Pattern Recognition

Part 8: (Artificial) Neural Networks

Gerhard Schmidt

Christian-Albrechts-Universität zu Kiel
Faculty of Engineering
Institute of Electrical and Information Engineering
Digital Signal Processing and System Theory

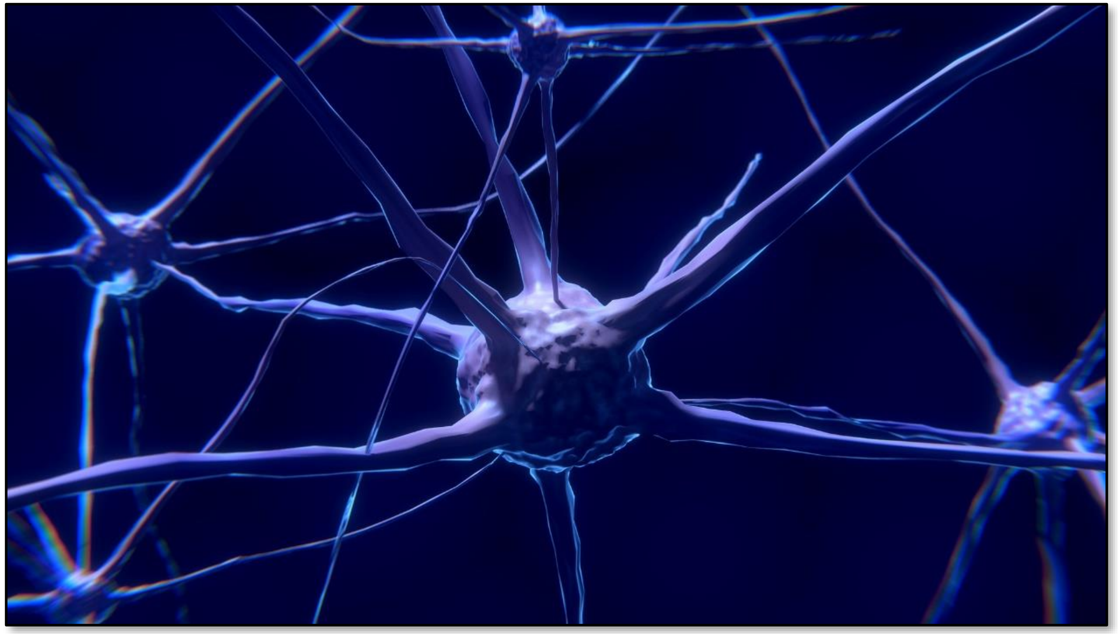
- Motivation and literature
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
- Reinforcement learning


Contents

Motivation and Literature

Neural networks:

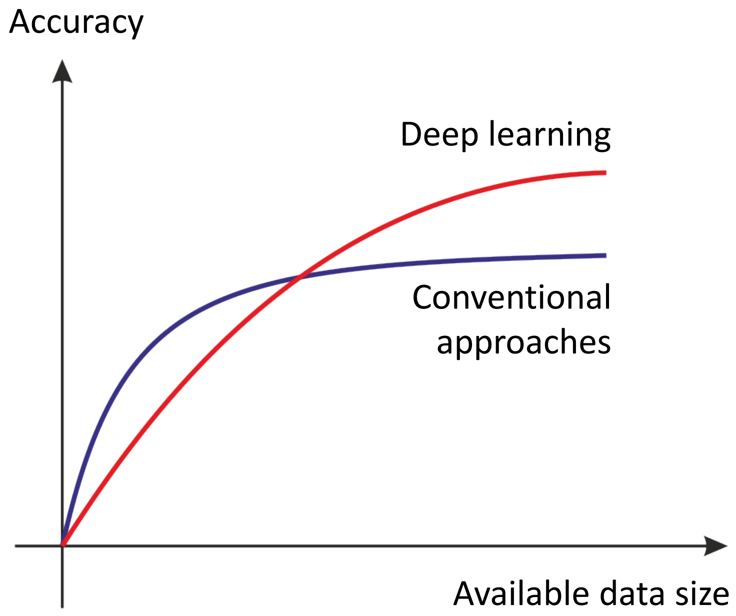
- Neural networks are a ***very popular*** machine learning technique.
- They ***simulate the mechanisms of learning in biological systems*** such as the human brain.
- The human brain / the nervous system contains cells which are called ***neurons***. The neurons are ***connected*** using ***axons*** and ***dendrites***. While learning the connections between neurons are changed.
- Within this lecture we will talk about ***artificial neural networks*** that mimic the processes in the human brain. The adjective “artificial” will be omitted for reasons of brevity.



Motivation and Literature

Deep learning:

- The advantage of neuronal structures is their ability to be ***adapted to several types of problems*** by ***changing their size and internal structure***.
- A few years ago so-called ***deep approaches*** appeared. This was one of the main factors for the success of neural networks.
- "Deep" means here to have on the one hand ***several/many hidden layers***. On the other hand it means that s***pecific training procedures*** are used.
- Compared to conventional (shallow) structures deep approaches are ***specially suited*** if a ***large amount of training data*** is available.



Motivation and Literature

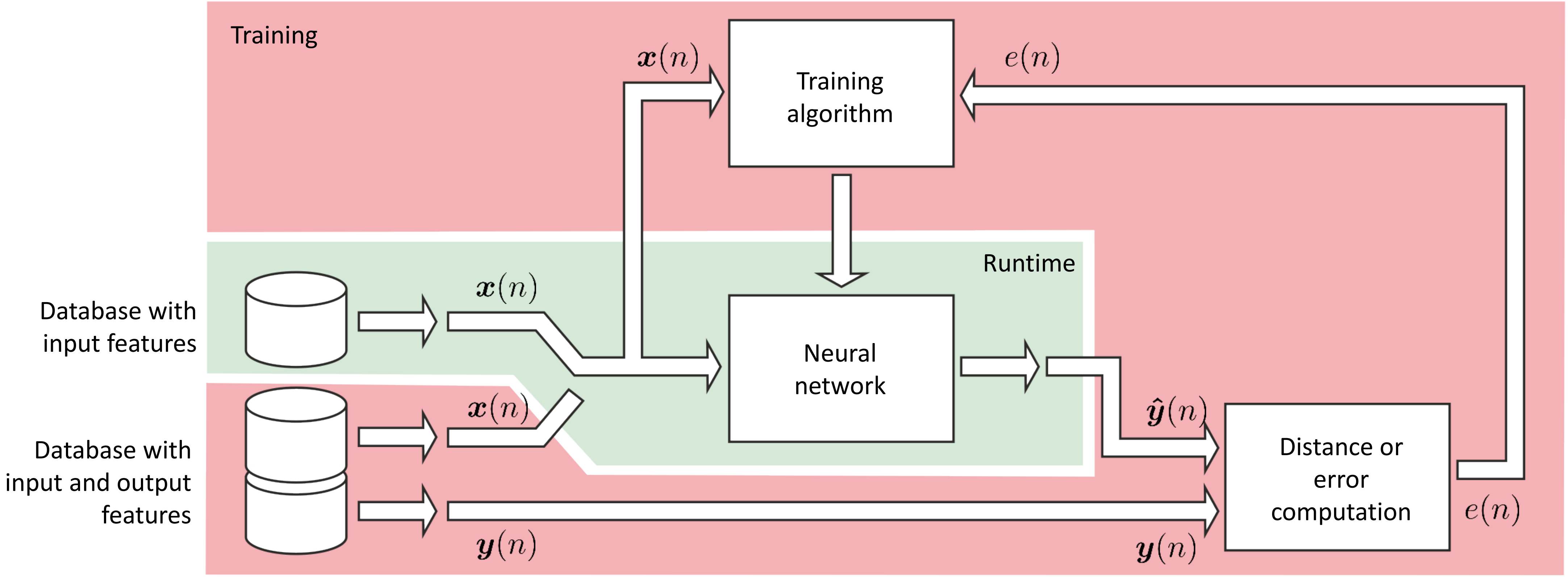
Literature:

- C. C. Aggarwal: ***Neural Networks and Deep Learning***, Springer, 2018
- A. Géron: ***Machine Learning mit Scikit-Learn & Tensorflow***, O'Reilly, 2018 (in German and English)
- I. Goodfellow, Y. Bengio, A. Courville: ***Deep Learning***, MITP, 2018 (in German and English)

- Motivation
- ***Structure of a (basic) neural network***
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks
- Reinforcement learning

Structure of a Neural Network – Basics

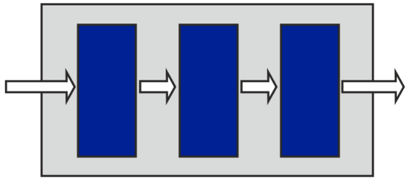
Basic structure during runtime and training:



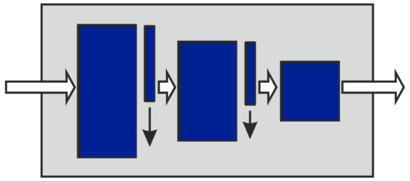
Structure of a Neural Network – Basics

Network structure(s):

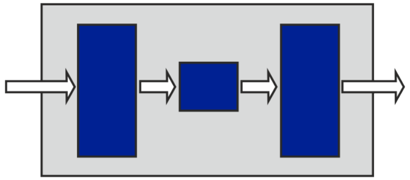
Multilayer perceptron



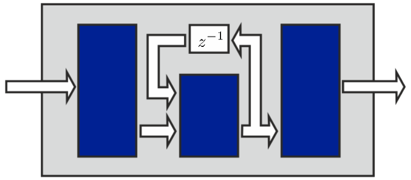
Convolutional neural network



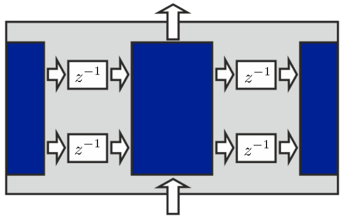
Autoencoder



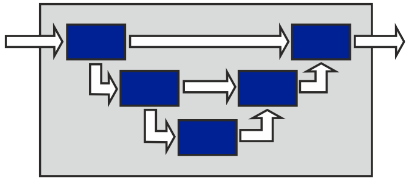
Recurrent neural network



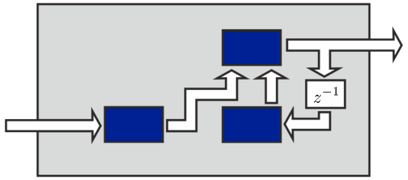
LSTM network



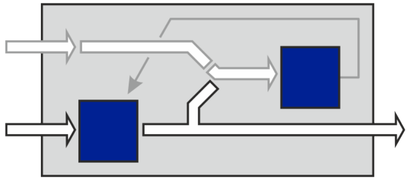
“U-Net”



Attention-based network

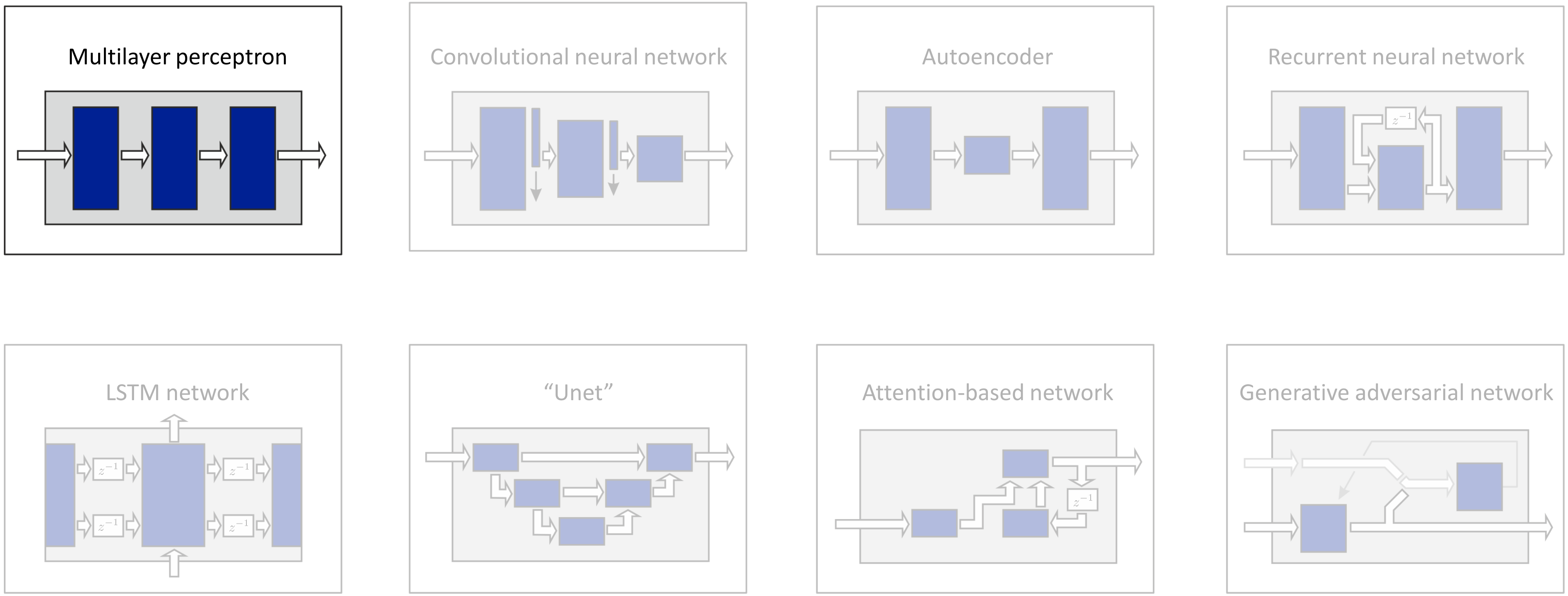


Generative adversarial network



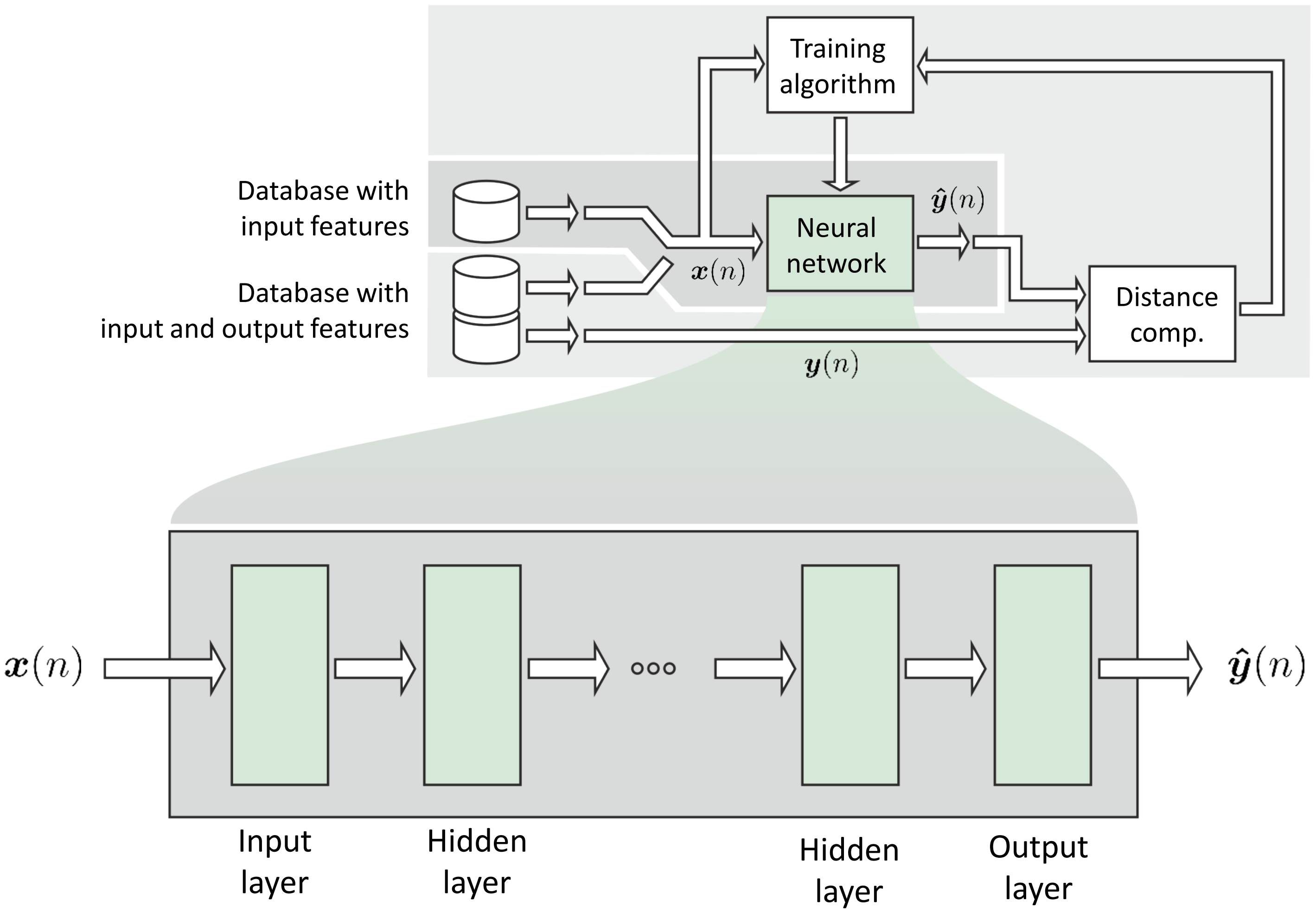
Structure of a Neural Network – Basics

Network structure(s):



Structure of a Neural Network – Basics

Network structure:



Structure of a Neural Network – Basics

Input layer:

- Sometimes only a ***"pass through" layer***

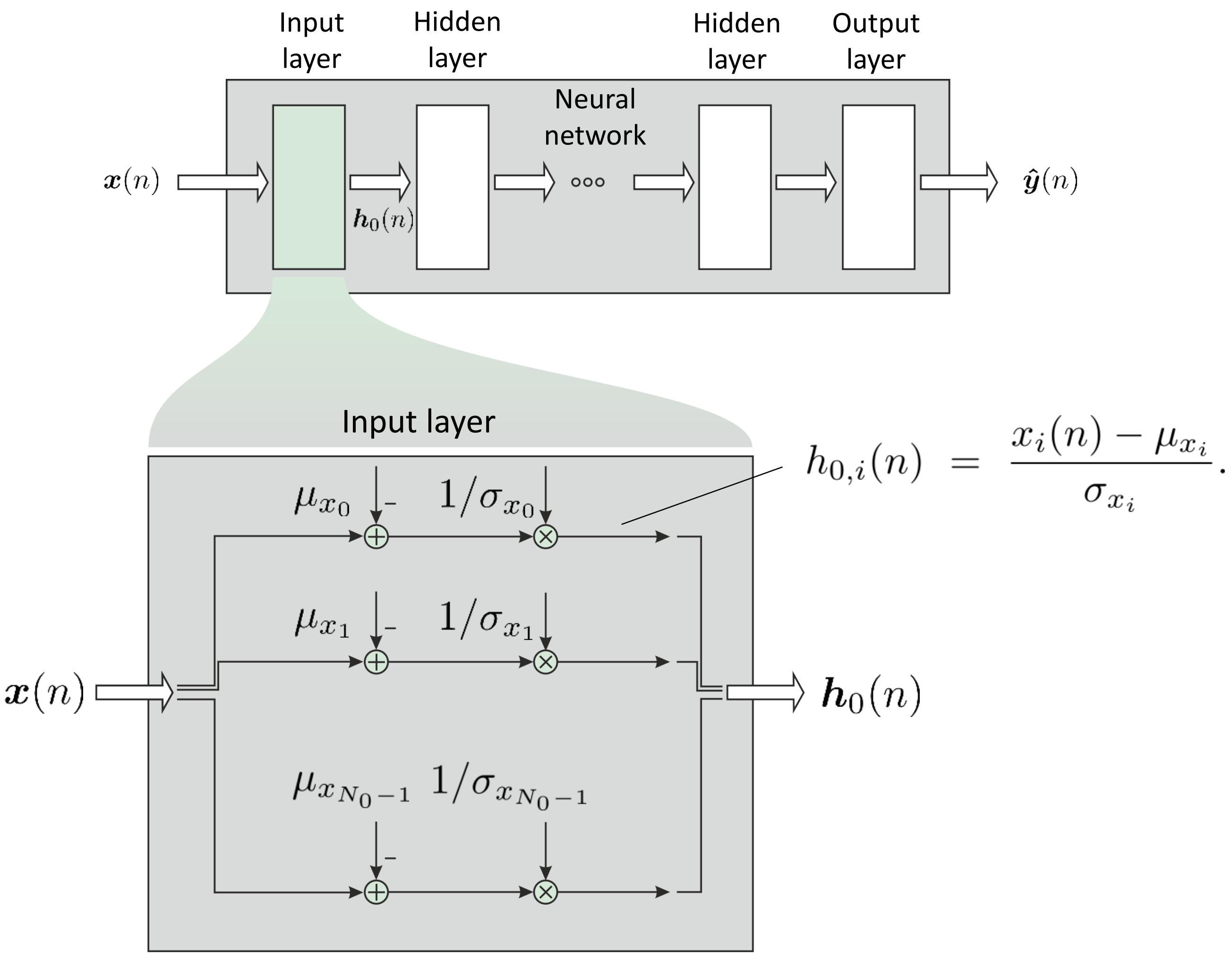
$$\boldsymbol{h}_0(n) \;=\; \boldsymbol{x}(n).$$

- Sometimes also a ***mean compensation*** and a ***normalization*** is performed:

$$h_{0,i}(n) \;=\; \frac{x_i(n) - \mu_{x_i}}{\sigma_{x_i}}.$$

Afterwards all individually normalized inputs are ***combined to a vector***:

$$\boldsymbol{h}_0(n) \;=\; \big[h_{0,0}(n),\, ...,\, h_{0,N_0-1}(n)\big]^{\mathrm{T}}$$



Structure of a Neural Network – Basics

Hidden layer:

- ***Linear weighting*** of inputs with ***bias***

$$x_{m,i}(n) = \boldsymbol{w}_{m,i}^{\mathrm{T}}\, \boldsymbol{h}_m(n) + b_{m,i}$$

with

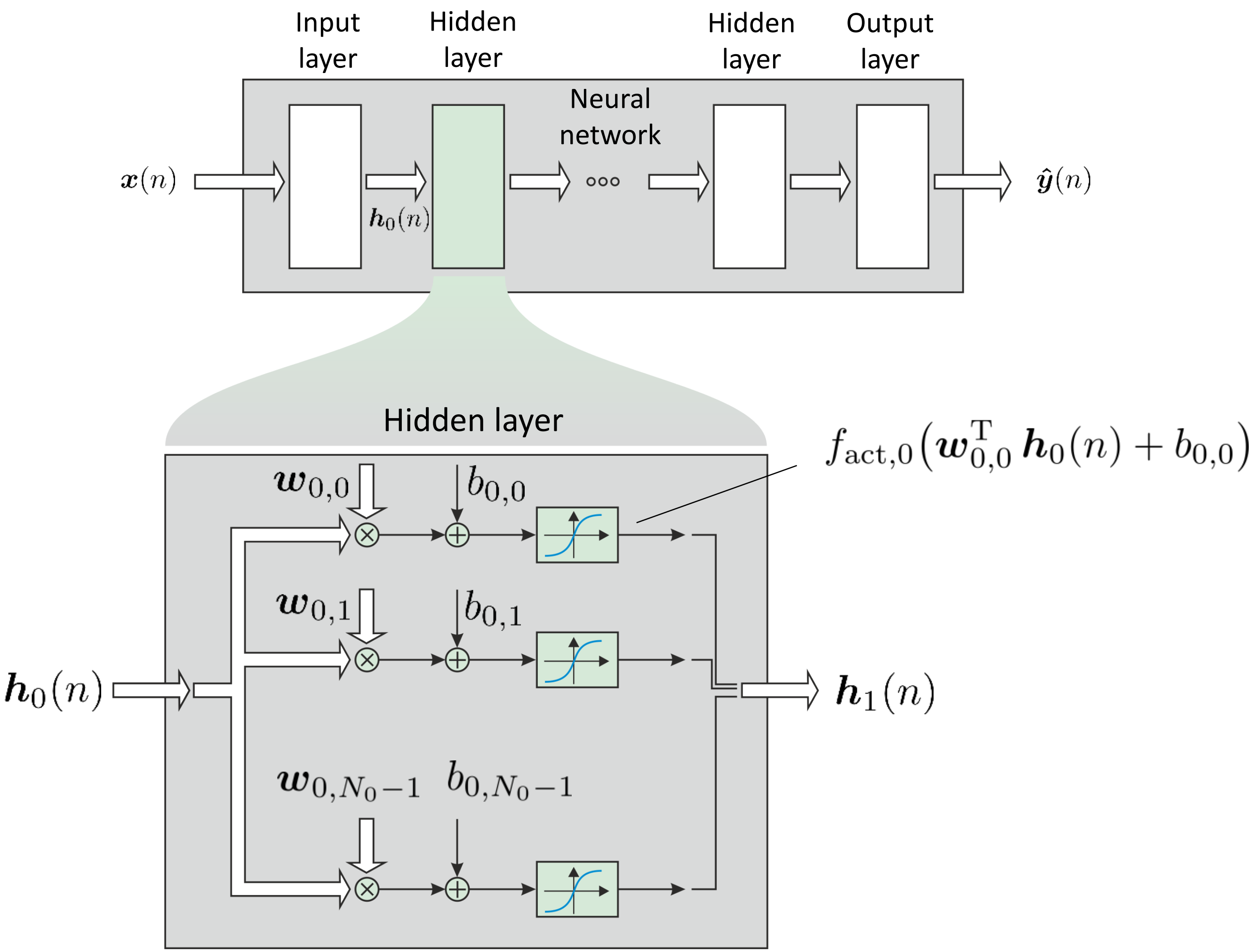
$$\boldsymbol{w}_{m,i} = \left[w_{m,0},\, \ldots,\, w_{m,N_{m-1}-1}\right]^{\mathrm{T}}.$$

- Nonlinear ***activation function***:

$$y_{m,i}(n) = f_{\mathrm{act},m}\big(x_{m,i}(n)\big).$$

- ***Combination*** of all results to a ***vector***:

$$\boldsymbol{h}_{m+1}(n) = \left[y_{m,0}(n),\, \ldots,\, y_{m,N_m-1}(n)\right]^{\mathrm{T}}.$$



Structure of a Neural Network – Basics

Activation functions – part 1:

- The sum of the weighted inputs plus the bias will be ***abbreviated*** with

$$x(n) \;=\; \boldsymbol{w}^{\mathrm{T}}\,\boldsymbol{h}(n) + b.$$

- Several ***activation functions*** exist, such as
  - the ***identity*** function

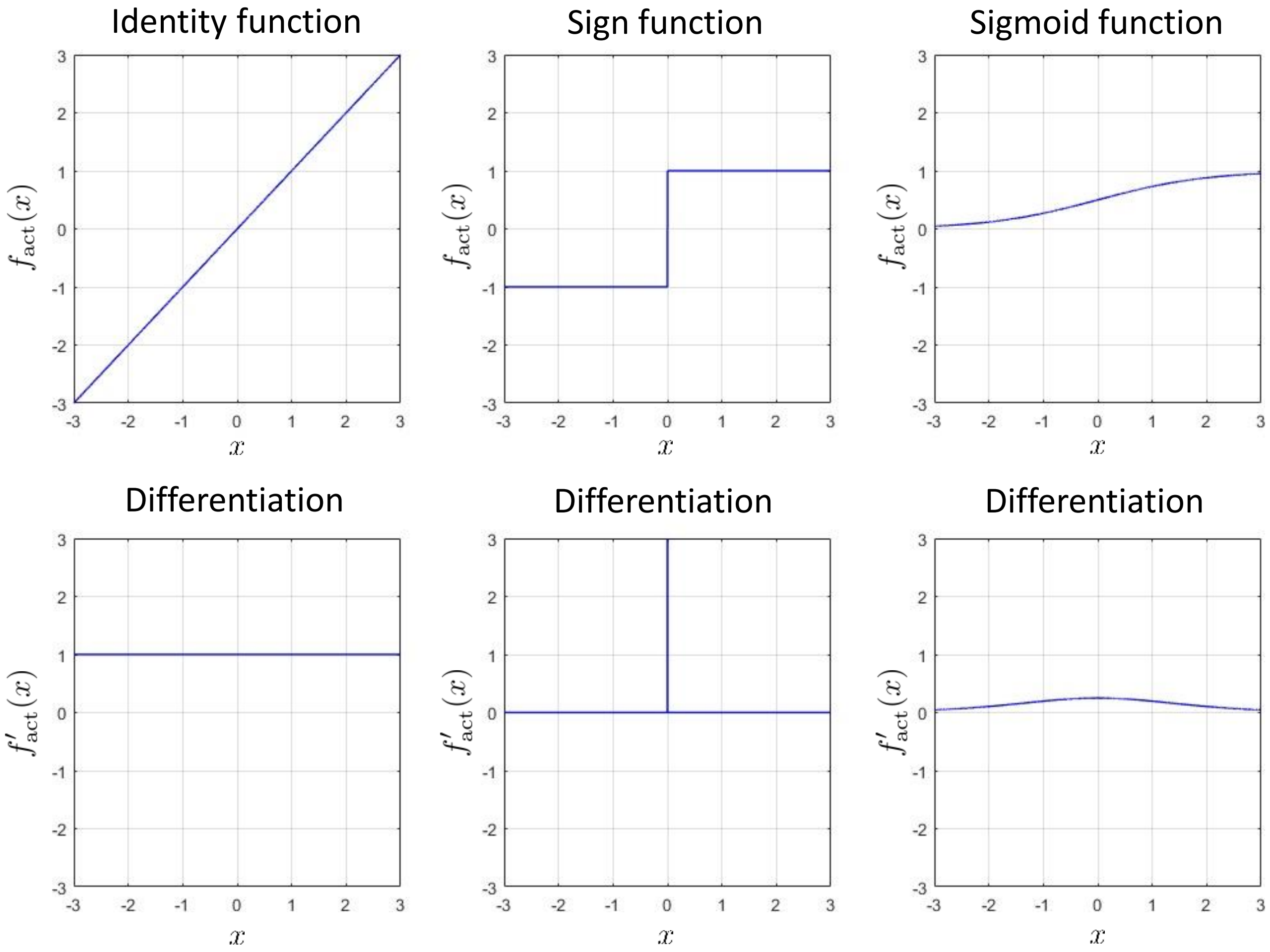
$$f_{\mathrm{act}}\big(x(n)\big) \;=\; x(n),$$

  - the ***sign*** function, or

$$f_{\mathrm{act}}\big(x(n)\big) \;=\; \mathrm{sign}\big(x(n)\big),$$

  - the ***sigmoid*** function

$$f_{\mathrm{act}}\big(x(n)\big) \;=\; \frac{1}{1+e^{-x(n)}}.$$



Structure of a Neural Network – Basics

Activation functions – part 2:

- Further ***activation functions***:
  - the ***tanh*** function

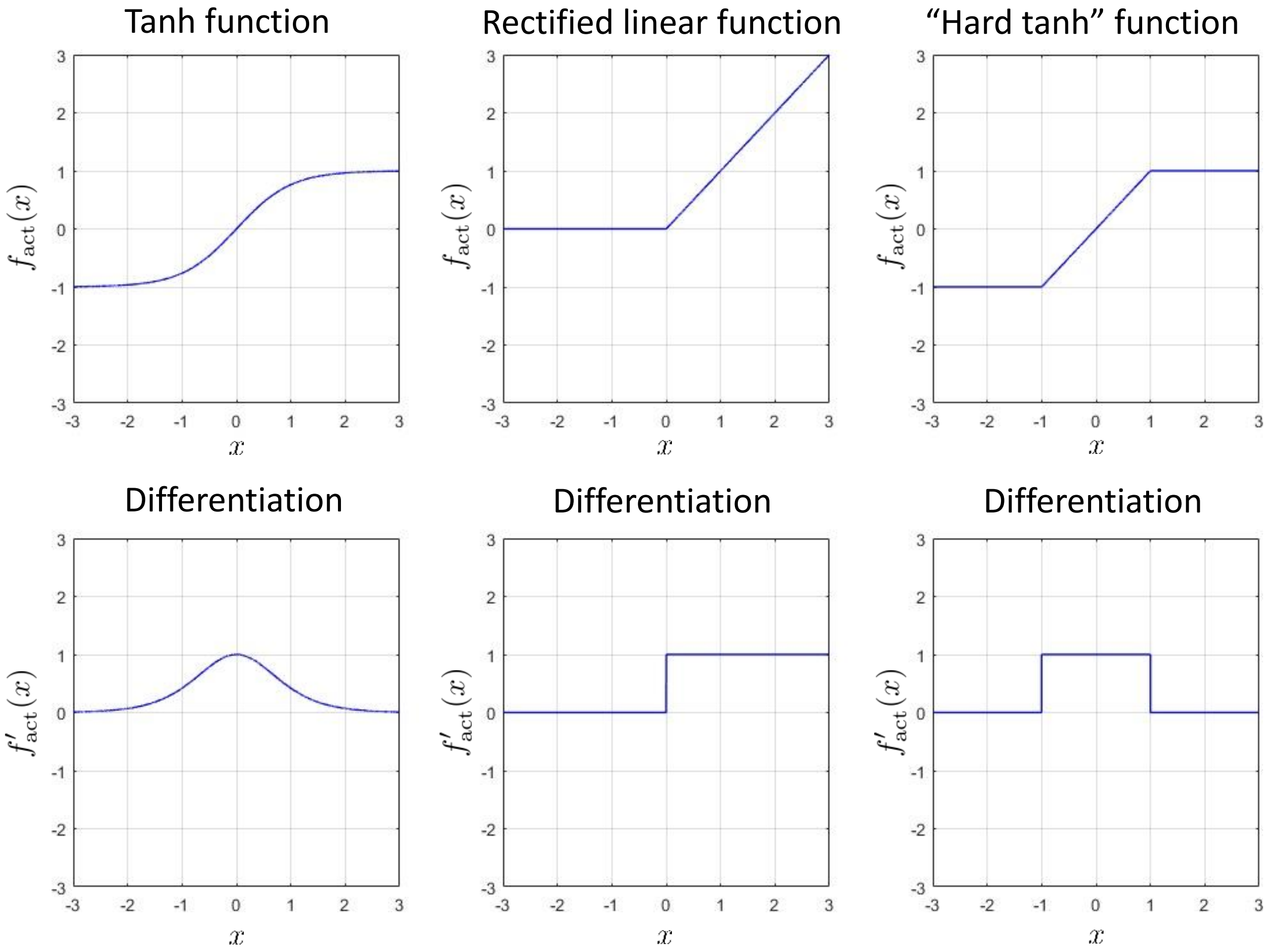
$$f_{\text{act}}\big(x(n)\big) \;=\; \frac{e^{2x(n)}-1}{e^{2x(n)}+1},$$

  - the ***rectified linear*** function (or unit, ReLU)

$$f_{\text{act}}\big(x(n)\big) \;=\; \max\{0,\, x(n)\},$$

  - the “***hard tanh***“ function

$$f_{\text{act}}\big(x(n)\big) \;=\; \max\Big\{\min\{1,\, x(n)\},\, -1\Big\}.$$



Christian-Albrechts-Universität zu Kiel

Structure of a Neural Network – Basics

Output layer:

- Sometimes only a ***"pass through" layer***

$$\hat{\boldsymbol{y}}(n) \;=\; \boldsymbol{h}_M(n).$$

- Sometimes also a ***limitation***

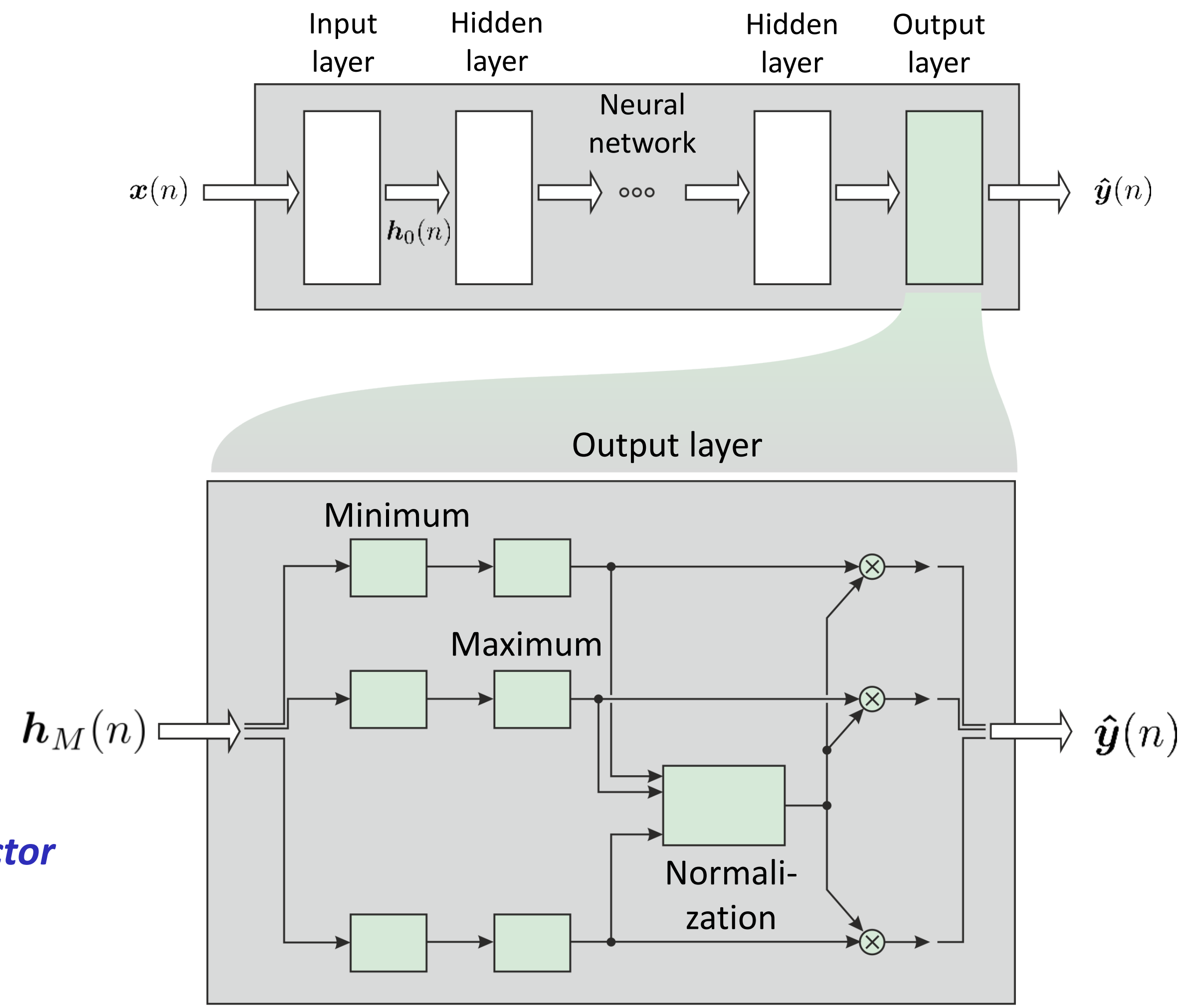
$$\hat{y}_{\mathrm{lim},i}(n) \;=\; \max\Big\{\hat{y}_{\min},\, \min\big\{\hat{y}_{\max},\, h_{M,i}(n)\big\}\Big\}$$

and a ***normalization*** is performed:

$$\hat{y}_i(n) \;=\; \frac{\hat{y}_{\mathrm{lim},i}(n)}{\sum\limits_{i=0}^{N_M-1} \hat{y}_{\mathrm{lim},i}(n)}.$$

The limited and normalized outputs are ***combined to a vector***

$$\hat{\boldsymbol{y}}(n) \;=\; \big[\hat{y}_0(n),\, ...,\, \hat{y}_{N_M-1}(n)\big]^{\mathrm{T}}.$$

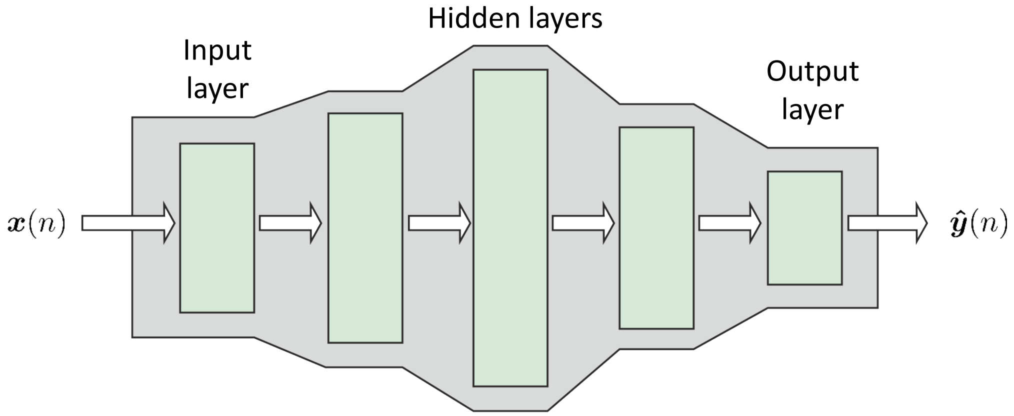


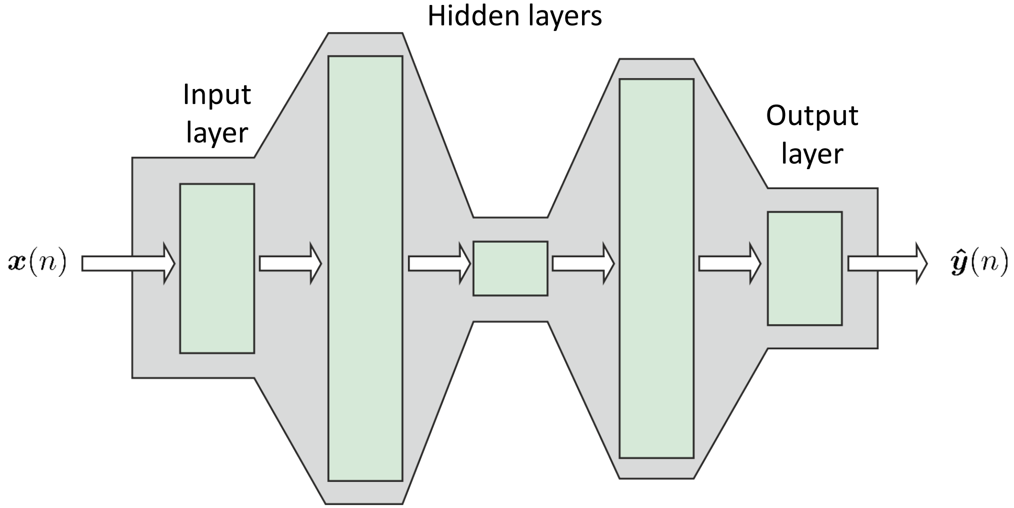
Structure of a Neural Network – Basics

Layer sizes:

- The ***input and the output layer size*** is usually given by the application. The input layer size is equal to the feature vector size and the output layer size is determined by the amount of output features.

  Sometimes ***more outputs than required*** are computed in order to modify the cost function.
- The entire ***size of the network*** (sum of all layer sizes) should be adjusted to the ***size of the available data***.
- In some applications so-called ***bottle neck layers*** are helpful.





- Motivation
- Structure of a (basic) neural network
- ***Applications of neural networks***
  - ***Real-time video object recognition***
  - ***Improving Image Resolution***
  - ***Automatic image colorization***
- Types of neural networks
- Basic training of neural networks
- Reinforcement learning

Applications of Neural Networks – Sources

Tesla:

- https://cleantechnica.com/2018/06/11/tesla-director-of-ai-discusses-programming-a-neural-net-for-autopilot-video/
- https://vimeo.com/272696002?cjevent=c27333cefa3511e883d900650a18050f

Pixel Recursive Super Resolution:

- R. Dahl, M. Norouzi and J. Shlens: ***Pixel Recursive Super Resolution***, 2017 IEEE International Conference on Computer Vision (ICCV), Venice, pp. 5449-5458, 2017.

Image colorization:

- http://iizuka.cs.tsukuba.ac.jp/projects/colorization/data/colorization_sig2016.pdf

Applications of Neural Networks – Real-time Video Object Recognition

Video object recognition for Tesla cars:

- Tesla uses ***cameras, radar and ultrasonic sensors*** to detect objects in the surrounding area. However, they rely mostly on computer vision by cameras.
- Their current system uses (mostly) a so-called ***convolutional network*** (details later on) for object recognition. New approaches use “CodeGen” (also the structure [not only the weights] of the network are adapted during the training).
- The main system for autonomous driving is a ***deep neural network***.

The following video is a full self driving demo by Tesla, where this legend is used:

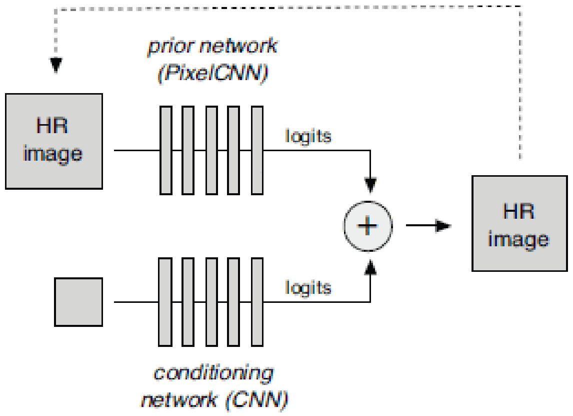
Applications of Neural Networks – Real-time Video Object Recognition

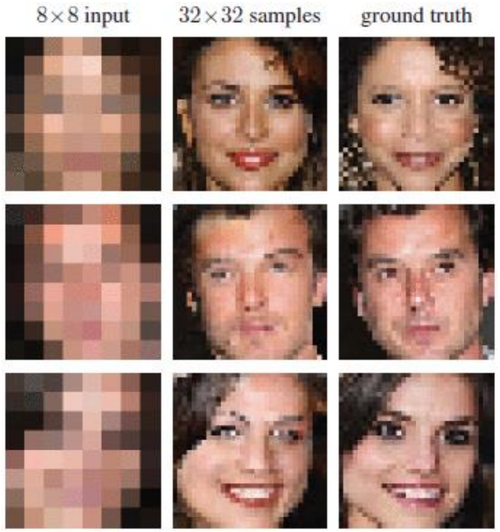
Applications of Neural Networks – Improving Image Resolution

"Super resolution is the problem of artificially enlarging a low resolution photograph to recover a plausible high resolution. […]"

Neural network types used:

- New probabilistic deep network architectures are used that are based on ***log-likelihood objectives***.
- Extension of "PixelCNNs" (conv. net.) and "ResNet" (residual net.)
- Basically two networks are used:
  - A "prior network" that captures serial dependencies of pixels (auto-regressive part of model) [PixelCNN] and
  - a "conditioning network" that captures the global structure of images (DCNN, similar to "SRResNet", feed-forward convolutional neural networks).





NN input NN output

Problems:

- As magnification increases the neural network needs to predict missing information such as:
  - complex variations of objects, viewpoints, illumination, …
  - Underspecified problem → many plausible high resolution images

Applications of Neural Networks – Automatic Image Colorization with Simultaneous Classification

Coloration of greyscale images:

- A ***convolutional network*** using low-level features to compute global features for ***classifying the image*** (rough type of image, what are the surroundings).
- A ***parallel network*** uses the same low-level features to compute ***mid-level features***.
- ***Fusion*** of global features (e.g. indoor or outdoor photo) and mid-level features are used ***for colorization*** of the image.
- Greyscale image is then used for ***luminance***.

