

RGL Realistic Graphics Lab

CVL Computer Vision Lab



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

Neural networks in practice

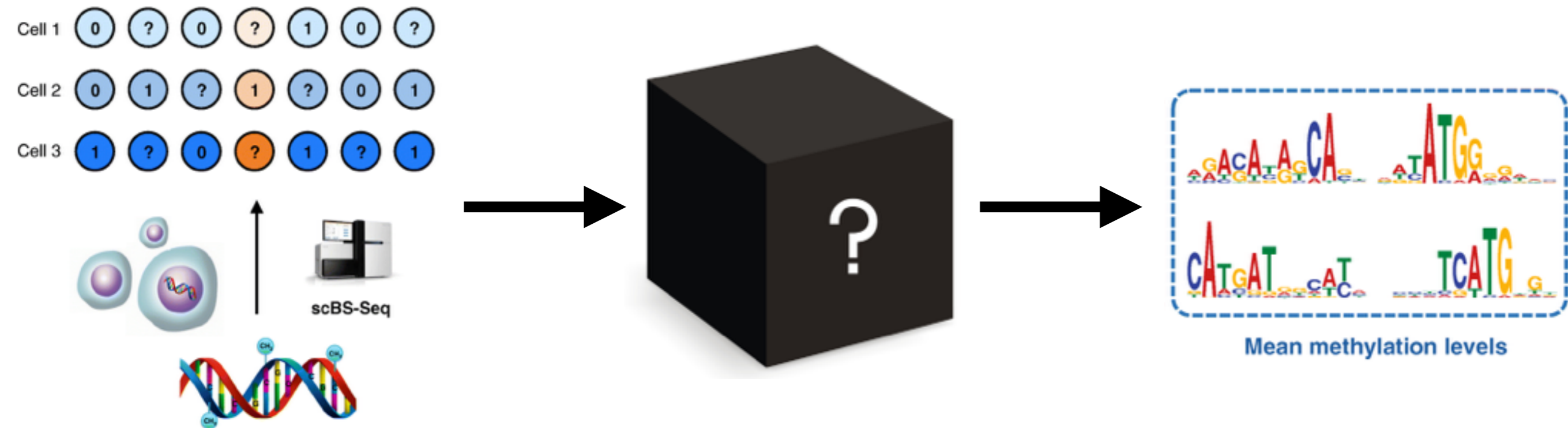
Neural networks (NNS) are a widely used — a tool to learn patterns from large databases.



Computer vision, image segmentation

Neural networks in practice

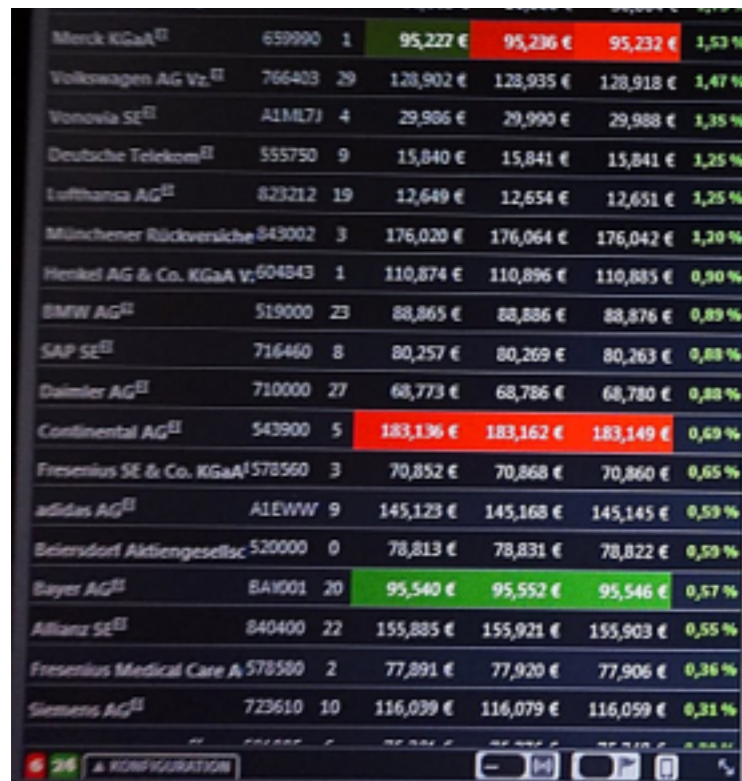
Neural networks (NNS) are a widely used — a tool to learn patterns from large databases.



Biology, DNA analysis

Neural networks in practice

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



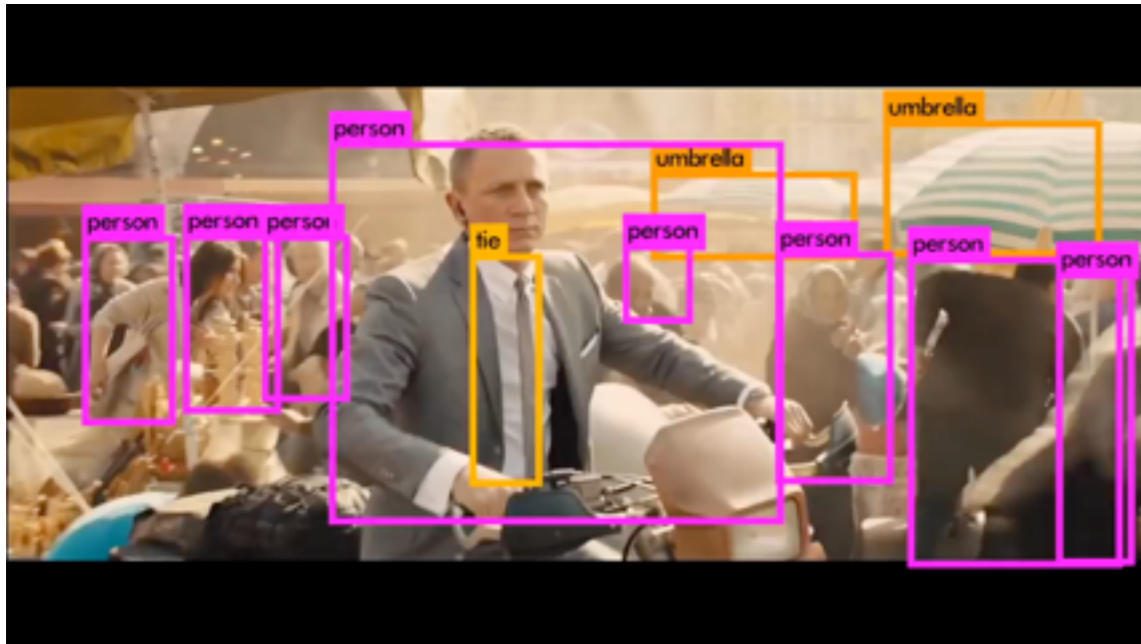
| | | | | | | |
|---|--------|----|-----------|-----------|-----------|--------|
| Merck KGaA ^{EQ} | 659900 | 1 | 95,227 € | 95,236 € | 95,232 € | 1,53 % |
| Volkswagen AG Vz ^{EQ} | 766403 | 29 | 128,902 € | 128,935 € | 128,918 € | 1,47 % |
| Vonovia SE ^{EQ} | A1ML7J | 4 | 29,986 € | 29,990 € | 29,988 € | 1,35 % |
| Deutsche Telekom ^{EQ} | 555750 | 9 | 15,840 € | 15,841 € | 15,841 € | 1,25 % |
| Lufthansa AG ^{EQ} | 823212 | 19 | 12,649 € | 12,654 € | 12,651 € | 1,25 % |
| Münchener Rückversicherungs-Gesellschaft AG ^{EQ} | 843002 | 3 | 176,020 € | 176,064 € | 176,042 € | 1,20 % |
| Henkel AG & Co. KGaA V ^{EQ} | 604843 | 1 | 110,874 € | 110,896 € | 110,885 € | 0,90 % |
| BMW AG ^{EQ} | 519000 | 23 | 88,865 € | 88,886 € | 88,876 € | 0,89 % |
| SAP SE ^{EQ} | 716460 | 8 | 80,257 € | 80,269 € | 80,263 € | 0,88 % |
| Daimler AG ^{EQ} | 710000 | 27 | 68,773 € | 68,786 € | 68,780 € | 0,88 % |
| Continental AG ^{EQ} | 543900 | 5 | 183,136 € | 183,162 € | 183,149 € | 0,69 % |
| Freemius SE & Co. KGaA ¹ | 578560 | 3 | 70,852 € | 70,868 € | 70,860 € | 0,65 % |
| adidas AG ^{EQ} | A1EWW | 9 | 145,123 € | 145,168 € | 145,145 € | 0,59 % |
| Beiersdorf Aktiengesellschaft ^{EQ} | 520000 | 0 | 78,813 € | 78,831 € | 78,822 € | 0,59 % |
| Bayer AG ^{EQ} | 8A1001 | 20 | 95,540 € | 95,552 € | 95,546 € | 0,57 % |
| Allianz SE ^{EQ} | 840400 | 22 | 155,885 € | 155,921 € | 155,903 € | 0,55 % |
| Freemius Medical Care A ^{EQ} | 578580 | 2 | 77,891 € | 77,920 € | 77,906 € | 0,36 % |
| Siemens AG ^{EQ} | 723610 | 10 | 116,039 € | 116,079 € | 116,059 € | 0,31 % |



Finance and risk management

Neural networks and visual computing

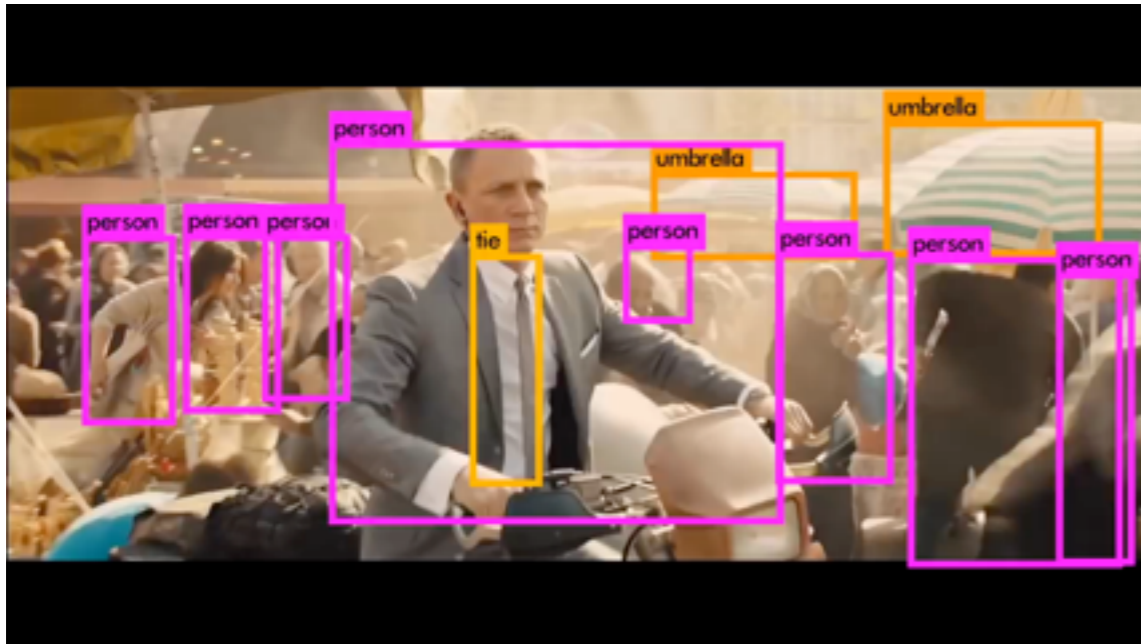
Convolutional NNs are particularly suited to extract semantic information from images.



Object and person detection [Redmon 2016]

Neural networks and visual computing

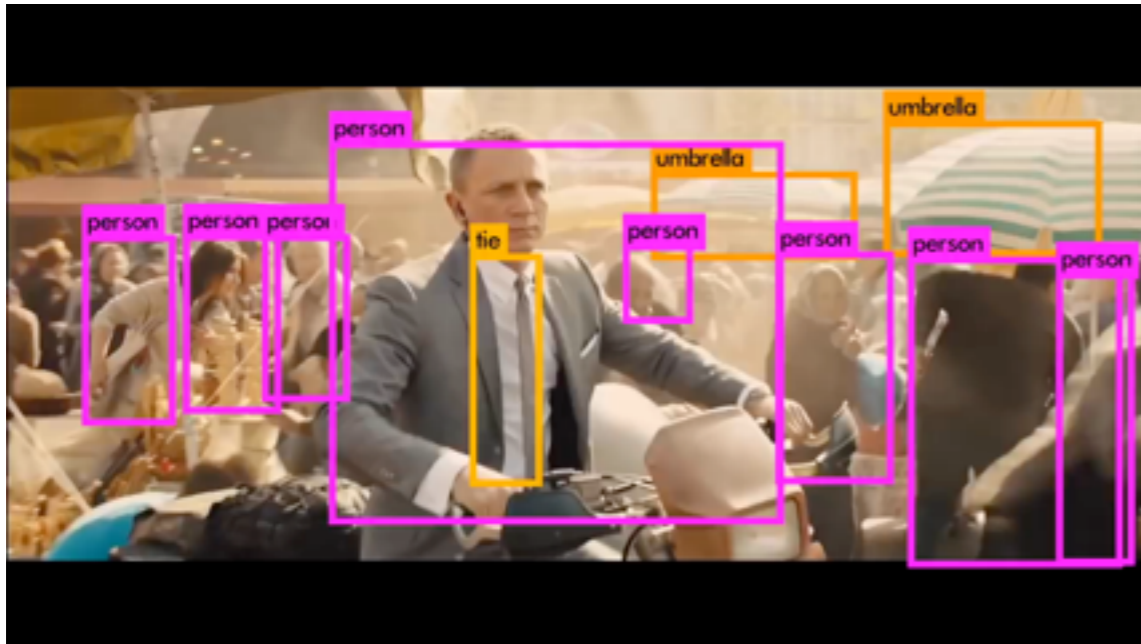
Convolutional NNs are particularly suited to extract semantic information from images.



Object and person detection [Redmon 2016]

Neural networks and visual computing

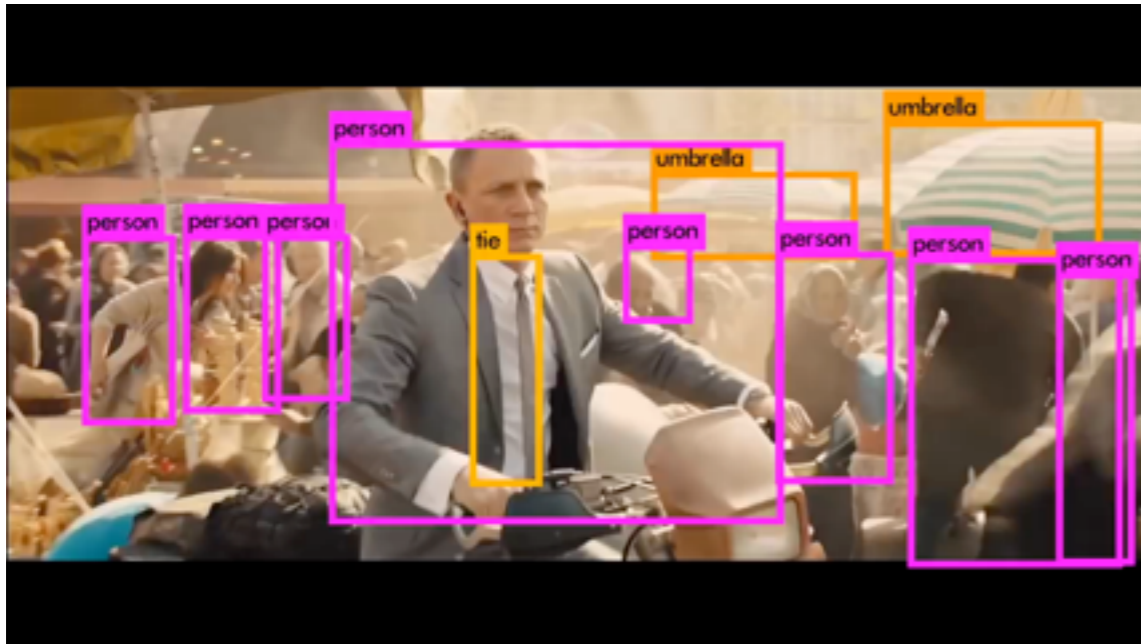
Convolutional NNs are particularly suited to extract semantic information from images.



Object and person detection [Redmon 2016]

Neural networks and visual computing

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



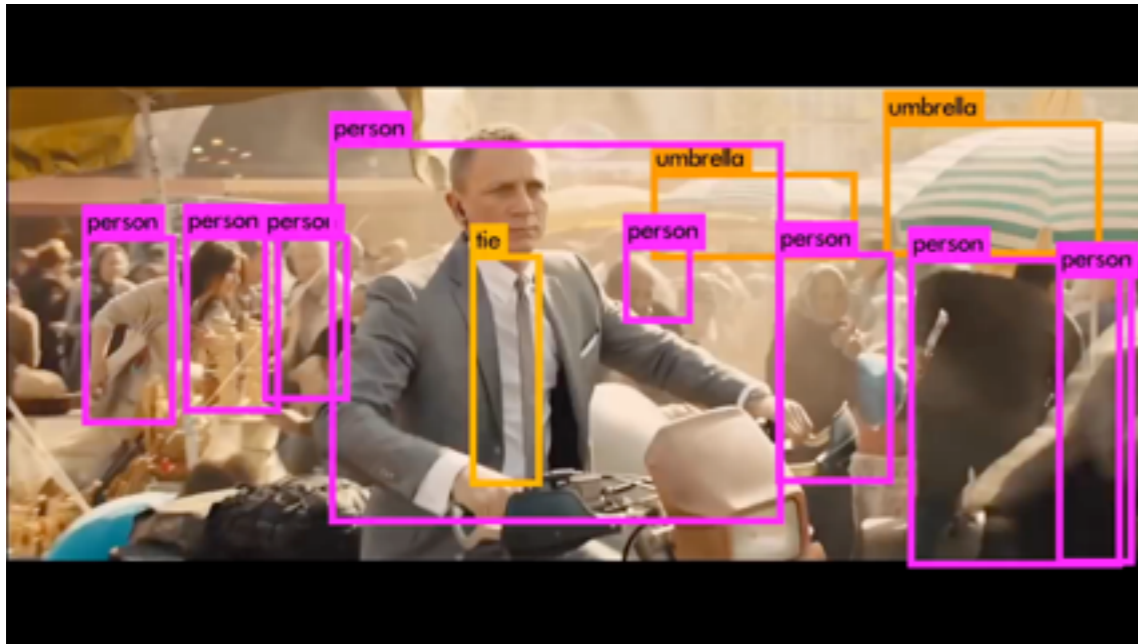
Object and person detection [Redmon 2016]



3D Human pose estimation [Mehta 2017]

Neural networks and visual computing

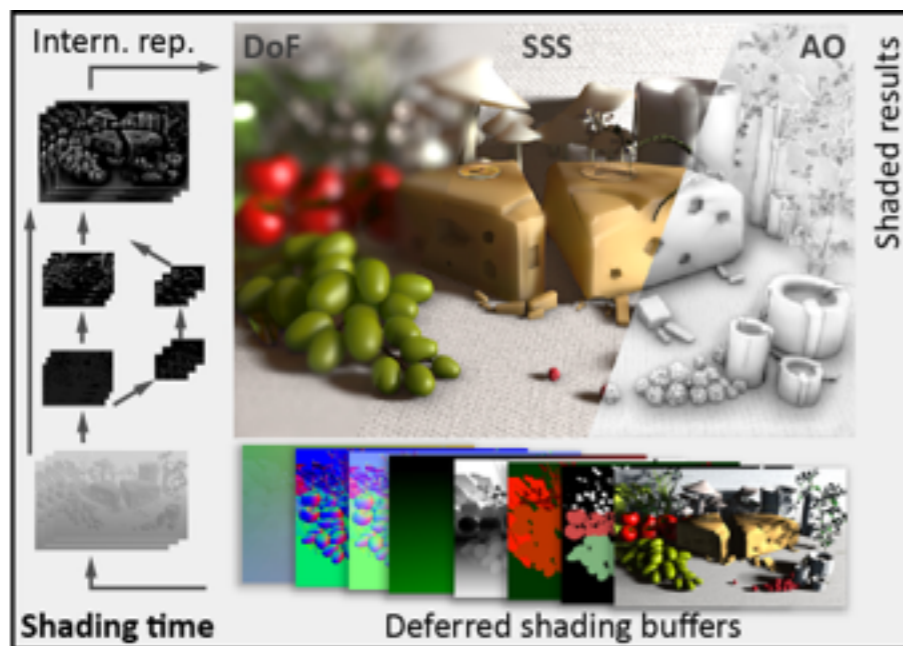
Convolutional NNs are particularly suited to extract semantic information from images.



Object and person detection [Redmon 2016]



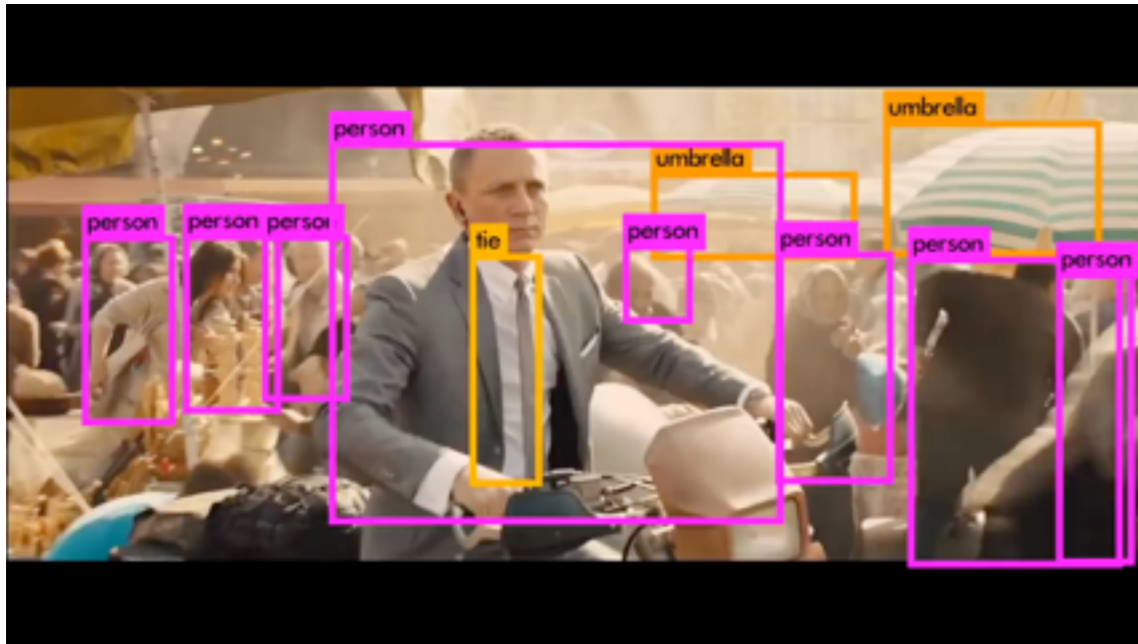
3D Human pose estimation [Mehta 2017]



Computer Graphics, rendering [Nalbach 2017]

Neural networks and visual computing

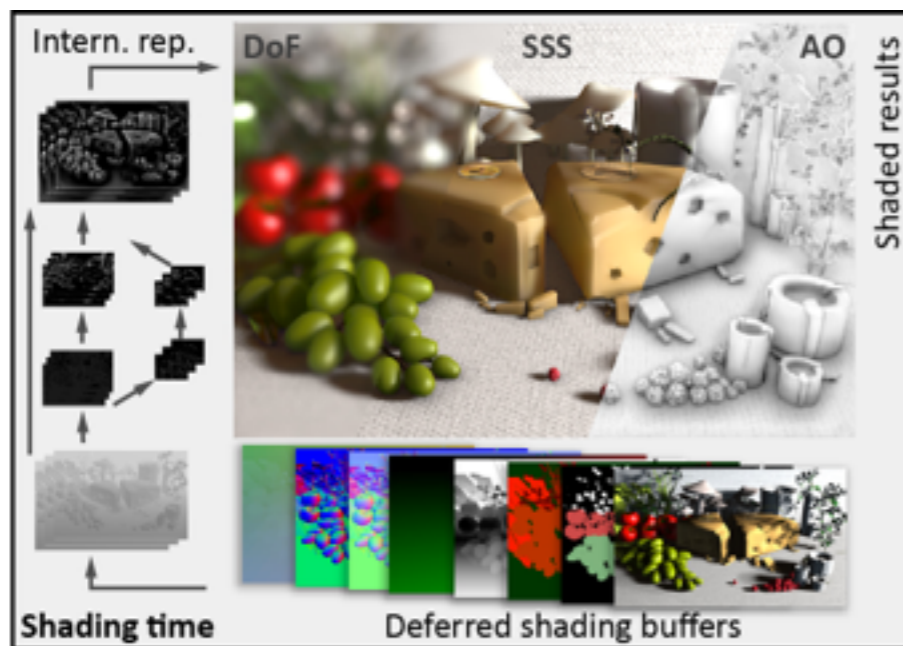
Convolutional NNs are particularly suited to extract semantic information from images.



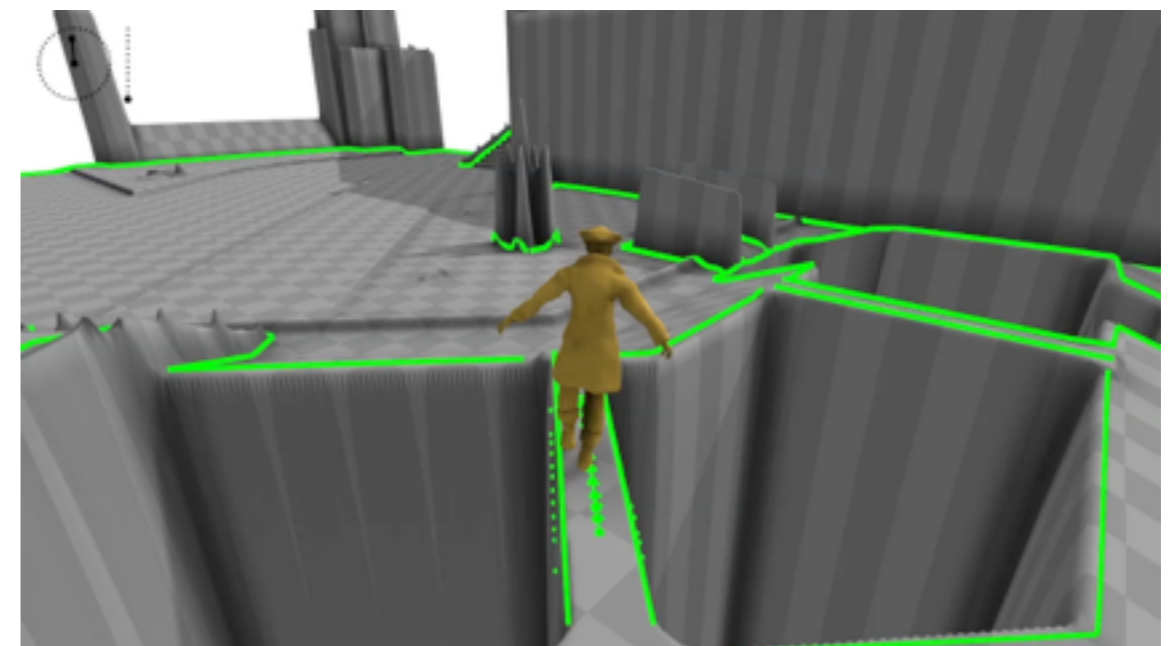
Object and person detection [Redmon 2016]



3D Human pose estimation [Mehta 2017]



Computer Graphics, rendering [Nalbach 2017]



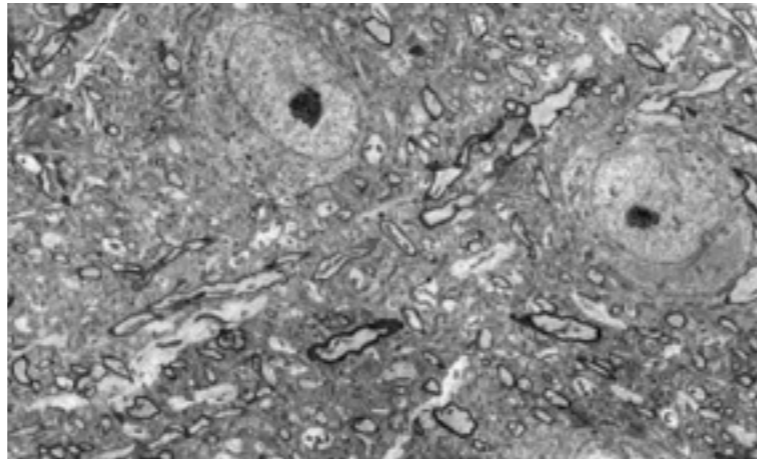
Animation, character control

Biological neural network

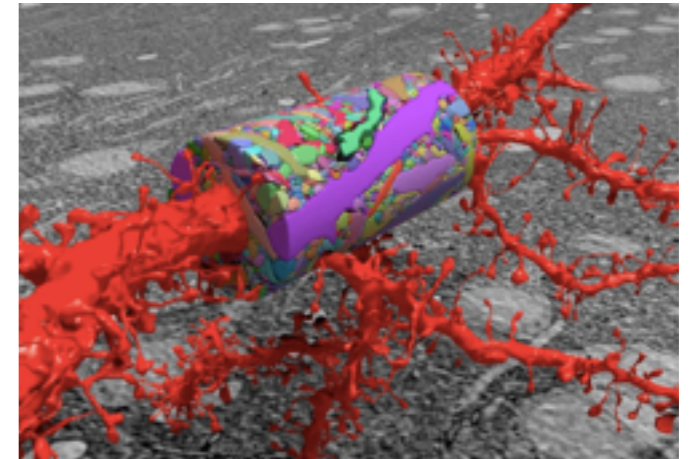
The name, but also the network structure, is inspired by our understanding of real brains.



Macroscopic scale
(human brain)



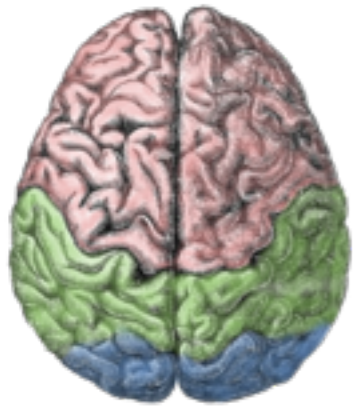
Microscopic slice
(neocortex)



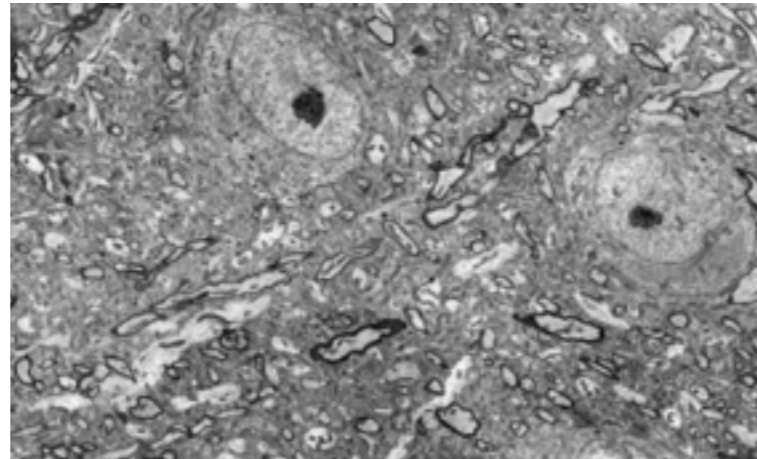
3D reconstruction
(dendrite and surrounding)

Biological neural network

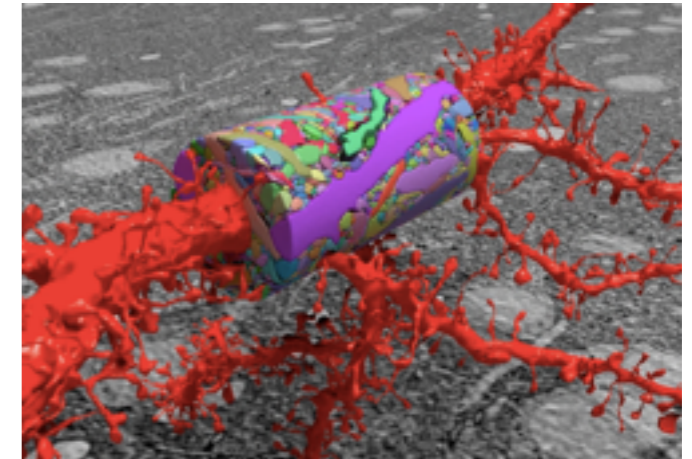
The name, but also the network structure, is inspired by our understanding of real brains.



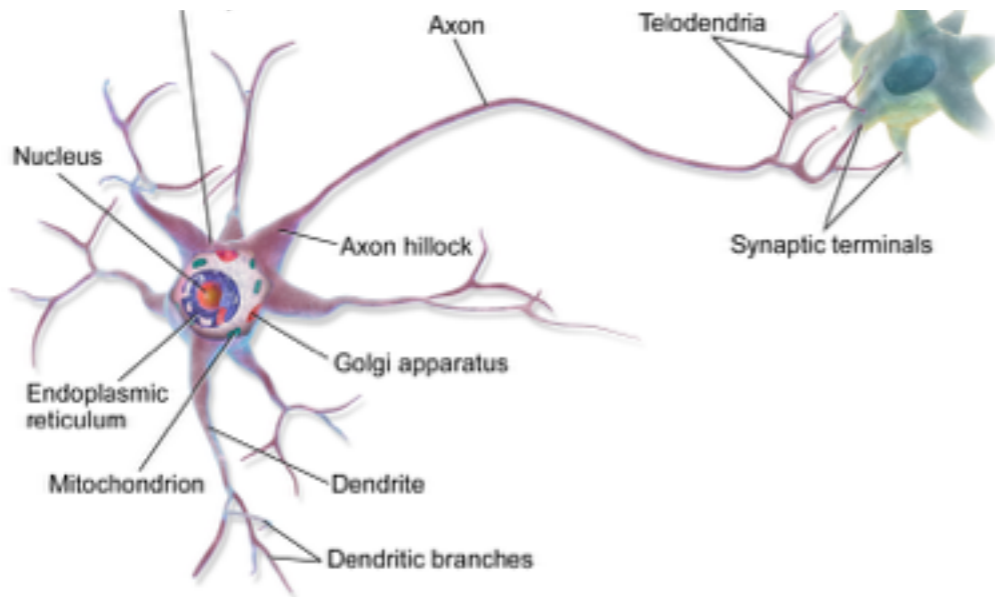
Macroscopic scale
(human brain)



Microscopic slice
(neocortex)



3D reconstruction
(dendrite and surrounding)



Biological model

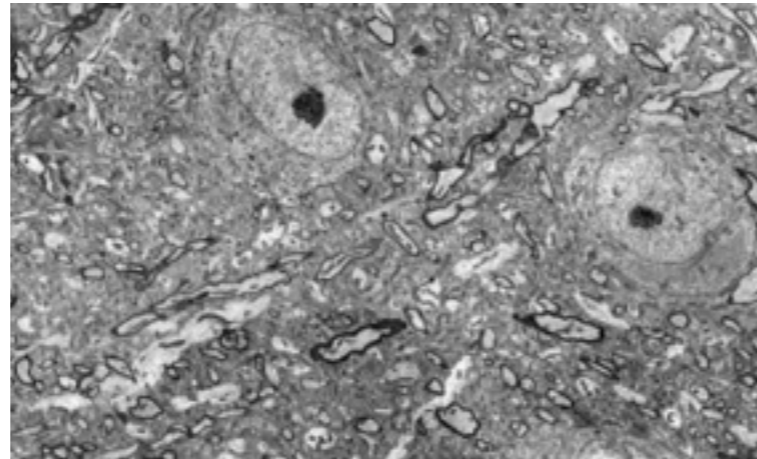
(100 billion neurons, 100 trillion connections)

Biological neural network

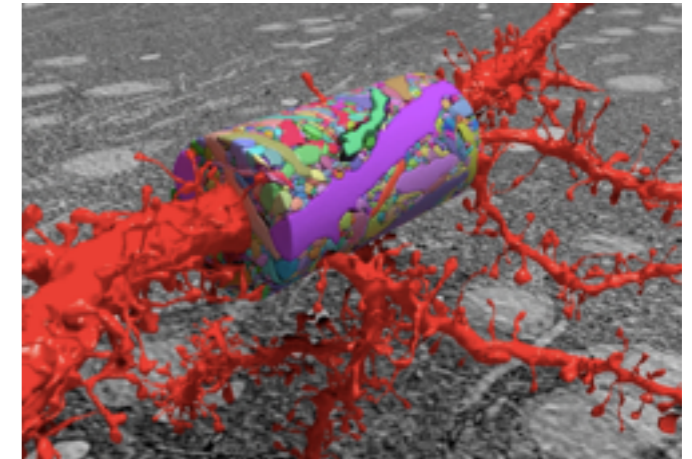
The name, but also the network structure, is inspired by our understanding of real brains.



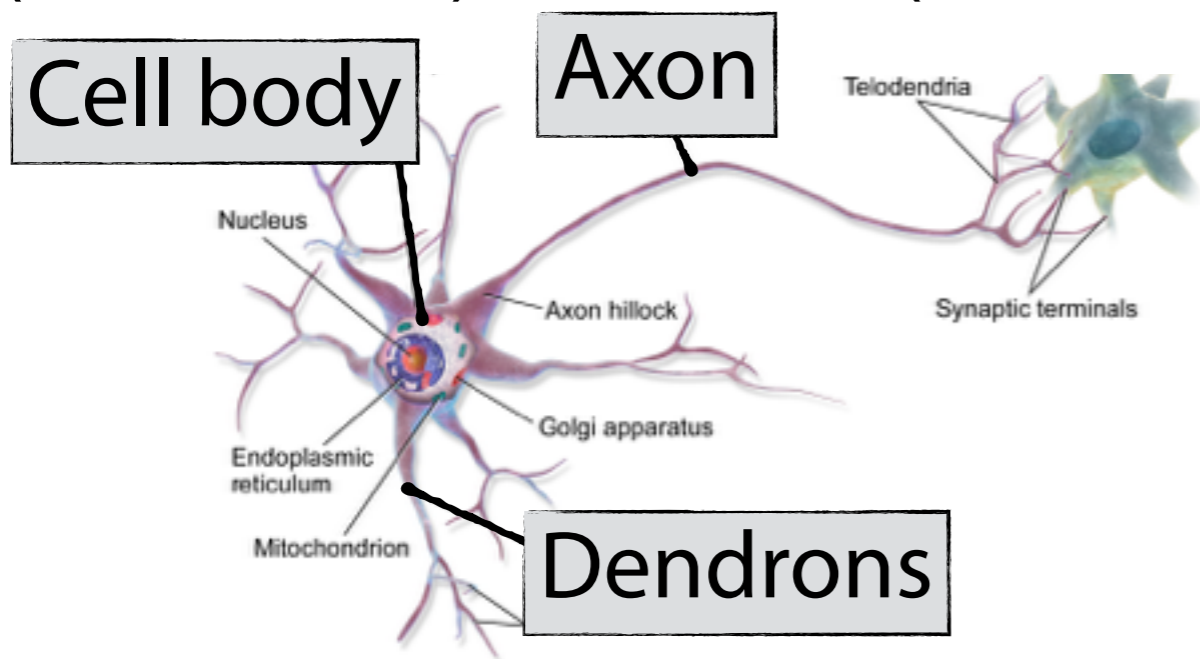
Macroscopic scale
(human brain)



Microscopic slice
(neocortex)



3D reconstruction
(dendrite and surrounding)



Biological model

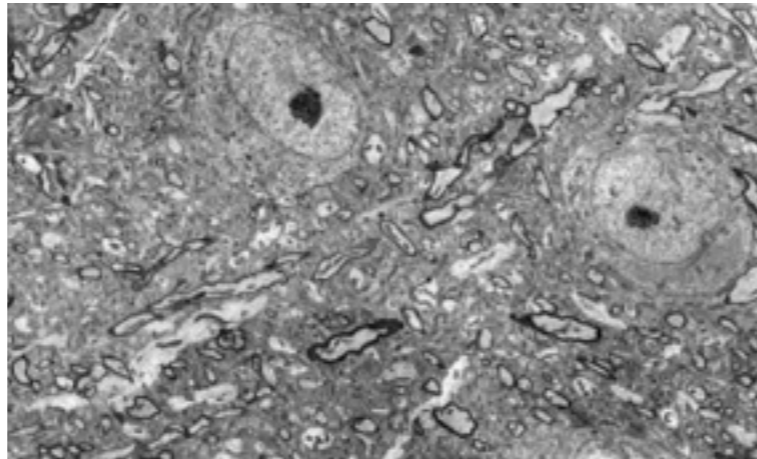
(100 billion neurons, 100 trillion connections)

Biological neural network

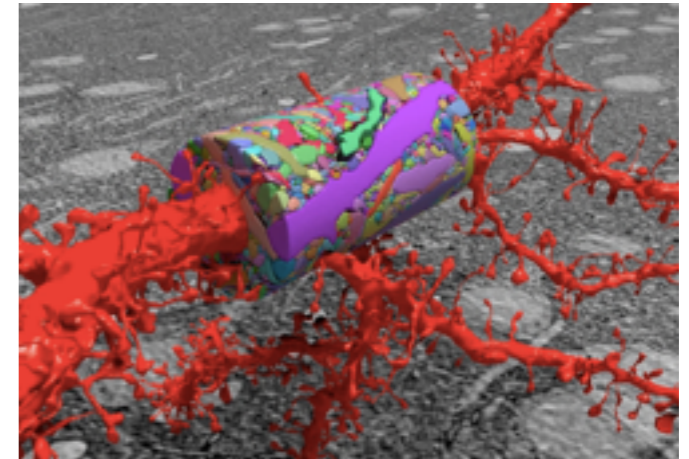
The name, but also the network structure, is inspired by our understanding of real brains.



