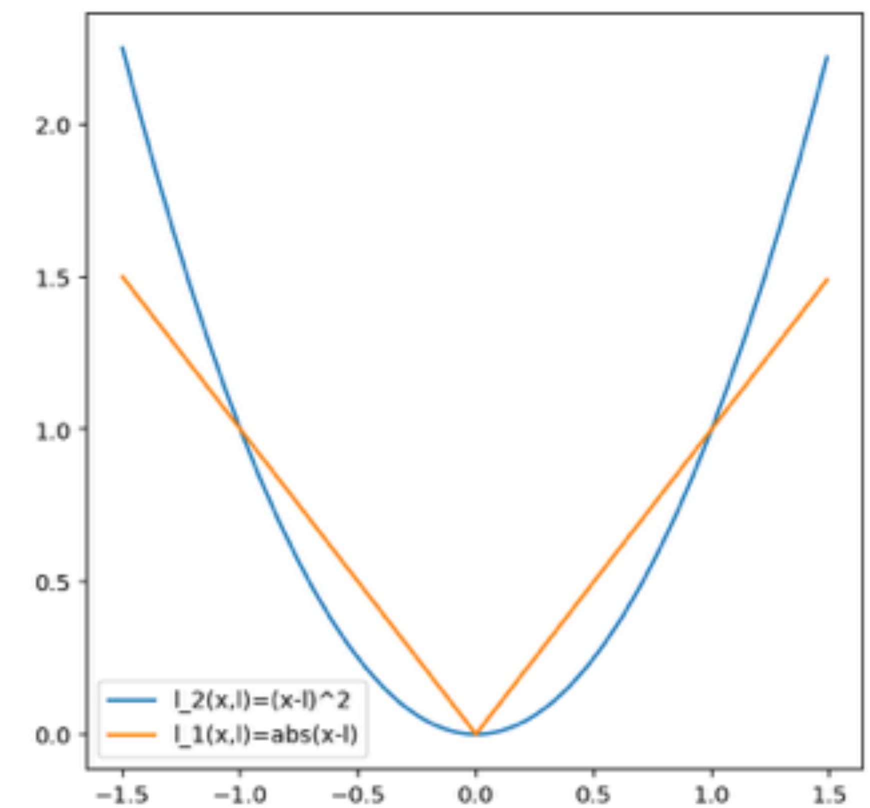
General form

$$\arg\min_{\theta} E(D, \theta)$$

Separable sum

$$E(D, \theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} l(\mathrm{nn}(\mathbf{x}^{(i)}, \theta), y^{(i)})$$

Loss functions



Quadratic loss

$$l_2(y, l) = (y - l)^2$$

Absolute loss

$$l_1(y, l) = |y - l|$$

Objective and loss function

The difference between prediction and label — a compromise of tractability and realism.

General form

$$\arg\min_{\theta} E(D, \theta)$$

Separable sum

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} l(\mathrm{nn}(\mathbf{x}^{(i)}, \theta), y^{(i)})$$

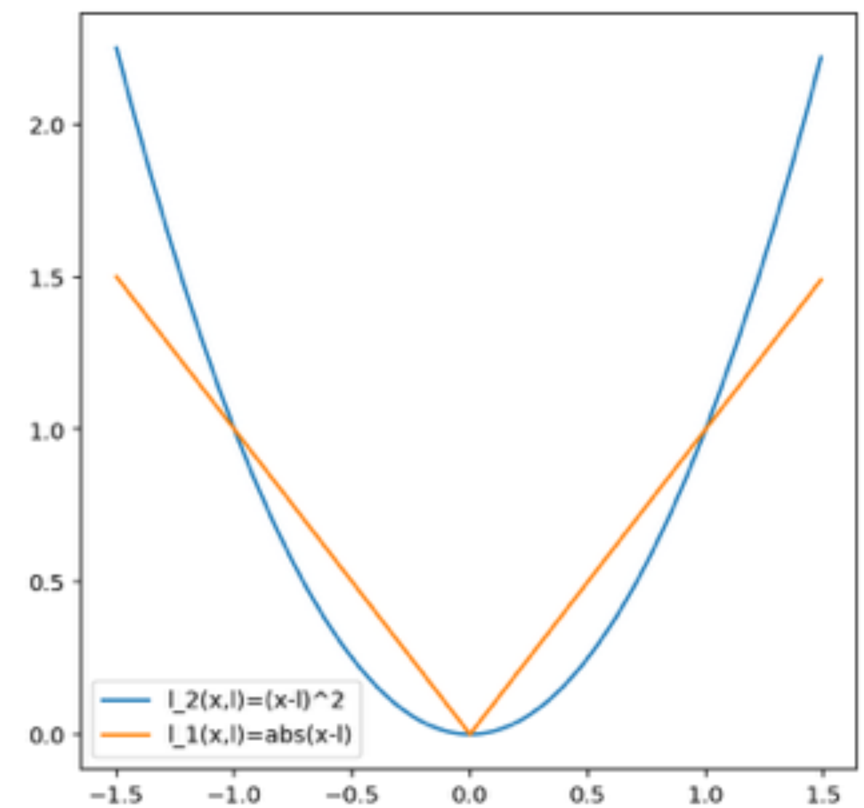
MNIST digit example

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} (\mathrm{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})^2$$

$=(\mathrm{nn}($  $, \theta) - 7)^2$

$+(\mathrm{nn}($  $, \theta) - 8)^2 \ldots$

Loss functions



Quadratic loss

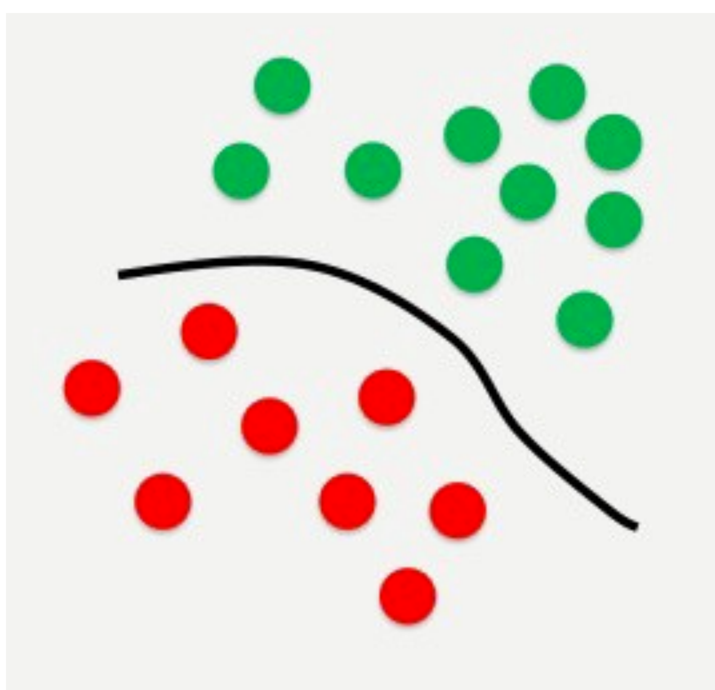
$$l_2(y, l) = (y - l)^2$$

Absolute loss

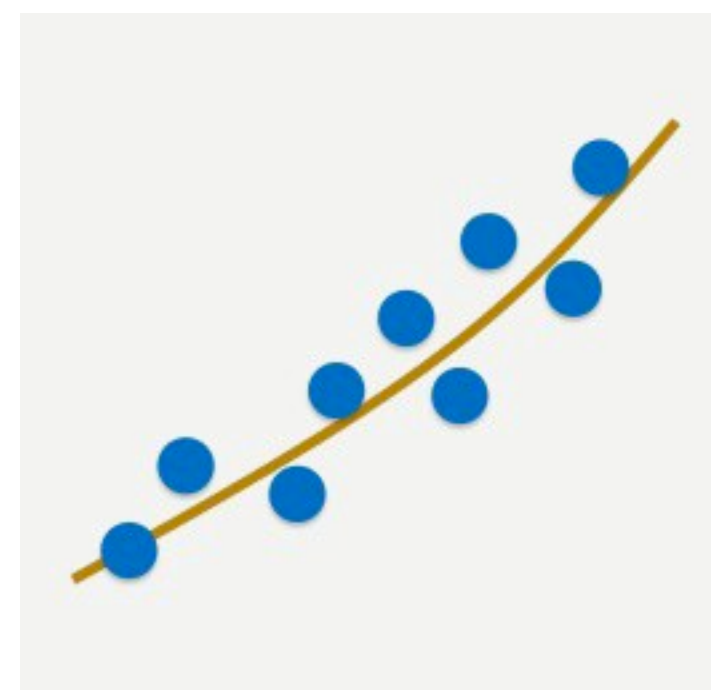
$$l_1(y, l) = |y - l|$$

Classification vs. regression

Classification

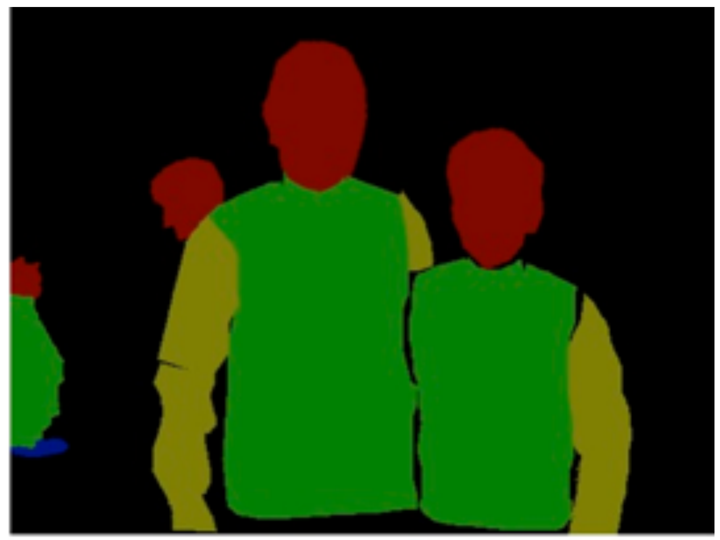


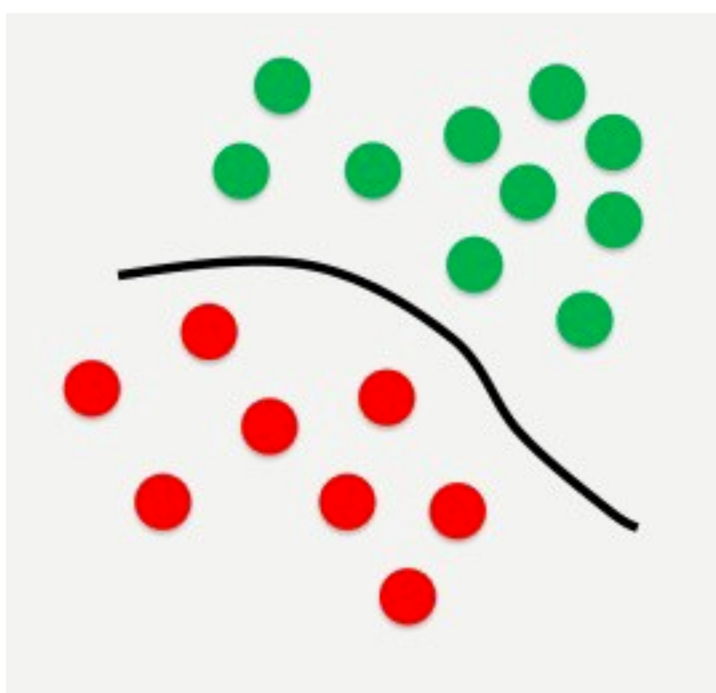
Regression



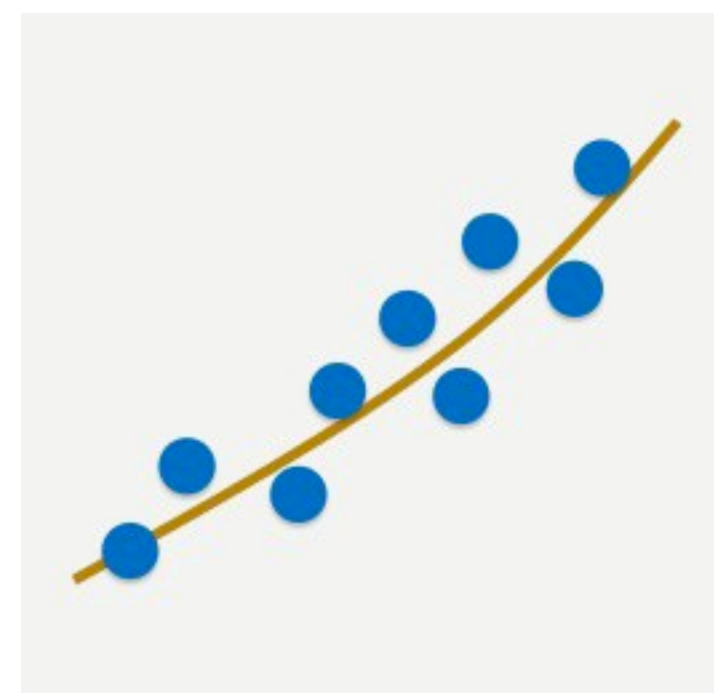
Classification vs. regression

Classification





Regression

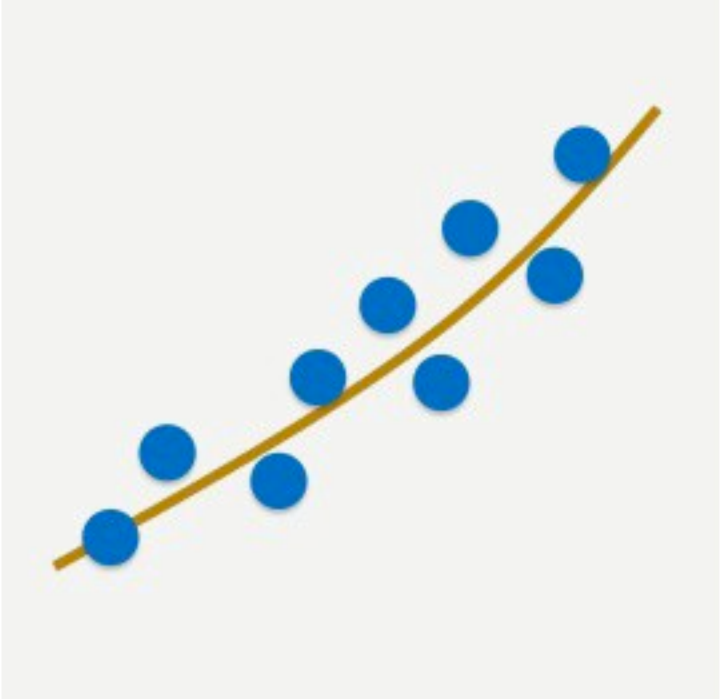


Classification vs. regression

- Regression
  - works for continuous values

$$\mathrm{nn}(\mathbf{x}) \rightarrow y \in \mathbb{R}$$

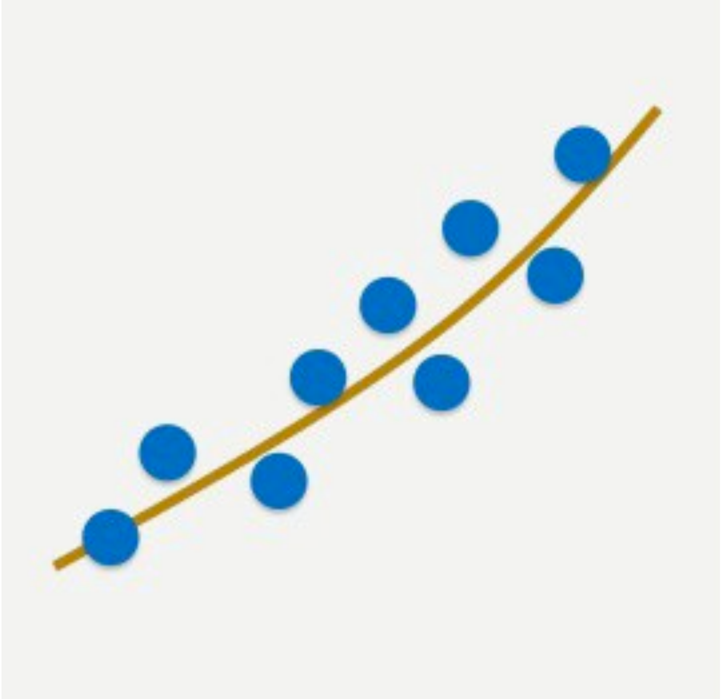
$$l_2(y, l) = (y - l)^2$$



Classification vs. regression

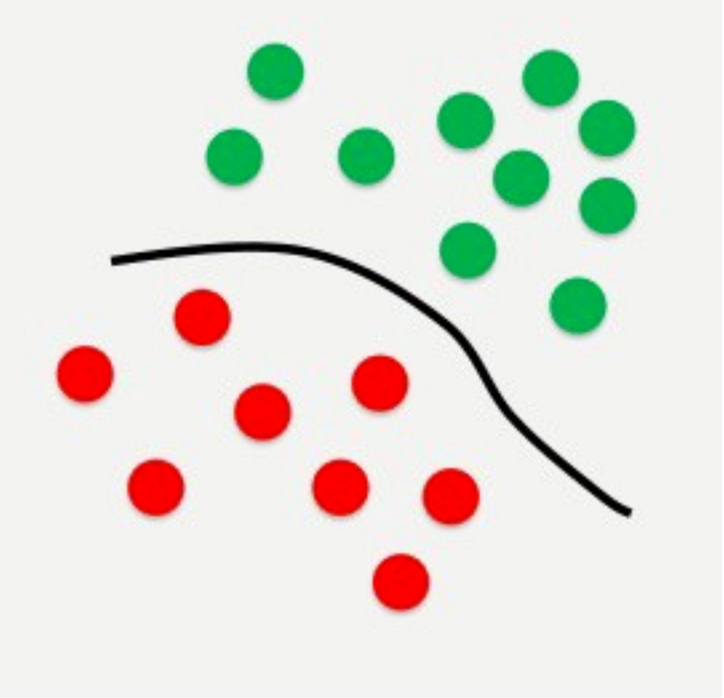
- Regression
  - works for continuous values

$$\text{nn}(\mathbf{x}) \rightarrow y \in \mathbb{R}$$
$$l_2(y, l) = (y - l)^2$$



- Classification
  - discrete classes
  - probabilistic interpretation (probability of class)