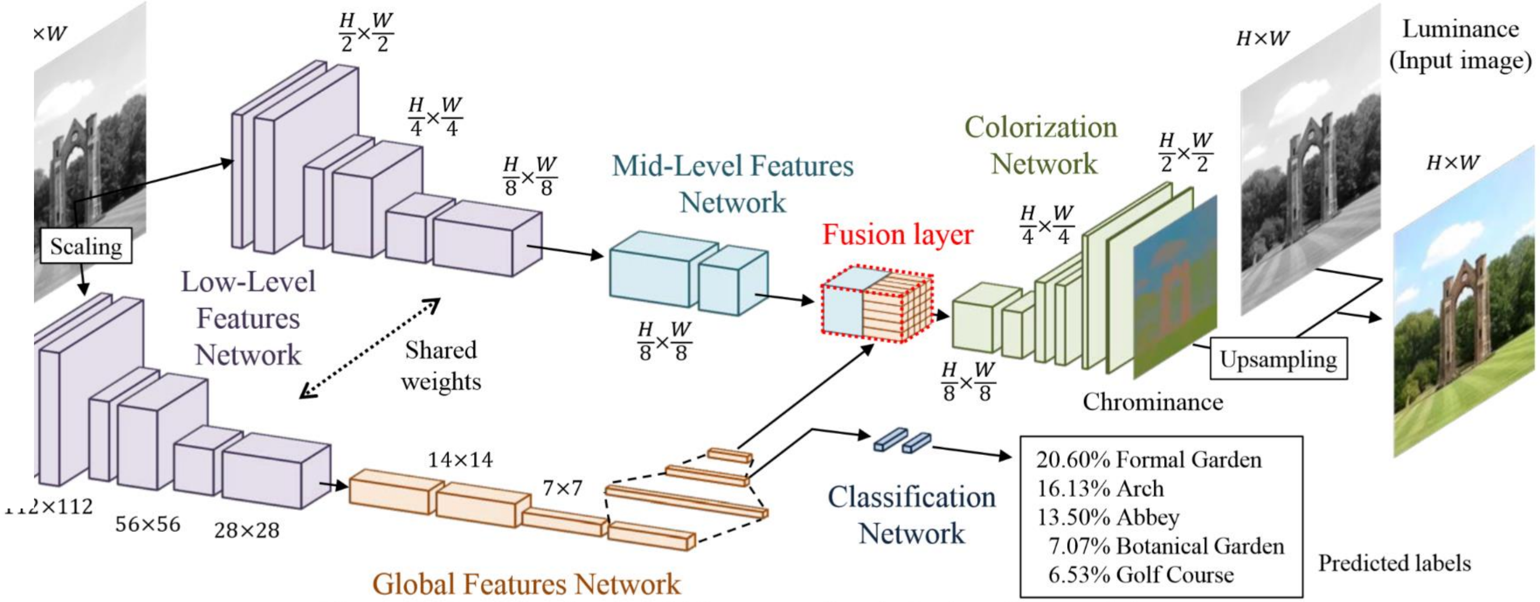


Figure 2: *Overview of our model for automatic colorization of grayscale images.*

Applications of Neural Networks – Automatic Image Colorization with Simultaneous Classification

Other examples:

(a) Cranberry Picking, Sep. 1911 (b) Burns Basement, May 1910 (c) Miner, Sep. 1937 (d) Scott's Run, Mar. 1937

Typical failure cases:

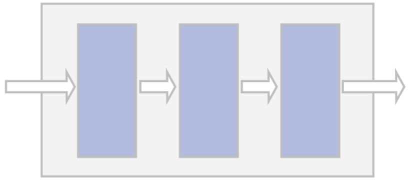
Input Ground truth Proposed


Contents

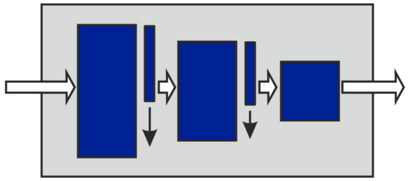
Types of Neural Networks

Network structure(s):

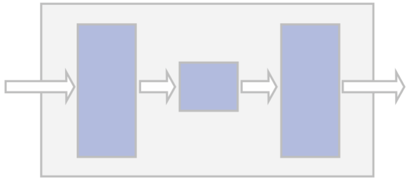
Multilayer perceptron



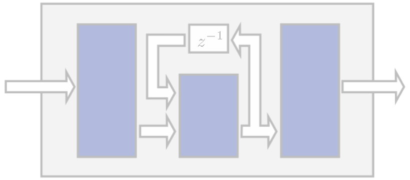
Convolutional neural network



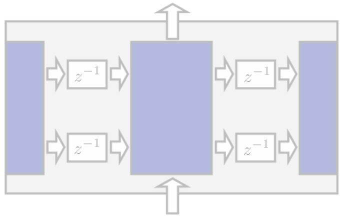
Autoencoder



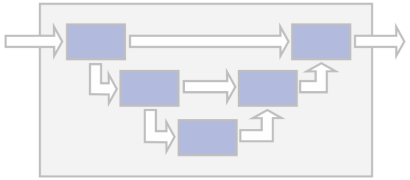
Recurrent neural network



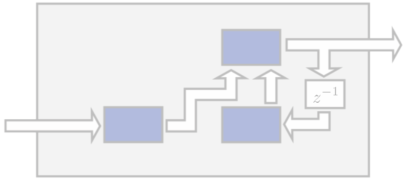
LSTM network



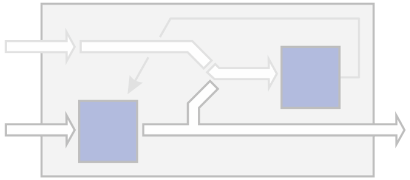
"Unet"



Attention-based network



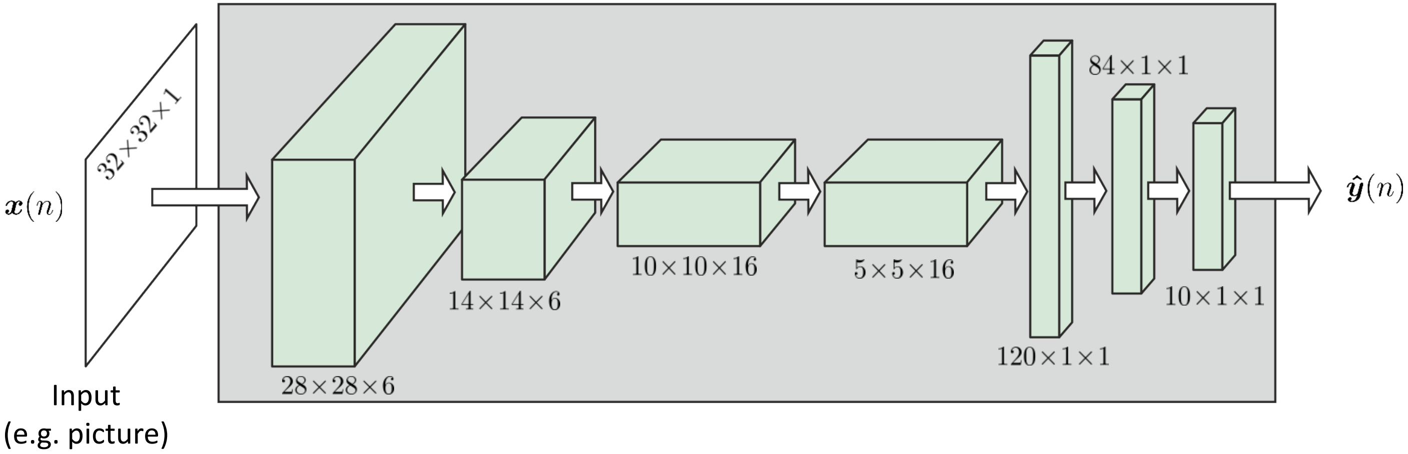
Generative adversarial network



Types of Neural Networks

Convolutional neural networks (CNNs):

- CNNs were part of the ***early times in deep approaches***.
- They are often applied in ***image*** and ***video applications***.
- Often ***three-dimensional layers*** with special ***ReLU activation functions*** followed by ***pooling*** (next slides) are used.
- The weights of the layers are used as in a “conventional” convolution, meaning that the ***same weights*** are used very often (e.g. for edge detection).

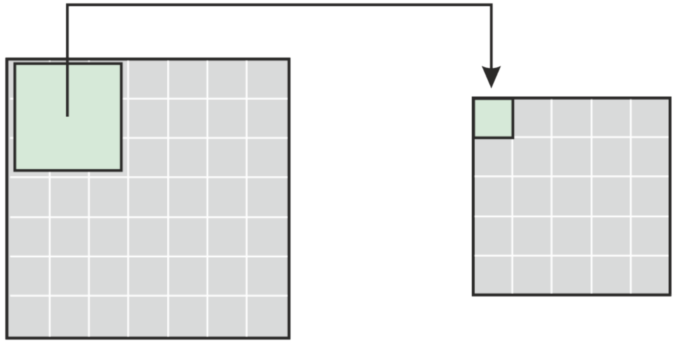


Source: Adopted from Charu C Aggarwal, ***Neural Networks and Deep Learning***, Springer, 2018

Types of Neural Networks

Convolutional neural networks (CNNs):

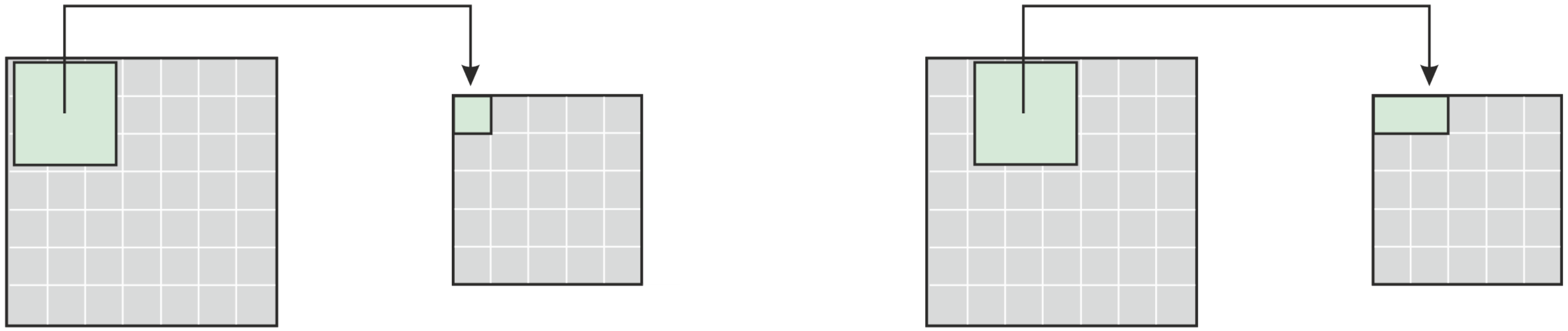
- Convolutional layers
  - Computing a ***weighted sum*** of a ***subset of the input data*** and applying an ***activation function*** to the weighted sum.



Types of Neural Networks

Convolutional neural networks (CNNs):

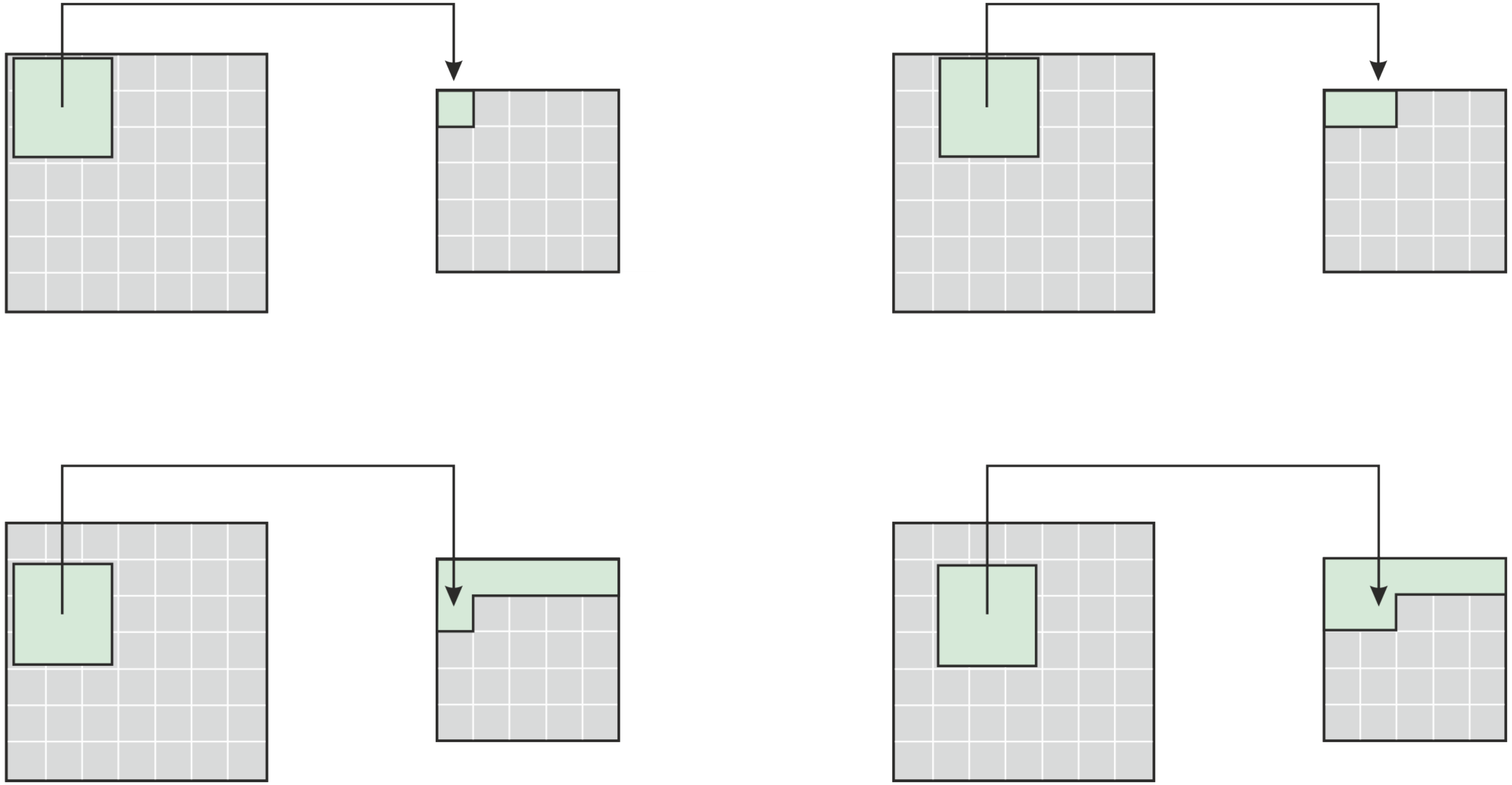
- Convolutional layers
  - Computing a ***weighted sum*** of a ***subset of the input data*** and applying an ***activation function*** to the weighted sum.
  - ***Shift the weighting filter*** (***kernel***) with the same coefficients but now to different input data.



Types of Neural Networks

Convolutional neural networks (CNNs):

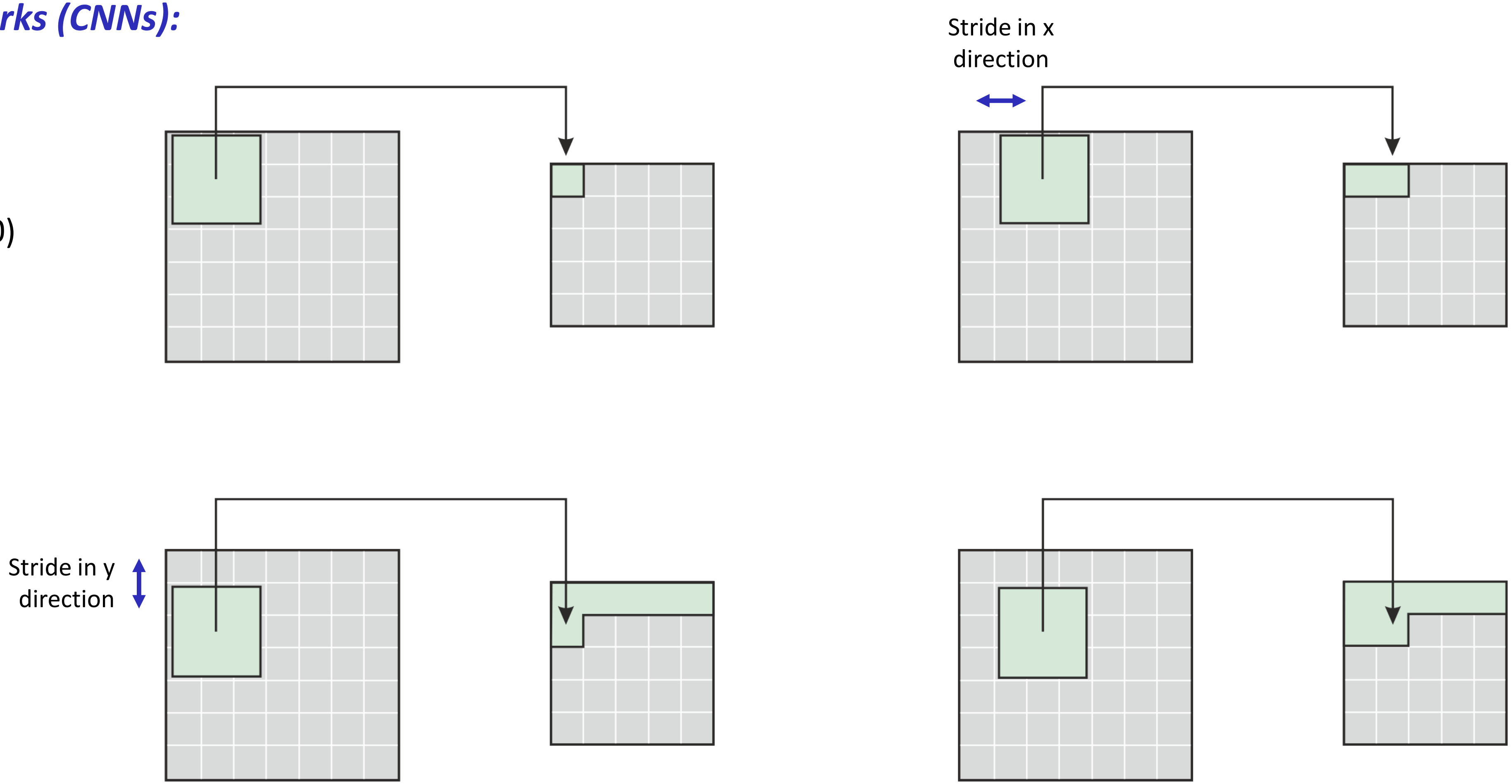
- Convolutional layers
  - Computing a ***weighted sum*** of a ***subset of the input data*** and applying an ***activation function*** to the weighted sum.
  - ***Shift the weighting filter*** (***kernel***) with the same coefficients but now to different input data.
  - Do this over the ***entire range*** of the input data.



Types of Neural Networks

Convolutional neural networks (CNNs):

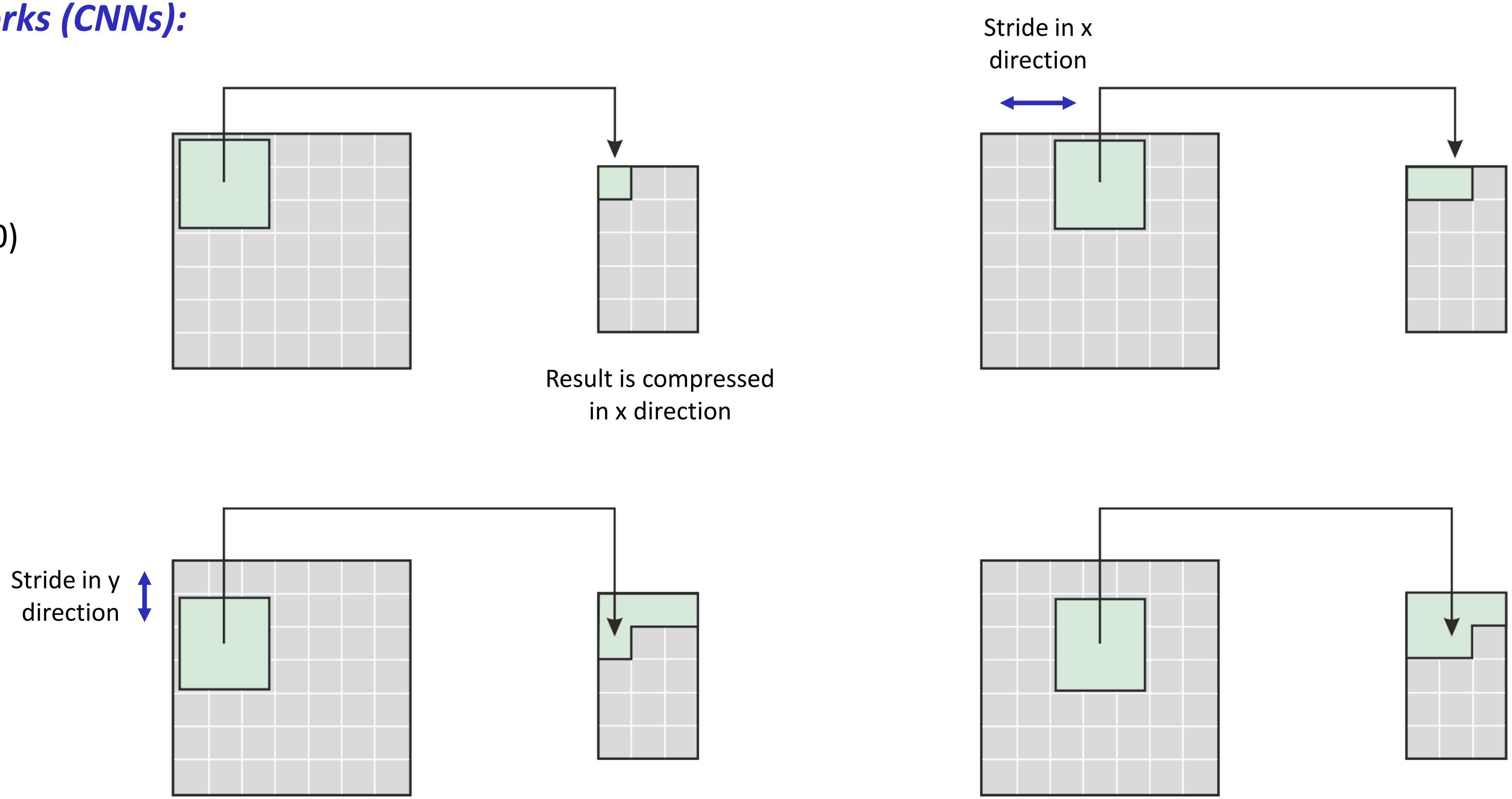
- Parameters of CNNs
  - ***Stride*** (x = 1, y = 1)
  - Padding (x = 0, y = 0)
  - Dilation (x =0, y = 0)



Types of Neural Networks

Convolutional neural networks (CNNs):

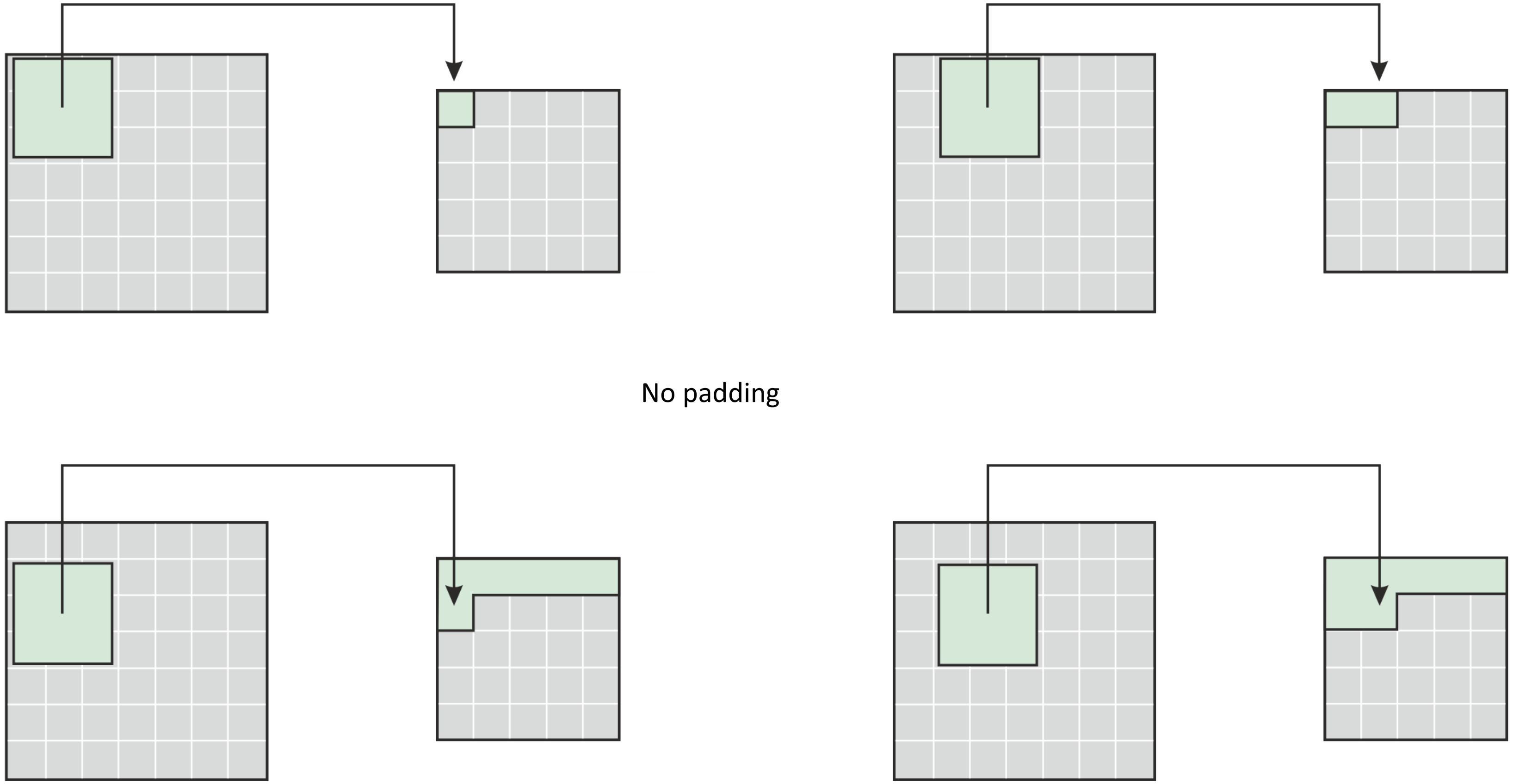
- Parameters of CNNs
  - ***Stride*** (x = 2, y = 1)
  - Padding (x = 0, y = 0)
  - Dilation (x =0, y = 0)



Types of Neural Networks

Convolutional neural networks (CNNs):

- Parameters of CNNs
  - Stride (x = 1, y = 1)
  - ***Padding*** (x = 0, y = 0)
  - Dilation (x =0, y = 0)

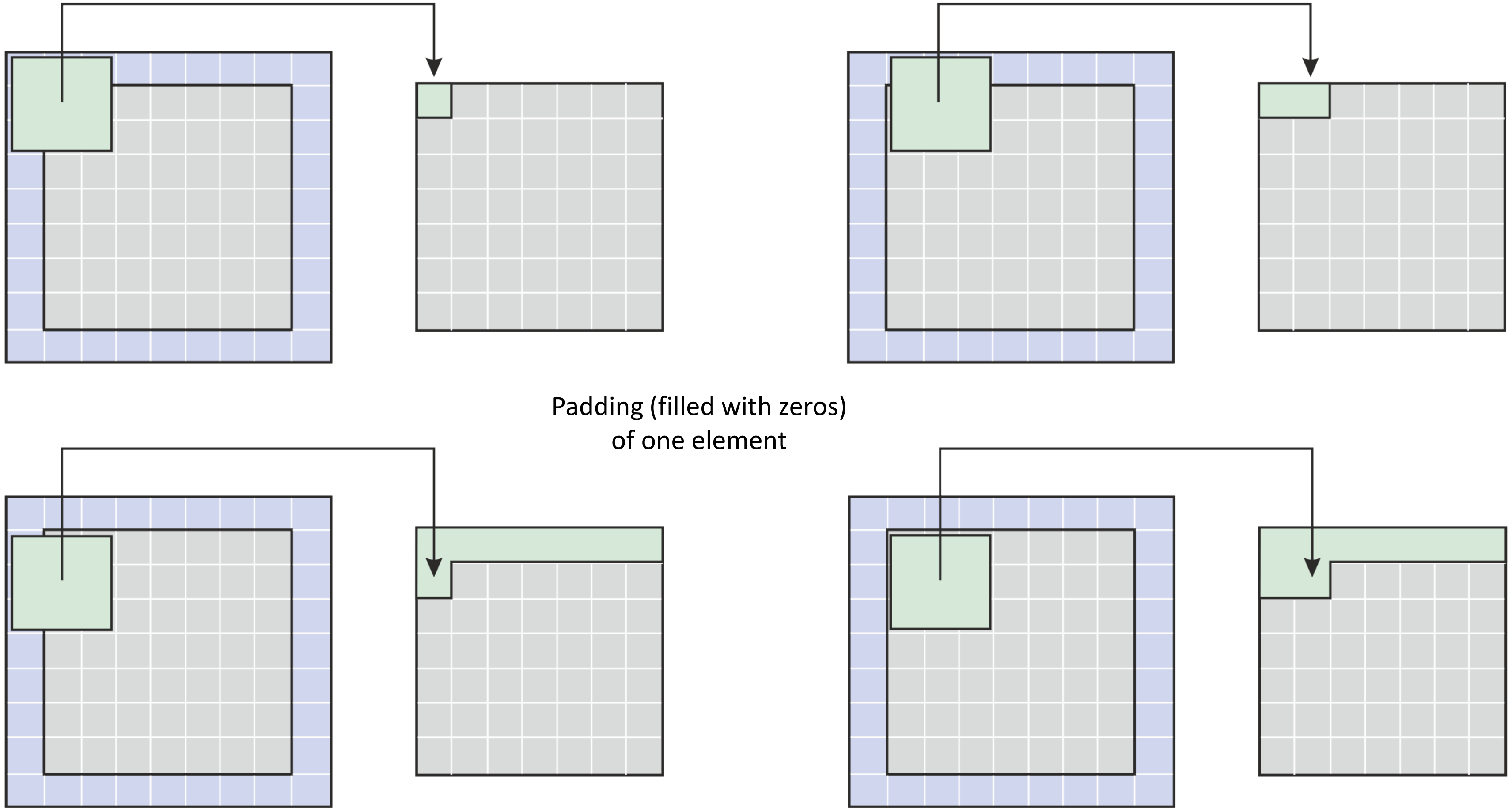


No padding

Types of Neural Networks

Convolutional neural networks (CNNs):

- Parameters of CNNs
  - Stride (x = 1, y = 1)
  - ***Padding*** (x = 1, y = 1)
  - Dilation (x =0, y = 0)



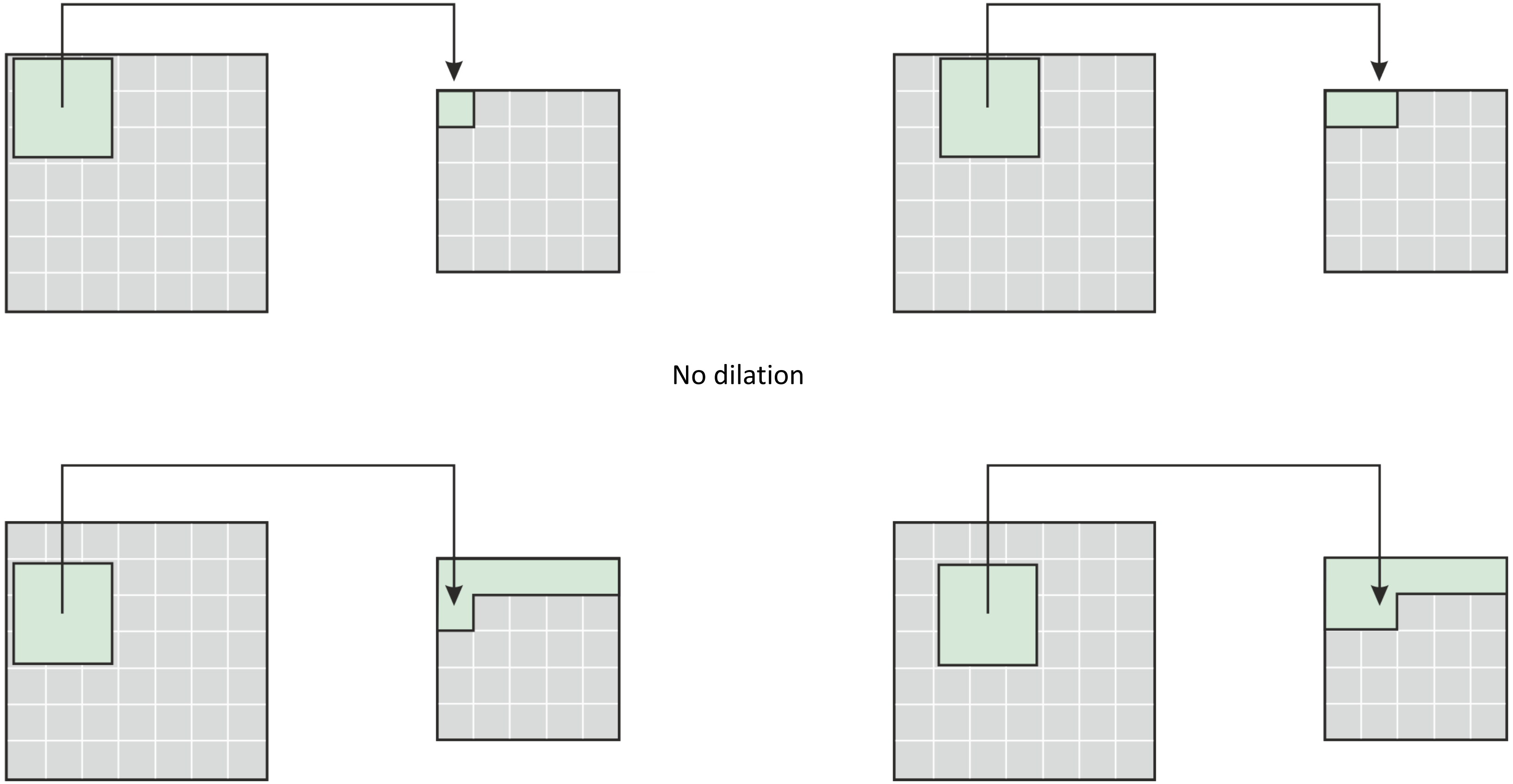
Padding (filled with zeros) of one element

Padding allows to keep the original data size!

Types of Neural Networks

Convolutional neural networks (CNNs):

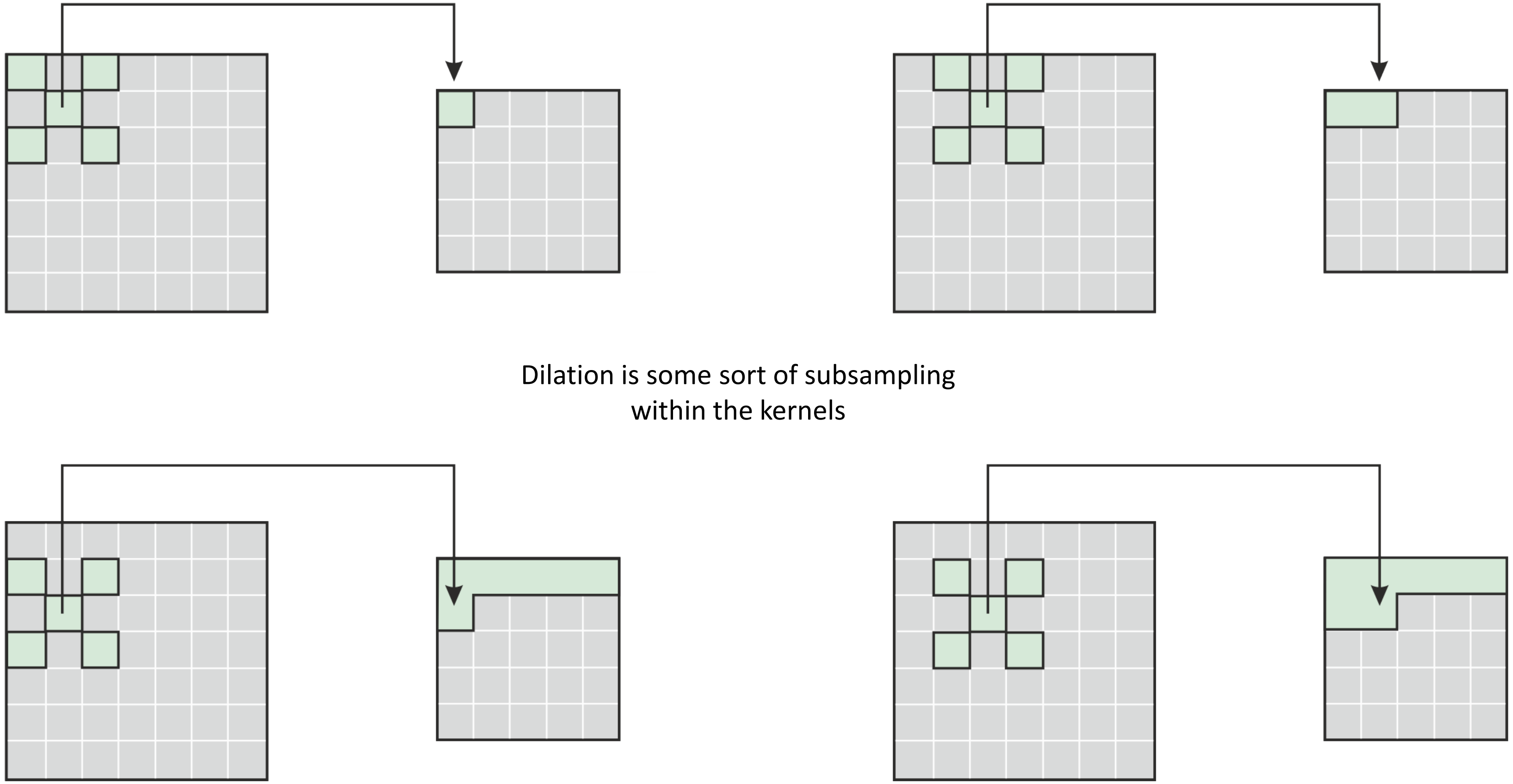
- Parameters of CNNs
  - Stride (x = 1, y = 1)
  - Padding (x = 0, y = 0)
  - ***Dilation*** (x =0, y = 0)



No dilation

Convolutional neural networks (CNNs):

- Parameters of CNNs
  - Stride (x = 1, y = 1)
  - Padding (x = 0, y = 0)
  - ***Dilation*** (x =1, y = 1)

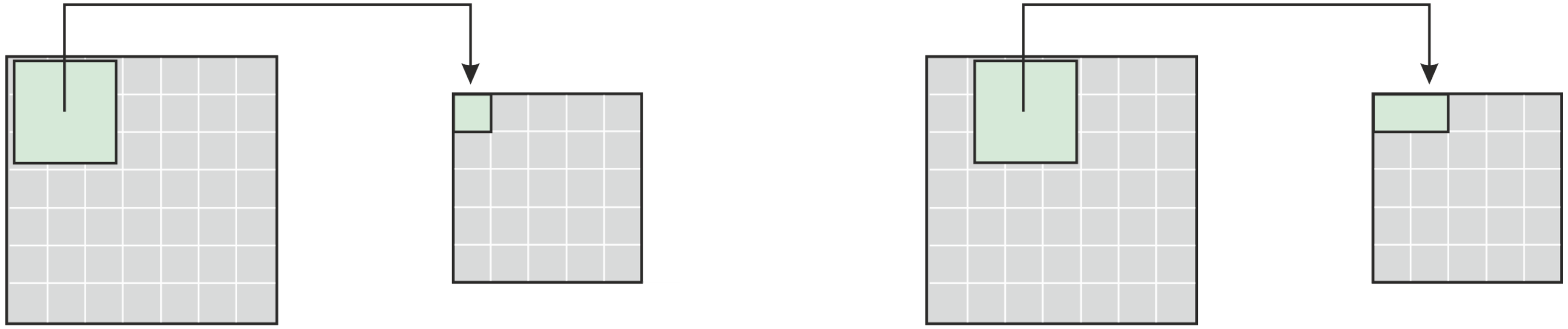


Dilation is some sort of subsampling within the kernels

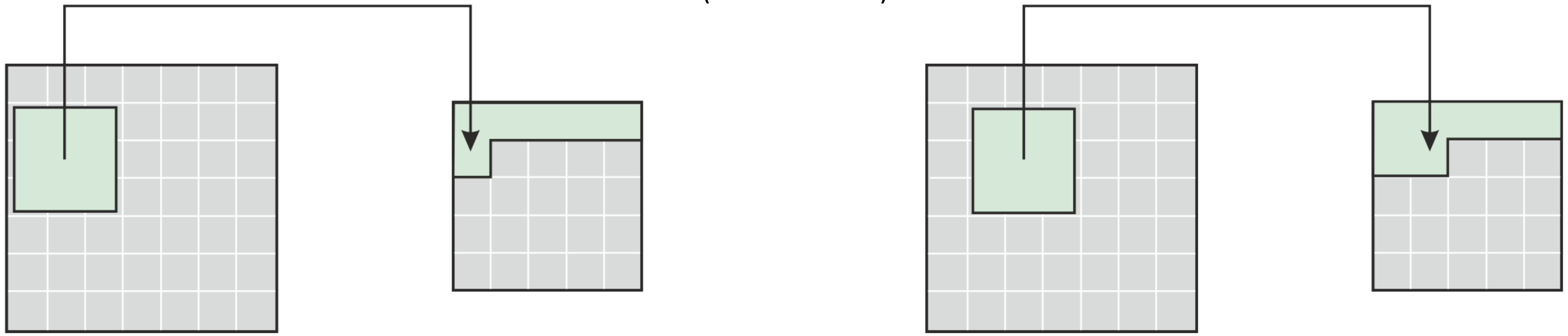
Types of Neural Networks

Convolutional neural networks (CNNs):

- Kernels of CNNs
  - ***First kernel***



A multitude of kernels leads to an extra dimension for the intermediate data structures (see next slide)

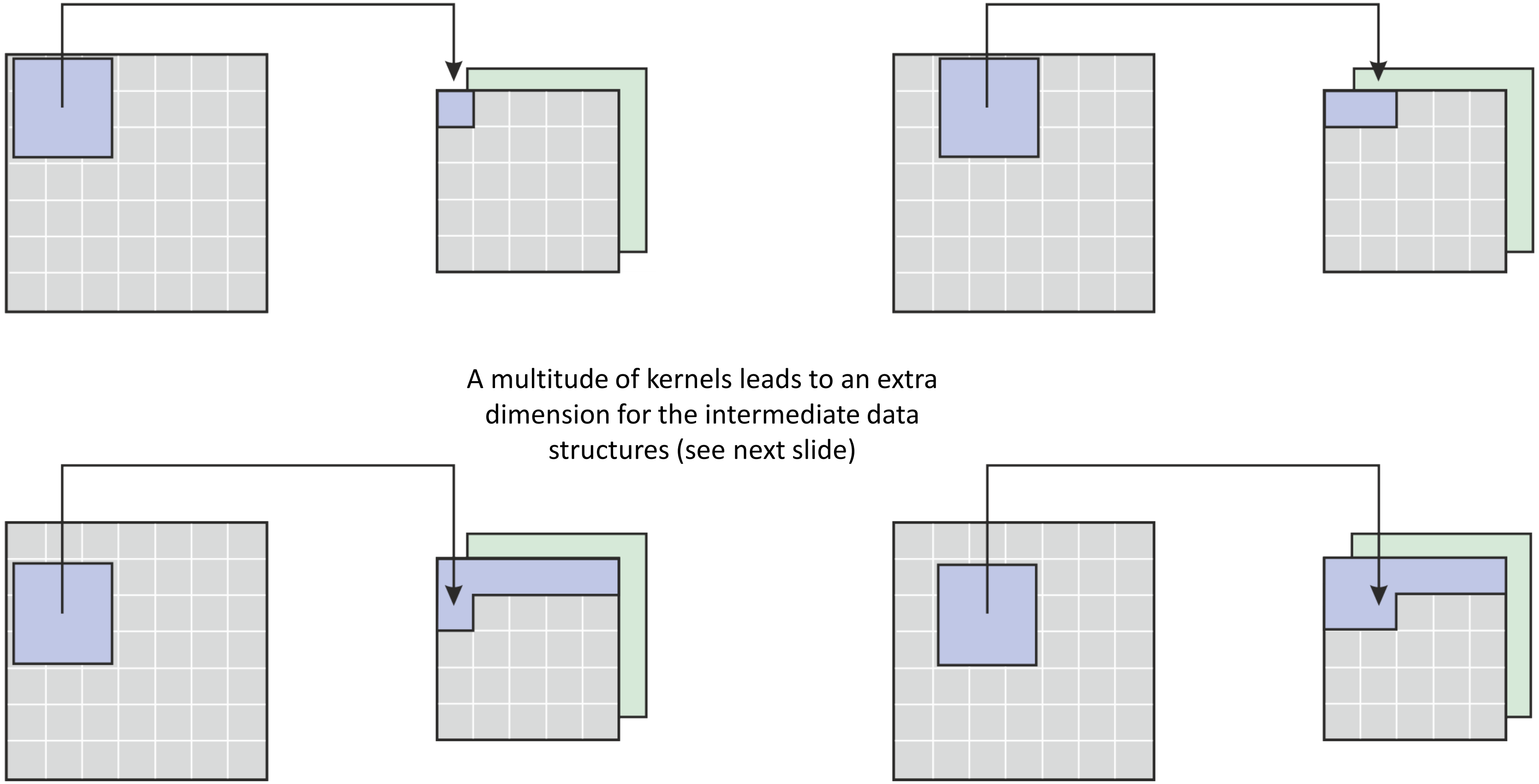


C | A | U
Christian-Albrechts-Universität zu Kiel

Types of Neural Networks

Convolutional neural networks (CNNs):

- Kernels of CNNs
  - First kernel
  - ***Second kernel***

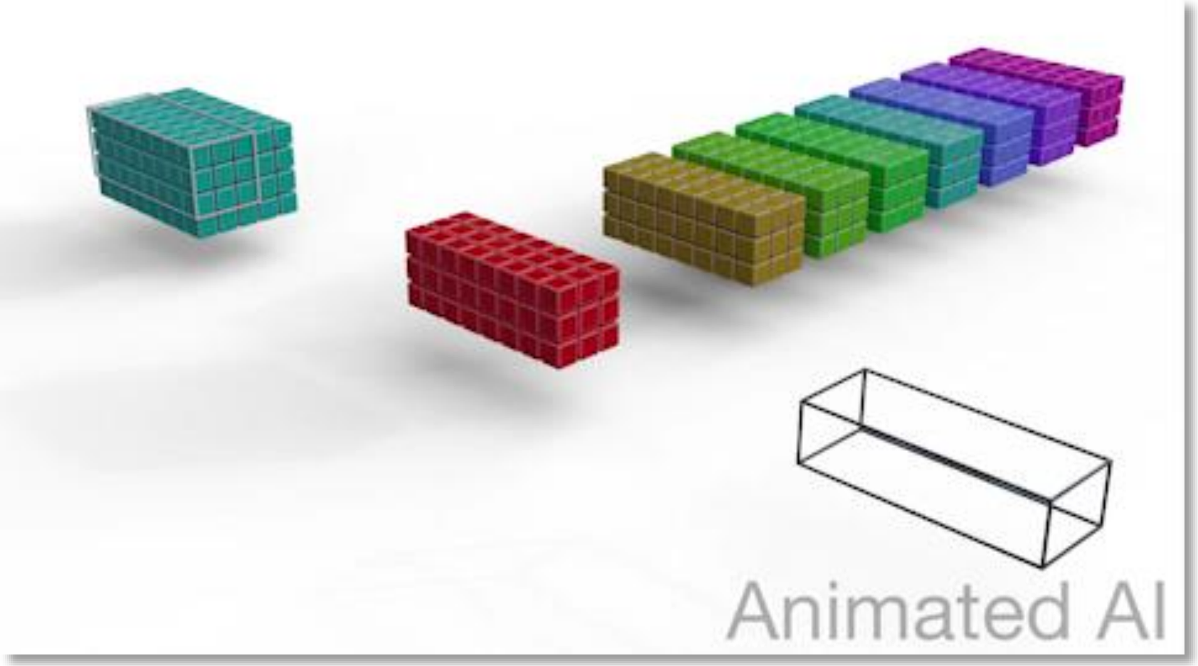


A multitude of kernels leads to an extra dimension for the intermediate data structures (see next slide)

Types of Neural Networks

Convolutional neural networks (CNNs):

- Kernels of CNNs
  - First kernel
  - Second kernel
  - *Usually "3D processing"*



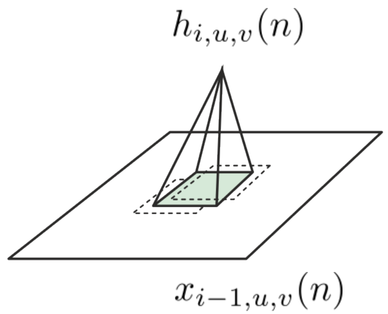
https://animatedai.github.io/

Types of Neural Networks

Convolutional neural networks (CNNs):

- Pooling can be realized e.g. by computing the ***maximum over an overlapping and moving part*** of the input:

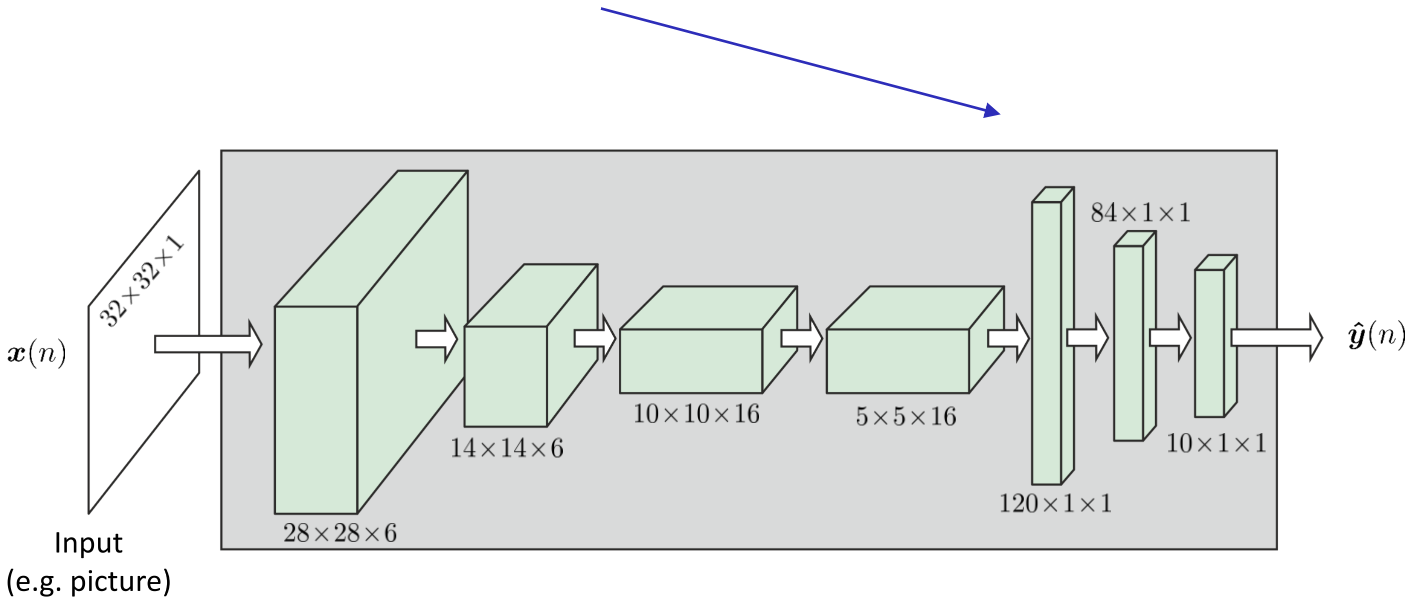
$$h_{i,u,v}(n) = f_{\text{pool}}\big(\boldsymbol{X}_{i-1}(n)\big) = \max_{l \in \{-N,N\}} \left\{ \max_{k \in \{-N,N\}} \left\{ x_{i-1,u+l,v+k}(n) \right\} \right\}$$



- The ***basic idea*** behind pooling is that it is important that a specific pattern is found in a certain area, but it's not important where exactly.
- Pooling is often combined with subsampling of the output structures (striding).