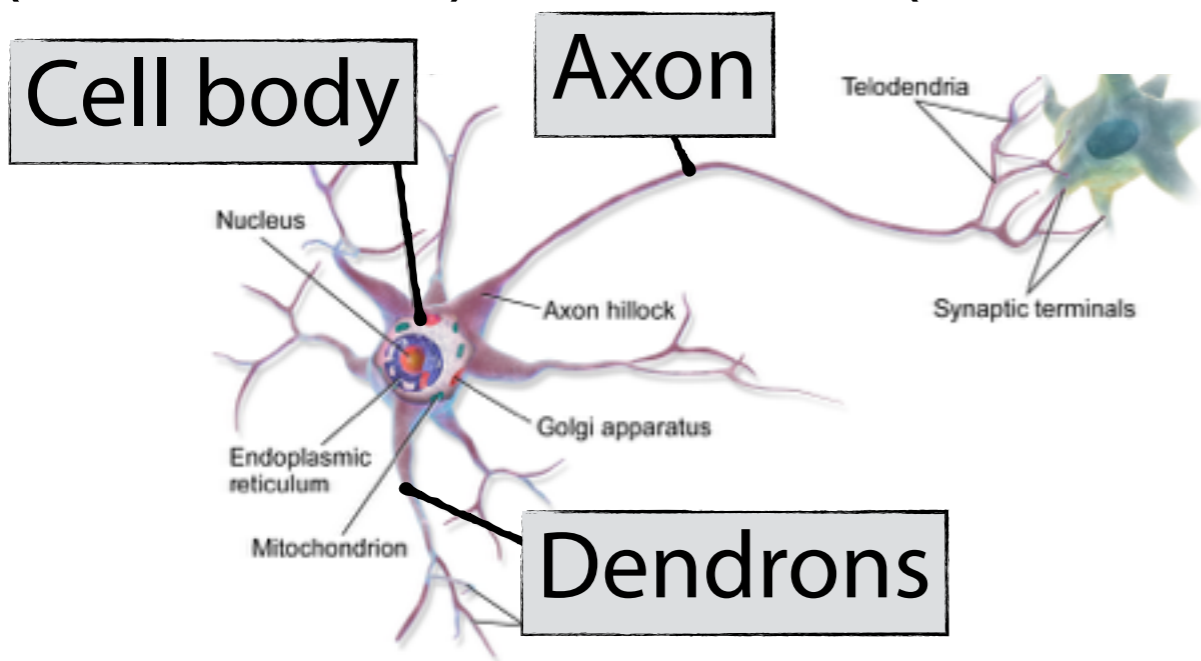
Macroscopic scale
(human brain)



Microscopic slice
(neocortex)

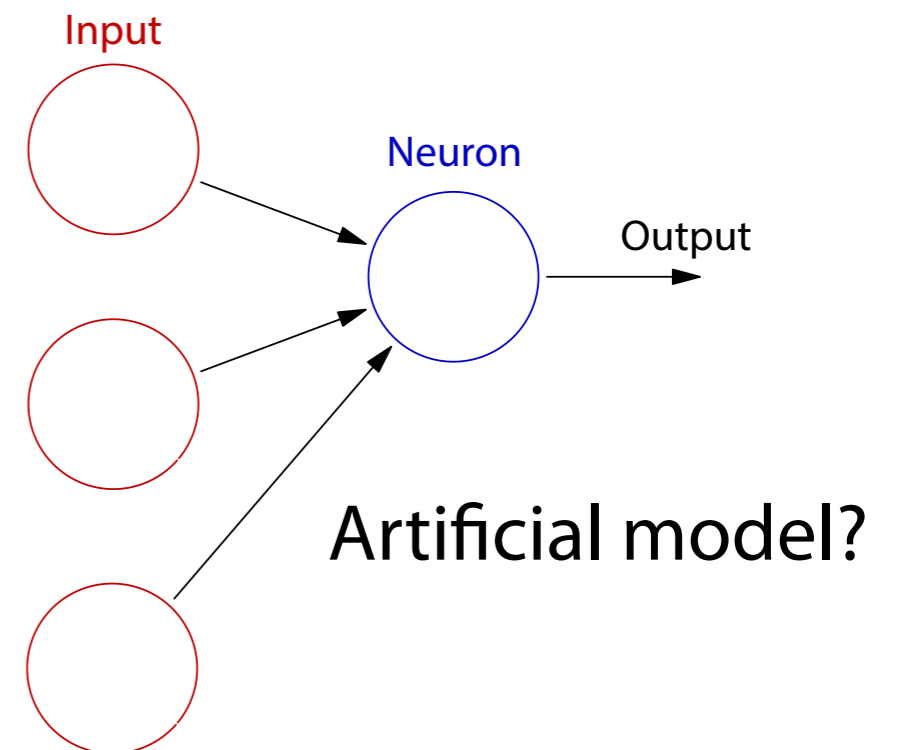


3D reconstruction
(dendrite and surrounding)



Biological model

(100 billion neurons, 100 trillion connections)

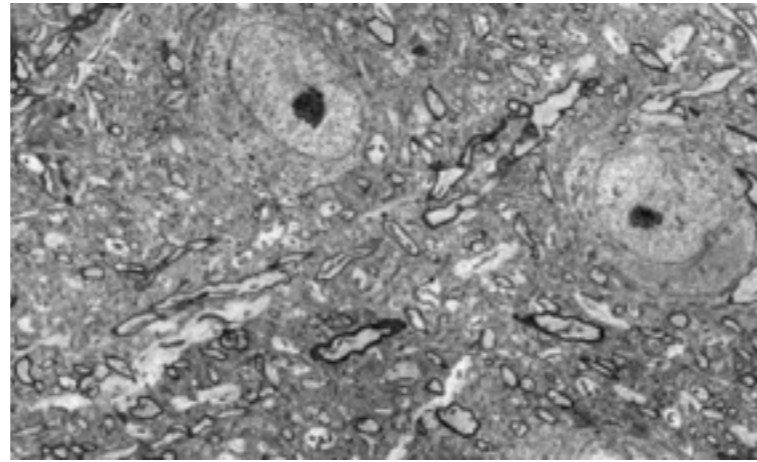


Biological neural network

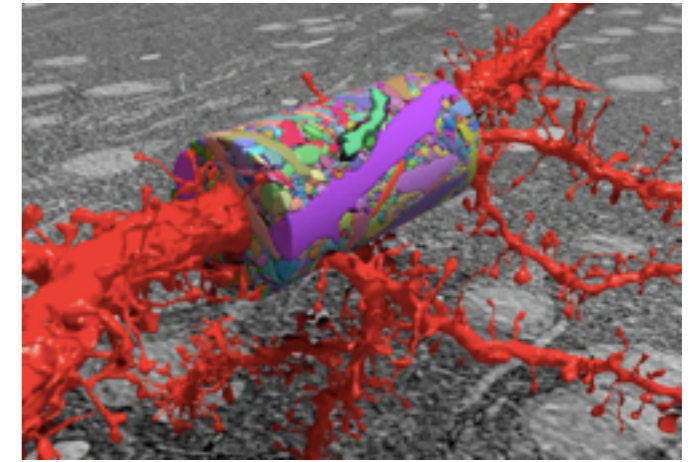
The name, but also the network structure, is inspired by our understanding of real brains.



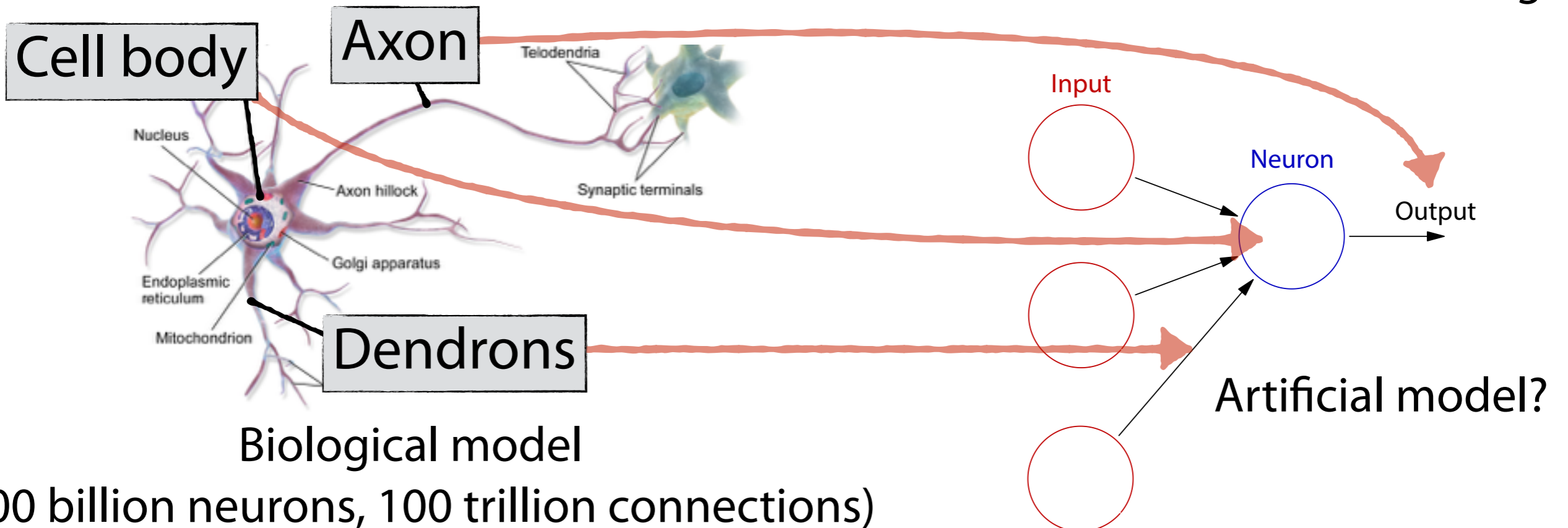
Macroscopic scale
(human brain)



Microscopic slice
(neocortex)



3D reconstruction
(dendrite and surrounding)

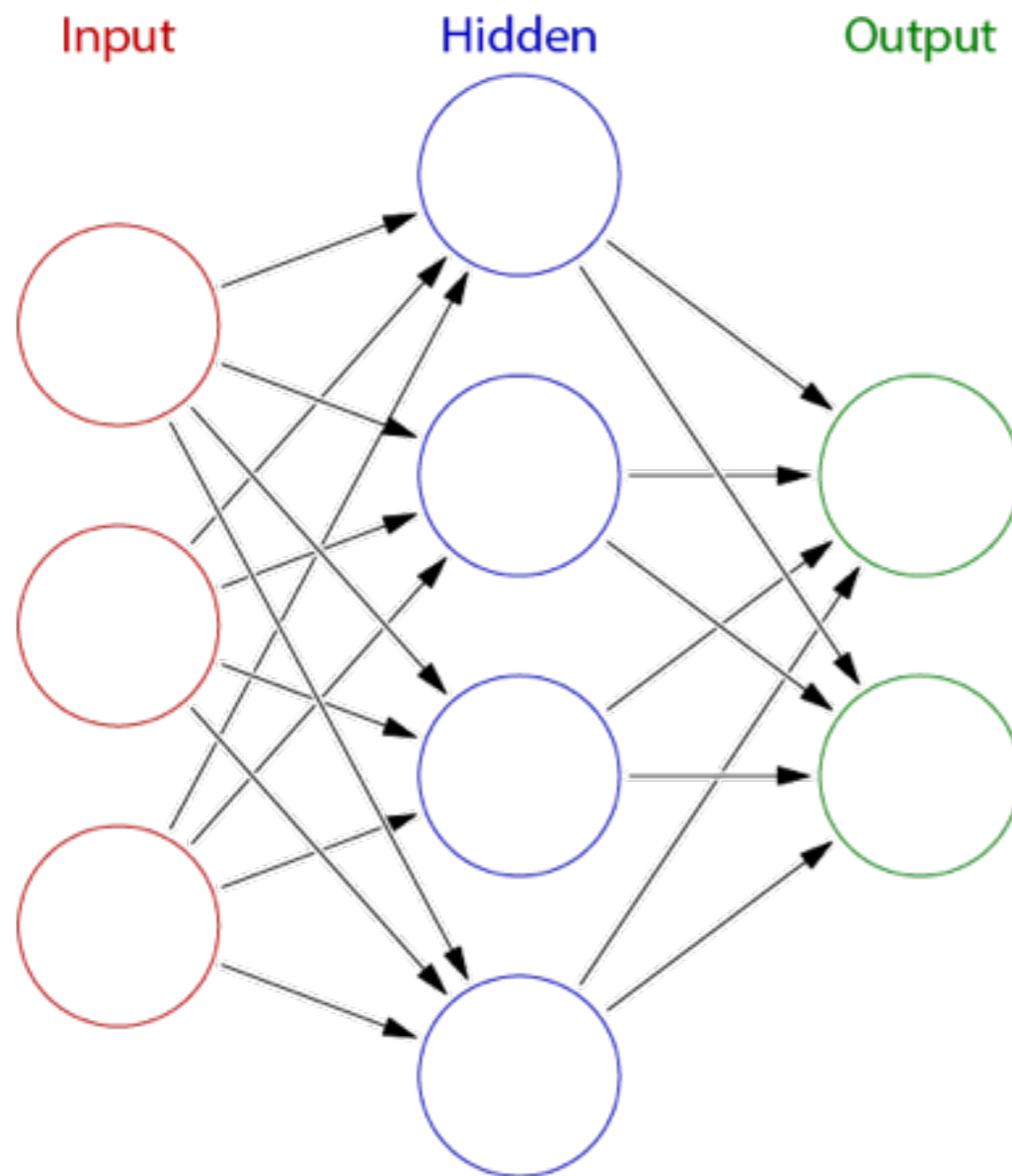


Artificial neural network



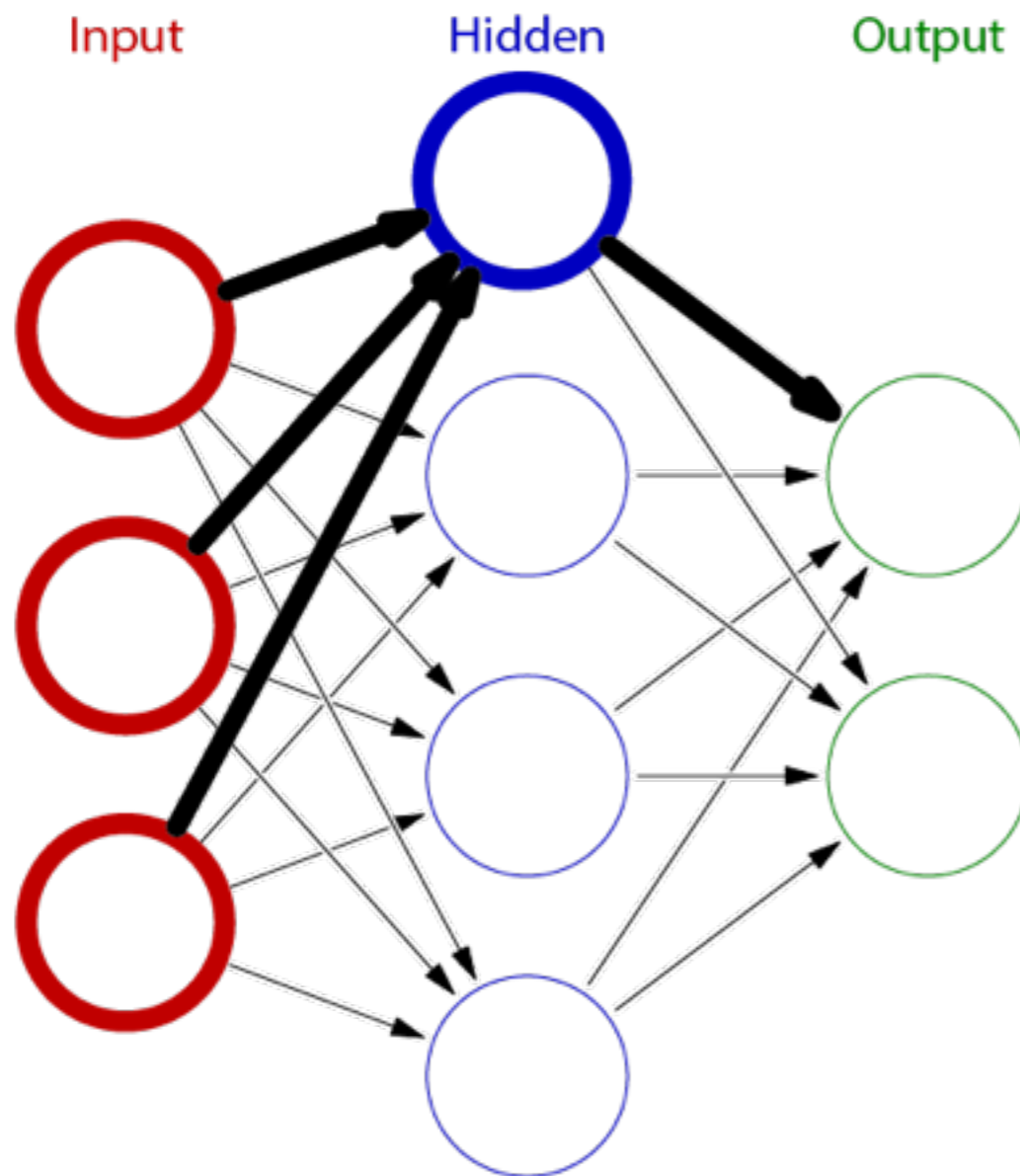
Artificial neural network — a graph

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



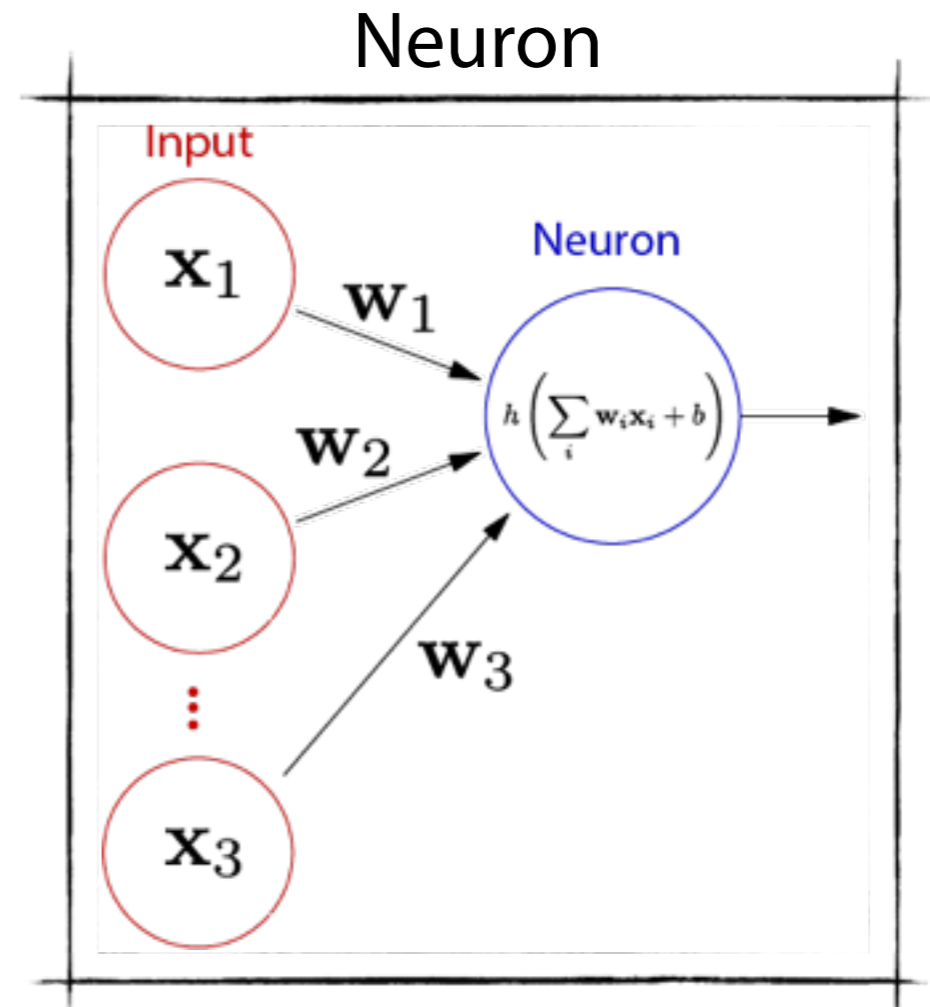
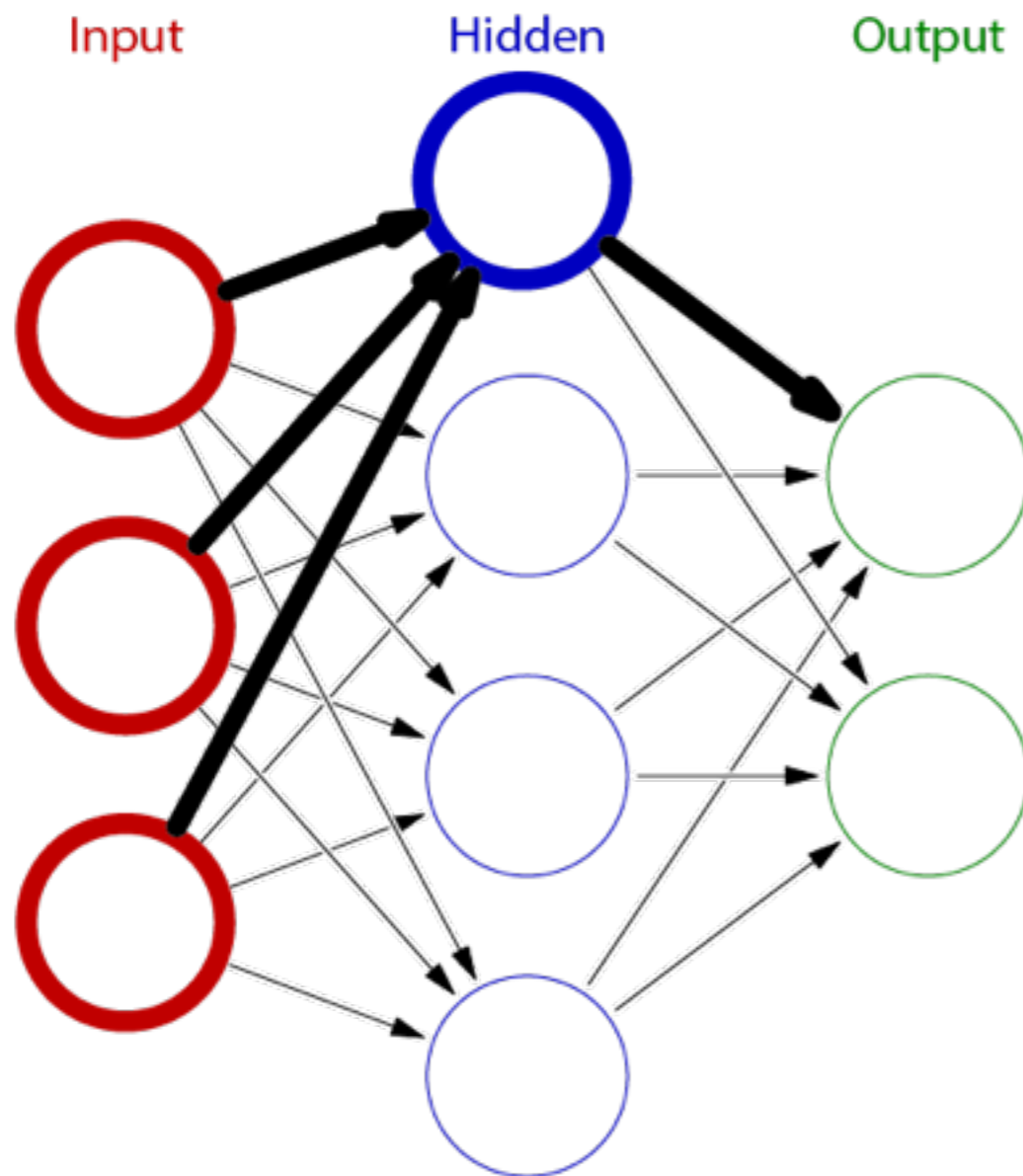
Artificial neural network — a graph

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



Artificial neural network — a graph

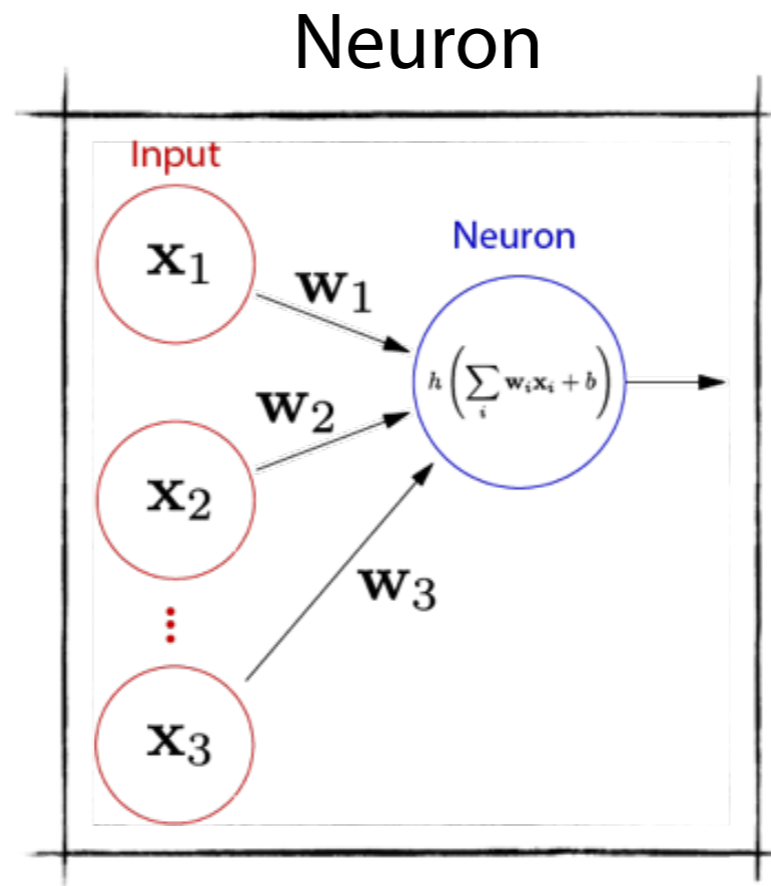
NNs are composed of simple primitives, neurons with multiple inputs and a single output.



- x_i input
- h activation function
- w_i weights
- b bias

Artificial neural network — building blocks

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

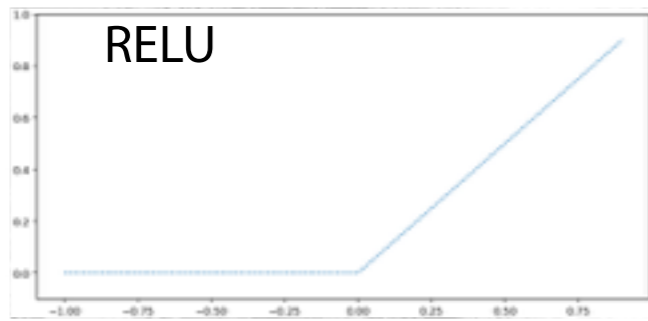
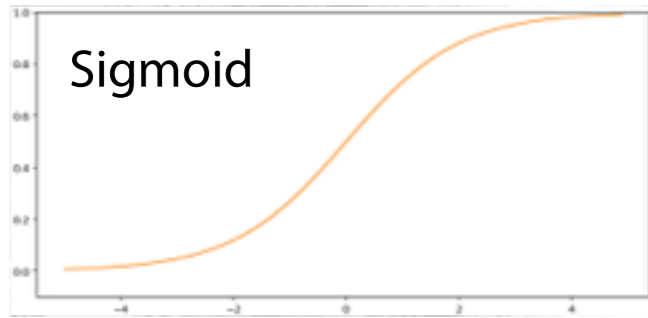


$$h \left(\sum_i w_i x_i + b \right)$$

Artificial neural network — building blocks

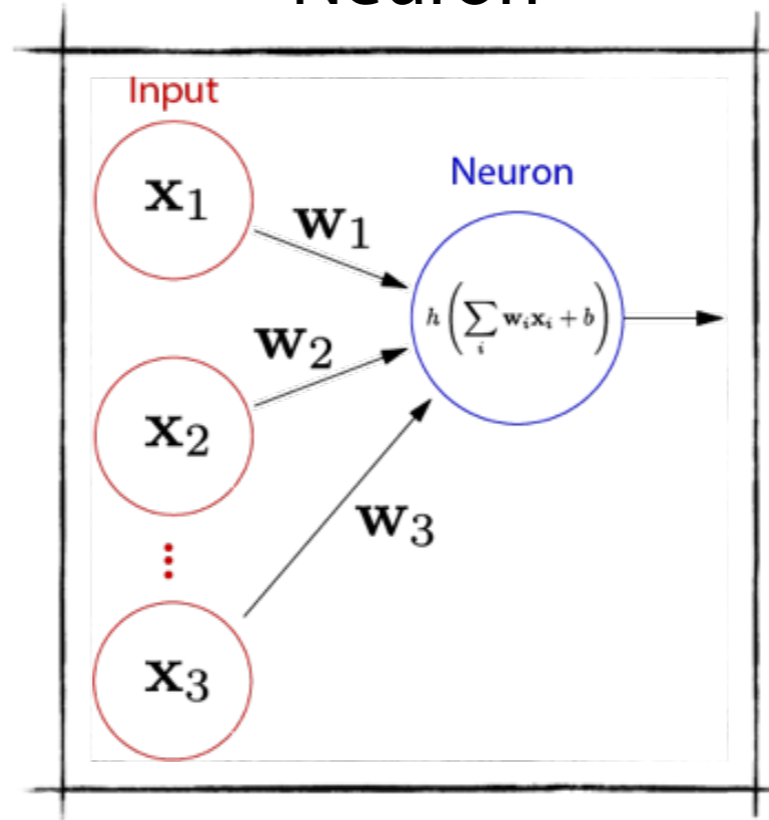
neuron: affine map + activation function, neuron w/o activation function = linear map

Activation function



$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$
$$\text{relu}(x) = \max\{0, x\}$$

Neuron

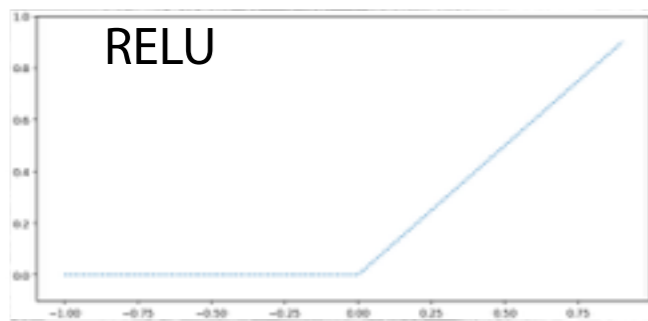
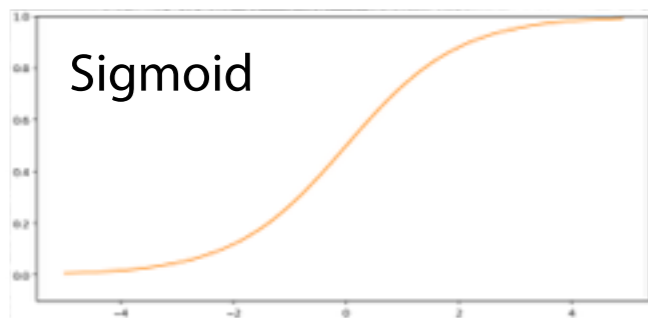


$$h\left(\sum_i w_i x_i + b\right)$$

Artificial neural network — building blocks

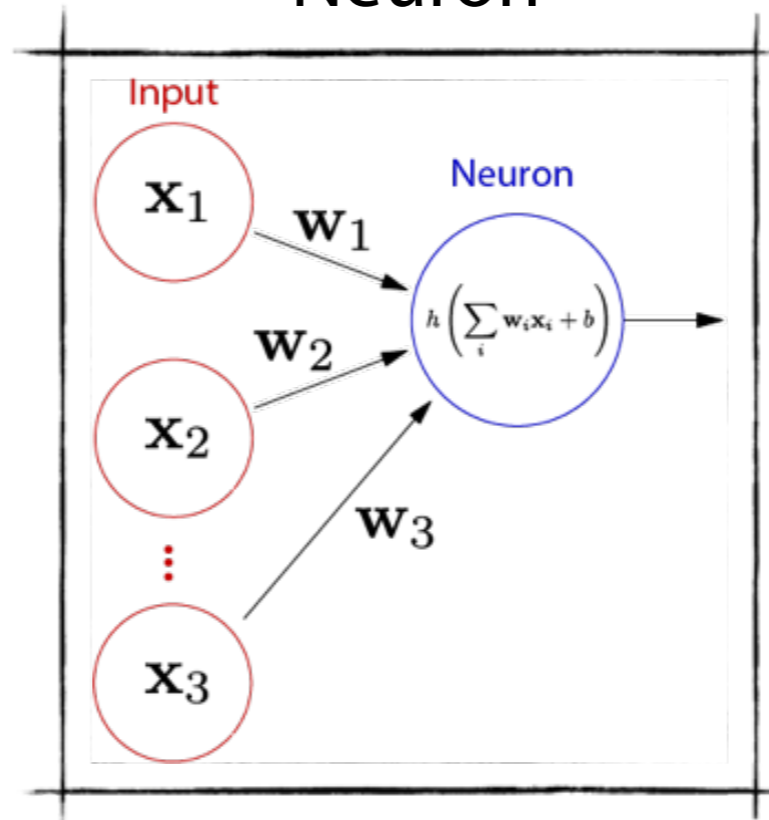
neuron: affine map + activation function, neuron w/o activation function = linear map

Activation function



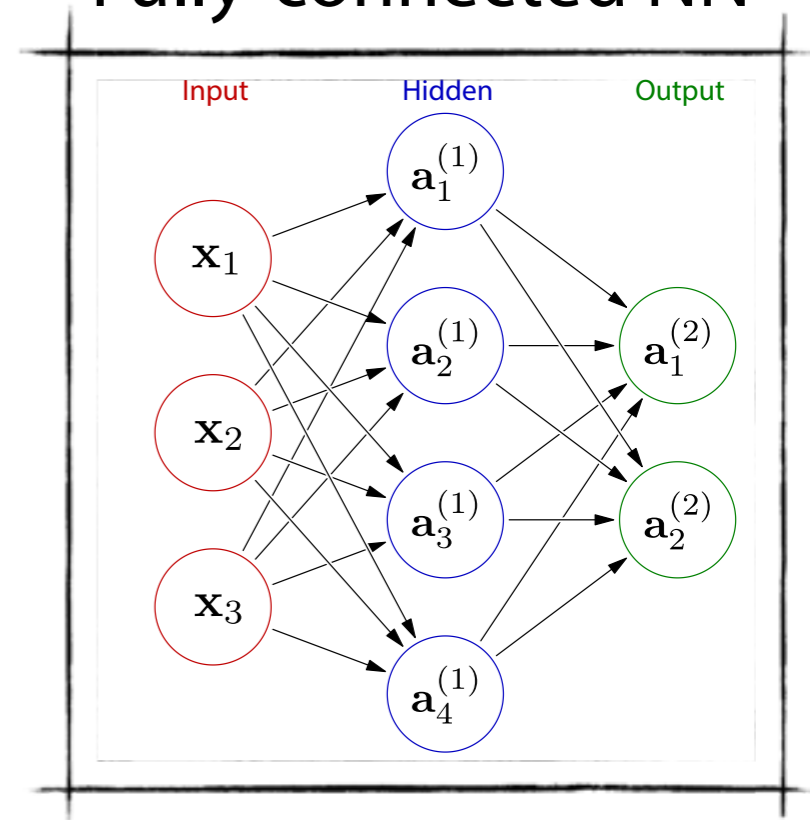
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$
$$\text{relu}(x) = \max\{0, x\}$$

Neuron



$$h\left(\sum_i w_i x_i + b\right)$$

Fully-connected NN

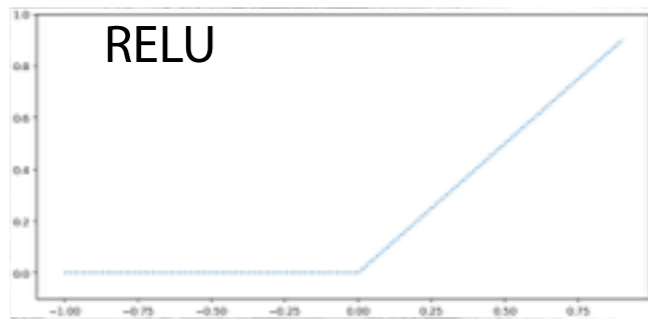
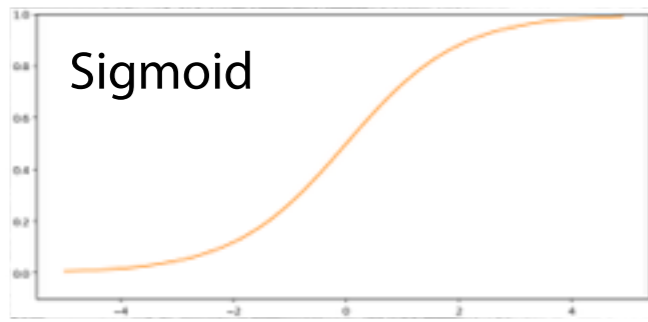


$$\mathbf{a}_1^{(1)} = h\left(\mathbf{w}_{1,1}^{(1)}x_1 + \mathbf{w}_{1,2}^{(1)}x_2 + \mathbf{w}_{1,3}^{(1)}x_3 + b_1^{(1)}\right)$$
$$\mathbf{a}_2^{(1)} = h\left(\mathbf{w}_{2,1}^{(1)}x_1 + \mathbf{w}_{2,2}^{(1)}x_2 + \mathbf{w}_{2,3}^{(1)}x_3 + b_2^{(1)}\right)$$
$$\mathbf{a}_3^{(1)} = h\left(\mathbf{w}_{3,1}^{(1)}x_1 + \mathbf{w}_{3,2}^{(1)}x_2 + \mathbf{w}_{3,3}^{(1)}x_3 + b_3^{(1)}\right)$$
$$\mathbf{a}_1^{(2)} = \mathbf{w}_{1,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{1,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{1,3}^{(2)}\mathbf{a}_3^{(1)} + b_1^{(2)}$$
$$\mathbf{a}_2^{(2)} = \mathbf{w}_{2,1}^{(2)}\mathbf{a}_1^{(1)} + \mathbf{w}_{2,2}^{(2)}\mathbf{a}_2^{(1)} + \mathbf{w}_{2,3}^{(2)}\mathbf{a}_3^{(1)} + b_2^{(2)}$$

Artificial neural network — building blocks

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

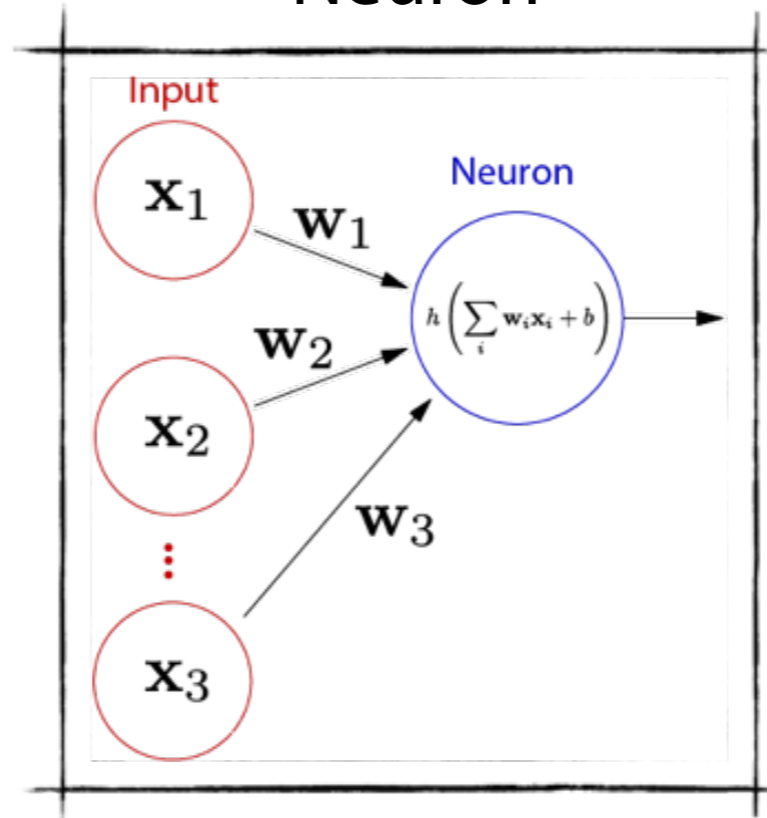
Activation function



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

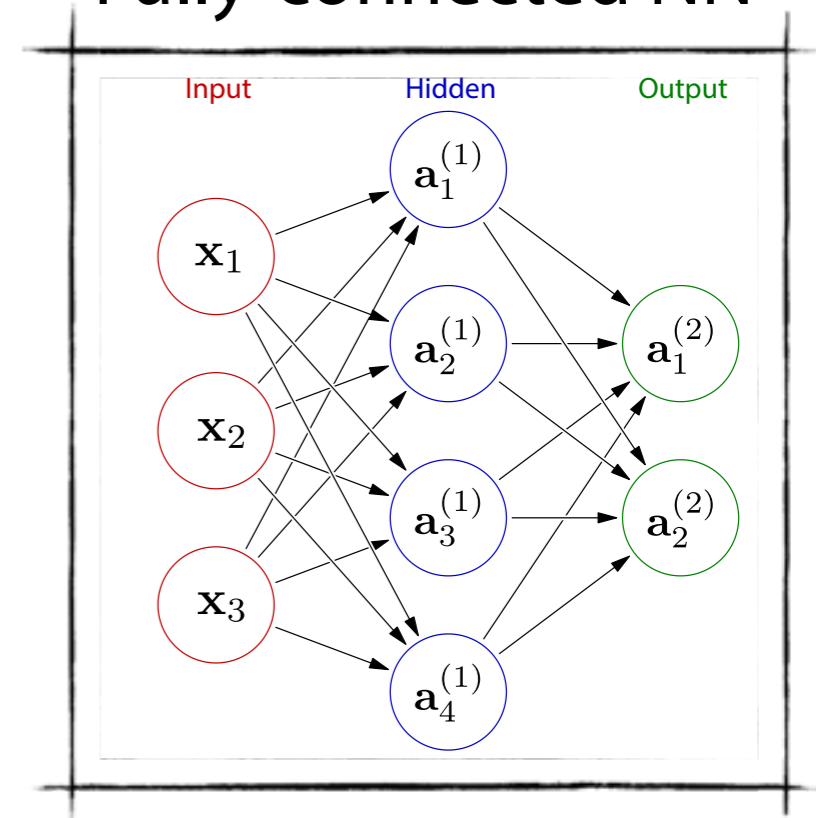
$$\text{relu}(x) = \max\{0, x\}$$

Neuron



$$h \left(\sum_i w_i x_i + b \right)$$

Fully-connected NN



$$a_1^{(1)} = h \left(w_{1,1}^{(1)} x_1 + w_{1,2}^{(1)} x_2 + w_{1,3}^{(1)} x_3 + b_1^{(1)} \right)$$

$$a_2^{(1)} = h \left(w_{2,1}^{(1)} x_1 + w_{2,2}^{(1)} x_2 + w_{2,3}^{(1)} x_3 + b_2^{(1)} \right)$$

$$a_3^{(1)} = h \left(w_{3,1}^{(1)} x_1 + w_{3,2}^{(1)} x_2 + w_{3,3}^{(1)} x_3 + b_3^{(1)} \right)$$

$$a_1^{(2)} = w_{1,1}^{(2)} a_1^{(1)} + w_{1,2}^{(2)} a_2^{(1)} + w_{1,3}^{(2)} a_3^{(1)} + b_1^{(2)}$$

$$a_2^{(2)} = w_{2,1}^{(2)} a_1^{(1)} + w_{2,2}^{(2)} a_2^{(1)} + w_{2,3}^{(2)} a_3^{(1)} + b_2^{(2)}$$

Linear without activation function!

$$\begin{aligned} a_1^{(2)} &= w_{1,1}^{(2)} (w_{1,1}^{(1)} x_1 + w_{1,2}^{(1)} x_2 + w_{1,3}^{(1)} x_3 + b_1^{(1)}) + \dots + b_1^{(2)} \\ &= c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 \end{aligned}$$

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.