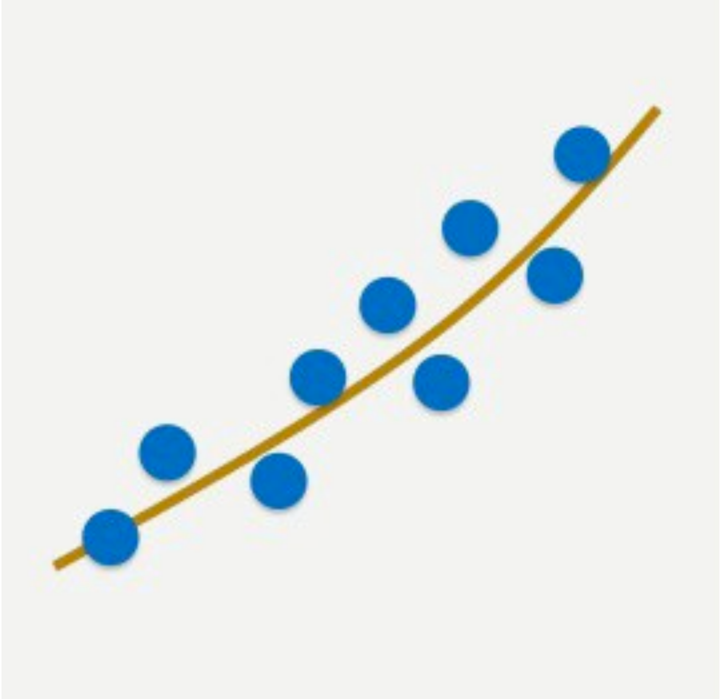
$$\text{nn}(\mathbf{x}) \rightarrow \mathbf{y} \in [0, 1]$$
$$l_2(\mathbf{y}, \mathbf{l}) = \|\mathbf{y} - \mathbf{l}\|^2$$



Classification vs. regression

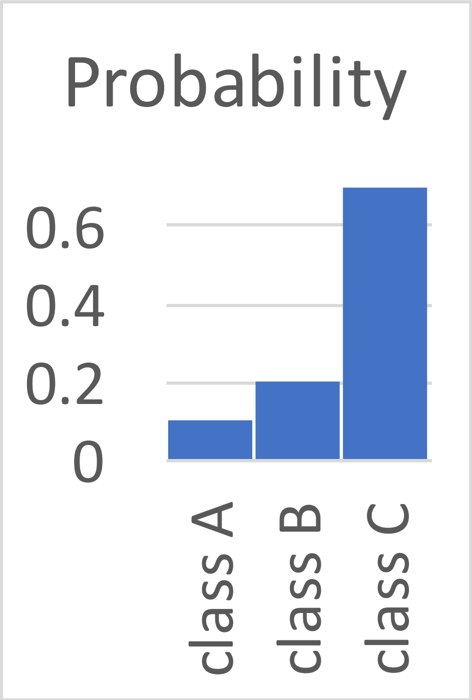
- Regression
  - works for continuous values

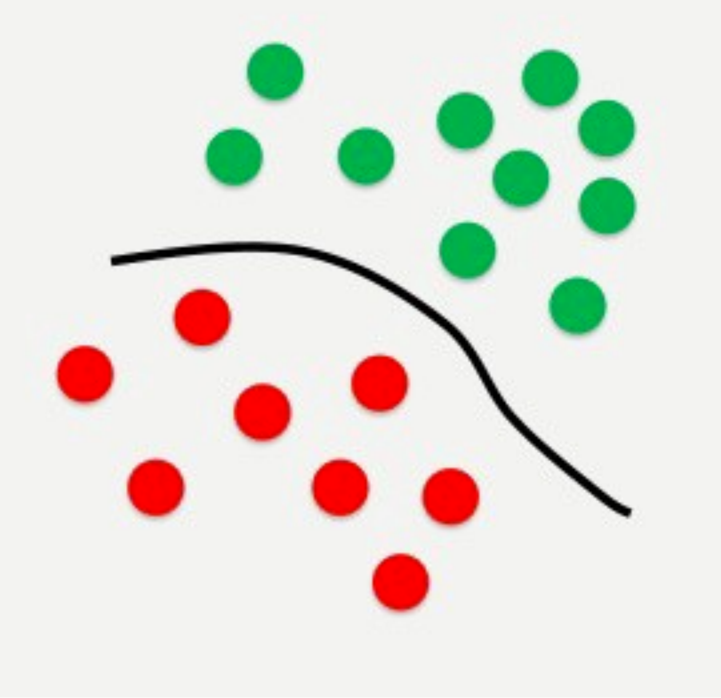
$$\mathrm{nn}(\mathbf{x}) \to y \in \mathbb{R}$$
$$l_2(y, l) = (y - l)^2$$



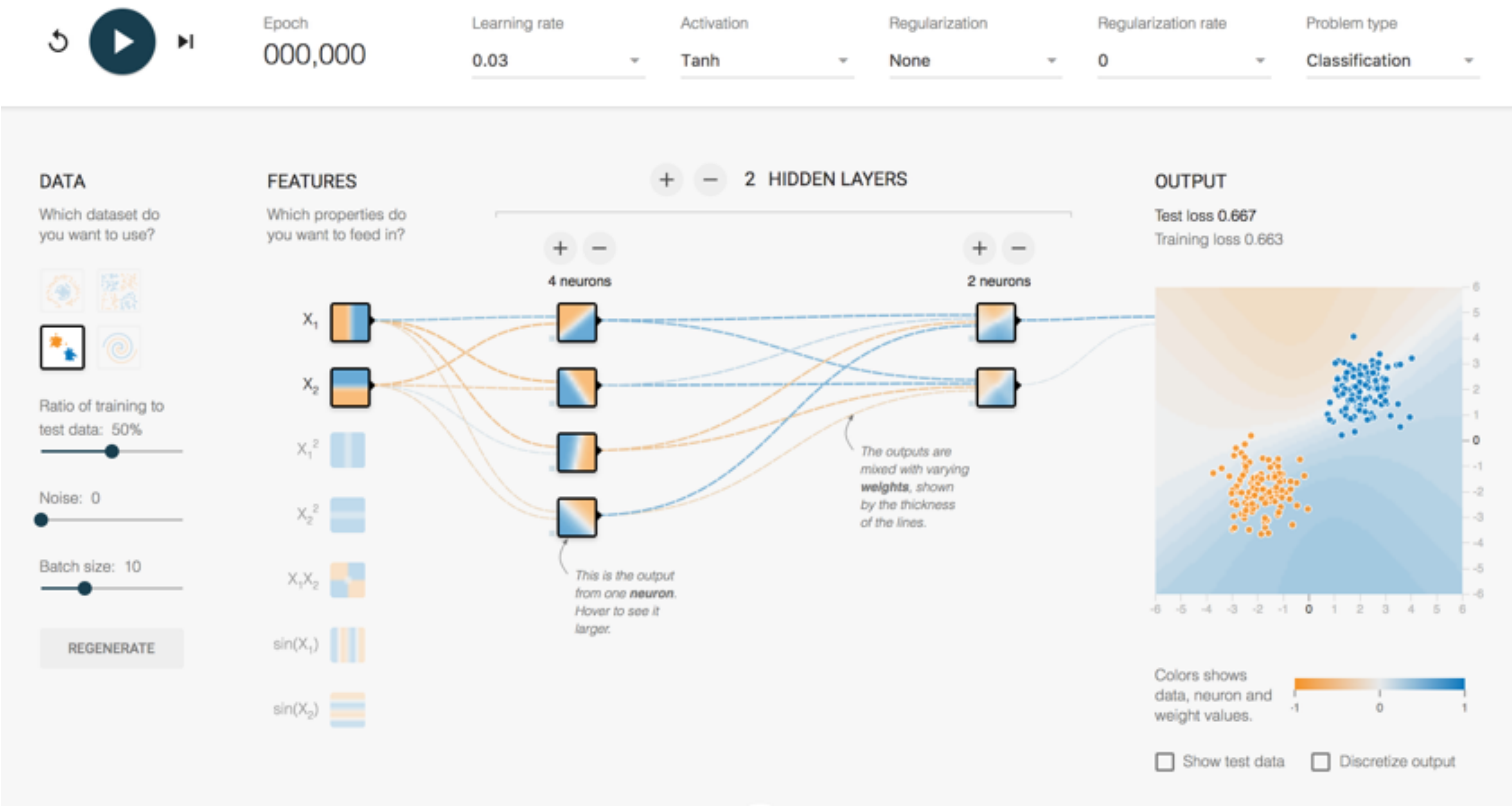
- Classification
  - discrete classes
  - probabilistic interpretation (probability of class)

$$\mathrm{nn}(\mathbf{x}) \to \mathbf{y} \in [0, 1]$$
$$l_2(\mathbf{y}, \mathbf{l}) = \|\mathbf{y} - \mathbf{l}\|^2$$





Neural network playground (regression)



Training a NN

.. requires 1) a representation, 2) dataset, 3) objective function, 4) NN model and 5) solver.

1) Representation (i/o domain)

$\longrightarrow$ 8

Input image $\in \mathbb{R}^{28\times 28}$

Label $\in \mathbb{R}$

Training a NN

.. requires 1) a representation, 2) dataset, 3) objective function, 4) NN model and 5) solver.

1) Representation (i/o domain)

$\longrightarrow$ 8

Input image $\in \mathbb{R}^{28\times 28}$

Label $\in \mathbb{R}$

2) Dataset D

[MNIST]

Training a NN

.. requires 1) a representation, 2) dataset, 3) objective function, 4) NN model and 5) solver.

1) Representation (i/o domain)

→ 8

Input image $\in \mathbb{R}^{28\times 28}$

Label $\in \mathbb{R}$

2) Dataset D

[MNIST]

3) Objective $\arg\min_{W} E(D,\theta)$

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)},y^{(i)})\in D} (\mathrm{nn}(\mathbf{x}^{(i)},\theta) - y^{(i)})^2$$

Training a NN

.. requires 1) a representation, 2) dataset, 3) objective function, 4) NN model and 5) solver.

1) Representation (i/o domain)

$\longrightarrow$ 8

Input image $\in \mathbb{R}^{28\times 28}$

Label $\in \mathbb{R}$

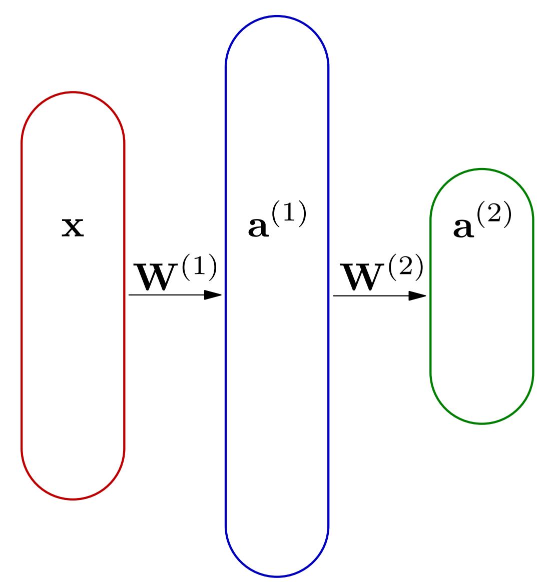
2) Dataset D

[MNIST]

3) Objective $\arg\min_W E(D,\theta)$

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)},y^{(i)})\in D} (\mathrm{nn}(\mathbf{x}^{(i)},\theta) - y^{(i)})^2$$

4) Model $\mathrm{nn}(\mathbf{x},\theta)$



Training a NN

.. requires 1) a representation, 2) dataset, 3) objective function, 4) NN model and 5) solver.

1) Representation (i/o domain)

→ 8

Input image $\in \mathbb{R}^{28\times 28}$

Label $\in \mathbb{R}$

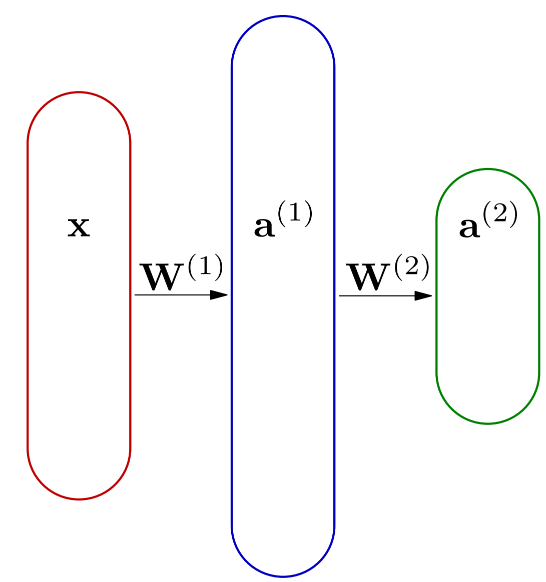
2) Dataset D

[MNIST]

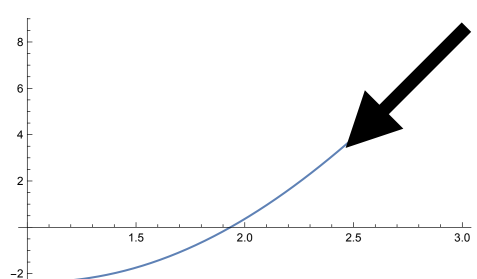
3) Objective $\arg\min_{W} E(D,\theta)$

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)})\in D} (\mathrm{nn}(\mathbf{x}^{(i)},\theta) - y^{(i)})^2$$

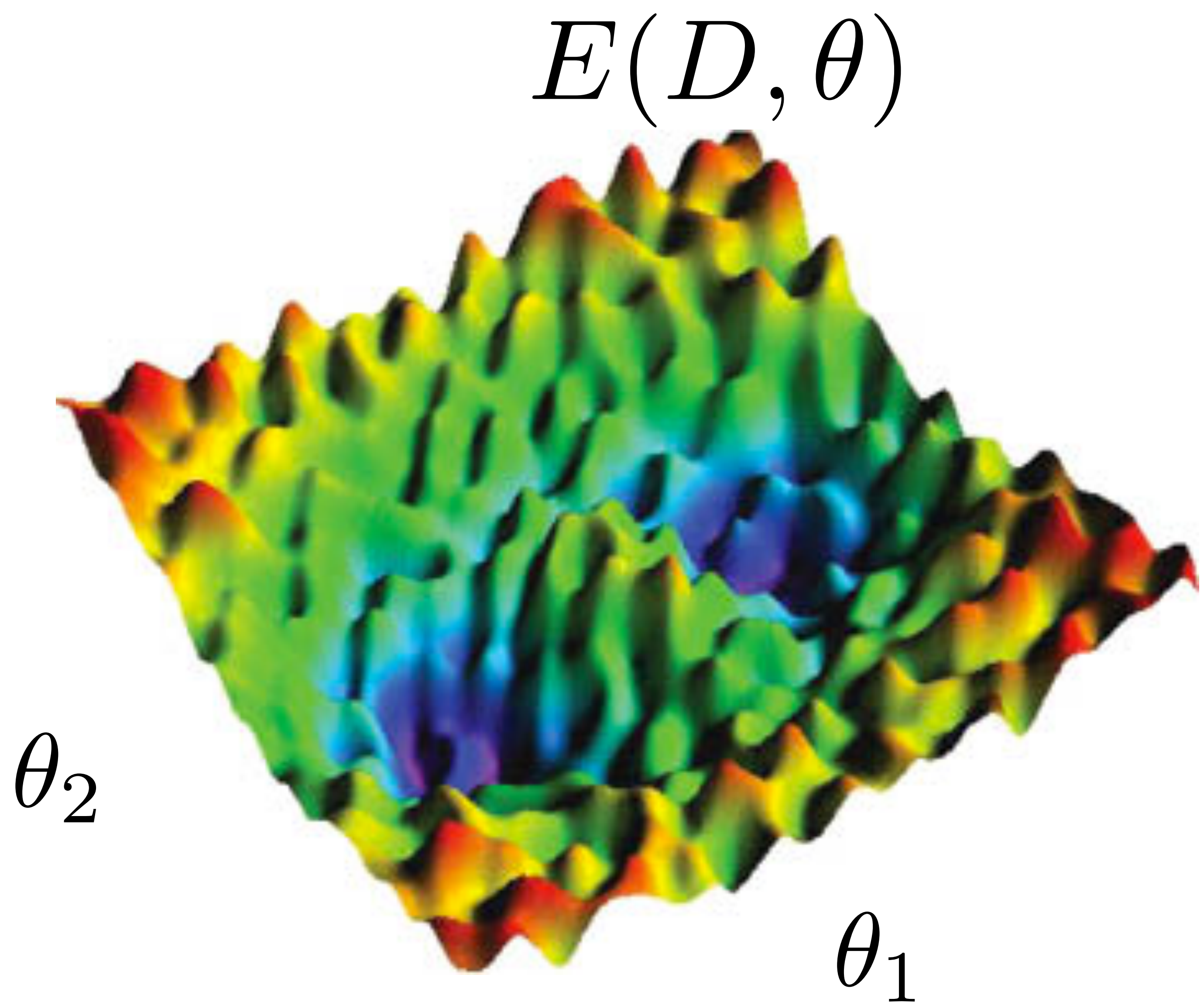
4) Model $\mathrm{nn}(\mathbf{x},\theta)$



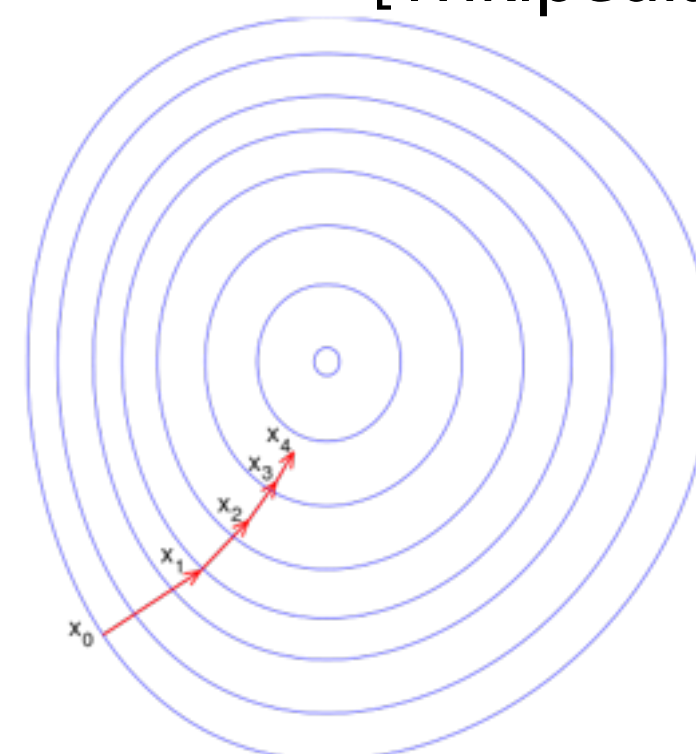
5) Solver



Iterative solver



[Wikipedia]



Stochastic gradient descent

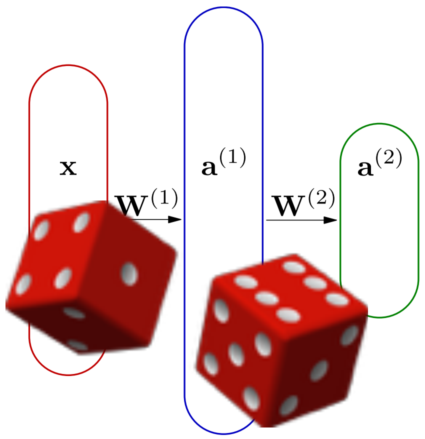
Iteratively optimizing over subsets of the dataset is efficient and works surprisingly well.
At each iteration a new subset is chosen to decent closer to the minimum of the full energy.