Convolutional neural networks (CNNs):

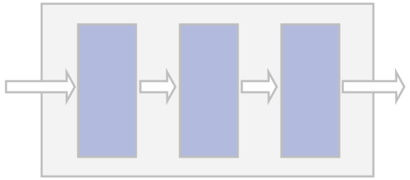
- At the end of the network structure the 3D data structures are rearranged into a single vector and a “conventional” network is used for generating the final output.



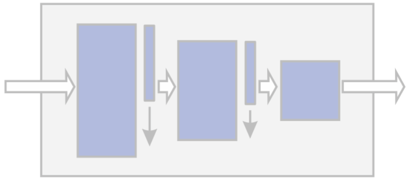
Types of Neural Networks

Network structure(s):

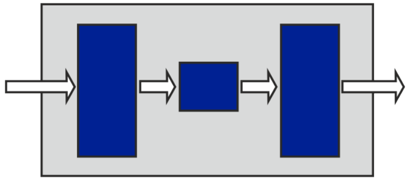
Multilayer perceptron



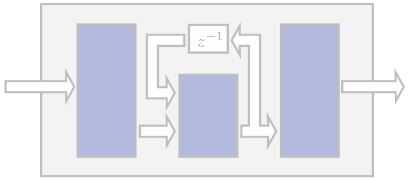
Convolutional neural network



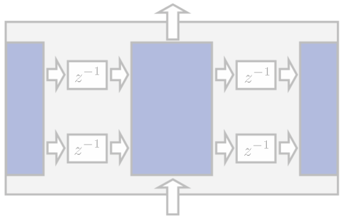
Autoencoder



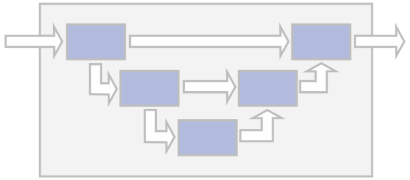
Recurrent neural network



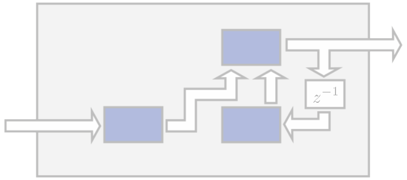
LSTM network



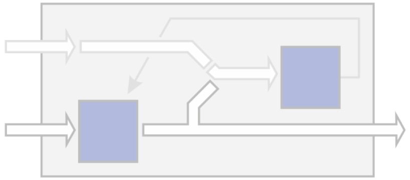
“U-net”



Attention-based network



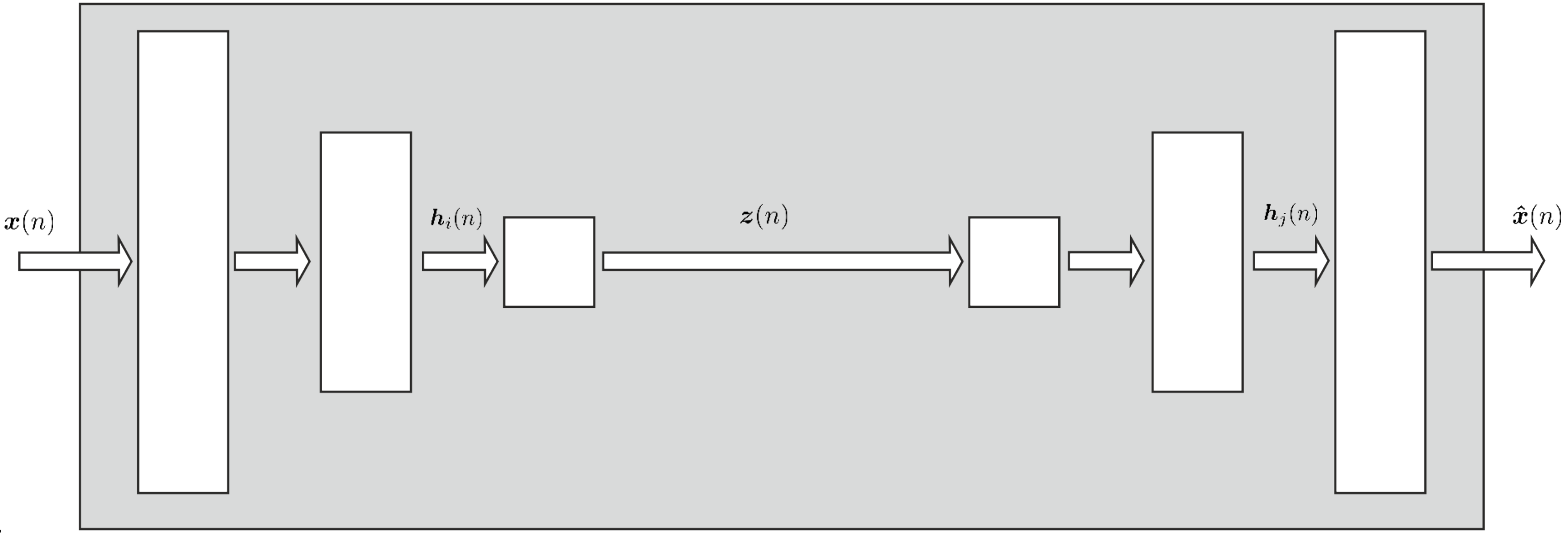
Generative adversarial network



Types of Neural Networks

Autoencoder networks:

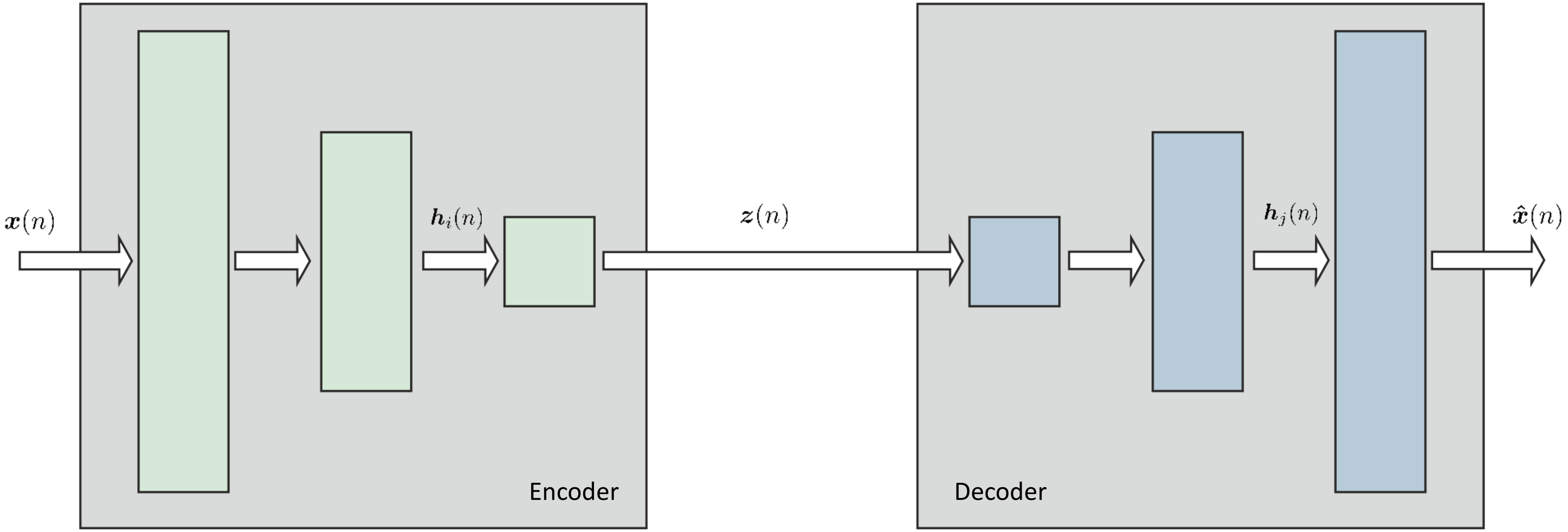
- Instead of mapping input vectors on features, it is tried to ***reconstruct the input*** at the output.
- In the middle of the network a ***bottleneck layer*** is used.
- This could be used for ***data compression*** (in some sense similar to a codebook)



Types of Neural Networks

Autoencoder networks:

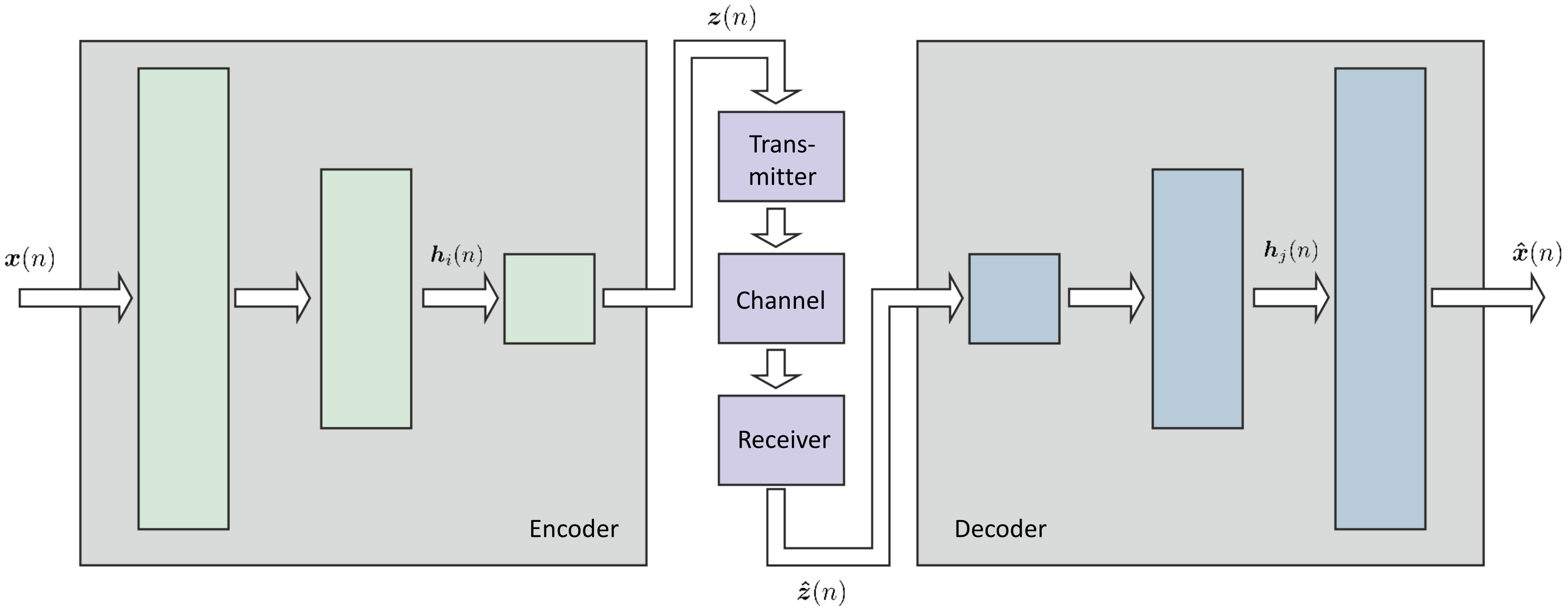
- The first part of the network is called ***(auto-) encoder***.
- The second part is called ***(auto-) decoder***.
- Can be seen as a ***nonlinear extension of a PCA-based data compression***.



Types of Neural Networks

Autoencoder networks:

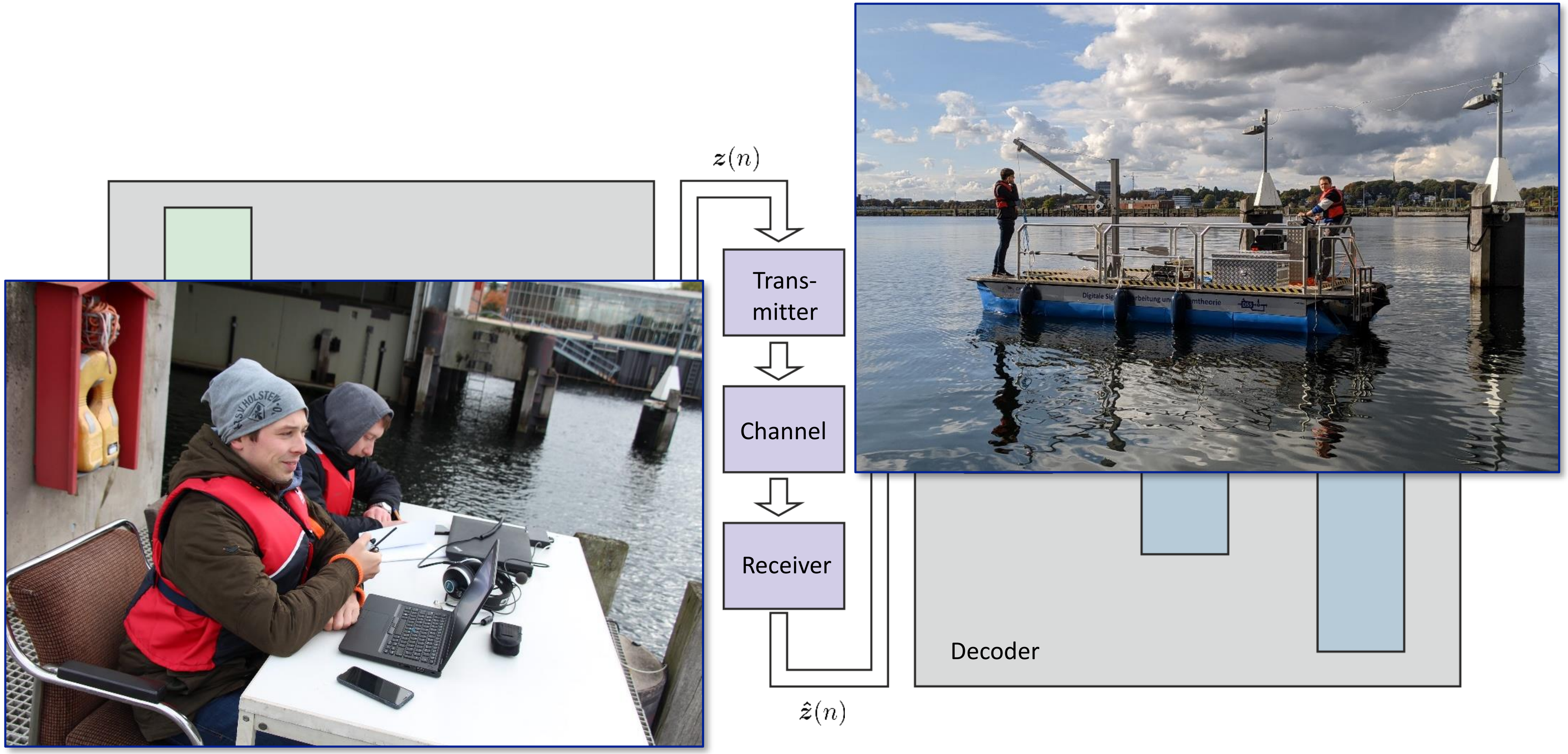
- Application example: ***under-water speech transmission***
- The ***spectral envelope*** of short speech frames is coded and transmitted (digital part).
- The ***residual signal*** is transmitted in an analog manner.



Types of Neural Networks

Autoencoder networks:

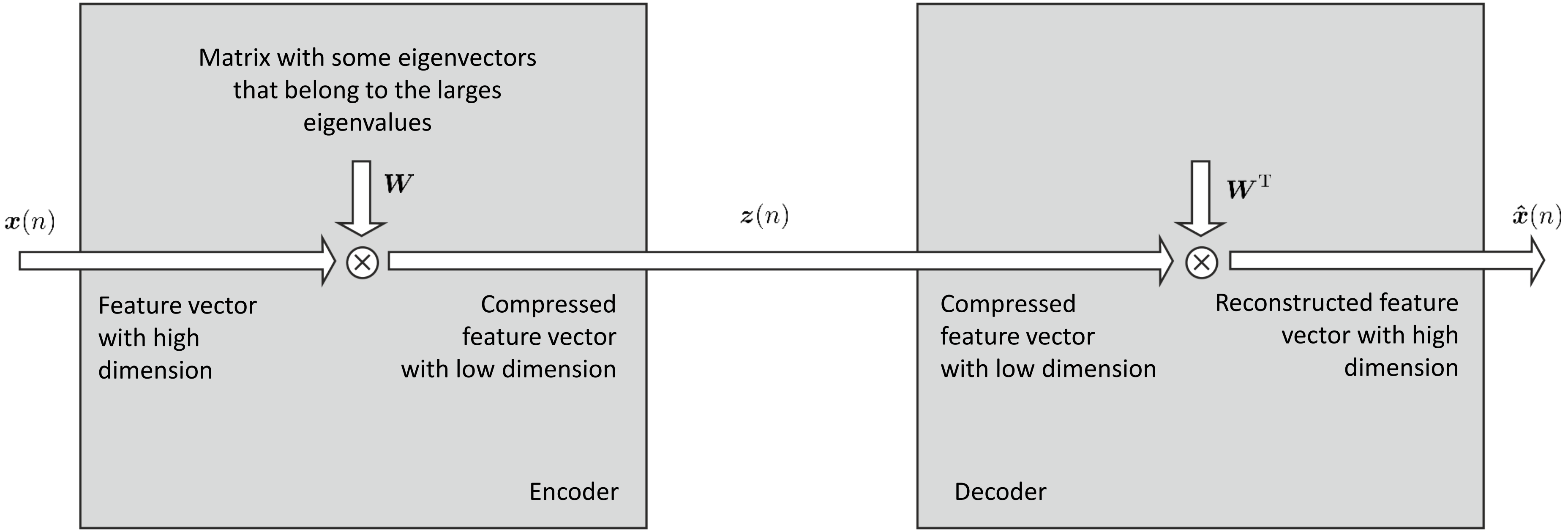
- Application example: ***under-water speech transmission***
- The ***spectral envelope*** of short speech frames is coded and transmitted (digital part).
- The ***residual signal*** is transmitted in an analog manner.



Types of Neural Networks

Autoencoder networks:

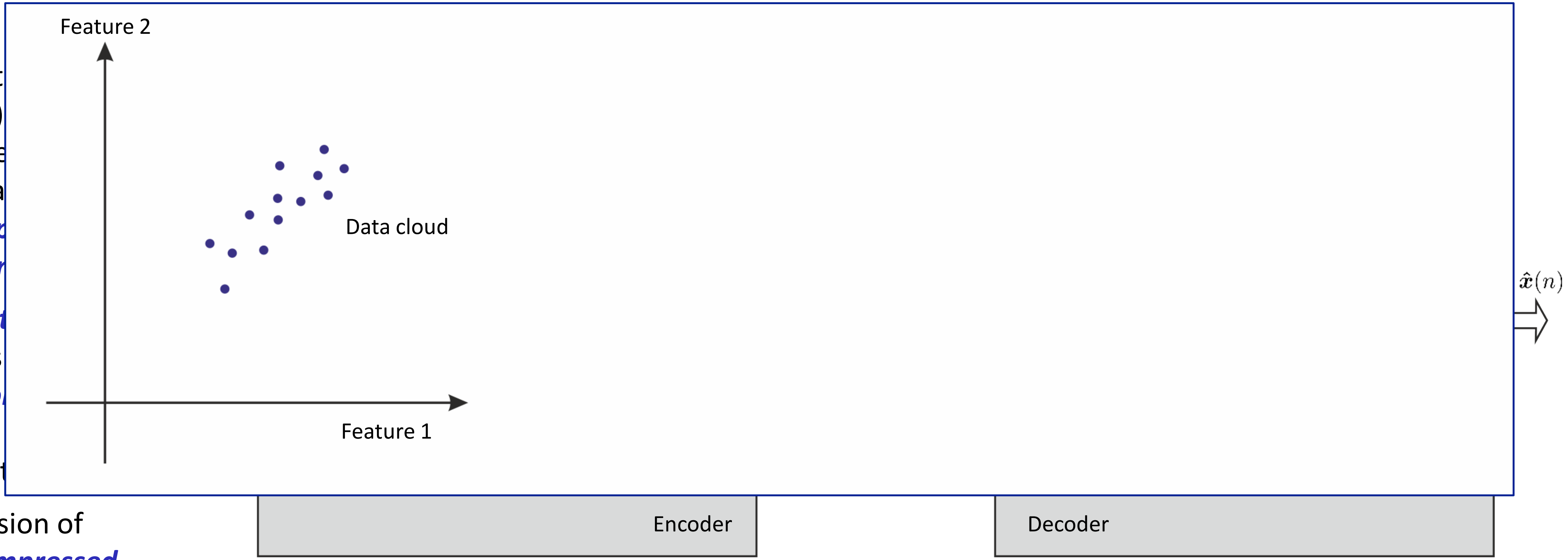
- "Conventional" (linear) data compression by means of PCA (***principle component analysis***).
- ***Eigenvectors*** and -values of the ***autocorrelation matrix*** are computed.
- Transmission of the ***compressed feature vectors***.



Types of Neural Networks

Autoencoder networks:

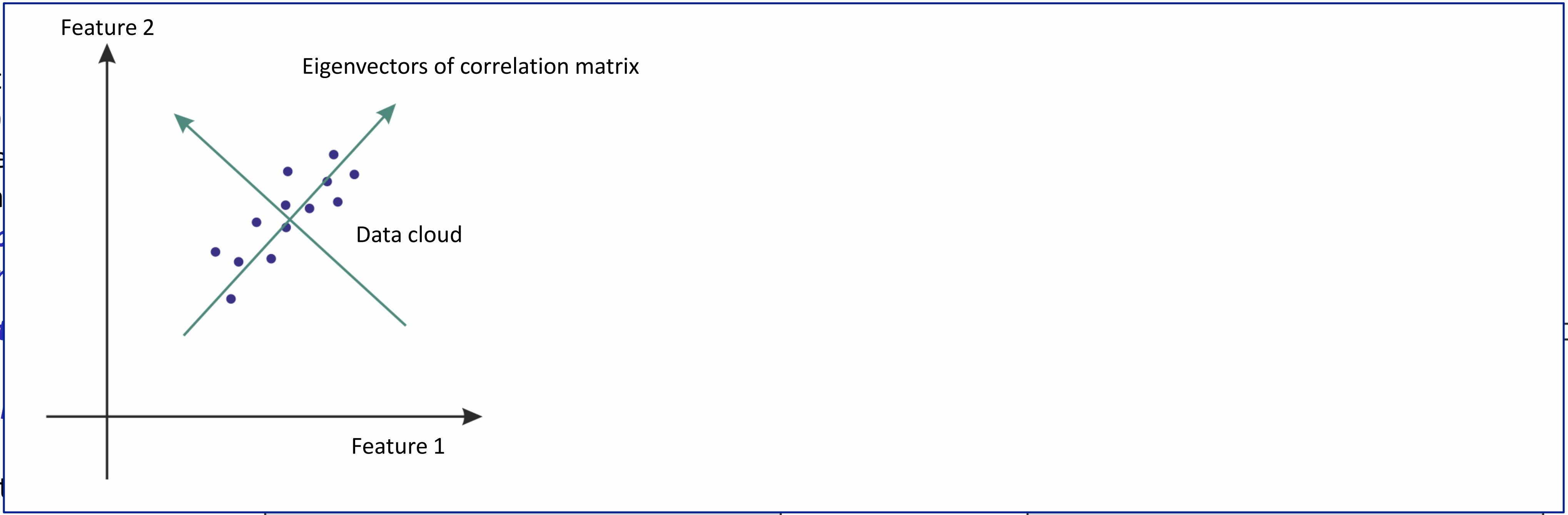
- “Convent
  (linear)
  compre
  by mea
  (***princip***
  nent ar
- ***Eigenvect***
  -values
  autoco
  matrix
  comput
- Transmission of the ***compressed feature vectors***.



Types of Neural Networks

Autoencoder networks:

- "Convent
  (linear)
  compre
  by mea
  (***princip***
  nent ar
- ***Eigenvect***
  -values
  autoco
  matrix
  comput
- Transmission of the ***compressed feature vectors***.



$\hat{\boldsymbol{x}}(n)$

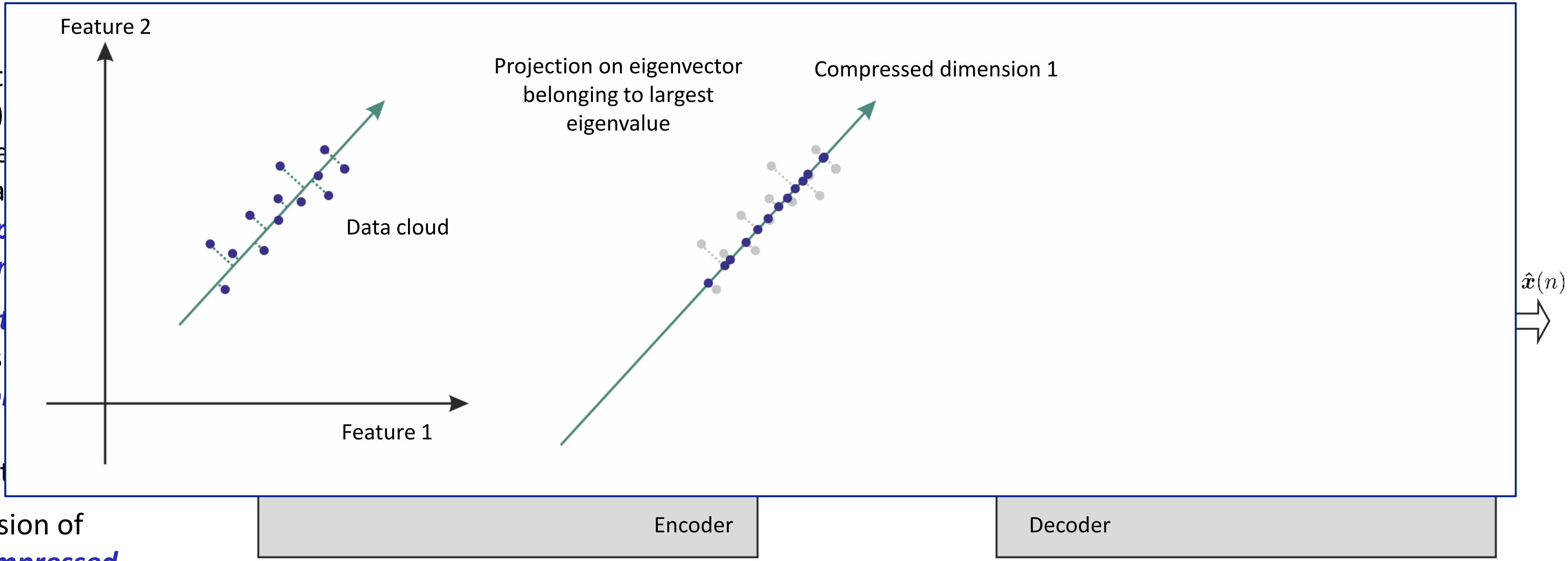
Encoder

Decoder

Types of Neural Networks

Autoencoder networks:

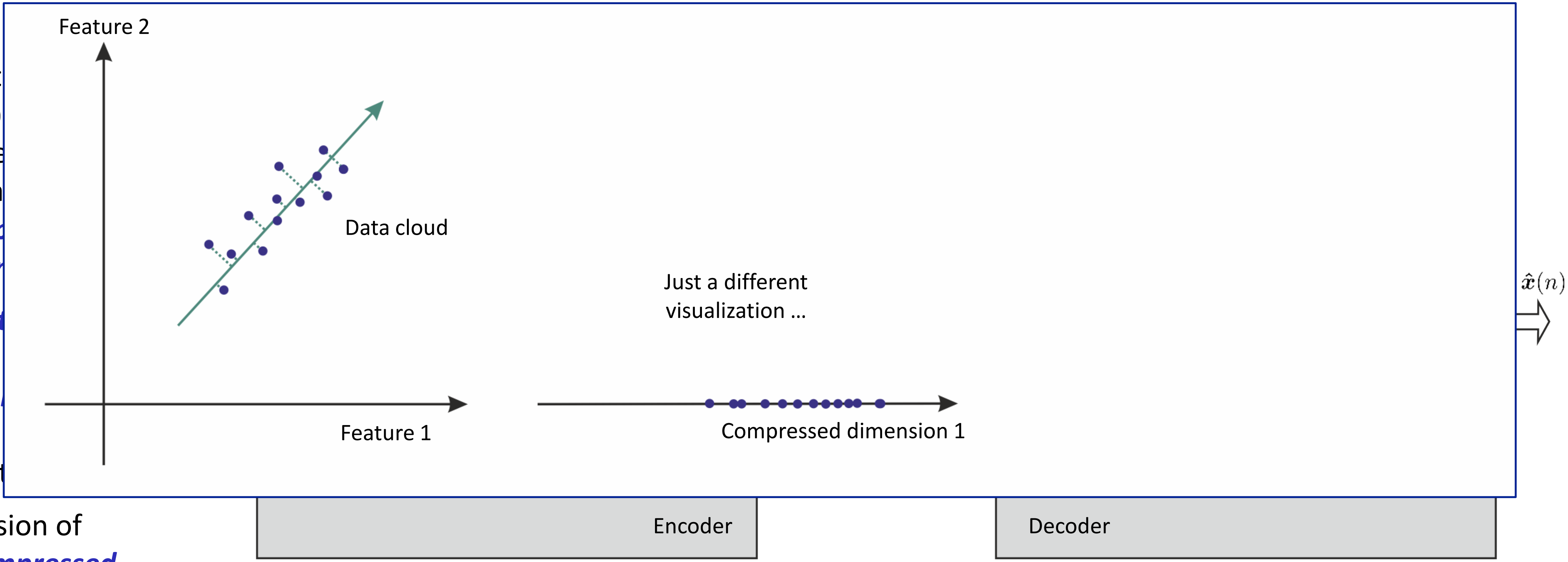
- "Convent
  (linear)
  compre
  by mea
  (*princip*
  nent ar
- *Eigenvect*
  -values
  autoco
  matrix
  comput
- Transmission of
  the *compressed*
  feature vectors.



Types of Neural Networks

Autoencoder networks:

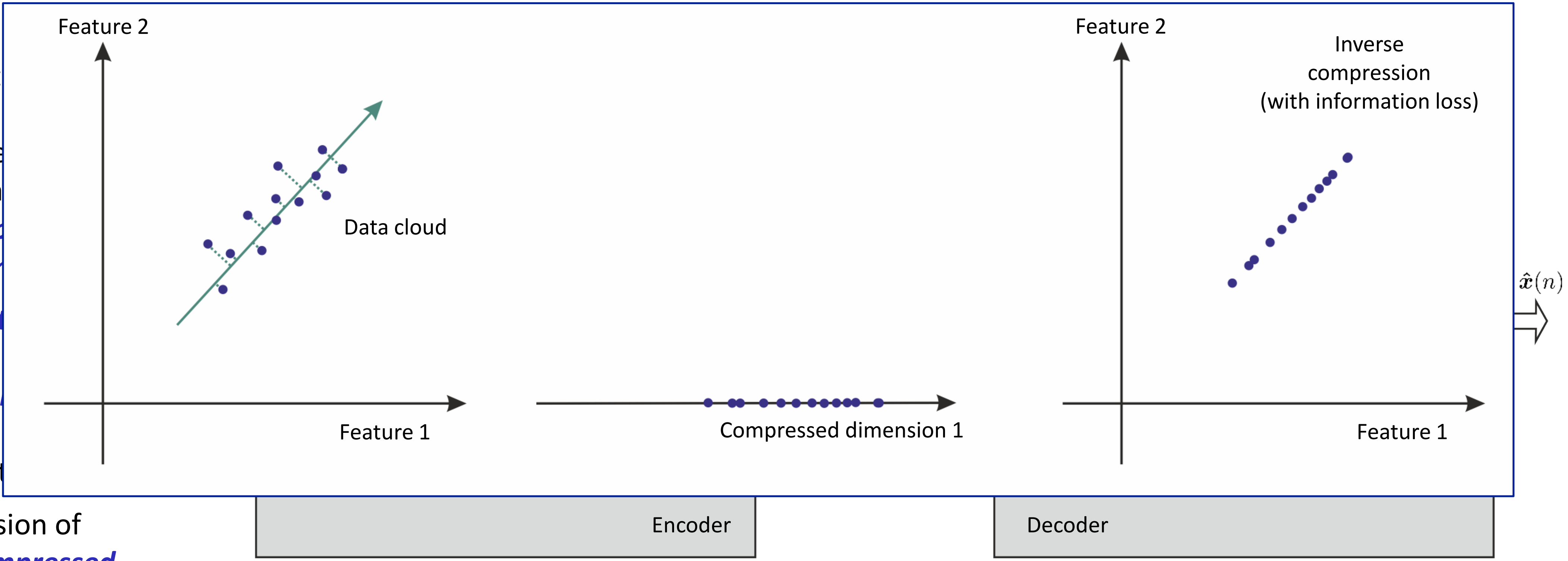
- "Convent
  (linear)
  compre
  by mea
  (***princip***
  nent ar
- ***Eigenvect***
  -values
  autoco
  matrix
  comput
- Transmission of the ***compressed feature vectors***.



Types of Neural Networks

Autoencoder networks:

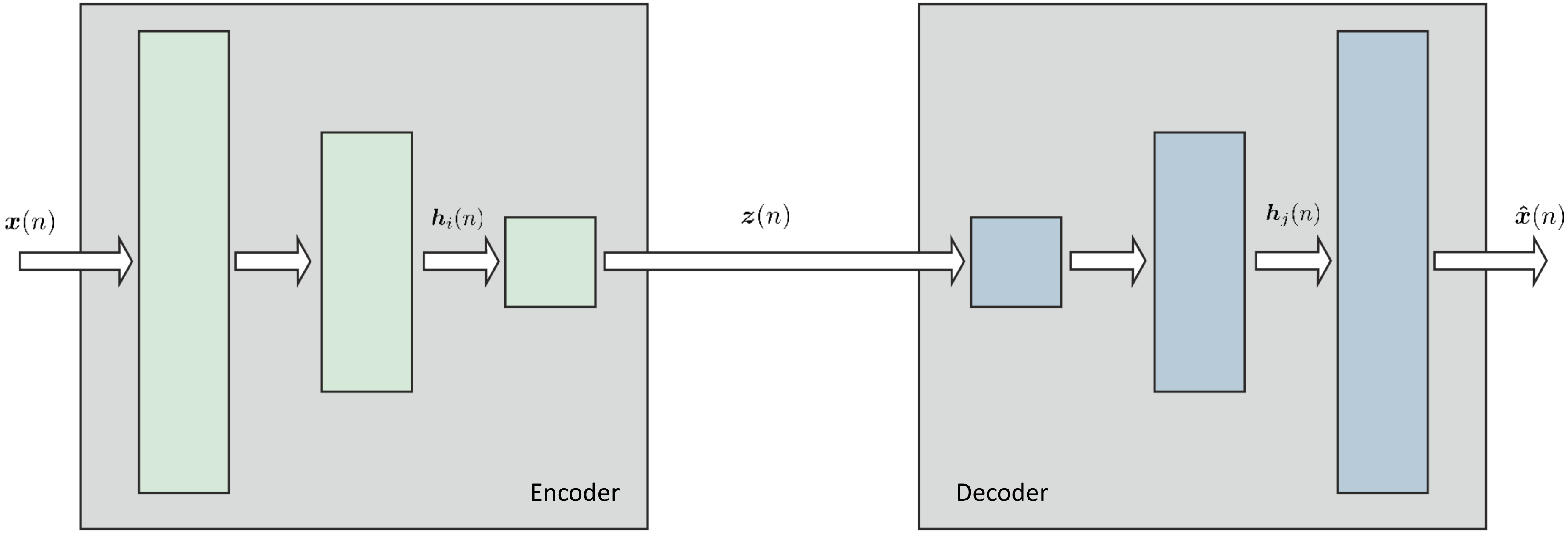
- “Convent
  (linear)
  compre
  by mea
  (*princip*
  nent ar
- ***Eigenvect***
  -values
  autoco
  matrix
  comput
- Transmission of
  the ***compressed***
  feature vectors.



Types of Neural Networks

Variational autoencoder networks:

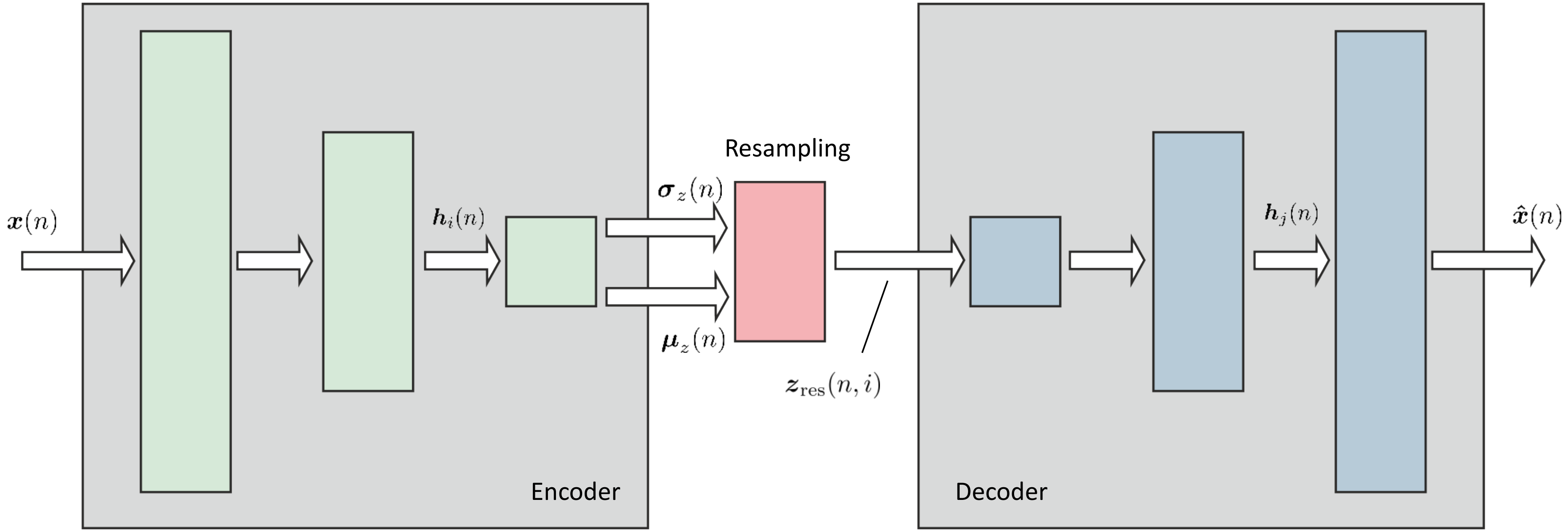
- In the basic setup ***overfitting*** and undesired behavior for "***unseen***" data occurs.
- This can be improved by ***modelling*** also the ***distribution of the latent variables*** (as a GMM).



Types of Neural Networks

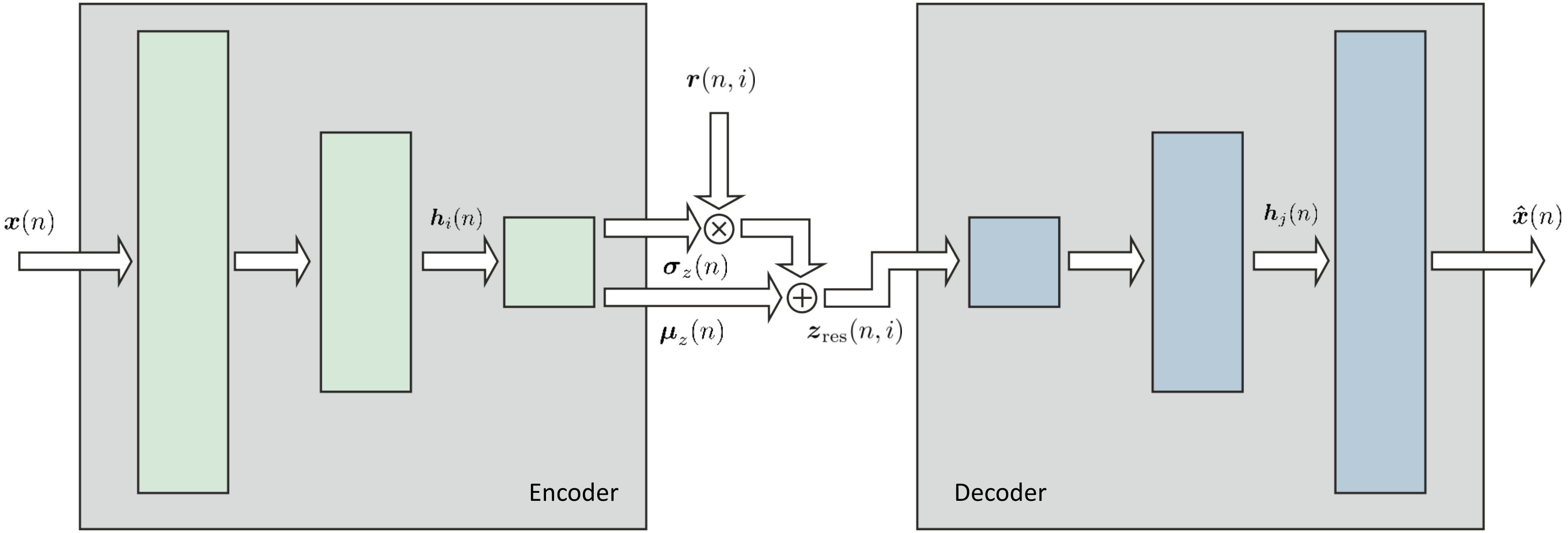
Variational autoencoder networks:

- Now the ***GMM parameters*** are ***estimated by the encoder***.
- Afterwards ***resampling*** is applied to vary the data and increase robustness to outliers.
- However, this is ***critical*** for training based on ***backpropagation***.



Variational autoencoder networks:

- With a little trick a random vector can be created that still ***allows pack propagation to work***.
- Using a ***random process generator*** resampled feature vectors are created:



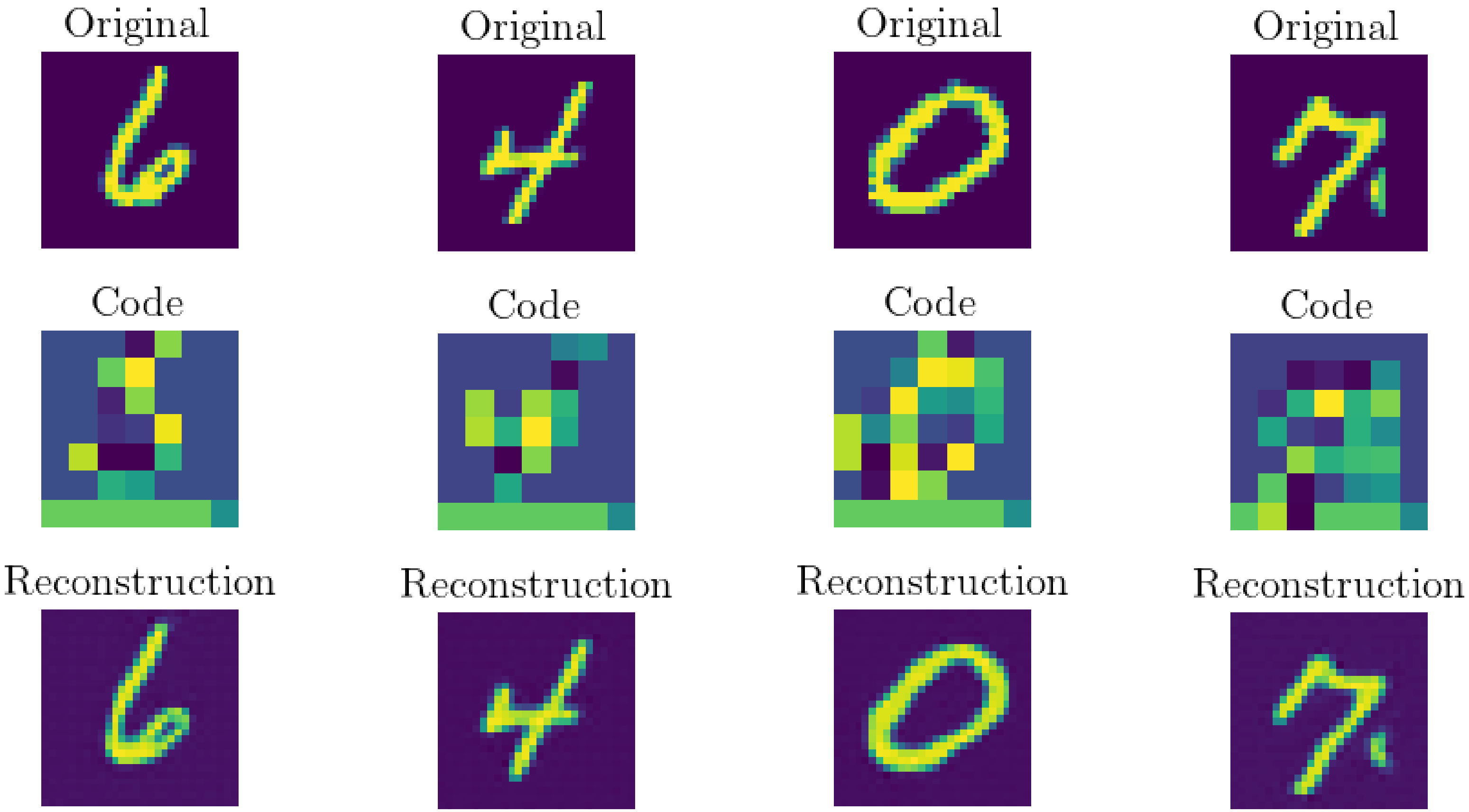
$$\boldsymbol{z}_{\mathrm{res}}(n) \;=\; \boldsymbol{\mu}_z(n) + \mathrm{diag}\{\boldsymbol{\sigma}_z(n)\}\,\boldsymbol{r}(n)$$

Types of Neural Networks

VQ-AE Example (MNIST):

- MNIST consists of handwritten digits
- Codes are assigned during training
- Basis for things like:

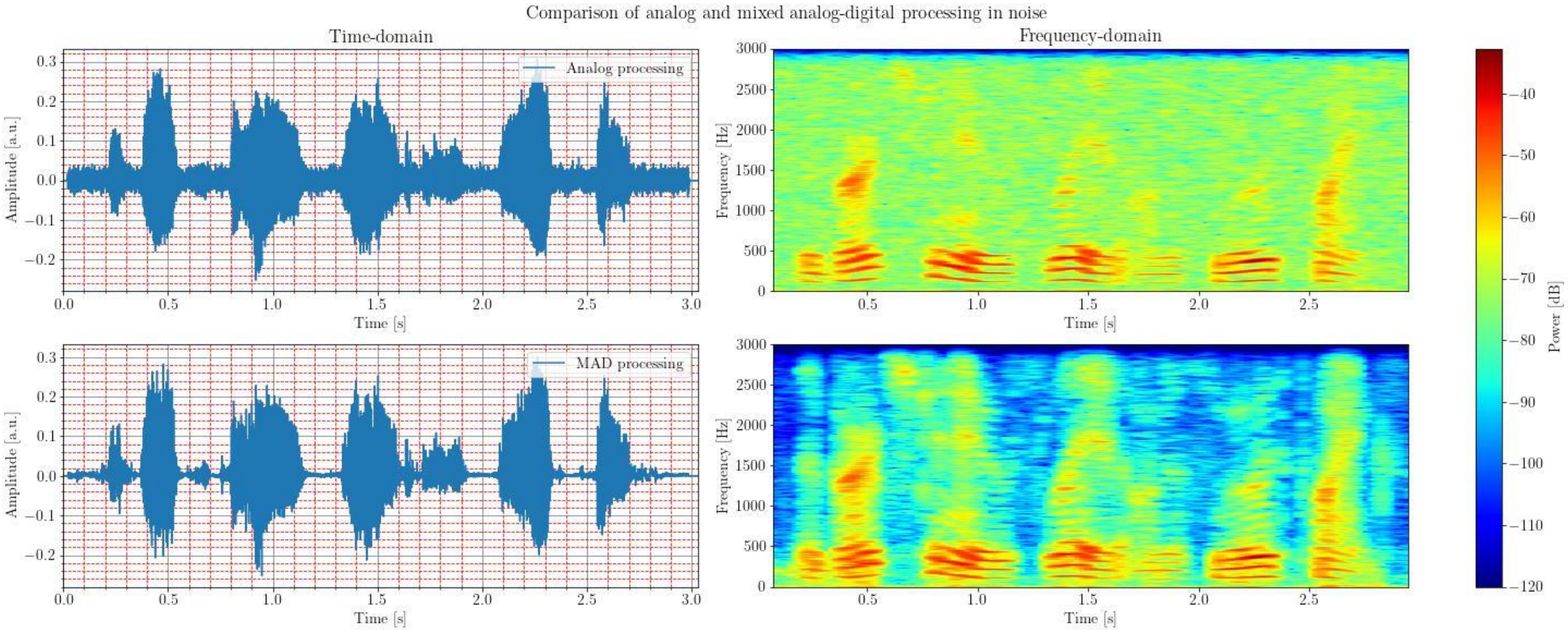
https://openai.com/blog/dall-e/



Types of Neural Networks

Example (Audio):

- Mixed analog/digital versus standard processing



Types of Neural Networks

Measurement setup:

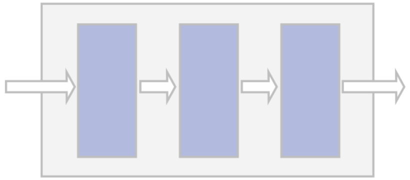
- Parameters:
  - *50 kHz base frequency*
  - *≈ 500 m distance*
  - *≈ 10-15 m water depth*
  - *Single Input, Single Output*
- Marinearsenal Kiel
- Submarine hangar to CASSy
- Mixed and traditional transmission