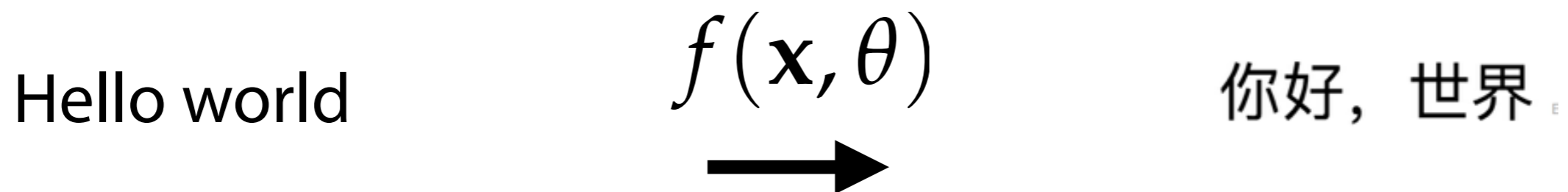
Examples

Artificial neural network — a function

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



Examples



Artificial neural network — a function

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



Examples

Hello world

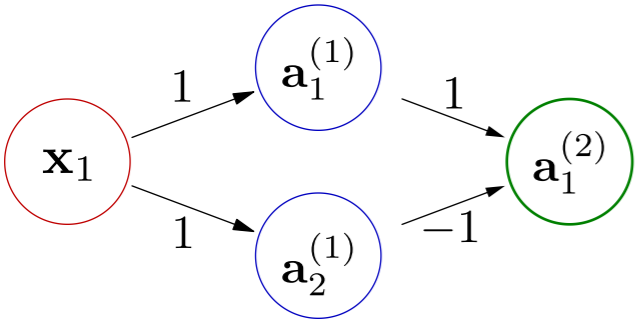
$$f(\mathbf{x}, \theta)$$

你好，世界。



Approximation power — universal

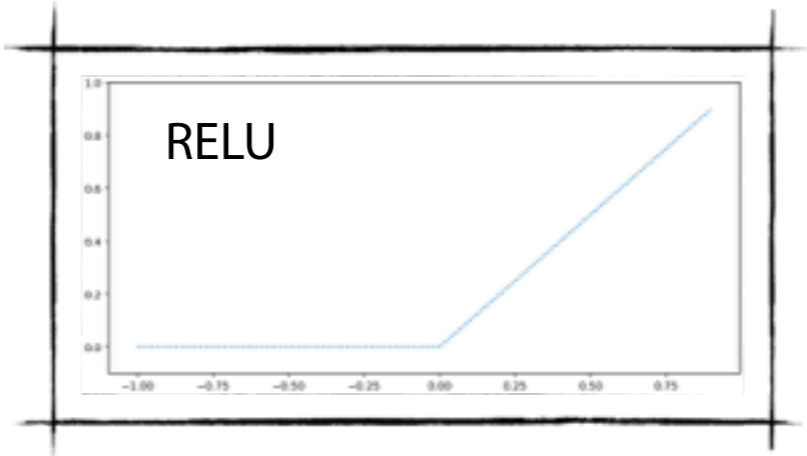
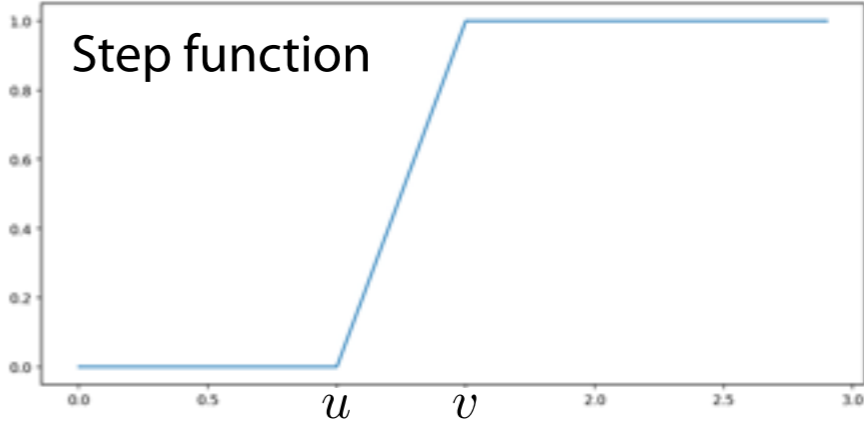
Four neurons can form a box-function, multiple boxes can approximate continuous functions.



$$a_1^{(1)} = \text{relu}(x - u)$$

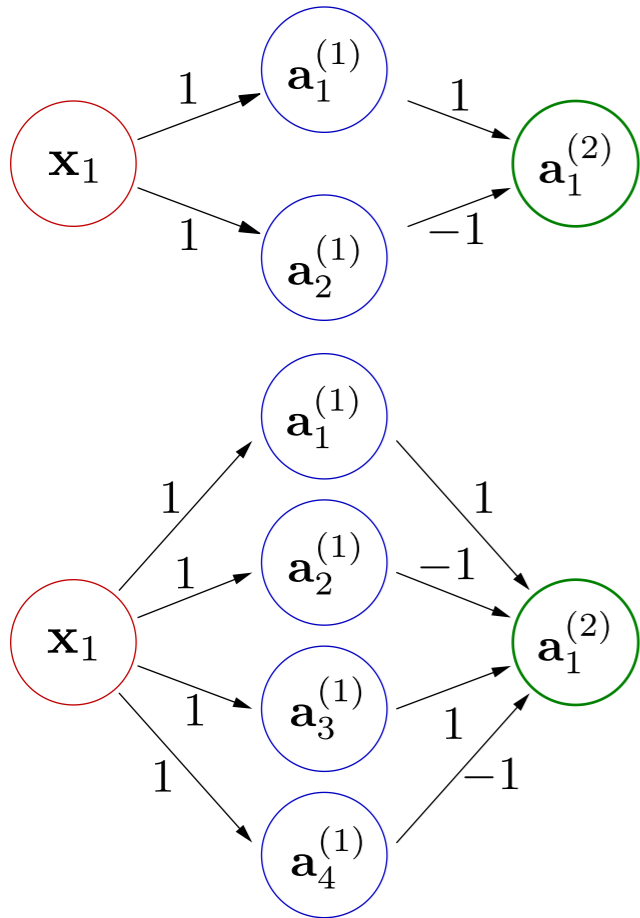
$$a_2^{(1)} = \text{relu}(x - v)$$

$$a_1^{(2)} = a_1^{(1)} - a_2^{(1)}$$



Approximation power — universal

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



$$a_1^{(1)} = \text{relu}(x - u)$$

$$a_2^{(1)} = \text{relu}(x - v)$$

$$a_1^{(2)} = a_1^{(1)} - a_2^{(1)}$$

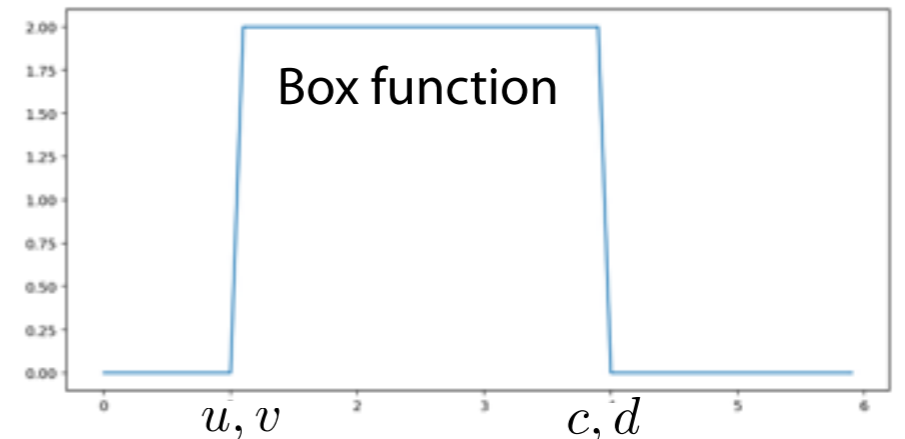
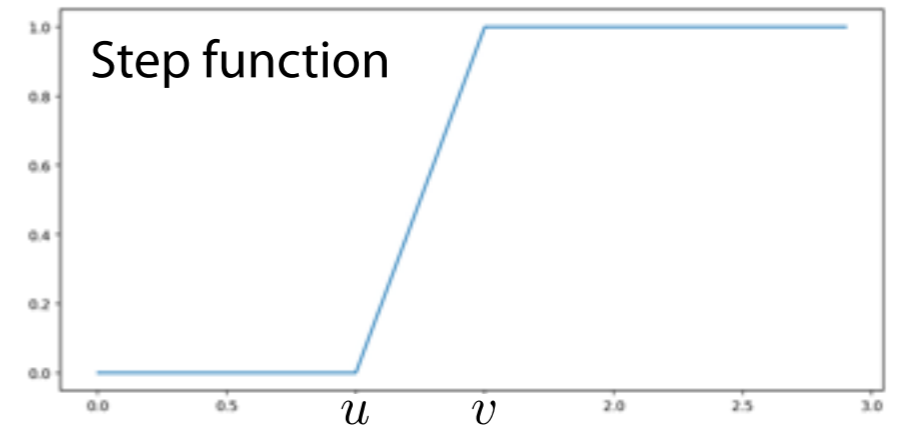
$$a_1^{(1)} = \text{relu}(x - u)$$

$$a_2^{(1)} = \text{relu}(x - v)$$

$$a_3^{(1)} = \text{relu}(x - c)$$

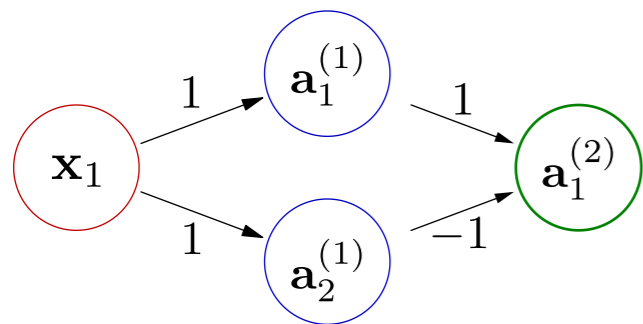
$$a_4^{(1)} = \text{relu}(x - d)$$

$$a_1^{(2)} = a_1^{(1)} - a_2^{(1)} - (a_3^{(1)} - a_4^{(1)})$$



Approximation power — universal

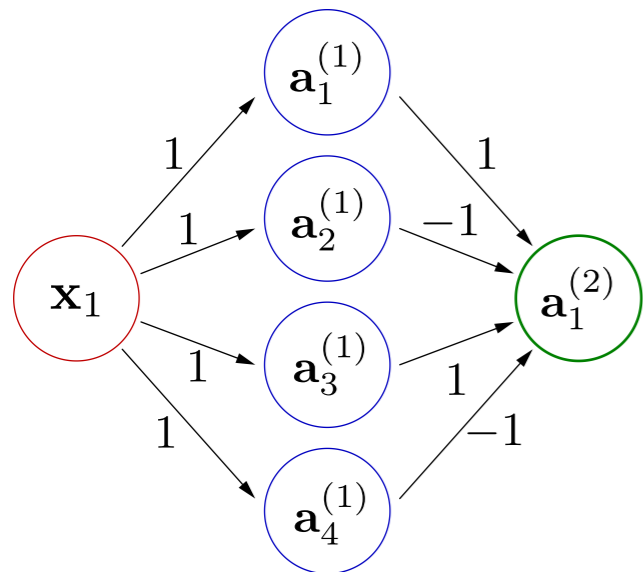
Four neurons can form a box-function, multiple boxes can approximate continuous functions.



$$a_1^{(1)} = \text{relu}(x - u)$$

$$a_2^{(1)} = \text{relu}(x - v)$$

$$a_1^{(2)} = a_1^{(1)} - a_2^{(1)}$$



$$a_1^{(1)} = \text{relu}(x - u)$$

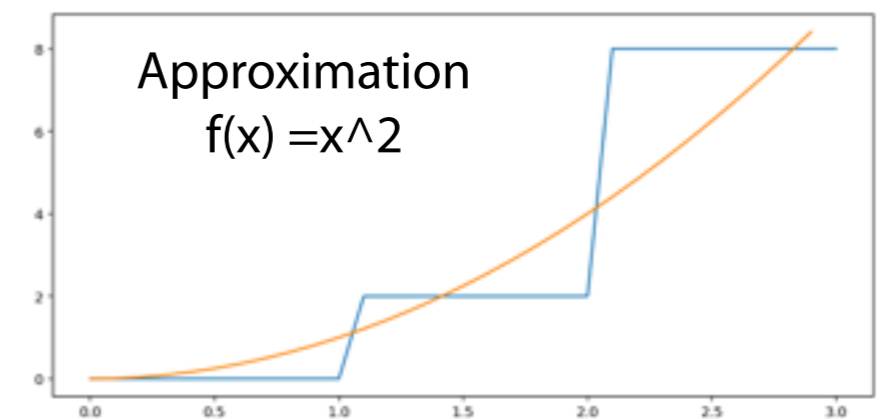
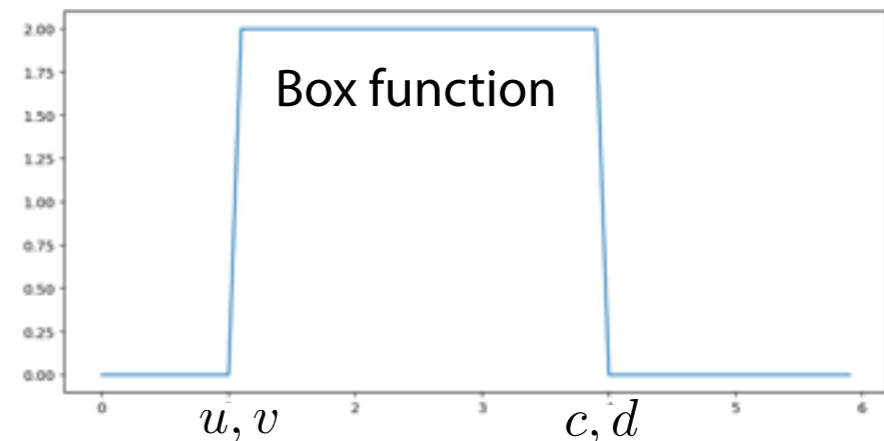
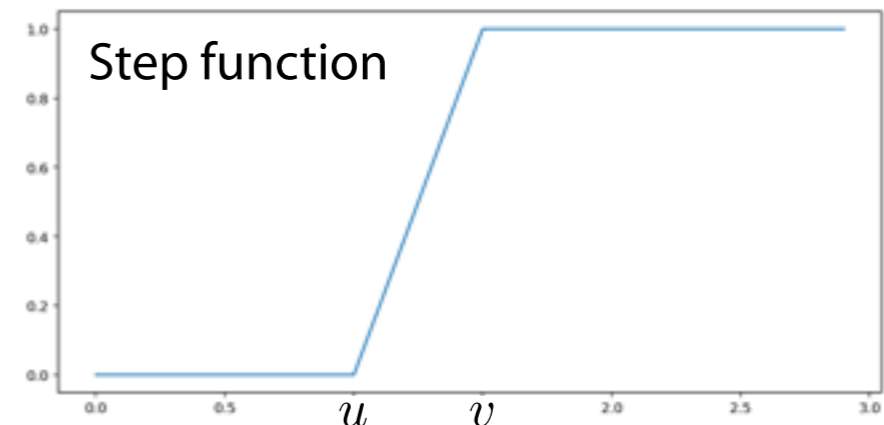
$$a_2^{(1)} = \text{relu}(x - v)$$

$$a_3^{(1)} = \text{relu}(x - c)$$

$$a_4^{(1)} = \text{relu}(x - d)$$

$$a_1^{(2)} = a_1^{(1)} - a_2^{(1)} - (a_3^{(1)} - a_4^{(1)})$$

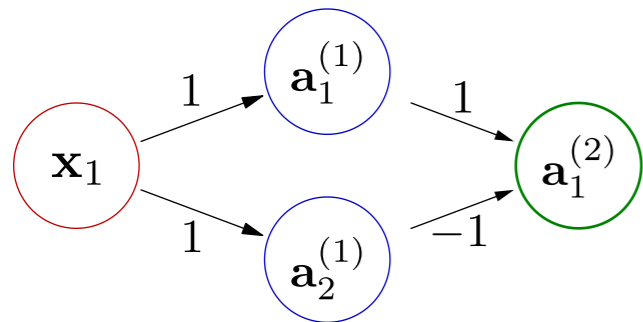
•
•
•



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

Approximation power — universal

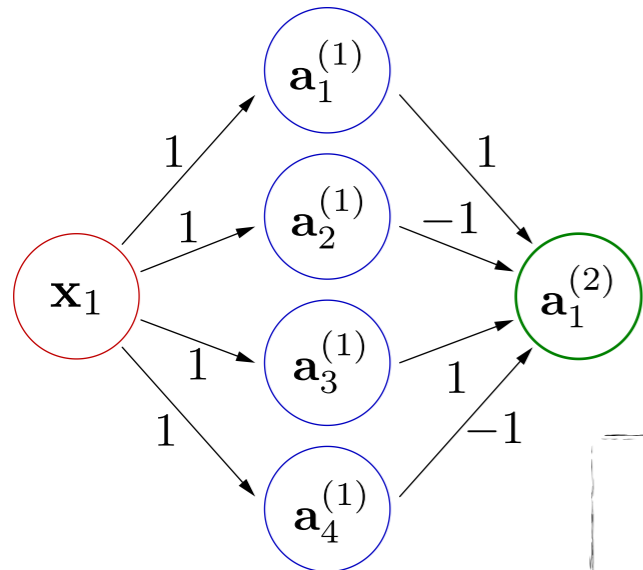
Four neurons can form a box-function, multiple boxes can approximate continuous functions.



$$a_1^{(1)} = \text{relu}(x - u)$$

$$a_2^{(1)} = \text{relu}(x - v)$$

$$a_1^{(2)} = a_1^{(1)} - a_2^{(1)}$$

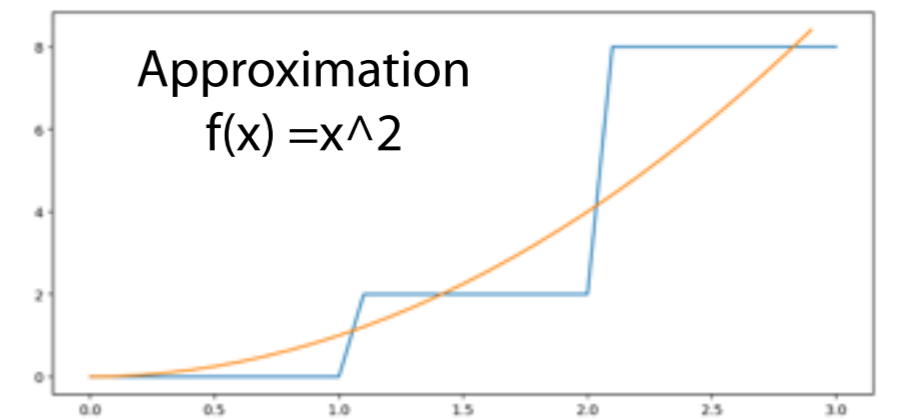
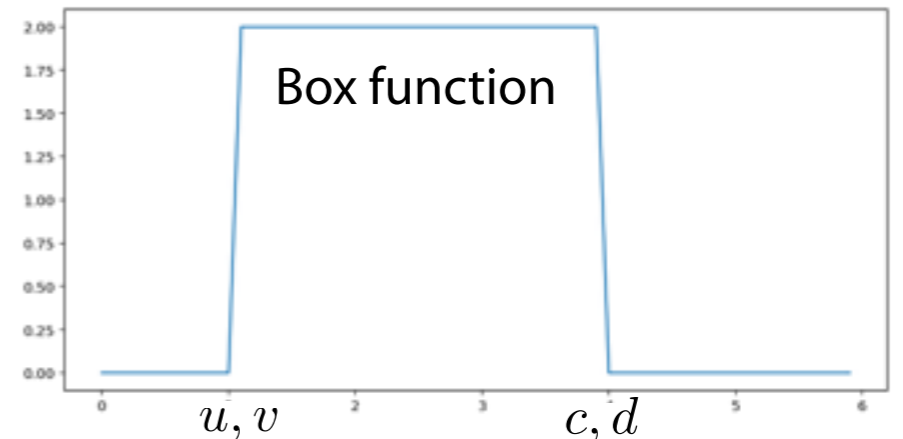
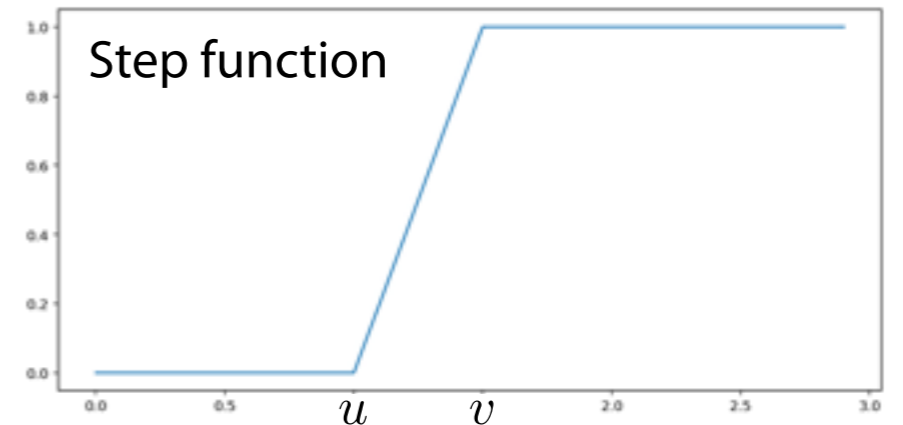


$$a_1^{(1)} = \text{relu}(x - u)$$

$$a_2^{(1)} = \text{relu}(x - v)$$

$$a_3^{(1)} = \text{relu}(x - c)$$

$$a_4^{(1)} = \text{relu}(x - d)$$



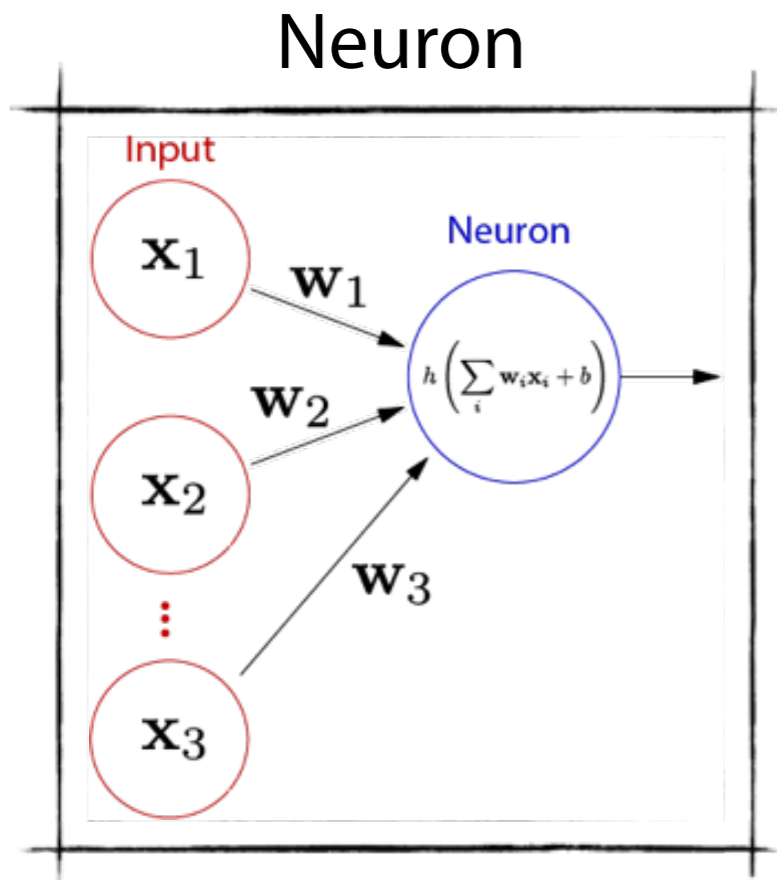
Approximation in 2D

[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.

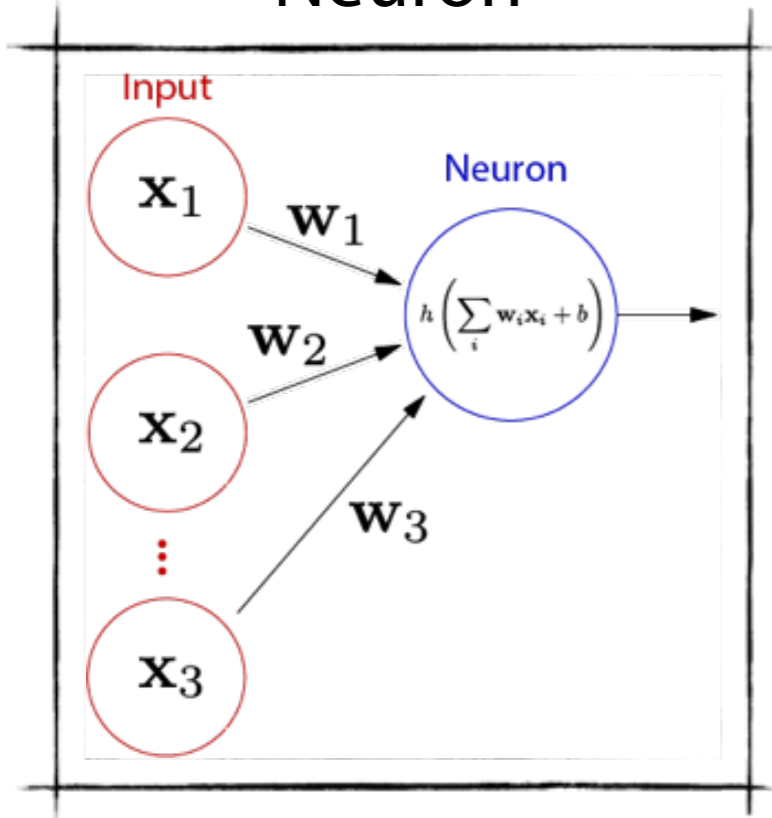


$$h\left(\sum_i w_i x_i + b\right)$$

Artificial neural network — structure

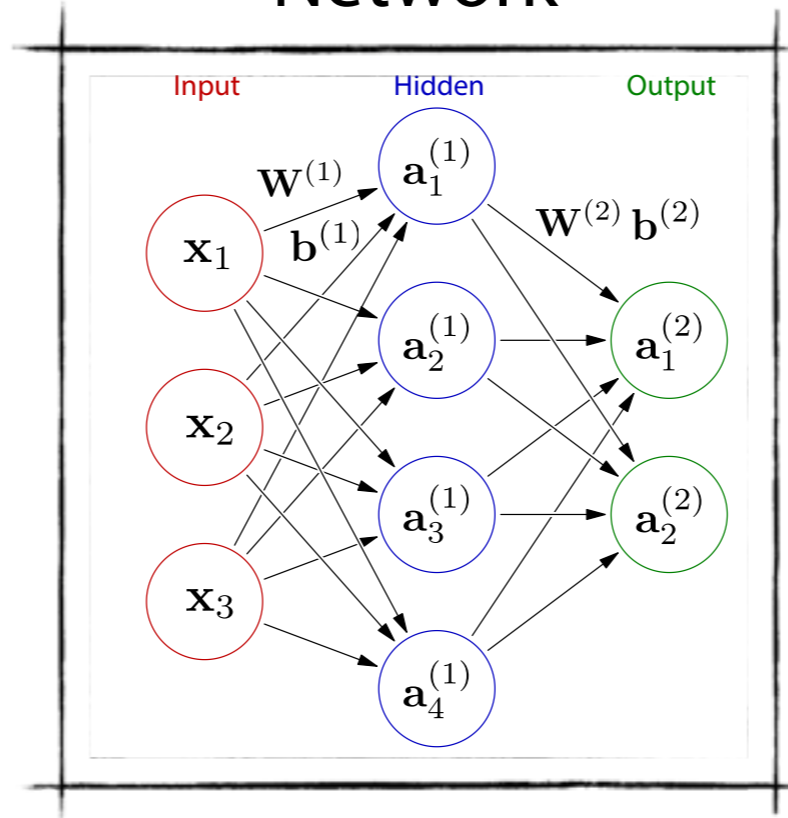
It is common and efficient to group neurons in layers and to use matrix-vector notation.

Neuron



$$h \left(\sum_i w_i x_i + b \right)$$

Network



$$a_1^{(1)} = h \left(w_{1,1}^{(1)} x_1 + w_{1,2}^{(1)} x_2 + w_{1,3}^{(1)} x_3 + b_1^{(1)} \right)$$

$$a_2^{(1)} = h \left(w_{2,1}^{(1)} x_1 + w_{2,2}^{(1)} x_2 + w_{2,3}^{(1)} x_3 + b_2^{(1)} \right)$$

$$a_3^{(1)} = h \left(w_{3,1}^{(1)} x_1 + w_{3,2}^{(1)} x_2 + w_{3,3}^{(1)} x_3 + b_3^{(1)} \right)$$

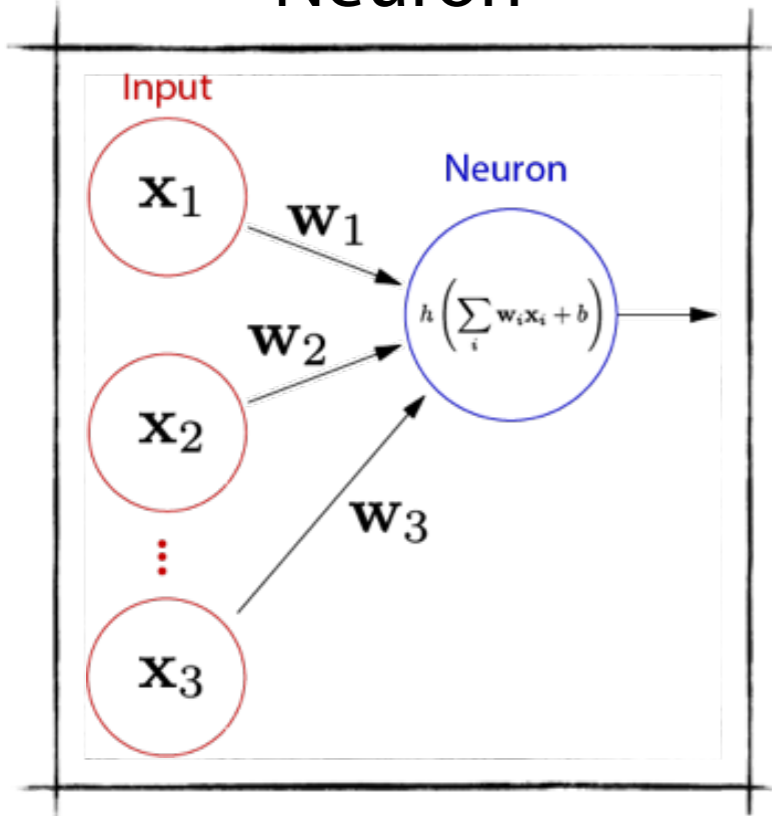
$$a_1^{(2)} = w_{1,1}^{(2)} a_1^{(1)} + w_{1,2}^{(2)} a_2^{(1)} + w_{1,3}^{(2)} a_3^{(1)} + b_1^{(2)}$$

$$a_2^{(2)} = w_{2,1}^{(2)} a_1^{(1)} + w_{2,2}^{(2)} a_2^{(1)} + w_{2,3}^{(2)} a_3^{(1)} + b_2^{(2)}$$

Artificial neural network — structure

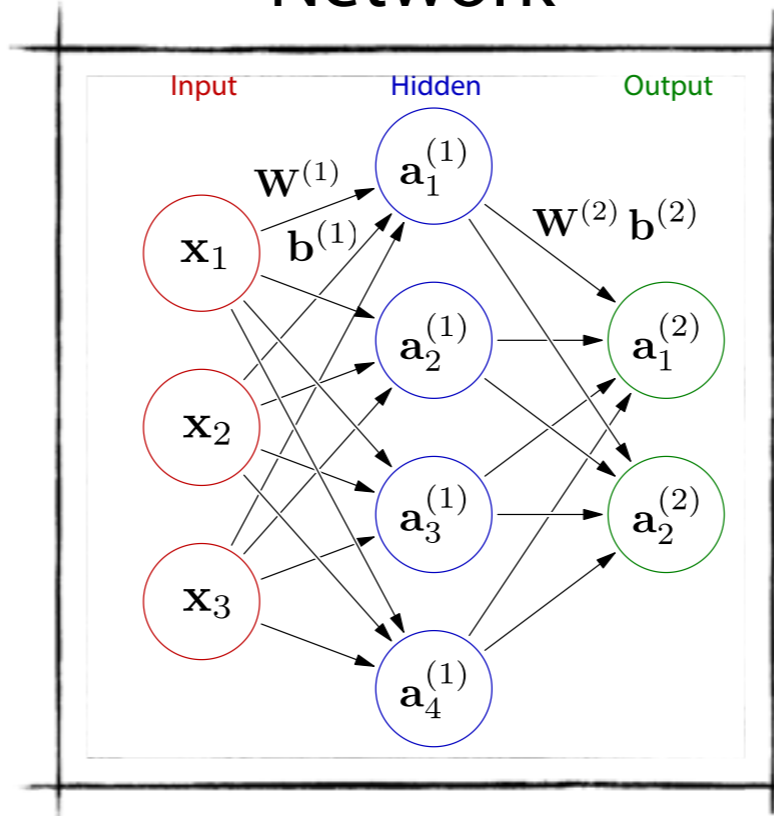
It is common and efficient to group neurons in layers and to use matrix-vector notation.

Neuron



$$h \left(\sum_i w_i x_i + b \right)$$

Network



$$a_1^{(1)} = h \left(w_{1,1}^{(1)} x_1 + w_{1,2}^{(1)} x_2 + w_{1,3}^{(1)} x_3 + b_1^{(1)} \right)$$

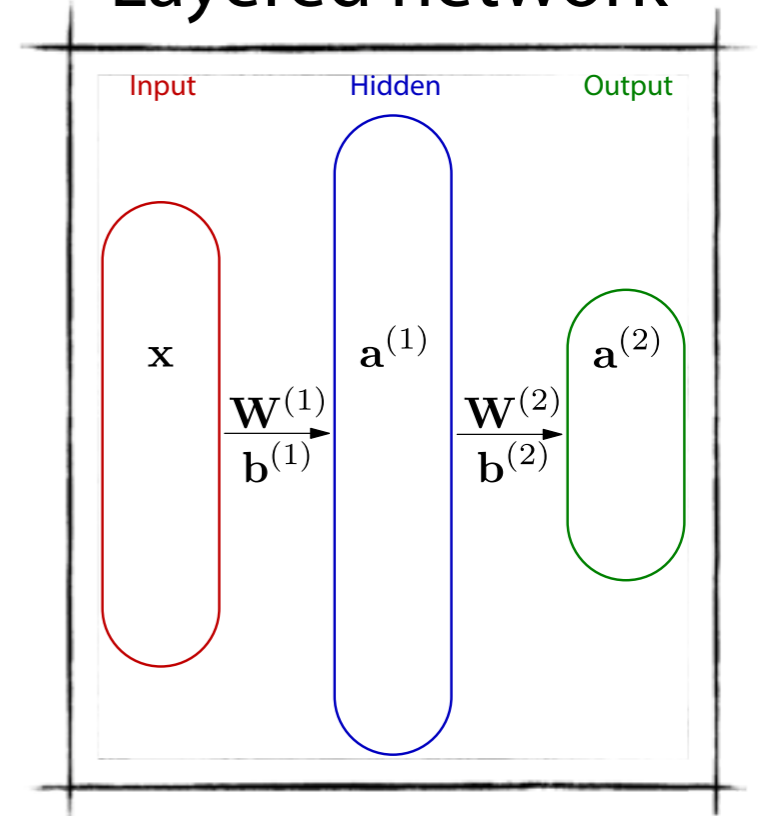
$$a_2^{(1)} = h \left(w_{2,1}^{(1)} x_1 + w_{2,2}^{(1)} x_2 + w_{2,3}^{(1)} x_3 + b_2^{(1)} \right)$$

$$a_3^{(1)} = h \left(w_{3,1}^{(1)} x_1 + w_{3,2}^{(1)} x_2 + w_{3,3}^{(1)} x_3 + b_3^{(1)} \right)$$

$$a_1^{(2)} = w_{1,1}^{(2)} a_1^{(1)} + w_{1,2}^{(2)} a_2^{(1)} + w_{1,3}^{(2)} a_3^{(1)} + b_1^{(2)}$$

$$a_2^{(2)} = w_{2,1}^{(2)} a_1^{(1)} + w_{2,2}^{(2)} a_2^{(1)} + w_{2,3}^{(2)} a_3^{(1)} + b_2^{(2)}$$

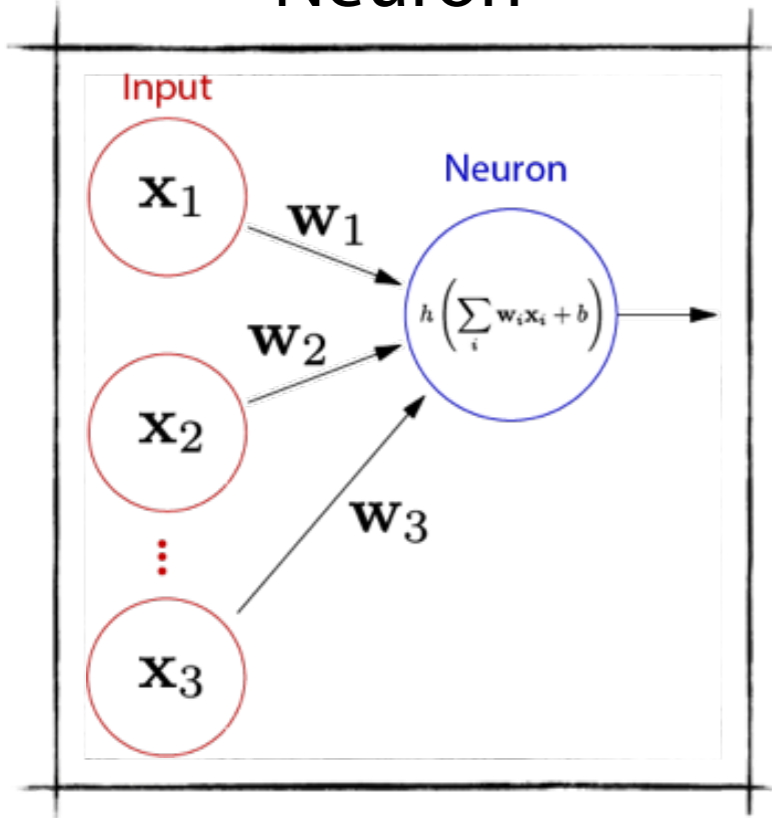
Layered network



Artificial neural network — structure

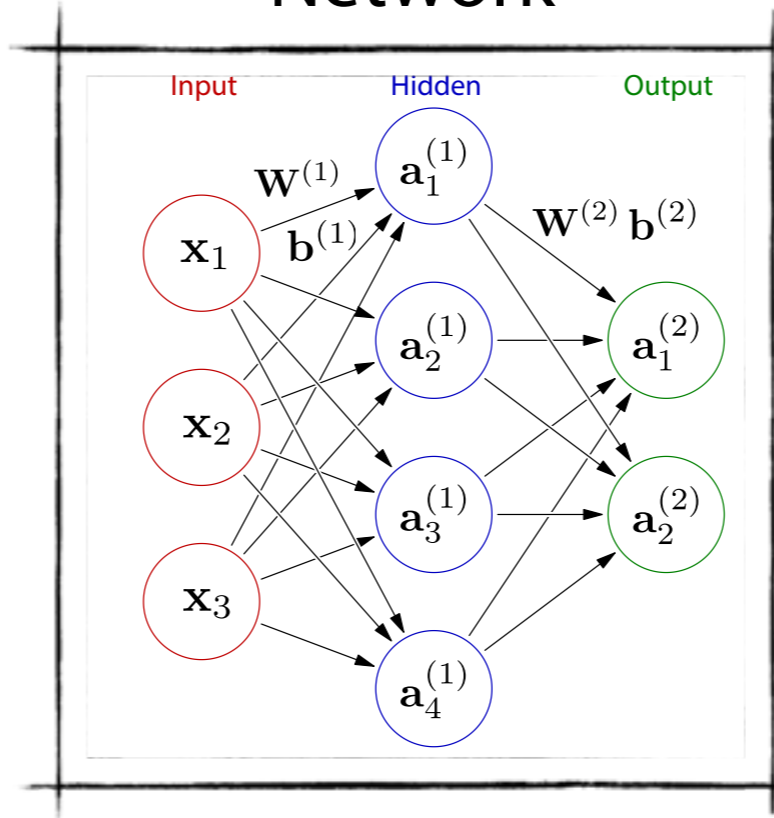
It is common and efficient to group neurons in layers and to use matrix-vector notation.