Stochastic gradient descent

Iteratively optimizing over subsets of the dataset is efficient and works surprisingly well.
At each iteration a new subset is chosen to decent closer to the minimum of the full energy.

- Start with a random initialization

$\mathbf{W}_{i,j} \sim U\left(-c, c\right)$, with $U(-c, c)$ the uniform distribution over $[-c, c]$.

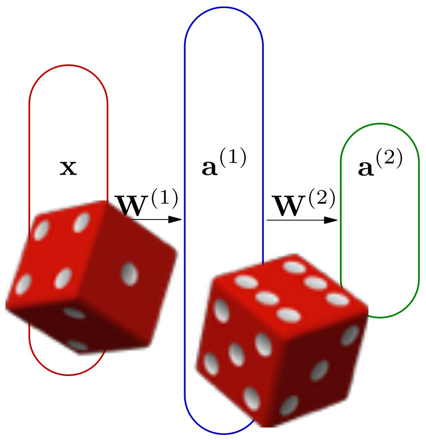


Stochastic gradient descent

Iteratively optimizing over subsets of the dataset is efficient and works surprisingly well.
At each iteration a new subset is chosen to decent closer to the minimum of the full energy.

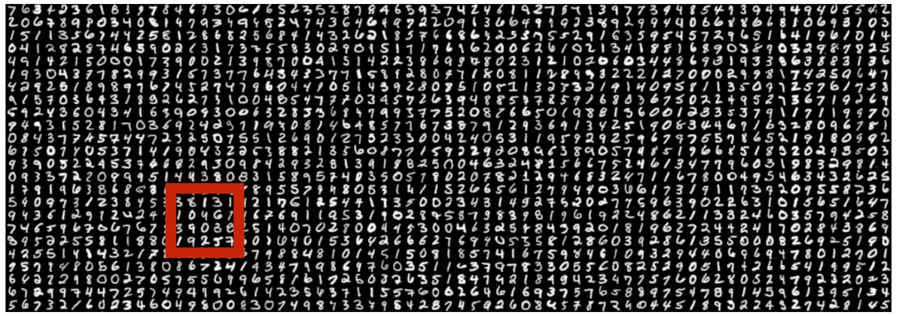
- Start with a random initialization

$\mathbf{W}_{i,j} \sim U(-c, c)$, with $U(-c, c)$ the uniform distribution over $[-c, c]$.



- Select a training subset (minibatch, 1-100 examples)

$\mathbb{B} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m_B)}, \mathbf{y}^{(m_B)})\}$

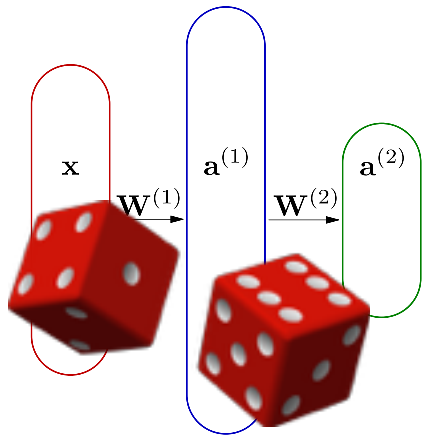


Stochastic gradient descent

Iteratively optimizing over subsets of the dataset is efficient and works surprisingly well.
At each iteration a new subset is chosen to decent closer to the minimum of the full energy.

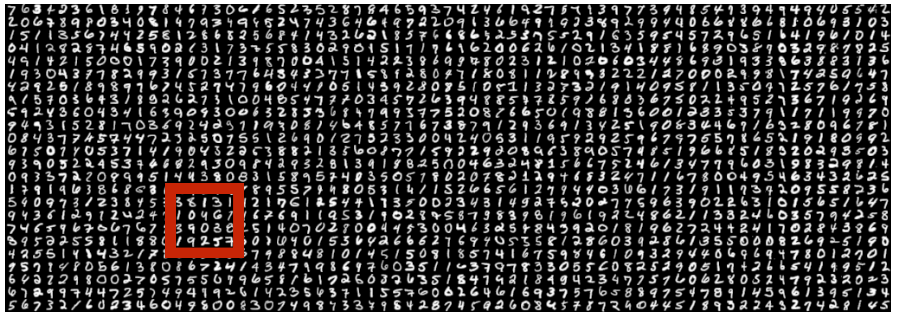
- Start with a random initialization

$\mathbf{W}_{i,j} \sim U(-c, c)$, with $U(-c,c)$ the uniform distribution over $[-c, c]$.



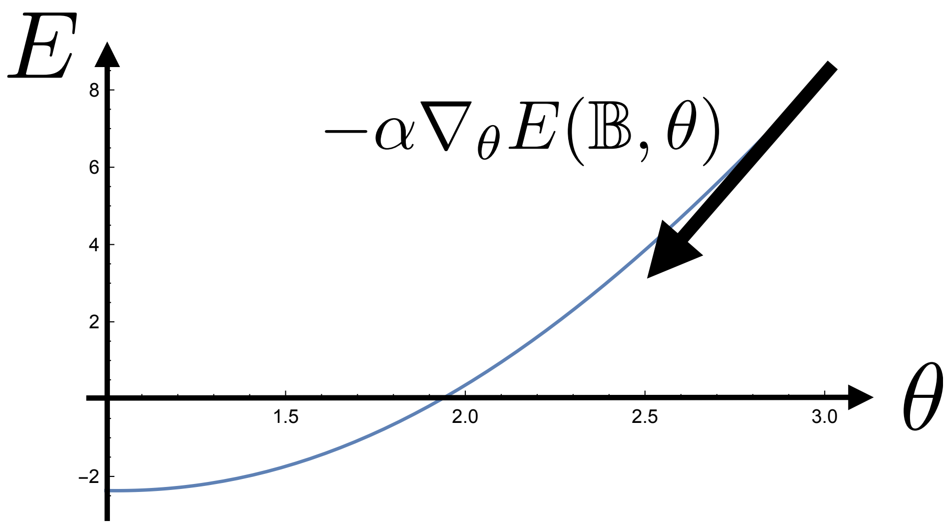
- Select a training subset (minibatch, 1-100 examples)

$$\mathbb{B} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m_B)}, \mathbf{y}^{(m_B)})\}$$



- Compute gradient with respect to parameters

$$\nabla_\theta E(\mathbb{B}, \theta)$$

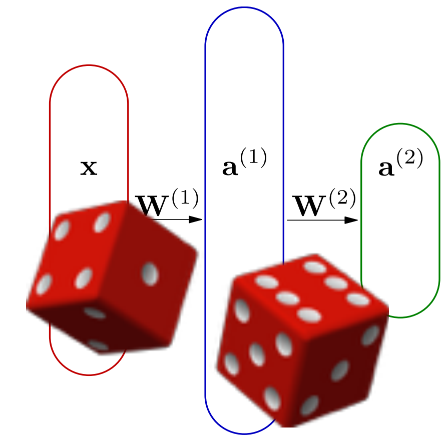


Stochastic gradient descent

Iteratively optimizing over subsets of the dataset is efficient and works surprisingly well.
At each iteration a new subset is chosen to decent closer to the minimum of the full energy.

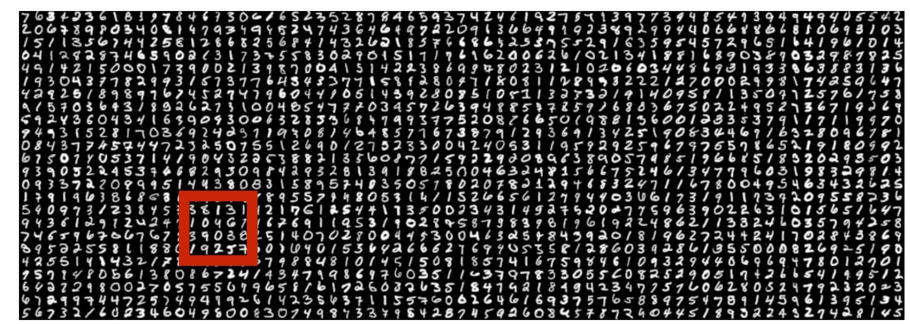
- Start with a random initialization

$\mathbf{W}_{i,j} \sim U(-c, c)$, with $U(-c, c)$ the uniform distribution over $[-c, c]$.



- Select a training subset (minibatch, 1-100 examples)

$$\mathbb{B} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m_B)}, \mathbf{y}^{(m_B)})\}$$

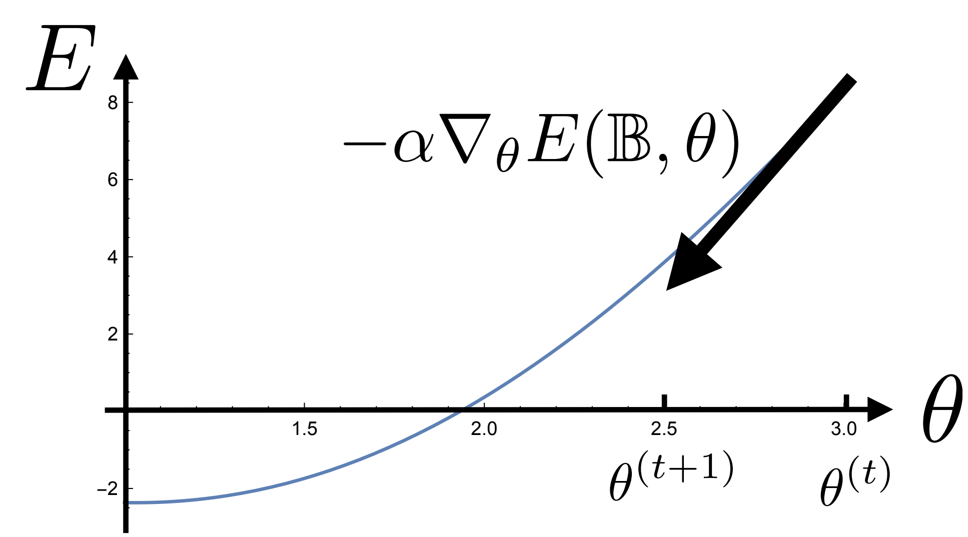


- Compute gradient with respect to parameters

$$\nabla_\theta E(\mathbb{B}, \theta)$$

- Update weights with learning rate α

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta E(\mathbb{B}, \theta)$$

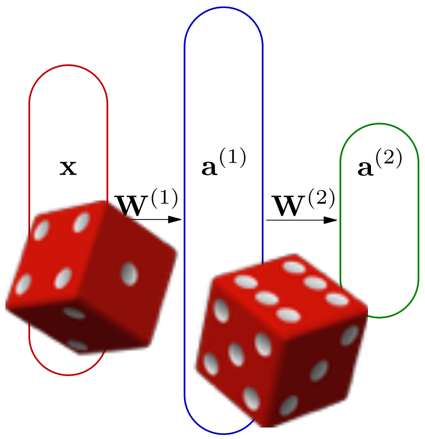


Stochastic gradient descent

Iteratively optimizing over subsets of the dataset is efficient and works surprisingly well.
At each iteration a new subset is chosen to decent closer to the minimum of the full energy.

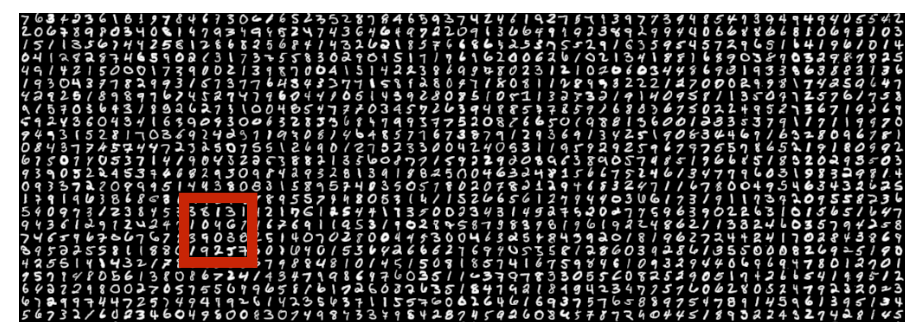
- Start with a random initialization

  $\mathbf{W}_{i,j} \sim U(-c, c)$, with $U(-c,c)$ the uniform distribution over $[-c, c]$.



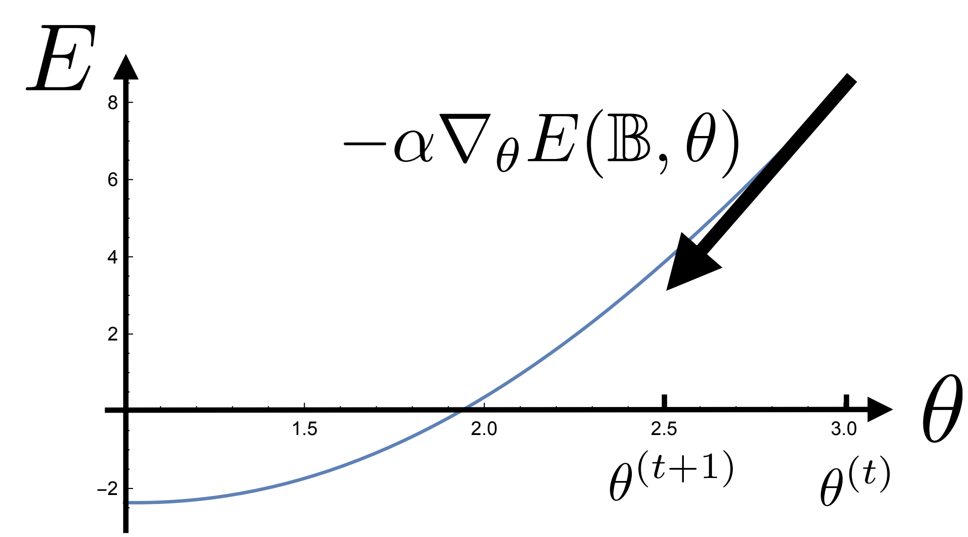
- Select a training subset (minibatch, 1-100 examples)

  $$\mathbb{B} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m_B)}, \mathbf{y}^{(m_B)})\}$$



- Compute gradient with respect to parameters

  $$\nabla_\theta E(\mathbb{B}, \theta)$$



- Update weights with learning rate α

  $$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta E(\mathbb{B}, \theta)$$

- Iterate on a new minibatch

Simplistic stochastic gradient descent

In the most simple case we choose a single sample per iteration and use the squared loss.

Full energy

$$E(D, \theta) = \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} (\mathrm{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})^2$$

$= (\mathrm{nn}($  $, \theta) - 7)^2$

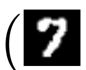
$+ (\mathrm{nn}($  $, \theta) - 8)^2 \ldots$

Simplistic stochastic gradient descent

In the most simple case we choose a single sample per iteration and use the squared loss.

Full energy

$$E(D,\theta) = \sum_{(\mathbf{x}^{(i)},y^{(i)})\in D} (\mathrm{nn}(\mathbf{x}^{(i)},\theta) - y^{(i)})^2$$

$=(\mathrm{nn}($  $, \theta) - 7)^2$

$+(\mathrm{nn}($  $, \theta) - 8)^2 \dots$

Approximation at iteration i

$$E(D,\theta) \approx E((\mathbf{x}^{(i)}, y^{(i)}), \theta)$$

$$E((\mathbf{x}^{(i)}, y^{(i)}), \theta) = (\mathrm{nn}(\mathbf{x}^{(i)},\theta) - y^{(i)})^2$$

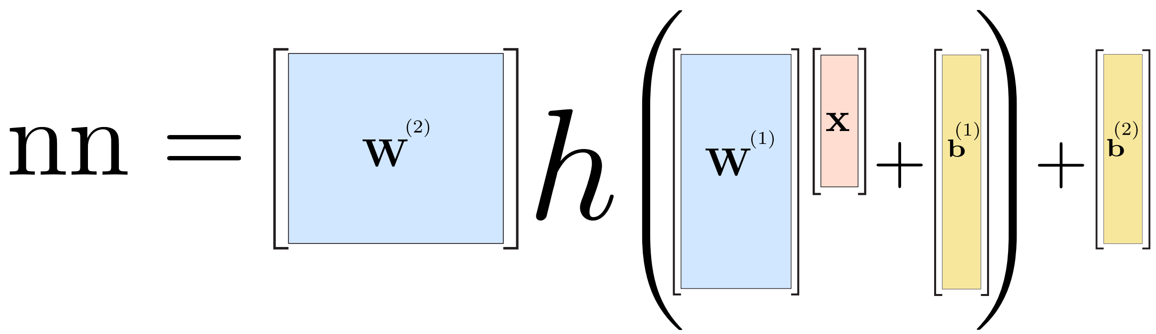
$=(\mathrm{nn}($  $, \theta) - 7)^2$

Differentiation

The NN consists of simple matrix operations — apply chain rule with matrix-vector notation.

NN function

$$\mathrm{nn} = \mathrm{linear}(h(\mathrm{linear}(\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}), \mathbf{W}^{(2)}, \mathbf{b}^{(2)})$$



Toy example, a scalar NN

$$\mathrm{nn}(x, w^{(1)}, w^{(2)}) = w^{(2)} h(w^{(1)} x)$$

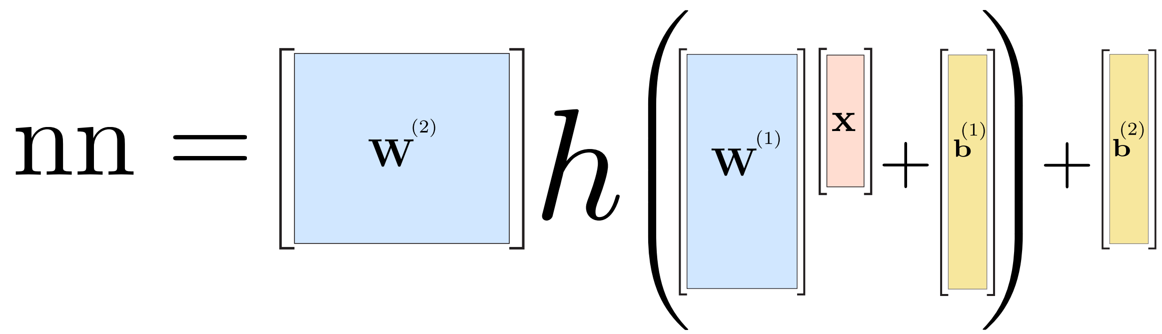
$$\frac{\partial\, \mathrm{nn}}{\partial w^{(1)}}(x, w^{(1)}, w^{(2)}) = w^{(2)} h'(w^{(1)} x) x$$

Differentiation

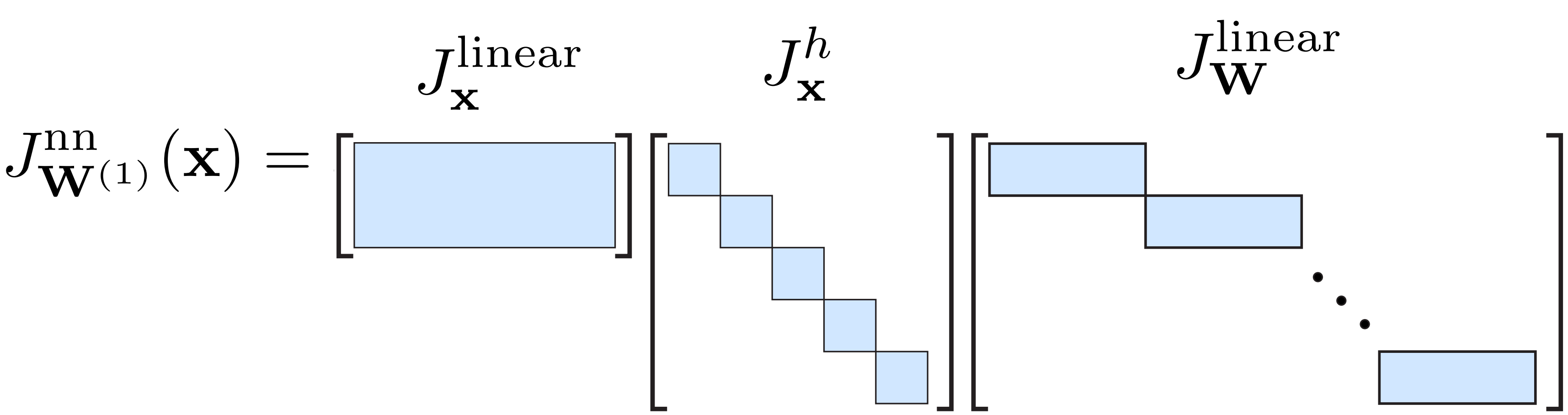
The NN consists of simple matrix operations — apply chain rule with matrix-vector notation.