- Traditional approach
- New approach

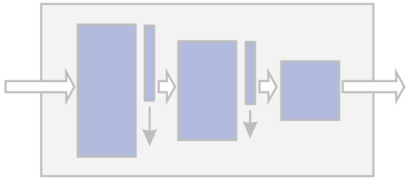
Types of Neural Networks

Network structure(s):

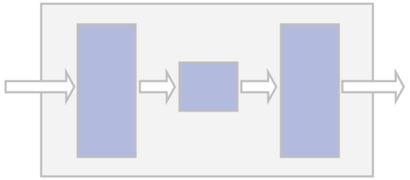
Multilayer perceptron



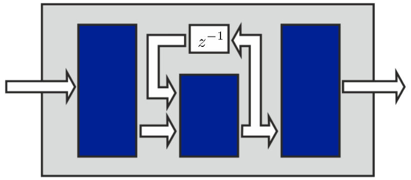
Convolutional neural network



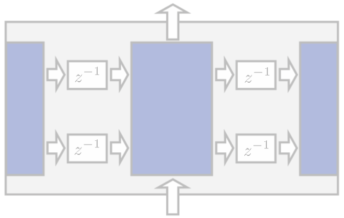
Autoencoder



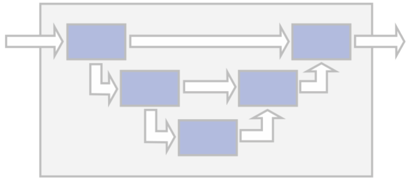
Recurrent neural network



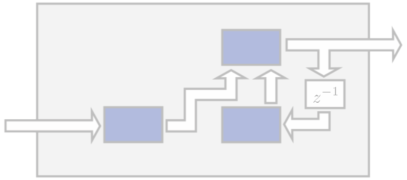
LSTM network



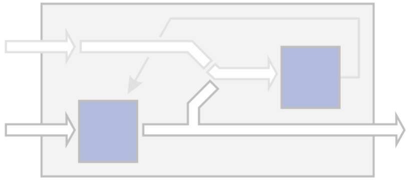
“U-net”



Attention-based network



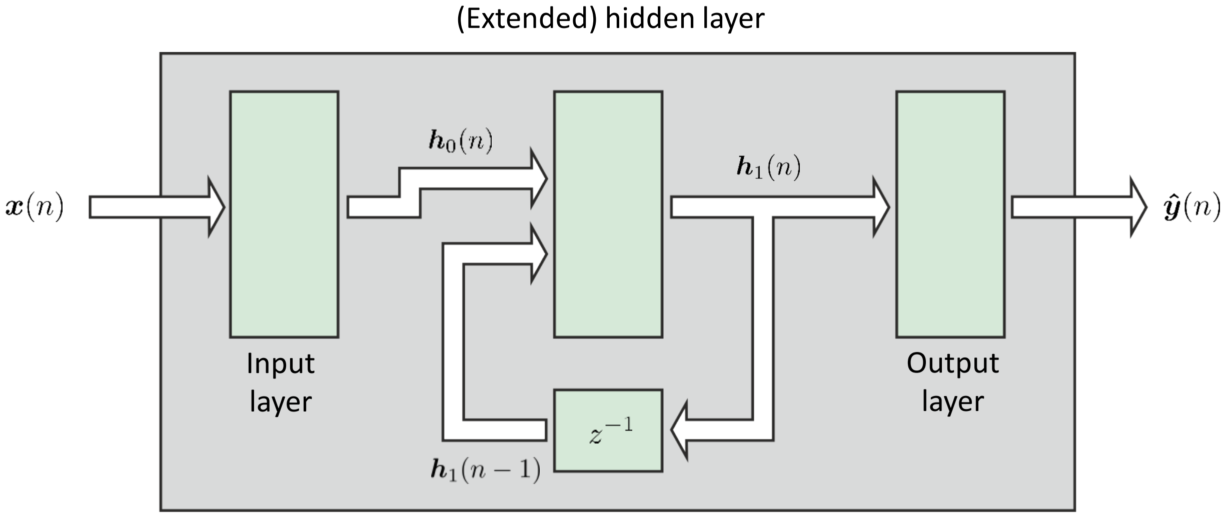
Generative adversarial network



Types of Neural Networks

Recurrent neural networks (RNNs):

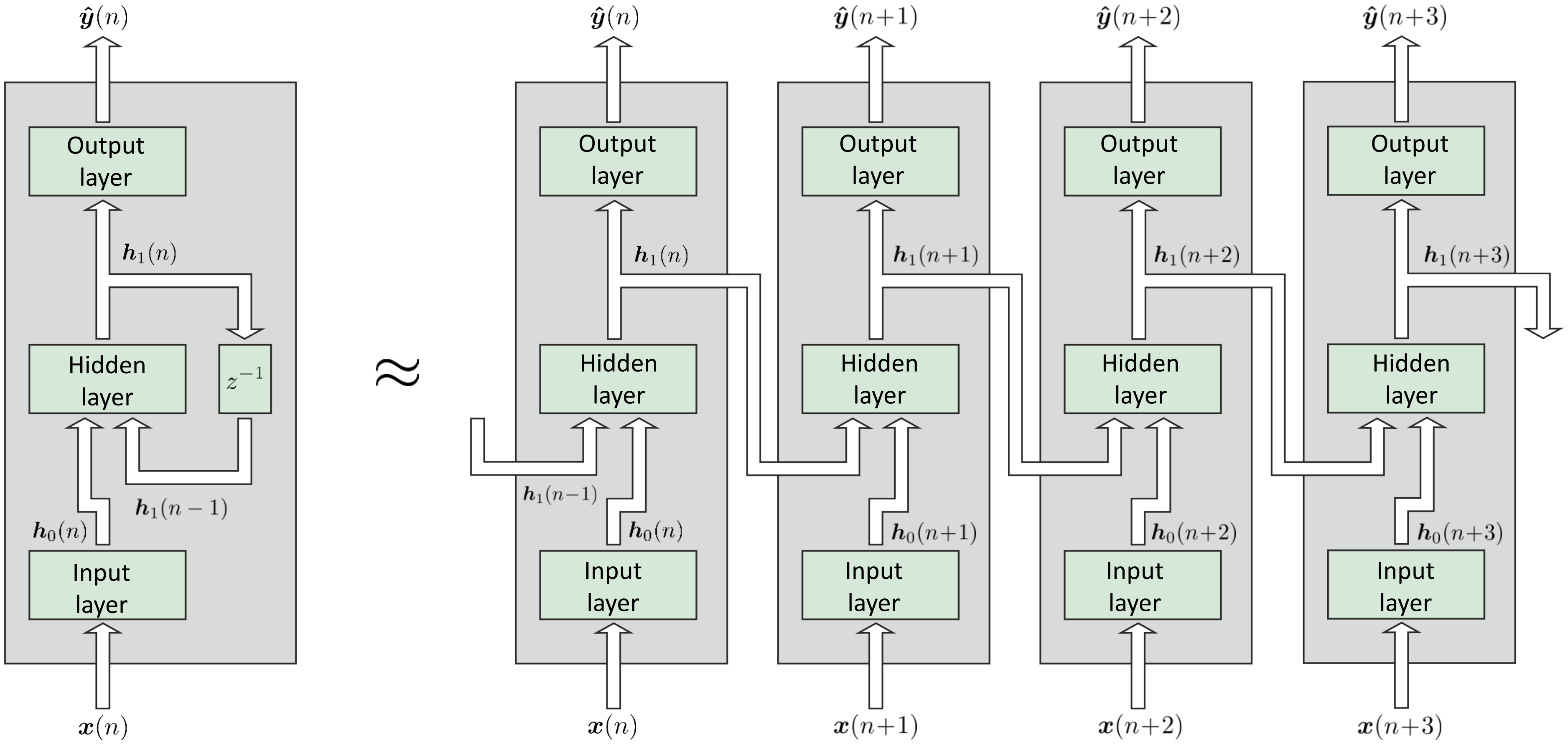
- ***Recursive branches*** are added to the network to allow for ***efficient modelling of temporal memory***.
- ***Stability*** (during operation) ***is not really an issue*** (in ***contrast to IIR filters***), since usually the activation functions include limitations.
- Very often the ***delay element is not depicted*** in literature of RNNs.



Types of Neural Networks

Recurrent neural networks (RNNs):

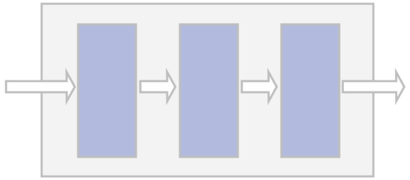
- ***Training*** could be done easily if the network is unfolded.
- Afterwards again a “standard” network with extended in- and outputs as well as with coefficient limitations can be trained.



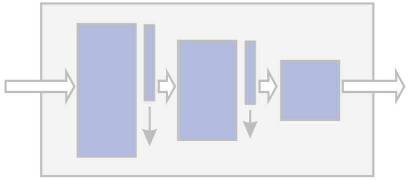
Types of Neural Networks

Network structure(s):

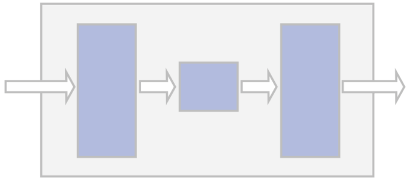
Multilayer perceptron



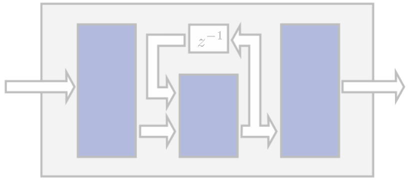
Convolutional neural network



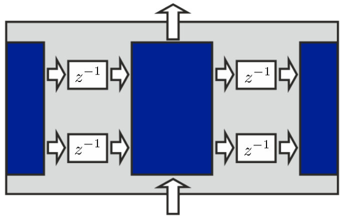
Autoencoder



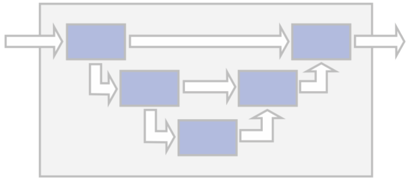
Recurrent neural network



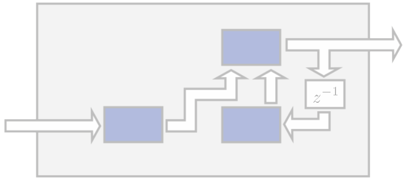
LSTM network



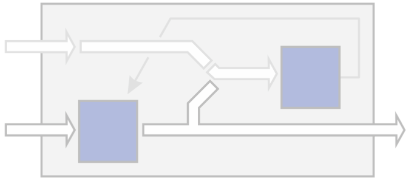
“U-net”



Attention-based network



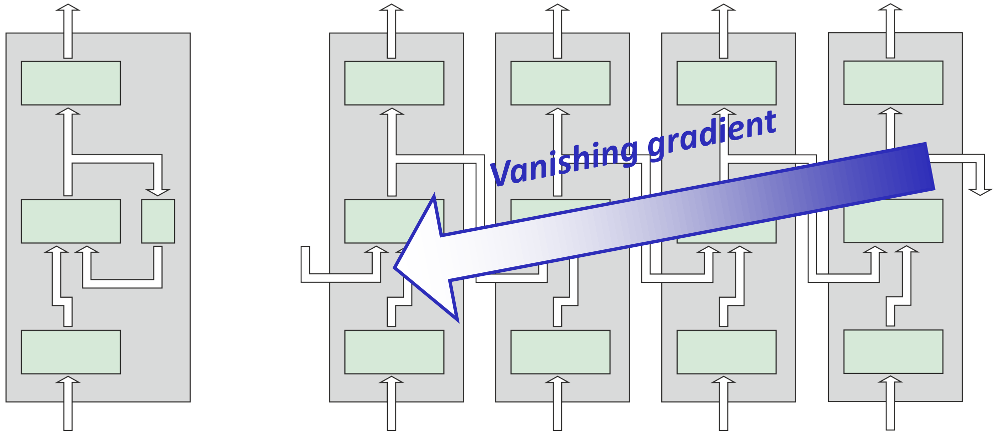
Generative adversarial network



Types of Neural Networks

Long-short-time memory networks (LSTMs):

- ***LSTMs*** are extensions of basic recurrent networks that don't suffer from the ***vanishing gradient problem***.

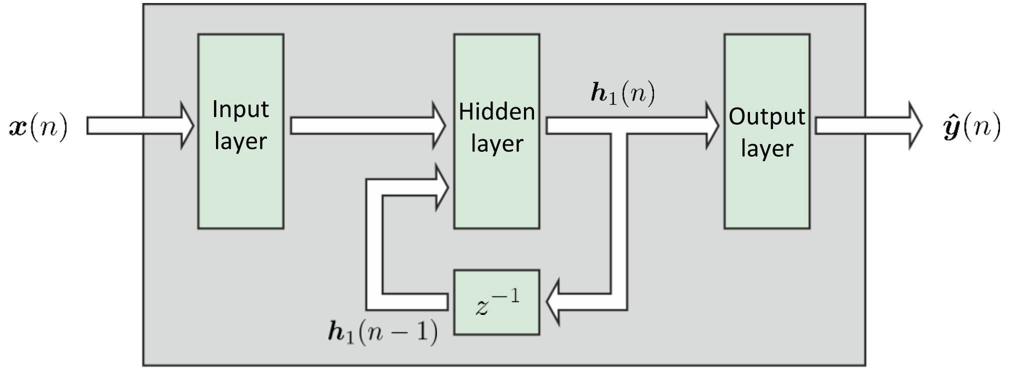


Types of Neural Networks

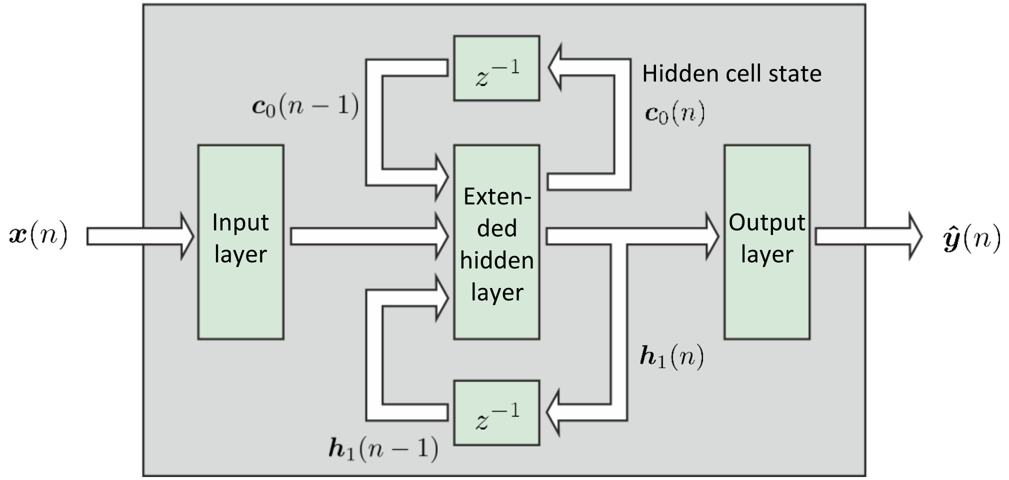
Long-short-time memory networks (LSTMs):

- ***LSTMs*** are extensions of basic recurrent networks that don't suffer from the ***vanishing gradient problem***.
- ***LSTMs*** are ***extended RNNs*** with an additional hidden ***cell state*** which serves as memory.
- Often used in ***classifying***, ***processing*** and ***making predictions*** based on ***time series data*** such as language translation.

Basic structure of a cell of a recurrent network



Basic structure of an LSTM cell

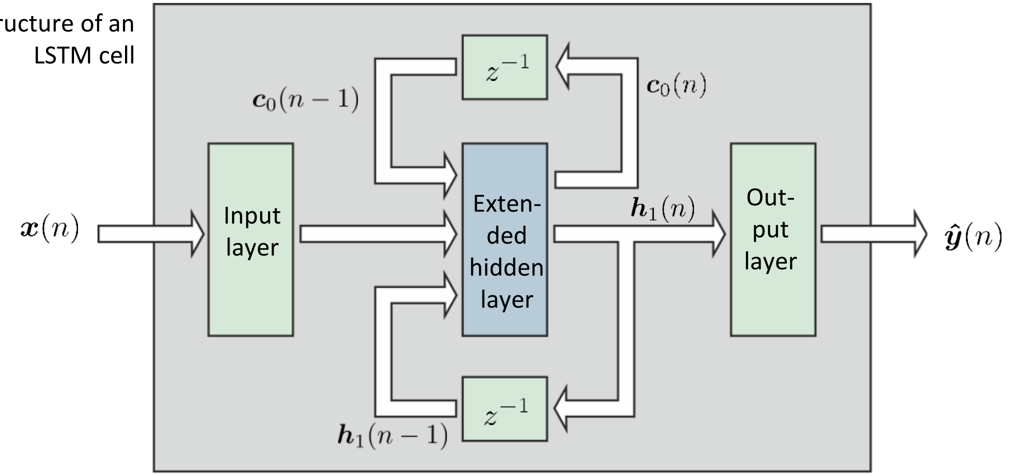


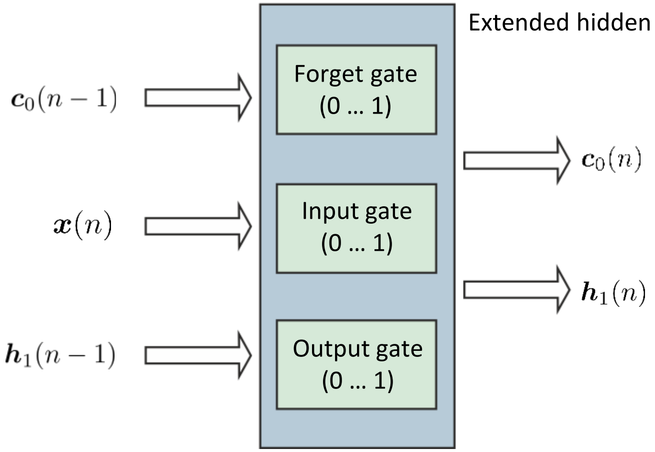
Types of Neural Networks

Long-short-time memory networks (LSTMs):

- ***LSTMs*** are extensions of basic recurrent networks that don't suffer from the ***vanishing gradient problem***.
- ***LSTMs*** are ***extended RNNs*** with an additional hidden ***cell state*** which serves as memory.
- Often used in ***classifying***, ***processing*** and ***making predictions*** based on ***time series data*** such as language translation.
- ***Three gates***:
  - ***Input*** gate
  - ***Forget*** gate
  - ***Output*** gate

Basic structure of an LSTM cell





Extended hidden layer

Types of Neural Networks

Long-short-time memory networks (LSTMs):

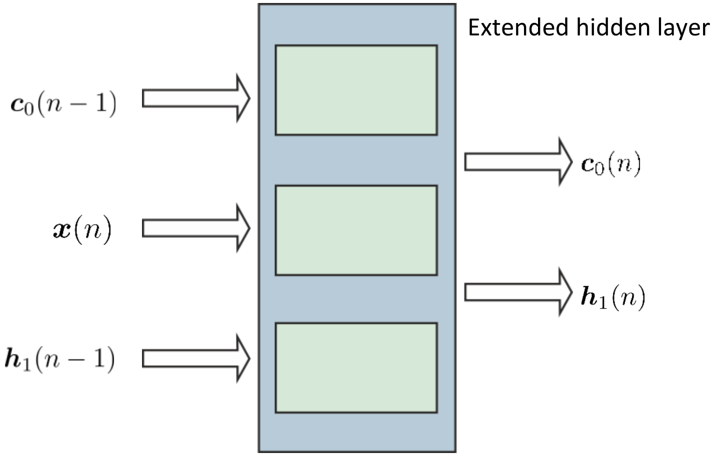
- ***LSTMs*** are extensions of basic recurrent networks that don’t suffer from the ***vanishing gradient problem***.
- ***LSTMs*** are ***extended RNNs*** with an additional hidden ***cell state*** which serves as memory.
- Often used in ***classifying***, ***processing*** and ***making predictions*** based on ***time series data*** such as language translation.
- ***Three gates***:
  - ***Input*** gate
  - ***Forget*** gate
  - ***Output*** gate

- ***Example*** from ***text processing***:

Gerhard is preparing lecture ***slides***.

Store “male” in a state

Store “plural” in another state



Types of Neural Networks

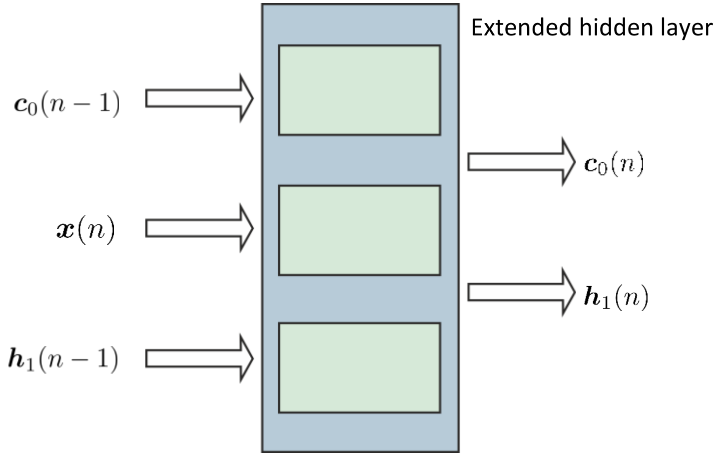
Long-short-time memory networks (LSTMs):

- ***LSTMs*** are extensions of basic recurrent networks that don't suffer from the ***vanishing gradient problem***.
- ***LSTMs*** are ***extended RNNs*** with an additional hidden ***cell state*** which serves as memory.
- Often used in ***classifying***, ***processing*** and ***making predictions*** based on ***time series data*** such as language translation.
- ***Three gates***:
    - ***Input*** gate
    - ***Forget*** gate
    - ***Output*** gate

- ***Example*** from ***text processing***:

Gerhard is preparing lecture slides. ***Jennifer*** is checking them.

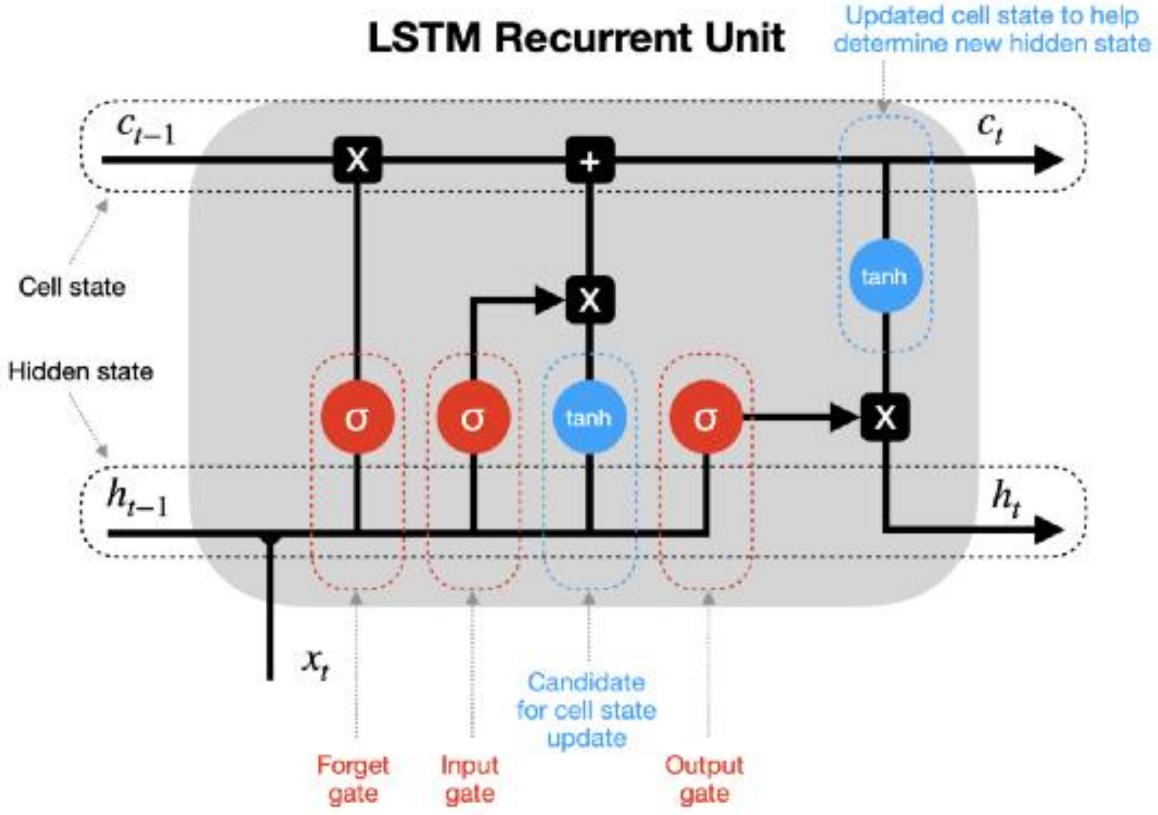
Forget "male" and store "female" in a state



Types of Neural Networks

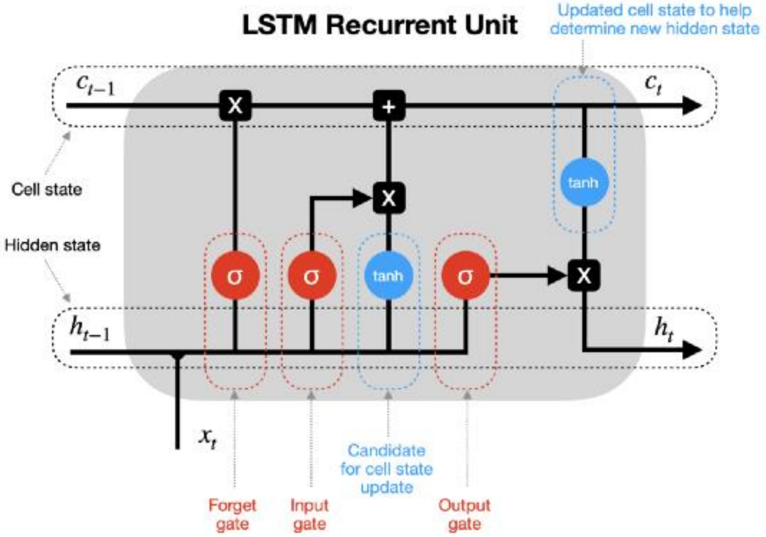
Long-short-time memory networks (LSTMs):

- ***LSTMs*** are extensions of basic recurrent networks that don't suffer from the ***vanishing gradient problem***.
- ***LSTMs*** are ***extended RNNs*** with an additional hidden ***cell state*** which serves as memory.
- Often used in ***classifying***, ***processing*** and ***making predictions*** based on ***time series data*** such as language translation.
- ***Three gates***:
  - ***Input*** gate
  - ***Forget*** gate
  - ***Output*** gate



Long-short-time memory networks (LSTMs):

- ***LSTMs*** are extensions of basic recurrent networks that don't suffer from the ***vanishing gradient problem***.
- ***LSTMs*** are ***extended RNNs*** with an additional hidden ***cell state*** which serves as memory.
- Often used in ***classifying***, ***processing*** and ***making predictions*** based on ***time series data*** such as language translation.
- ***Three gates***:
  - ***Input*** gate
  - ***Forget*** gate
  - ***Output*** gate



- ***Input*** gate:

$$\boldsymbol{i}_1(n) \;=\; \boldsymbol{\sigma}\Big(\boldsymbol{W}_{\mathrm{in},0}\left[\boldsymbol{x}^{\mathrm{T}}(n),\, \boldsymbol{h}_0^{\mathrm{T}}(n)\right]^{\mathrm{T}} + \boldsymbol{b}_{\mathrm{in},0}\Big)$$

- ***Forgot*** gate:

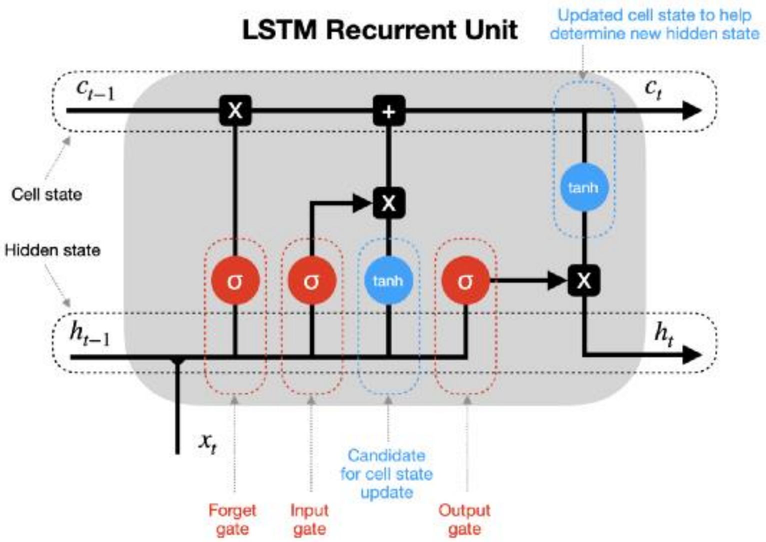
$$\boldsymbol{f}_1(n) \;=\; \boldsymbol{\sigma}\Big(\boldsymbol{W}_{\mathrm{in},0}\left[\boldsymbol{x}^{\mathrm{T}}(n),\, \boldsymbol{h}_0^{\mathrm{T}}(n)\right]^{\mathrm{T}} + \boldsymbol{b}_{\mathrm{for},1}\Big)$$

- ***Output*** gate:

$$\boldsymbol{o}_1(n) \;=\; \boldsymbol{\sigma}\Big(\boldsymbol{W}_{\mathrm{out},0}\left[\boldsymbol{x}^{\mathrm{T}}(n),\, \boldsymbol{h}_0^{\mathrm{T}}(n)\right]^{\mathrm{T}} + \boldsymbol{b}_{\mathrm{out},0}\Big)$$

Long-short-time memory networks (LSTMs):

- ***LSTMs*** are extensions of basic recurrent networks that don't suffer from the ***vanishing gradient problem***.
- ***LSTMs*** are ***extended RNNs*** with an additional hidden ***cell state*** which serves as memory.
- Often used in ***classifying***, ***processing*** and ***making predictions*** based on ***time series data*** such as language translation.
- ***Three gates***:
  - ***Input*** gate
  - ***Forget*** gate
  - ***Output*** gate



- ***Input*** gate:

$$\boldsymbol{i}_1(n) = \boldsymbol{\sigma}\Big(\boldsymbol{W}_{\text{in},0}\,\big[\boldsymbol{x}^{\text{T}}(n),\,\boldsymbol{h}_0^{\text{T}}(n)\big]^{\text{T}} + \boldsymbol{b}_{\text{in},0}\Big)$$

- ***Forgot*** gate:

$$\boldsymbol{f}_1(n) = \boldsymbol{\sigma}\Big(\boldsymbol{W}_{\text{in},0}\,\big[\boldsymbol{x}^{\text{T}}(n),\,\boldsymbol{h}_0^{\text{T}}(n)\big]^{\text{T}} + \boldsymbol{b}_{\text{for},1}\Big)$$

- ***Output*** gate:

$$\boldsymbol{o}_1(n) = \boldsymbol{\sigma}\Big(\boldsymbol{W}_{\text{out},0}\,\big[\boldsymbol{x}^{\text{T}}(n),\,\boldsymbol{h}_0^{\text{T}}(n)\big]^{\text{T}} + \boldsymbol{b}_{\text{out},0}\Big)$$

- ***Cell state*** update:

$$\bar{\boldsymbol{c}}_1(n) = \mathbf{tanh}\Big(\boldsymbol{W}_{\text{c},0}\,\big[\boldsymbol{x}^{\text{T}}(n),\,\boldsymbol{h}_0^{\text{T}}(n)\big]^{\text{T}} + \boldsymbol{b}_{\text{c},0}\Big)$$

$$\boldsymbol{c}_1(n) = \mathbf{diag}\{\boldsymbol{f}_1(n)\}\,\boldsymbol{c}_1(n-1) + \mathbf{diag}\{\boldsymbol{i}_1(n)\}\,\bar{\boldsymbol{c}}_1(n)$$

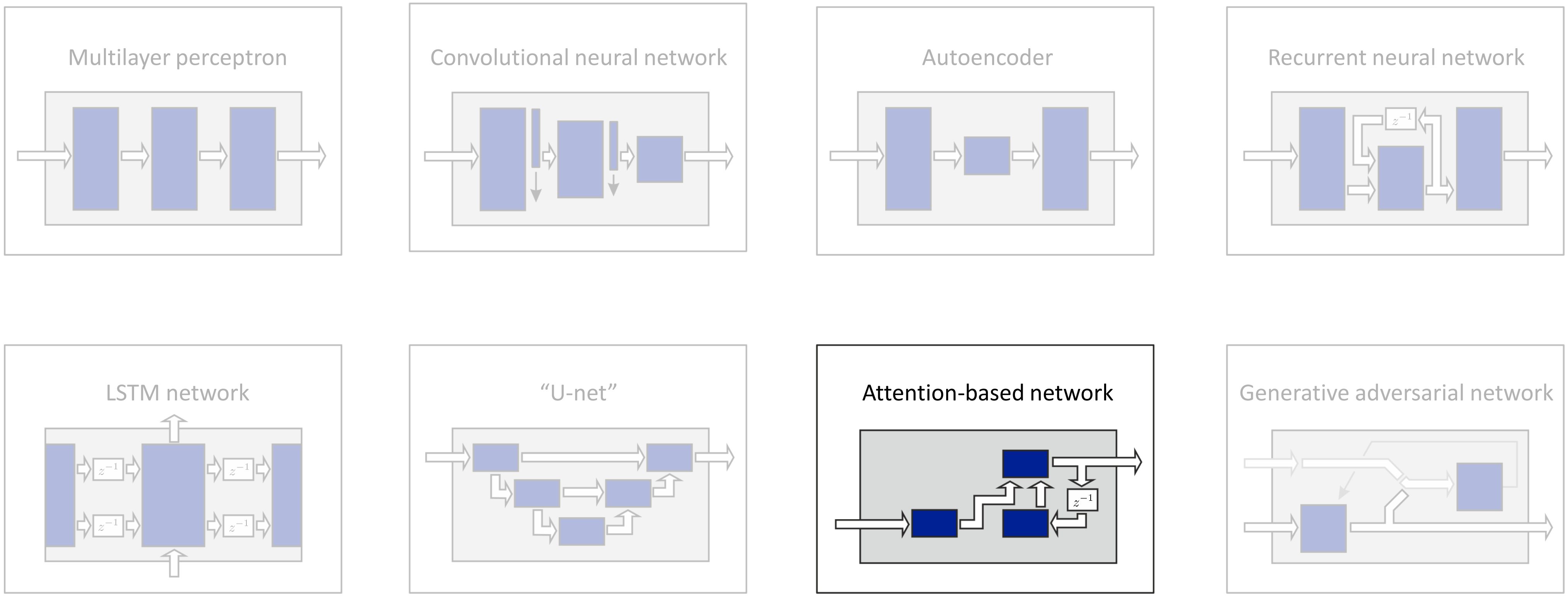
- ***Hidden state*** update:

$$\boldsymbol{h}_1(n) = \mathbf{diag}\Big\{\mathbf{tanh}\{\boldsymbol{c}_1(n)\}\Big\}\,\boldsymbol{o}_1(n)$$

C | A | U
Christian-Albrechts-Universität zu Kiel

Types of Neural Networks

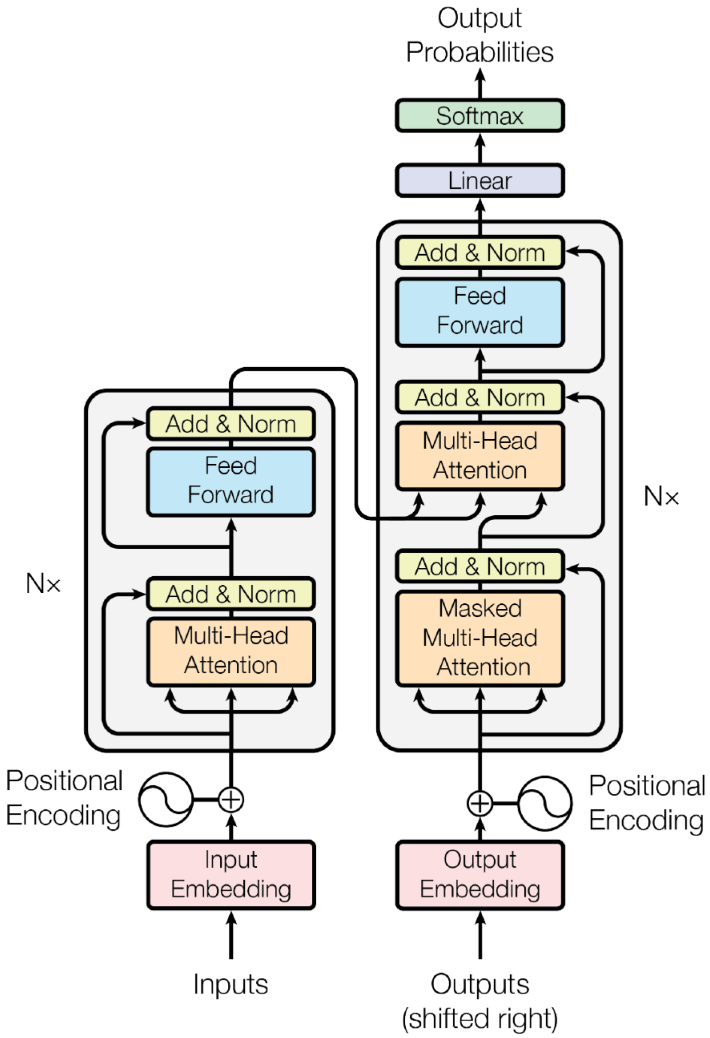
Network structure(s):



Types of Neural Networks

Attention-based networks:

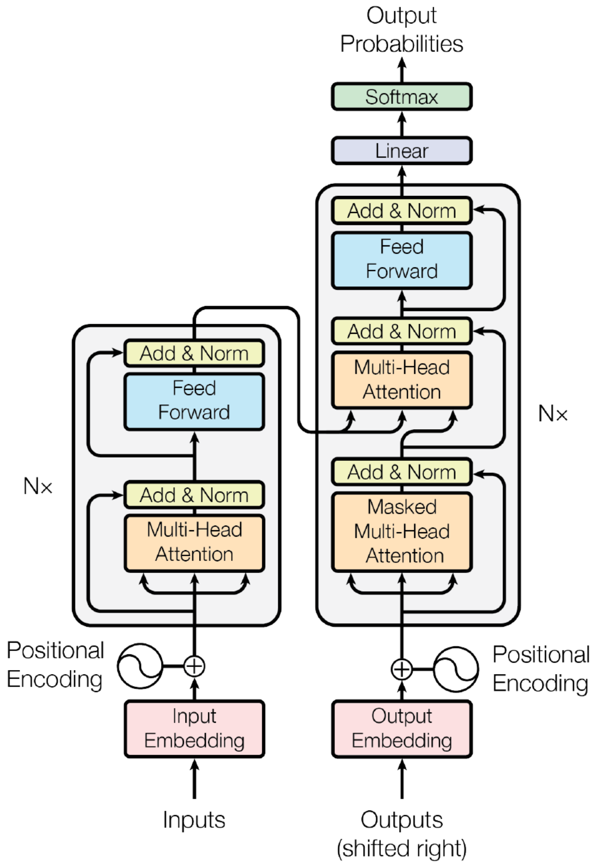
- So-called ***transformers*** in combination with ***attention-based preprocessing*** is often used for the translation of texts (input in one language, output in another).
- "Attention" was ***invented*** by Vaswani, Ashish, Shazeer, et al. in 2017 (see graphic on the left)
- Consists of a ***encoder*** and a ***decoder*** part.
- We will not go into all details of transformers (see hint at the end of this slide section), but since ***attention*** can be used in ***several other applications***, we will go a bit into detail here.

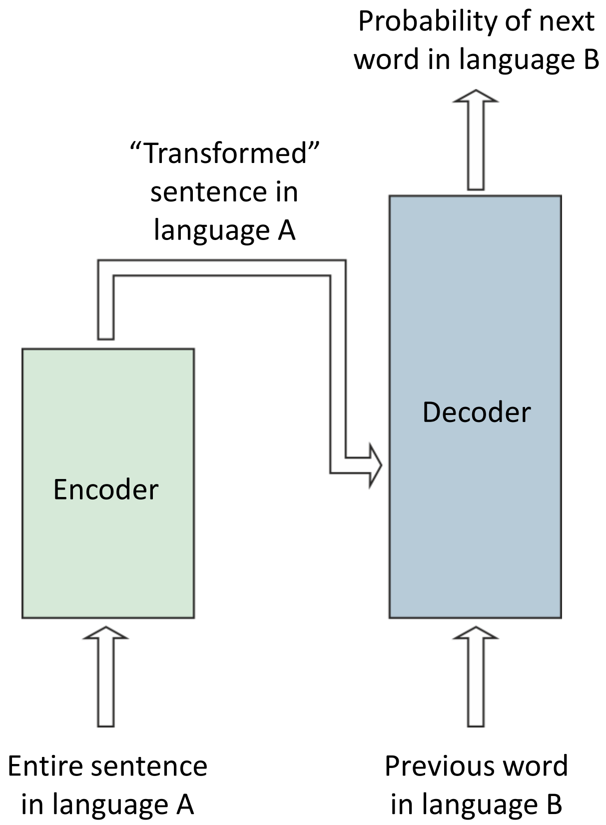


Types of Neural Networks

Attention-based networks:

- Simplification to understand the basic principle

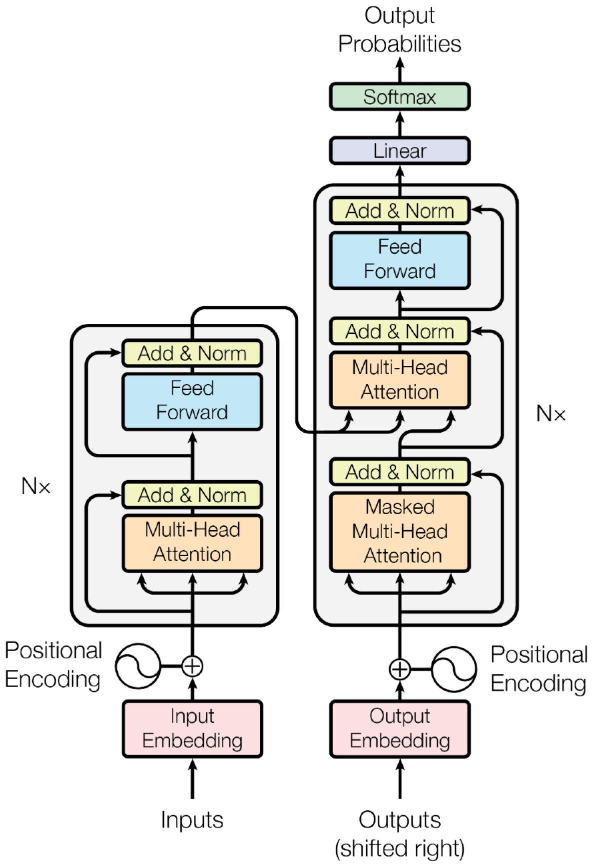


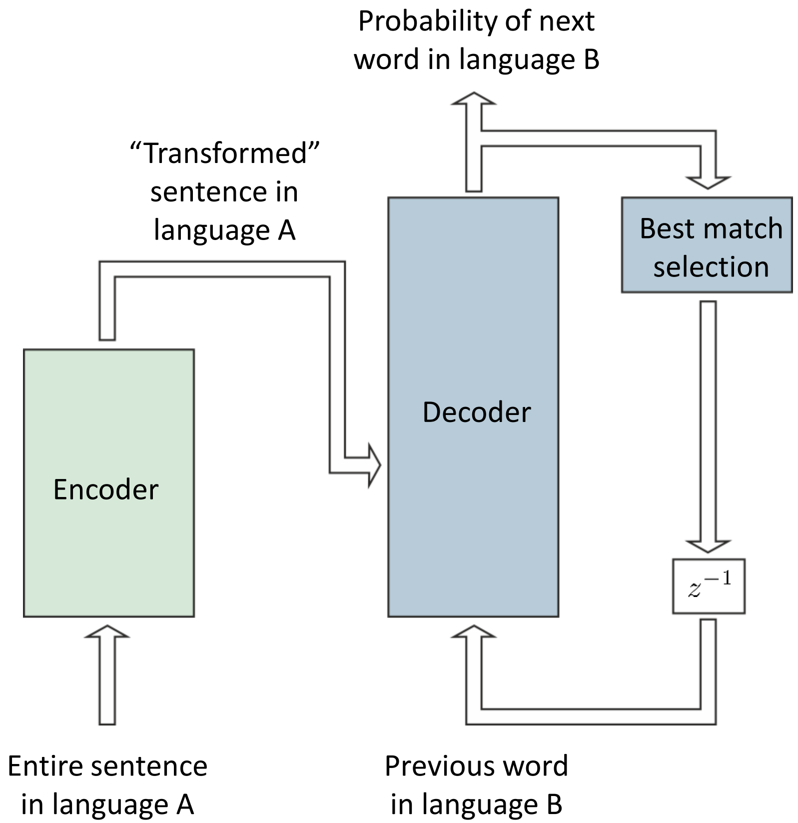


Types of Neural Networks

Attention-based networks:

- Simplification to understand the basic principle
- Recurrent principle of the decoder

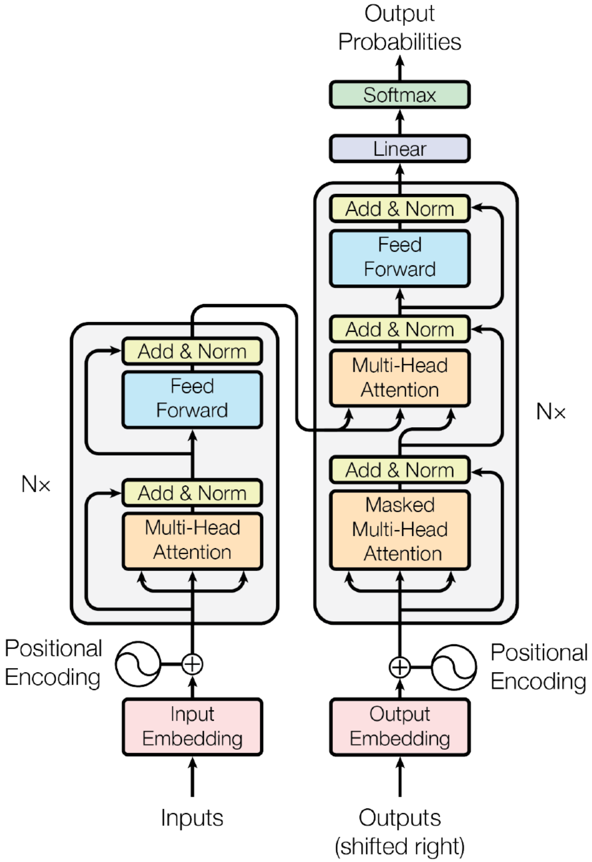


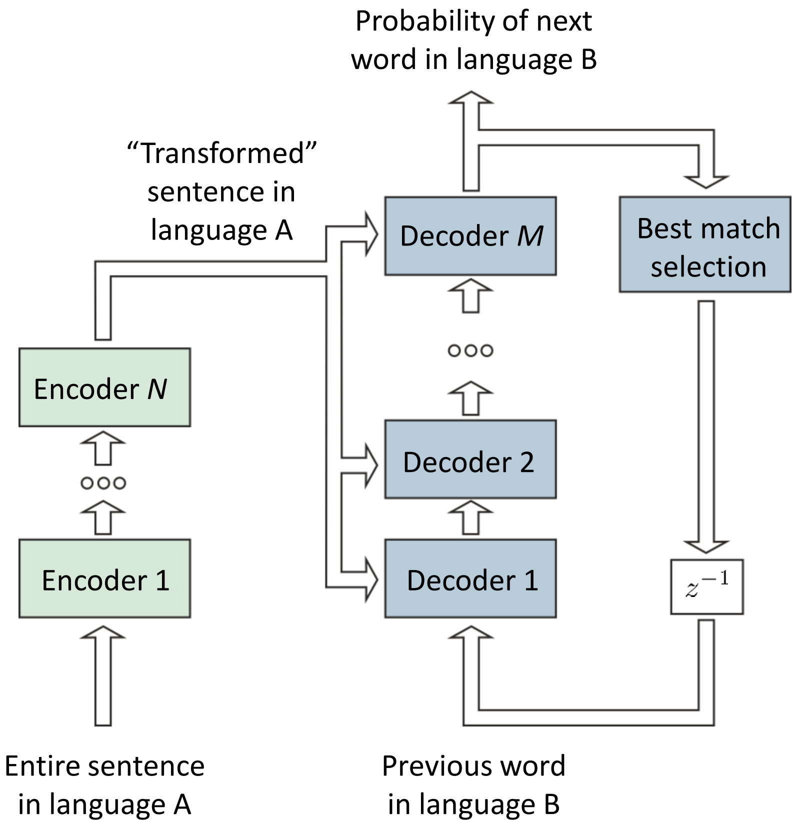


Types of Neural Networks

Attention-based networks:

- Simplification to understand the basic principle
- Recurrent principle of the decoder
- Multiple encoder and decoder stages are connected.
- Input and output vectors have the same size and the same "definition".