Neuron



$$h\left(\sum_i w_i x_i + b\right)$$

Network



$$a_1^{(1)} = h\left(w_{1,1}^{(1)}x_1 + w_{1,2}^{(1)}x_2 + w_{1,3}^{(1)}x_3 + b_1^{(1)}\right)$$

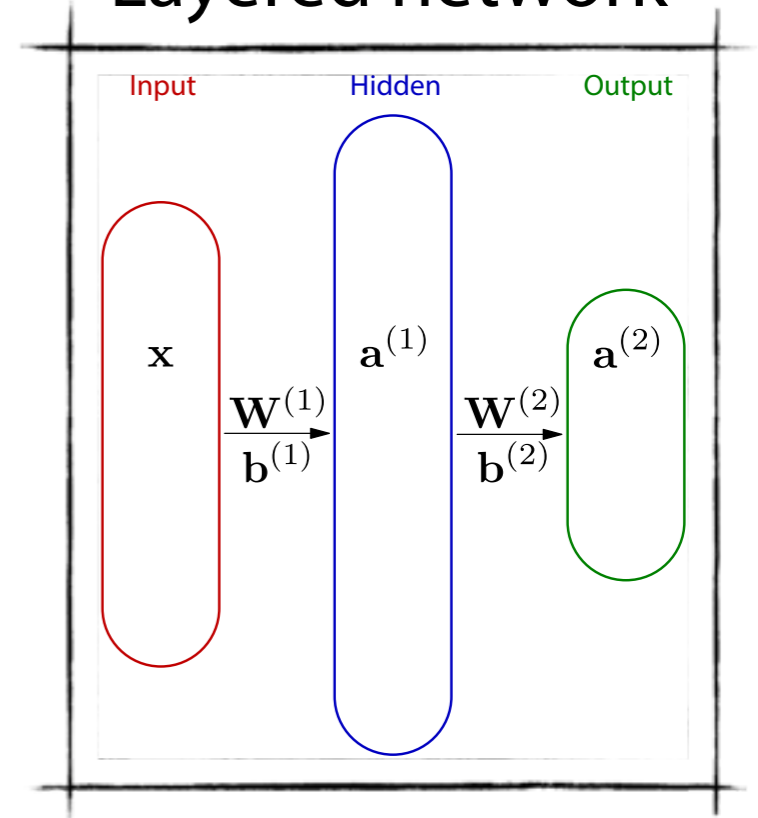
$$a_2^{(1)} = h\left(w_{2,1}^{(1)}x_1 + w_{2,2}^{(1)}x_2 + w_{2,3}^{(1)}x_3 + b_2^{(1)}\right)$$

$$a_3^{(1)} = h\left(w_{3,1}^{(1)}x_1 + w_{3,2}^{(1)}x_2 + w_{3,3}^{(1)}x_3 + b_3^{(1)}\right)$$

$$a_1^{(2)} = w_{1,1}^{(2)}a_1^{(1)} + w_{1,2}^{(2)}a_2^{(1)} + w_{1,3}^{(2)}a_3^{(1)} + b_1^{(2)}$$

$$a_2^{(2)} = w_{2,1}^{(2)}a_1^{(1)} + w_{2,2}^{(2)}a_2^{(1)} + w_{2,3}^{(2)}a_3^{(1)} + b_2^{(2)}$$

Layered network



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

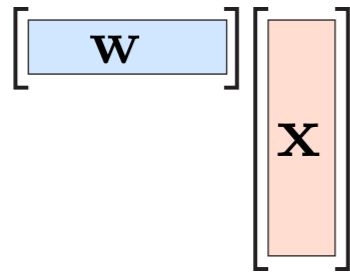
$$a^{(2)} = W^{(2)}a^{(1)} + b^{(2)}$$

Linear and affine transformations

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

Linear

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



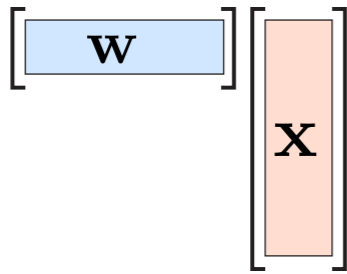
A diagram illustrating the dot product $\mathbf{w} \cdot \mathbf{x}$. It shows a horizontal light blue box labeled \mathbf{w} and a vertical light orange box labeled \mathbf{x} . Both boxes are enclosed in square brackets, representing the vectors \mathbf{w} and \mathbf{x} .

Linear and affine transformations

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

Linear

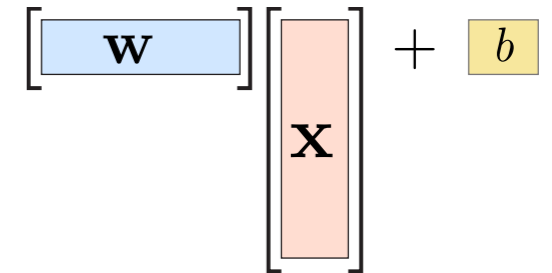
$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i$$
$$= \mathbf{w} \cdot \mathbf{x}$$



A diagram illustrating the dot product of a weight vector \mathbf{w} and an input vector \mathbf{x} . The weight vector \mathbf{w} is represented by a horizontal blue box, and the input vector \mathbf{x} is represented by a vertical orange box.

Affine

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



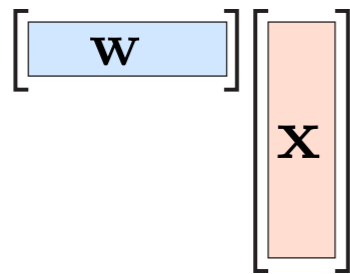
A diagram illustrating the affine transformation equation. It shows a blue box labeled \mathbf{w} next to a vertical orange box labeled \mathbf{x} , followed by a plus sign and a yellow box labeled b .

Linear and affine transformations

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

Linear

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

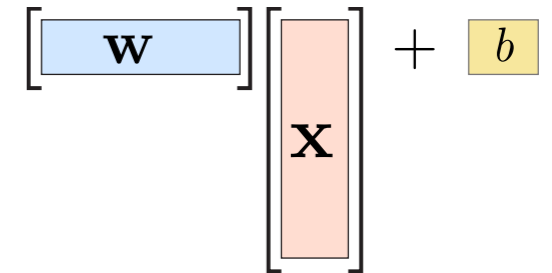


A diagram illustrating the dot product of a weight vector \mathbf{w} and an input vector \mathbf{x} . The weight vector \mathbf{w} is represented by a horizontal blue box, and the input vector \mathbf{x} is represented by a vertical orange box.

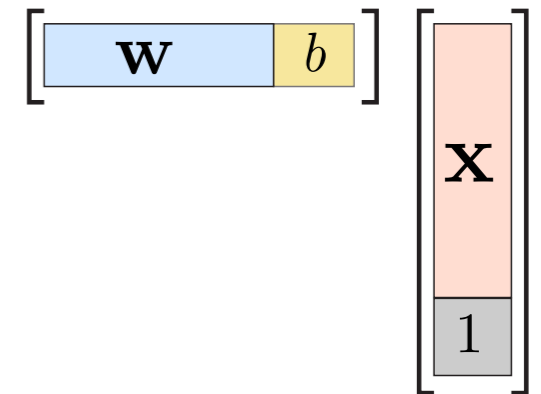
Affine

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

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



A diagram showing the linear part of an affine transformation: a blue box labeled \mathbf{w} multiplied by a vertical orange box labeled \mathbf{x} , plus a yellow box labeled b .



A diagram showing the augmented vector representation of an affine transformation. A blue box labeled \mathbf{w} and a yellow box labeled b are combined into a single horizontal box. This is multiplied by a vertical orange box labeled \mathbf{x} with a grey box labeled 1 at the bottom.

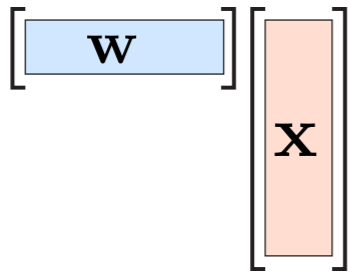


Linear and affine transformations

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

Linear

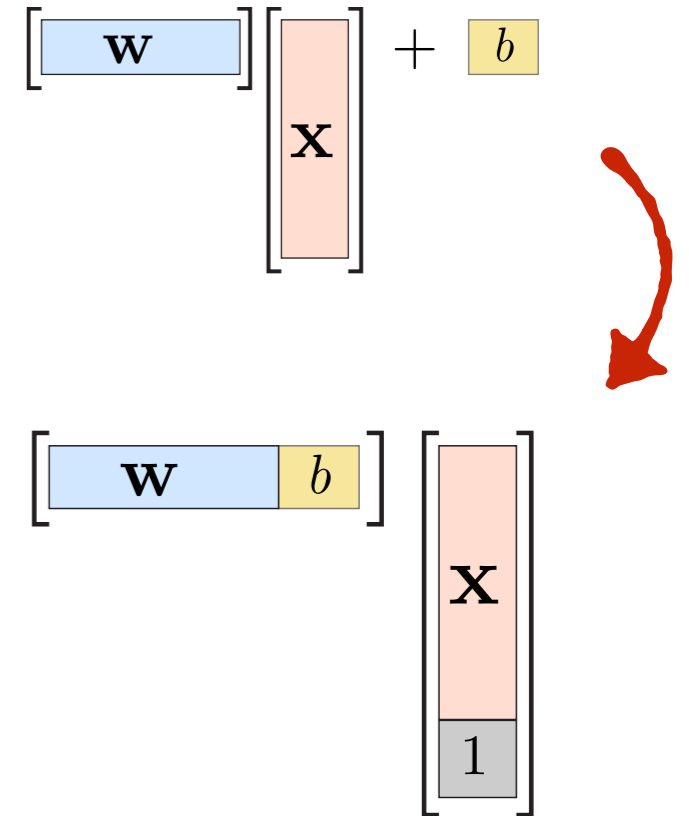
$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i$$
$$= \mathbf{w} \cdot \mathbf{x}$$



Affine

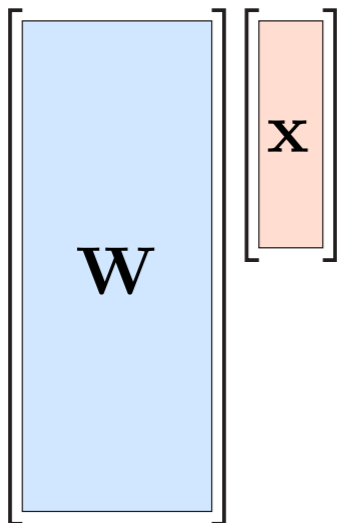
$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i + b$$
$$= \mathbf{w} \cdot \mathbf{x} + b$$
$$= \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

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



Multidimensional

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

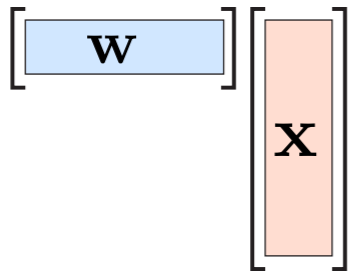


Linear and affine transformations

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

Linear

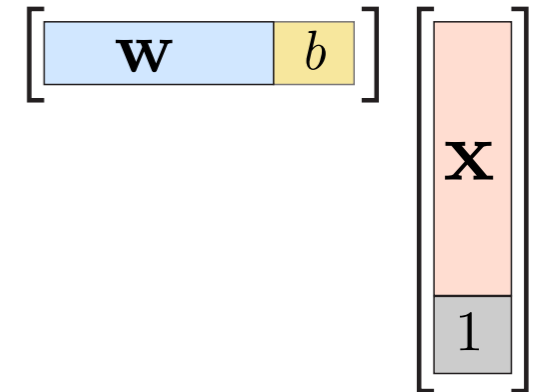
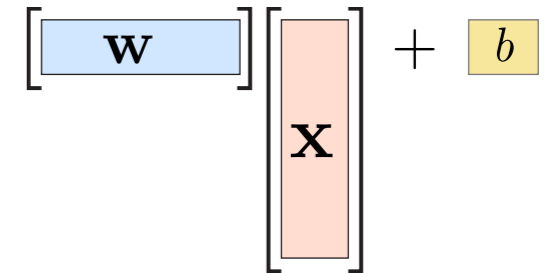
$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i$$
$$= \mathbf{w} \cdot \mathbf{x}$$



Affine

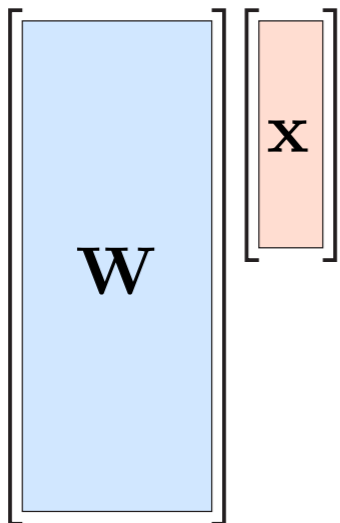
$$f(\mathbf{x}) = \sum_i \mathbf{w}_i \mathbf{x}_i + b$$
$$= \mathbf{w} \cdot \mathbf{x} + b$$
$$= \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

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



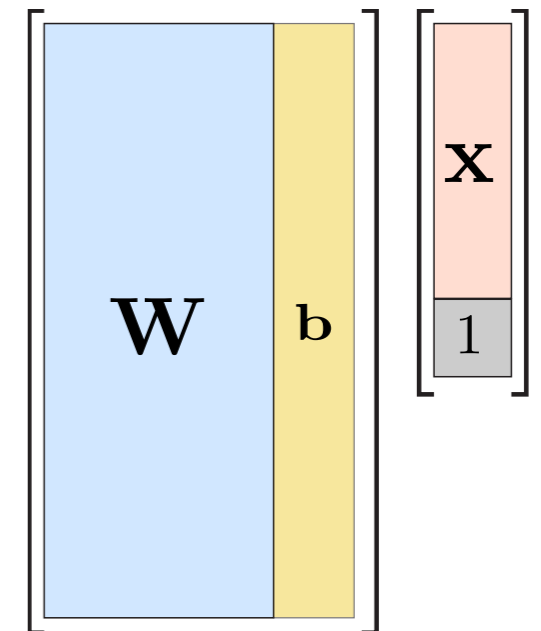
Multidimensional

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



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

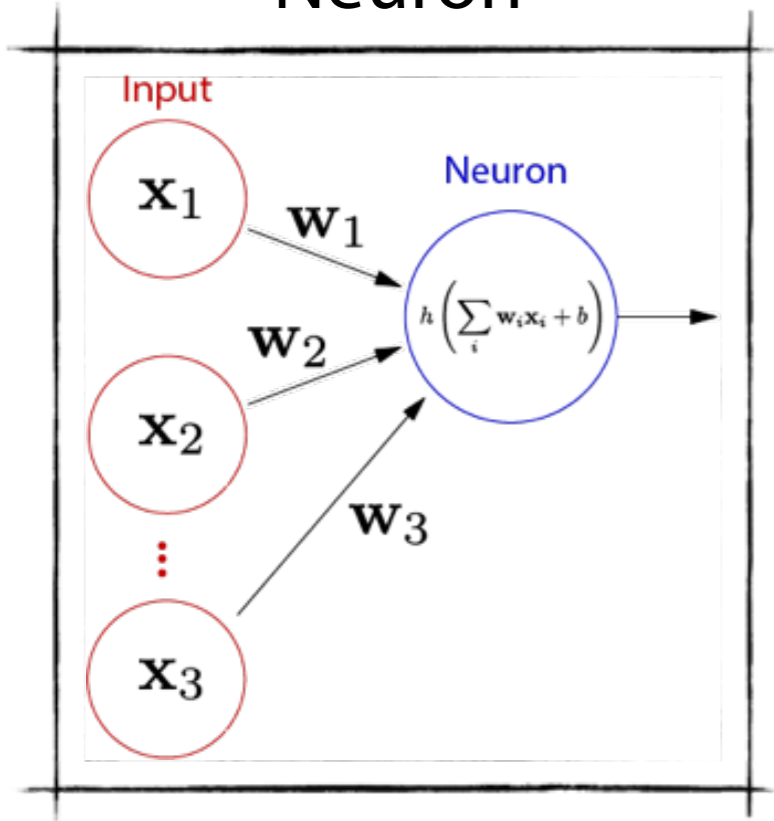
with $\tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{w}_{1,1} & \mathbf{w}_{1,2} & \dots & \mathbf{w}_{1,n} & b_1 \\ \mathbf{w}_{2,1} & \mathbf{w}_{2,2} & \dots & \mathbf{w}_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \end{pmatrix}$



Artificial neural network — matrix notation

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

Neuron



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

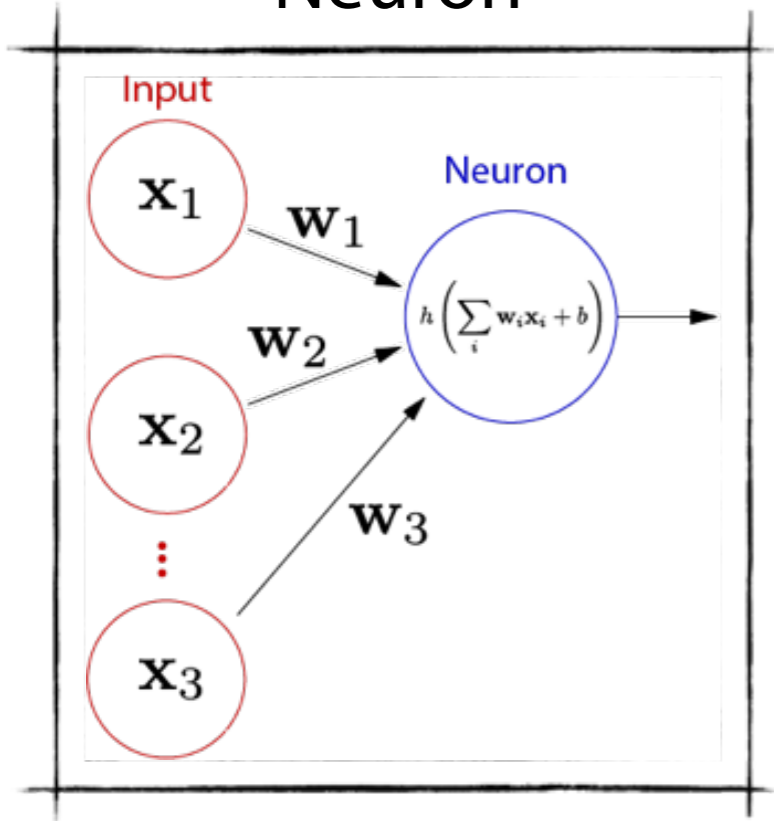
Matrix representation

$$\begin{bmatrix} \mathbf{w} \end{bmatrix} + \begin{bmatrix} \mathbf{x} \end{bmatrix}$$

Artificial neural network — matrix notation

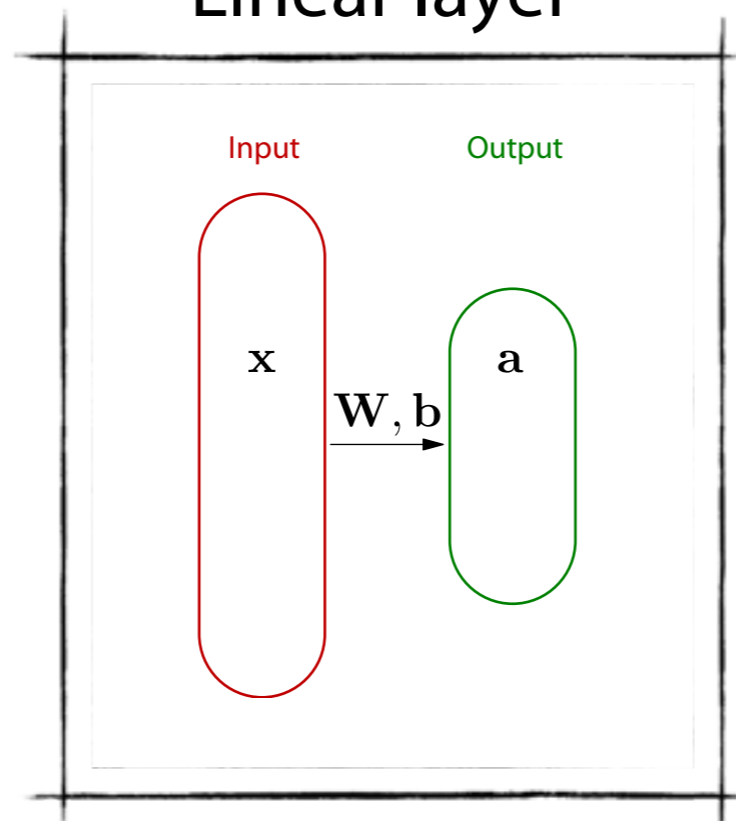
Layered NNs are simply a chain of matrix multiplications and activation functions.

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

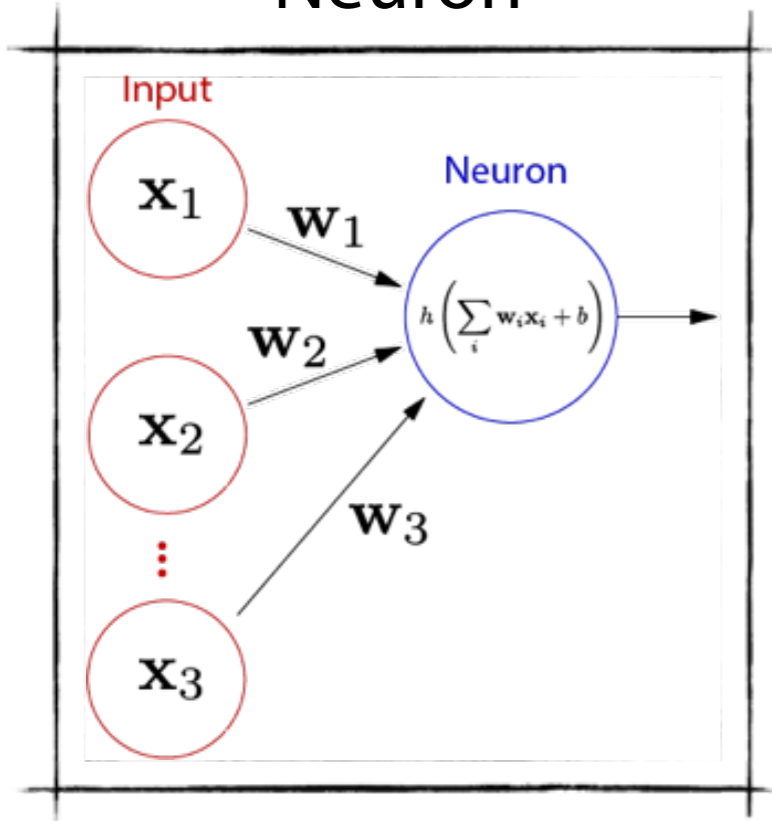
$$\begin{bmatrix} \mathbf{w} \end{bmatrix} + \begin{bmatrix} \mathbf{x} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{W} \end{bmatrix} + \begin{bmatrix} \mathbf{x} \end{bmatrix} + \begin{bmatrix} \mathbf{b} \end{bmatrix}$$

Artificial neural network — matrix notation

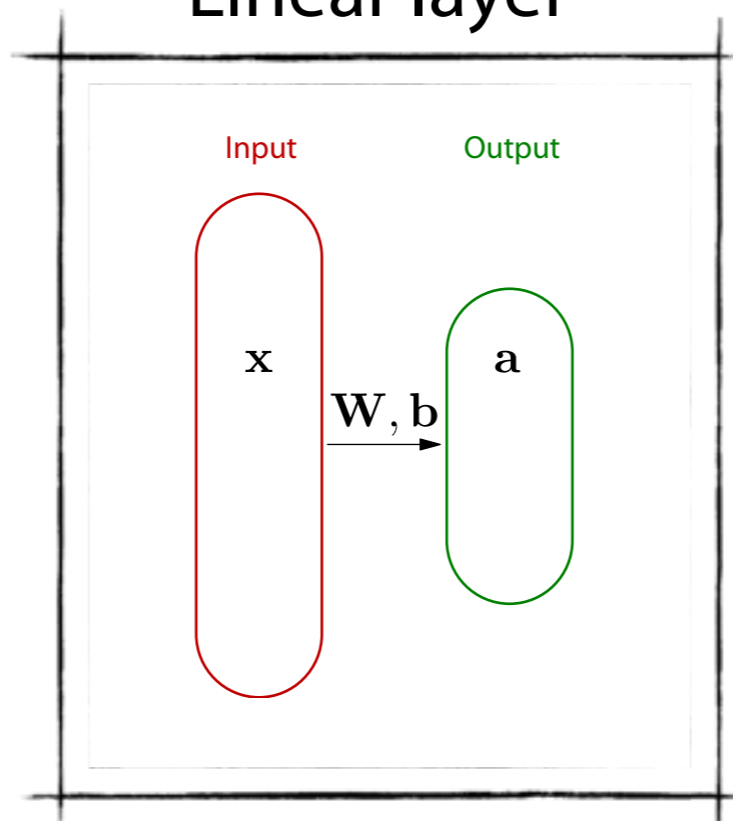
Layered NNs are simply a chain of matrix multiplications and activation functions.

Neuron



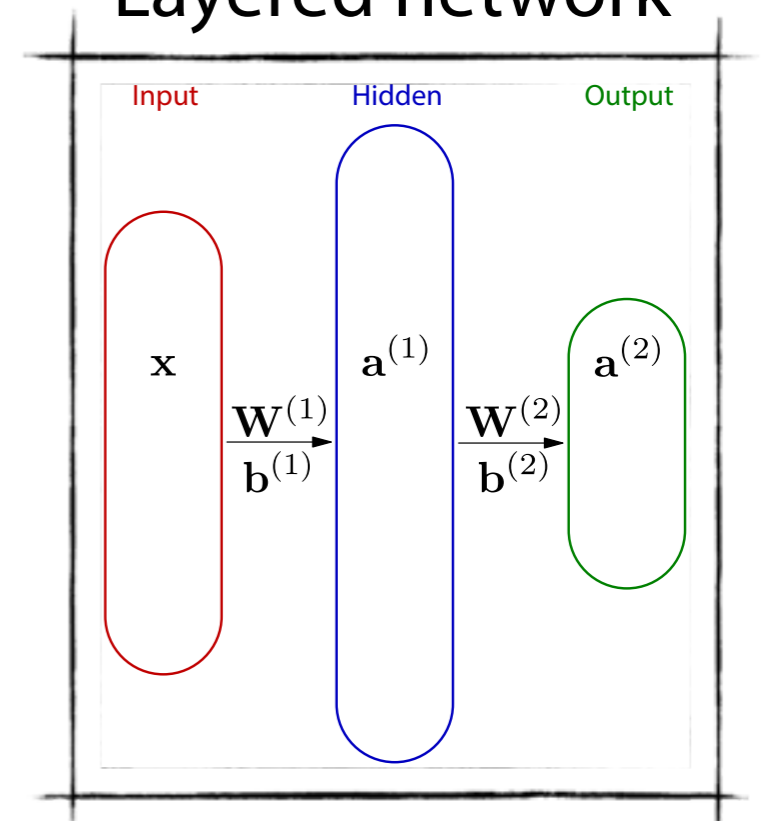
$$h(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$$

Linear layer



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

Layered network



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

Matrix representation

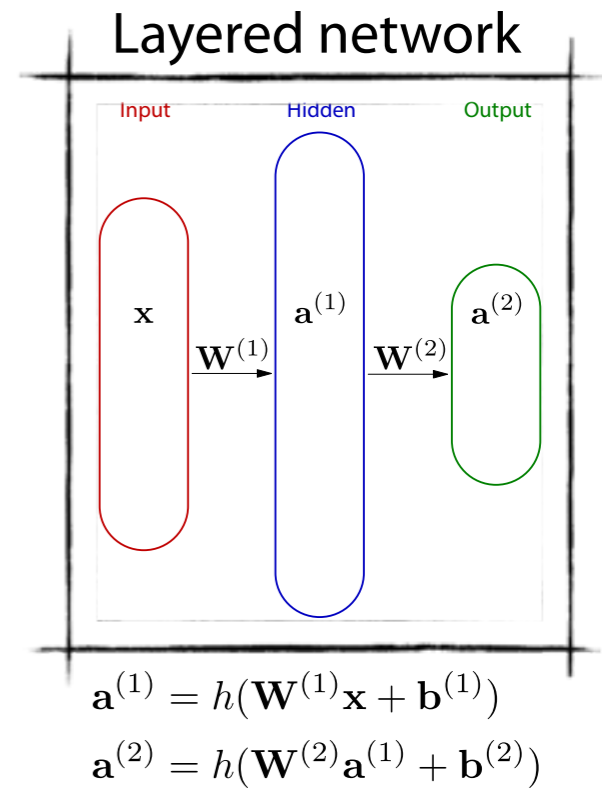
$$\begin{bmatrix} \mathbf{W} \end{bmatrix} + \begin{bmatrix} \mathbf{x} \end{bmatrix} + \begin{bmatrix} \mathbf{b} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{W} \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix} + \begin{bmatrix} \mathbf{b} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{W}^{(2)} \end{bmatrix} h \left(\begin{bmatrix} \mathbf{W}^{(1)} \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix} + \begin{bmatrix} \mathbf{b}^{(1)} \end{bmatrix} \right) + \begin{bmatrix} \mathbf{b}^{(2)} \end{bmatrix}$$

Artificial neural network — matrix notation

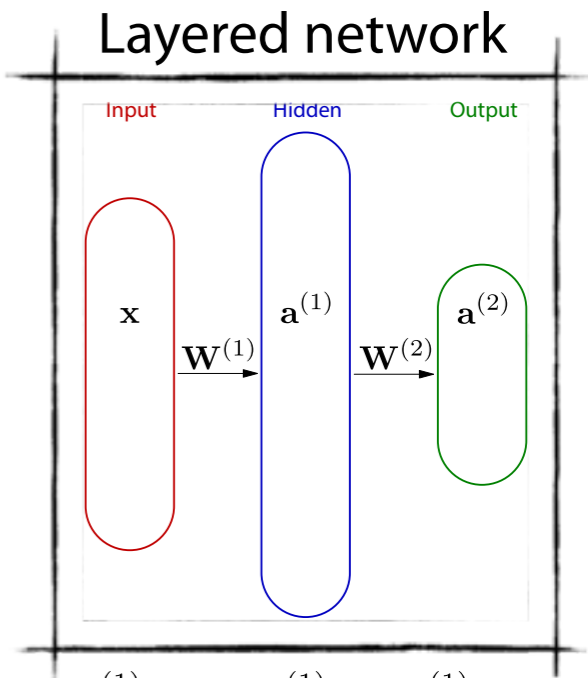
Layered NNs are simply a chain of matrix multiplications and activation functions.



$$\mathbf{W}^{(2)} h \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right) + \mathbf{b}^{(2)}$$

Artificial neural network — matrix notation

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



The shape of W defines the network structure

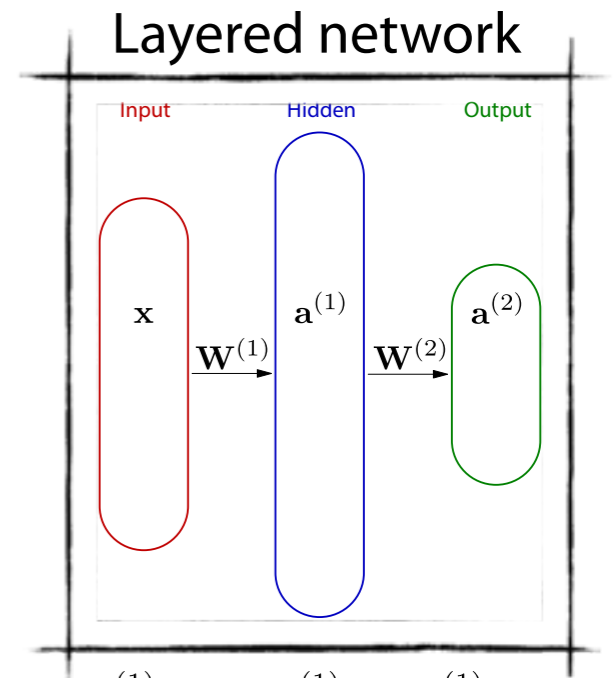
$$a^{(1)} = h(W^{(1)}x + b^{(1)})$$
$$a^{(2)} = h(W^{(2)}a^{(1)} + b^{(2)})$$

The diagram shows the matrix notation for the neural network output. It consists of a large blue square matrix labeled $w^{(2)}$ followed by the activation function h in a large italicized font. This is followed by a large black parenthesis containing a blue rectangular matrix labeled $w^{(1)}$, a pink vertical rectangle labeled x , and a plus sign followed by a yellow vertical rectangle labeled $b^{(1)}$. Outside the parenthesis is another plus sign followed by a yellow vertical rectangle labeled $b^{(2)}$. Red arrows point from the text 'The shape of W defines the network structure' to the $w^{(1)}$ and $w^{(2)}$ matrices.

$$w^{(2)} h \left(w^{(1)} x + b^{(1)} \right) + b^{(2)}$$

Artificial neural network — matrix notation

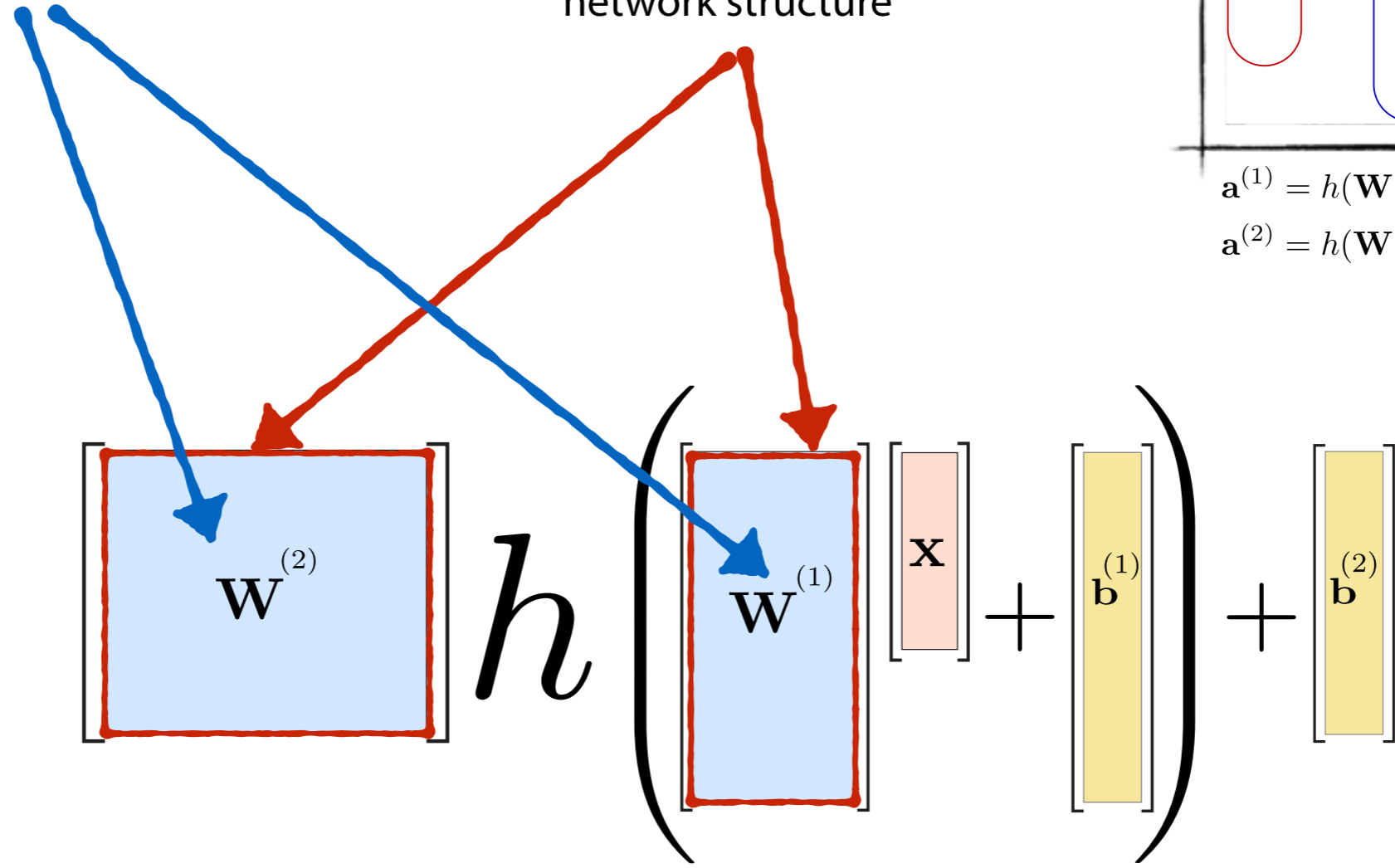
Layered NNs are simply a chain of matrix multiplications and activation functions.



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

The values of \mathbf{W} define the functionality.

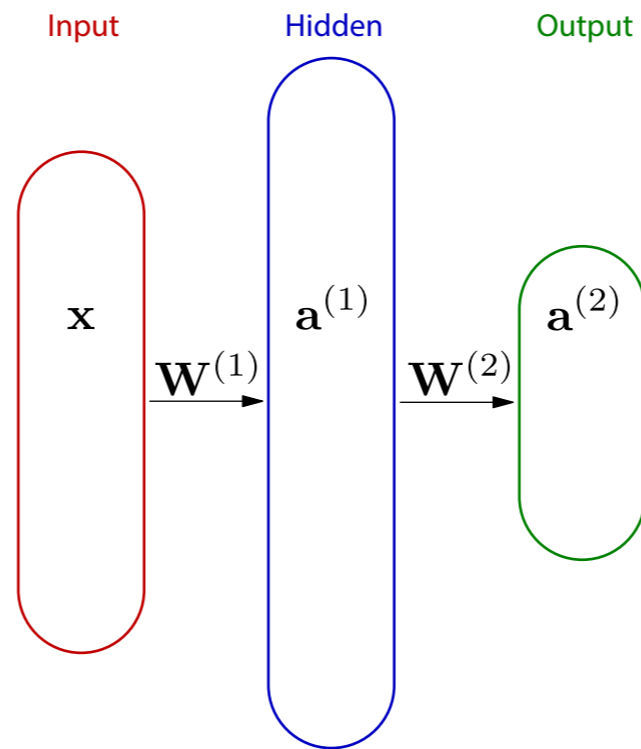
The shape of \mathbf{W} defines the network structure



Deep Learning

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

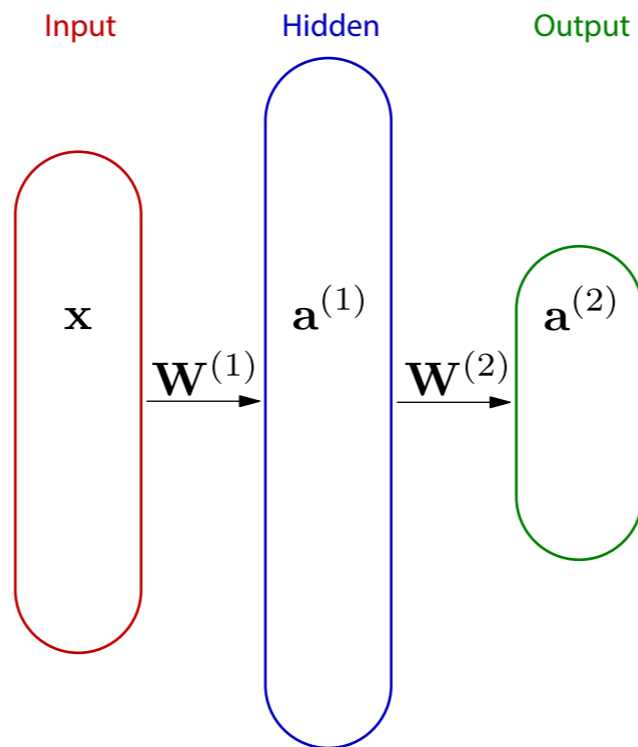
Shallow Learning



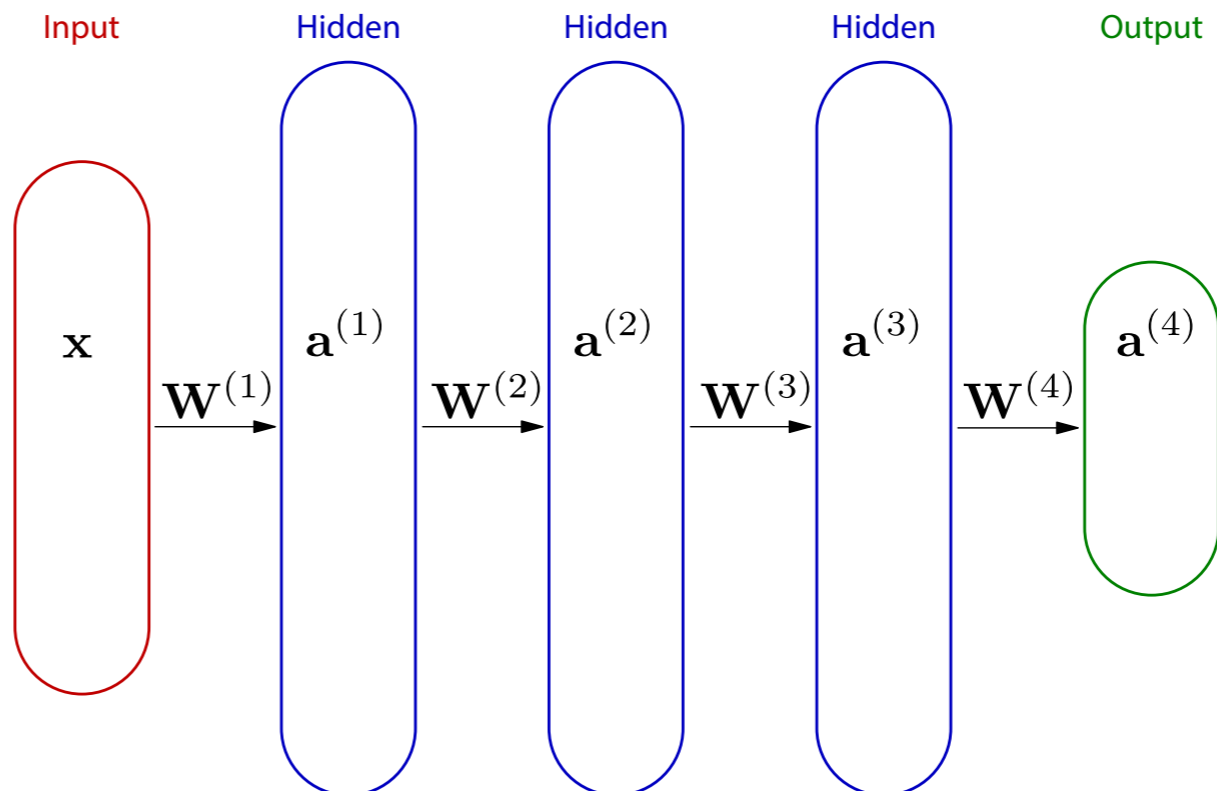
Deep Learning

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.