NN function

$$\mathrm{nn} = \mathrm{linear}(h(\mathrm{linear}(\mathbf{x}, \mathbf{W}^{(1)}, \mathbf{b}^{(1)}), \mathbf{W}^{(2)}, \mathbf{b}^{(2)})$$



NN Jacobian matrix



Jacobian matrices

Two of the three involved Jacobian matrix types have sparse structure, we exploit that later.

$$\mathbf{J}_{\mathbf{x}}^{f} = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_m}{\partial \mathbf{x}_n} \end{bmatrix}$$

Jacobian matrices

Two of the three involved Jacobian matrix types have sparse structure, we exploit that later.

function to differentiate…

$$\mathbf{J}_{\mathbf{x}}^{f} = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_m}{\partial \mathbf{x}_n} \end{bmatrix}$$

…with respect to x

Jacobian matrices

Two of the three involved Jacobian matrix types have sparse structure, we exploit that later.

function to differentiate…

$$\mathbf{J}_{\mathbf{x}}^{f} = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_m}{\partial \mathbf{x}_n} \end{bmatrix}$$

…with respect to x

$$h = \mathrm{relu}(\mathbf{x}) \longrightarrow \mathbf{J}_{\mathbf{x}}^{h} = \begin{bmatrix} h' & & & & & \\ & h' & & & & \\ & & h' & & & \\ & & & h' & & \\ & & & & h' & \\ & & & & & h' \end{bmatrix}$$

Jacobian matrices

Two of the three involved Jacobian matrix types have sparse structure, we exploit that later.

function to differentiate…

$$\mathbf{J}_{\mathbf{x}}^{f} = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_m}{\partial \mathbf{x}_n} \end{bmatrix}$$

…with respect to x

$$h = \mathrm{relu}(\mathbf{x}) \longrightarrow \mathbf{J}_{\mathbf{x}}^{h} = \begin{bmatrix} h' & & & & & \\ & h' & & & & \\ & & h' & & & \\ & & & h' & & \\ & & & & h' & \\ & & & & & h' \end{bmatrix}$$

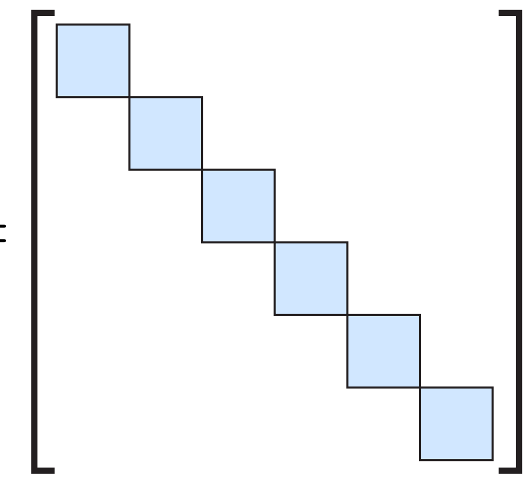
$$\mathrm{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b} \longrightarrow \mathbf{J}_{\mathbf{x}}^{\mathrm{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} = \begin{bmatrix} \end{bmatrix}$$

Jacobi matrices

Two of the three involved Jacobi matrix types have sparse structure, we exploit that later.

$$\mathbf{J}^f_{\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_m}{\partial \mathbf{x}_n} \end{bmatrix}$$

$$\mathbf{J}_{\mathbf{b}}^{\mathrm{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} =$$

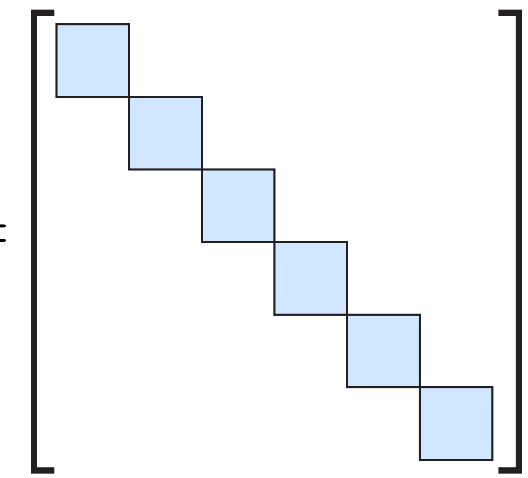


Jacobi matrices

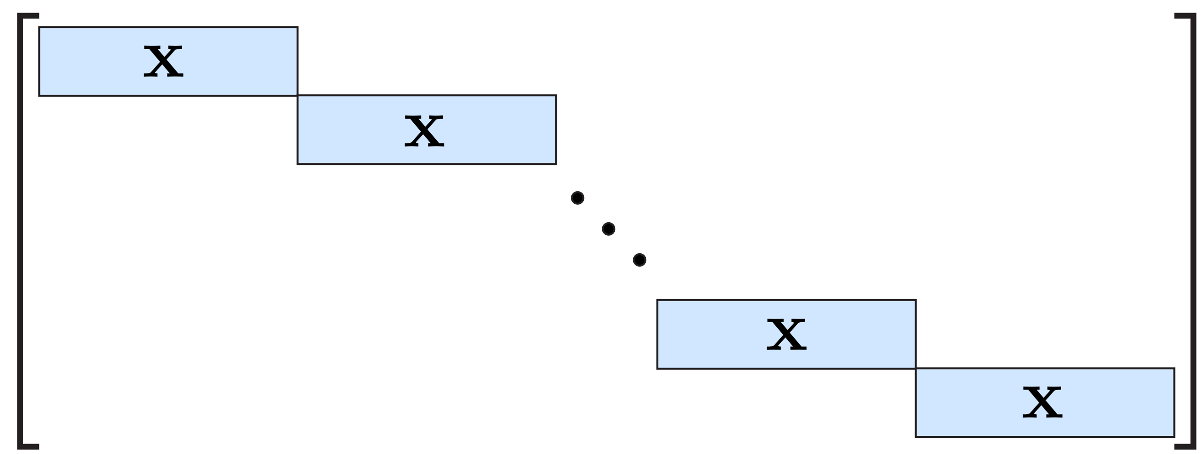
Two of the three involved Jacobi matrix types have sparse structure, we exploit that later.

$$\mathbf{J}_{\mathbf{x}}^{f} = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_m}{\partial \mathbf{x}_n} \end{bmatrix}$$

$$\mathbf{J}_{\mathbf{b}}^{\mathrm{linear}(\mathbf{x},\,\mathbf{W},\,\mathbf{b})} =$$



$$\mathbf{J}_{\mathbf{W}}^{\mathrm{linear}(\mathbf{x},\,\mathbf{W},\,\mathbf{b})} =$$

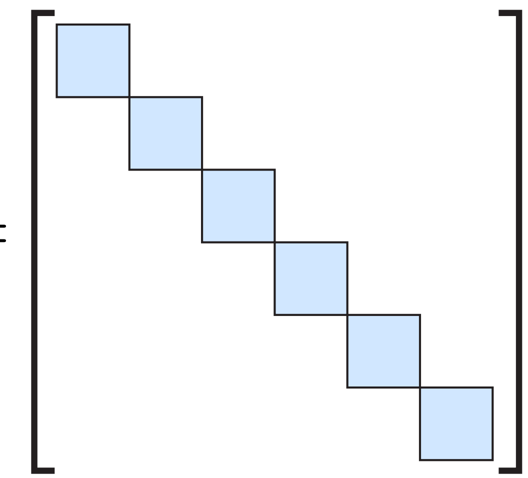


Jacobi matrices

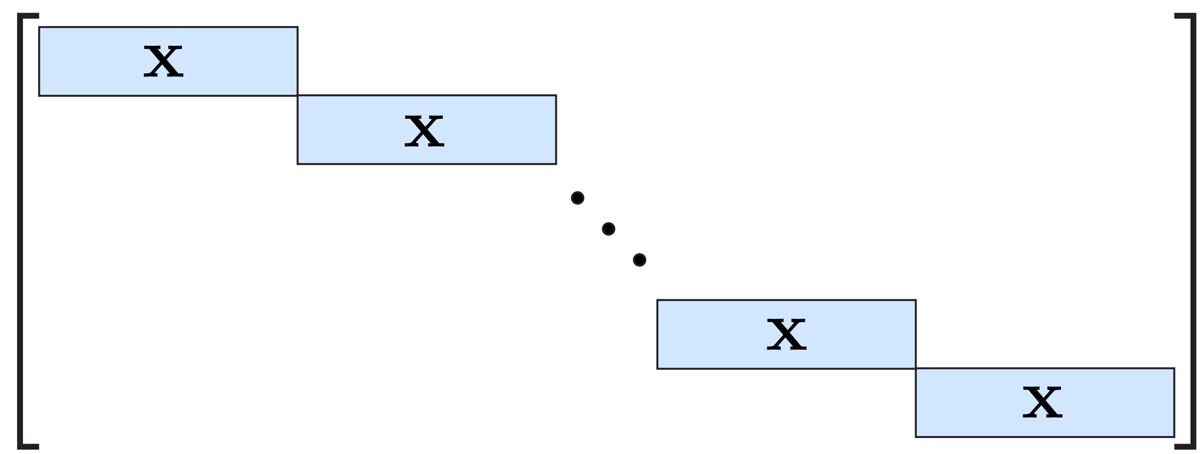
Two of the three involved Jacobi matrix types have sparse structure, we exploit that later.

$$\mathbf{J}_{\mathbf{x}}^{f} = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_m}{\partial \mathbf{x}_n} \end{bmatrix}$$

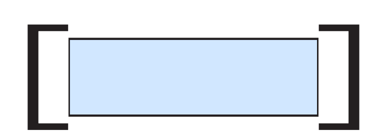
$$\mathbf{J}_{\mathbf{b}}^{\mathrm{linear}(\mathbf{x},\, \mathbf{W},\, \mathbf{b})} =$$



$$\mathbf{J}_{\mathbf{W}}^{\mathrm{linear}(\mathbf{x},\, \mathbf{W},\, \mathbf{b})} =$$

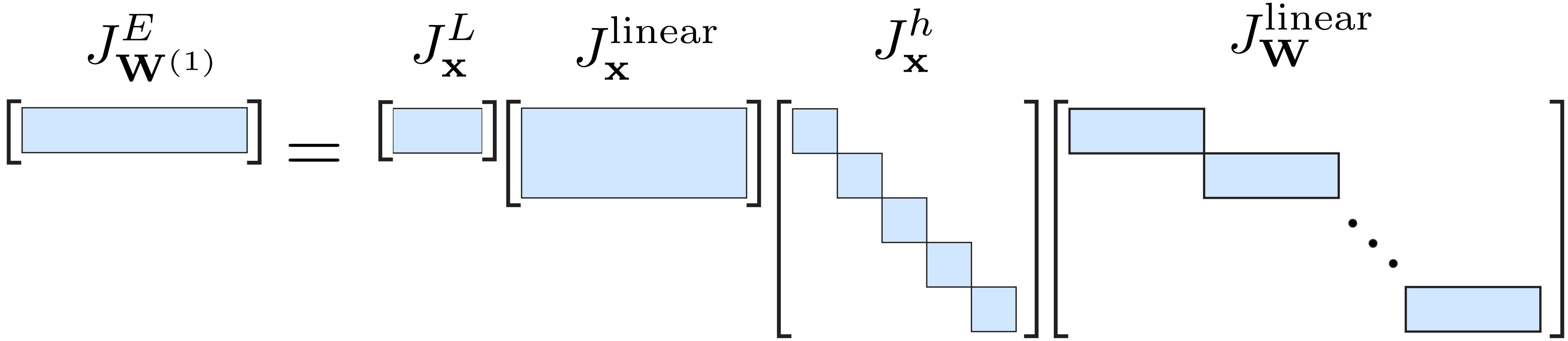


$$L(\mathbf{x}, \mathbf{v}) = \sum_i (\mathbf{x}_i - \mathbf{y}_i)^2 \longrightarrow J_{\mathbf{x}}^{L} =$$



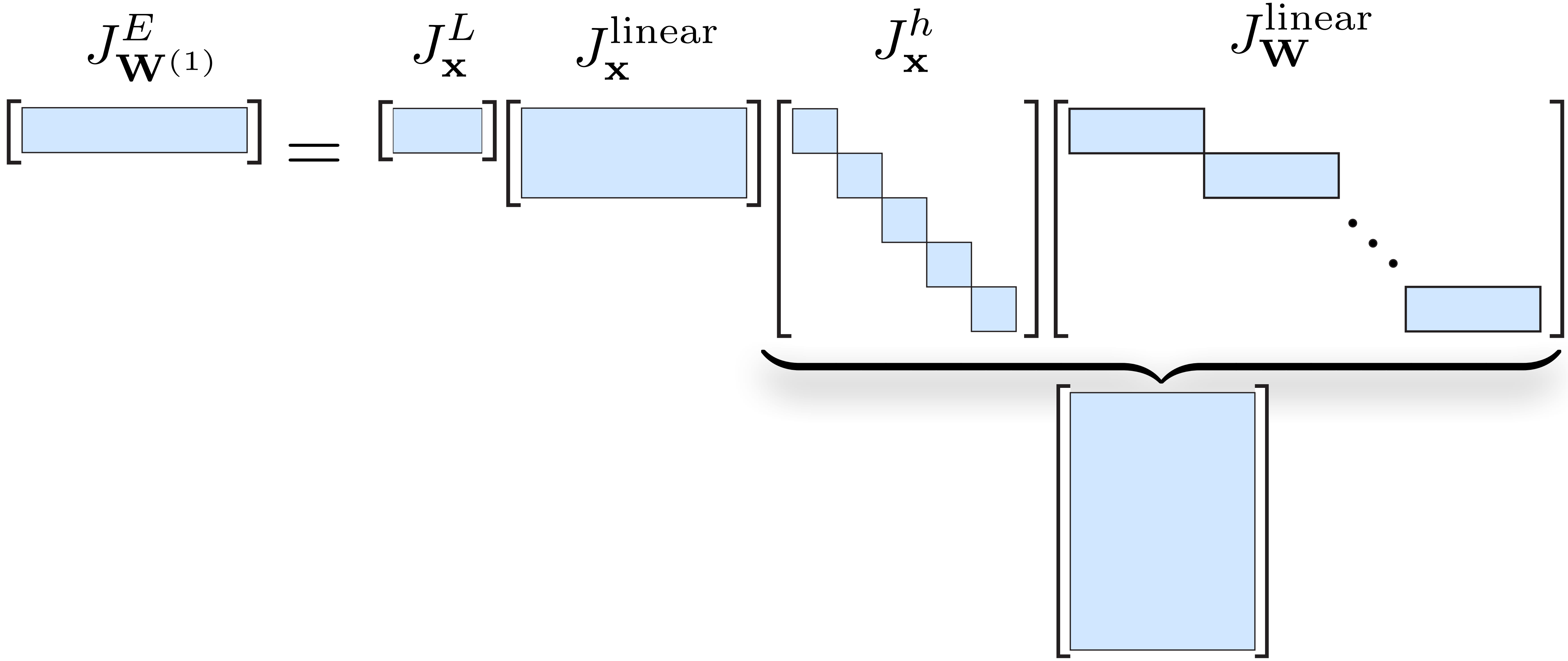
Automatic differentiation — forward mode

In forward mode, Jacobian matrices are multiplied from back to front, i.e., in the same way as x passes through the network in the forward pass.