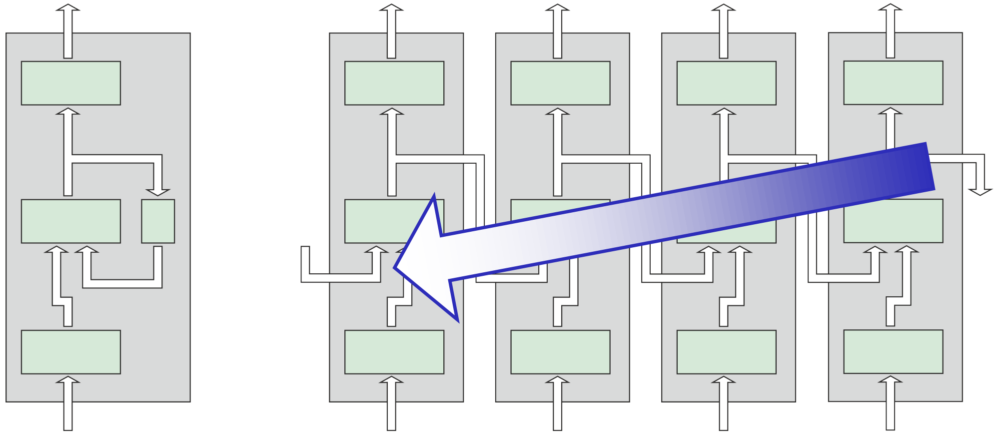




Types of Neural Networks

Attention-based networks:

- The ***problem*** with ***recurrent networks*** that are trained with large “defolding” are ***vanishing*** (and/or exploiting) ***gradients***.
- However, in translation a ***large context*** is required.
- ***Example***:
  - Have a look on the ***context*** of “it”.
  - ***Predict*** the ***next word***.

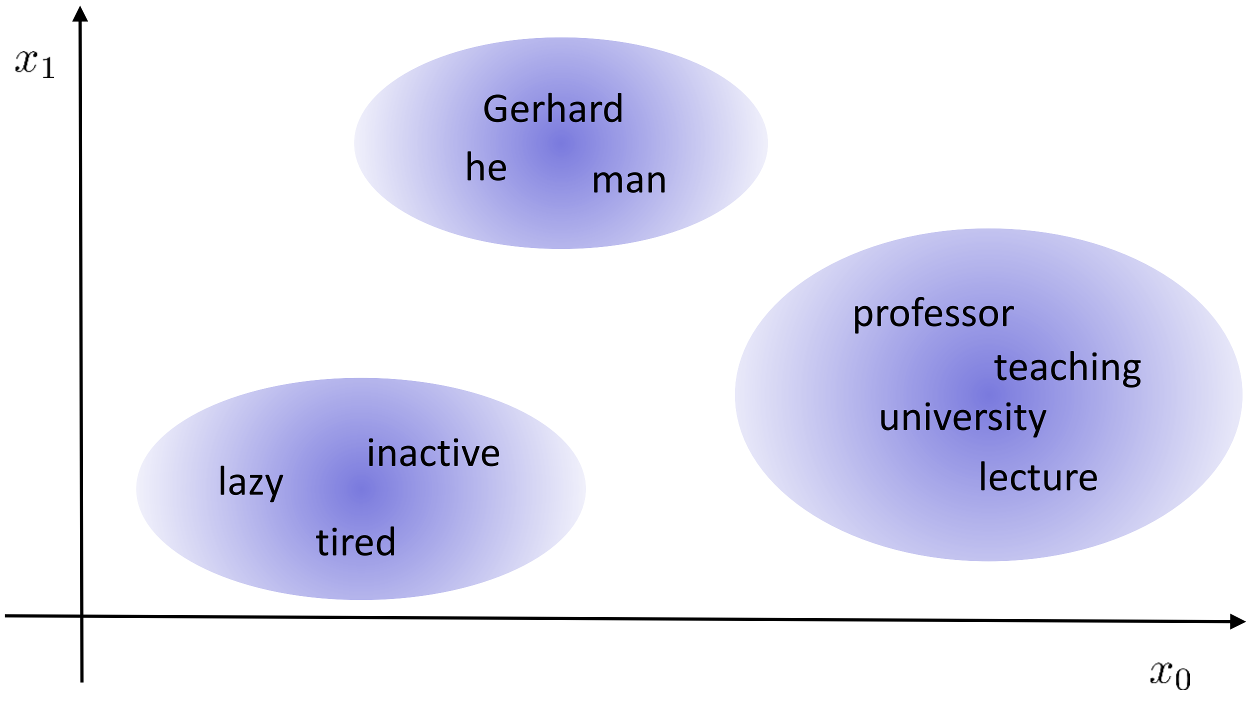


Gerhard ordered a new ***notebook***. When ***it*** arrived at home, his daughter thought ***it*** was for her and was very happy. However, ***it*** was not working as expected and Gerhard had to send ***it*** ...

Types of Neural Networks

Attention-based networks:

- Basic principle of word embedding
  - Words are converted in to a high dimensional vector space.
  - Spatial closeness indicates a (strong) “relationship”.



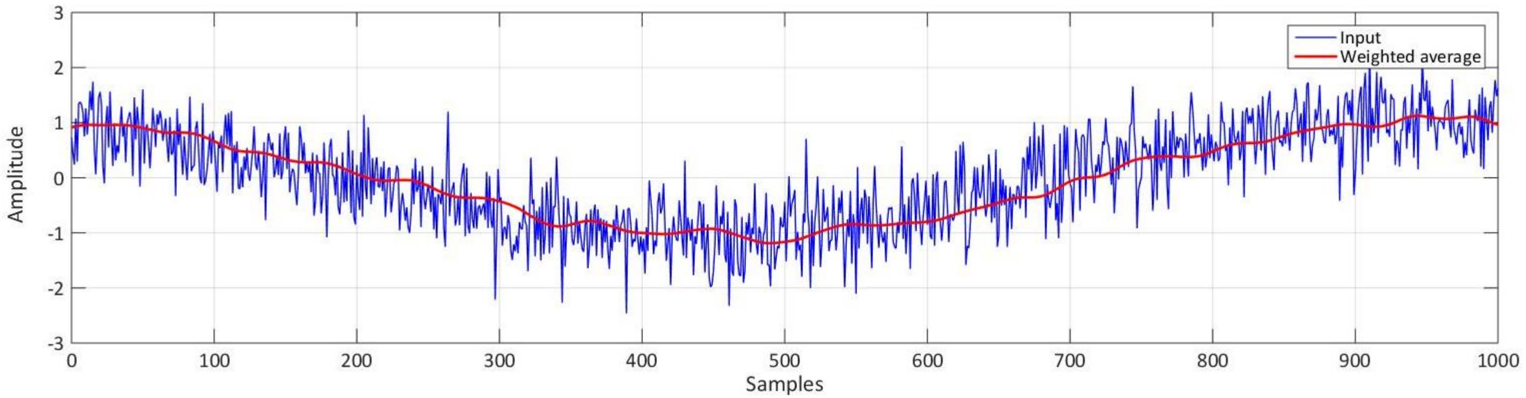
Types of Neural Networks

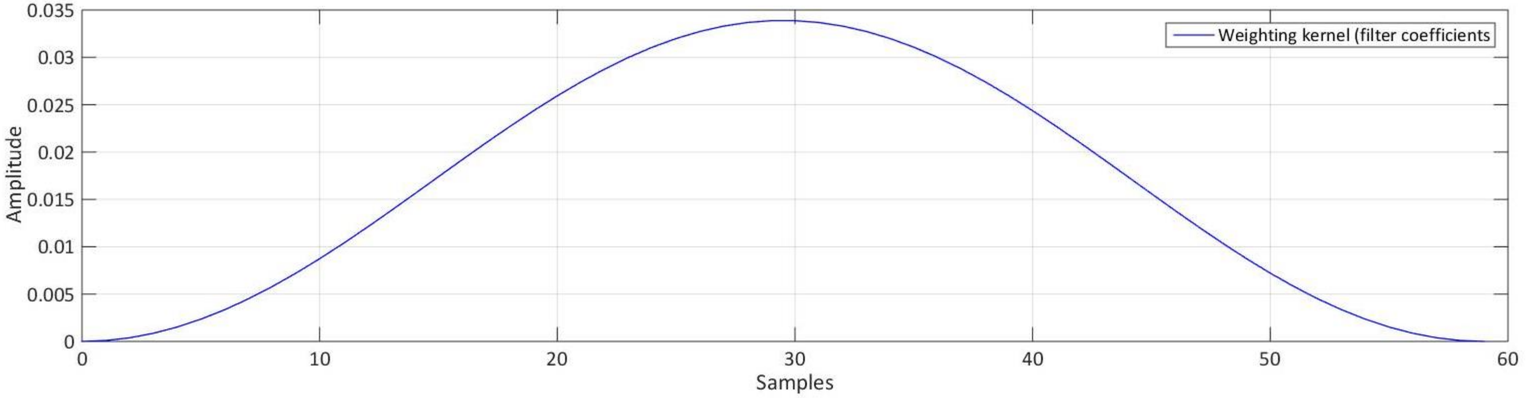
Attention-based networks:

- Basic principle of weighted averaging:

$$
\begin{aligned}
y(n) &= \sum_{m=0}^{N-1} \tilde{w}(m)\, v(n-m) \\
&= \sum_{m=0}^{N-1} w(m,n)\, v(m)
\end{aligned}
$$

  - Here spatial / temporal closeness is mapped on weights.
  - The kernel is a Hann window.
  - Importance or SNR of the input samples is not taken into account.
  - Also, “relationships” among the input samples are not exploited.



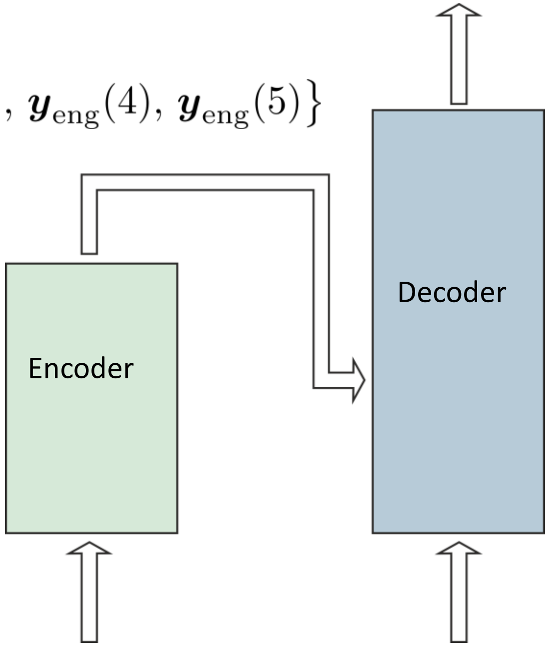


Types of Neural Networks

Attention-based networks:

- Basic principle of transformers (simplified)
  - Text translation
    - First input is converted and encoded.

$$\{\boldsymbol{y}_{\mathrm{eng}}(0),\, \boldsymbol{y}_{\mathrm{eng}}(1),\, \boldsymbol{y}_{\mathrm{eng}}(2),\, \boldsymbol{y}_{\mathrm{eng}}(3),\, \boldsymbol{y}_{\mathrm{eng}}(4),\, \boldsymbol{y}_{\mathrm{eng}}(5)\}$$



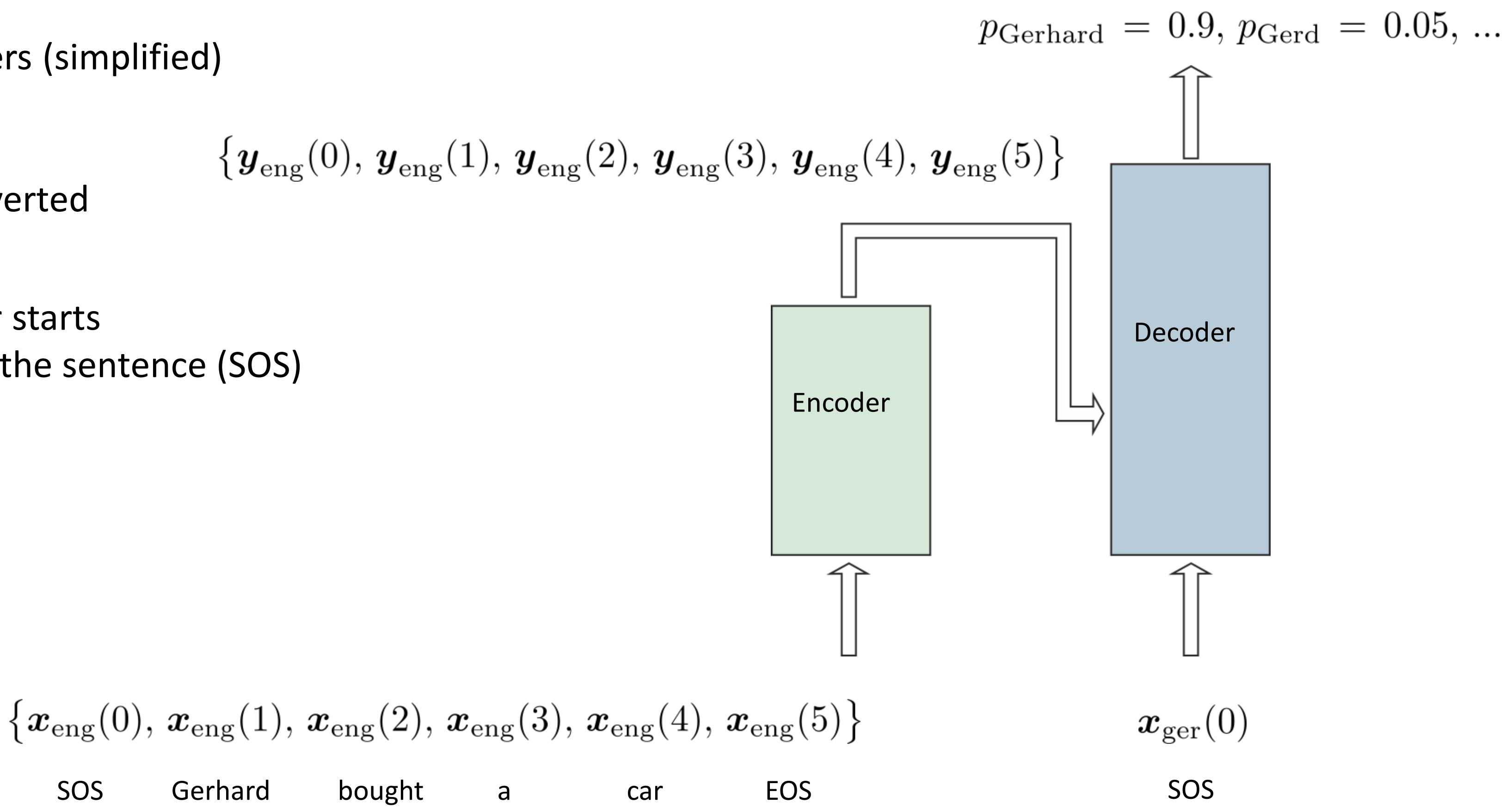
$$\{\boldsymbol{x}_{\mathrm{eng}}(0),\, \boldsymbol{x}_{\mathrm{eng}}(1),\, \boldsymbol{x}_{\mathrm{eng}}(2),\, \boldsymbol{x}_{\mathrm{eng}}(3),\, \boldsymbol{x}_{\mathrm{eng}}(4),\, \boldsymbol{x}_{\mathrm{eng}}(5)\}$$

SOS Gerhard bought a car EOS

Types of Neural Networks

Attention-based networks:

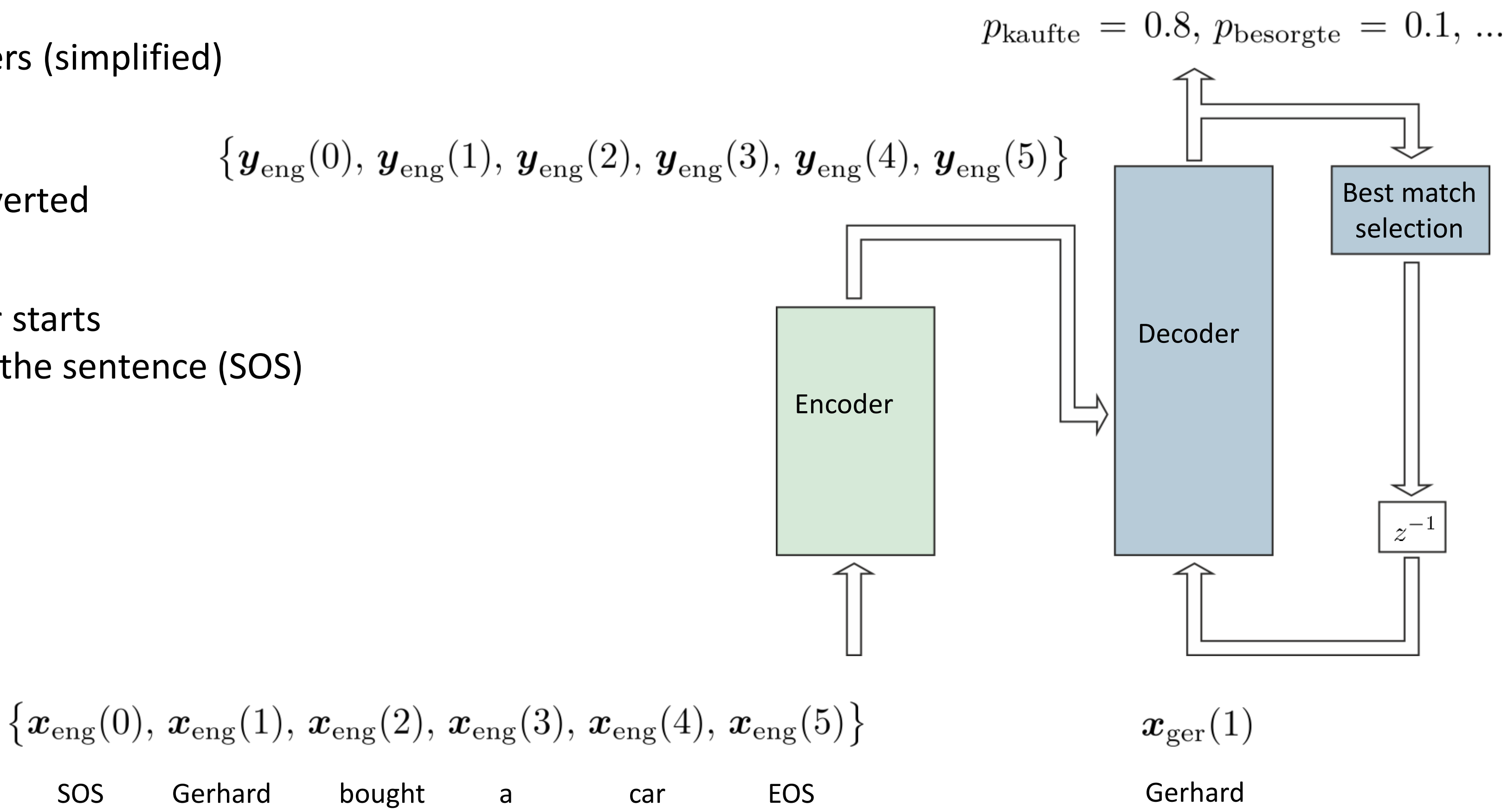
- Basic principle of transformers (simplified)
  - Text translation
    - First input is converted and encoded.
    - Next the decoder starts with the start of the sentence (SOS)



Types of Neural Networks

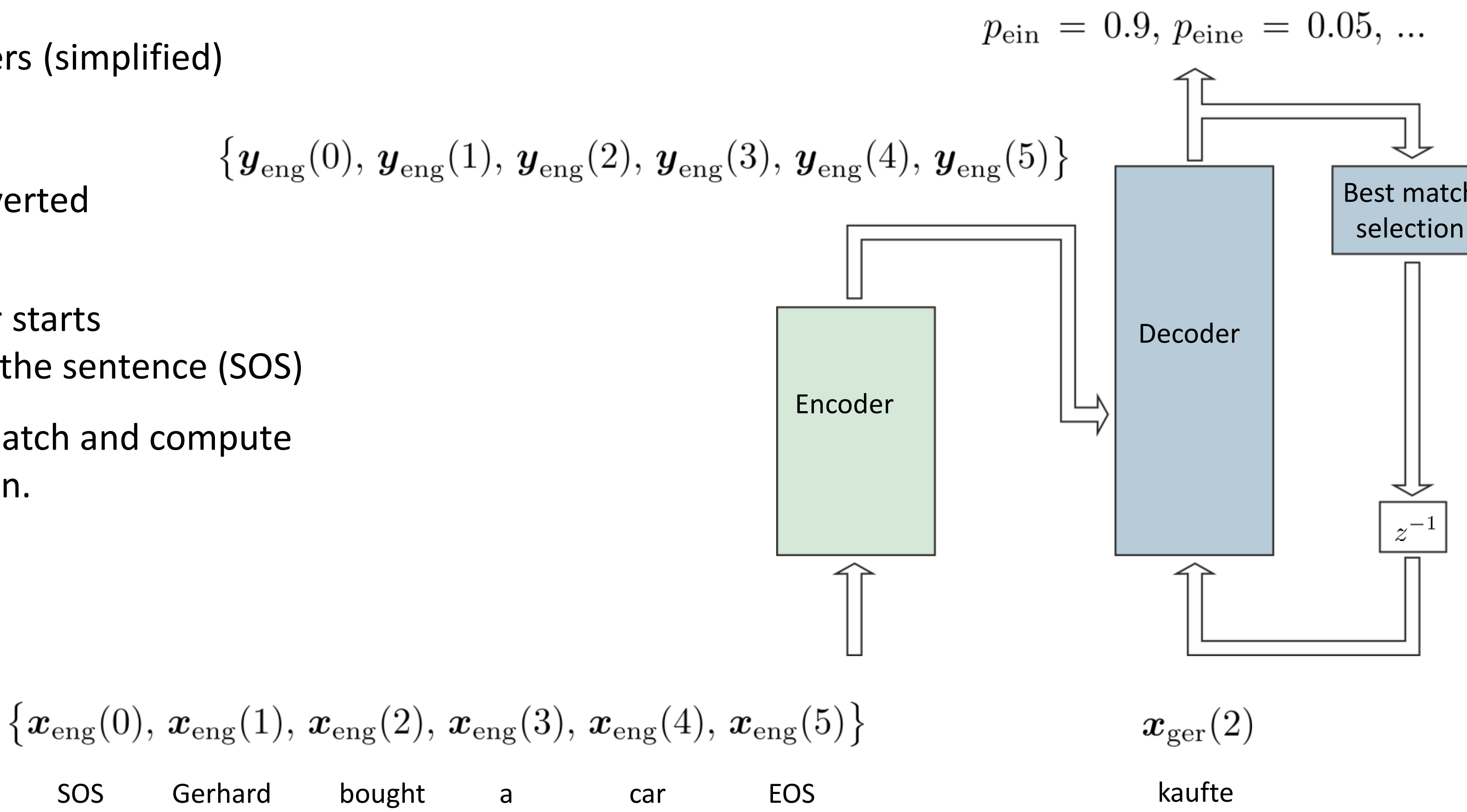
Attention-based networks:

- Basic principle of transformers (simplified)
  - Text translation
    - First input is converted and encoded.
    - Next the decoder starts with the start of the sentence (SOS)



Attention-based networks:

- Basic principle of transformers (simplified)
  - Text translation
    - First input is converted and encoded.
    - Next the decoder starts with the start of the sentence (SOS)
    - Select the best match and compute the decoder again.

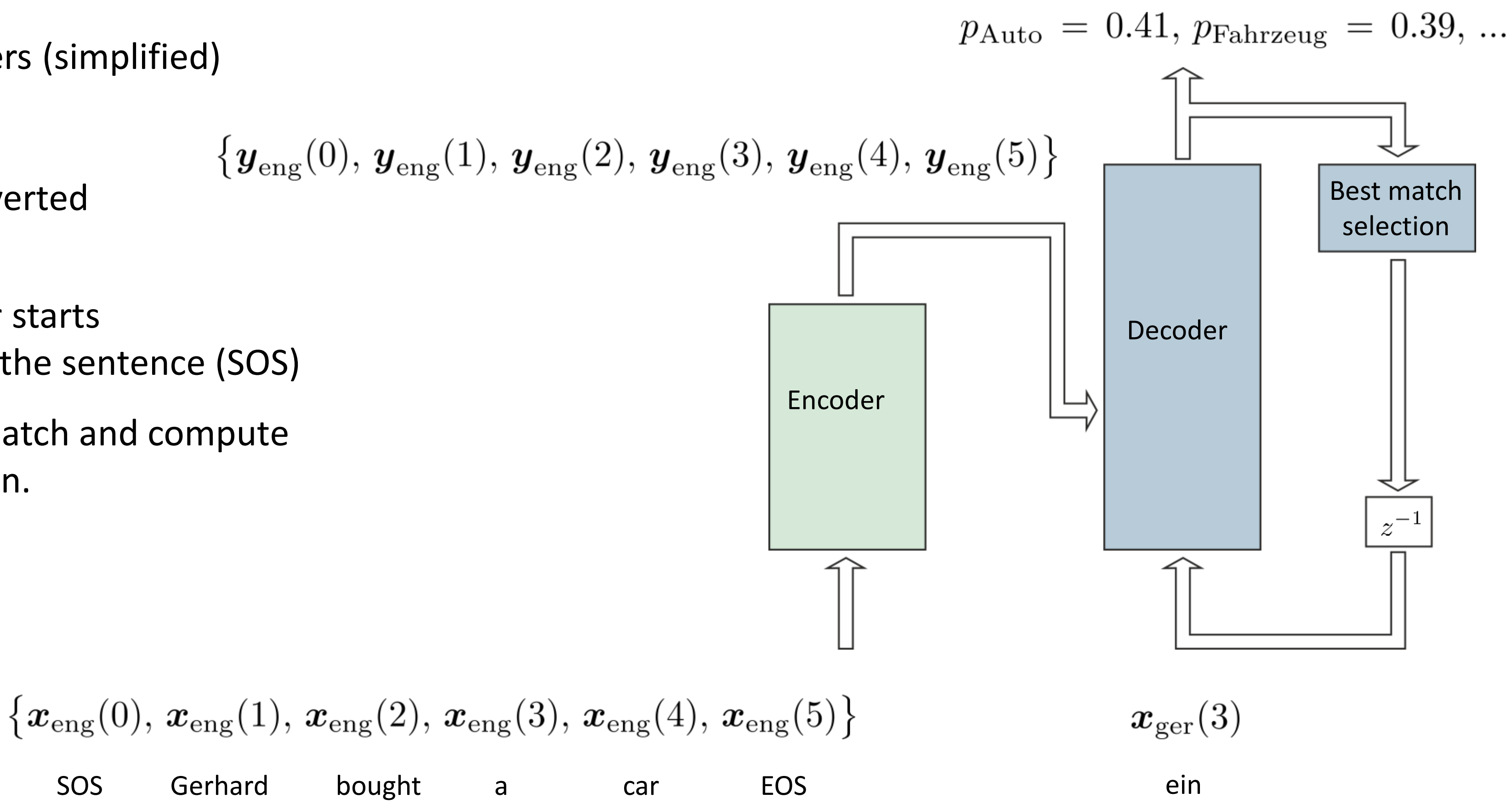


C | A | U
Christian-Albrechts-Universität zu Kiel

Types of Neural Networks

Attention-based networks:

- Basic principle of transformers (simplified)
  - Text translation
    - First input is converted and encoded.
    - Next the decoder starts with the start of the sentence (SOS)
    - Select the best match and compute the decoder again.
    - And so on …



Attention-based networks:

- Basic principle of attention
  - Input words: SOS Gerhard is lazy but he likes to be a professor EOF
  - Input vectors: $\boldsymbol{x}(0)\quad \boldsymbol{x}(1)\quad \boldsymbol{x}(2)\quad \boldsymbol{x}(3)\quad \boldsymbol{x}(4)\quad \boldsymbol{x}(5)\quad \boldsymbol{x}(6)\quad \boldsymbol{x}(7)\quad \boldsymbol{x}(8)\quad \boldsymbol{x}(9)\quad \boldsymbol{x}(10)\quad \boldsymbol{x}(N-1)$
  - Queries: $\boldsymbol{q}(n) = \boldsymbol{W}_{\mathrm{q}}\,\boldsymbol{x}(n)$
  - Keys: $\boldsymbol{k}(n) = \boldsymbol{W}_{\mathrm{k}}\,\boldsymbol{x}(n)$
  - Preliminary weights: $w_{\mathrm{pre}}(n,m) = \boldsymbol{k}^{\mathrm{T}}(n)\,\boldsymbol{q}(m)$
  - Final weights: $\{w(0,m),\,...,\,w(N-1,m)\} = \mathrm{softmax}\{w_{\mathrm{pre}}(0,m),\,...,\,w_{\mathrm{pre}}(N-1,m)\}$
  - New embedding: $\boldsymbol{y}(n) = \sum_{m=0}^{N-1} w(m,n)\,\boldsymbol{x}(m)$

Attention-based networks:

- Basic principle of (self) attention heads
  - Input vectors: $$\boldsymbol{x}(n) = \big[x_0(n),\, x_1(n),\, ...,\, x_{D-1}(n)\big]^{\mathrm{T}}$$
  - Queries: $$\boldsymbol{q}(n) = \boldsymbol{W}_{\mathrm{q}}\, \boldsymbol{x}(n)$$
  - Keys: $$\boldsymbol{k}(n) = \boldsymbol{W}_{\mathrm{k}}\, \boldsymbol{x}(n)$$
  - ***Values***: $$\boldsymbol{v}(n) = \boldsymbol{W}_{\mathrm{v}}\, \boldsymbol{x}(n)$$

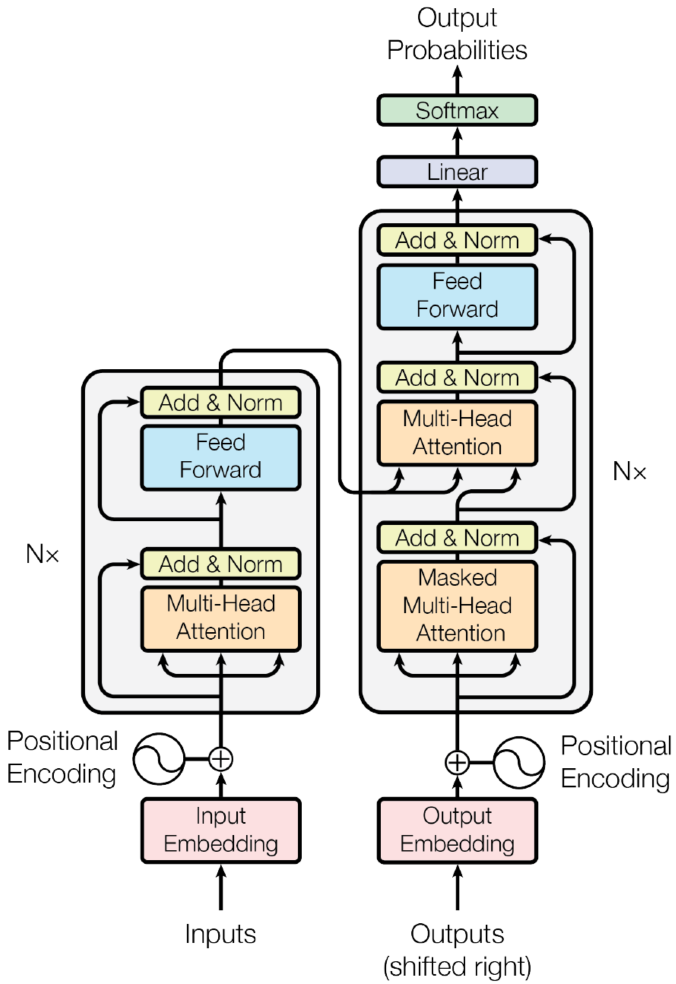
These three matrices will be optimized during the training.

  - ***Preliminary weights***: $$w_{\mathrm{pre}}(n,m) = \frac{\boldsymbol{k}^{\mathrm{T}}(n)\,\boldsymbol{q}(m)}{\sqrt{D}}$$
  - Final weights: $$\big\{w(0,m),\, ...,\, w(N-1,m)\big\} = \mathrm{softmax}\big\{w_{\mathrm{pre}}(0,m),\, ...,\, w_{\mathrm{pre}}(N-1,m)\big\}$$
  - ***New embedding***: $$\boldsymbol{y}(n) = \sum_{m=0}^{N-1} w(m,n)\,\boldsymbol{v}(m)$$

Types of Neural Networks

Attention-based networks:

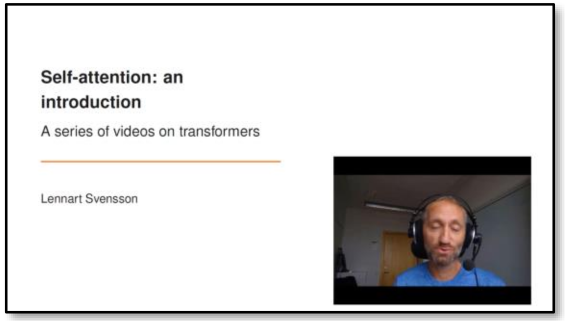
- Full structure
  - Beside “self attention” also “masked attention” is used in the decoder.
  - Each attention block is followed by an adder and a normalization unit (mean subtraction and division by standard deviation).
  - Afterwards a simple feed forward network is computed.

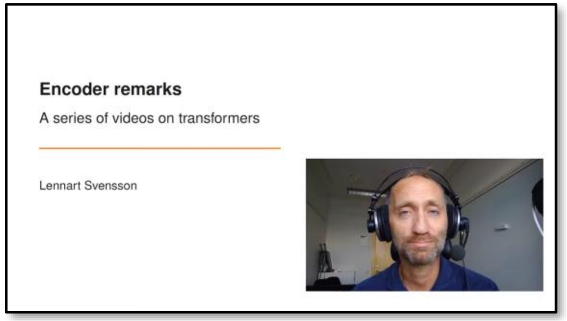


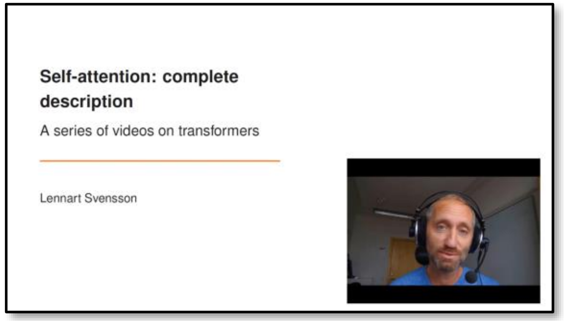
Types of Neural Networks

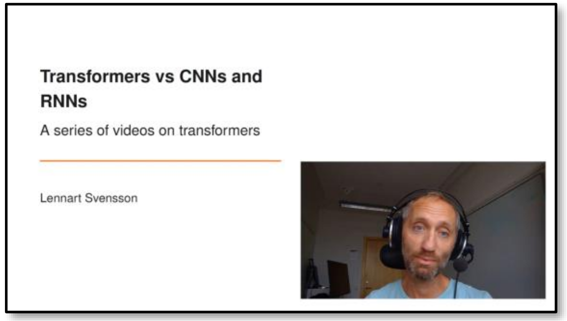
Attention-based networks:

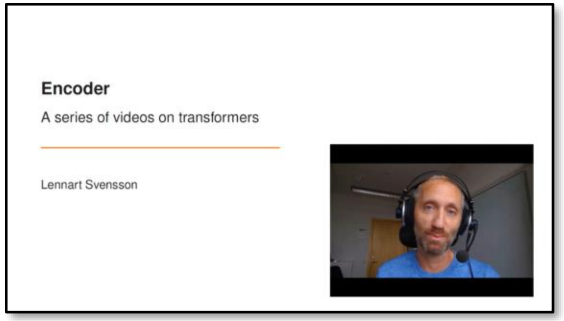
- Very ***good explanation*** from ***Lennart Svensson***, Chalmers University of Technology, Göteborg, Sweden
- ***YouTube*** videos
  - [***https://www.youtube.com/watch?v=0SmNEp4zTpc***](https://www.youtube.com/watch?v=0SmNEp4zTpc)
  - [***https://www.youtube.com/watch?v=ER_KqqtoikA***](https://www.youtube.com/watch?v=ER_KqqtoikA)
  - (see playlist for further seven videos)

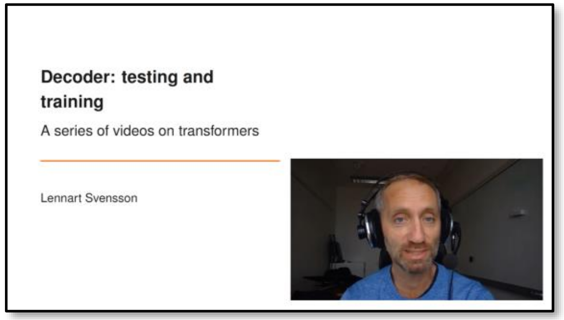








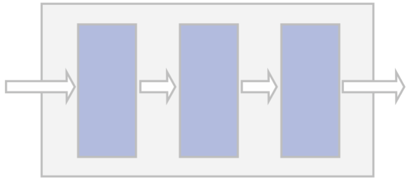




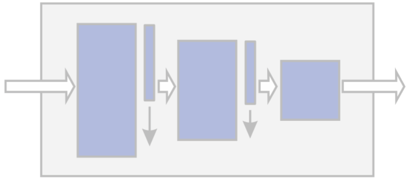
Types of Neural Networks

Network structure(s):

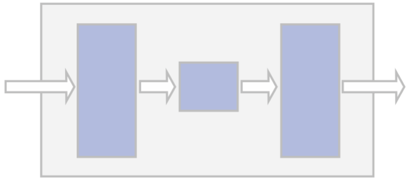
Multilayer perceptron



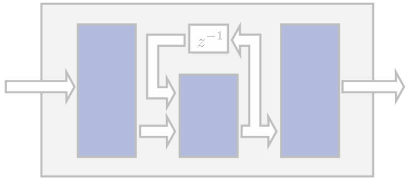
Convolutional neural network



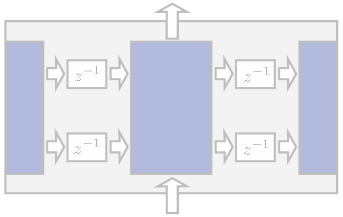
Autoencoder



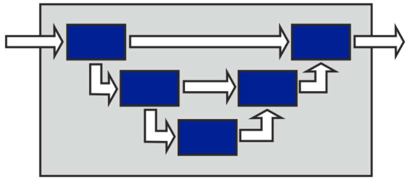
Recurrent neural network



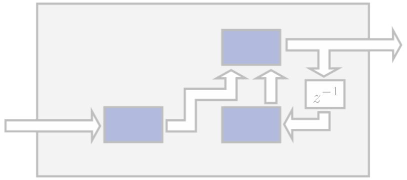
LSTM network



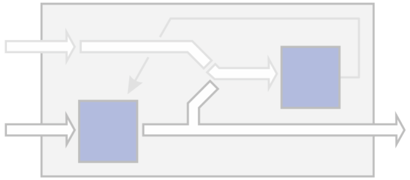
"U-net"



Attention-based network



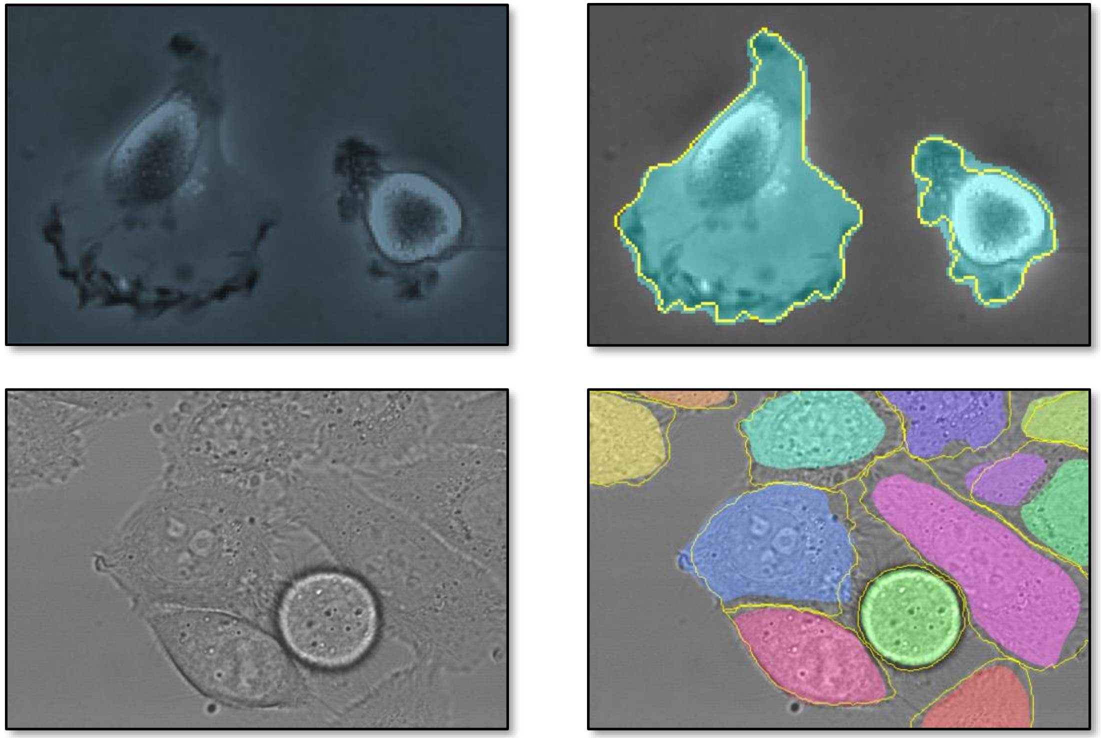
Generative adversarial network



U-net

Motivation:

- Originally designed for ***image segmentation*** (in contrast to image classification) for medical applications
- ***Two*** new ***ideas***:
  - ***Data*** (input and labels) ***duplication*** with modification ([non-linear] stretching, rotation, subsampling, …)
  - ***New network architecture*** consisting of a ***contraction (encoder)*** and ***expansion (decoder)*** path with a ***bottleneck*** in between



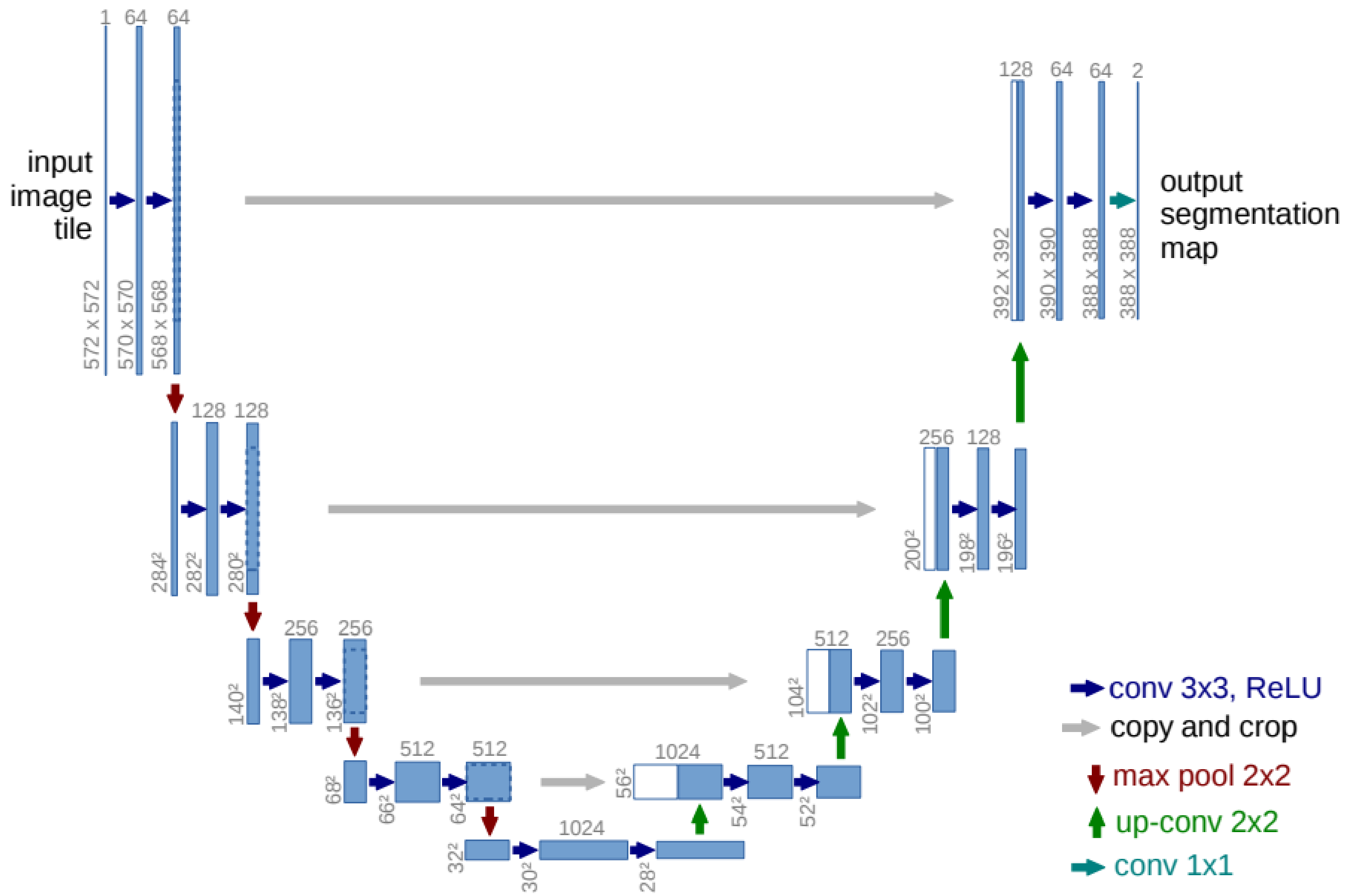
Input pictures from the PhC-U373 data set (ISBI cell tracking challenge)

Output pictures (yellow = manual labeling, colored areas = u-net results)

U-net

Motivation:

- Originally designed for ***image segmentation*** (in contrast to image classification) for medical applications
- ***Two*** new ***ideas***:
  - ***Data*** (input and labels) ***duplication*** with modification ([non-linear] stretching, rotation, subsampling, …)
  - ***New network architecture*** consisting of a ***contraction (encoder)*** and ***expansion (decoder)*** path with a ***bottleneck*** in between

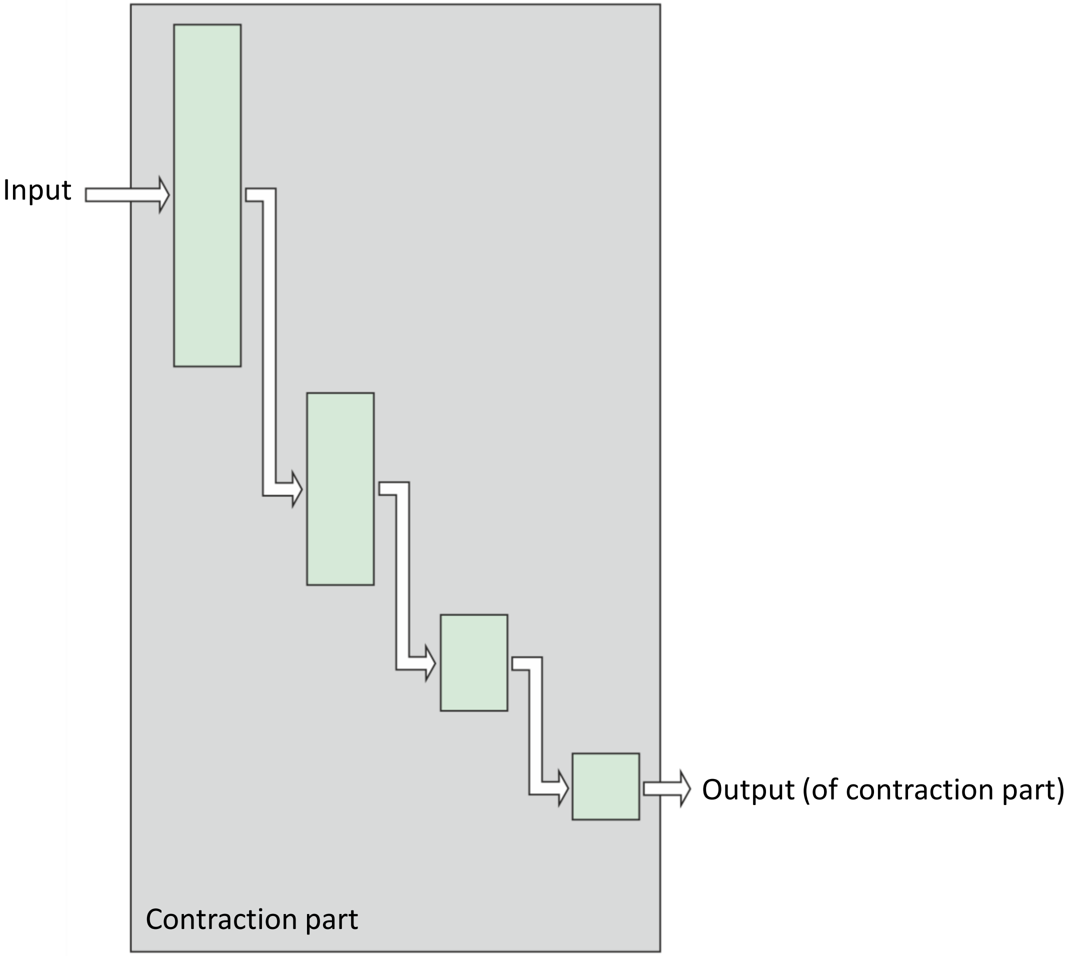


Ronneberger, Fischer and Brox, ***U-Net: Convolutional Neural Networks for Biomdical Image Segmentation***, In: Medical Image Computing and Computer-Assisted Intervention (MICCAI), LNCS, Springer, 2015, vol. 9351, p. 234-241. Available at: https://arxiv.org/abs/1505.04597

U-net

Structure:

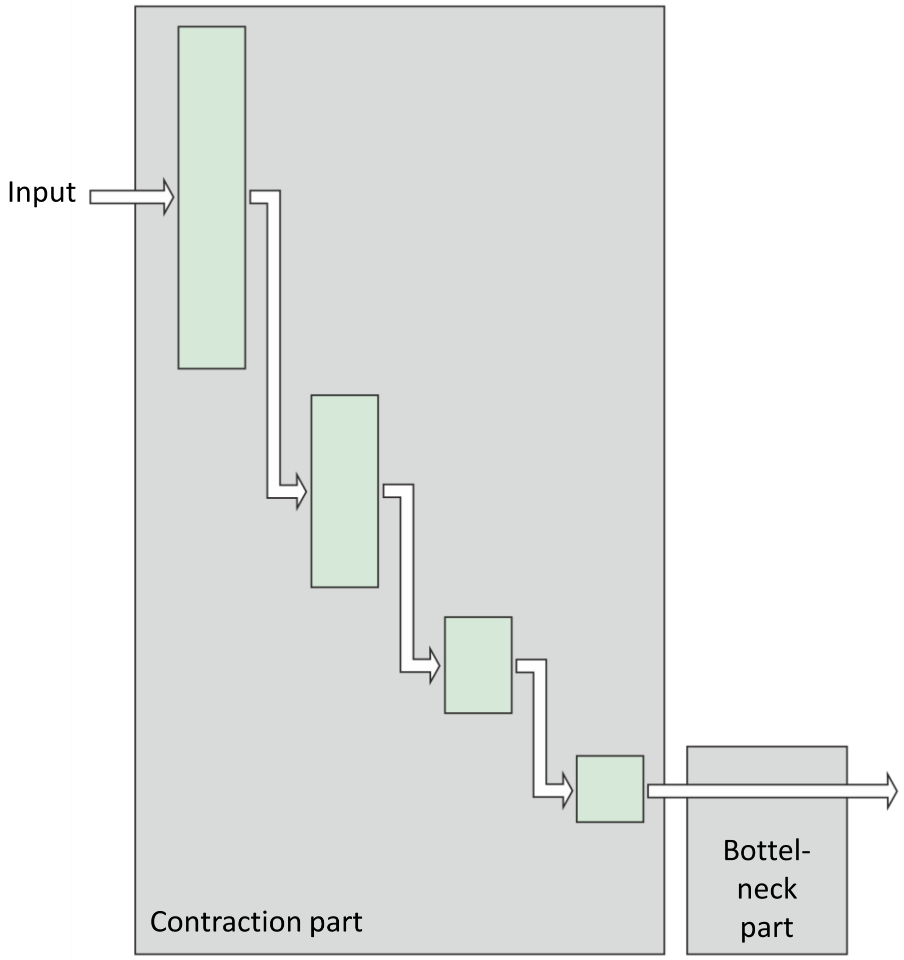
- The ***contraction path*** is about "what is to be seen in the image", and not so much where (contextual feature extraction).
  - It is built of convolution layers (multiple layers per green/blue box) and ReLU activation, followed by a pooling layer to reduce the image resolution.
  - The number of feature maps (filters) increases with each level, the image resolution decreases.