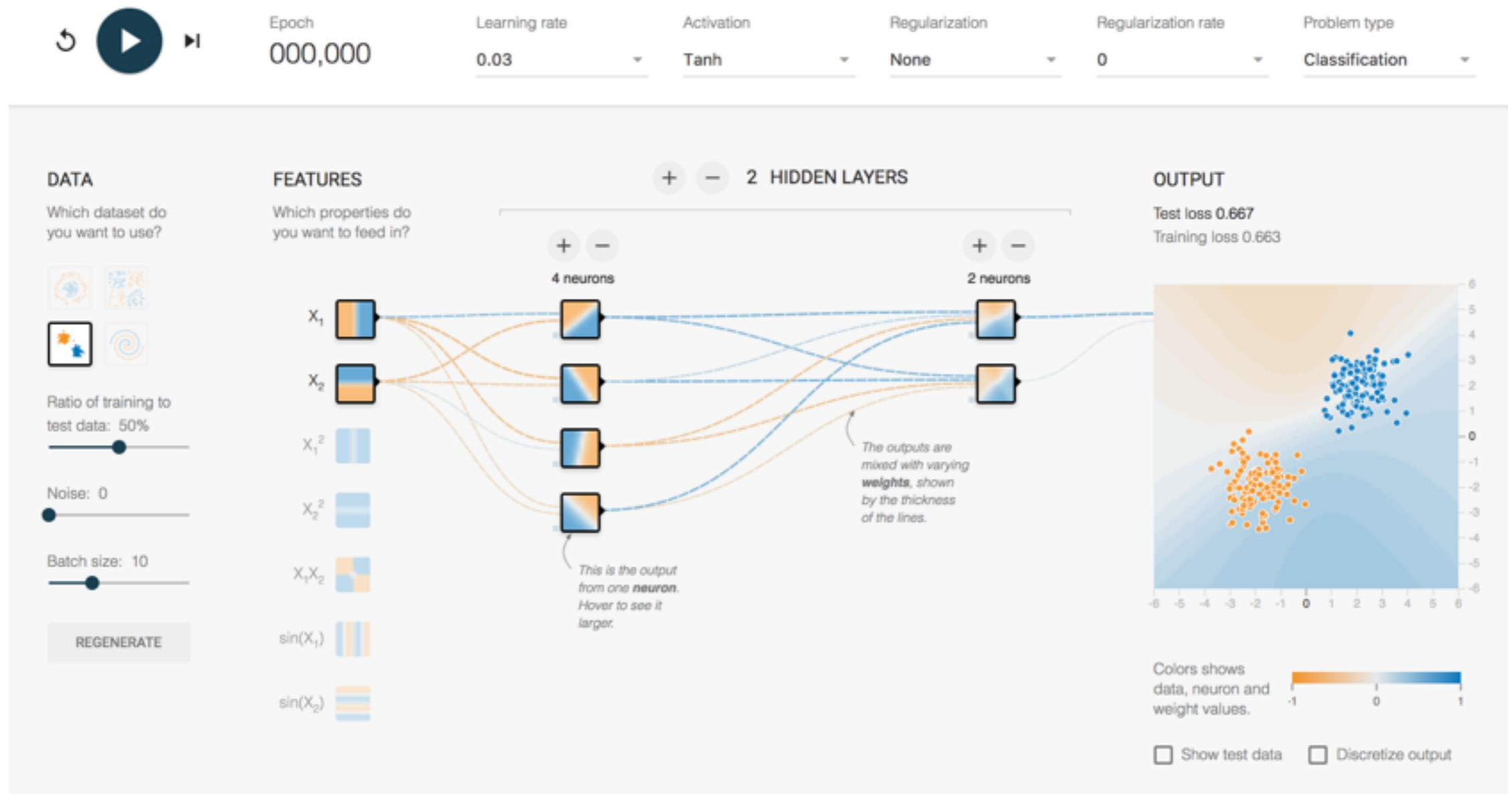
Functional representation

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

Matrix representation

$$\text{nn} = h \left(\boxed{\mathbf{W}^{(d)}} \dots h \left(\boxed{\mathbf{W}^{(1)}} \boxed{\mathbf{x}} + \boxed{\mathbf{b}^{(1)}} \right) \dots + \boxed{\mathbf{b}^{(d)}} \right)$$

Neural network playground (classification)

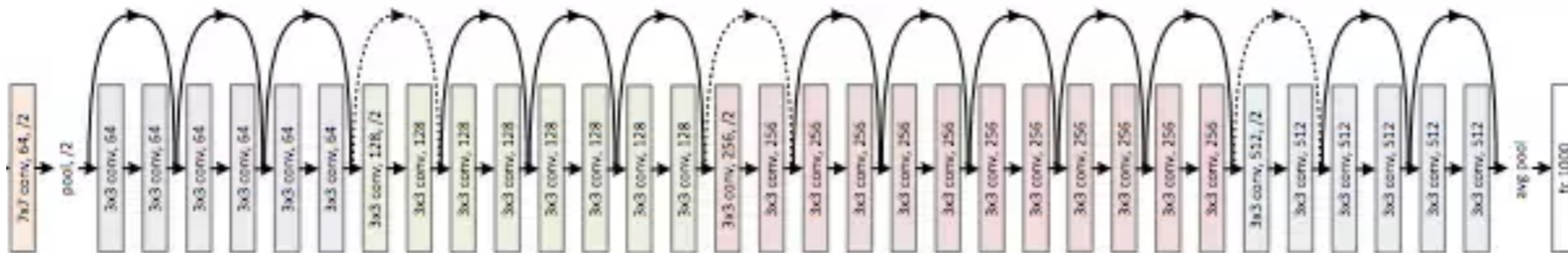


<http://playground.tensorflow.org>

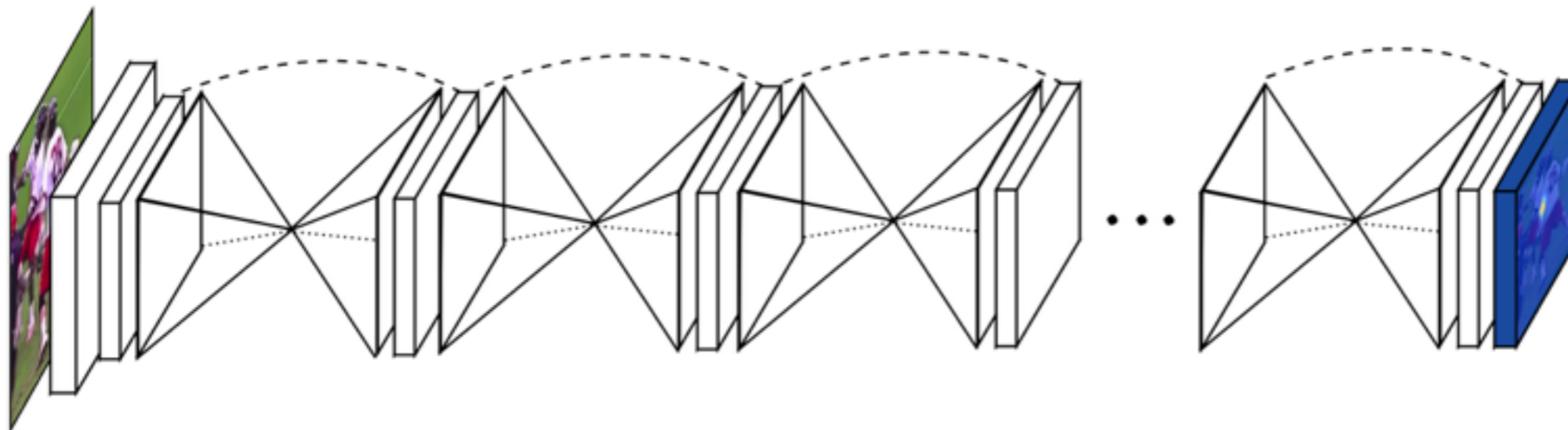
Deeeeep Learning

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.

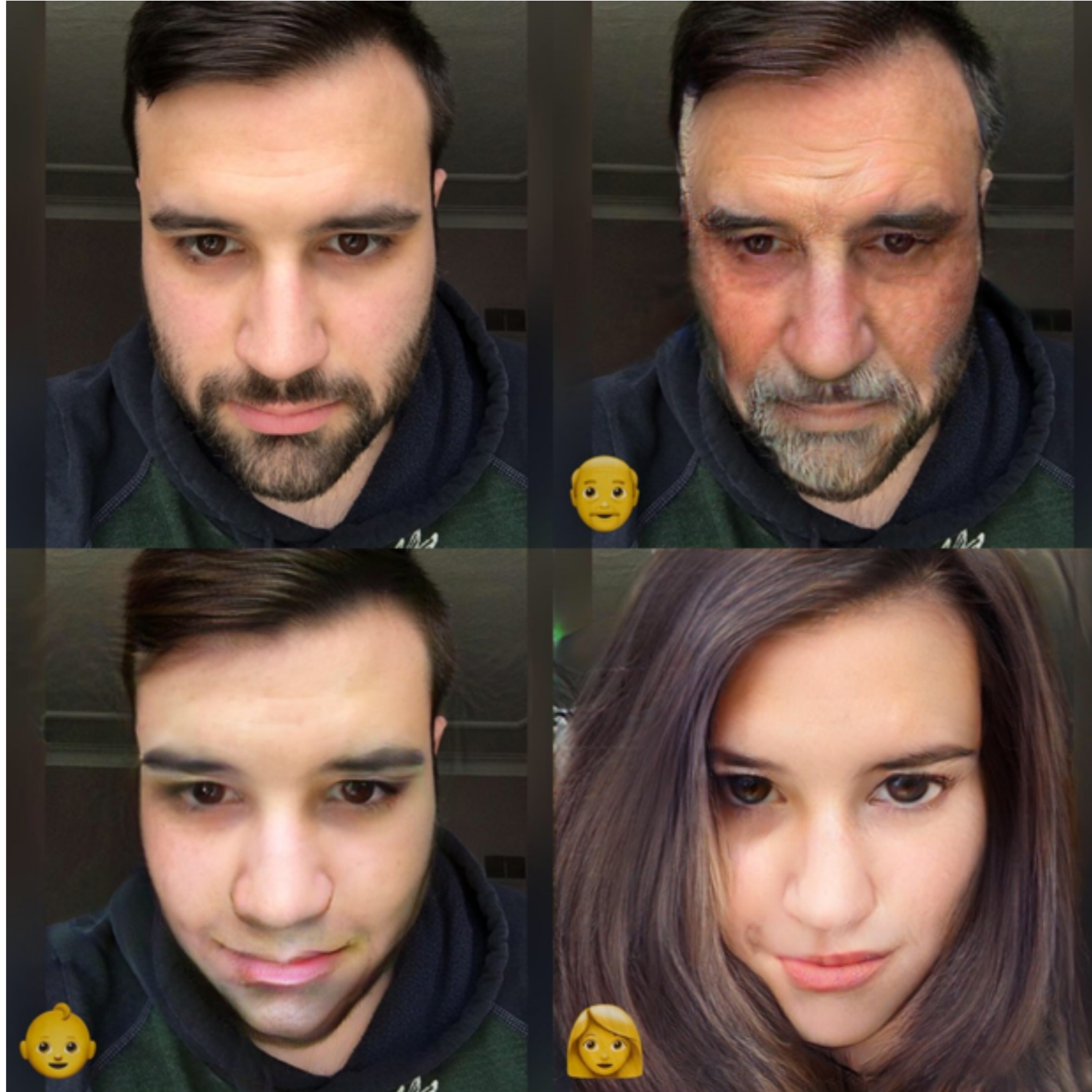


Image to image translation



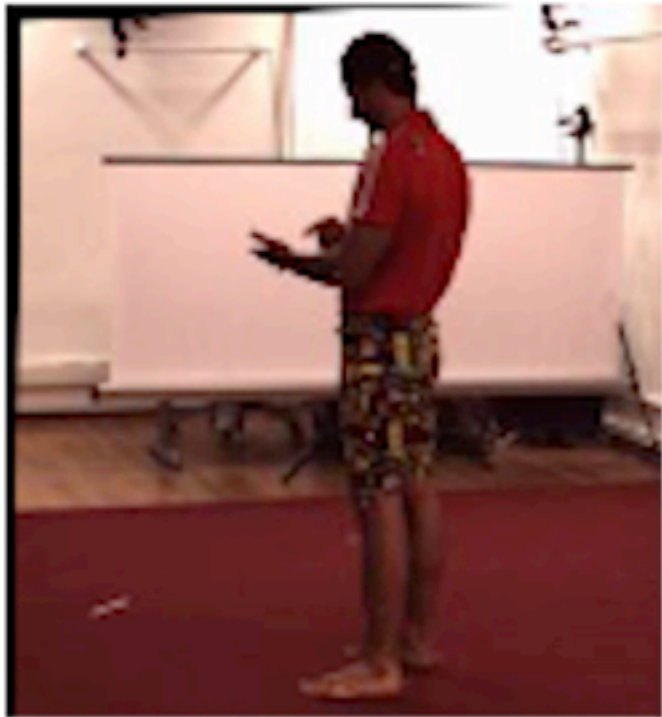
[Everybody dance now. Chan et al. 2018]

Image to image translation



[Everybody dance now. Chan et al. 2018]

My research on human motion capture



[Rhodin et al. 2018]

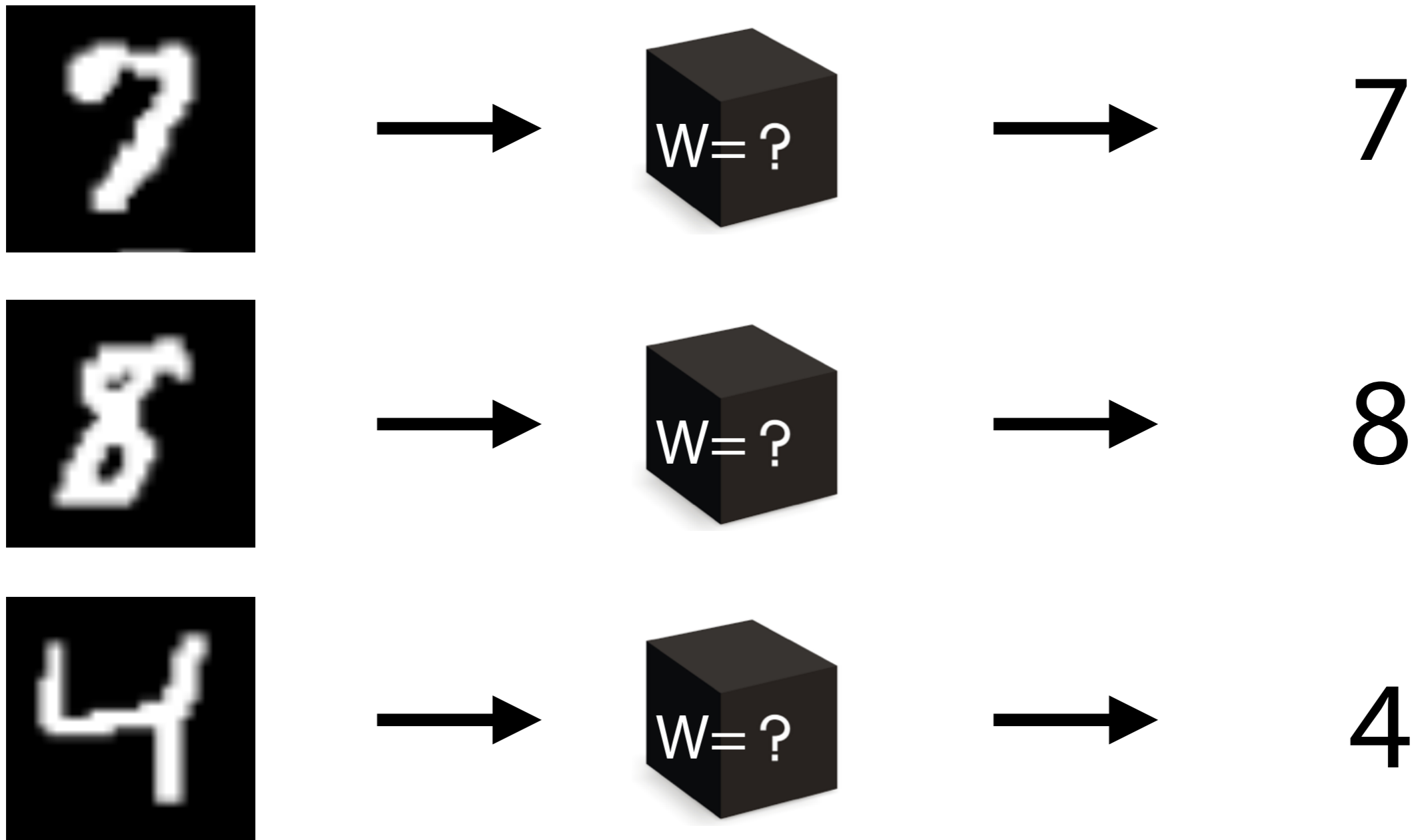
My research on human motion capture



[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.



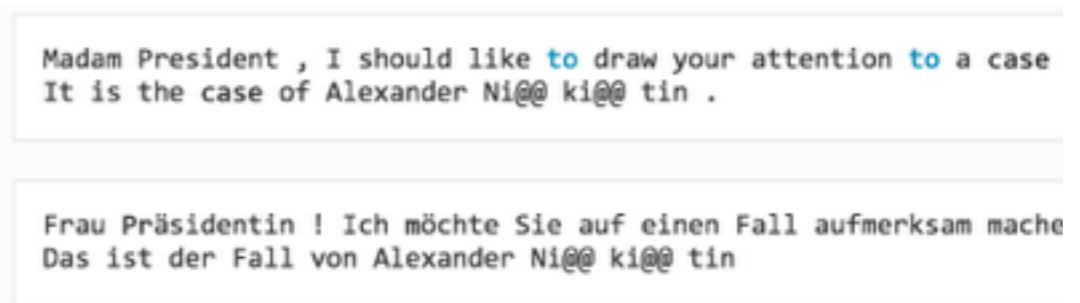
Image net
(14 million image-class pairs)



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

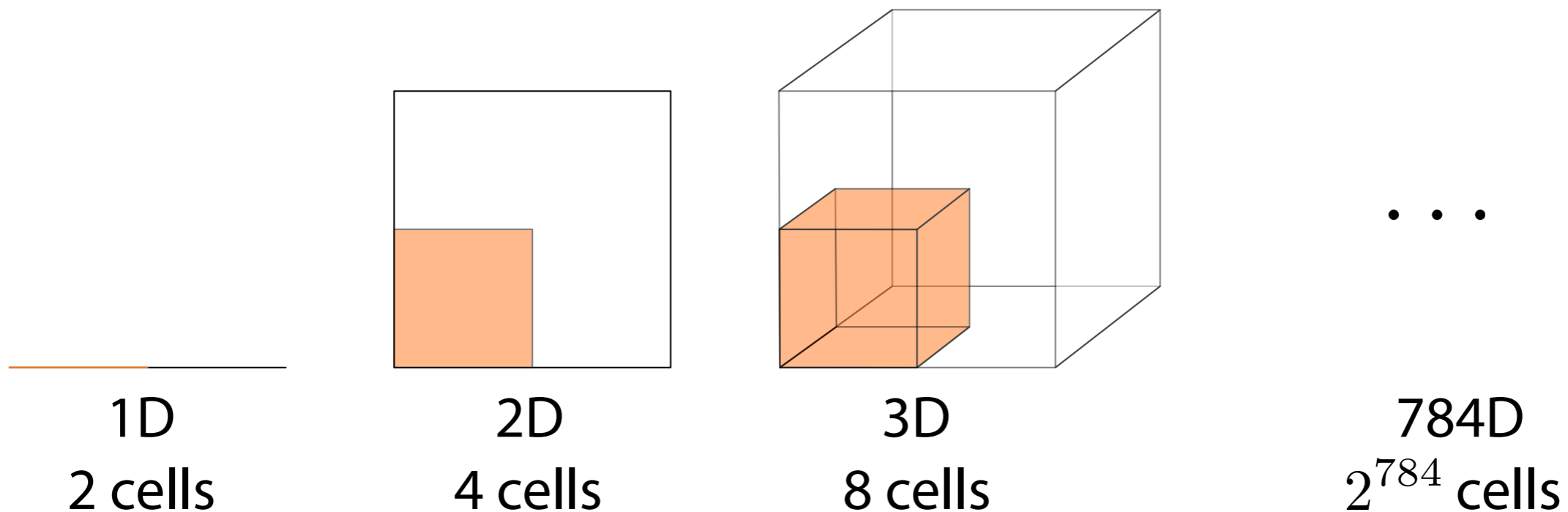
It contains 70 000 digit examples and is one of the most well-known and used test beds.



Sampling?

In high dimensions, the curse 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.

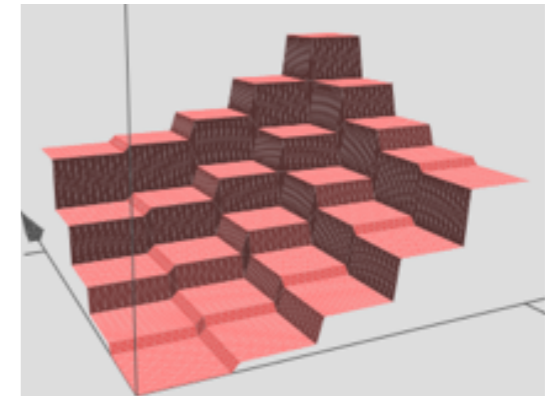
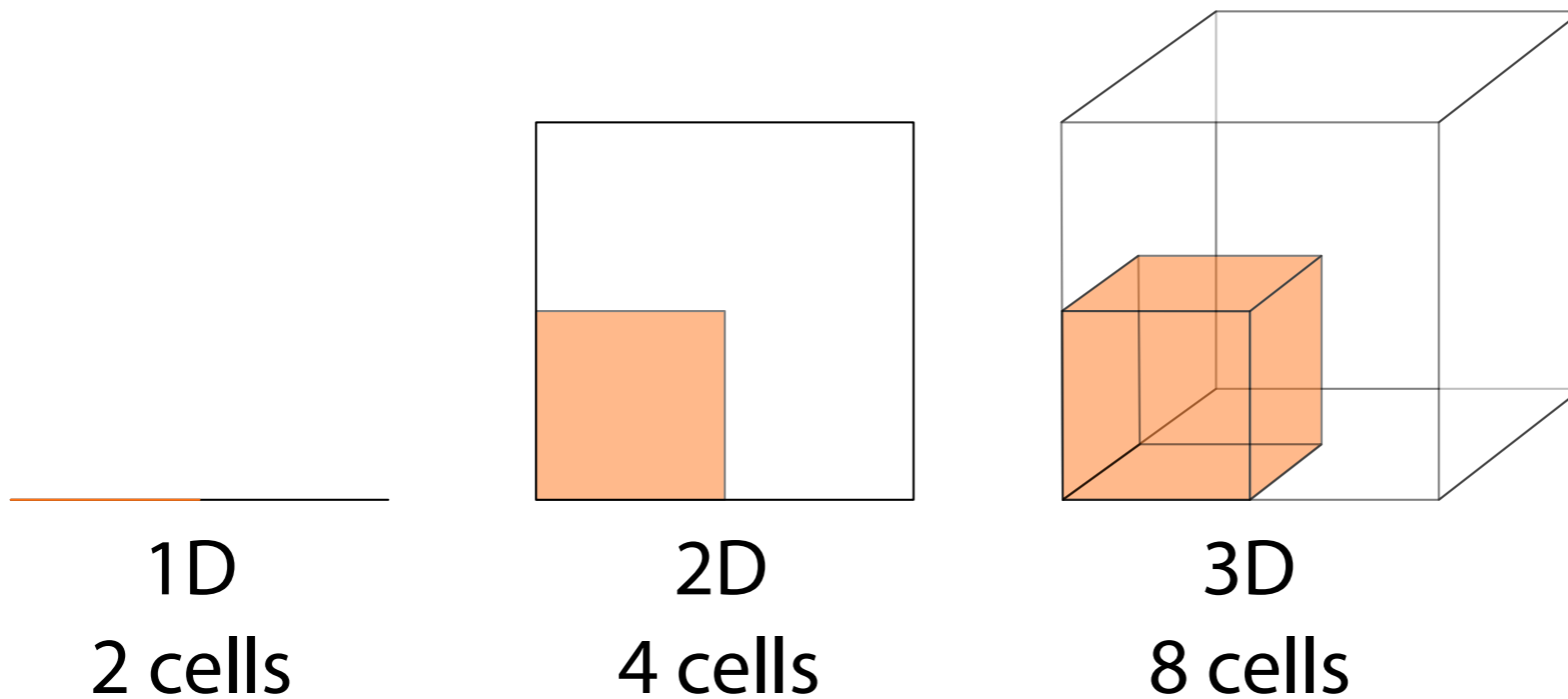
- Sampling, curse of dimensionality



Sampling?

In high dimensions, the curse 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, curse of dimensionality



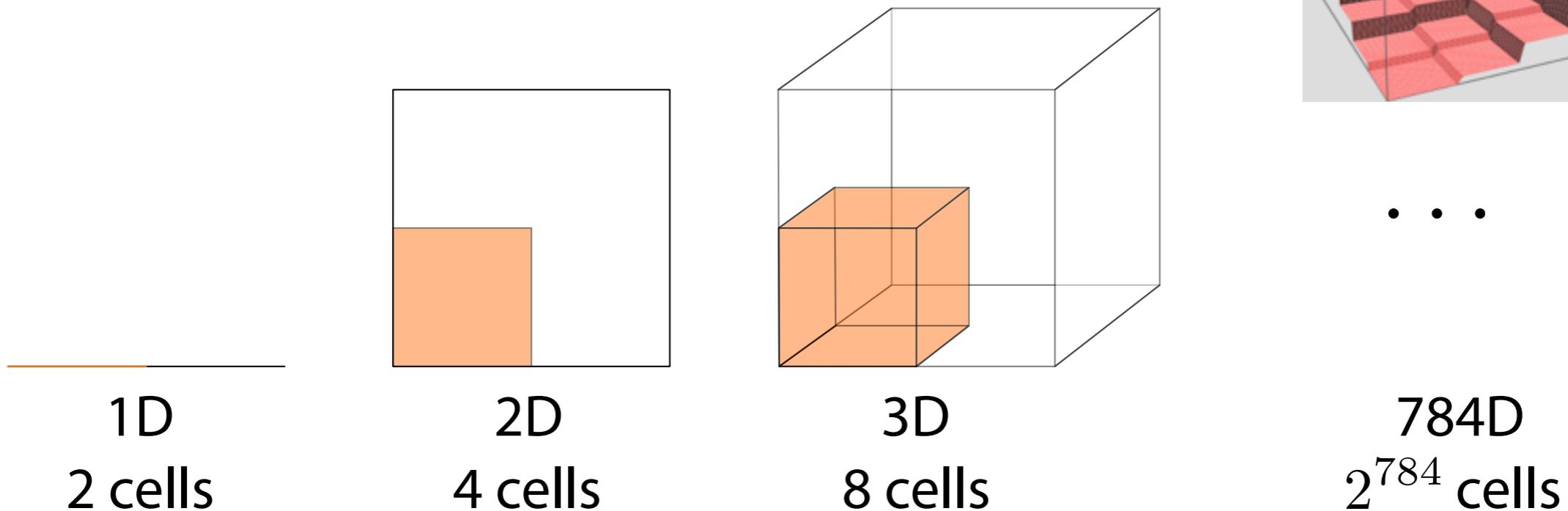
...

784D
 2^{784} cells

Sampling?

In high dimensions, the curse 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, curse of dimensionality

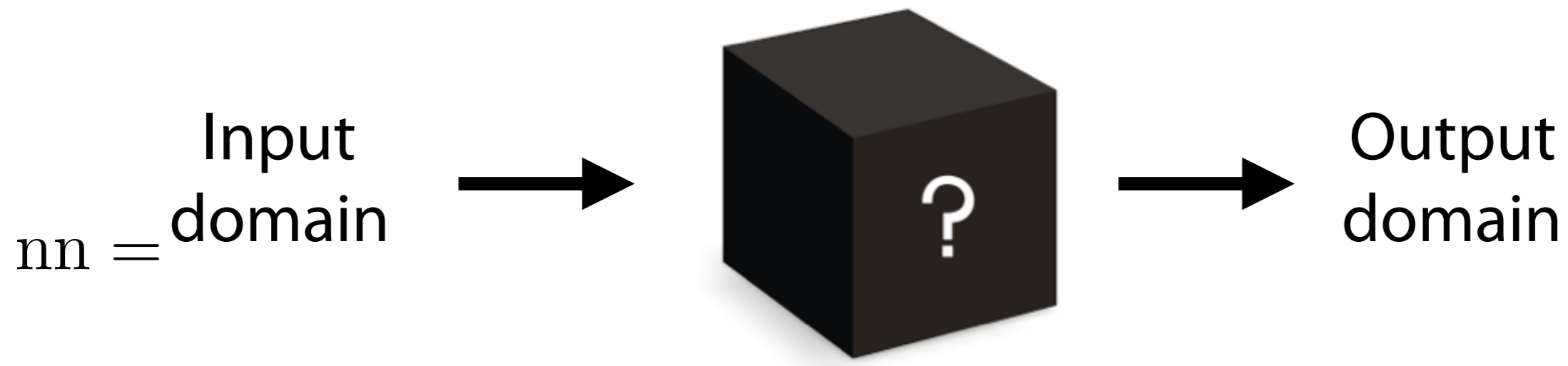


- Non-convex problem

- 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.



Function

Parameters

Linear regression

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

$$\mathbf{w} \in \mathbb{R}^n$$

Polynomial regression

$$f(x) = \mathbf{w} \cdot (1, x, x^2, \dots, x^{k-1})$$

$$\mathbf{w} \in \mathbb{R}^k$$

Neural Networks

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

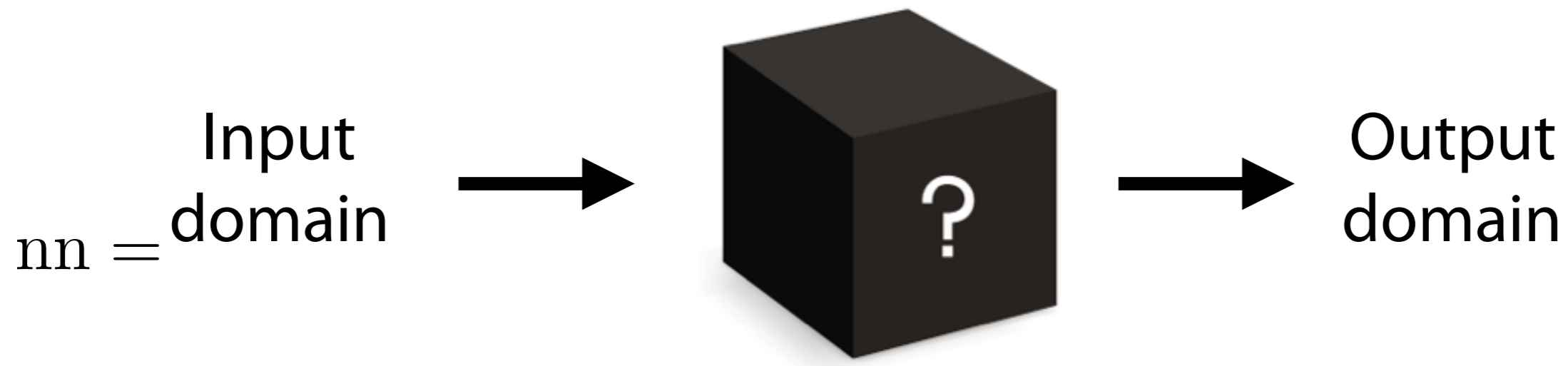
$$\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(d)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \dots, \mathbf{b}^{(d)}\}$$

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

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

Neural network — a parametric function

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



Function

Parameters

Linear regression

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

$$\mathbf{w} \in \mathbb{R}^n$$

increase

Polynomial regression

$$f(x) = \mathbf{w} \cdot (1, x, x^2, \dots, x^{k-1})$$

$$\mathbf{w} \in \mathbb{R}^k$$

Neural Networks

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

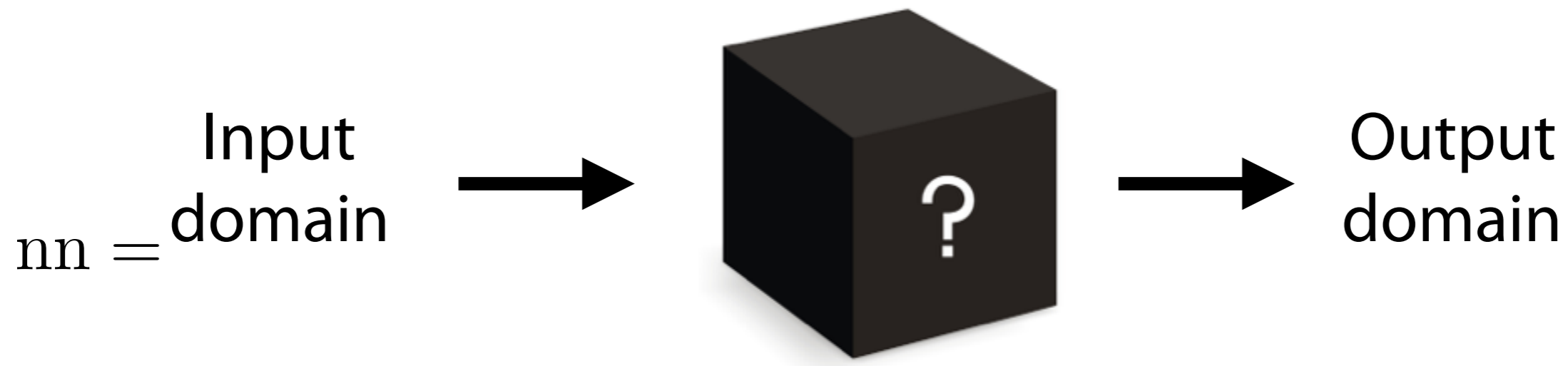
$$\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(d)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \dots, \mathbf{b}^{(d)}\}$$

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

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

Neural network — a parametric function

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



Function

Parameters

Linear regression

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

$$\mathbf{w} \in \mathbb{R}^n$$

increase

Polynomial regression

$$f(x) = \mathbf{w} \cdot (1, x, x^2, \dots, x^{k-1})$$

$$\mathbf{w} \in \mathbb{R}^k$$

increase

Neural Networks

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

$$\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(d)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \dots, \mathbf{b}^{(d)}\}$$

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

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

An optimization problem

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

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

An optimization problem

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

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

Akin to optimization problems in previous lectures

Linear least squares

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

Non-linear least squares

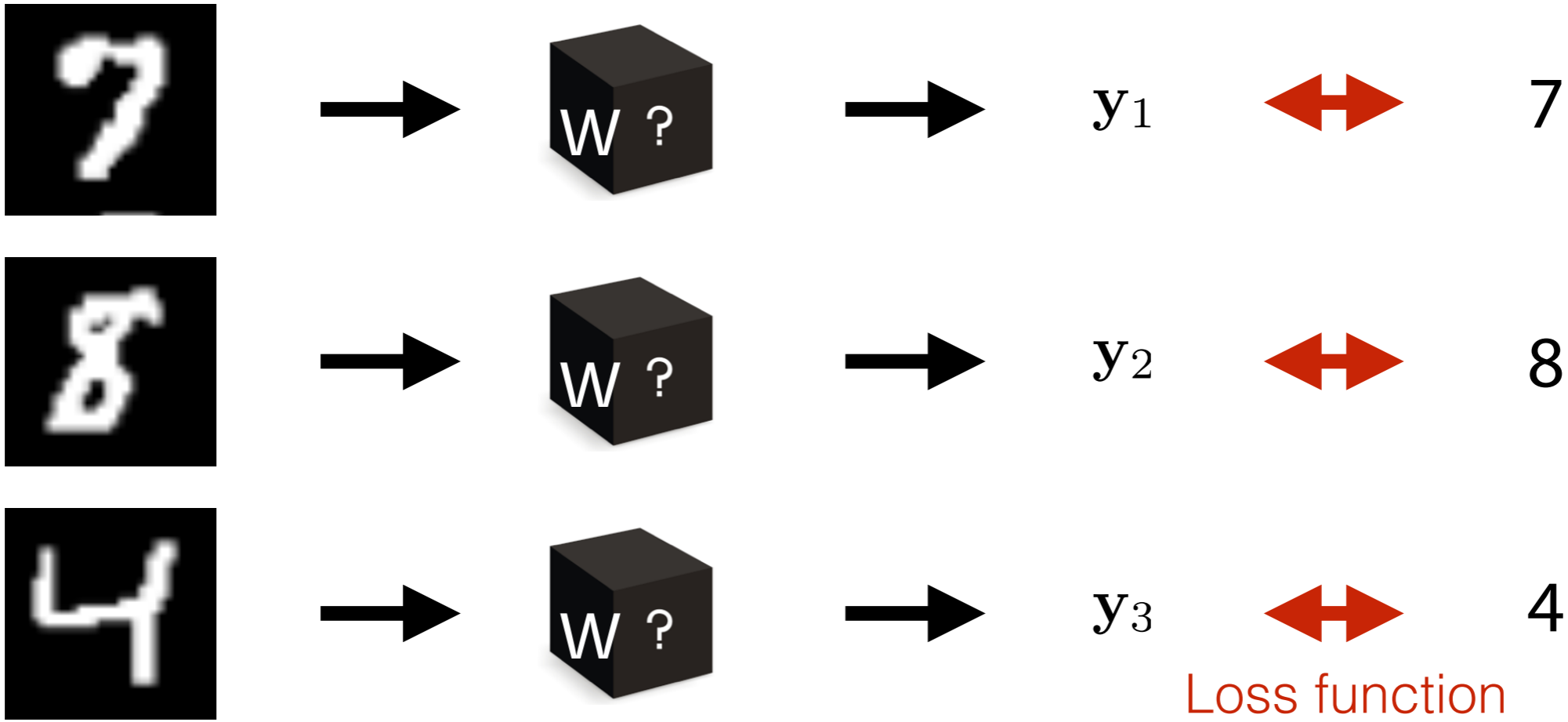
$$\min_{\mathbf{x}} \|\mathbf{f}(\mathbf{x}) - \mathbf{b}\|^2$$

An optimization problem

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

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

$$\theta = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \dots\}$$



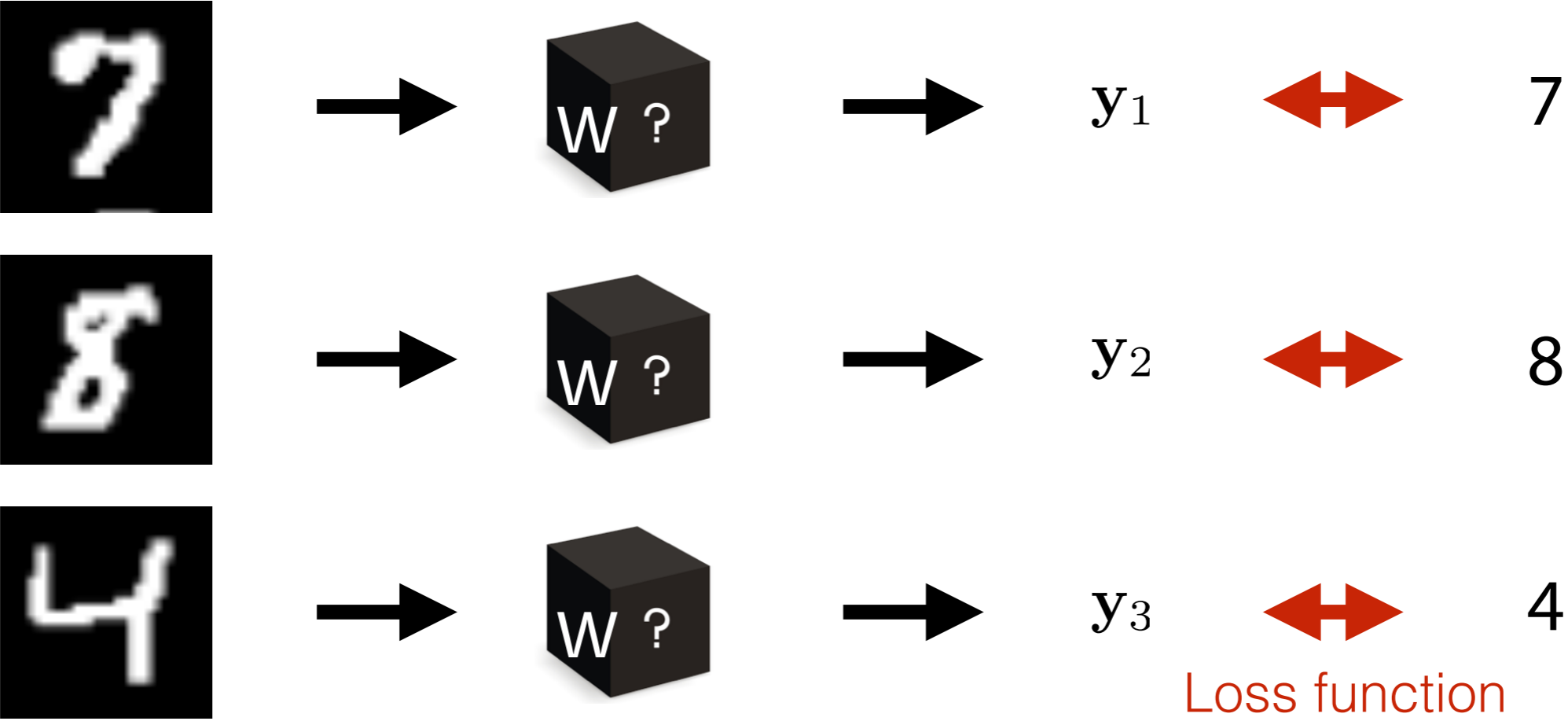
An optimization problem

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

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

Parameter

$$\theta = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \dots\}$$



Loss function
(distance metric)