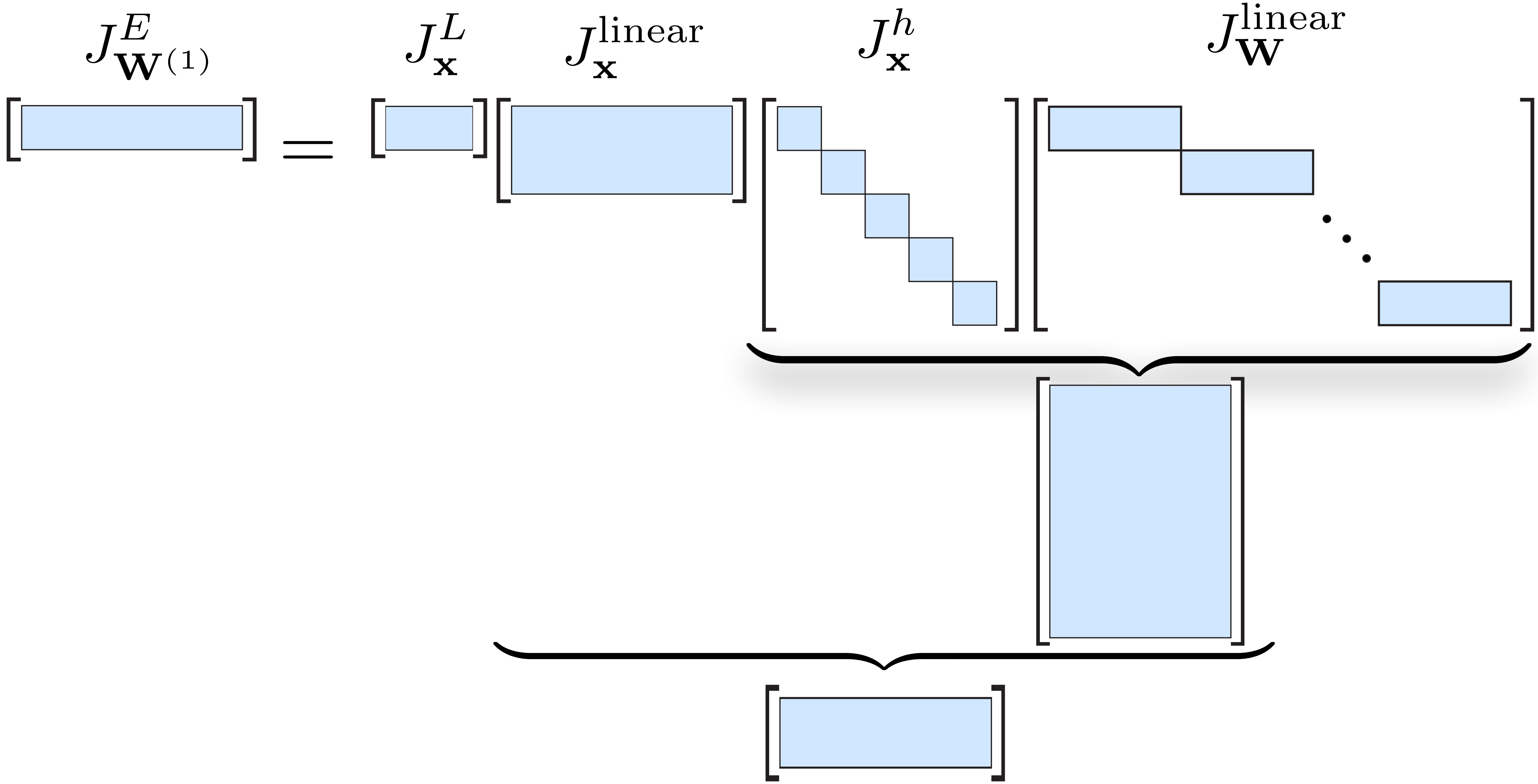
Automatic differentiation — forward mode

In forward mode, Jacobian matrices are multiplied from back to front, i.e., in the same way as x passes through the network in the forward pass.



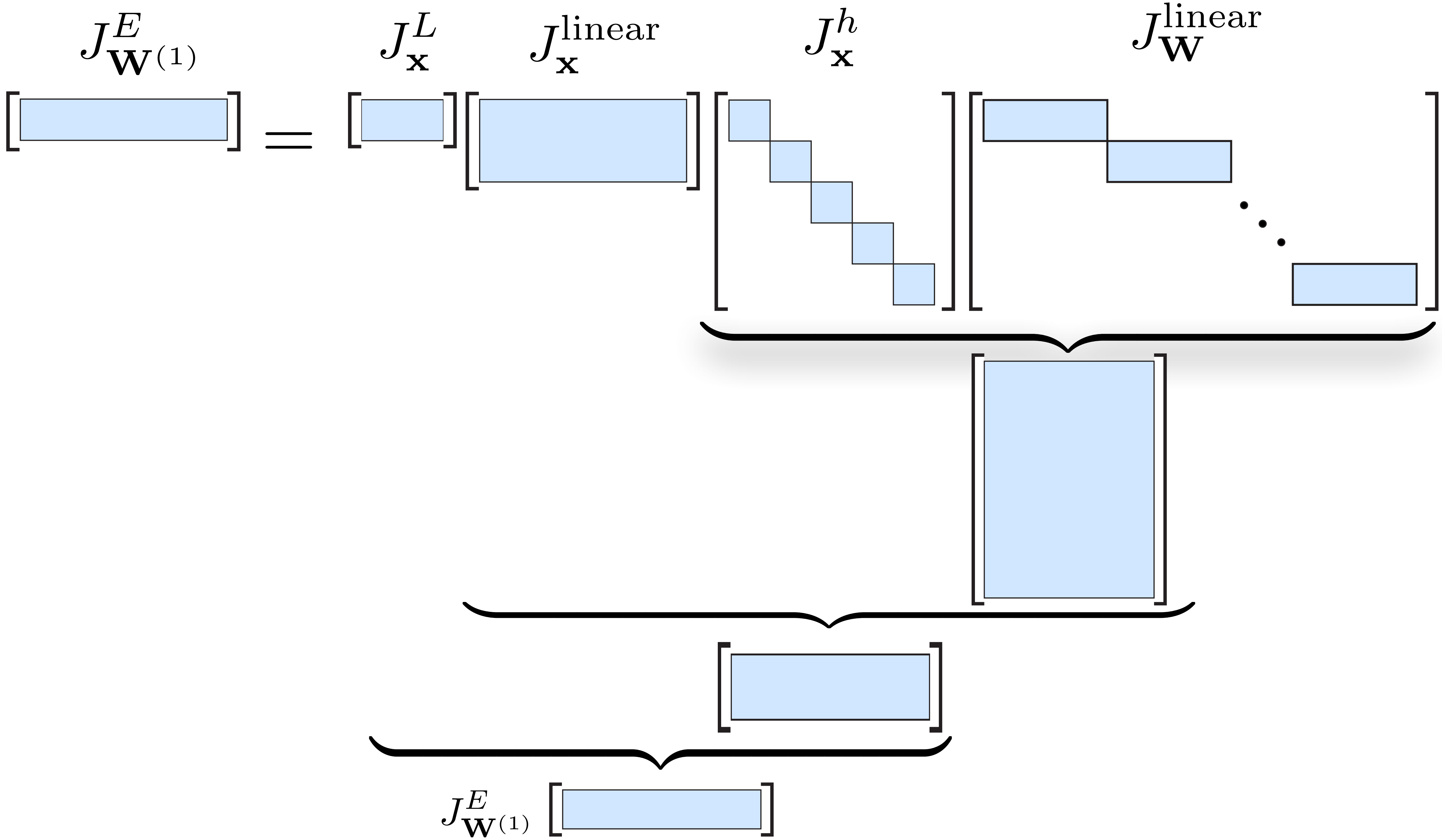
Automatic differentiation — forward mode

In forward mode, Jacobian matrices are multiplied from back to front, i.e., in the same way as x passes through the network in the forward pass.



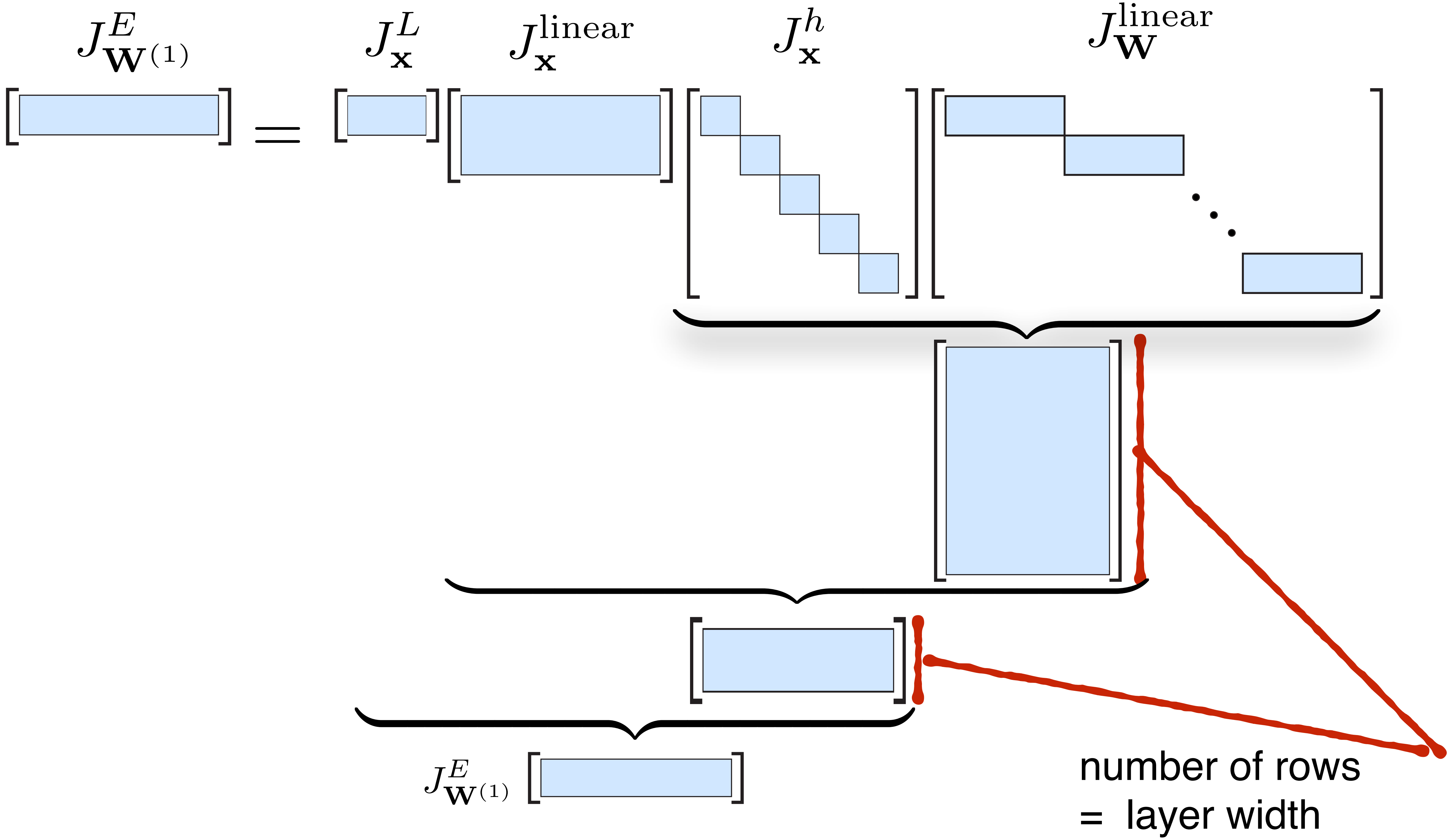
Automatic differentiation — forward mode

In forward mode, Jacobian matrices are multiplied from back to front, i.e., in the same way as x passes through the network in the forward pass.



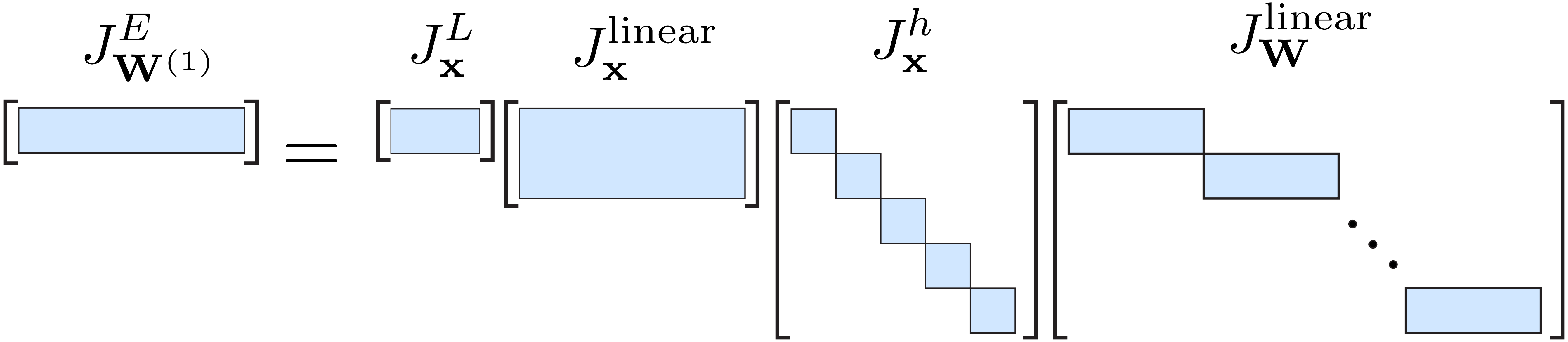
Automatic differentiation — forward mode

In forward mode, Jacobian matrices are multiplied from back to front, i.e., in the same way as x passes through the network in the forward pass.



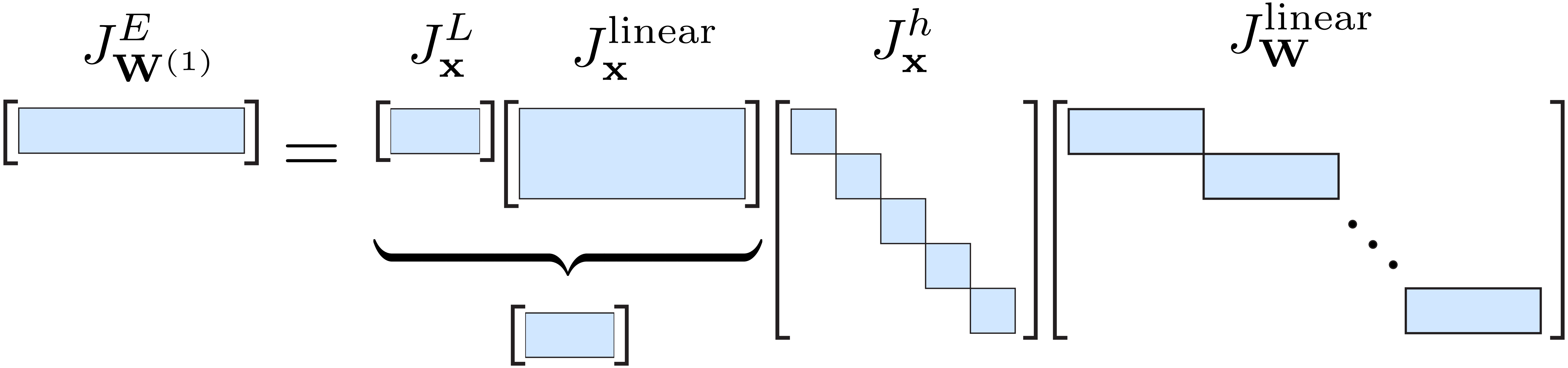
Automatic differentiation —reverse mode

In reverse mode automatic differentiation Jacobi matrices are multiplied from front to back.



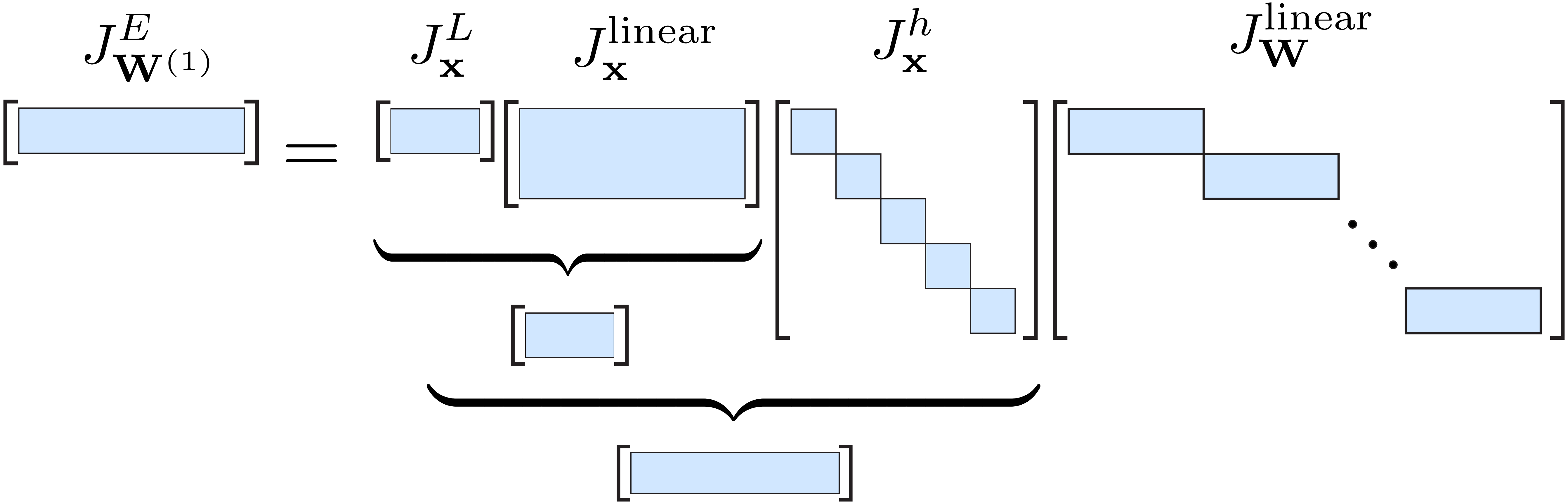
Automatic differentiation —reverse mode

In reverse mode automatic differentiation Jacobi matrices are multiplied from front to back.



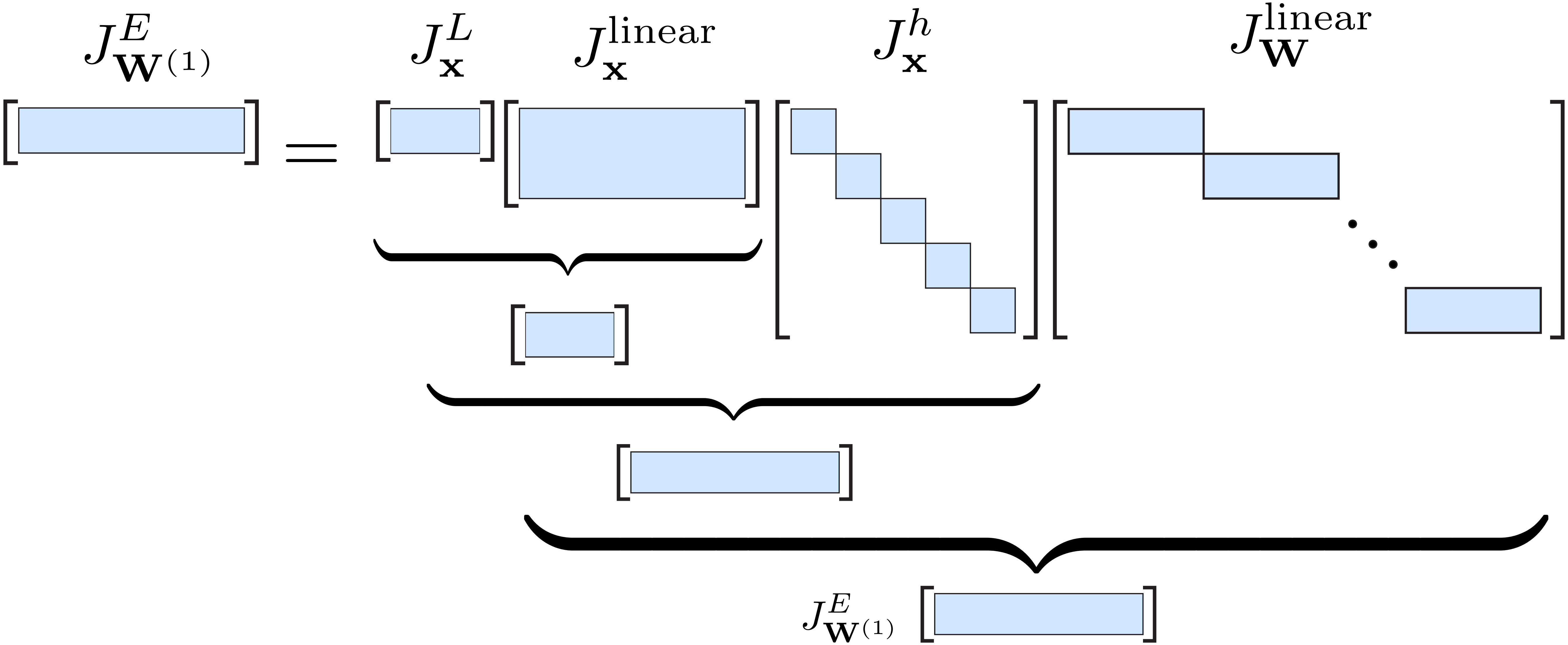
Automatic differentiation —reverse mode

In reverse mode automatic differentiation Jacobi matrices are multiplied from front to back.