U-net

Structure:

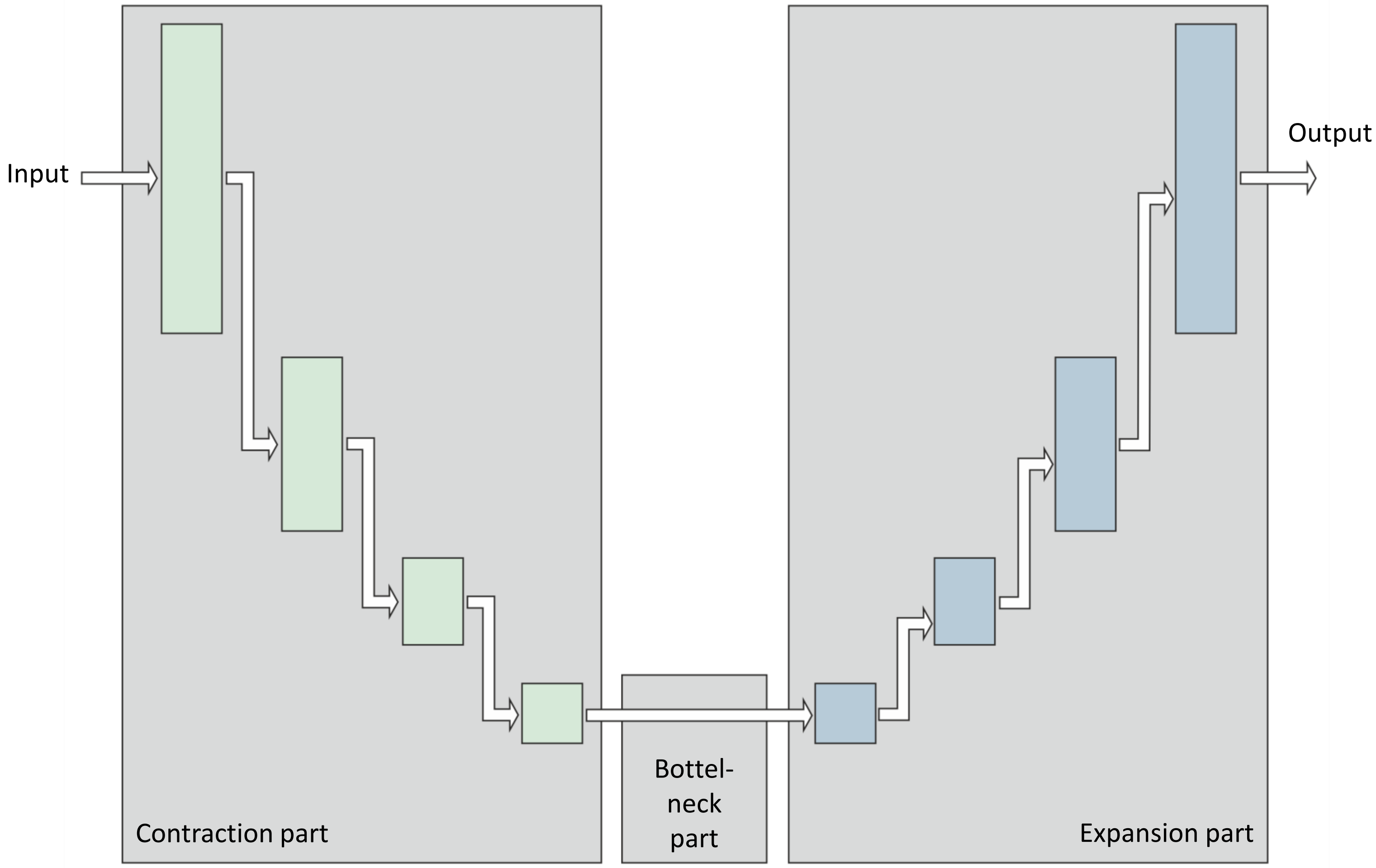
- The ***bottleneck*** is used to compress features in a more concise way.



U-net

Structure:

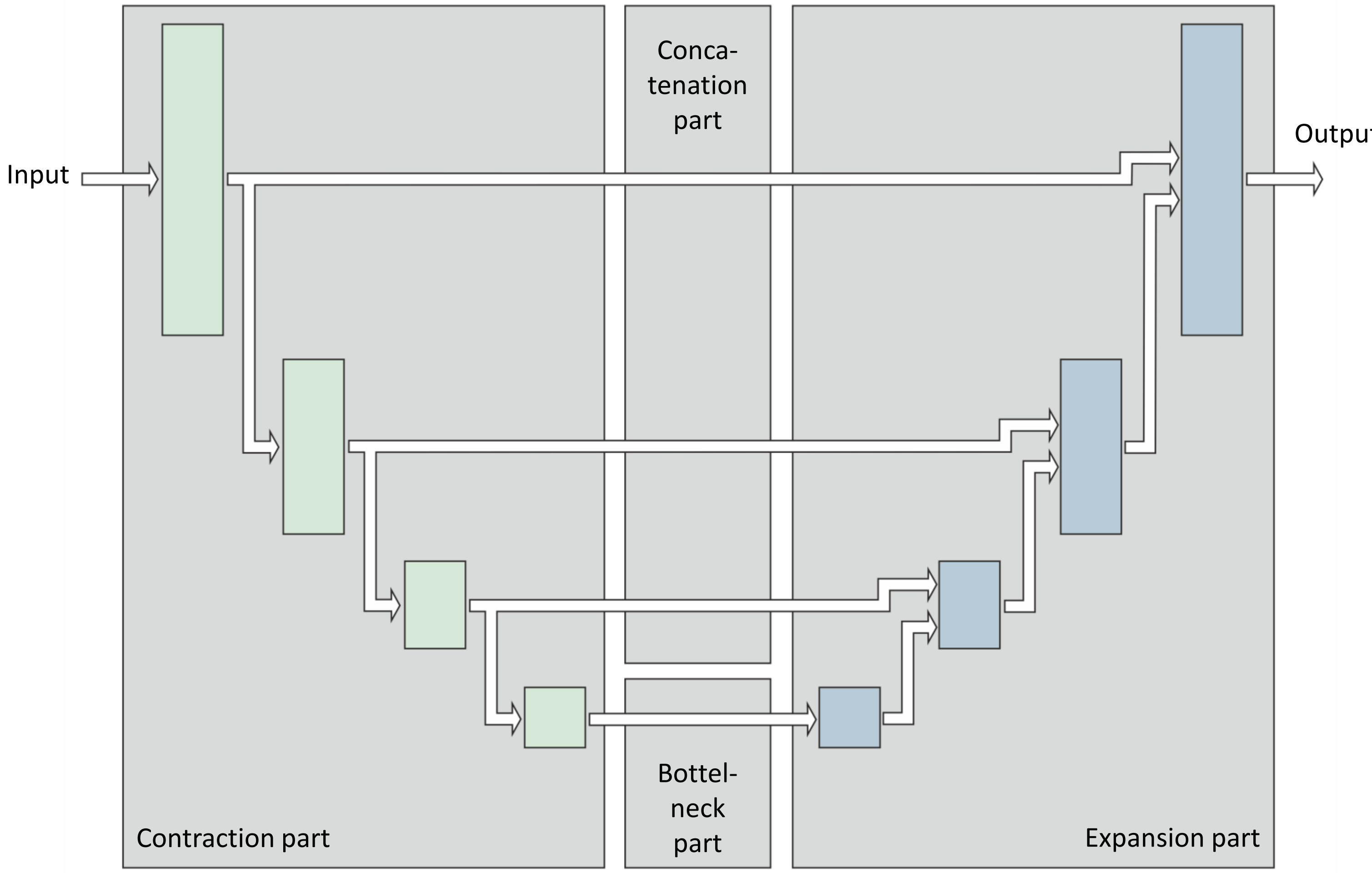
- The ***expansion path*** is used to localize where features appear within the image. It creates a high-resolution image map.
  - It uses upconvolution (upsampling) to increase the image resolution.



U-net

Structure:

- The ***expansion path*** is used to localize where features appear within the image. It creates a high-resolution image map.
  - It uses “upconvolution” (upsampling) to increase the image resolution,
  - concatenates the feature maps with those from the contraction path at the same level,
  - and applies convolution.

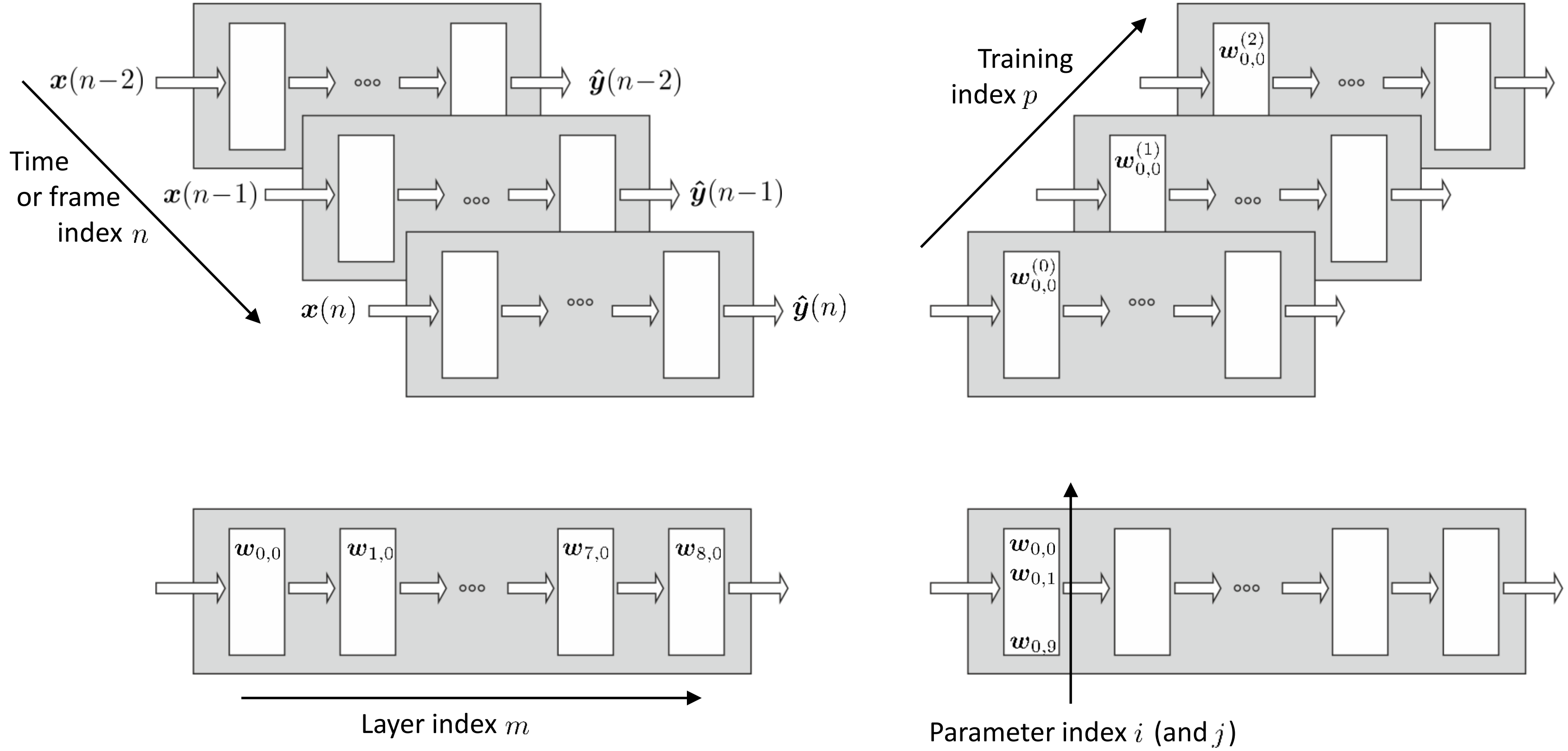



Contents

Training of Neural Networks – Basics

Preliminary items – part 1:

- In order to be mathematically correct, ***several indices*** are necessary:
  - ***Time*** or frame ***index*** n .
  - ***Layer index*** m.
  - ***Parameter index*** i.
  - ***Training index*** p.
- However, some of the indices will be ***dropped*** in the following slides for the reason of ***better readability***.



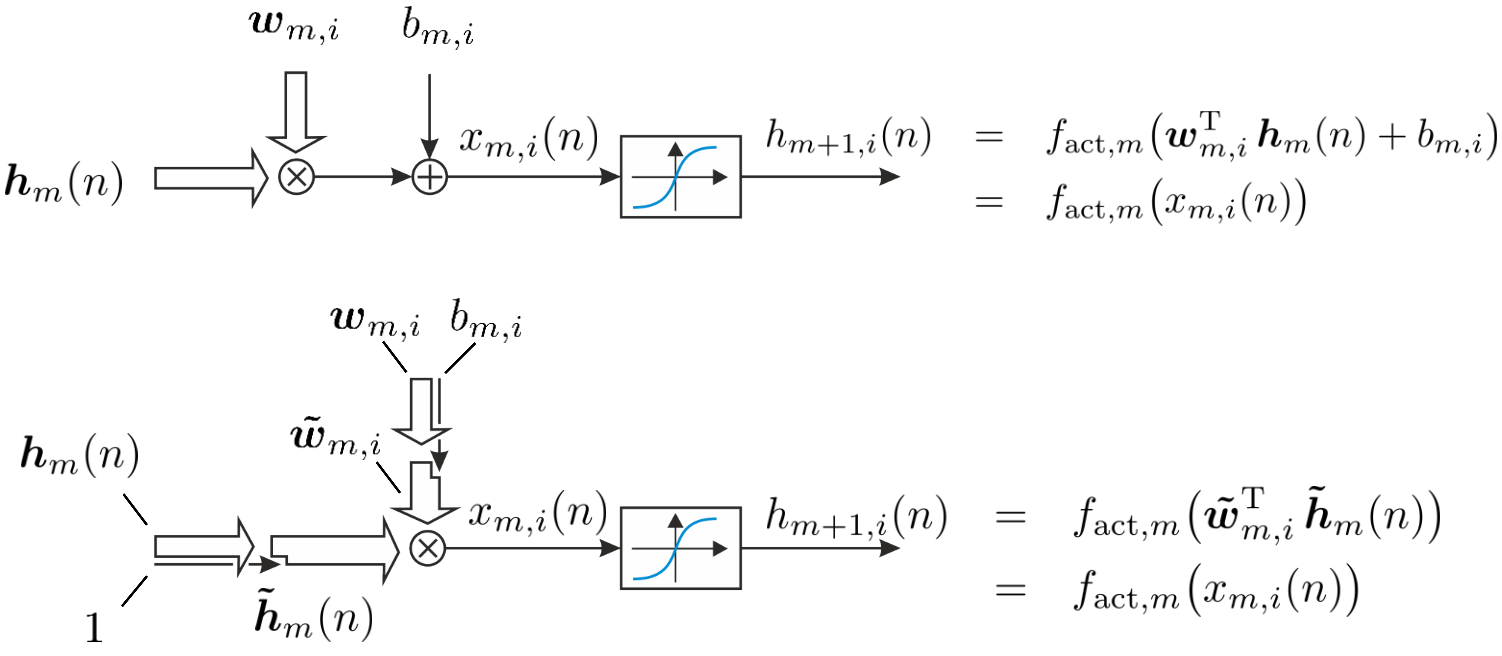
Training of Neural Networks – Basics

Preliminary items – part 2:

- For a simpler description ***extended parameter vectors*** and ***extended signal vectors*** will be used in the following:

$$\tilde{\boldsymbol{h}}_m(n) = \left[\boldsymbol{h}_m^{\mathrm{T}}(n),\, 1\right]^{\mathrm{T}},$$

$$\tilde{\boldsymbol{w}}_{m,i}(n) = \left[\boldsymbol{w}_{m,i}^{\mathrm{T}}(n),\, b_{m,i}\right]^{\mathrm{T}}.$$



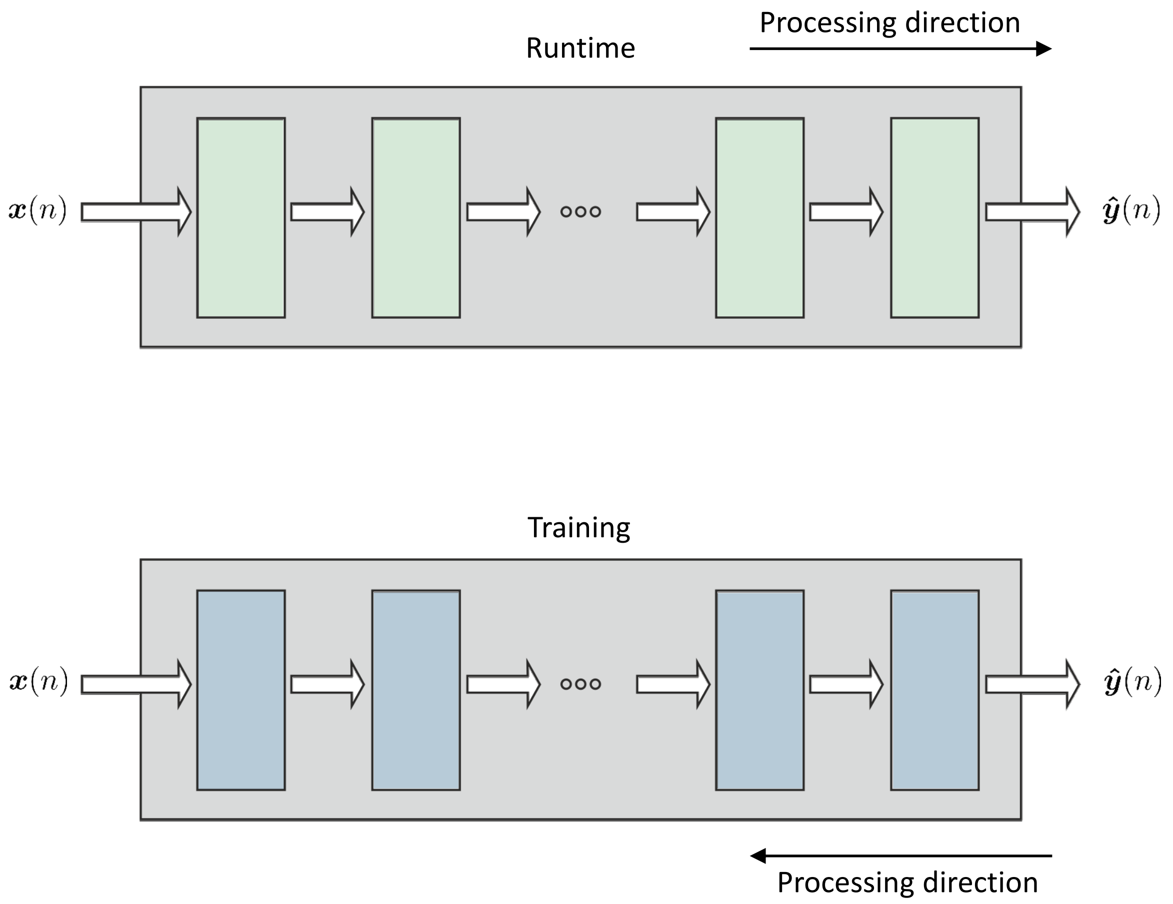
- The ***input*** of the activation function will be denoted with

$$x_{m,i}(n) = \boldsymbol{w}_{m,i}^{\mathrm{T}}\,\boldsymbol{h}_m(n) + b_{m,i} = \tilde{\boldsymbol{w}}_{m,i}^{\mathrm{T}}\,\tilde{\boldsymbol{h}}_m(n).$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- A popular training algorithm for neural networks is the so-called ***back-propagation algorithm***.
- The algorithm is minimizing a cost function by means of ***gradient descent*** steps.
- The ***chain rule*** in differentiation plays an important role and it is necessary that ***the activation functions*** are ***continuous*** and differentiable.
- While the network is computed during run-time from the input layer to the output layer, the back-propagation algorithm works ***from the output*** layer ***to the input*** one.



Training of Neural Networks – Back Propagation

Cost function:

- A basic goal of the network might be to ***minimize the average norm of the difference between the desired and the estimated feature vectors***:

$$C = \sum_{n=0}^{N-1} \big\|\boldsymbol{y}(n) - \hat{\boldsymbol{y}}(n)\big\|_2^2 \longrightarrow \min .$$

- In order to achieve this goal all parameters of the neural network are corrected in ***negative gradient direction (method of steepest descent)***:

$$-\boldsymbol{\nabla}_{\tilde{\boldsymbol{w}}_{m,i}} C = -\frac{\partial C}{\partial \tilde{\boldsymbol{w}}_{m,i}} .$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- The ***cost function*** is "***refined***" as follows:

Training index

$$C^{(p)} = \sum_{n=0}^{N-1} \underbrace{\left\| \boldsymbol{y}(n) - \hat{\boldsymbol{y}}^{(p)}(n) \right\|_2^2}_{e^{(p)}(n)} = \sum_{n=0}^{N-1} e^{(p)}(n) \longrightarrow \min .$$

- The ***gradient*** of the cost function consists of several ***partial differentiations***:

$$\boldsymbol{\nabla}_{\tilde{\boldsymbol{w}}_{m,i}^{(p)}} C^{(p)} = \frac{\partial C^{(p)}}{\partial \tilde{\boldsymbol{w}}_{m,i}^{(p)}} = \left[\frac{\partial C^{(p)}}{\partial \tilde{w}_{m,i,0}^{(p)}}, \frac{\partial C^{(p)}}{\partial \tilde{w}_{m,i,1}^{(p)}}, \frac{\partial C^{(p)}}{\partial \tilde{w}_{m,i,2}^{(p)}}, \ldots \right]^{\mathrm{T}} .$$

- The ***parameters are updated*** during the training process according to:

$$\tilde{\boldsymbol{w}}_{m,i}^{(p+1)} = \tilde{\boldsymbol{w}}_{m,i}^{(p)} - \frac{\alpha}{2} \frac{\partial C^{(p)}}{\partial \tilde{\boldsymbol{w}}_{m,i}^{(p)}} .$$

Step-size parameter

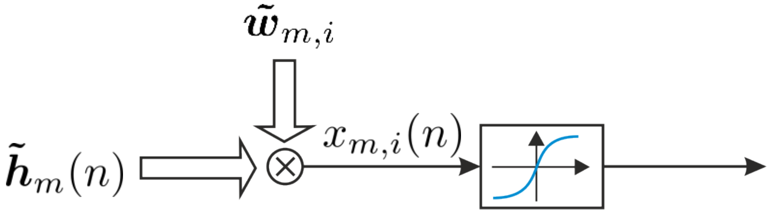
Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- We will focus now on a ***single differentiation*** (with respect to only one parameter). Here, we ***insert the details of the cost function*** and we ***omit the training index for better readability***:

$$\frac{\partial C}{\partial \tilde{w}_{m,i,j}} = \frac{\partial}{\partial \tilde{w}_{m,i,j}} \sum_{n=0}^{N-1} e(n) = \sum_{n=0}^{N-1} \frac{\partial e(n)}{\partial \tilde{w}_{m,i,j}}.$$

- Keep the structure of the individual neurons in mind ….



$$\begin{aligned} \tilde{h}_{m+1,i}(n) &\\ = f_{\mathrm{act},m}&\big(\tilde{\boldsymbol{w}}_{m,i}^{\mathrm{T}}\,\tilde{\boldsymbol{h}}_m(n)\big) \\ = f_{\mathrm{act},m}&\big(x_{m,i}(n)\big) \end{aligned}$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- First, we will compute the update of the weights in the ***output layer*** ($m = M$):

$$\frac{\partial C}{\partial \tilde{w}_{M,i,j}} = \frac{\partial}{\partial \tilde{w}_{M,i,j}} \sum_{n=0}^{N-1} e(n) = \sum_{n=0}^{N-1} \frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}}.$$

- All individual gradients (individual for all input frames n) can be summed and then an update is performed or an update can be performed after each gradient computation. For reasons of brevity we will compute now only ***individual gradients***. In order to compute the gradient, we ***split the global gradient into a product of two simpler gradients***:

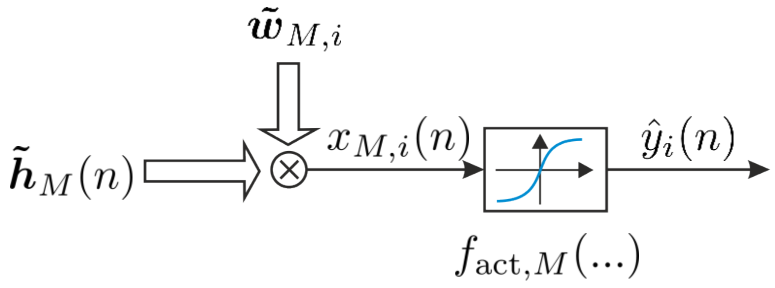
$$\frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} = \frac{\partial e(n)}{\partial x_{M,i}(n)} \frac{\partial x_{M,i}(n)}{\partial \tilde{w}_{M,i,j}}.$$

- This "***trick***" ***will be repeated*** but now for the multivariate case to compute the gradients for the ***weights of the hidden layers***:

$$\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = \sum_{k} \frac{\partial e(n)}{\partial x_{M,k}(n)} \frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} \frac{\partial x_{M-1,i}(n)}{\partial \tilde{w}_{M-1,i,j}}.$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:



- Let's start now with the ***gradient for the weights of the output layer***:

$$\frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} = \frac{\partial e(n)}{\partial x_{M,i}(n)} \, \frac{\partial x_{M,i}(n)}{\partial \tilde{w}_{M,i,j}}.$$

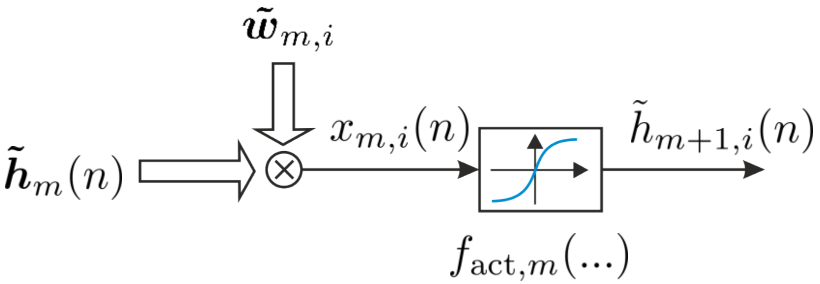
$$\frac{\partial x_{M,i}(n)}{\partial \tilde{w}_{M,i,j}} = \frac{\partial}{\partial \tilde{w}_{M,i,j}} \sum_d \tilde{w}_{M,i,d}\, \tilde{h}_{M,d}(n) = \tilde{h}_{M,j}(n),$$

$$\begin{aligned}\frac{\partial e(n)}{\partial x_{M,i}} &= \frac{\partial}{\partial x_{M,i}} \sum_d \big(y_d(n) - \hat{y}_d(n)\big)^2 \\ &= -2 \sum_d \big(y_d(n) - \hat{y}_d(n)\big) \frac{\partial \hat{y}_d(n)}{\partial x_{M,i}(n)} \\ &= -2 \big(y_i(n) - \hat{y}_i(n)\big)\, f'_{\text{act},M}\big(x_{M,i}(n)\big).\end{aligned}$$

$$\frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} = -2 \big(y_i(n) - \hat{y}_i(n)\big)\, f'_{\text{act},M}\big(x_{M,i}(n)\big)\; \tilde{h}_{M,j}(n)$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:



- For the ***second last layer*** we can do the ***same for the first and the last term***:

$$\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = \sum_k \frac{\partial e(n)}{\partial x_{M,k}(n)} \frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} \frac{\partial x_{M-1,i}(n)}{\partial \tilde{w}_{M-1,i,j}}.$$

$$\frac{\partial x_{M-1,i}(n)}{\partial \tilde{w}_{M-1,i,j}} = \tilde{h}_{M-1,j}(n),$$

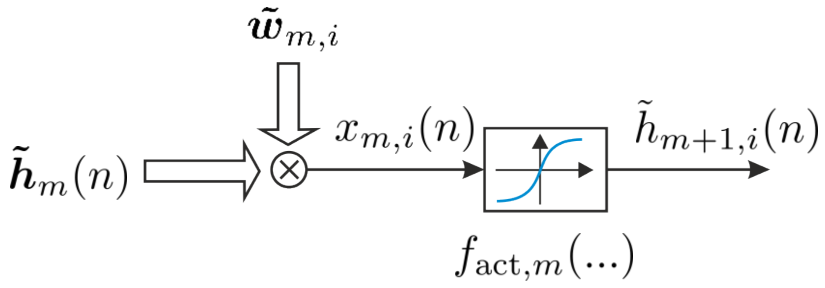
$$\frac{\partial e(n)}{\partial x_{M,k}} = -2\left(y_k(n) - \hat{y}_k(n)\right) f'_M\left(x_{M,k}(n)\right).$$

- Now only the ***center term*** is missing:

$$\frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} = \ldots$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:



- The ***missing term***:

$$
\begin{aligned}
\frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} &= \frac{\partial}{\partial x_{M-1,i}(n)} \sum_d \tilde{h}_{M,d}(n)\, \tilde{w}_{M,k,d} \\
&= \frac{\partial}{\partial x_{M-1,i}(n)} \sum_d f_{\text{act},M-1}\big(x_{M-1,d}(n)\big)\, \tilde{w}_{M,k,d} \\
&= f'_{\text{act},M-1}\big(x_{M-1,i}(n)\big)\, \tilde{w}_{M,k,i}
\end{aligned}
$$

- Putting ***everything together*** leads to:

$$
\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = -2\, \tilde{h}_{M-1,j}(n) \sum_k \big(y_k(n) - \hat{y}_k(n)\big)\, f'_{\text{act},M}\big(x_{M,k}(n)\big)\, f'_{\text{act},M-1}\big(x_{M-1,i}(n)\big)\, \tilde{w}_{M,k,i}.
$$

Back-propagation algorithm:

- Two more layers to see the structure:

$$
\begin{aligned}
\frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} &= \frac{\partial e(n)}{\partial x_{M,i}(n)} \frac{\partial x_{M,i}(n)}{\partial \tilde{w}_{M,i,j}}, \\
\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} &= \sum_k \frac{\partial e(n)}{\partial x_{M,k}(n)} \frac{\partial x_{M,k}(n)}{\partial x_{M-1,i}(n)} \frac{\partial x_{M-1,i}(n)}{\partial \tilde{w}_{M-1,i,j}}, \\
\frac{\partial e(n)}{\partial \tilde{w}_{M-2,i,j}} &= \sum_k \frac{\partial e(n)}{\partial x_{M,k}(n)} \sum_\ell \frac{\partial x_{M,k}(n)}{\partial x_{M-1,\ell}(n)} \frac{\partial x_{M-1,\ell}(n)}{\partial x_{M-2,i}(n)} \frac{\partial x_{M-2,i}(n)}{\partial \tilde{w}_{M-2,i,j}}, \\
\frac{\partial e(n)}{\partial \tilde{w}_{M-3,i,j}} &= \sum_k \frac{\partial e(n)}{\partial x_{M,k}(n)} \sum_\ell \frac{\partial x_{M,k}(n)}{\partial x_{M-1,\ell}(n)} \sum_m \frac{\partial x_{M-1,\ell}(n)}{\partial x_{M-2,m}(n)} \frac{\partial x_{M-2,m}(n)}{\partial x_{M-3,i}(n)} \frac{\partial x_{M-3,i}(n)}{\partial \tilde{w}_{M-3,i,j}}.
\end{aligned}
$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- Interesting is, that the individual differentiations can be ***computed recursively***. Let's have a first look on the results (the third last layer was not derived before, but it's straight forward). Let's ***start with the last layer***:

$$\frac{\partial e(n)}{\partial \tilde{w}_{M,i,j}} = -2\left(y_i(n) - \hat{y}_i(n)\right) f'_{\mathrm{act},M}\left(x_{M,i}(n)\right) \; \tilde{h}_{M,j}(n)$$

- Here we introduce the following "***helping***" ***variables***:

$$\delta_{M,i}(n) = \left(y_i(n) - \hat{y}_i(n)\right) f'_{\mathrm{act},M}\left(x_{M,i}(n)\right).$$

- To be a bit more precise, we ***add also the iteration index***:

$$\delta^{(p)}_{M,i}(n) = \left(y_i(n) - \hat{y}^{(p)}_i(n)\right) f'_{\mathrm{act},M}\left(x^{(p)}_{M,i}(n)\right).$$

- Now the ***update of the parameters of the last layer*** (change in negative gradient direction) can be written as

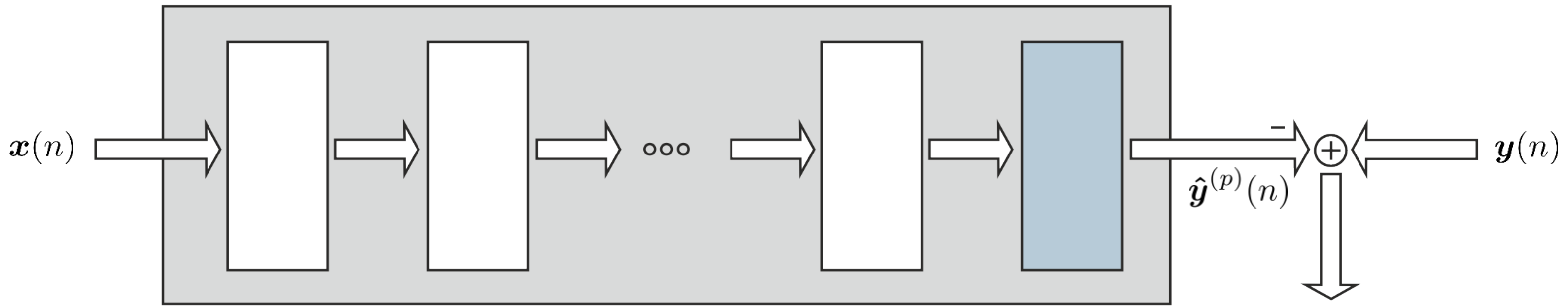
$$\tilde{w}^{(p+1)}_{M,i,j} = \tilde{w}^{(p)}_{M,i,j} + \alpha \sum_{n=0}^{N-1} \delta^{(p)}_{M,i}(n)\, \tilde{h}^{(p)}_{M,j}(n).$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- Visualization – ***last layer***:

Compute in forward direction $\tilde{h}^{(p)}_{M,i}(n)$ and $x^{(p)}_{M,i}(n)$!



Initialize helping variables in backward direction

$$\delta^{(p)}_{M,i}(n) \quad = \quad \big(y_i(n) - \hat{y}^{(p)}_i(n)\big)\, f'_{\mathrm{act},M}\big(x^{(p)}_{M,i}(n)\big)$$

and update the parameter of the last layer

$$\tilde{w}^{(p+1)}_{M,i,j} \quad = \quad \tilde{w}^{(p)}_{M,i,j} + \alpha \sum_{n=0}^{N-1} \delta^{(p)}_{M,i}(n)\, \tilde{h}^{(p)}_{M,j}(n).$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- Now the ***second last layer***:

$$\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = -2\,\tilde{h}_{M-1,j}(n) \sum_k \big(y_k(n) - \hat{y}_k(n)\big)\, f'_{\text{act},M}\big(x_{M,k}(n)\big)\, f'_{\text{act},M-1}\big(x_{M-1,i}(n)\big)\, \tilde{w}_{M,k,i}.$$

- Here we can insert the "***helping***" ***variables*** from the last layer:

$$\begin{aligned}\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} &= -2\,\tilde{h}_{M-1,j}(n) \sum_k \underbrace{\big(y_k(n) - \hat{y}_k(n)\big)\, f'_{\text{act},M}\big(x_{M,k}(n)\big)}_{\delta_{M,k}(n)}\, f'_{\text{act},M-1}\big(x_{M-1,i}(n)\big)\, \tilde{w}_{M,k,i} \\ &= -2\,\tilde{h}_{M-1,j}(n) \sum_k \delta_{M,k}(n)\, f'_{\text{act},M-1}\big(x_{M-1,i}(n)\big)\, \tilde{w}_{M,k,i}.\end{aligned}$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- Result of last slide:

$$\frac{\partial e(n)}{\partial \tilde{w}_{M-1,i,j}} = -2\,\tilde{h}_{M-1,j}(n) \sum_k \delta_{M,k}(n)\, f'_{\mathrm{act},M-1}\big(x_{M-1,i}(n)\big)\, \tilde{w}_{M,k,i}.$$

- Again, this could be separated in two steps. First a ***helping variable*** is ***updated*** (again, now with the training index):

$$\delta^{(p)}_{M-1,i}(n) = f'_{\mathrm{act},M-1}\big(x^{(p)}_{M-1,i}(n)\big) \sum_k \delta^{(p)}_{M,k}(n)\, \tilde{w}^{(p)}_{M,k,i}.$$

- Now, the ***update of the parameters of the second last layer*** can be performed according to

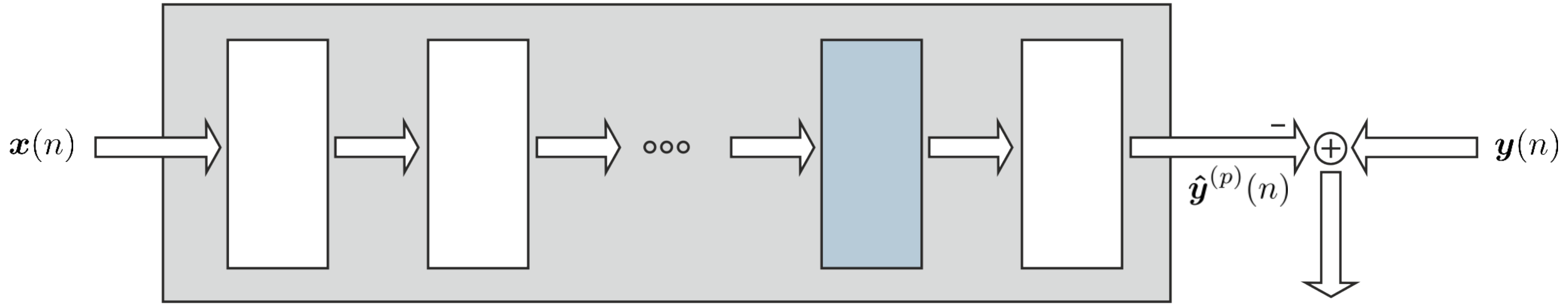
$$\tilde{w}^{(p+1)}_{M-1,i,j} = \tilde{w}^{(p)}_{M-1,i,j} + \alpha \sum_{n=0}^{N-1} \delta^{(p)}_{M-1,i}(n)\, \tilde{h}^{(p)}_{M-1,j}(n).$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- Visualization – ***second last layer***:

Compute in forward direction $\tilde{h}^{(p)}_{M-1,i}(n)$ and $x^{(p)}_{M-1,i}(n)$!



Update helping variables in backward direction

$$\delta^{(p)}_{M-1,i}(n) \quad = \quad f'_{\mathrm{act},M-1}\big(x^{(p)}_{M-1,i}(n)\big) \sum_k \delta^{(p)}_{M,k}(n)\,\tilde{w}^{(p)}_{M,k,i}$$

and update the parameter of the second last layer

$$\tilde{w}^{(p+1)}_{M-1,i,j} \quad = \quad \tilde{w}^{(p)}_{M-1,i,j} + \alpha \sum_{n=0}^{N-1} \delta^{(p)}_{M-1,i}(n)\,\tilde{h}^{(p)}_{M-1,j}(n).$$

Training of Neural Networks – Back Propagation

Back-propagation algorithm:

- This goes on until the first layer is reached. First an update of the helping variables:

$$\delta_{m-1,i}^{(p)}(n) \quad = \quad f'_{\mathrm{act},m-1}\big(x_{m-1,i}^{(p)}(n)\big) \sum_k \delta_{m,k}^{(p)}(n)\, \tilde{w}_{m,k,i}^{(p)}.$$

- And then an update of the network parameters:

$$\tilde{w}_{m-1,i,j}^{(p+1)} \quad = \quad \tilde{w}_{m-1,i,j}^{(p)} + \alpha \sum_{n=0}^{N-1} \delta_{m-1,i}^{(p)}(n)\, \tilde{h}_{m-1,j}^{(p)}(n).$$

- As in the case of codebooks, GMMs, HMMs it is checked by using test and validation data, if the cost function does increase. In that case the ***training is stopped***. Furthermore, several ***variants of this basic update strategies*** have been published. Details can be found in the references.


Contents

Training of Neural Networks – Update Rules

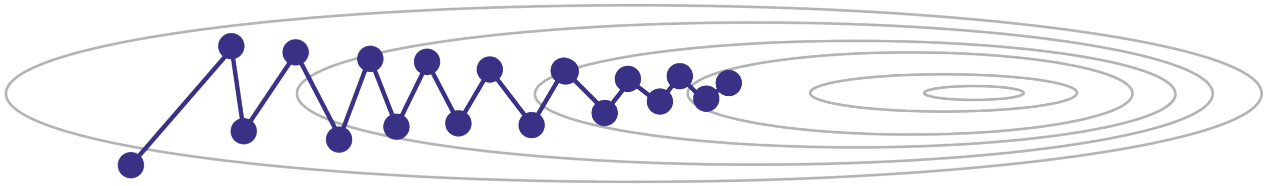
Extensions for gradient-based corrections:

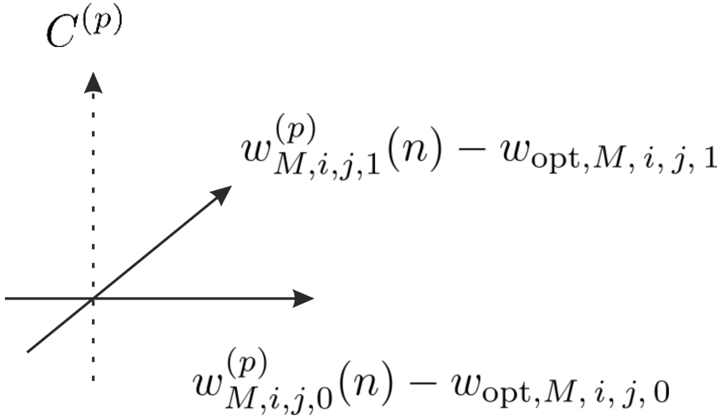
- Two ***basic extensions***
  - Gradient descent with momentum (***Momentum***)
  - Root mean square propagation (***RMSprop***)
- ***Combination*** of both
  - Adaptive moment estimation (***Adam***)

Training of Neural Networks – Update Rules

Extensions for gradient-based corrections:

- Two ***basic extensions***
  - Gradient descent with momentum (***Momentum***)
  - Root mean square propagation (***RMSprop***)
- ***Combination*** of both
  - Adaptive moment estimation (***Adam***)





Training of Neural Networks – Update Rules

Extensions for gradient-based corrections:

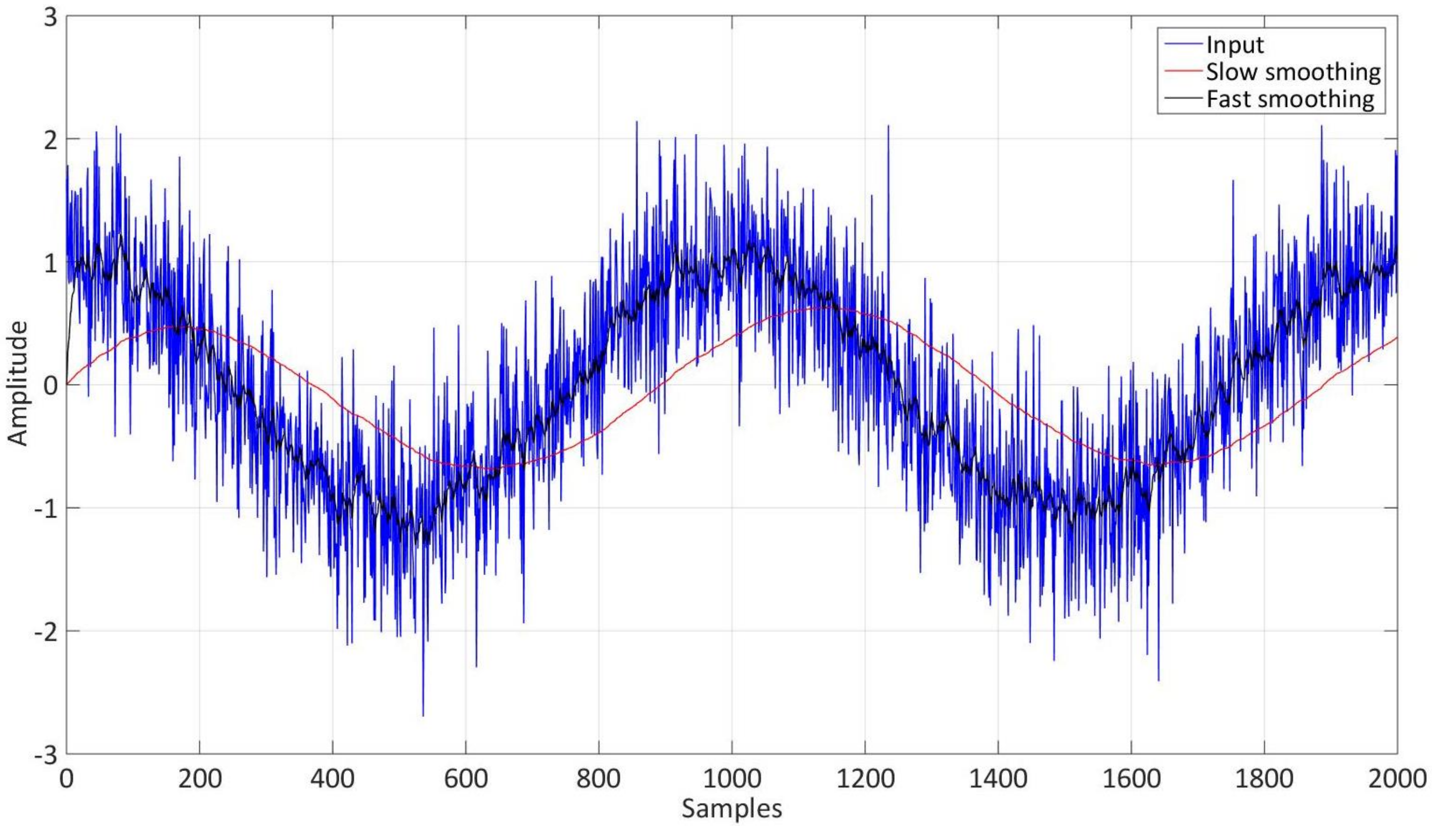
- Recursive smoothing

$$y(n) = \beta\, y(n-1) + (1-\beta)\, x(n)$$

$$\beta_{\text{fast}} = 0.9$$
$$\beta_{\text{fast}} = 0.99$$

- A compromise between being able to follow (desired) trends in the signal and the amount of noise reduction has to be found.
- After being converged this estimation is bias-free.
- In contrast to this version:

$$y(n) = \beta\, y(n-1) + x(n)$$



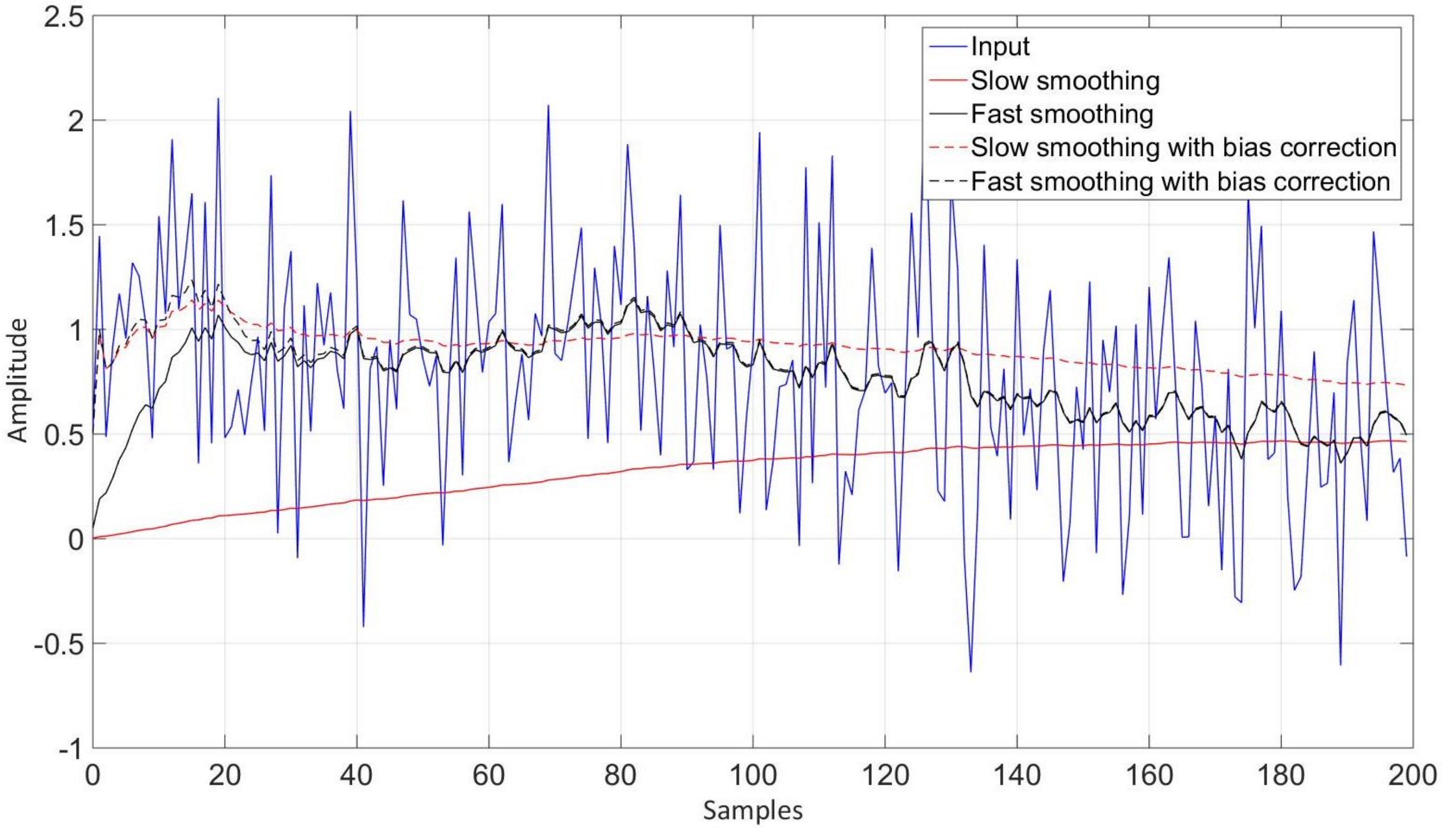
Training of Neural Networks – Update Rules

Extensions for gradient-based corrections:

- Recursive smoothing with bias correction (mainly for the startup phase):

$$
\begin{aligned}
y(n) &= \beta\, y(n-1) + (1-\beta)\, x(n) \\
b_{\text{corr}}(n) &= b_{\text{corr}}(n-1)\, \beta \\
y_{\text{corr}}(n) &= \frac{y(n)}{1 - b_{\text{corr}}(n)}
\end{aligned}
$$

- Needs to be done only for the first few samples.