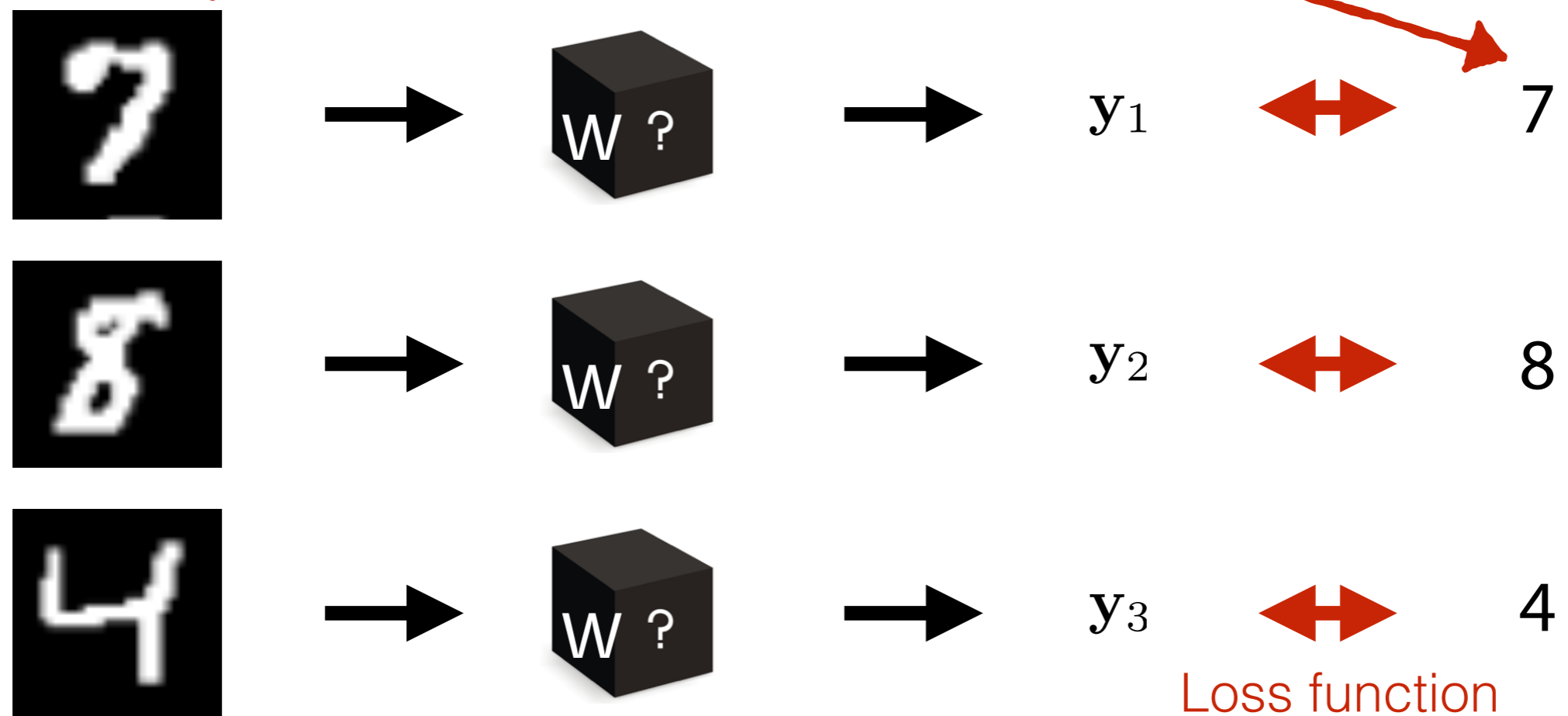
An optimization problem

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

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

Parameter $\theta = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \dots\}$

Input Dataset 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

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

Objective and loss function

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

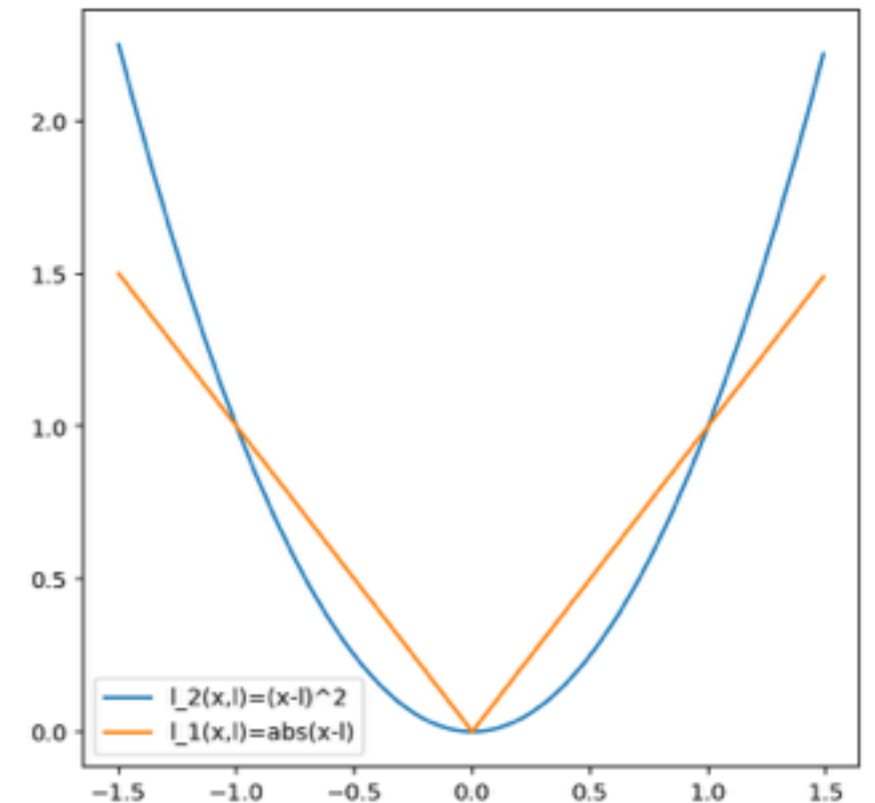
General form

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

Separable sum

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

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

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

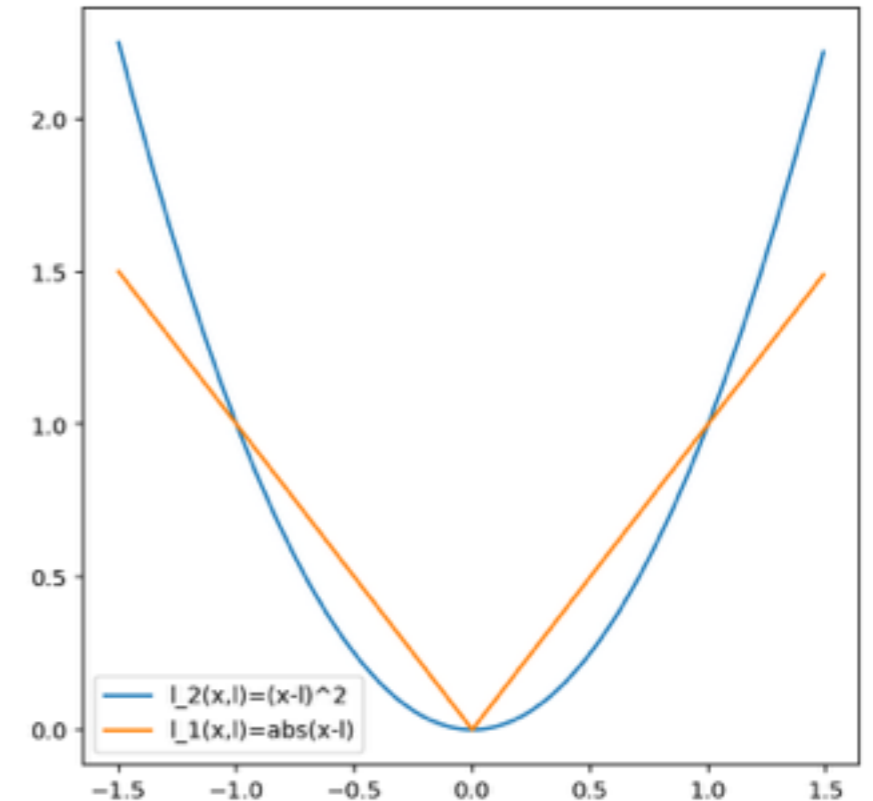
Separable sum

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

MNIST digit example

$$\begin{aligned} E(D, \theta) &= \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} (\text{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})^2 \\ &= (\text{nn}(\mathbf{7}, \theta) - 7)^2 \\ &+ (\text{nn}(\mathbf{8}, \theta) - 8)^2 \dots \end{aligned}$$

Loss functions



Quadratic loss

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

Absolute loss

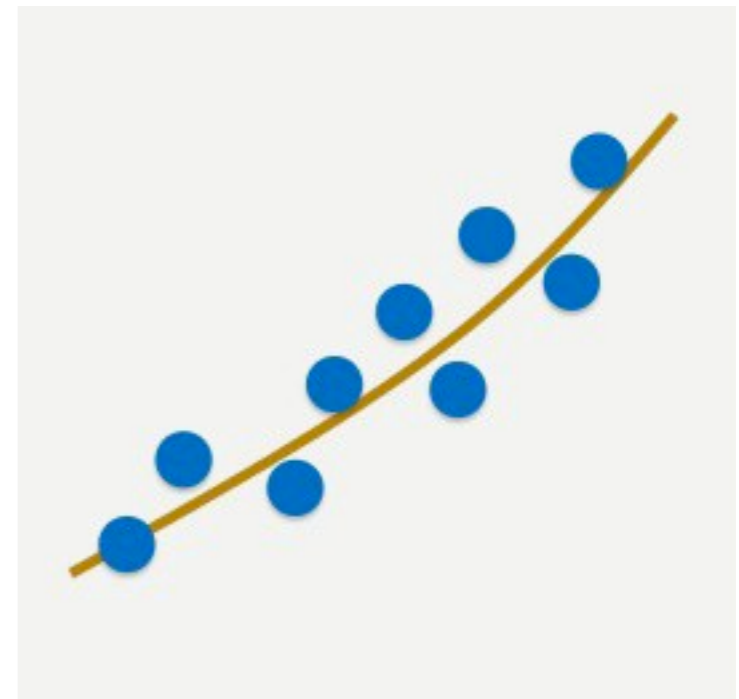
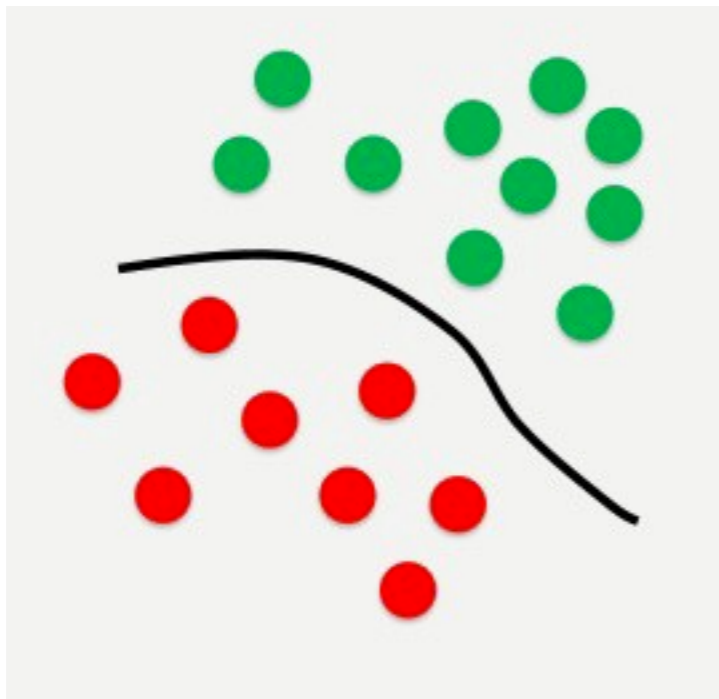
$$l_1(y, l) = |y - l|$$

Classification vs. regression

Classification



Regression

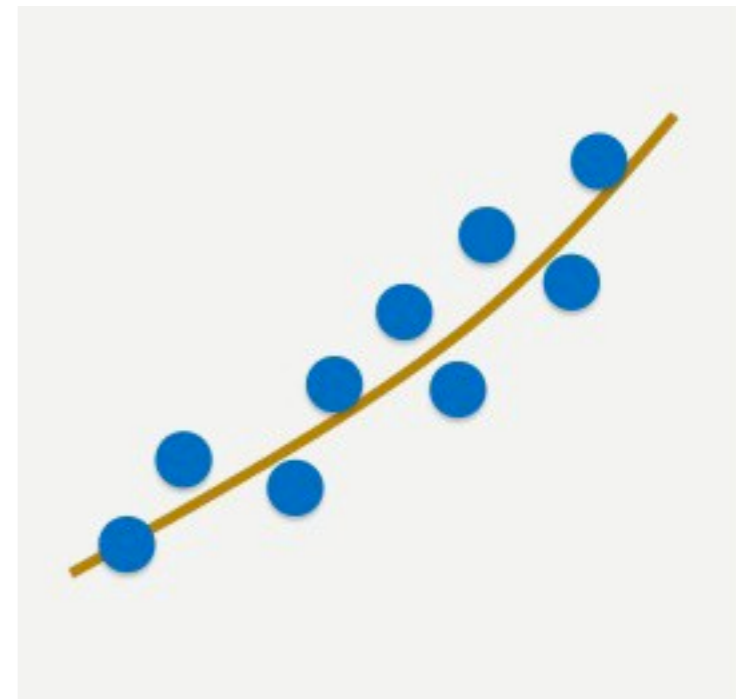
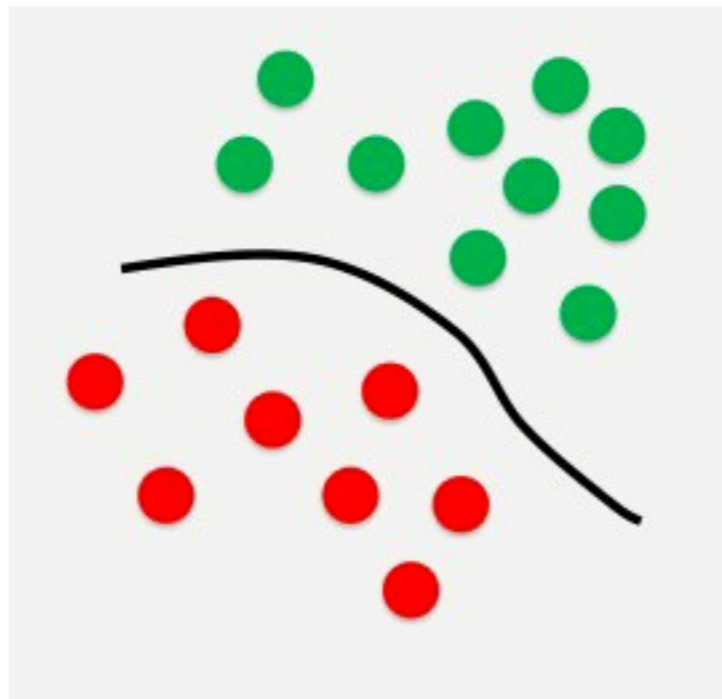
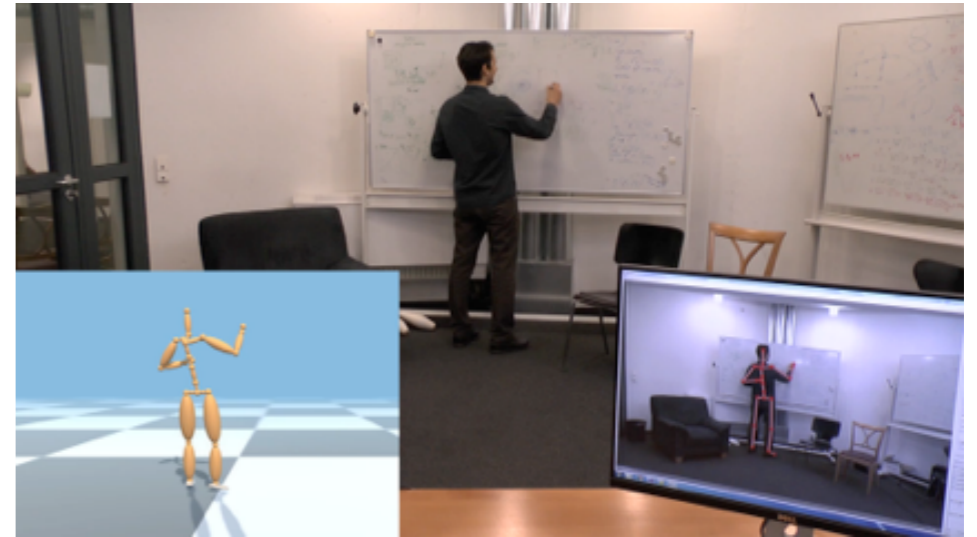


Classification vs. regression

Classification



Regression

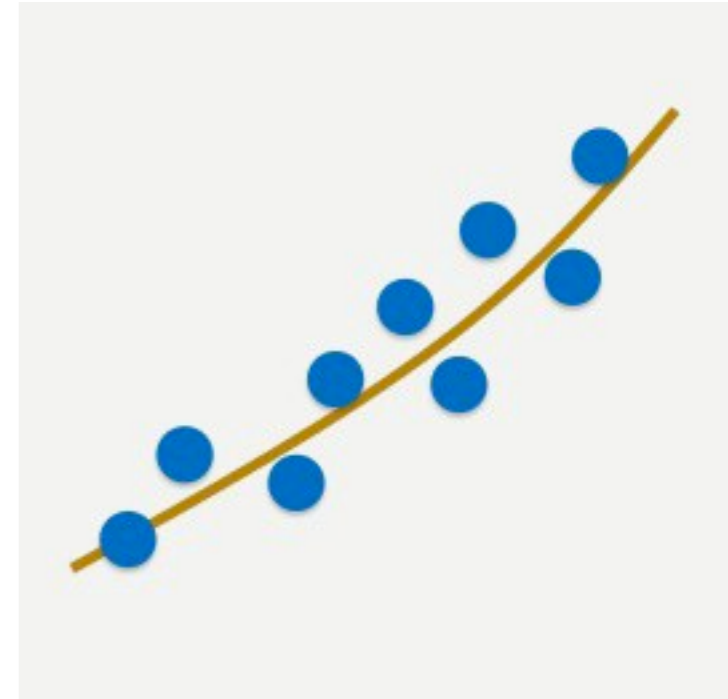


Classification vs. regression

- Regression
 - works for continuous values

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

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



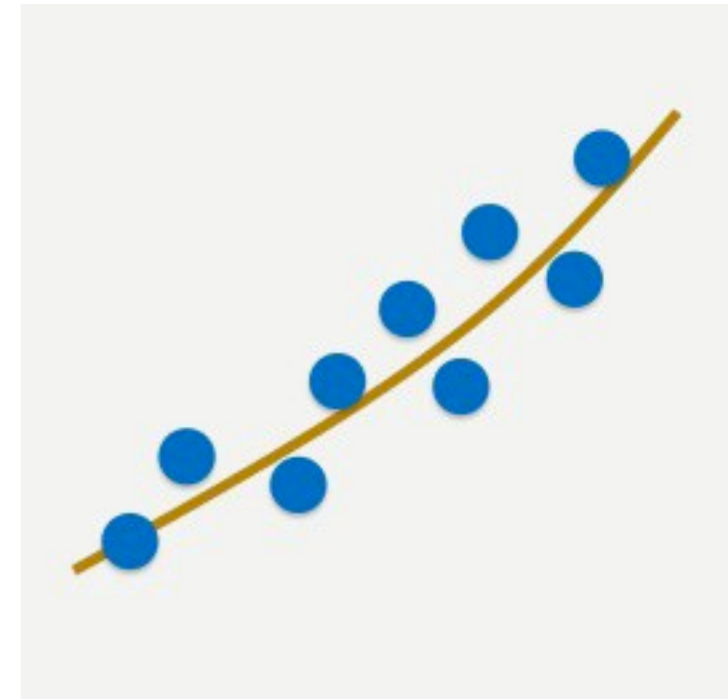
Classification vs. regression

- Regression

- works for continuous values

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

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

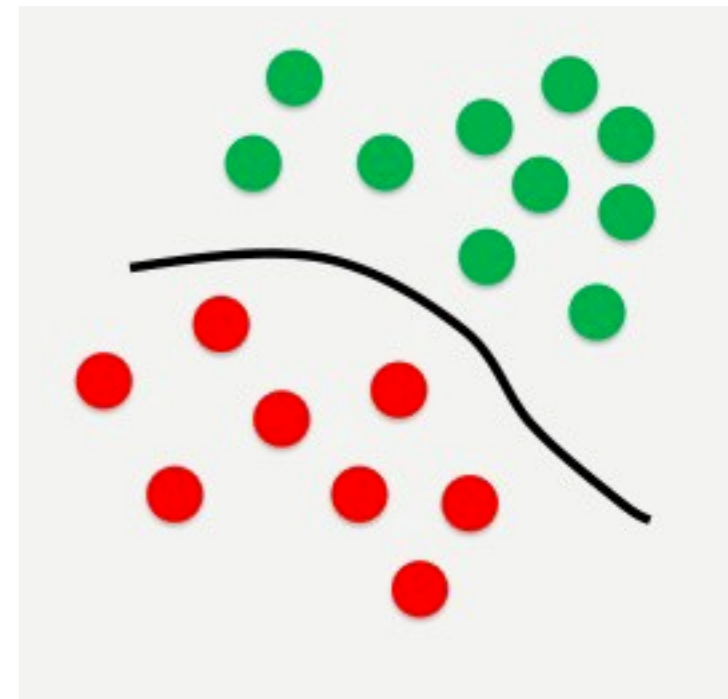


- Classification

- discrete classes
- probabilistic interpretation (probability of class)

$$\text{nn}(\mathbf{x}) \rightarrow \mathbf{y} \in [0, 1]$$

$$l_2(\mathbf{y}, \mathbf{l}) = \|\mathbf{y} - \mathbf{l}\|^2$$



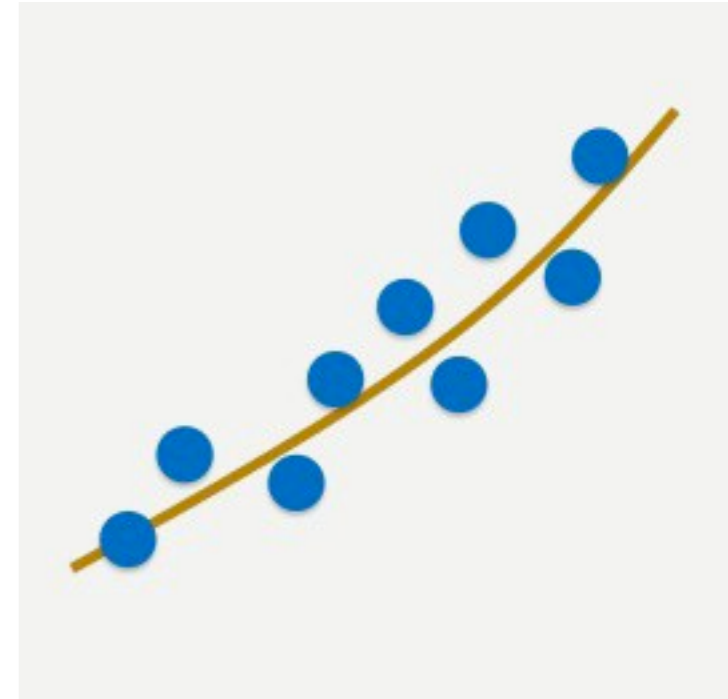
Classification vs. regression

- Regression

- works for continuous values

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

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

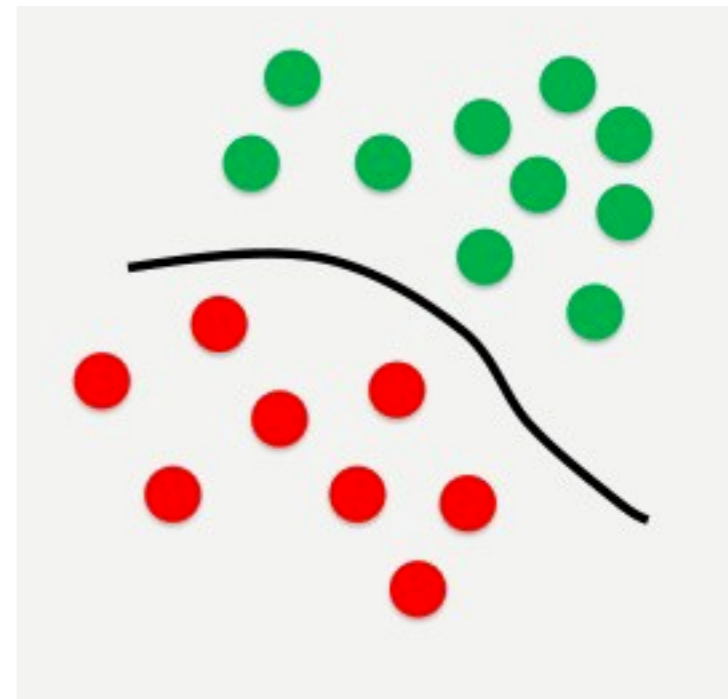
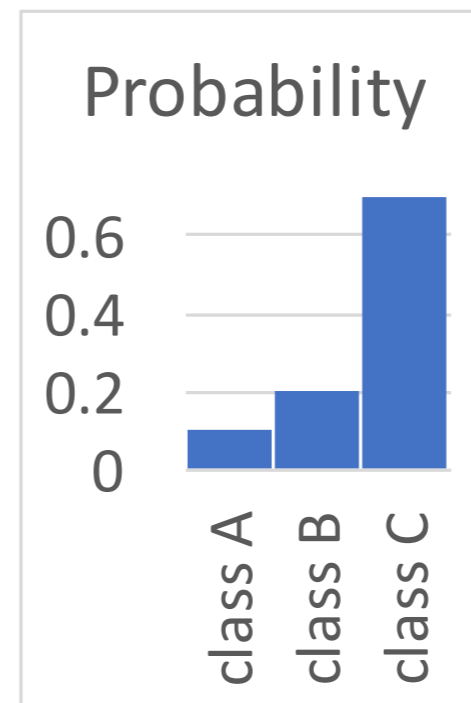


- Classification

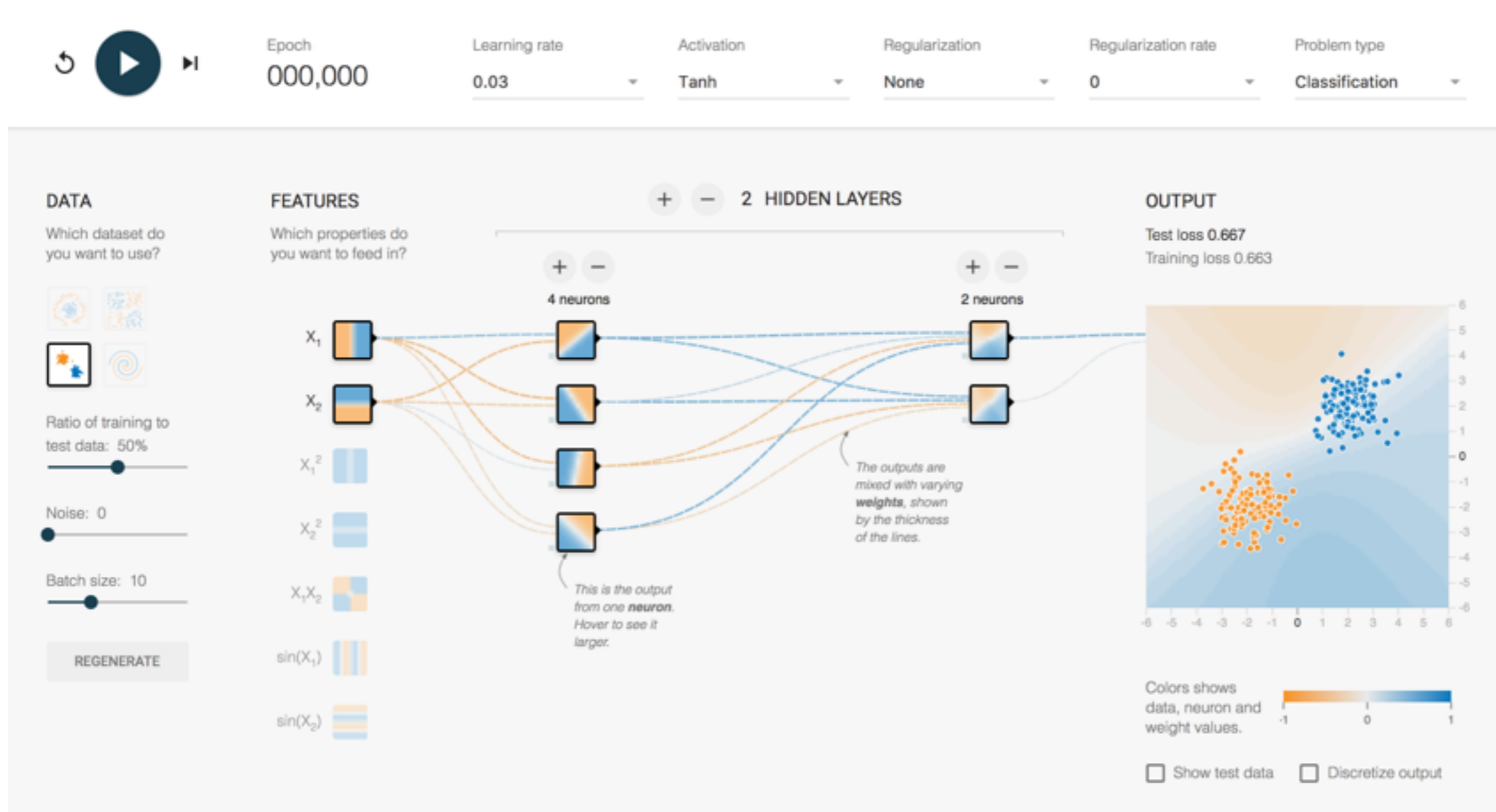
- discrete classes
- probabilistic interpretation (probability of class)

$$\text{nn}(\mathbf{x}) \rightarrow \mathbf{y} \in [0, 1]$$

$$l_2(\mathbf{y}, \mathbf{l}) = \|\mathbf{y} - \mathbf{l}\|^2$$



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)



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)



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



8

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)



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



8

Label
 $\in \mathbb{R}$

2) Dataset D



[MNIST]

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

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

Training a NN

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

1) Representation (i/o domain)



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



8

Label
 $\in \mathbb{R}$

2) Dataset D

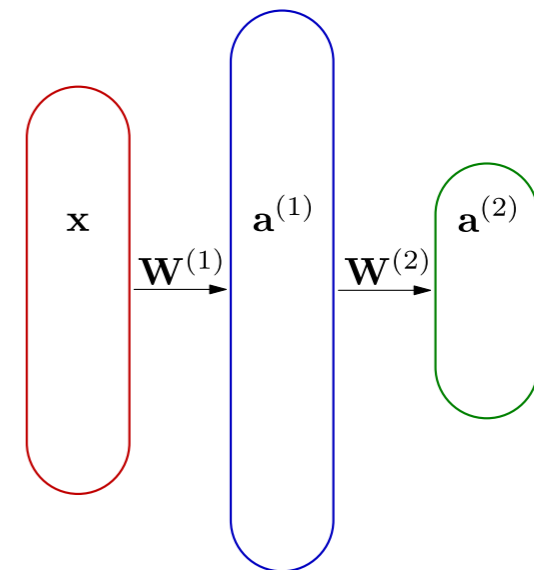


[MNIST]

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

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

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



Training a NN

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

1) Representation (i/o domain)



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



8

Label
 $\in \mathbb{R}$

2) Dataset D

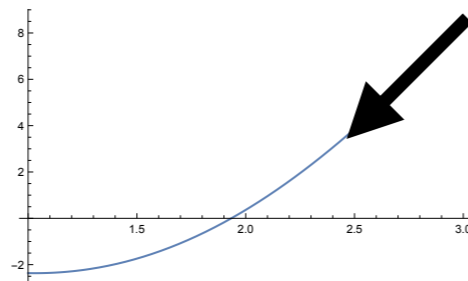


[MNIST]

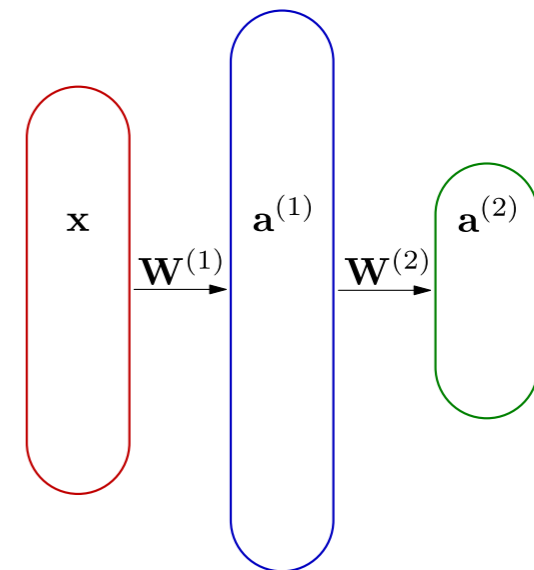
3) Objective $\arg \min_W E(D, \theta)$

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

5) Solver

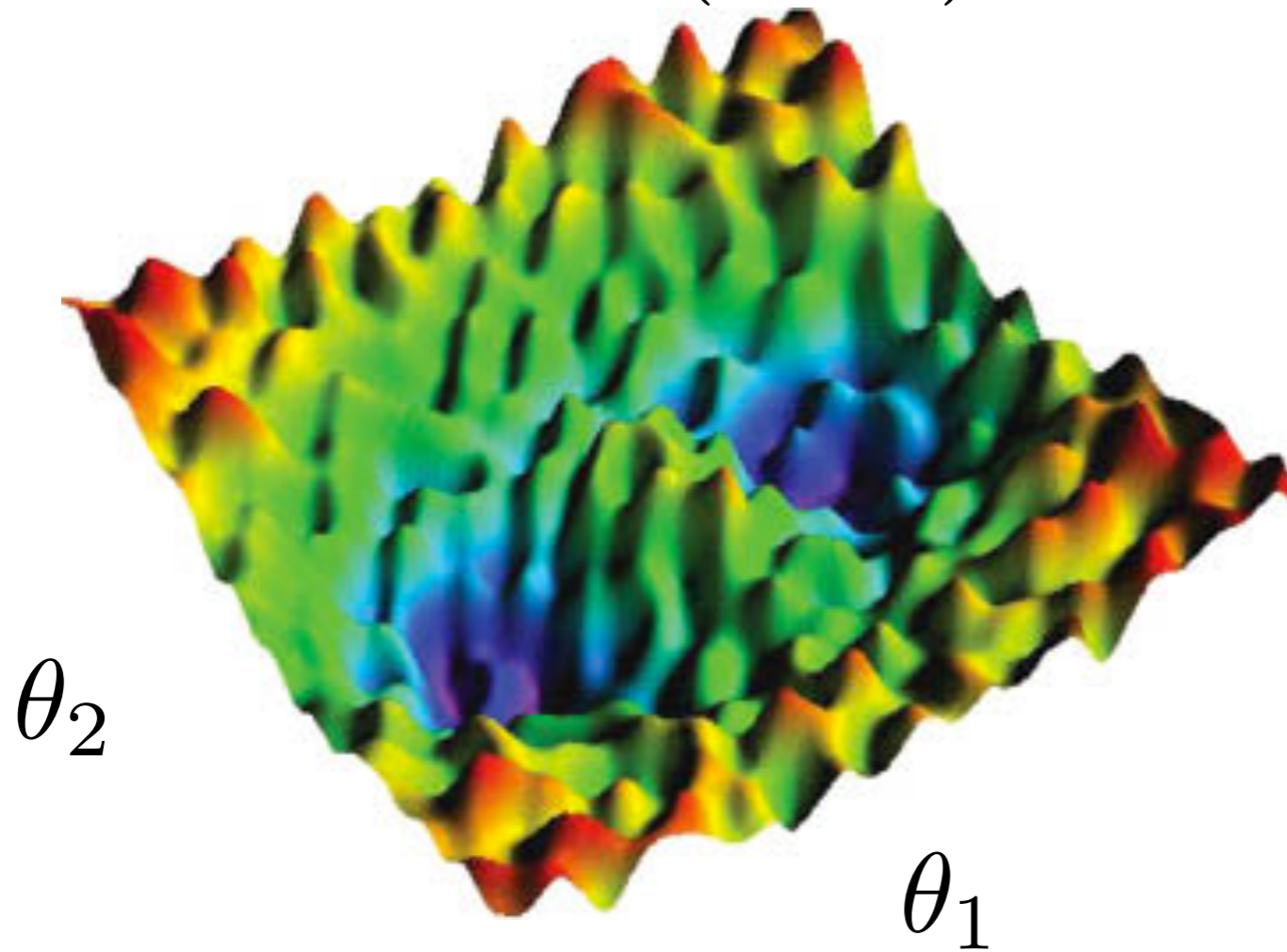


4) Model $\text{nn}(\mathbf{x}, \theta)$

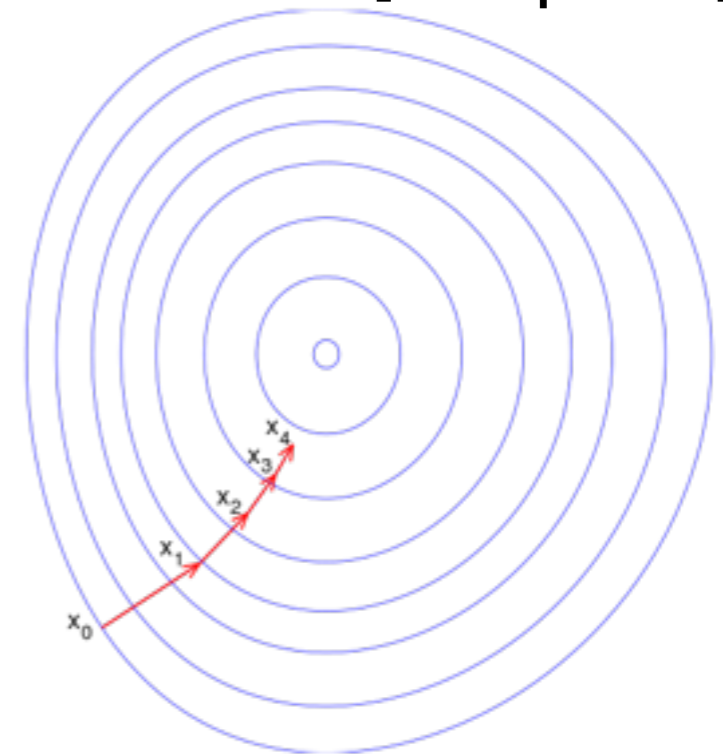


Iterative solver

$$E(D, \theta)$$



[Wikipedia]



Stochastic gradient descent

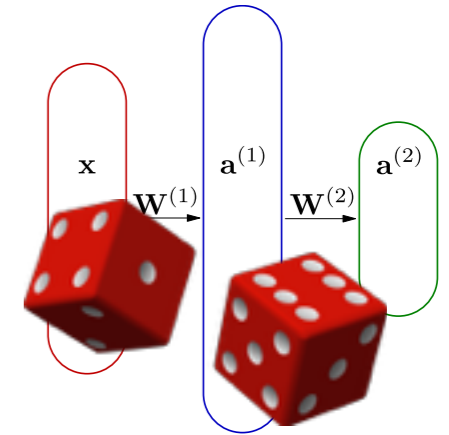
Iteratively optimizing over subsets of the dataset is efficient and works surprisingly well. At each iteration a new subset is chosen to decent closer to the minimum of the full energy.

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

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

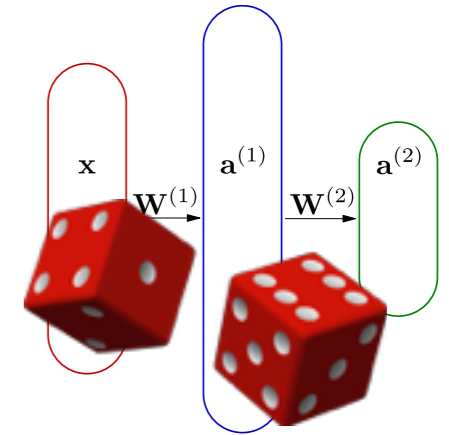


Stochastic gradient descent

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.

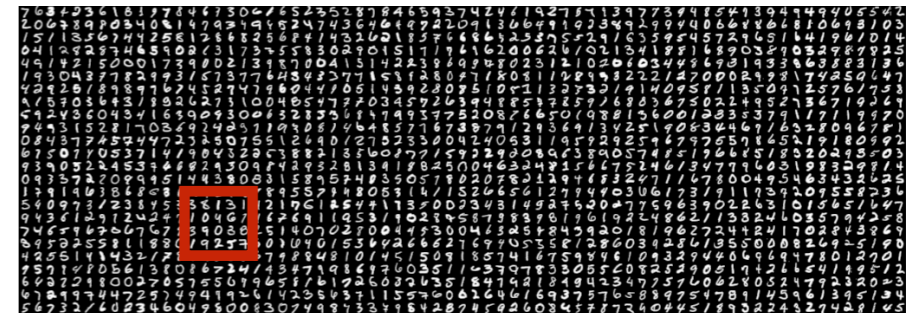
- Start with a random initialization

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



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

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

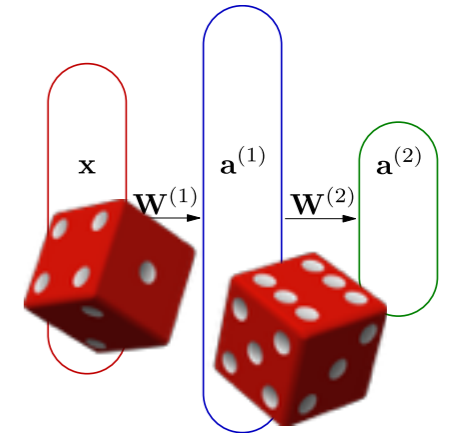


Stochastic gradient descent

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.