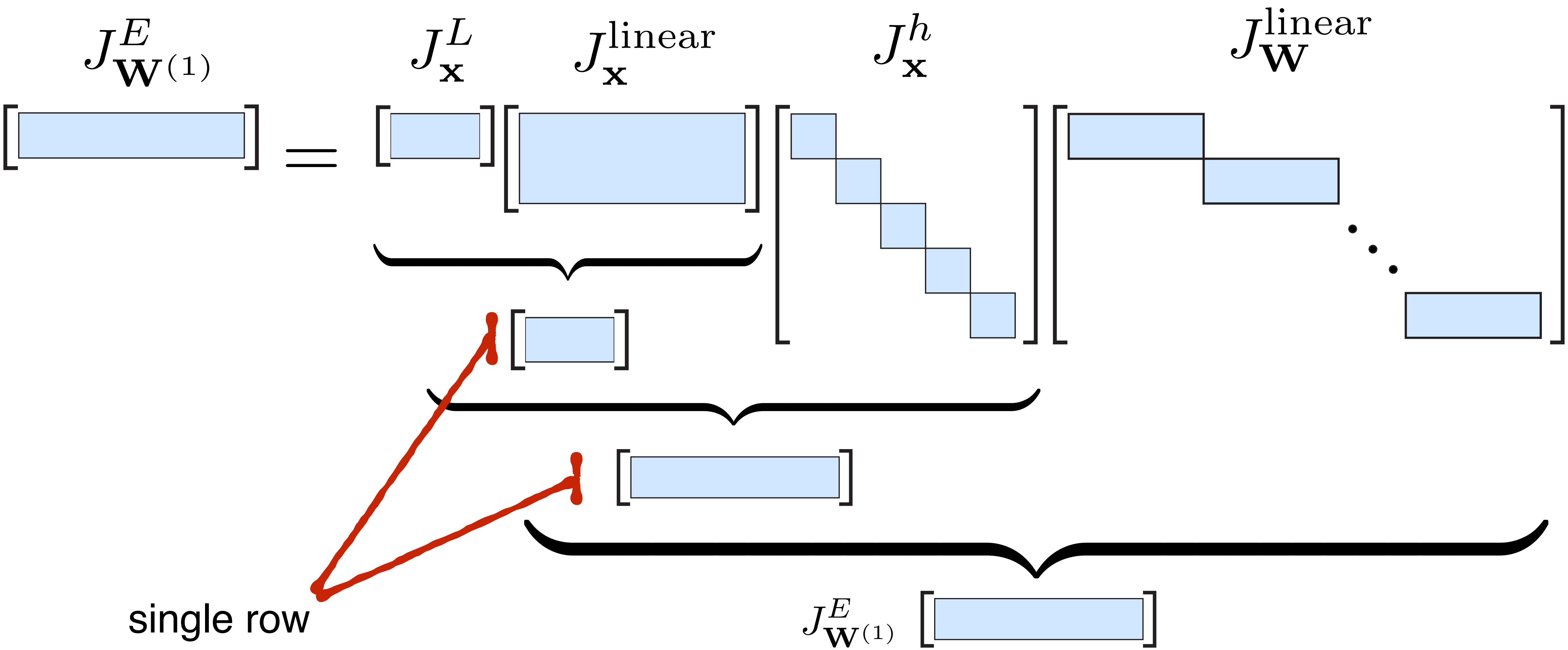
Automatic differentiation —reverse mode

In reverse mode automatic differentiation Jacobi matrices are multiplied from front to back.



Automatic differentiation —reverse mode

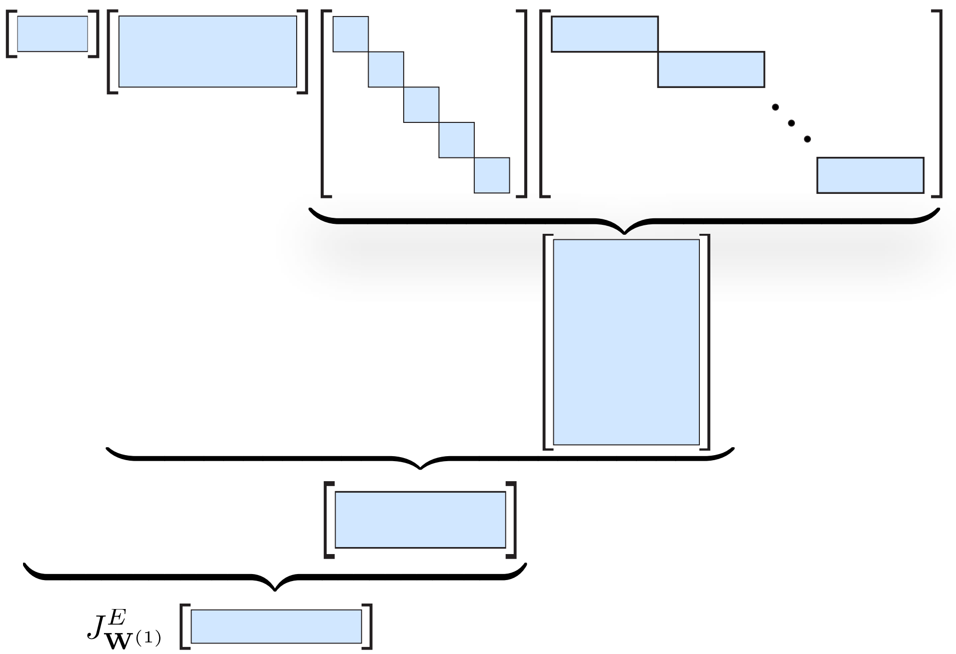
In reverse mode automatic differentiation Jacobi matrices are multiplied from front to back.



Forward vs. reverse mode

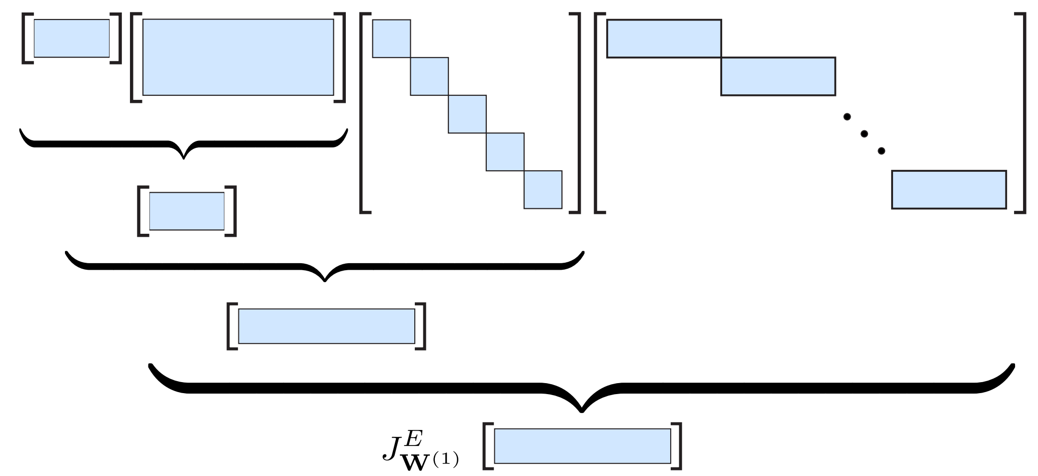
Reverse accumulation is more efficient for NNs since the objective function is a scalar.

Forward accumulation



Forward accumulation is more efficient for functions that have more outputs than inputs.

Reverse accumulation



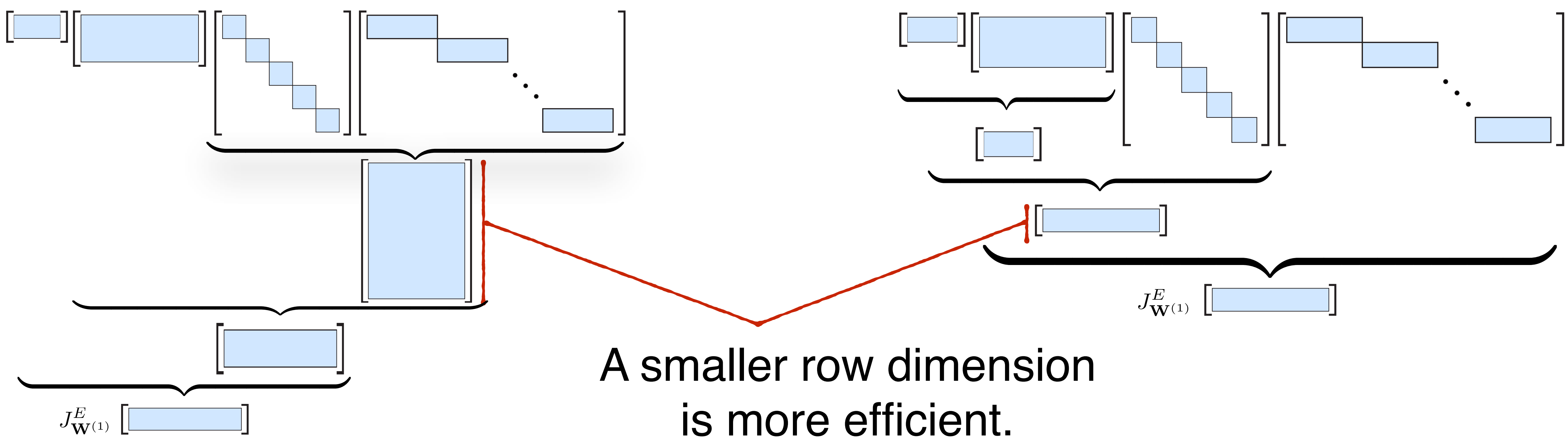
Reverse accumulation is more efficient for functions that have more inputs than outputs.

Forward vs. reverse mode

Reverse accumulation is more efficient for NNs since the objective function is a scalar.

Forward accumulation

Reverse accumulation



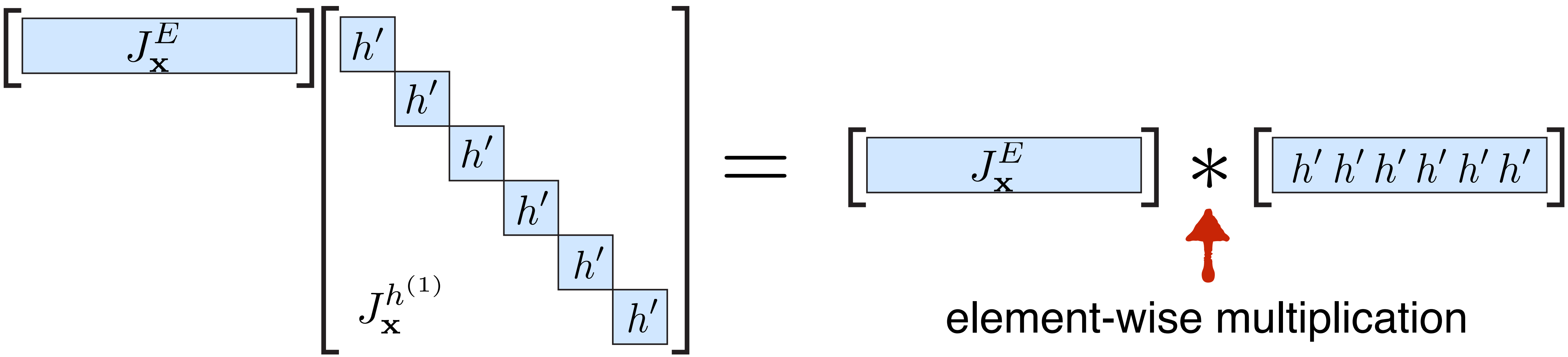
Forward accumulation is more efficient for functions that have more outputs than inputs.

Reverse accumulation is more efficient for functions that have more inputs than outputs.

Backpropagation — a special case

Creating the Jacobian matrices is expensive. Instead, matrix products can be simplified.

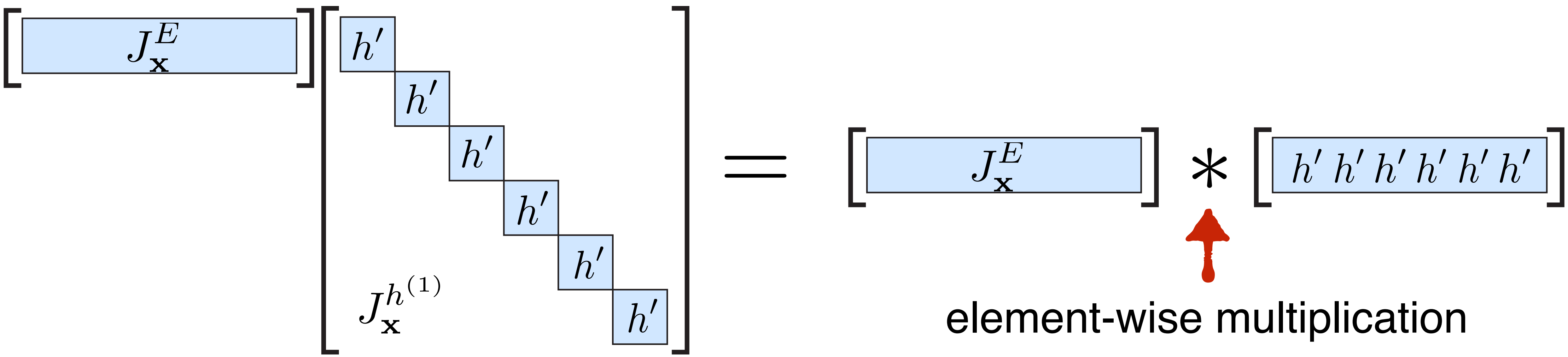
Backpropagation through activation function



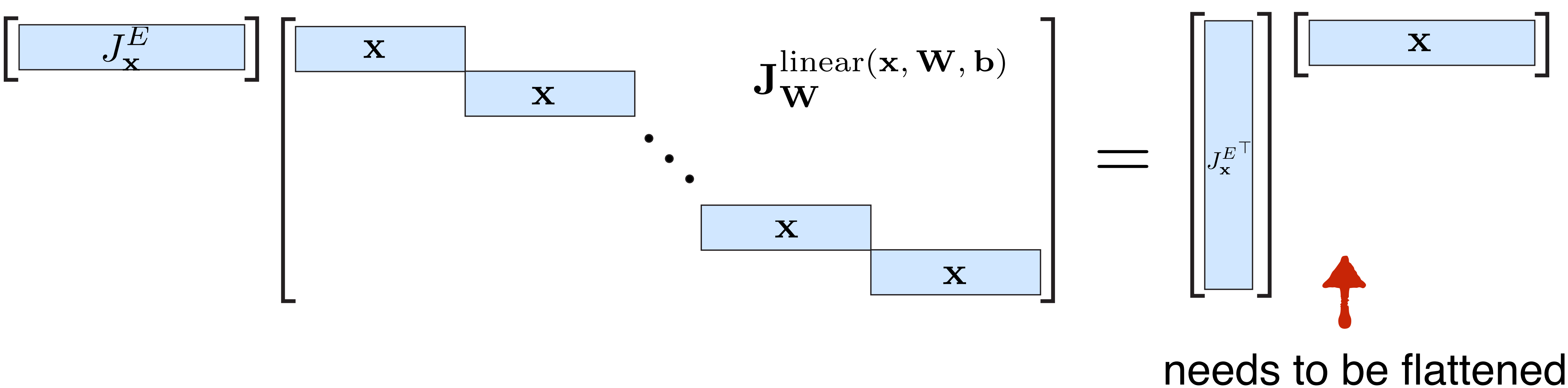
Backpropagation — a special case

Creating the Jacobian matrices is expensive. Instead, matrix products can be simplified.

Backpropagation through activation function



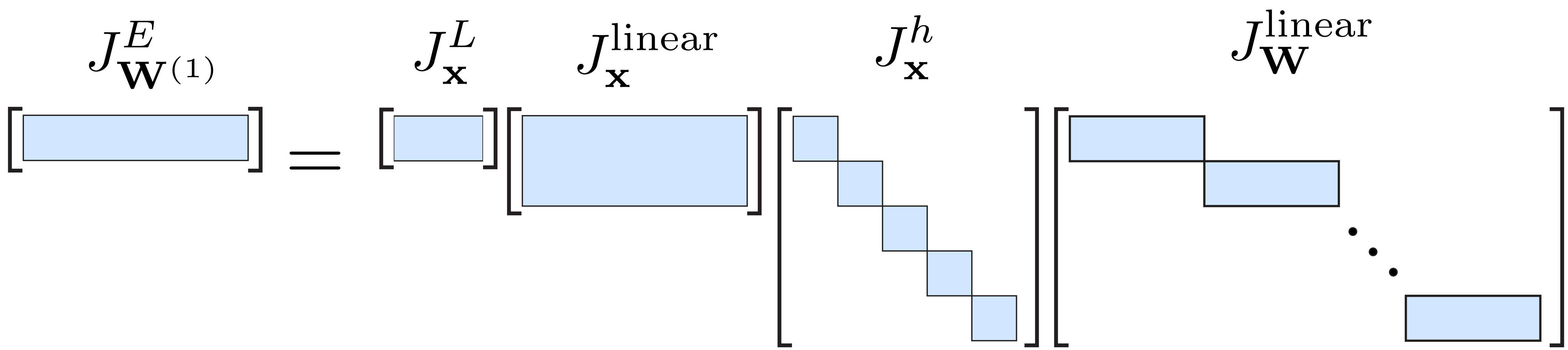
Backpropagation through linear layer



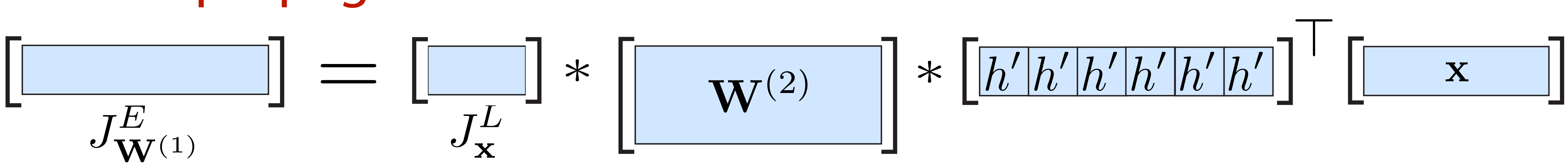
Backpropagation of a two hidden layer NN

Backpropagation is a form of reverse automatic differentiation, where the Jacobi matrix is not explicitly computed. The gradient is propagated by simpler equivalent operations.