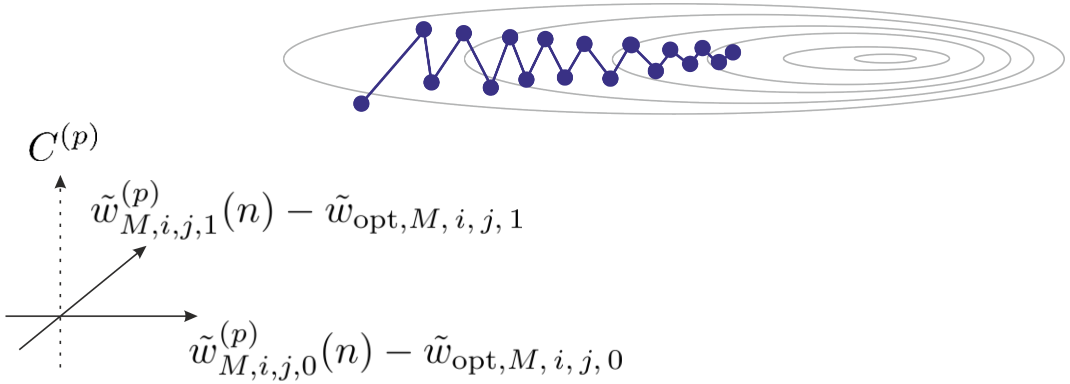
Training of Neural Networks – Update Rules

Extensions for gradient-based corrections:

- Gradient descent with momentum (***Momentum***)
  - Previous update rule (without momentum, in vector notation)

$$\tilde{\boldsymbol{w}}_{m,i}^{(p+1)} = \tilde{\boldsymbol{w}}_{m,i}^{(p)} - \frac{\alpha}{2} \frac{\partial C^{(p)}}{\partial \tilde{\boldsymbol{w}}_{m,i}^{(p)}}.$$

Step-size parameter



  - Previous update rule (without momentum, in scalar notation)

$$\tilde{w}_{m,i,k}^{(p+1)} = \tilde{w}_{m,i,k}^{(p)} - \frac{\alpha}{2} \frac{\partial C^{(p)}}{\partial \tilde{w}_{m,i,k}^{(p)}} = \tilde{w}_{m,i,k}^{(p)} - \frac{\alpha}{2} \Delta_{m,i,k}^{(p)}.$$

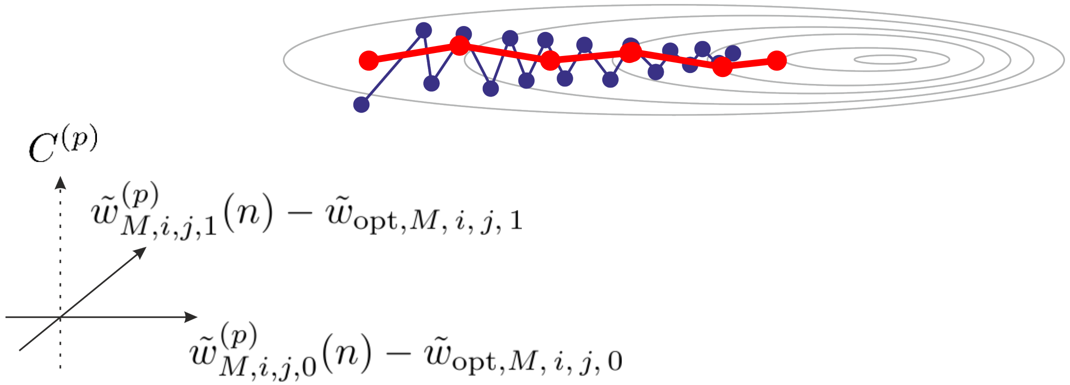
Step-size parameter

Training of Neural Networks – Update Rules

Extensions for gradient-based corrections:

- Gradient descent with momentum (***Momentum***)
  - Previous update rule (without momentum, in scalar notation)

$$\tilde{w}^{(p+1)}_{m,i,k} = \tilde{w}^{(p)}_{m,i,k} - \frac{\alpha}{2}\,\Delta^{(p)}_{m,i,k}.$$



  - IIR smoothing of potential updates

$$\overline{\Delta}^{(p)}_{m,i,k} = \beta\,\overline{\Delta}^{(p-1)}_{m,i,k} + (1-\beta)\,\Delta^{(p)}_{m,i,k}.$$

  - Correction into smoothed update correction

$$\boxed{\tilde{w}^{(p+1)}_{m,i,k} = \tilde{w}^{(p)}_{m,i,k} - \frac{\overline{\alpha}}{2}\,\overline{\Delta}^{(p)}_{m,i,k}.}$$

Adjusted step-size parameter

Training of Neural Networks – Update Rules

Extensions for gradient-based corrections:

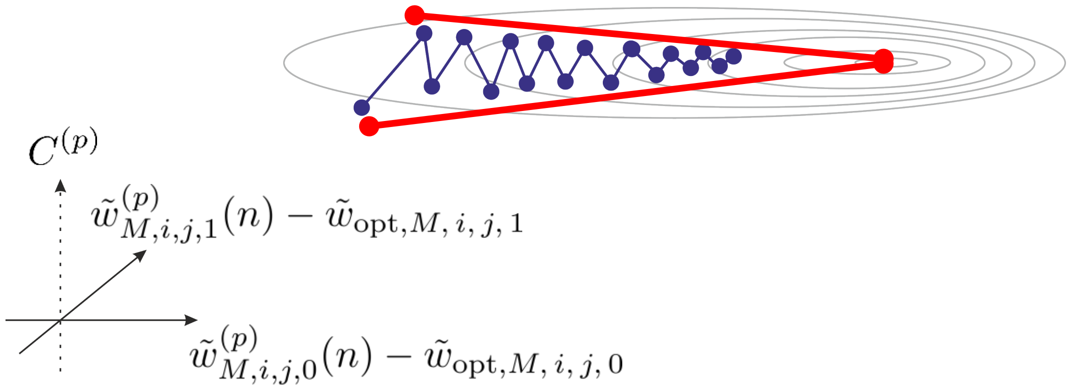
- Root mean square propagation (***RMSprop***)
  - The short-term variations of the gradient estimations might vary and it's usually advantages to take them into account as well.
  - Therefore the short-term variations can be estimated as well:

$$\overline{\Sigma}^{(p)}_{m,i,k} = \beta\,\overline{\Sigma}^{(p-1)}_{m,i,k} + (1-\beta)\left(\Delta^{(p)}_{m,i,k}\right)^2.$$

  - Afterwards the update can be normalized with the square root of this variance estimate:

$$\boxed{\tilde{w}^{(p+1)}_{m,i,k} = \tilde{w}^{(p)}_{m,i,k} - \frac{\tilde{\alpha}}{2}\,\frac{\Delta^{(p)}_{m,i,k}}{\sqrt{\overline{\Sigma}^{(p)}_{m,i,k}}}.}$$

Adjusted step-size parameter



Training of Neural Networks – Update Rules

Extensions for gradient-based corrections:

- Adaptive moment estimation (***Adam***)
  - A ***combination of both attempts*** leads to the so-called Adam optimization rule:

$$\overline{\Delta}_{m,i,k}^{(p)} = \beta_\Delta \, \overline{\Delta}_{m,i,k}^{(p-1)} + \big(1-\beta_\Delta\big)\, \Delta_{m,i,k}^{(p)}$$

$$\overline{\Sigma}_{m,i,k}^{(p)} = \beta_\Sigma \, \overline{\Sigma}_{m,i,k}^{(p-1)} + \big(1-\beta_\Sigma\big) \left(\Delta_{m,i,k}^{(p)}\right)^2$$

$$b_{\Delta,\mathrm{corr}}^{(p)} = b_{\Delta,\mathrm{corr}}^{(p-1)} \, \beta_\Delta$$

$$b_{\Sigma,\mathrm{corr}}^{(p)}(n) = b_{\Sigma,\mathrm{corr}}^{(p-1)} \, \beta_\Sigma$$

$$\overline{\Delta}_{\mathrm{corr},m,\,i,\,k}^{(p)} = \frac{\overline{\Delta}_{m,i,k}^{(p)}}{1-b_{\Delta,\mathrm{corr}}^{(p)}}$$

$$\overline{\Sigma}_{\mathrm{corr},m,\,i,\,k}^{(p)} = \frac{\overline{\Sigma}_{m,i,k}^{(p)}}{1-b_{\Sigma,\mathrm{corr}}^{(p)}}$$

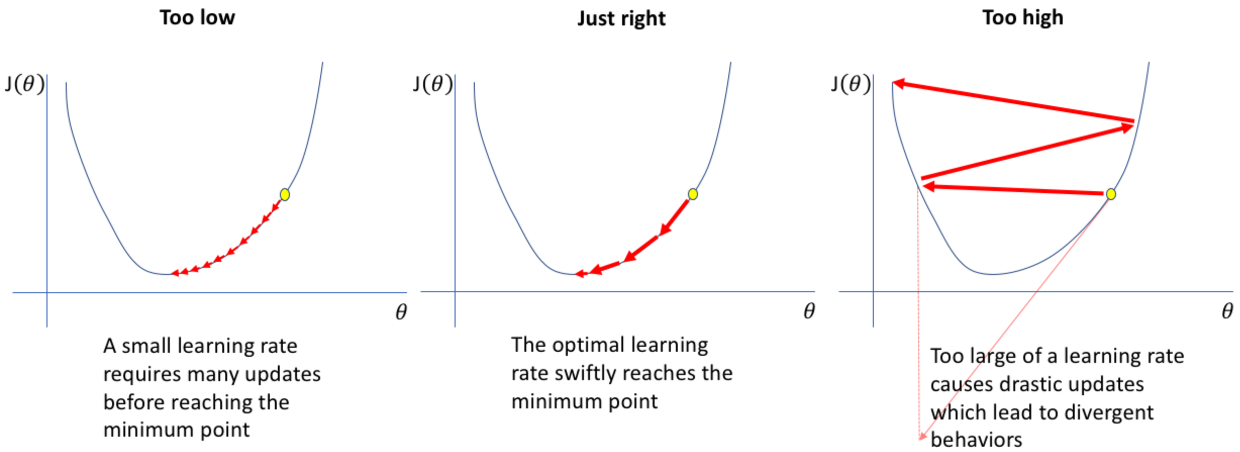
$$\boxed{\tilde{w}_{m,i,k}^{(p+1)} = \tilde{w}_{m,i,k}^{(p)} - \frac{\breve{\alpha}}{2} \, \frac{\overline{\Delta}_{\mathrm{corr},m,\,i,\,k}^{(p)}}{\sqrt{\overline{\Sigma}_{\mathrm{corr},m,\,i,\,k}^{(p)}}}.}$$


Contents

Learning Rate Scheduling Schemes

Basics:

- The ***learning rate (LR)*** is one of the most ***important hyperparameter***s for the training of neural networks.
- The ***most common*** approach is to use a ***fixed learning rate*** for the entire training.
- ***Fixed Schedules*** change (typically reduce) the learning rate after a ***fixed amount of gradient descent steps.***
- ***Adaptive Scheduling*** changes (typically reduces) the learning rate ***if some kind of condition is met***.

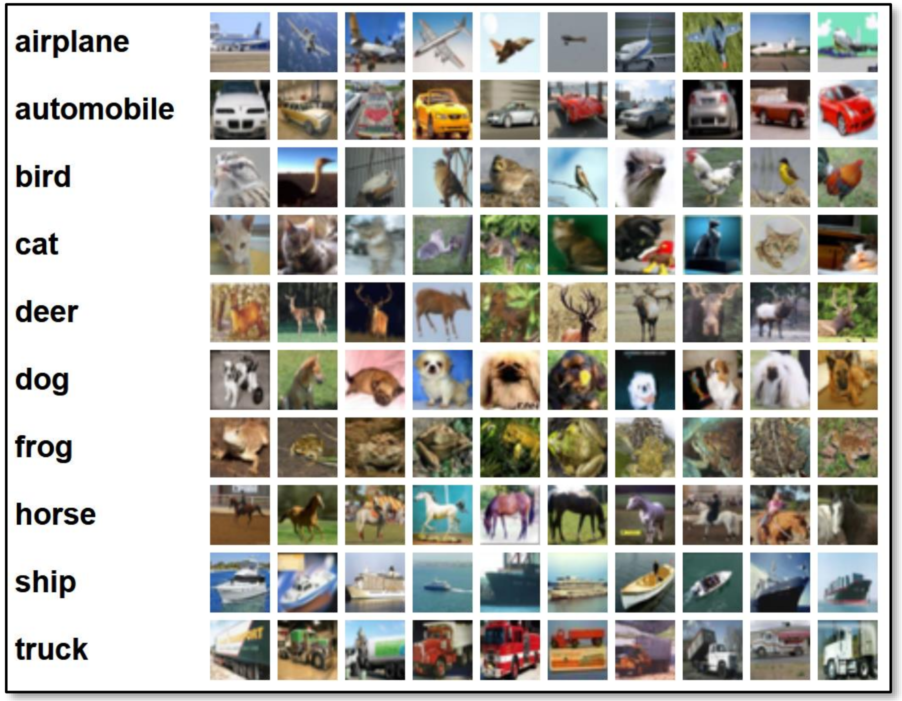


https://www.jeremyjordan.me/nn-learning-rate/

Learning Rate Scheduling Schemes

Practical example:

- Training a ***CNN*** on the ***CIFAR-10*** dataset with ***cross entropy*** as loss function

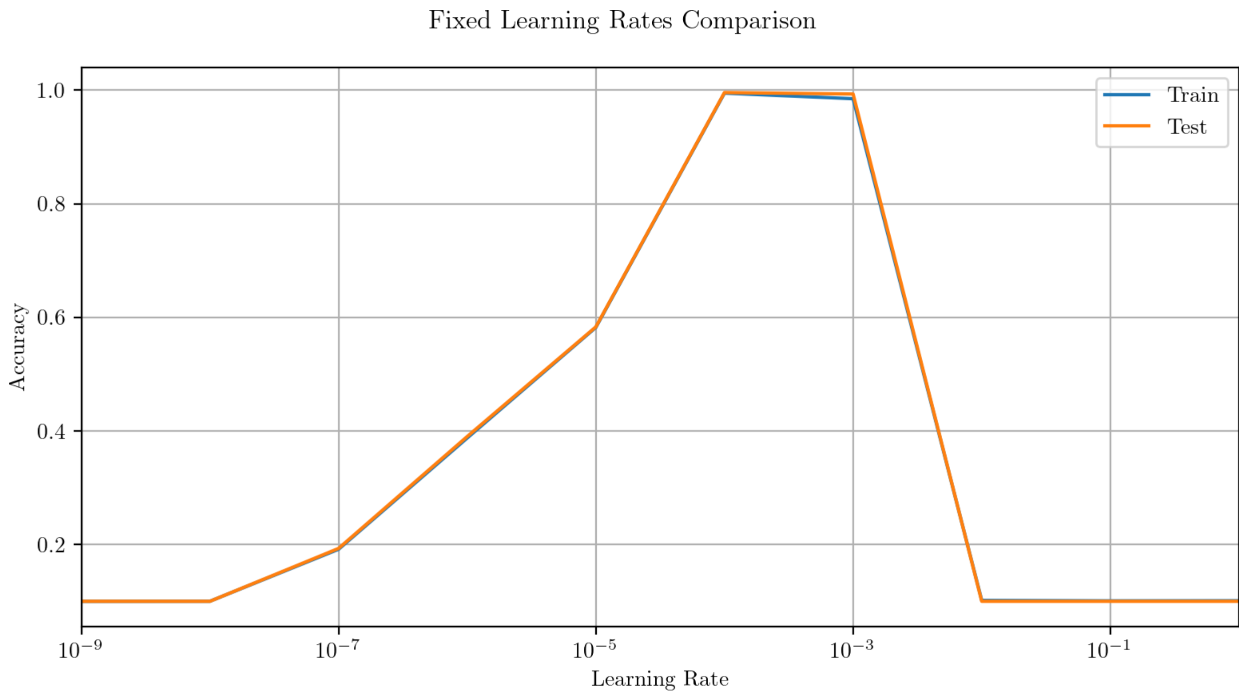



```python
# Generic CNN (https://www.kaggle.com/code/shadabhussain/cifar-10-cnn-using-pytorch)
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 64 x 16 x 16

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 128 x 8 x 8

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 256 x 4 x 4

            nn.Flatten(),
            nn.Linear(256*4*4, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 10))

    def forward(self, input):
        return self.network(input)
```

Learning Rate Scheduling Schemes

Finding the best fixed learning rate

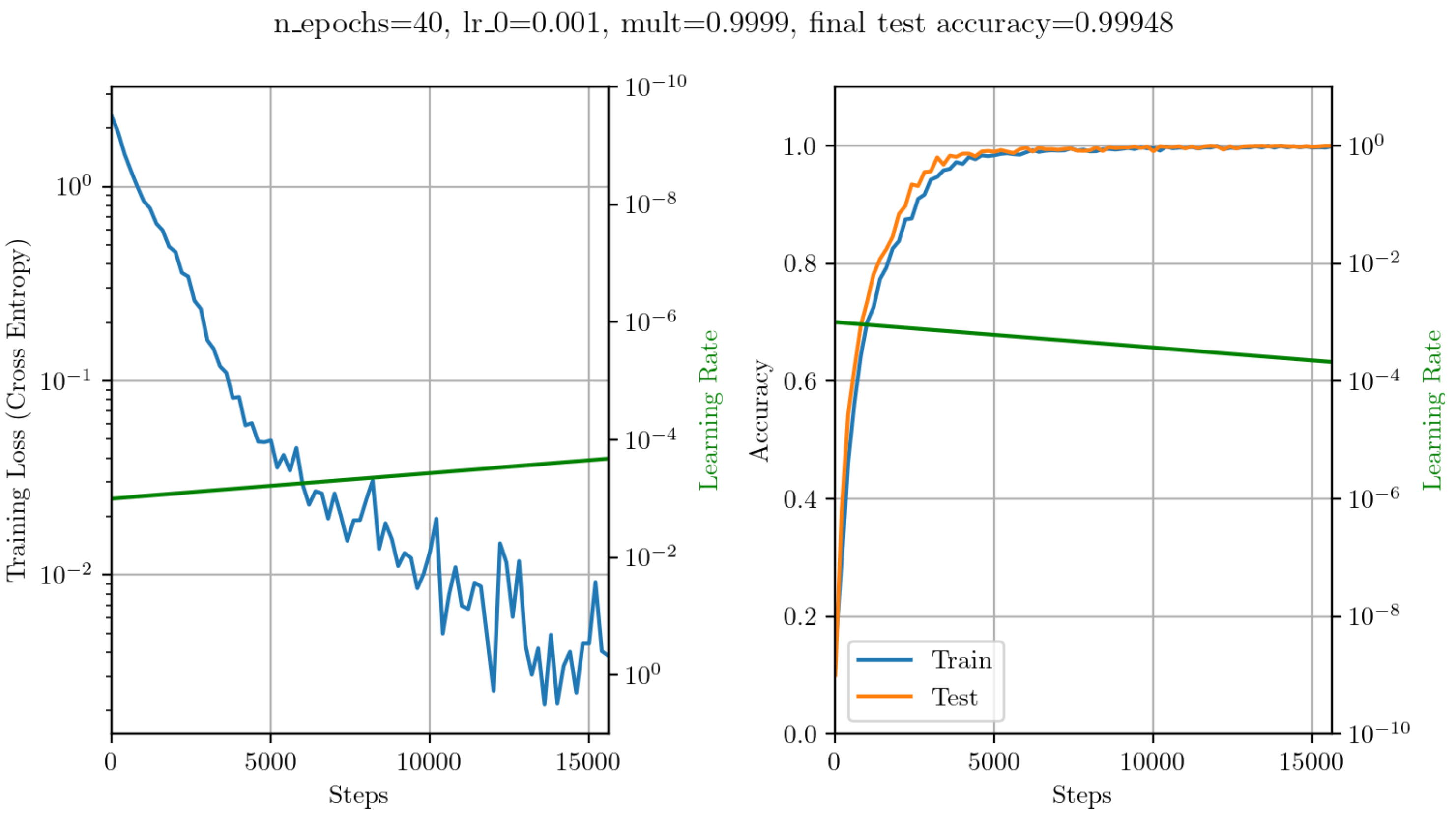
- The ***first step*** (and often only step) is usually to start with a ***fixed learning rate.***
- If the learning rate is ***too big***, the network will ***diverge***.
- If the learning rate is ***too small***, ***slow converge*** is usually the result.
- Only an ***appropriate*** learning rate will lead to ***timely convergence*** and good test metrics.



Learning Rate Scheduling Schemes

Using fixed scheduling:

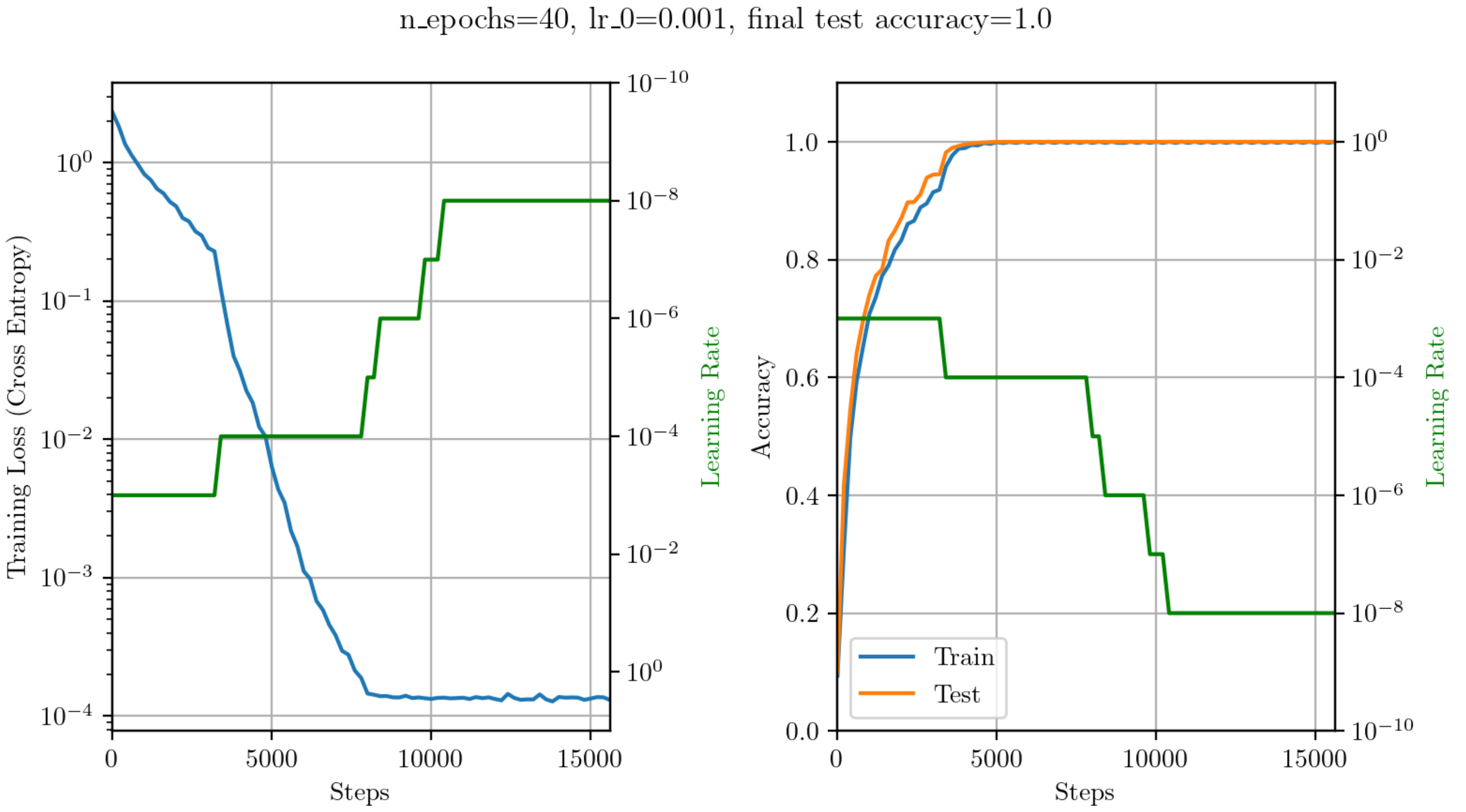
- Using ***fixed scheduling*** can help to achieve a better test metric earlier.
- Starting with the ***highest converging fixed learning rate*** and ***reducing the learning rate over time*** should lead to a higher test accuracy after the same amount of steps.
- This does, however, introduce ***new hyperparameters*** that need to get tuned.



Learning Rate Scheduling Schemes

Using adaptive scheduling:

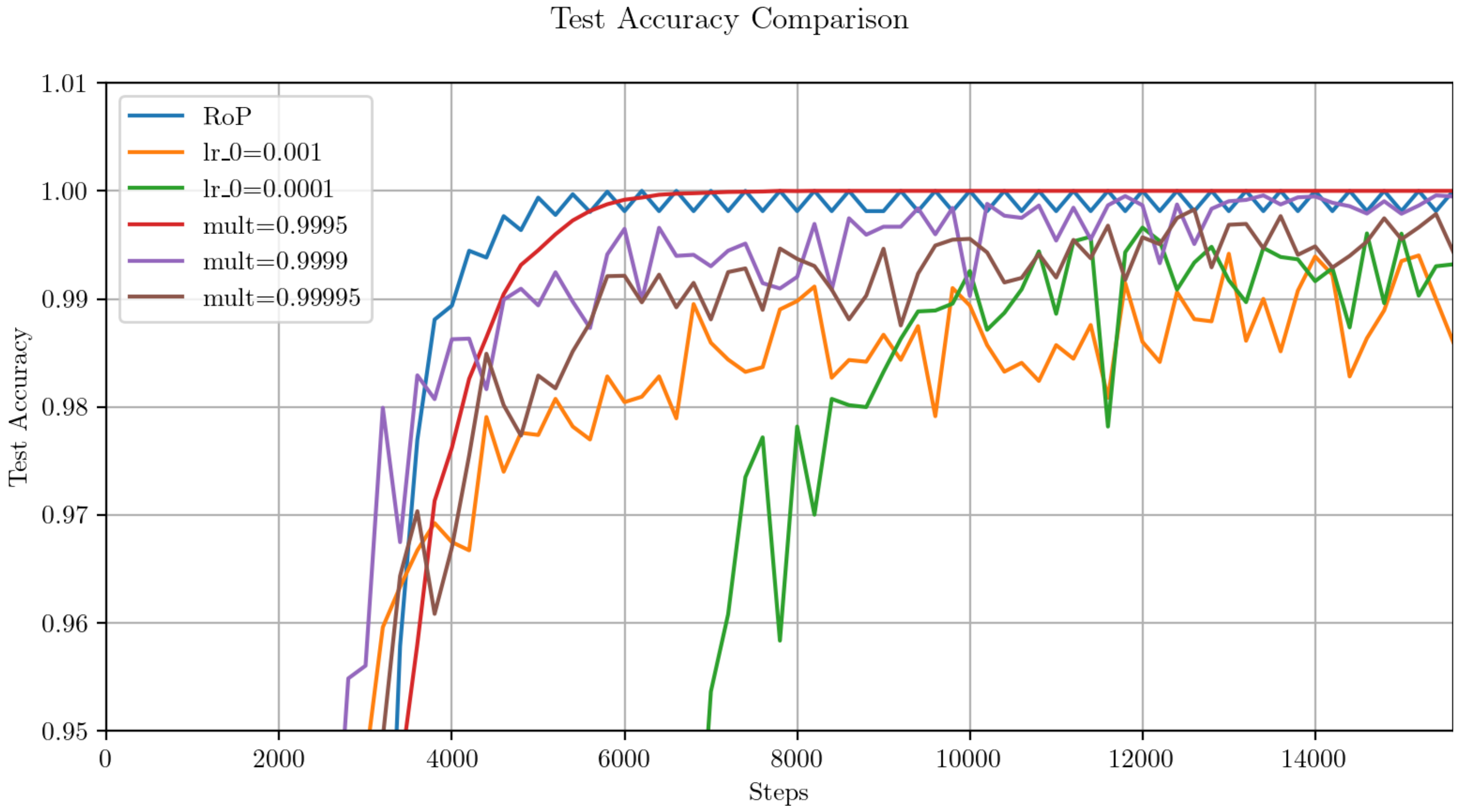
- Using ***adaptive scheduling*** can eliminate a lot of the guess work.
- Starting with the ***highest converging fixed learning rate*** and ***reducing the learning after a number of steps without improvement*** will almost always lead to better results.
- While there are still hyperparameters to tune, reducing the learning rate on the condition that the improvement of the network already stopped is more forgiving than using a fixed schedule with bad hyperparameters.



Learning Rate Scheduling Schemes

Conclusion:

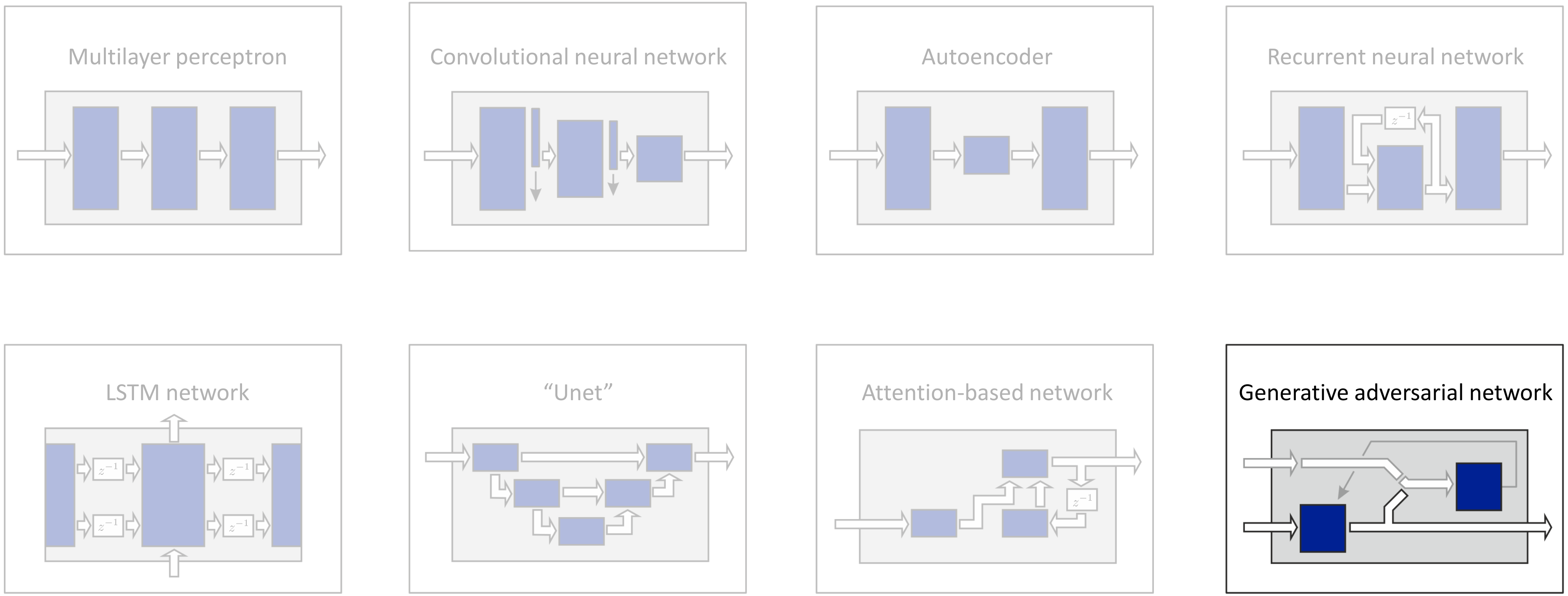
- Always ***start*** by optimizing for a ***fixed learning rate.***
- Take ***inspiration*** on what schedule people are using on ***similar problems.***
- If you have too much time and computational power, feel free to ***experiment*** with the wide variety of learning rate schedules available in common ***ML libraries*** but don't expect any miracles.



- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- ***Basic training of neural networks***
  - Backpropagation
  - Update rules
  - Learning rate scheduling
  - ***Generative adversarial networks***
- Reinforcement learning

Types of Neural Networks

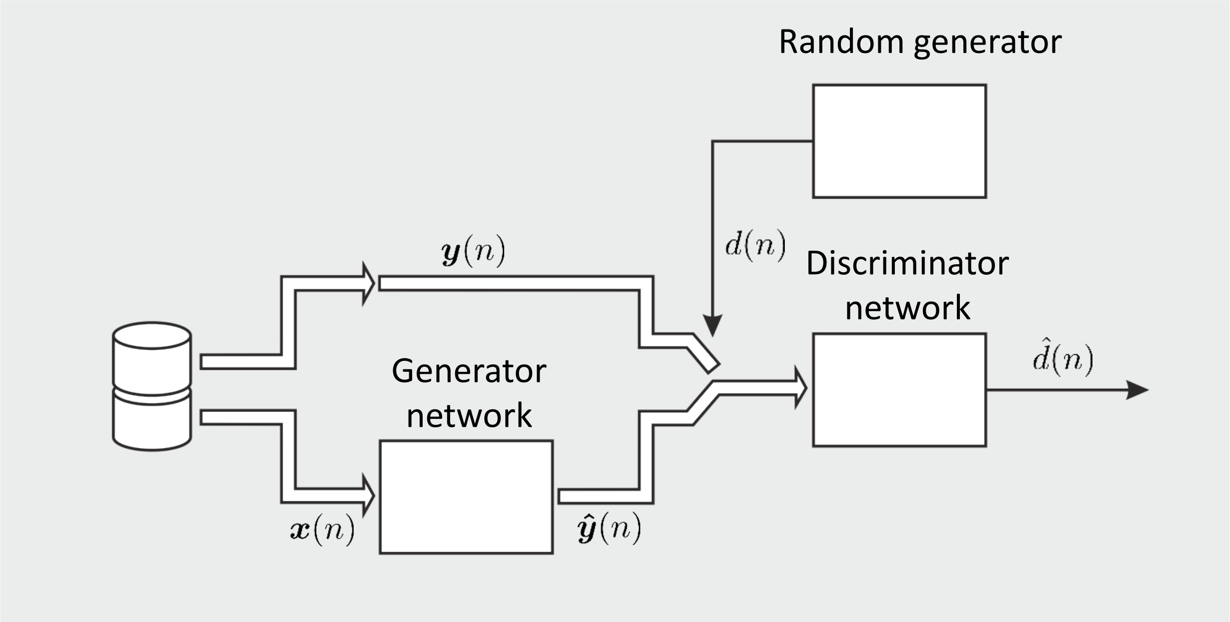
Network structure(s):



Training of Neural Networks – Generative Adversarial Networks

Basics of generative adversarial networks (GANs):

- GANs are ***not a new network type***, it's more a ***special way of training***.
- During ***runtime*** a single "standard" neural network is used. This network is called the ***generator network***.
- During ***training*** a second network is additionally used, called the ***discriminator network***.
- The job of the second network is to ***estimate***, whether the input (of the decision network) stems from ***true (desired) data or*** is the ***output of the generator network***.
- During the training the generator and the discriminator network are ***trained in an alternating fashion***.



Training of Neural Networks – Generative Adversarial Networks

Motivation of GANs:

Source: P. Isola, J.-Y. Zhu, T. Zhou, A. A. Efros: ***Image-to-Image Translation with Conditional Adversarial Networks***, CoRR, vol. abs/1611.07004, 2016.

- Example from ***image-to-image translations*** (creation of realistically looking images from label maps).
- GANs are good candidates if ***smoothed results are undesired***.
- ***Conditional GANs*** were compared to conventionally trained networks.
- ***Cost function*** is not the mean squared error (or variants of it) any more.

Input | Output of a conventionally trained network | Output of a conditional GAN | Desired output

Training of Neural Networks – Generative Adversarial Networks

Structure of the training procedure:

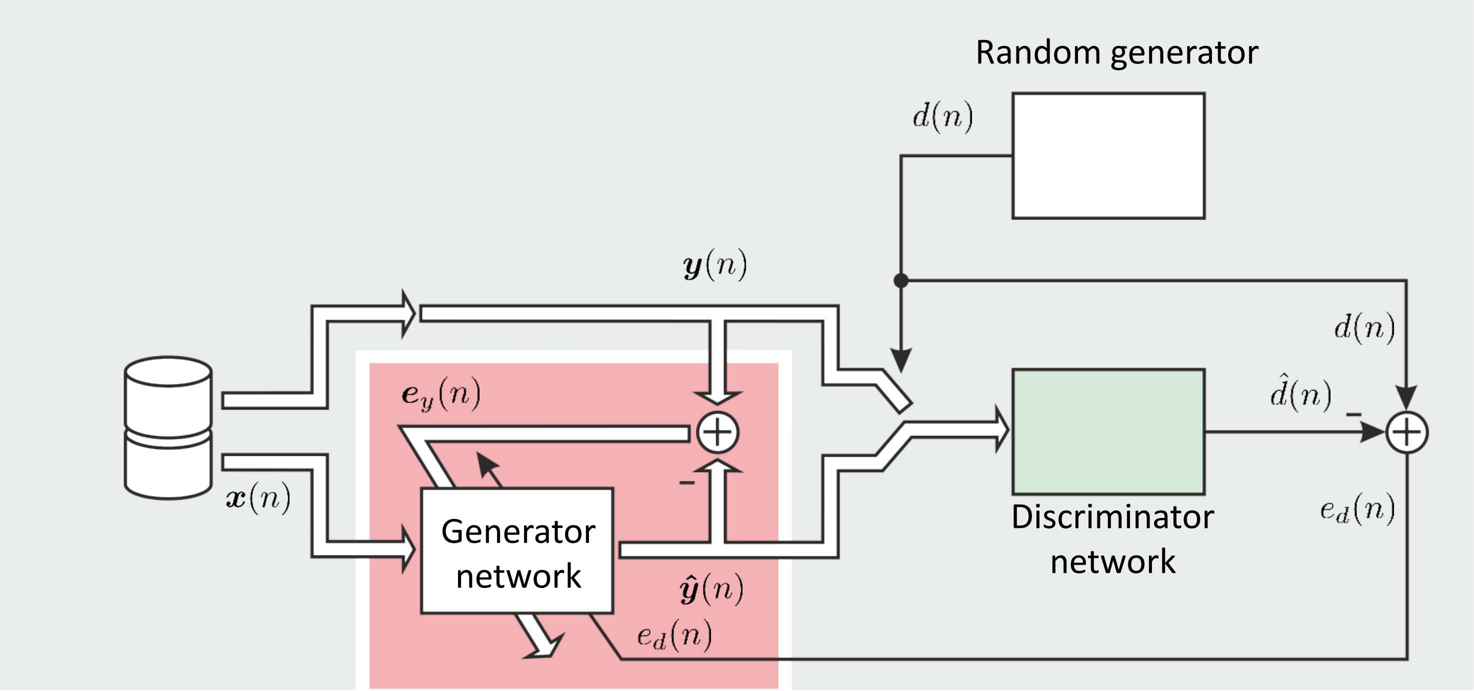
- Training of the ***generator network***:
  - The ***discriminator network*** is kept ***fixed***.
  - A weighted sum of the average ***norm of the error*** of the generator network

    $$\|\boldsymbol{e}_y(n)\|^2 = \|\boldsymbol{y}(n) - \hat{\boldsymbol{y}}(n)\|^2$$

    and the inverse of the average classification error is

    $$\frac{1}{e_d^2(n)} = \frac{1}{\big(d(n) - \hat{d}(n)\big)^2}$$

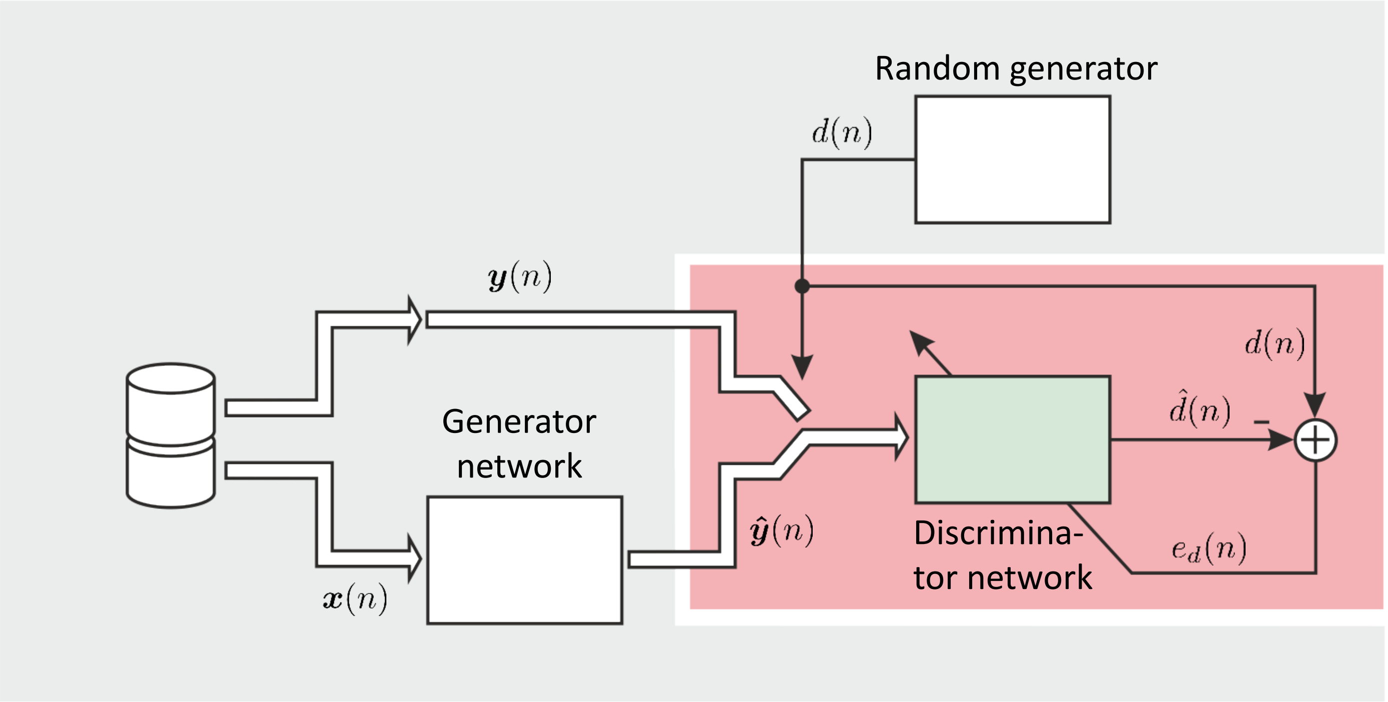
    minimized (as one variant).



Training of Neural Networks – Generative Adversarial Networks

Structure of the training procedure:

- Training of the ***discriminator network***:
  - The ***generator network*** is kept ***fixed***.
  - The average ***power of the error*** $e_d(n) = d(n) - \hat{d}(n)$ (as one variant) of the discriminator network is ***minimized***.



Training of Neural Networks – Generative Adversarial Networks

Bandwidth extension:

- For bandwidth extension GANs are also an ***interesting alternative*** (especially conditional GANs).
- The ***spectral envelope*** is estimated using ***GANs***, the ***excitation signal*** is created by ***spectral repetition*** of the narrowband excitation signal.

Source: J. Sautter. F. Faubel, M. Buck, G. Schmidt: ***Artificial Bandwidth Extension Using a Conditional Generative Adversarial Network with Discriminative Training*** , Proc. ICASSP, 2019.

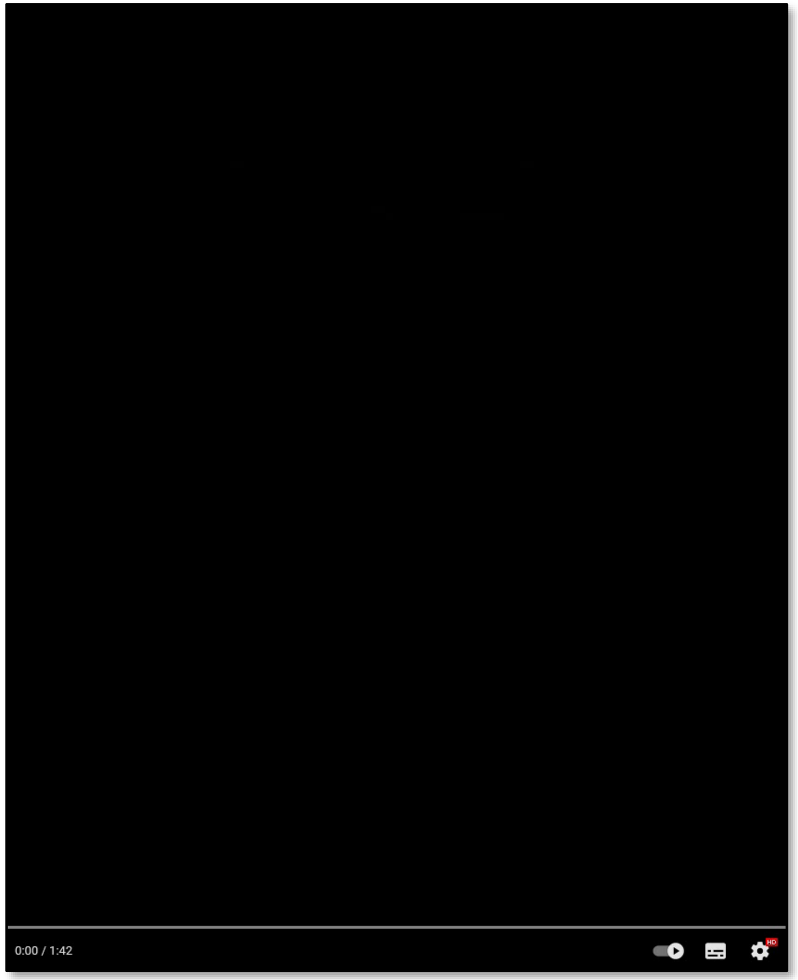
	Bandlimited input	Convent. network	Conditional GAN	Desired wide-band output
Günther Jauch				
Angela Merkel				
Christoph Waltz				
Gabriele Susanne Kerner (Nena)				


Contents

Deep Reinforcement Learning

Reinforced Learning

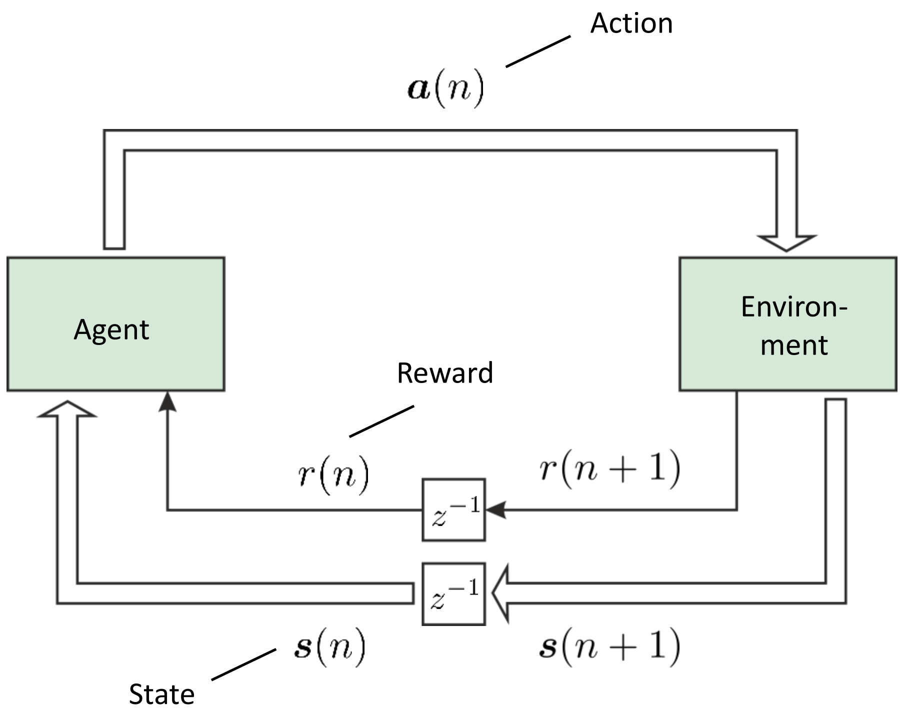
- Started with games, now also other applications are treated.