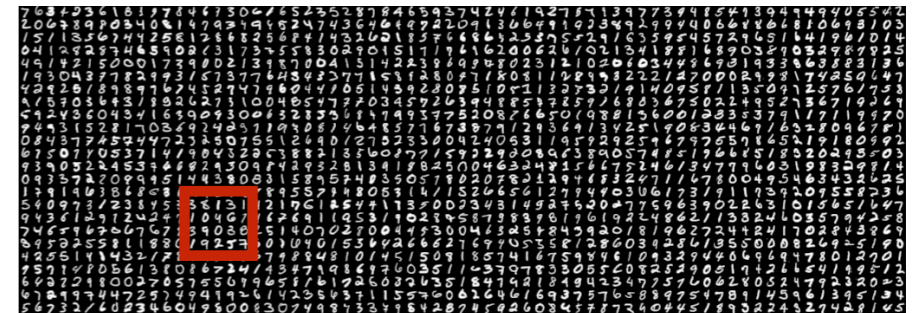
- Start with a random initialization

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



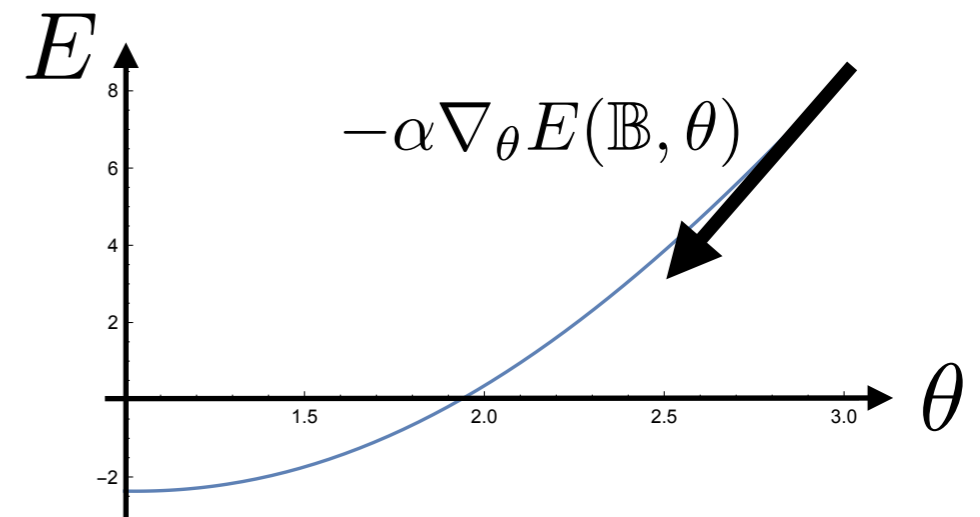
- Select a training subset (minibatch, 1-100 examples)

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



- Compute gradient with respect to parameters

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

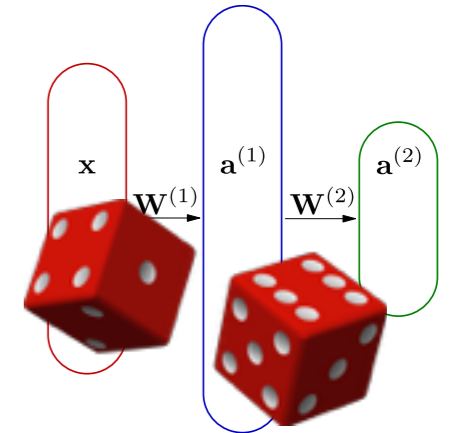


Stochastic gradient descent

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)

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

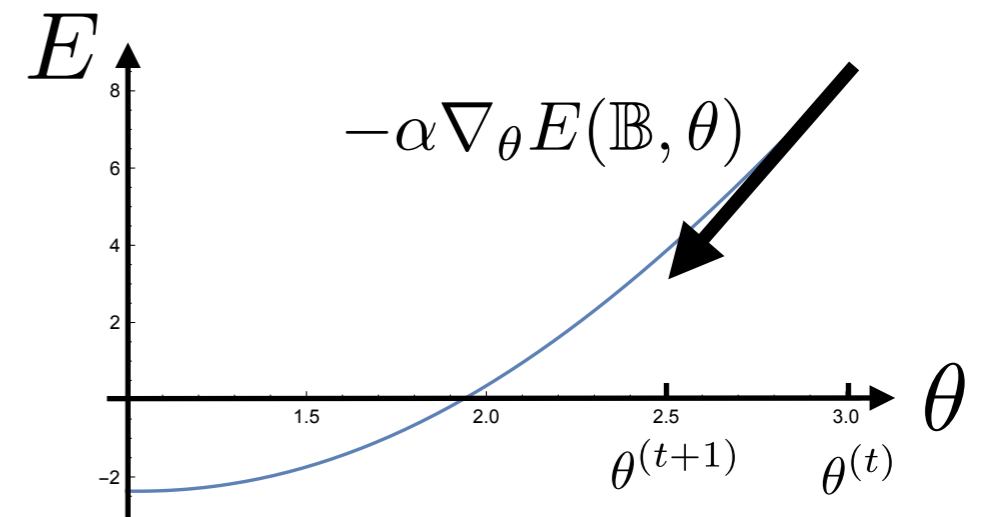


- Compute gradient with respect to parameters

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

- Update weights with learning rate α

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

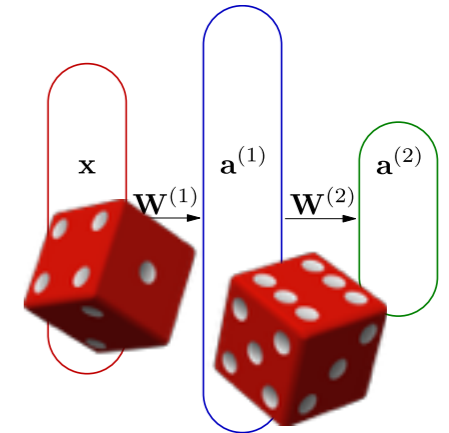


Stochastic gradient descent

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.

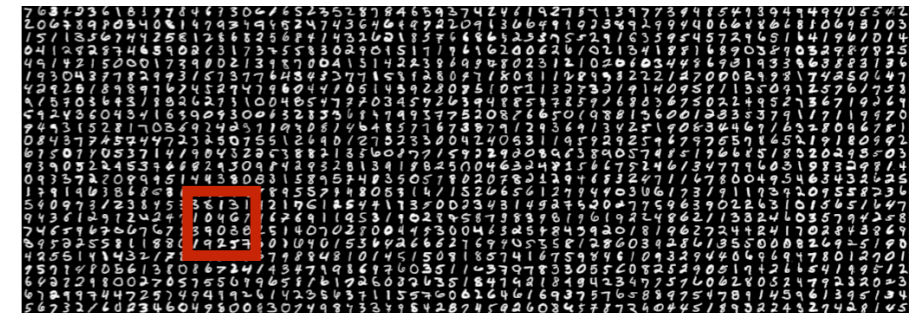
- Start with a random initialization

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



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

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



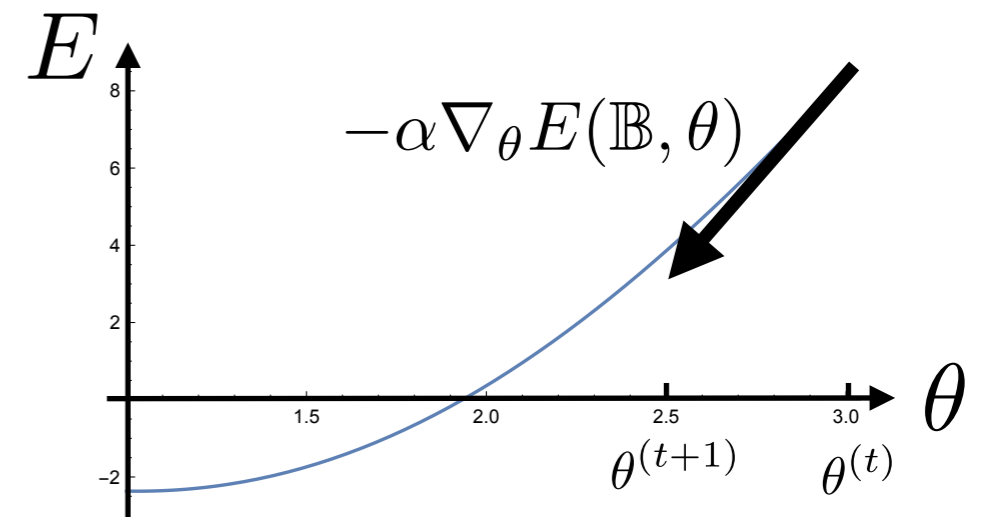
- Compute gradient with respect to parameters

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

- 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

$$\begin{aligned} E(D, \theta) &= \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} (\text{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})^2 \\ &= (\text{nn}(\mathbf{7}, \theta) - 7)^2 \\ &\quad + (\text{nn}(\mathbf{8}, \theta) - 8)^2 \dots \end{aligned}$$



Simplistic stochastic gradient descent

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

Full energy

$$\begin{aligned} E(D, \theta) &= \sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in D} (\text{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})^2 \\ &= (\text{nn}(\mathbf{7}, \theta) - 7)^2 \\ &\quad + (\text{nn}(\mathbf{8}, \theta) - 8)^2 \dots \end{aligned}$$

Approximation at iteration i

$$\begin{aligned} E(D, \theta) &\approx E((\mathbf{x}^{(i)}, y^{(i)}), \theta) \\ E((\mathbf{x}^{(i)}, y^{(i)}), \theta) &= (\text{nn}(\mathbf{x}^{(i)}, \theta) - y^{(i)})^2 \\ &= (\text{nn}(\mathbf{7}, \theta) - 7)^2 \end{aligned}$$



Differentiation

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

NN function

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

$$\text{nn} = \boxed{\mathbf{w}^{(2)}} h \left(\boxed{\mathbf{w}^{(1)}} \boxed{\mathbf{x}} + \boxed{\mathbf{b}^{(1)}} \right) + \boxed{\mathbf{b}^{(2)}}$$

Toy example, a scalar NN

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

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

Differentiation

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

NN function

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

$$\text{nn} = \begin{bmatrix} \mathbf{w}^{(2)} \end{bmatrix} h \left(\begin{bmatrix} \mathbf{w}^{(1)} \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix} + \begin{bmatrix} \mathbf{b}^{(1)} \end{bmatrix} \right) + \begin{bmatrix} \mathbf{b}^{(2)} \end{bmatrix}$$

NN Jacobian matrix

$$J_{\mathbf{W}^{(1)}}^{\text{nn}}(\mathbf{x}) = \begin{bmatrix} J_{\mathbf{x}}^{\text{linear}} \end{bmatrix} \begin{bmatrix} J_{\mathbf{x}}^h \end{bmatrix} \begin{bmatrix} J_{\mathbf{W}}^{\text{linear}} \end{bmatrix}$$

Jacobian matrices

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

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

Jacobian matrices

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

function to differentiate...

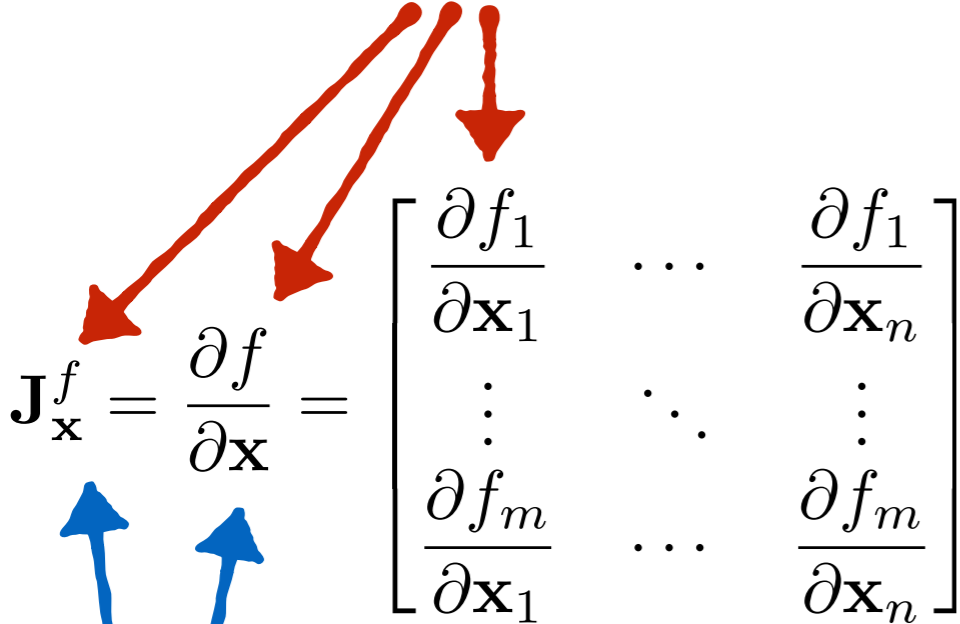
$$\mathbf{J}_x^f = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \dots & \frac{\partial f_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathbf{x}_1} & \dots & \frac{\partial f_m}{\partial \mathbf{x}_n} \end{bmatrix}$$

...with respect to \mathbf{x}

Jacobian matrices

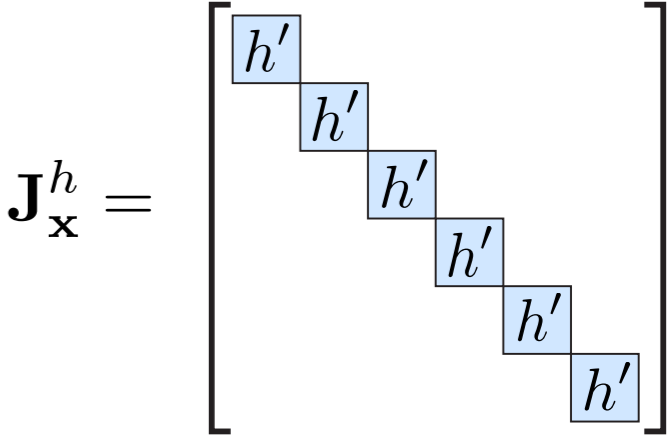
Two of the three involved Jacobian matrix types have sparse structure, we exploit that later.

function to differentiate...

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


...with respect to x

$$h = \text{relu}(\mathbf{x}) \longrightarrow$$

$$\mathbf{J}_x^h = \begin{bmatrix} h' & & & & \\ & h' & & & \\ & & h' & & \\ & & & h' & \\ & & & & h' \end{bmatrix}$$


Jacobian matrices

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

function to differentiate...

$$\mathbf{J}_x^f = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

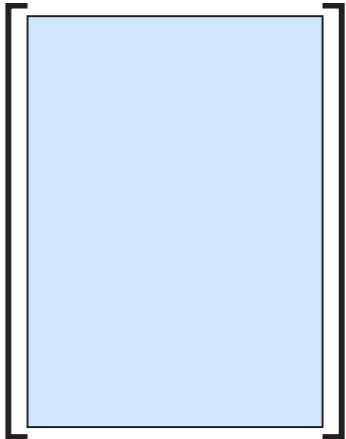
...with respect to \mathbf{x}

$$h = \text{relu}(\mathbf{x}) \longrightarrow$$

$$\mathbf{J}_x^h = \begin{bmatrix} h' & & & & \\ & h' & & & \\ & & h' & & \\ & & & h' & \\ & & & & h' \end{bmatrix}$$

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

$$\mathbf{J}_x^{\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} =$$



Jacobi matrices

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

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

$$\mathbf{J}_{\mathbf{b}}^{\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} = \begin{bmatrix} \blacksquare & & & & & \\ & \blacksquare & & & & \\ & & \blacksquare & & & \\ & & & \blacksquare & & \\ & & & & \blacksquare & \\ & & & & & \blacksquare \end{bmatrix}$$

Jacobi matrices

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

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

$$\mathbf{J}_{\mathbf{b}}^{\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} = \begin{bmatrix} \square & & & & & \\ & \square & & & & \\ & & \square & & & \\ & & & \square & & \\ & & & & \square & \\ & & & & & \square \end{bmatrix}$$

$$\mathbf{J}_{\mathbf{W}}^{\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} = \begin{bmatrix} \mathbf{X} & & & & & \\ & \mathbf{X} & & & & \\ & & \ddots & & & \\ & & & \mathbf{X} & & \\ & & & & \mathbf{X} & \end{bmatrix}$$

Jacobi matrices

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

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

$$\mathbf{J}_{\mathbf{b}}^{\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} = \begin{bmatrix} \square & & & & & \\ & \square & & & & \\ & & \square & & & \\ & & & \square & & \\ & & & & \square & \\ & & & & & \square \end{bmatrix}$$

$$\mathbf{J}_{\mathbf{W}}^{\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} = \begin{bmatrix} \mathbf{X} & & & & \\ & \mathbf{X} & & & \\ & & \ddots & & \\ & & & \mathbf{X} & \\ & & & & \mathbf{X} \end{bmatrix}$$

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

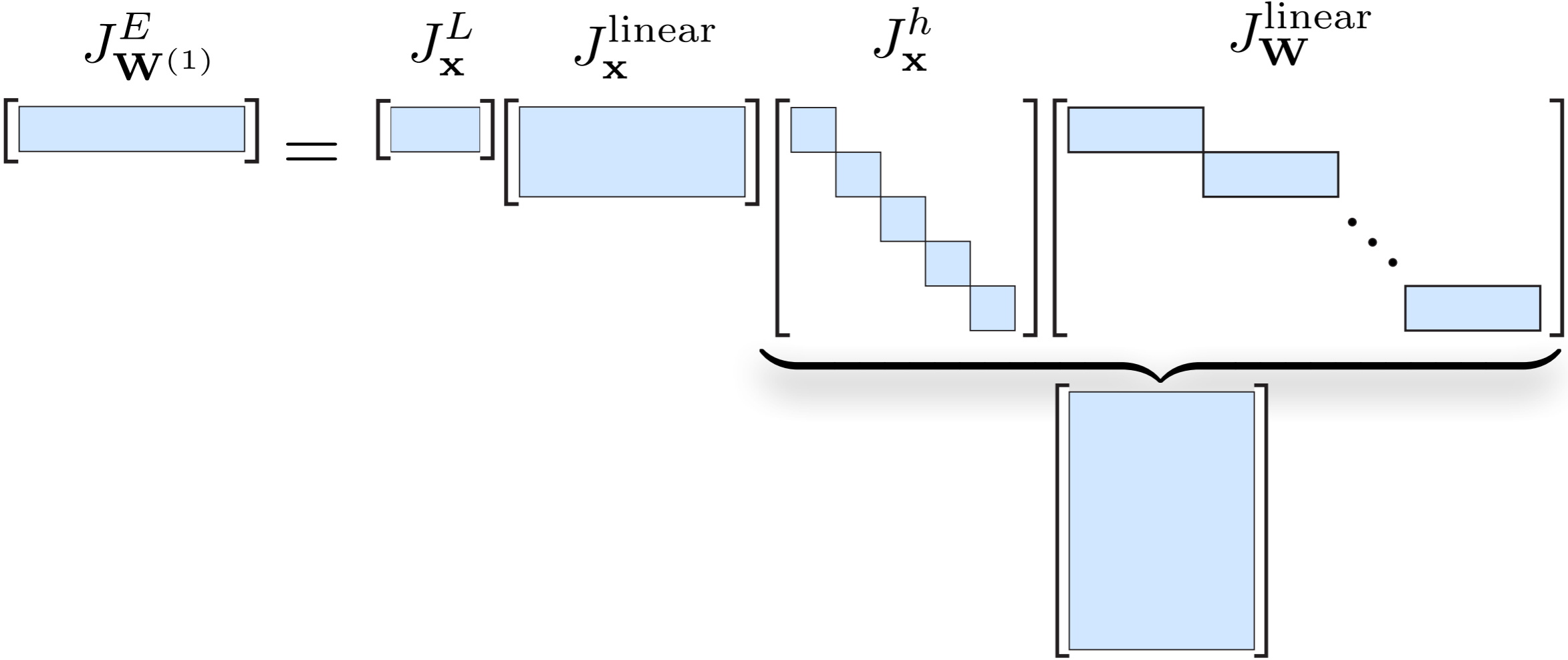
Automatic differentiation — forward mode

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.

$$J_{\mathbf{W}^{(1)}}^E = J_{\mathbf{x}}^L J_{\mathbf{x}}^{\text{linear}} J_{\mathbf{x}}^h J_{\mathbf{W}}^{\text{linear}}$$

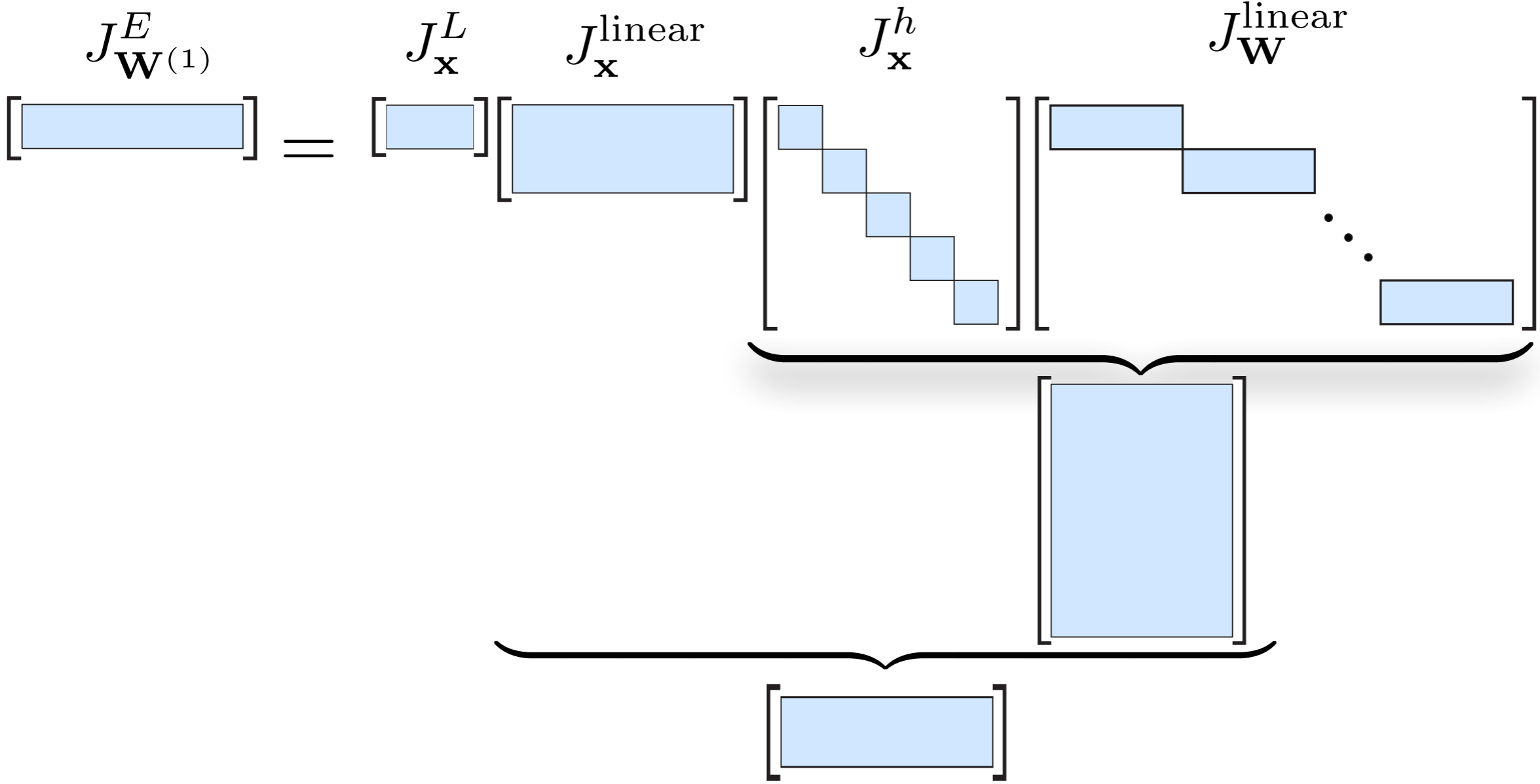
Automatic differentiation — forward mode

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.



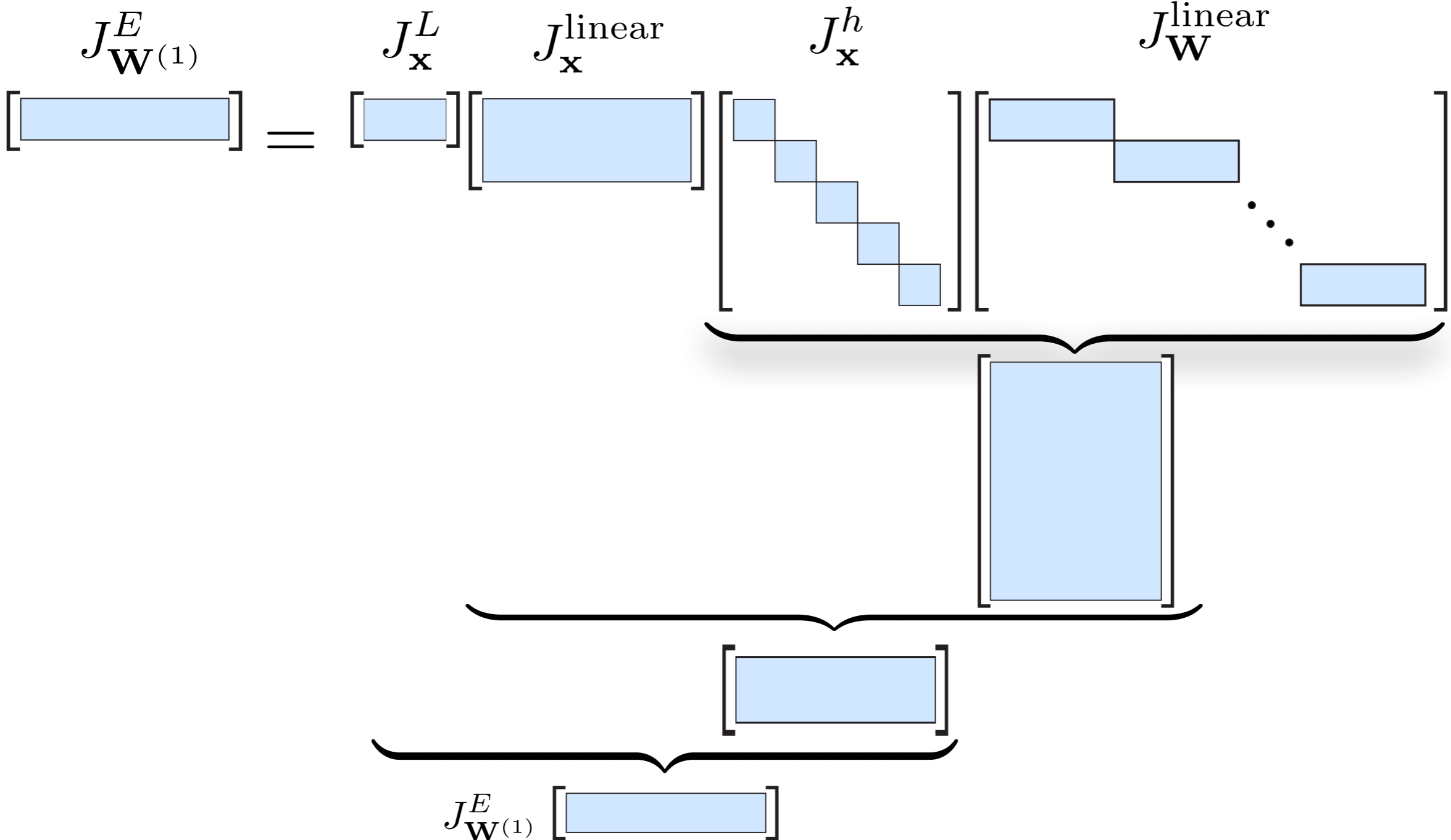
Automatic differentiation — forward mode

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.



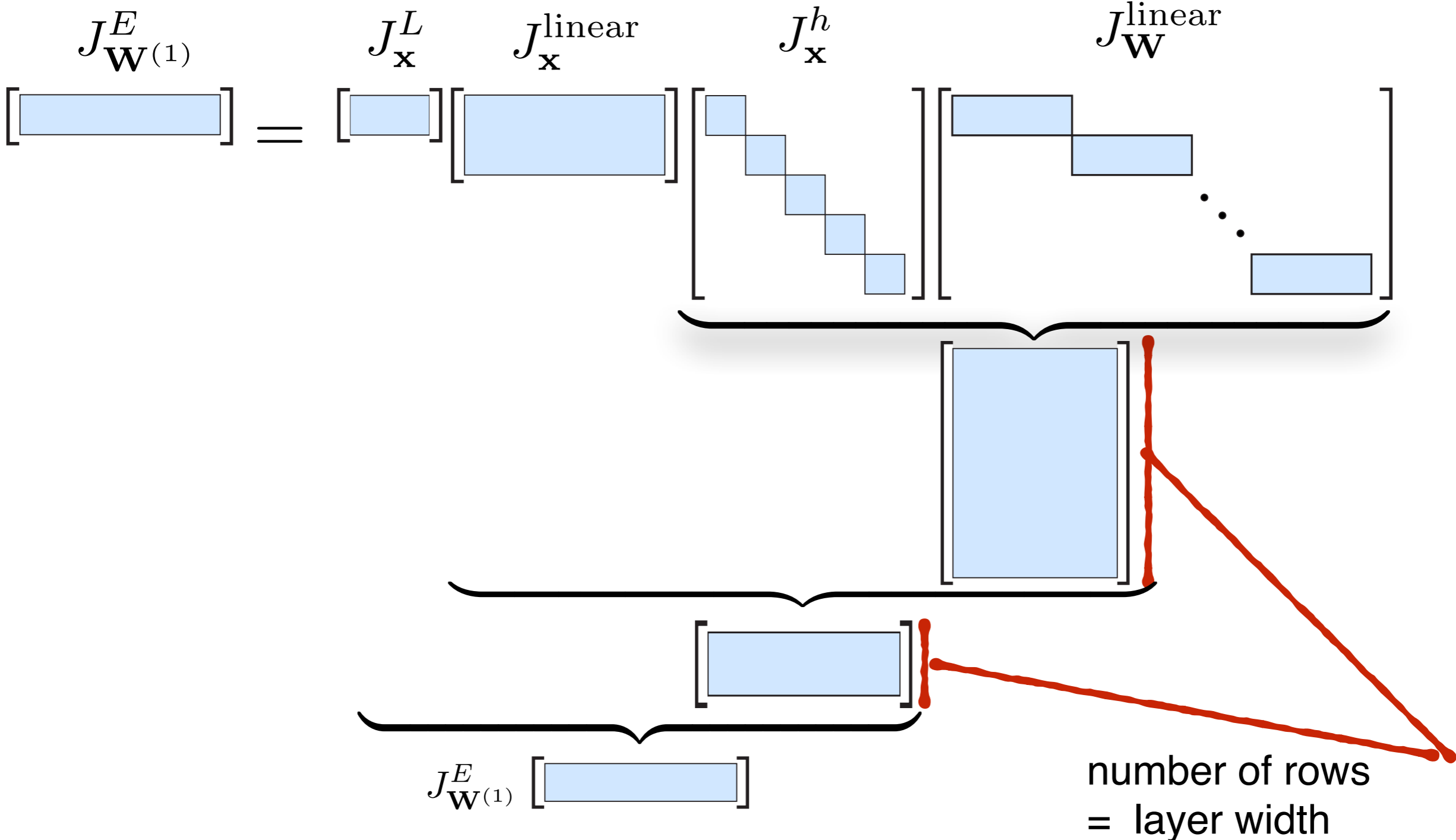
Automatic differentiation — forward mode

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.



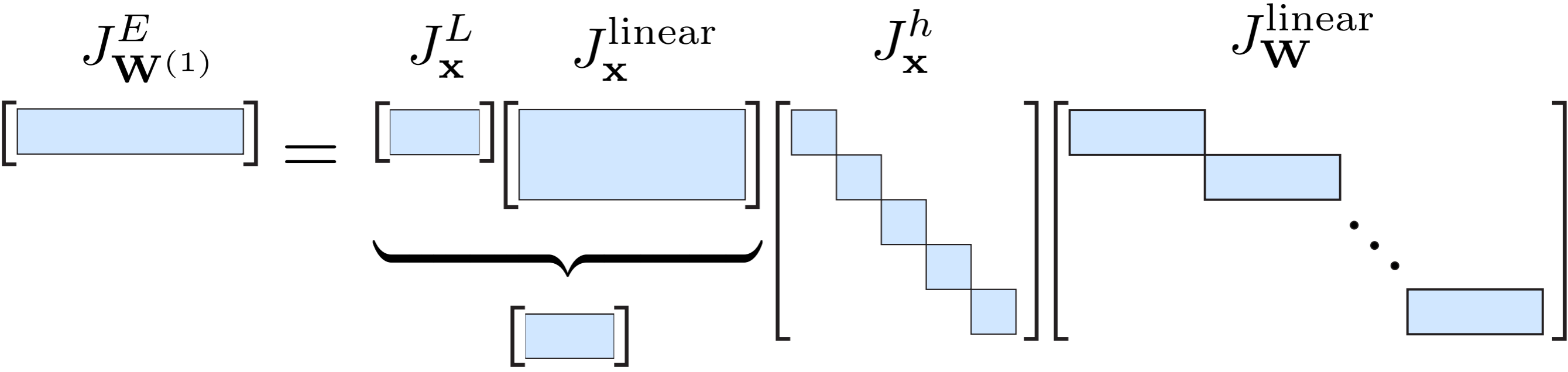
Automatic differentiation — reverse mode

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

$$J_{\mathbf{W}^{(1)}}^E = J_{\mathbf{x}}^L J_{\mathbf{x}}^{\text{linear}} J_{\mathbf{x}}^h J_{\mathbf{W}}^{\text{linear}}$$

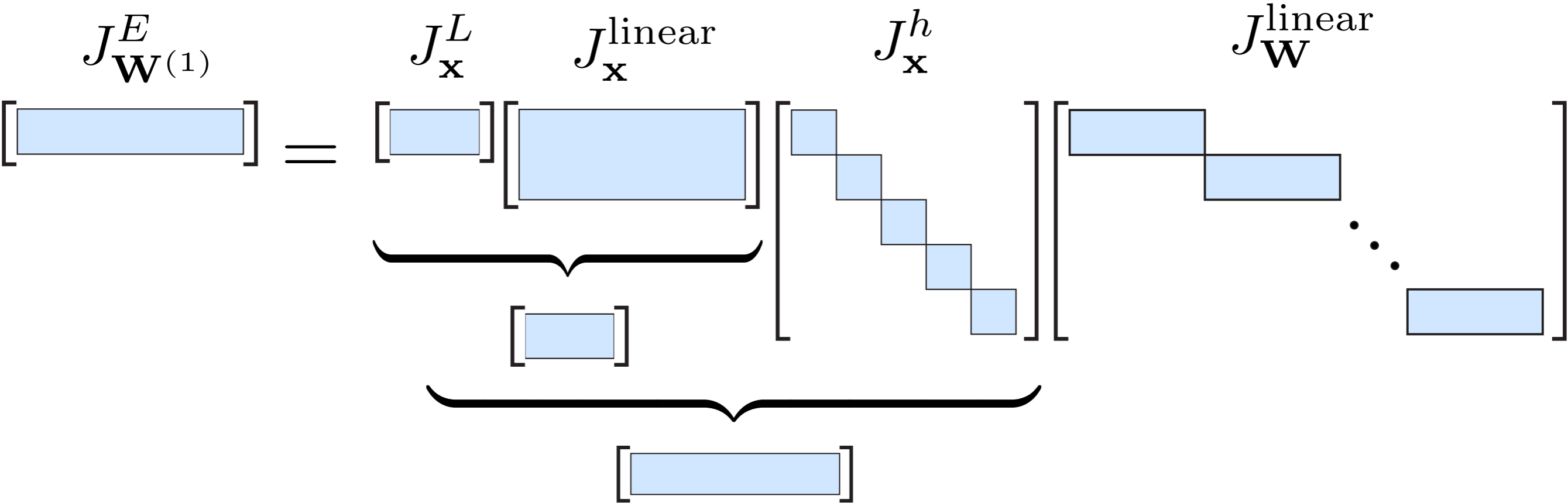
Automatic differentiation — reverse mode

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



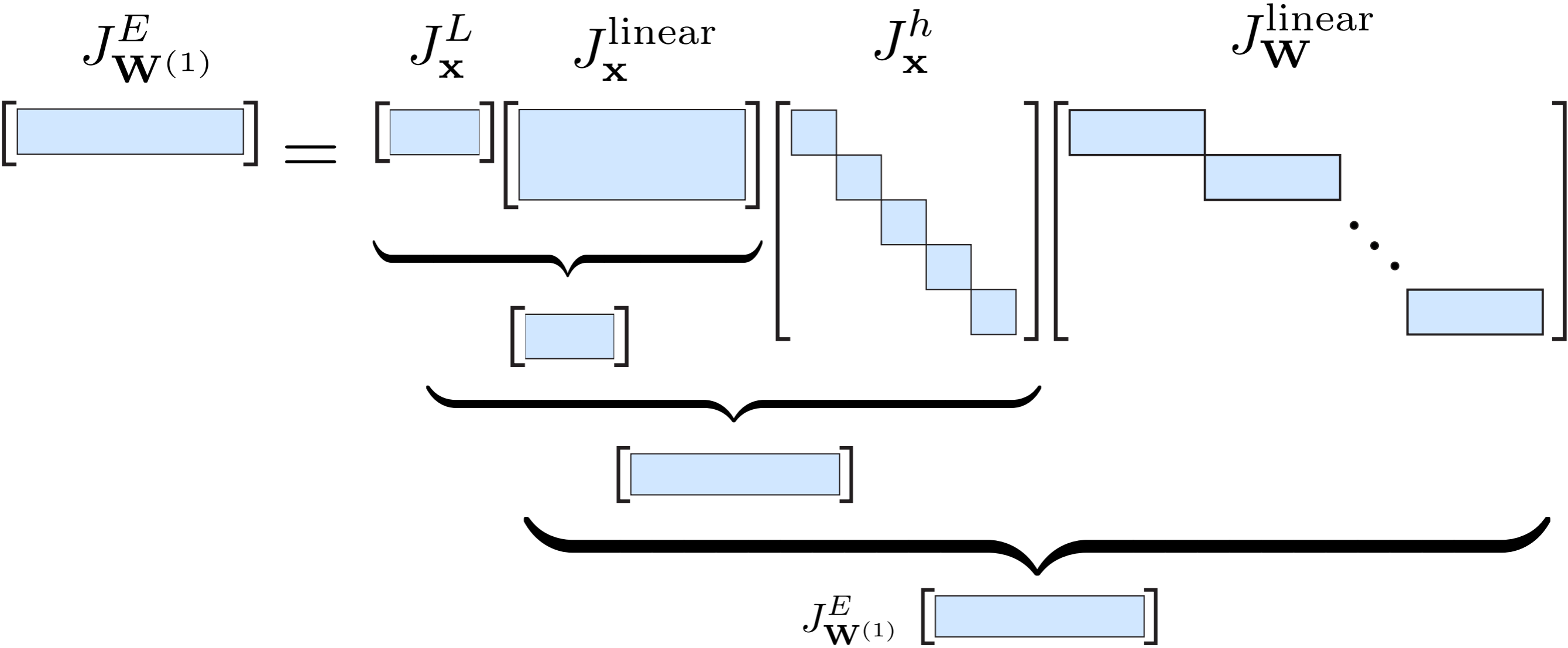
Automatic differentiation — reverse mode

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



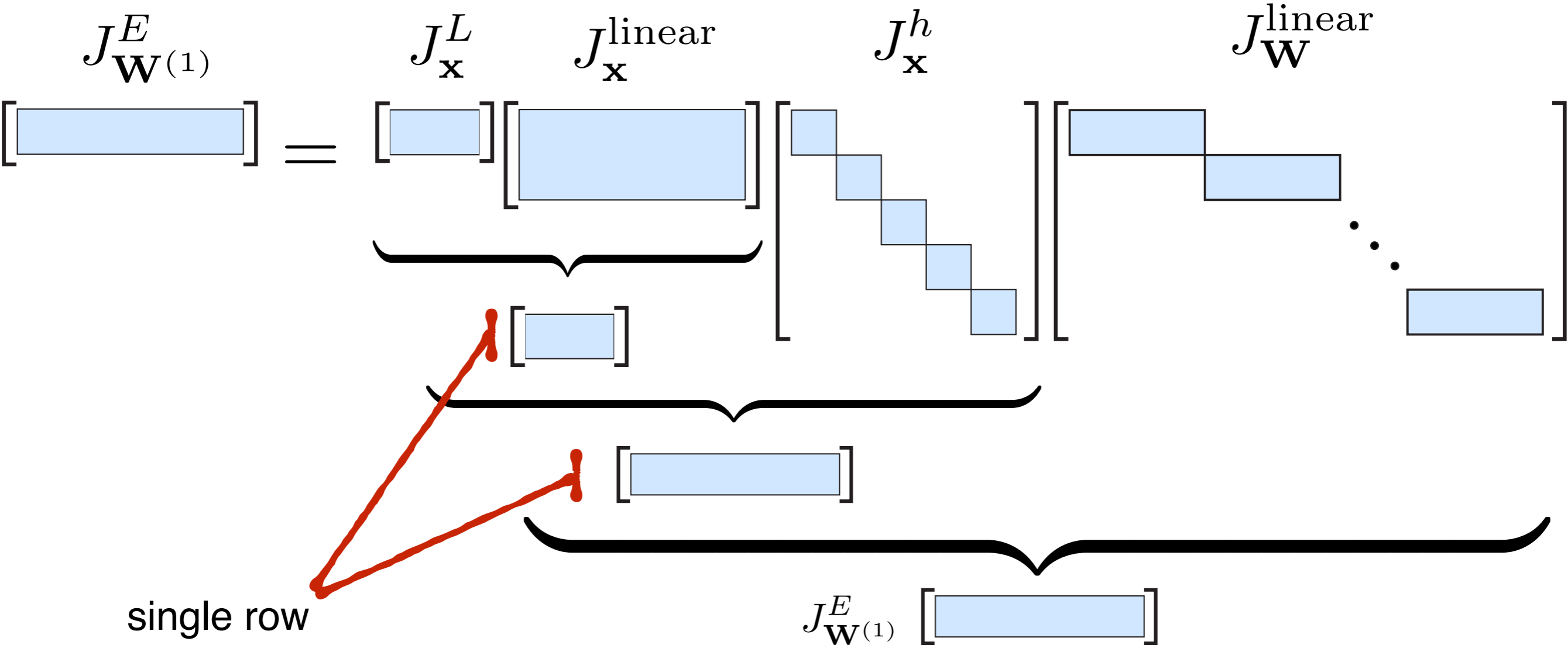
Automatic differentiation — reverse mode

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



Automatic differentiation — reverse mode

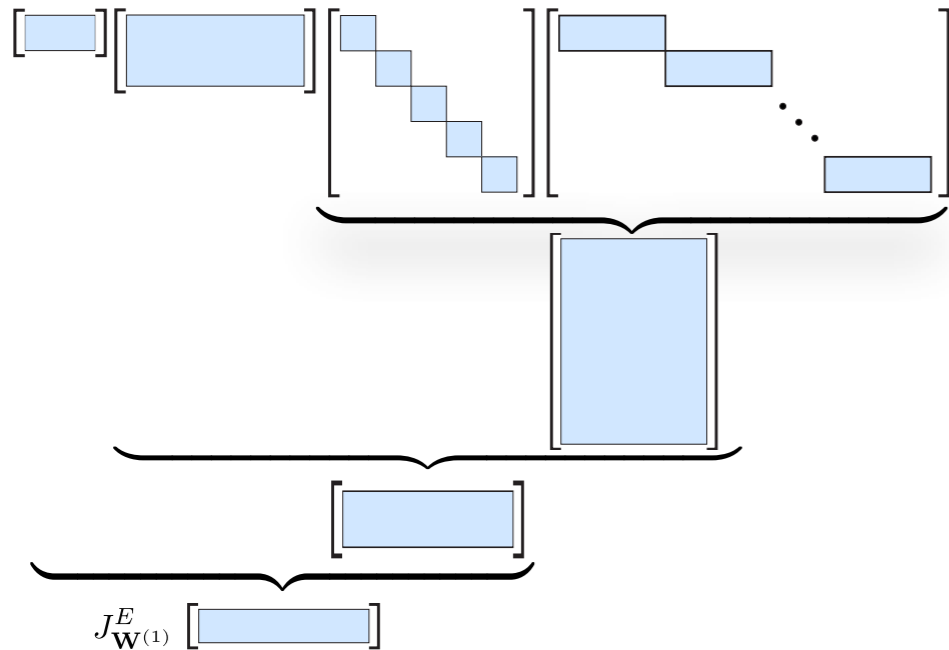
In reverse mode automatic differentiation Jacobi matrices are multiplied from front to back.