Jacobian formulation



Compact backpropagation



NN in a nutshell

Successful training of NNs requires well chosen yet simple building blocks.

NN in a nutshell

Successful training of NNs requires well chosen yet simple building blocks.

1. Problem definition
    - input and output representation

NN in a nutshell

Successful training of NNs requires well chosen yet simple building blocks.

1. Problem definition
    - input and output representation
2. Dataset
    - pairs of desired input and output

NN in a nutshell

Successful training of NNs requires well chosen yet simple building blocks.

1. Problem definition
    - input and output representation
2. Dataset
    - pairs of desired input and output

$$\sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} L(\mathrm{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})$$

3. Objective and loss function
    - sum over loss on dataset samples

NN in a nutshell

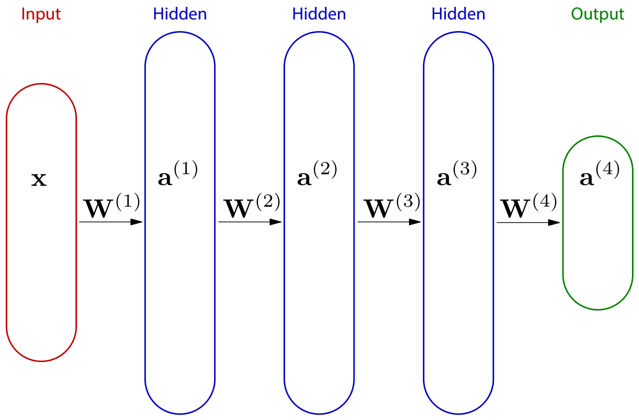
Successful training of NNs requires well chosen yet simple building blocks.

1. Problem definition
   - input and output representation

2. Dataset
   - pairs of desired input and output

$$\sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} L(\mathrm{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})$$

3. Objective and loss function
   - sum over loss on dataset samples



4. NN model
   - stack of linear and non-linear layers

NN in a nutshell

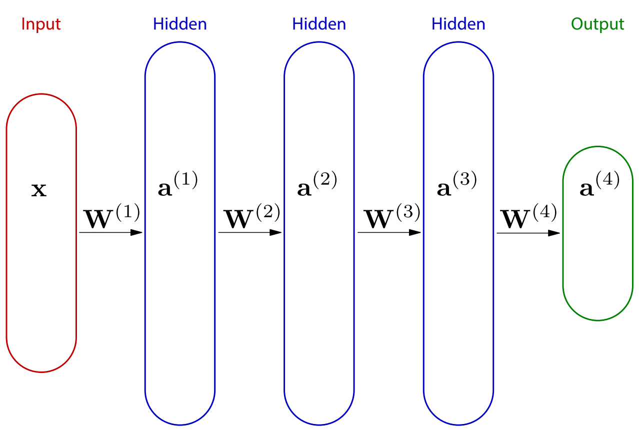
Successful training of NNs requires well chosen yet simple building blocks.

1. Problem definition
   - input and output representation

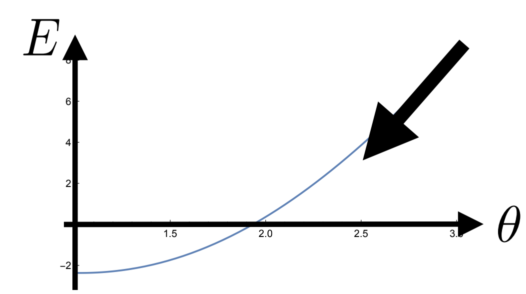
2. Dataset
   - pairs of desired input and output

$$\sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} L(\mathrm{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})$$

3. Objective and loss function
   - sum over loss on dataset samples



4. NN model
   - stack of linear and non-linear layers



5. Optimization procedure (solver)
   - iterative gradient descent approximation

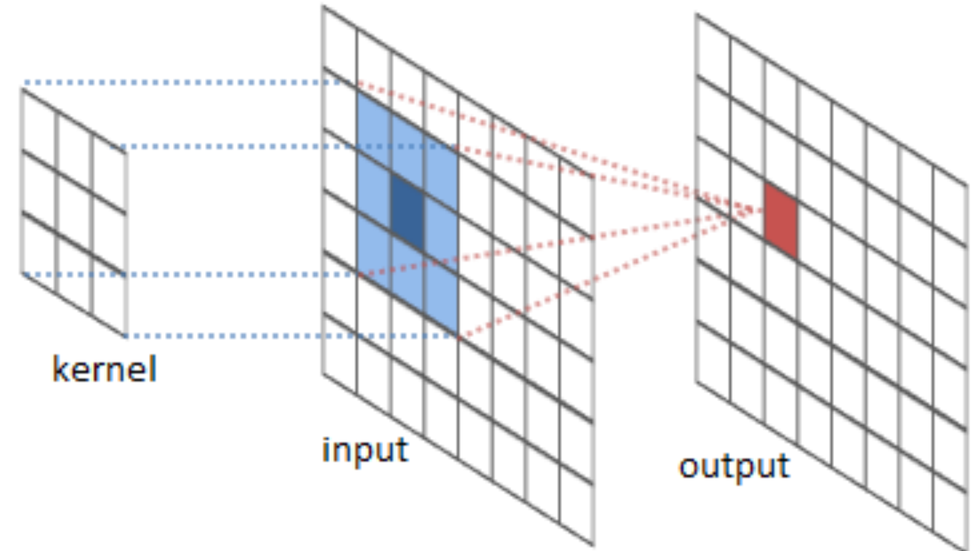
Outlook
Advanced NNs

Convolution, Filters

Filtering of images with local kernels has a long history, for instance for edge detection.

- Local kernel

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

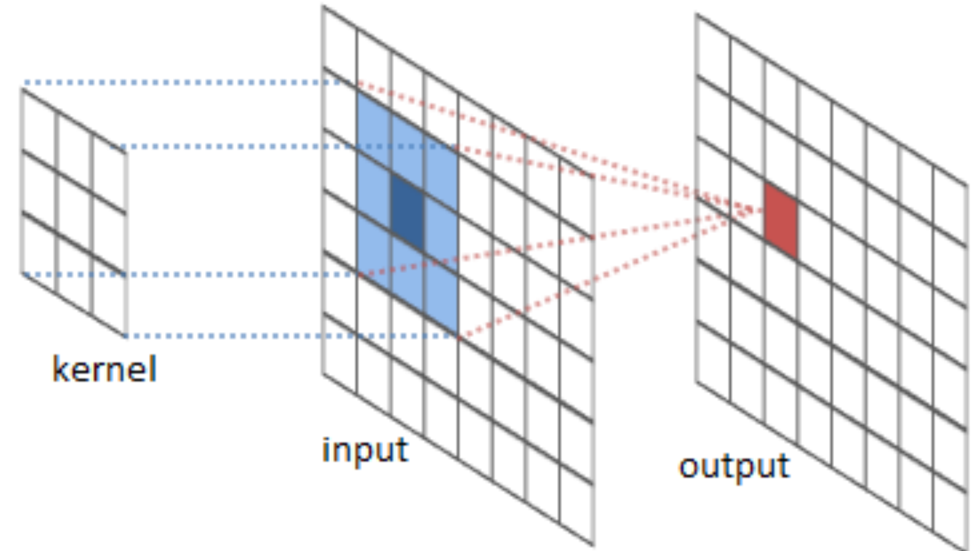


Convolution, Filters

Filtering of images with local kernels has a long history, for instance for edge detection.

- Local kernel

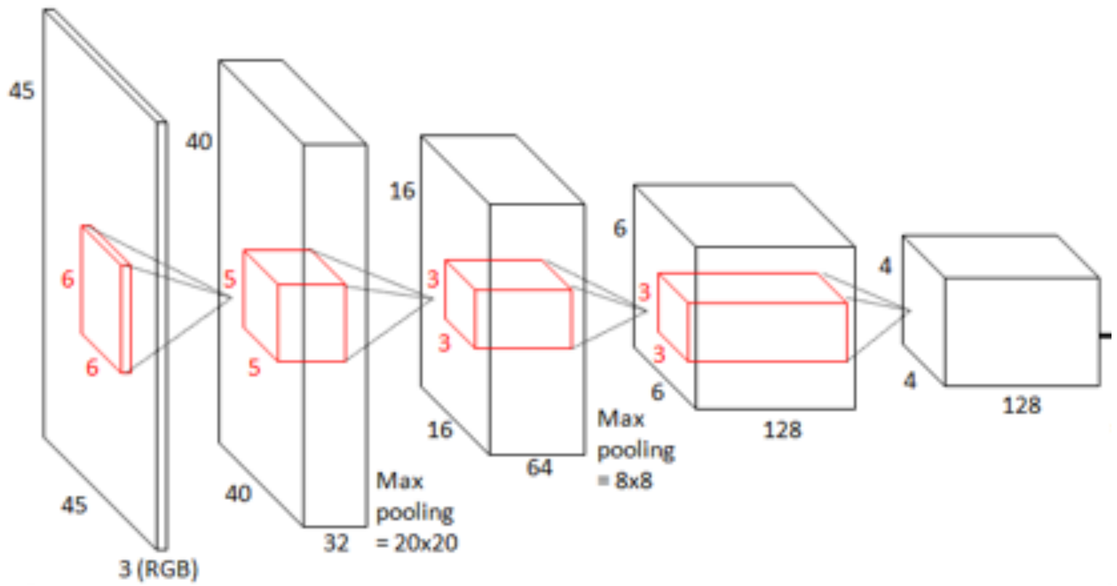
$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

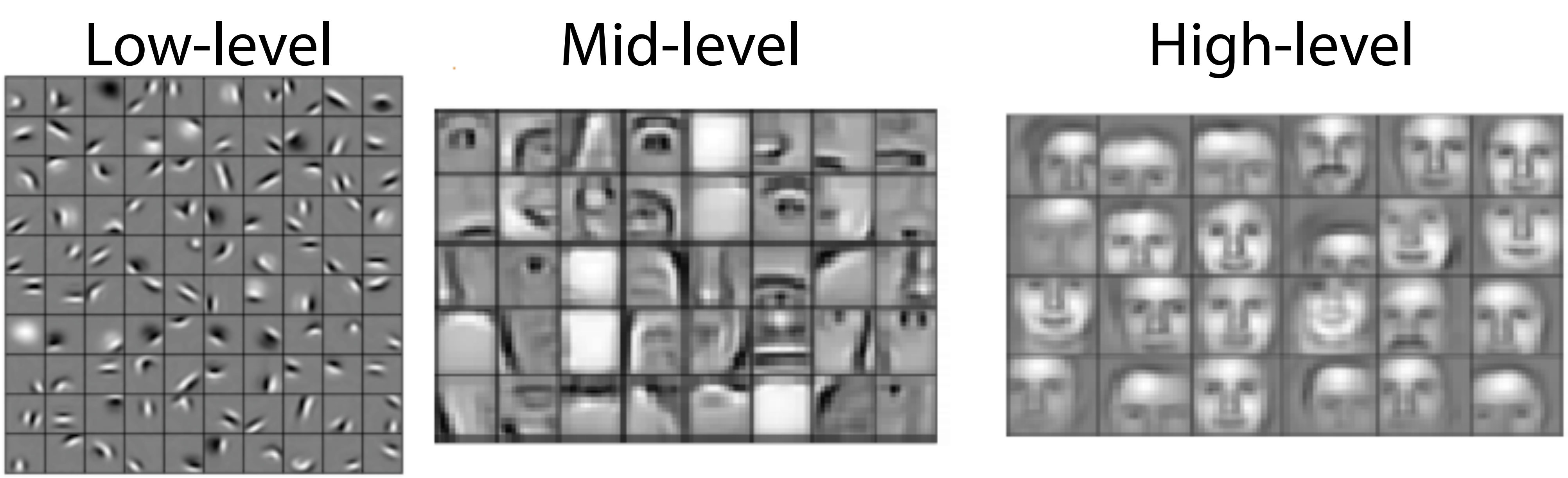


Convolutional Neural Networks

Convolutional NNs apply convolutions with trainable weights — weight sharing.

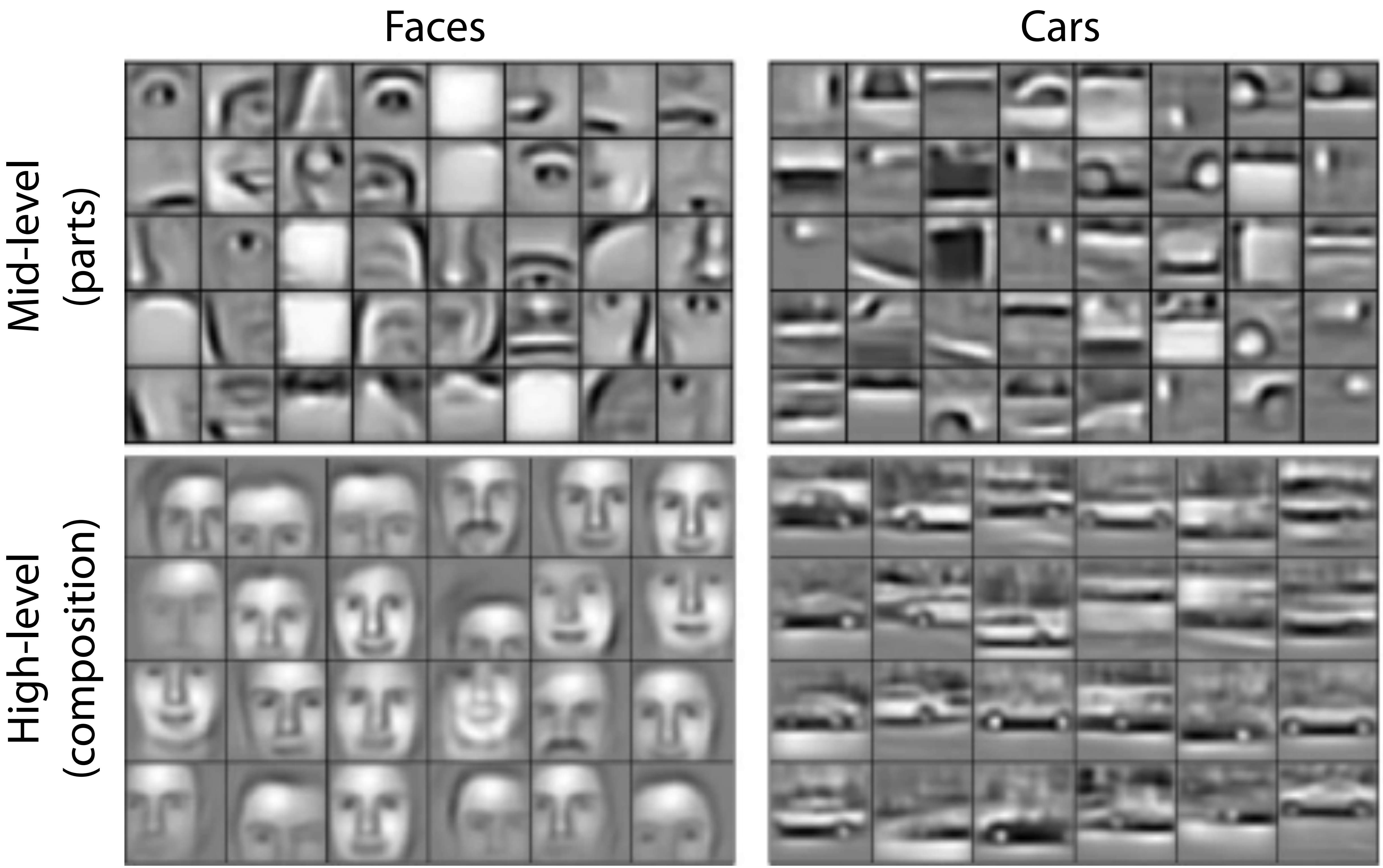
- Local operations, weights shared





Data dependent filter

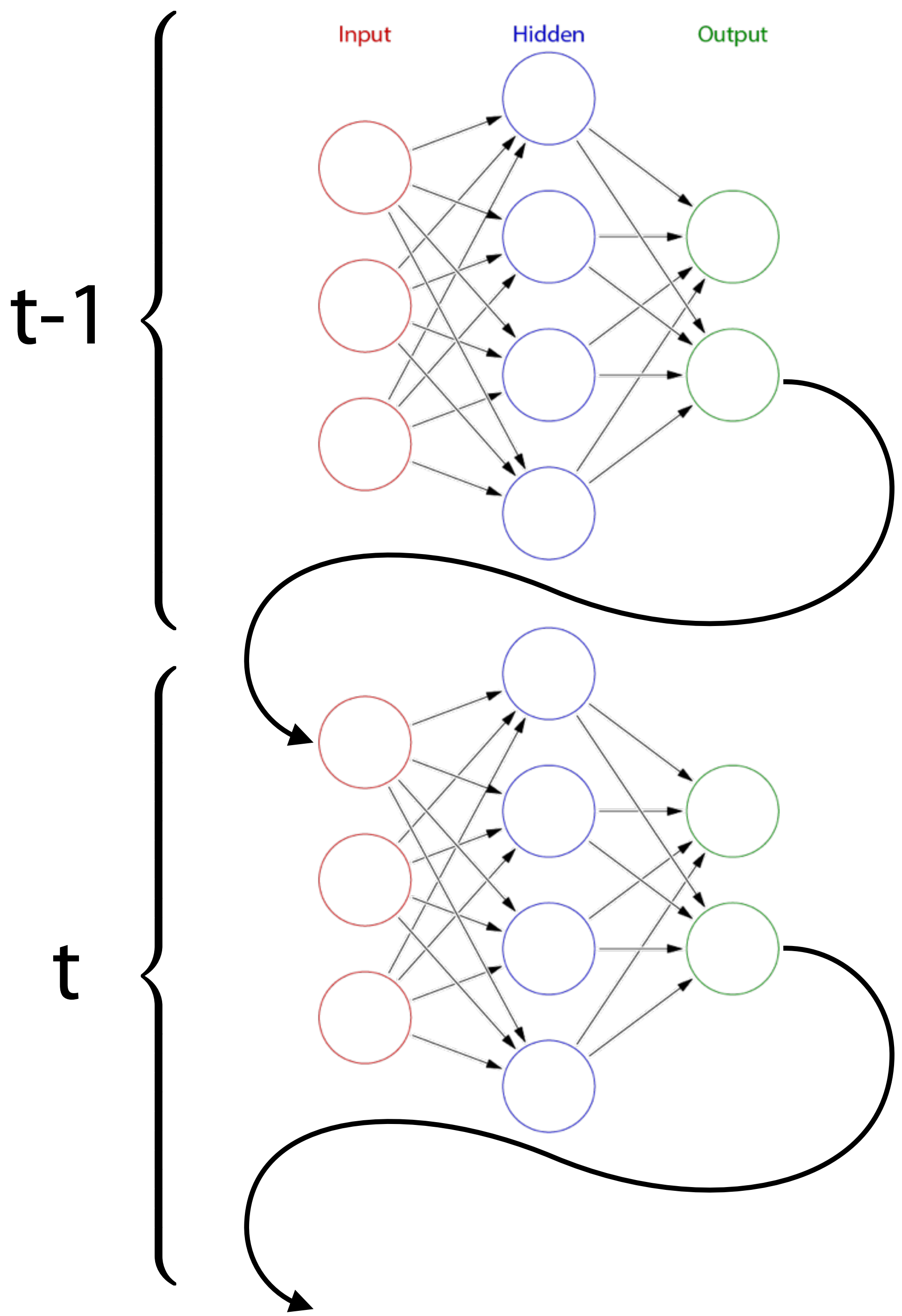
NNs capture the training examples. A network has different features on a different dataset.



Recurrent neural networks

Network structures can be complex, e.g. the input at time t can be the output of time t-1.

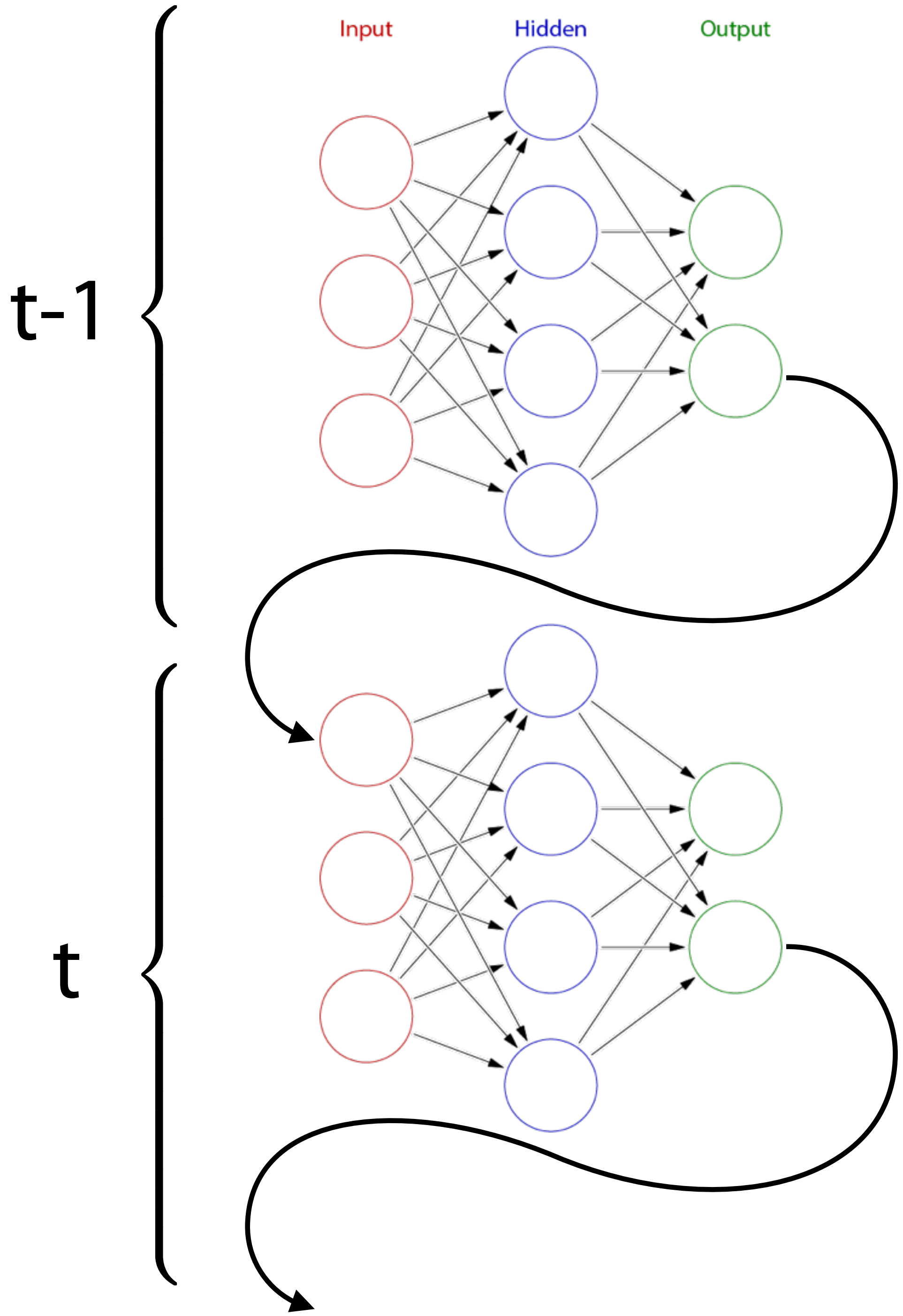
Stacking multiple NNs



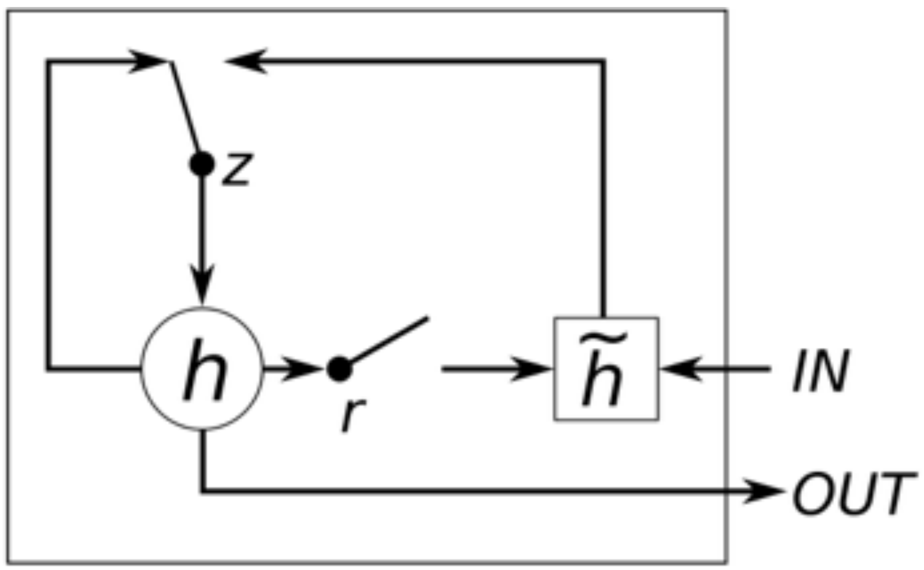
Recurrent neural networks

Network structures can be complex, e.g. the input at time t can be the output of time t-1.

Stacking multiple NNs



Gated Recurrent Units (GRU)



- A simplification of Long-term Short-Term Memory (LSTM)

Generative adversarial networks (GANs)

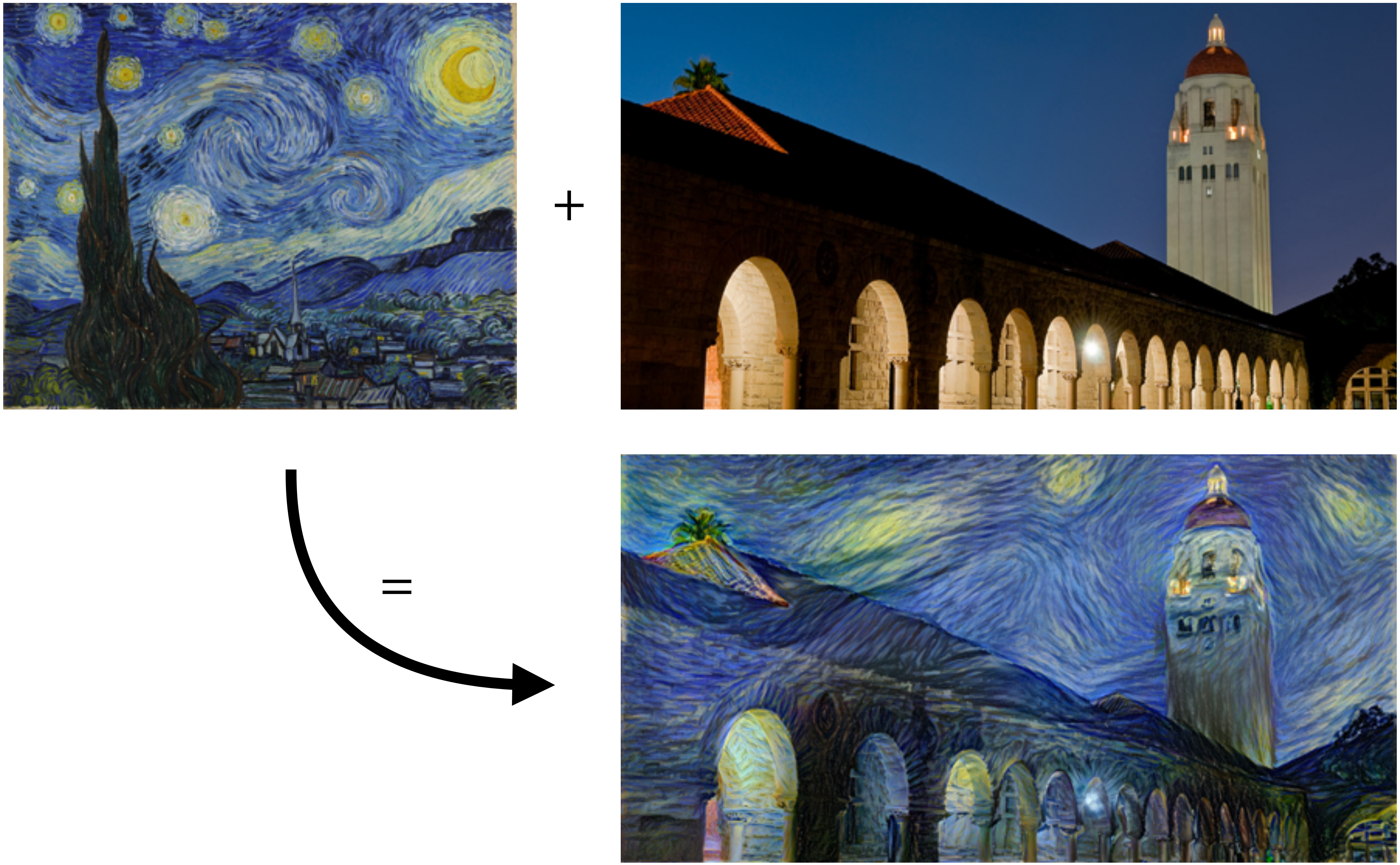
[Karras et al.]

Generative adversarial networks (GANs)

[Karras et al.]

Neural style transfer

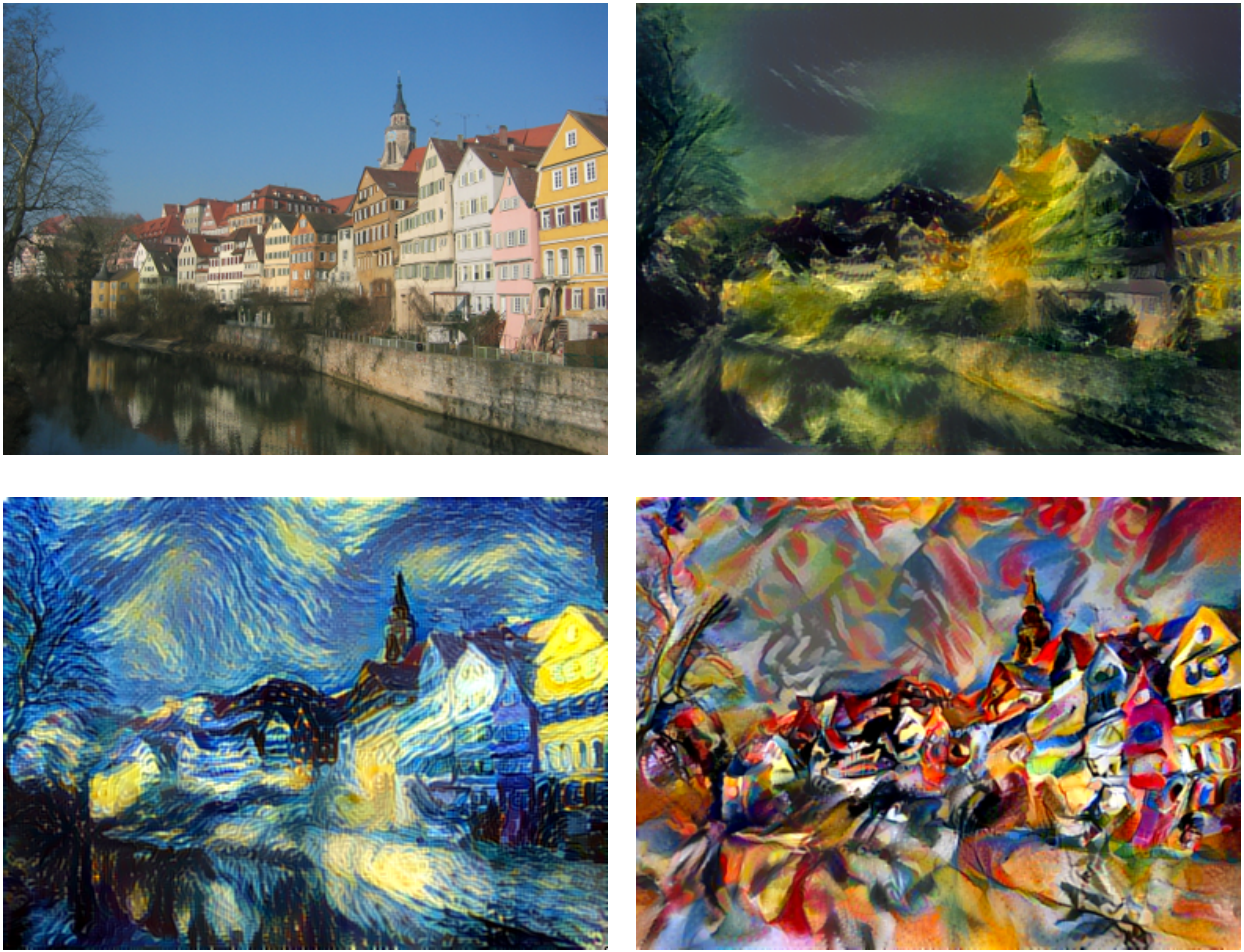
Networks can disentangle style and content, recombinations lead to artistic pieces.



[Gatys et al.]

Neural style transfer

The style transfer works on many different styles.



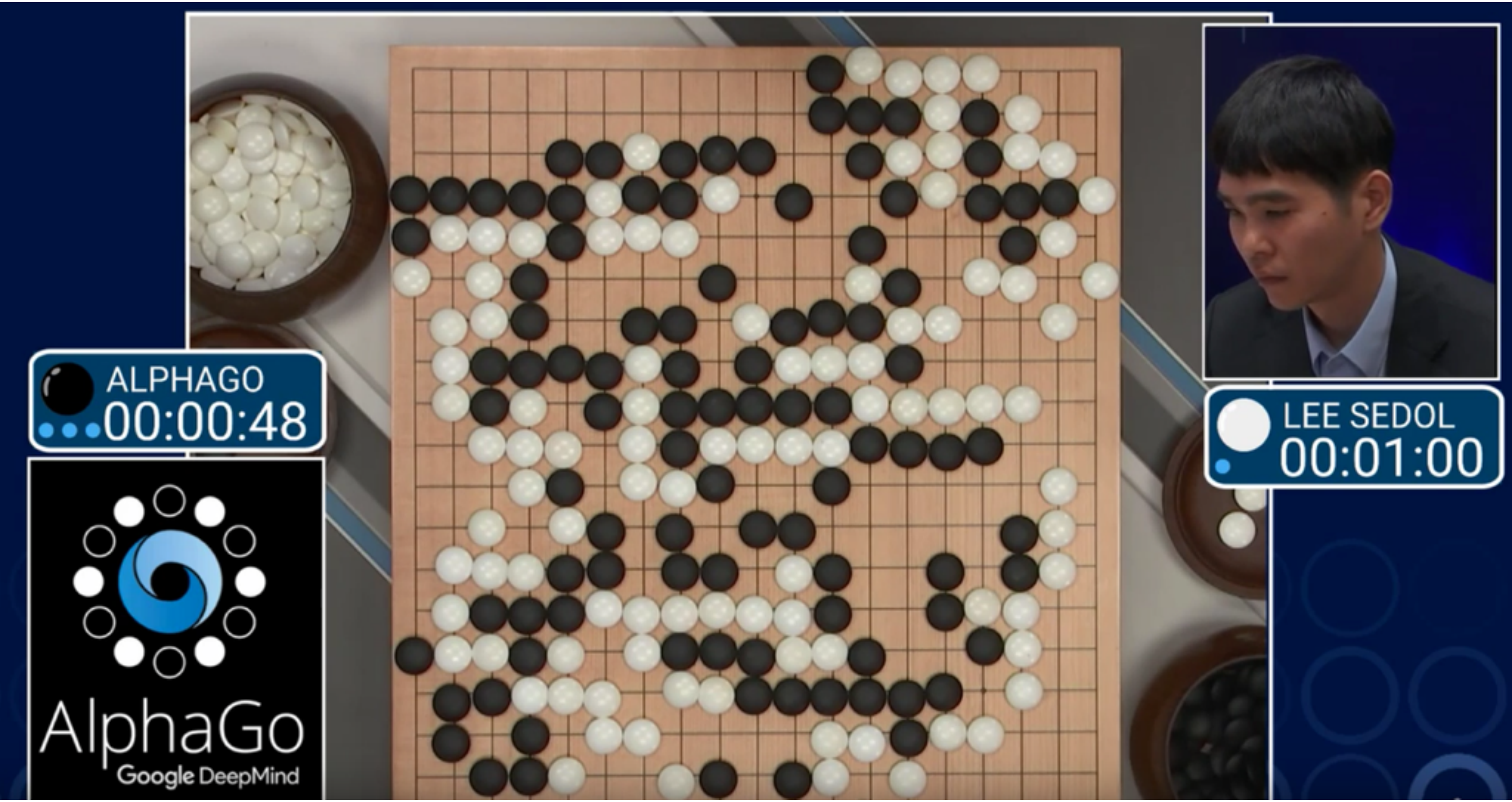
Understanding neural networks

By optimising the input image instead of the network weights, the learned patterns are revealed.

Deep Dreams

AlphaGo

Training NNs requires well defined environments, such as the strict rules of a GO game.



Dangerous or of merit?

- Social impact
  - Replaces repetitive jobs
  - Creates new jobs
- Dangers
  - Fake news
  - Bias of data
  - General artificial intelligence (GAI)

> Open Letter on Artificial Intelligence
>
> - Stephen Hawking, Elon Musk, and dozens of artificial intelligence experts

Virtual dubbing

[www.synthesia.io]

Virtual dubbing

[www.synthesia.io]

Homework 5

Building a neural network to classify fashion items from their pictures.