Forward vs. reverse mode

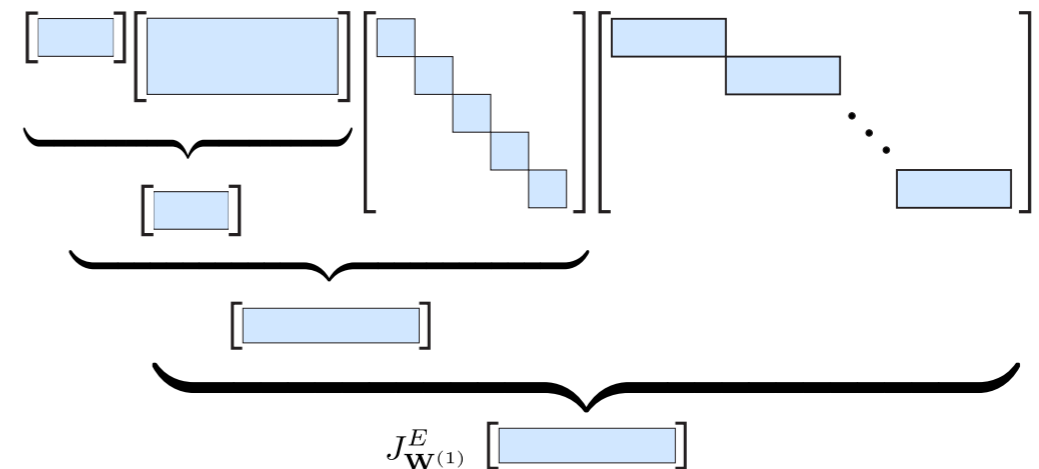
Reverse accumulation is more efficient for NNs since the objective function is a scalar.

Forward accumulation



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

Reverse accumulation

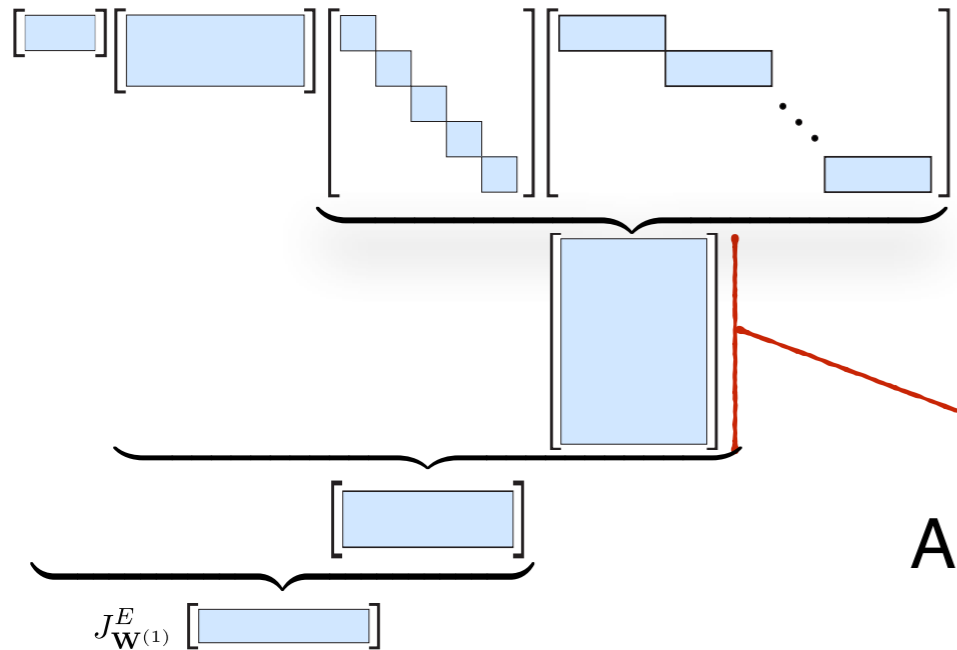


Reverse accumulation is more efficient for functions that have more inputs than outputs.

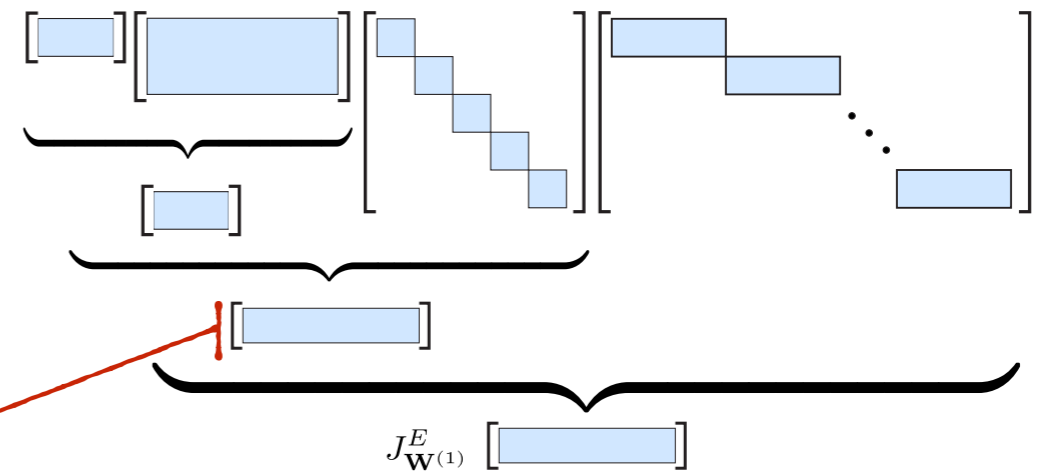
Forward vs. reverse mode

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

Forward accumulation



Reverse accumulation



A smaller row dimension is more efficient.

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

$$\begin{bmatrix} J_{\mathbf{x}}^E \\ J_{\mathbf{x}}^{h^{(1)}} \end{bmatrix} = \begin{bmatrix} h' & & & & \\ & h' & & & \\ & & h' & & \\ & & & h' & \\ & & & & h' \end{bmatrix} = \begin{bmatrix} J_{\mathbf{x}}^E \end{bmatrix} * \begin{bmatrix} h' & h' & h' & h' & h' \end{bmatrix}$$


↑
element-wise multiplication

Backpropagation — a special case

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

Backpropagation through activation function


$$\begin{bmatrix} J_{\mathbf{x}}^E \end{bmatrix} \begin{bmatrix} h' & & & & \\ & h' & & & \\ & & h' & & \\ & & & h' & \\ & & & & h' \\ & & & & & J_{\mathbf{x}}^{h^{(1)}} \end{bmatrix} = \begin{bmatrix} J_{\mathbf{x}}^E \end{bmatrix} * \begin{bmatrix} h' & h' & h' & h' & h' & h' \end{bmatrix}$$



 element-wise multiplication

Backpropagation through linear layer

$$\begin{bmatrix} J_{\mathbf{x}}^E \end{bmatrix} \begin{bmatrix} \mathbf{x} & & & & \\ & \mathbf{x} & & & \\ & & \ddots & & \\ & & & \mathbf{x} & \\ & & & & \mathbf{x} \end{bmatrix} J_{\mathbf{W}}^{\text{linear}(\mathbf{x}, \mathbf{W}, \mathbf{b})} = \begin{bmatrix} J_{\mathbf{x}}^{E\top} \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix}$$



 needs to be flattened

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

$$J_{\mathbf{W}^{(1)}}^E = J_{\mathbf{x}}^L J_{\mathbf{x}}^{\text{linear}} J_{\mathbf{x}}^h J_{\mathbf{W}}^{\text{linear}}$$

Compact backpropagation

$$J_{\mathbf{W}^{(1)}}^E = J_{\mathbf{x}}^L * \mathbf{W}^{(2)} * [h' h' h' h' h' h']^T [\mathbf{x}]$$

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

3. Objective and loss function

- sum over loss on dataset samples



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

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

3. Objective and loss function

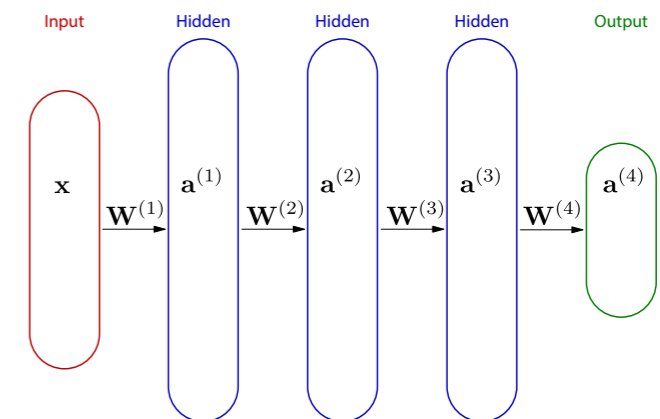
- sum over loss on dataset samples

4. NN model

- stack of linear and non-linear layers



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



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

3. Objective and loss function

- sum over loss on dataset samples

4. NN model

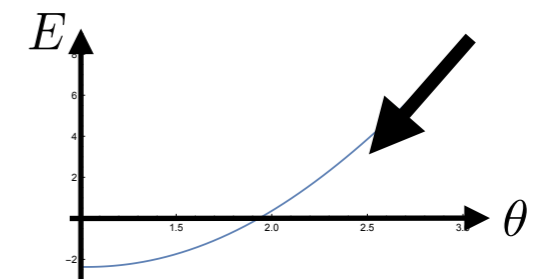
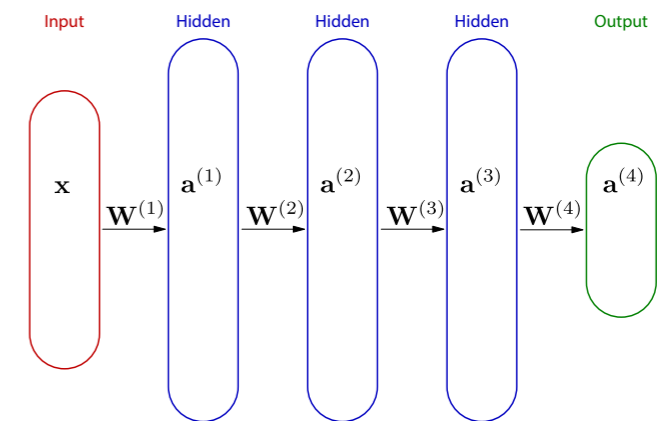
- stack of linear and non-linear layers

5. Optimization procedure (solver)

- iterative gradient descent approximation



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



Outlook

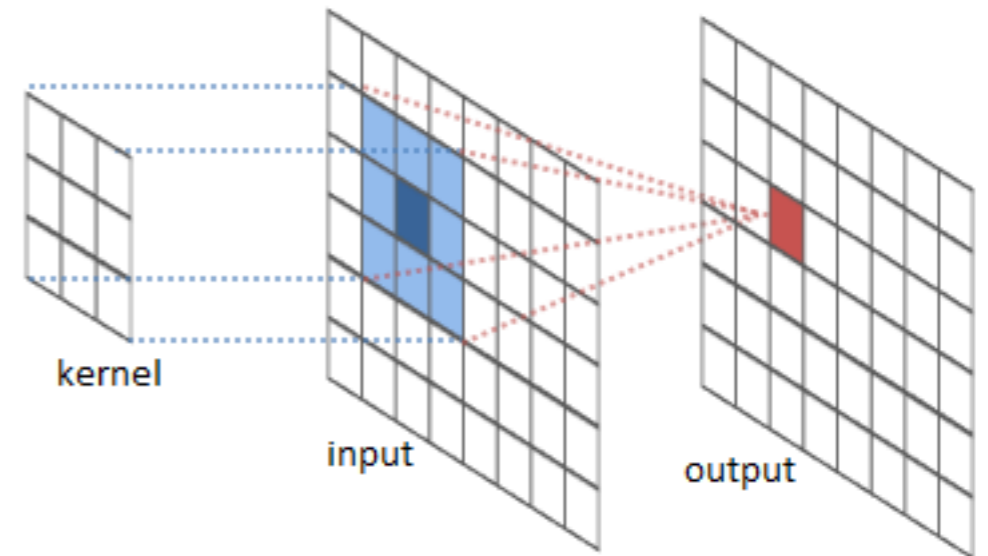
Advanced NNs

Convolution, Filters

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

- Local kernel

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

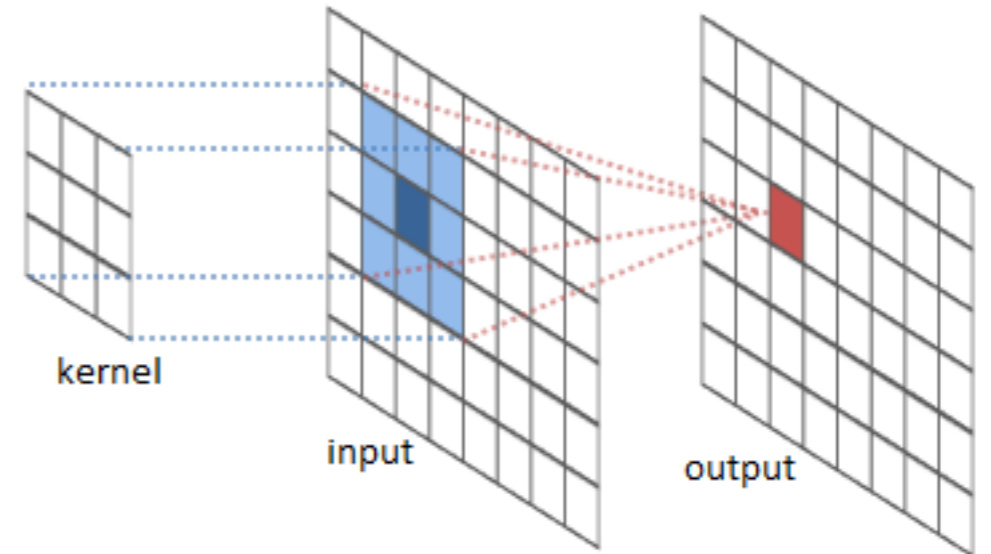


Convolution, Filters

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

- Local kernel

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



[cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12]

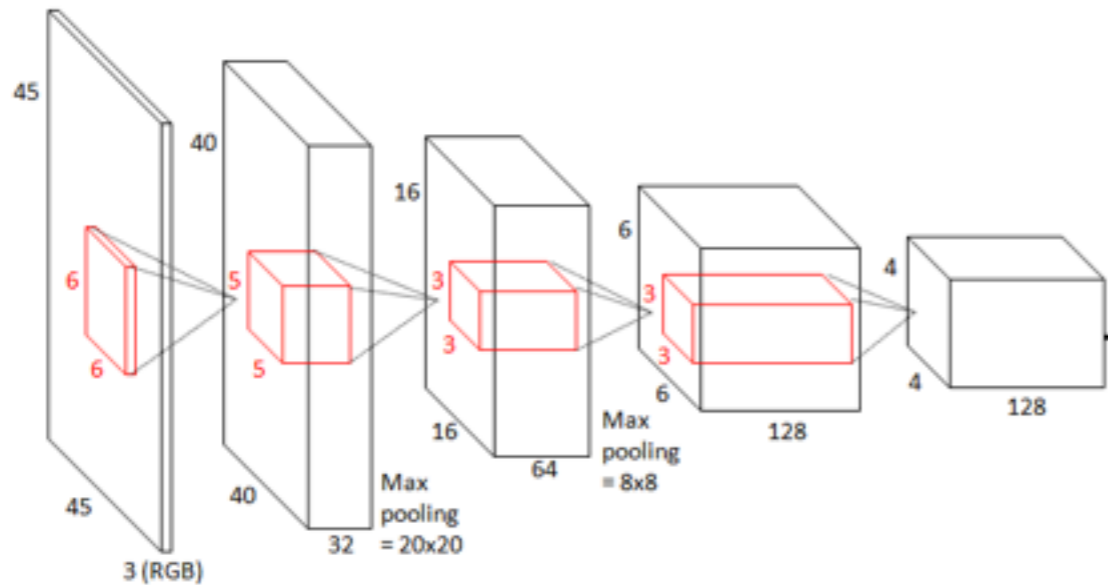


Input

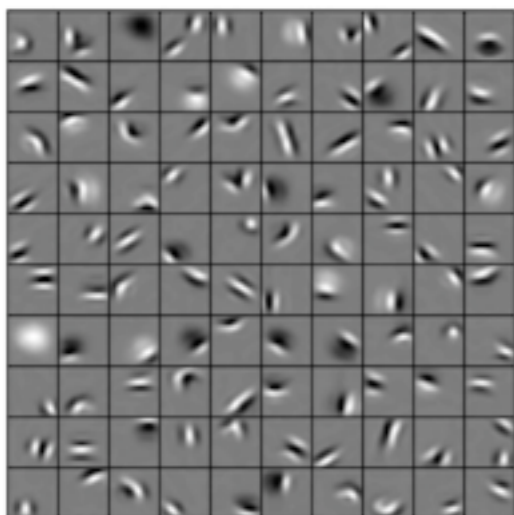
Convolutional Neural Networks

Convolutional NNs apply convolutions with trainable weights — weight sharing.

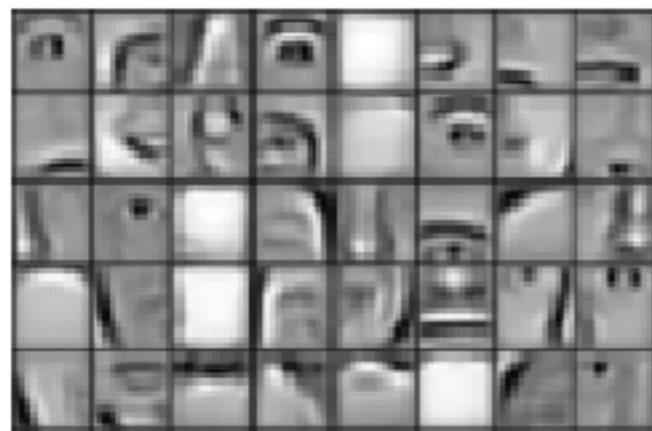
- Local operations, weights shared



Low-level



Mid-level



High-level



Data dependent filter

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

Faces

Cars

Mid-level
(parts)



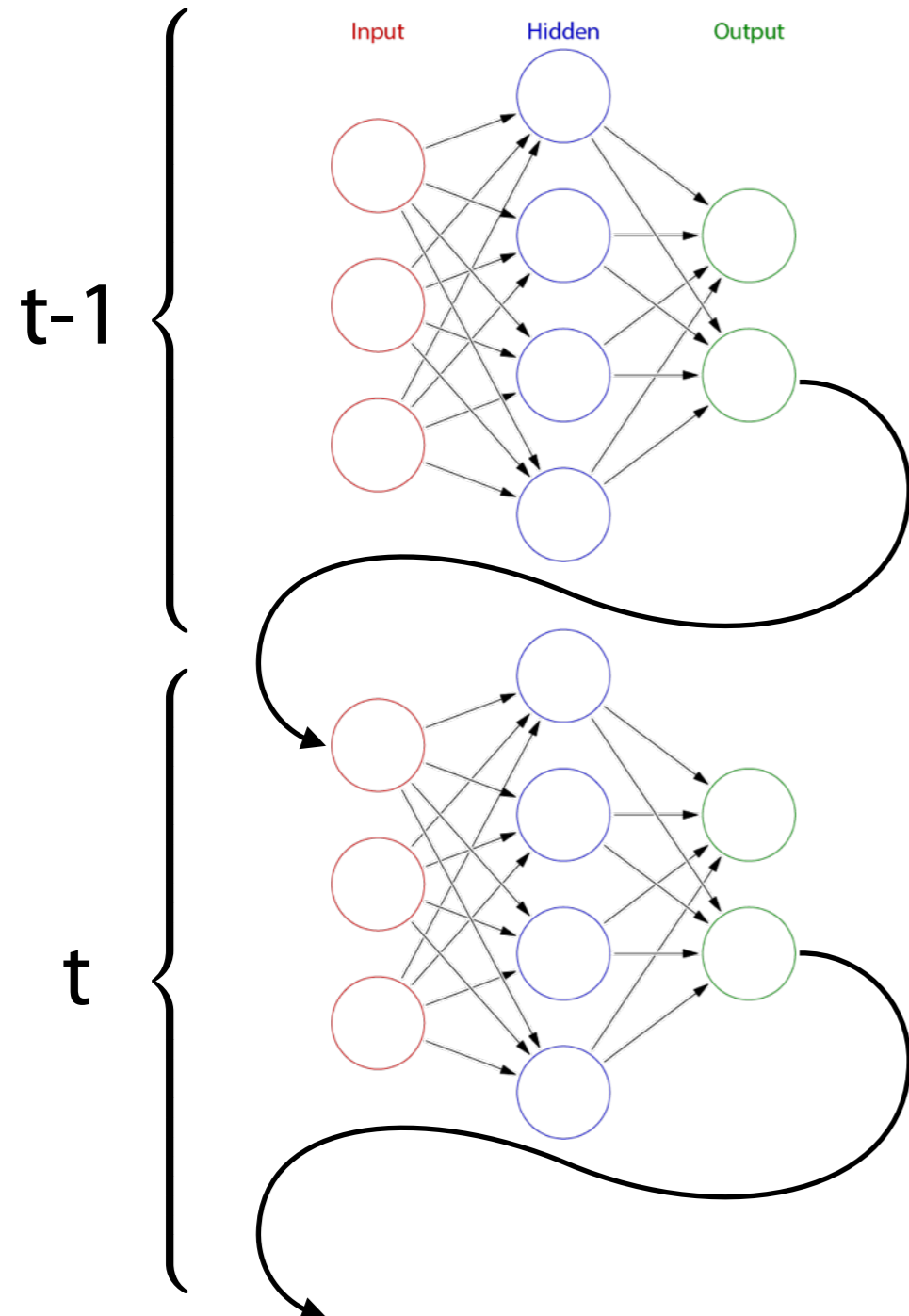
High-level
(composition)



Recurrent neural networks

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

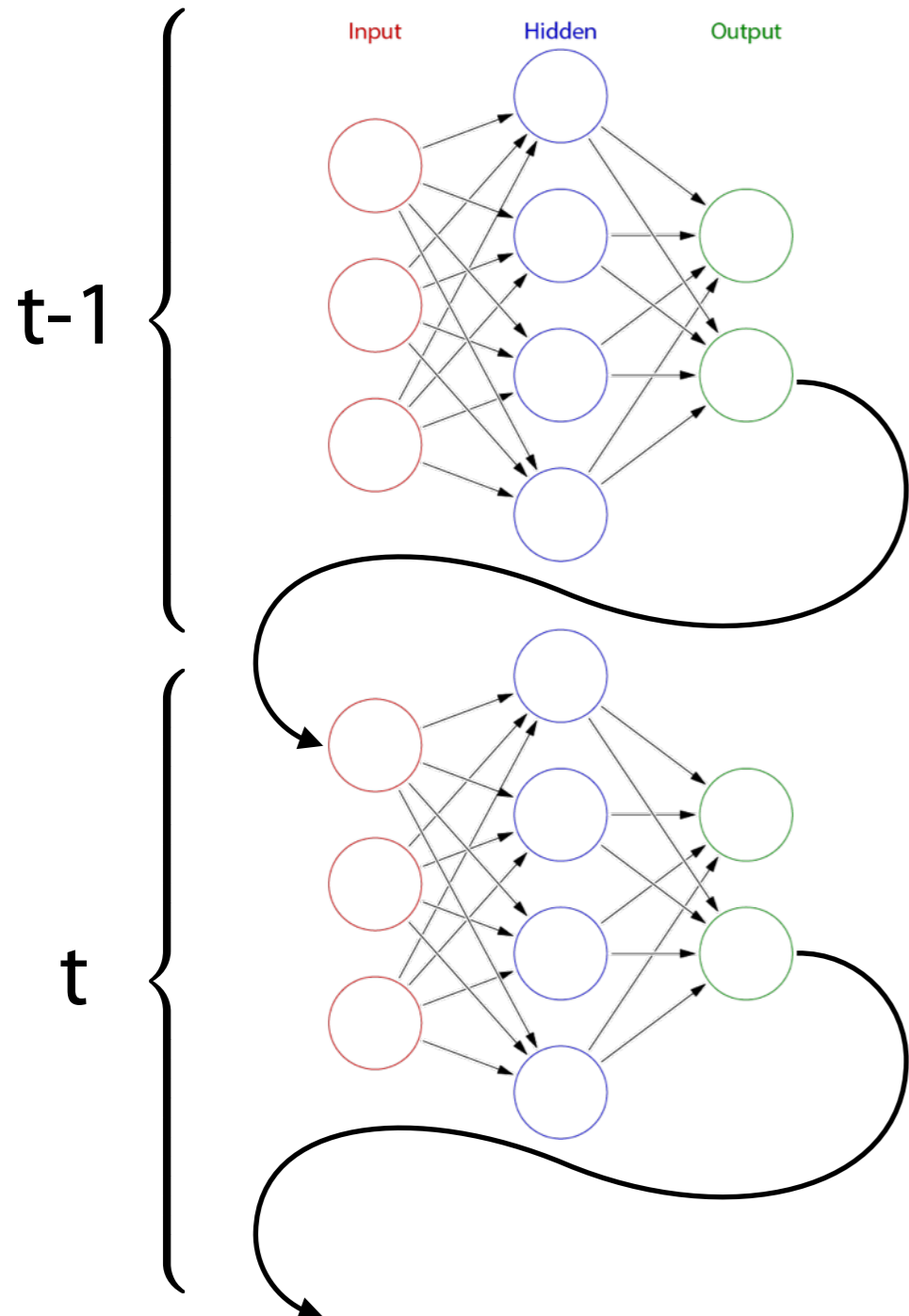
Stacking multiple NNs



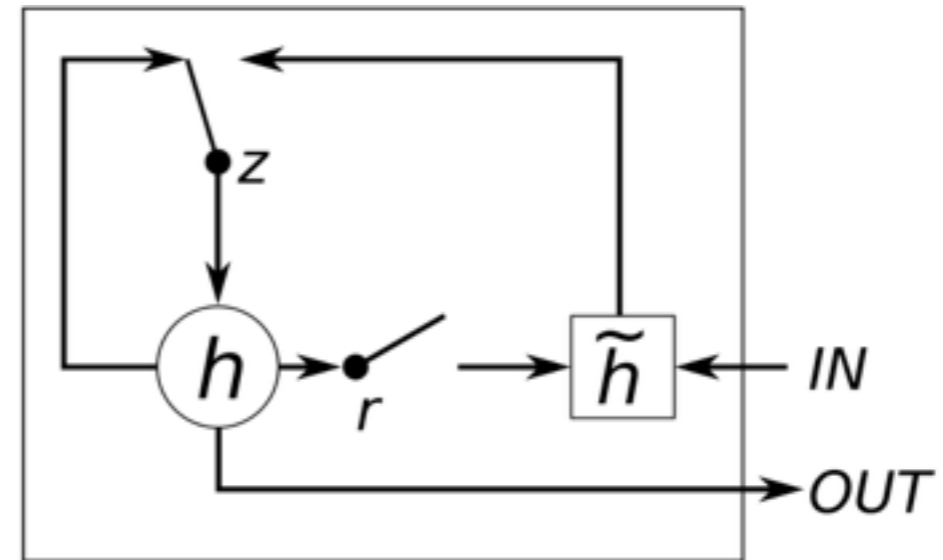
Recurrent neural networks

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)



Generative adversarial networks (GANs)

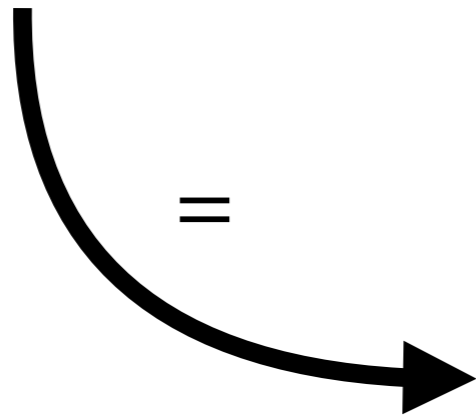


Neural style transfer

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



+



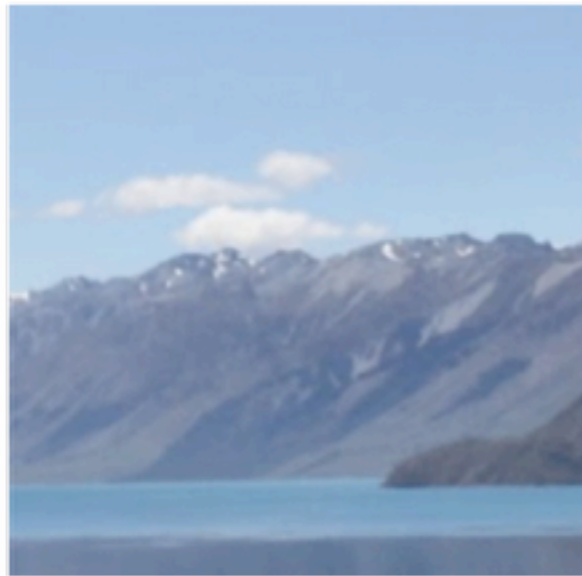
Neural style transfer

The style transfer works on many different styles.



Understanding neural networks

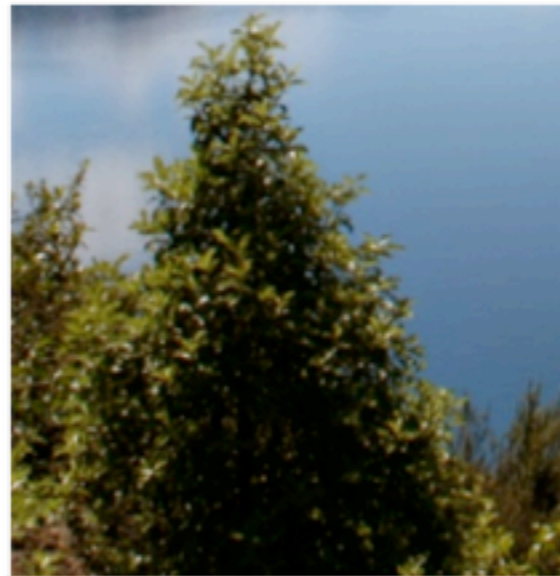
By optimising the input image instead of the network weights, the learned patterns are revealed.



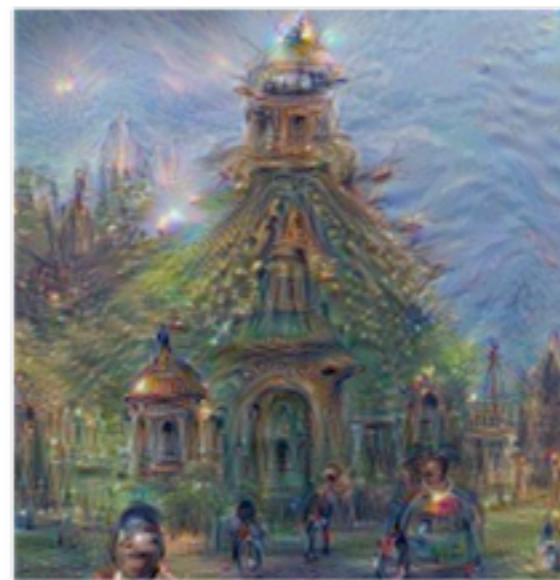
Horizon



Towers & Pagodas



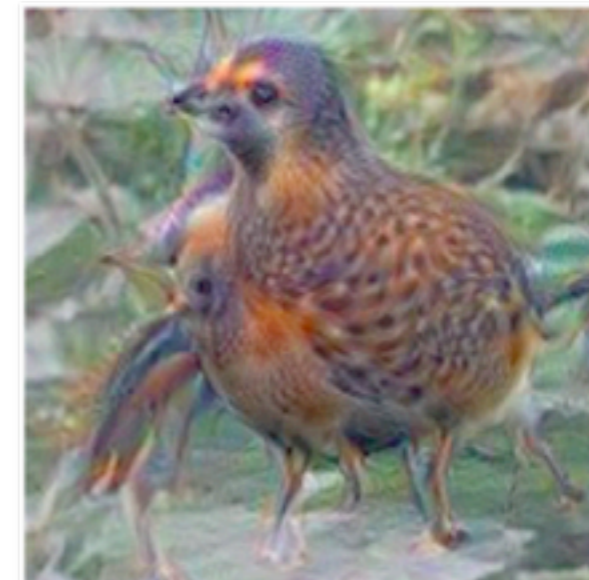
Trees



Buildings



Leaves



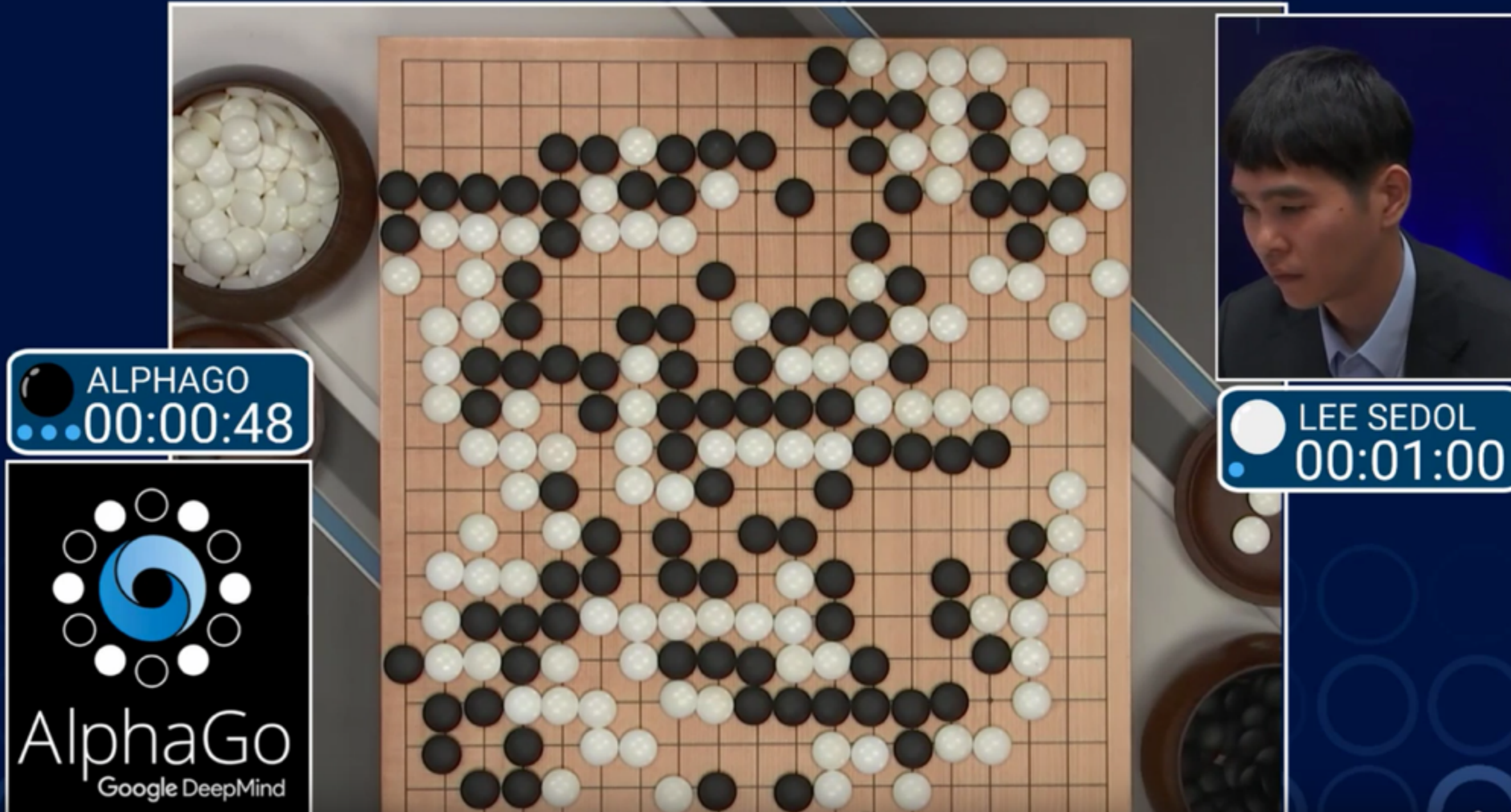
Birds & Insects

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

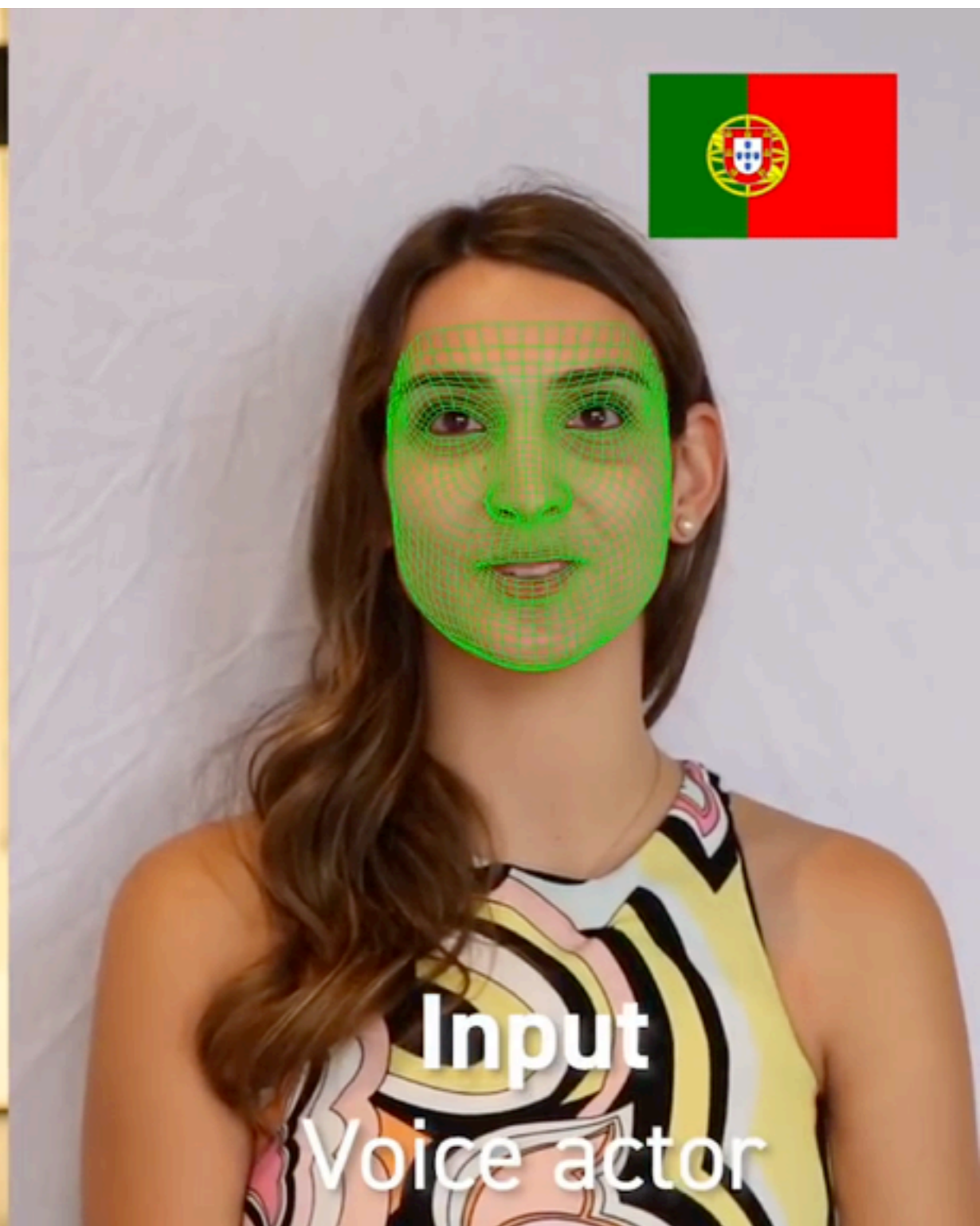
- Stephen Hawking, Elon Musk, and dozens of artificial intelligence experts

Virtual dubbing



[\[www.synthesia.io\]](http://www.synthesia.io)

Virtual dubbing



[\[www.synthesia.io\]](http://www.synthesia.io)

Homework 5

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