Deep Reinforcement Learning

Reinforced Learning

- Started with games, now also other applications are treated.
- A deep learning algorithm motivated by the mechanisms of (human) learning through ***reinforcement of wanted*** and ***punishment of unwanted behaviors***.
- The algorithm deploys an ***agent*** maximizing a ***reward signal*** by ***interacting with its environment*** through ***action choices***.
- The reward signal encodes the ***control goal***, rewarding action choices causing ***state transitions*** towards the ***goal state*** and punishing transitions towards unfavorable states.
- The ***feedback-loop*** of environment interactions, the returned reward signal and environment state transitions are modeled as a ***Markov decision process***.

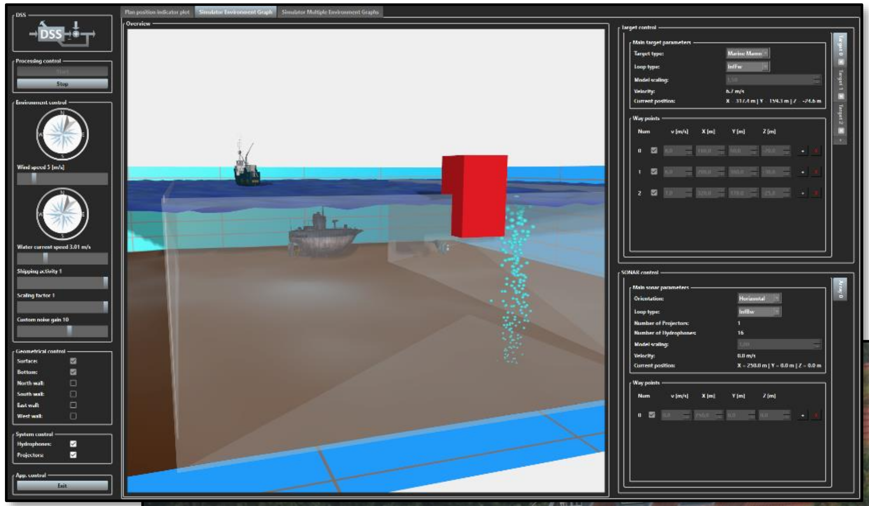


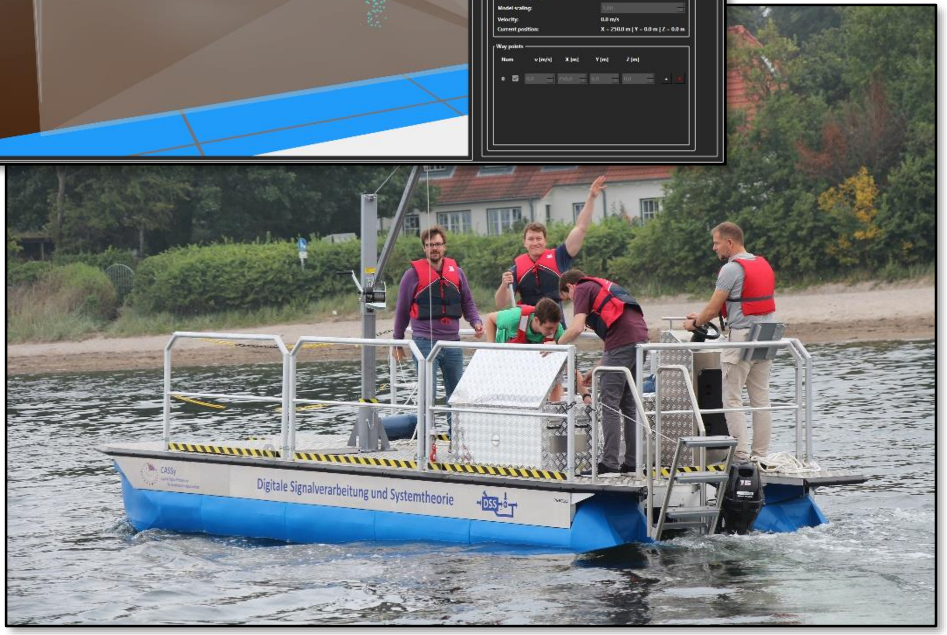
Agent-environment interaction loop

Deep Reinforcement Learning

Reinforced Learning

- Started with games, now also other applications are treated.
- A deep learning algorithm motivated by the mechanisms of (human) learning through ***reinforcement of wanted*** and ***punishment of unwanted behaviors***.
- The algorithm deploys an ***agent*** maximizing a ***reward signal*** by ***interacting with its environment*** through ***action choices***.
- The reward signal encodes the ***control goal***, rewarding action choices causing ***state transitions*** towards the ***goal state*** and punishing transitions towards unfavorable states.
- The ***feedback-loop*** of environment interactions, the returned reward signal and environment state transitions are modeled as a ***Markov decision process***.





Deep Reinforcement Learning

Agent

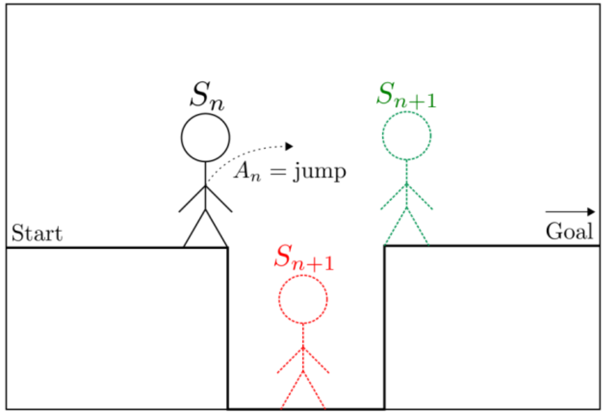
- ***Agent's task***: ***Map*** a received ***state*** observation to a corresponding ***action*** for the next environment interaction
  - Represented by a ***deep neural network*** (e.g. CNN)
- Action choice evaluation with respect to ***state-action values*** (***Q function***)
  - Expected discounted reward upon performing a specific action in a state

$$Q_\pi(\boldsymbol{s}_0,\, \boldsymbol{a}_0,\, n) \;=\; \mathrm{E}\left\{\sum_{k=0}^{N-1} \gamma^k\, r(n+k+1)\,\Big|\, \boldsymbol{s}(n) = \boldsymbol{s}_0,\, \boldsymbol{a}(n) = \boldsymbol{a}_0\right\}.$$

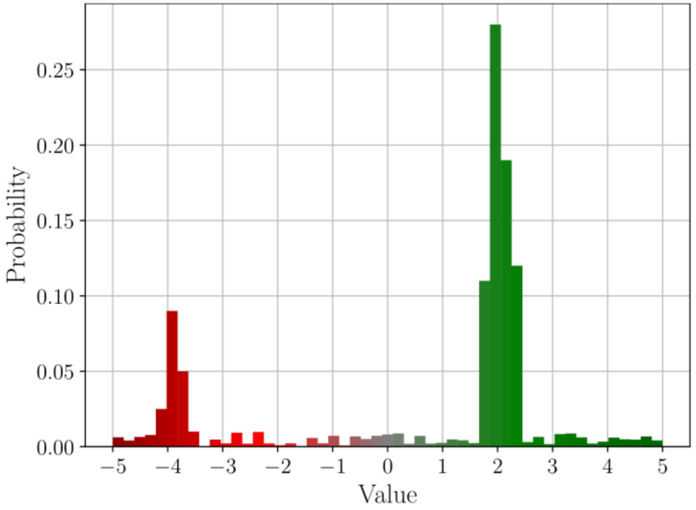
Discounted reward

- Optimal ***behavior policy*** chooses actions maximizing state-action values

$$\pi^*(\boldsymbol{s}) \;=\; \underset{\boldsymbol{a}}{\operatorname{argmax}}\left\{Q(\boldsymbol{s},\, \boldsymbol{a})\right\}.$$



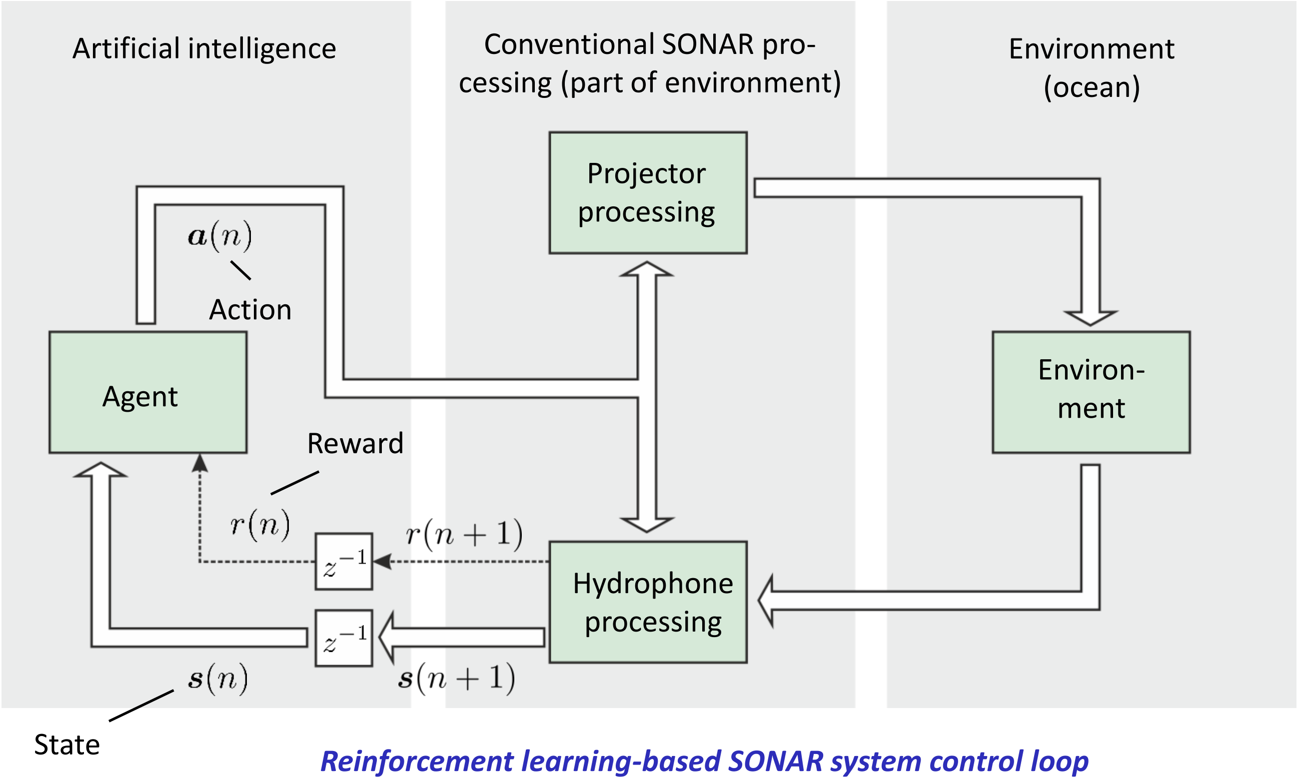
Multiple possible state transitions for same action choice



Multimodal state-action value distribution

Applications

- ***Real-time, autonomous, and robust control*** (of a system) under environmental constraints
- Able to handle ***complex parametrization state spaces***
  - Manage increasing complexity of modern systems
- ***Example***: Long-term autonomous ***parametrization control of a MIMO-SONAR system*** for monitoring or detection purposes
  - Monitoring of a port environment
  - Detection of gas bubbles in the water column
  - ***Scan parametrization adjustment*** in relation to observed environment

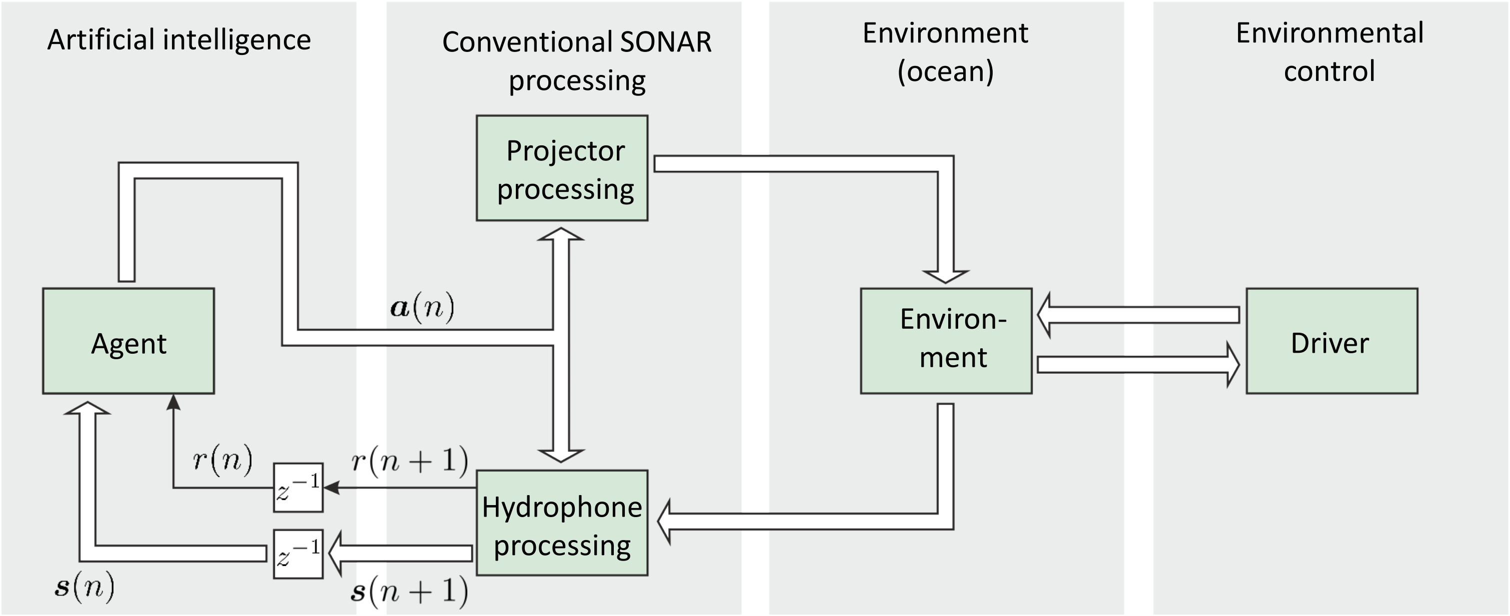


Reinforcement learning-based SONAR system control loop

Deep Reinforcement Learning

Training

- Neither supervised nor unsupervised training
- Instead: ***Dynamically generated data*** by a ***virtual training environment***
  - ***Emulates*** state ***dynamics*** and returns ***observations*** of the ***real environment***

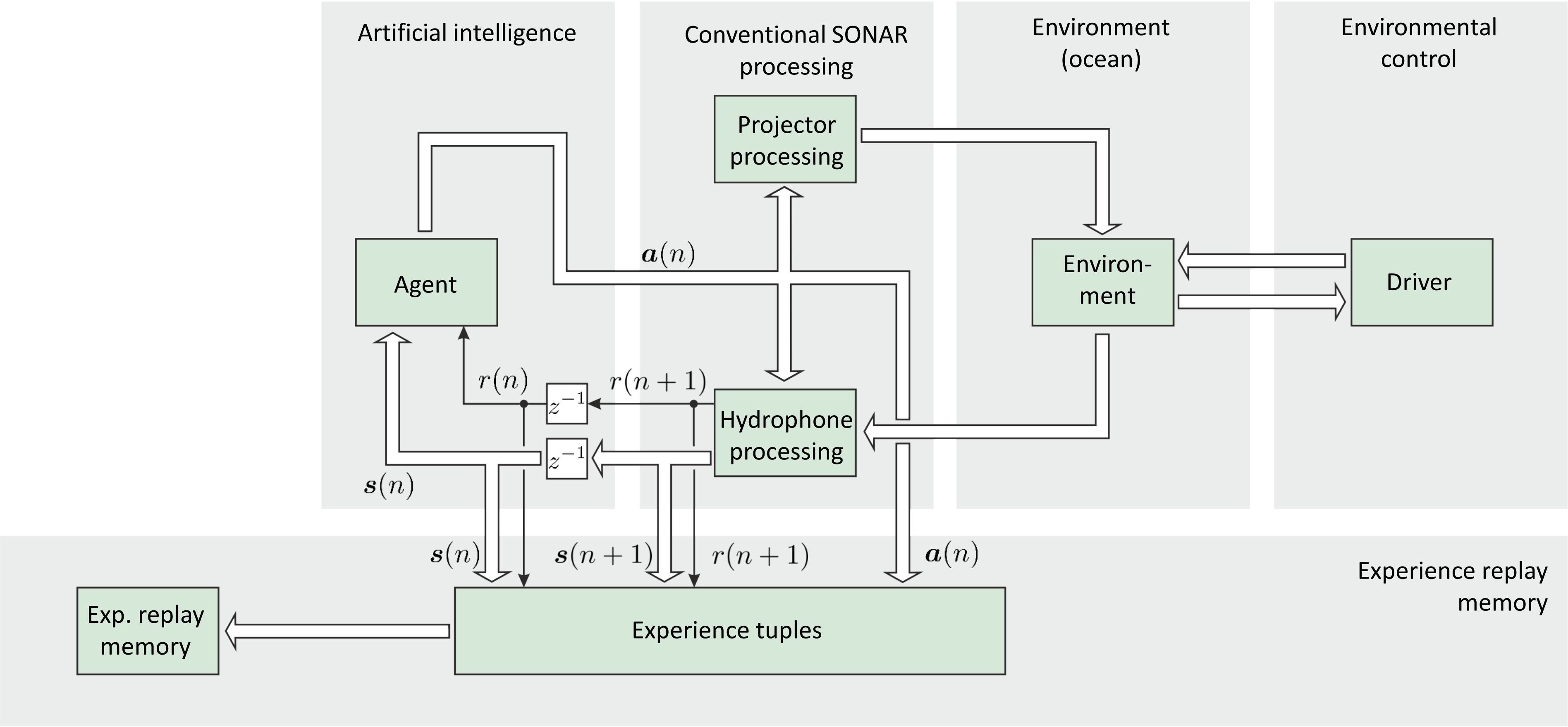


Deep reinforcement learning training architecture

Deep Reinforcement Learning

Training

- Neither supervised nor unsupervised training
- Instead: ***Dynamically generated data*** by a ***virtual training environment***
- ***Collection phase:*** Freeze agent's policy to collect action, state, and rewards transitions as ***experiences*** in a ***experience replay memory***

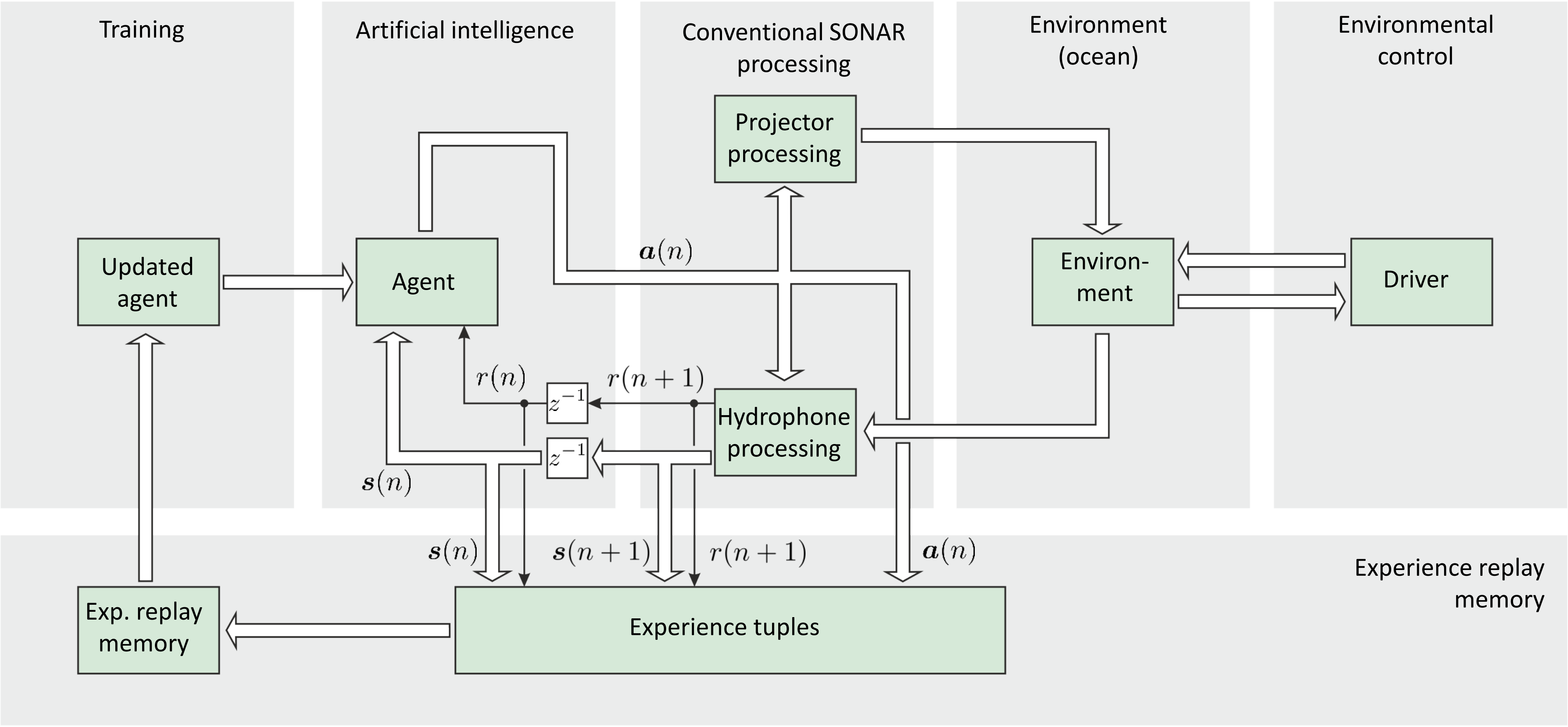


Deep reinforcement learning training architecture

Deep Reinforcement Learning

Training

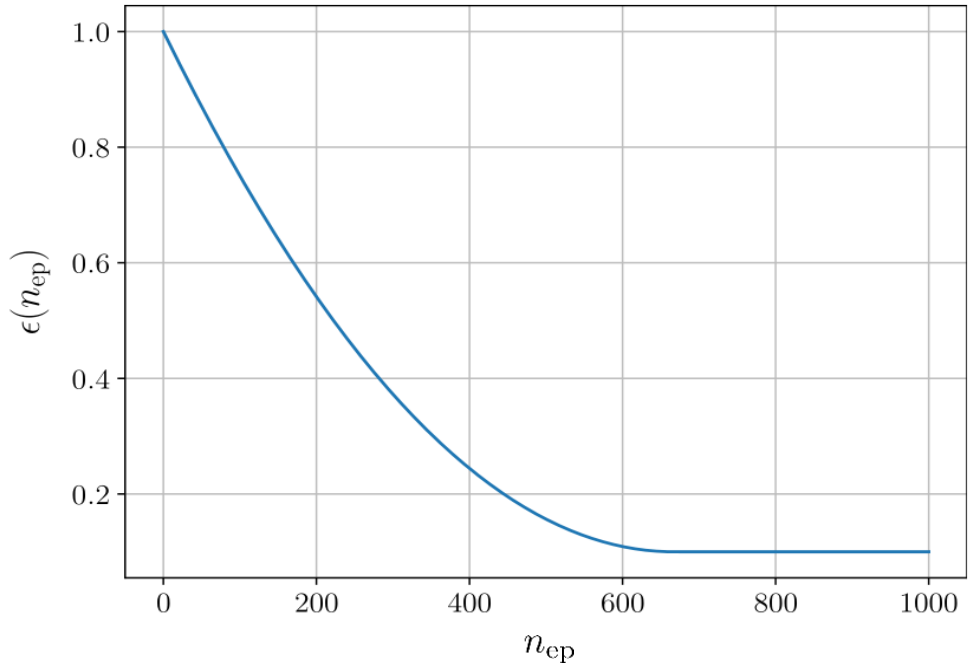
- Neither supervised nor unsupervised training
- Instead: ***Dynamically generated data*** by a ***virtual training environment***
- ***Collection phase***
- ***Training phase***
  - Resample experience replay memory to train the neural network



Deep reinforcement learning training architecture

Exploration versus Exploitation

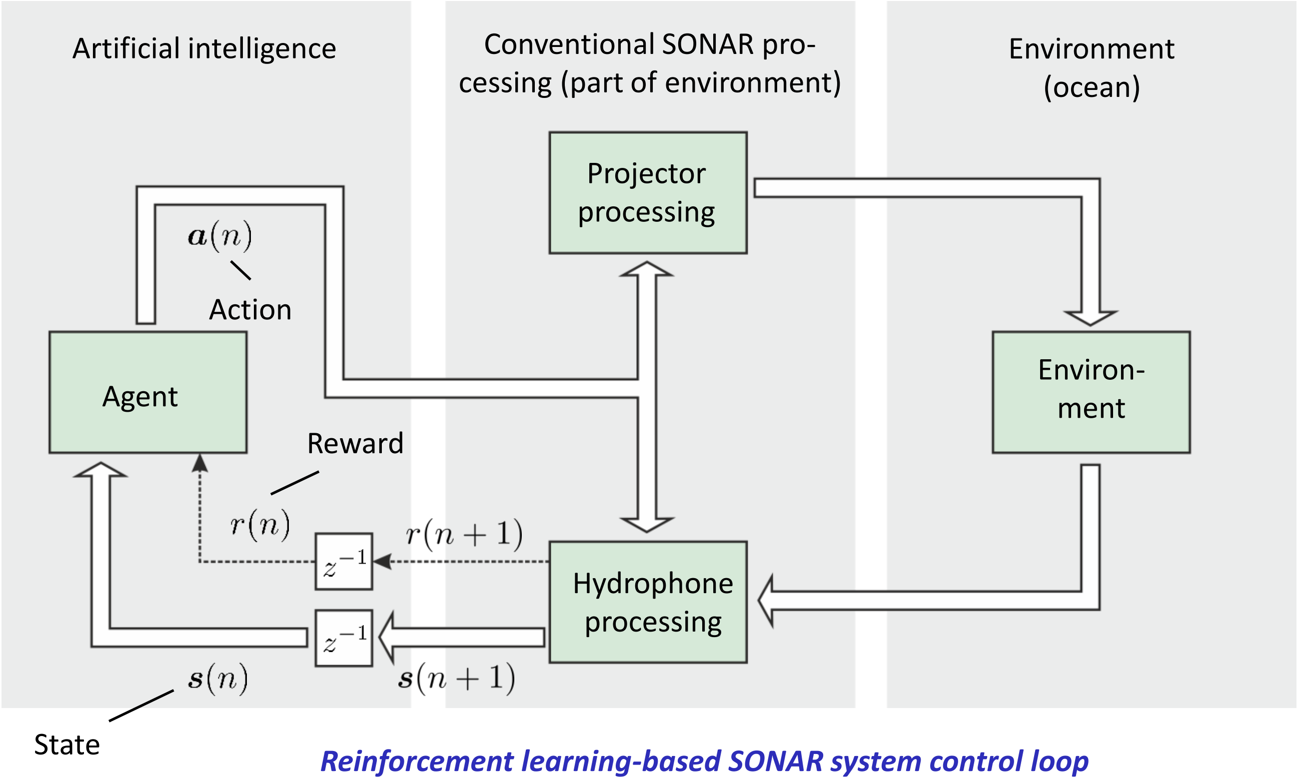
- How to set the agent's ***initial policy*** for collecting experiences?
    - No a priori environment information: ***Random initialization***
- ***Exploration rate*** ϵ
    - ***Probability*** of ***acting*** according to a ***random policy***
    - Guarantees ***random exploration of unknown environment***
    - ***Decayed*** over total training episodes
        - Transition from exploration to exploitation
- ***Exploitation*** of gathered experiences
    - ***Improve policy*** by learning which actions maximize state-action values for which environment states



Exploration rate decay over total training episodes

Applications (repeated)

- ***Real-time, autonomous, and robust control*** (of a system) under environmental constraints
- Able to handle ***complex parametrization state spaces***
  - Manage increasing complexity of modern systems
- ***Example***: Long-term autonomous ***parametrization control of a MIMO-SONAR system*** for monitoring or detection purposes
  - Monitoring of a port environment
  - Detection of gas bubbles in the water column
  - ***Scan parametrization adjustment*** in relation to observed environment

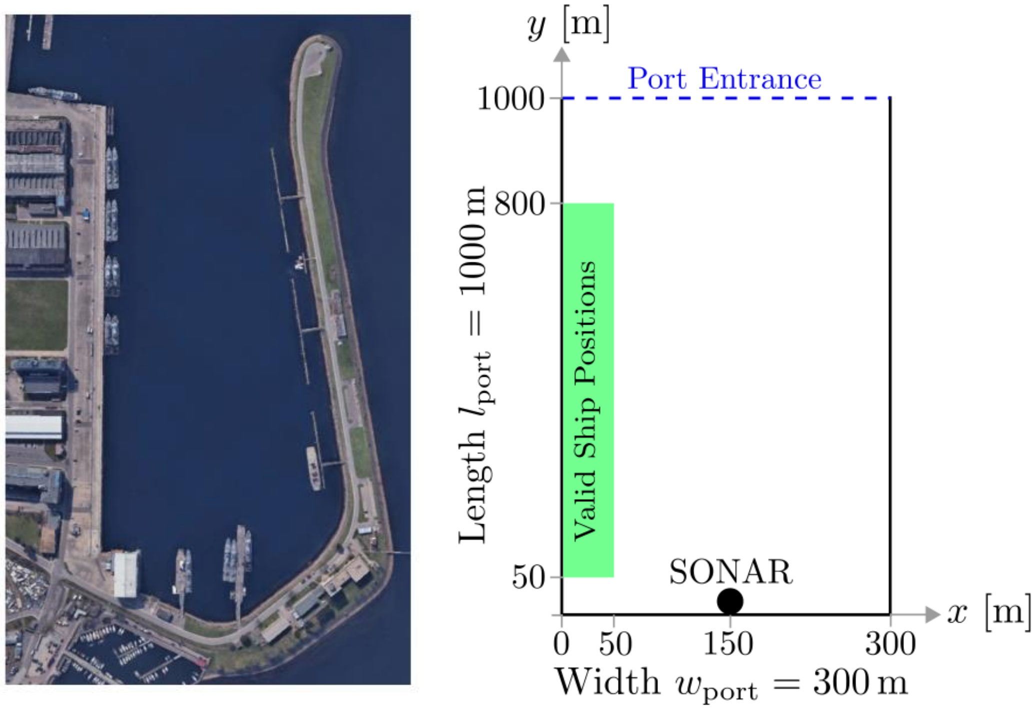


Reinforcement learning-based SONAR system control loop

Deep Reinforcement Learning

Example: SONAR Port Monitoring

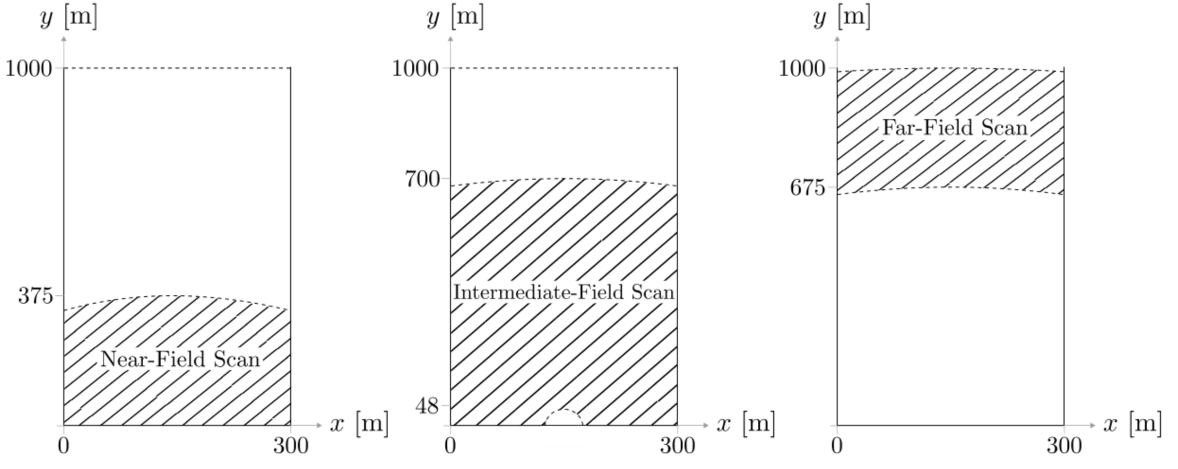
- ***Port environment*** with ships stationed inside
  - ***Monitoring*** for potential intruders trying to damage a ship
  - SONAR system inside the port is able to ***scan different areas*** of the port by ***utilizing different scan modes***
- ***Virtual training environment*** models real port environment
  - Simulated acoustic targets & SONAR scan observations
- ***Scan modes*** differ in their system parametrization
  - Signal- and ping durations
  - Transmit power
  - Transmit and receive configuration
    - SIMO, MISO, MIMO
  - Beamforming operation



WTD marine arsenal as port environment model

Example: SONAR Port Monitoring

- ***Port environment*** with ships stationed inside
  - ***Monitoring*** for potential intruders trying to damage a ship
  - SONAR system inside the port is able to ***scan different areas*** of the port by ***utilizing different scan modes***
- ***Virtual training environment*** models real port environment
  - Simulated acoustic targets & SONAR scan observations
- ***Scan modes*** differ in their system parametrization
  - Signal- and ping durations
  - Transmit power
  - Transmit and receive configuration
    - SIMO, MISO, MIMO
  - Beamforming operation

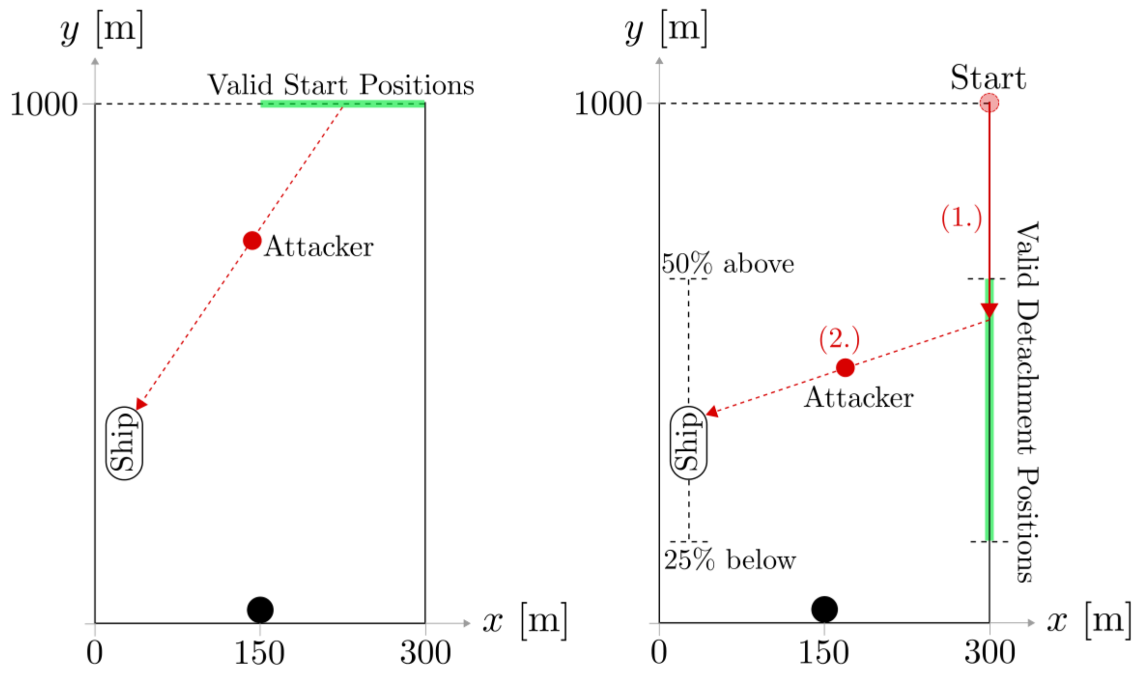


SONAR scan modes

Deep Reinforcement Learning

Agent's Goal

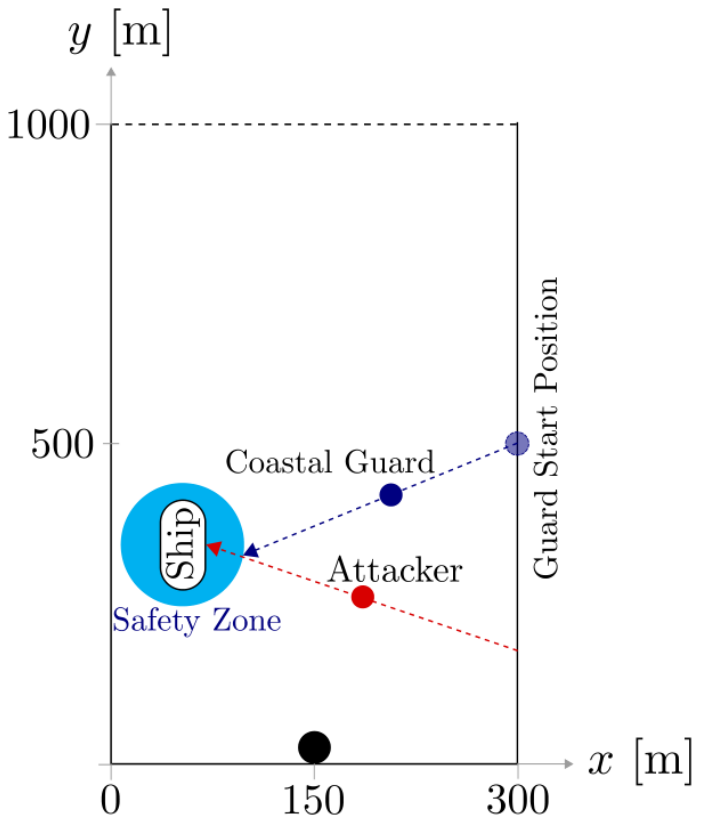
- ***Reliable detection*** of potential attackers through ***proper choice of the SONAR scans*** to be performed
  - Ensonify the attacker's area of location
  - ***Scan modes*** represent the agent's ***action space***
- ***Coastal guards*** are sent out for ***interception*** if a potential attacker is assumed to be present
  - ***Detect as fast as possible*** to avoid potential harm
  - ***Avoid*** unnecessary ***false alarms***
- Deployment on mobile platforms with ***limited energy resources***
  - ***Save energy*** through standby mode if the current risk is low



Attacker following different stategies to reach target ship

Agent‘s Goal

- ***Reliable detection*** of potential attackers through ***proper choice of the SONAR scans*** to be performed
  - Ensonify the attacker‘s area of location
  - ***Scan modes*** represent the agent‘s ***action space***
- ***Coastal guards*** are sent out for ***interception*** if a potential attacker is assumed to be present
  - ***Detect as fast as possible*** to avoid potential harm
  - ***Avoid*** unnecessary ***false alarms***
- Deployment on mobile platforms with ***limited energy resources***
  - ***Save energy*** through standby mode if the current risk is low



Interception of attacker through coastal guards

Reward Function

- Designed according to given goals
  - ***Physically motivated costs*** for performing scans
  - ***Safety costs*** for reliable and fast detections
  - ***Monetary costs*** for coastal guard alarms
- ***Reinforce wanted behavior*** (pos. reward)
  - Saving energy (performing no scan)
  - Enabling successful coastal guard interception
    - Reliable & fast detection
- ***Punish unwanted behavior*** (neg. reward)
  - Slow/missed detections
  - False alarms
    - Interferes with normal port routines and operation costs money
  - Wasting energy (unecessary scans)

Tx power and scaling · *Num. proj.* · *Signal duration*

$$\tilde{r}_{\text{cost}}(A_n) = \begin{cases} \underbrace{-P \cdot P_{\text{scale}}}_{\text{Tx power and scaling}} \cdot \underbrace{N_{T_x}}_{\text{Num. proj.}} \cdot \underbrace{t_{\text{sig}}(A_n)}_{\text{Signal duration}}, & A_n \in 0, 1, 2 \\ \frac{1}{2} \cdot \min\left[P \cdot P_{\text{scale}} \cdot N_{T_x} \cdot t_{\text{sig}}(A_n)\right], & A_n = 3 \end{cases}$$

$$r_{\text{cost}}(A_n) = \frac{\tilde{r}_{\text{cost}}(A_n)}{\max\left[|\tilde{r}_{\text{cost}}(A_n)|\right] \cdot 1\,\text{Ws}} \cdot r_{\text{norm}}$$

Action	Cost
Near-field	$-1.1 \cdot 10^{-3}$
Intermediate-field	$-1.4 \cdot 10^{-3}$
Far-field	$-1.0 \cdot 10^{-2}$
No scan	$+5.5 \cdot 10^{-4}$

Reward Function

- Designed according to given goals
  - ***Physically motivated costs*** for performing scans
  - ***Safety costs*** for reliable and fast detections
  - ***Monetary costs*** for coastal guard alarms
- ***Reinforce wanted behavior*** (pos. reward)
  - Saving energy (performing no scan)
  - Enabling successful coastal guard interception
    - Reliable & fast detection
- ***Punish unwanted behavior*** (neg. reward)
  - Slow/missed detections
  - False alarms
    - Interferes with normal port routines and operation costs money
  - Wasting energy (unecessary scans)

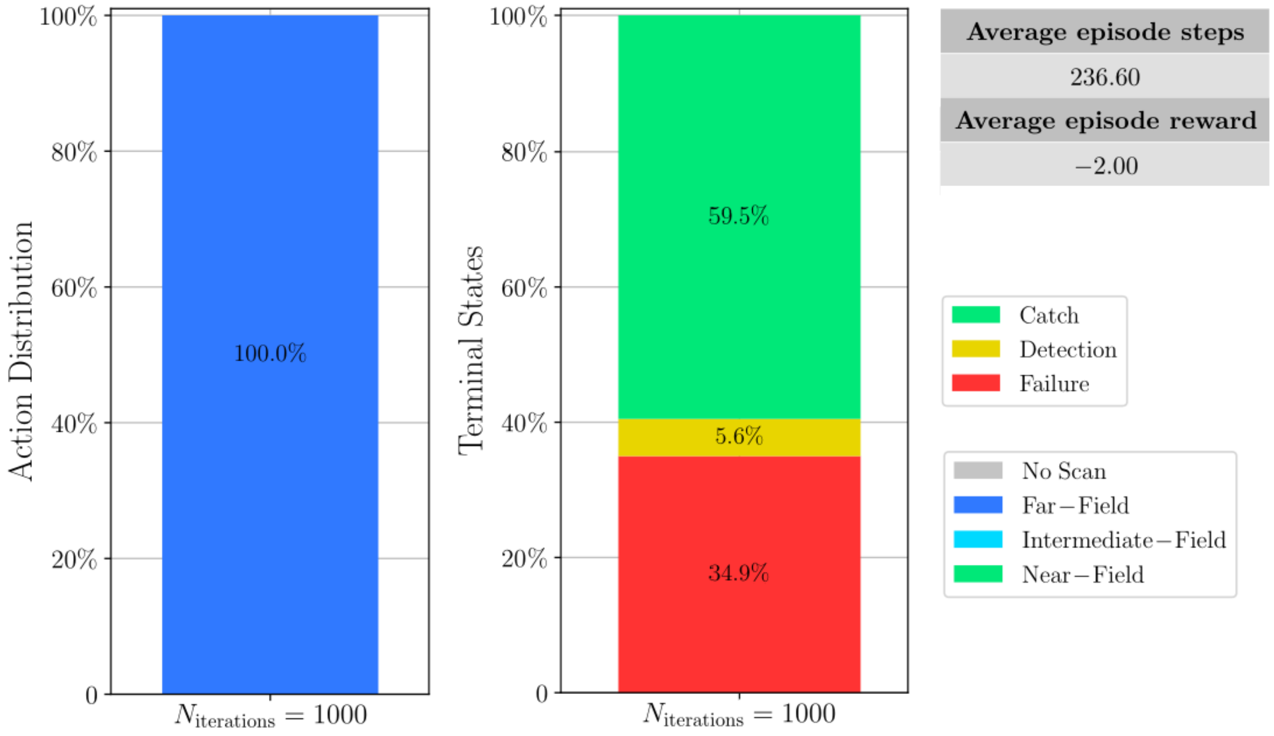
Combination of multiple reward and cost terms

$$\begin{aligned} r_{\text{sum}}(n) = {} & r_{\text{cost}}(A_n) + r_{\text{history}}(n) + r_{\text{time}}(n) + \\ & r_{\text{detect}}(n) + r_{\text{lost}}(n) + r_{\text{catch}}(n) + r_{\text{failure}}(n) \end{aligned}$$

Deep Reinforcement Learning

Training Results

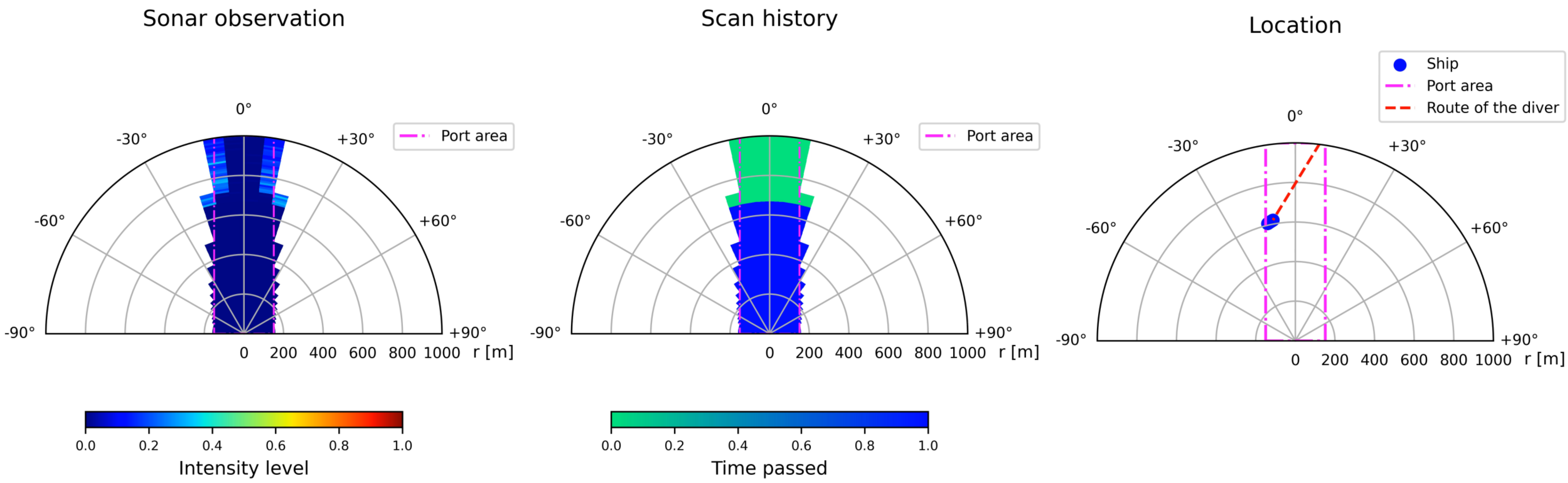
- Agent's learned ***policy depends on total training iterations***
  - Longer training enables agent to ***fully explore the environment*** and experience multiple scenarios
  - ***Strategy improves*** over time by exploitation of gathered attack scenarios & outcome knowledge
- ***1000 training iterations***
  - Only ***basic strategy*** of scanning the far-field is learned
  - Agent always monitors the port entrance
    - Assumes attacker to enter there
    - Misses attackers hiding in wall reflections!



Action choices and evaluation statistics

Deep Reinforcement Learning

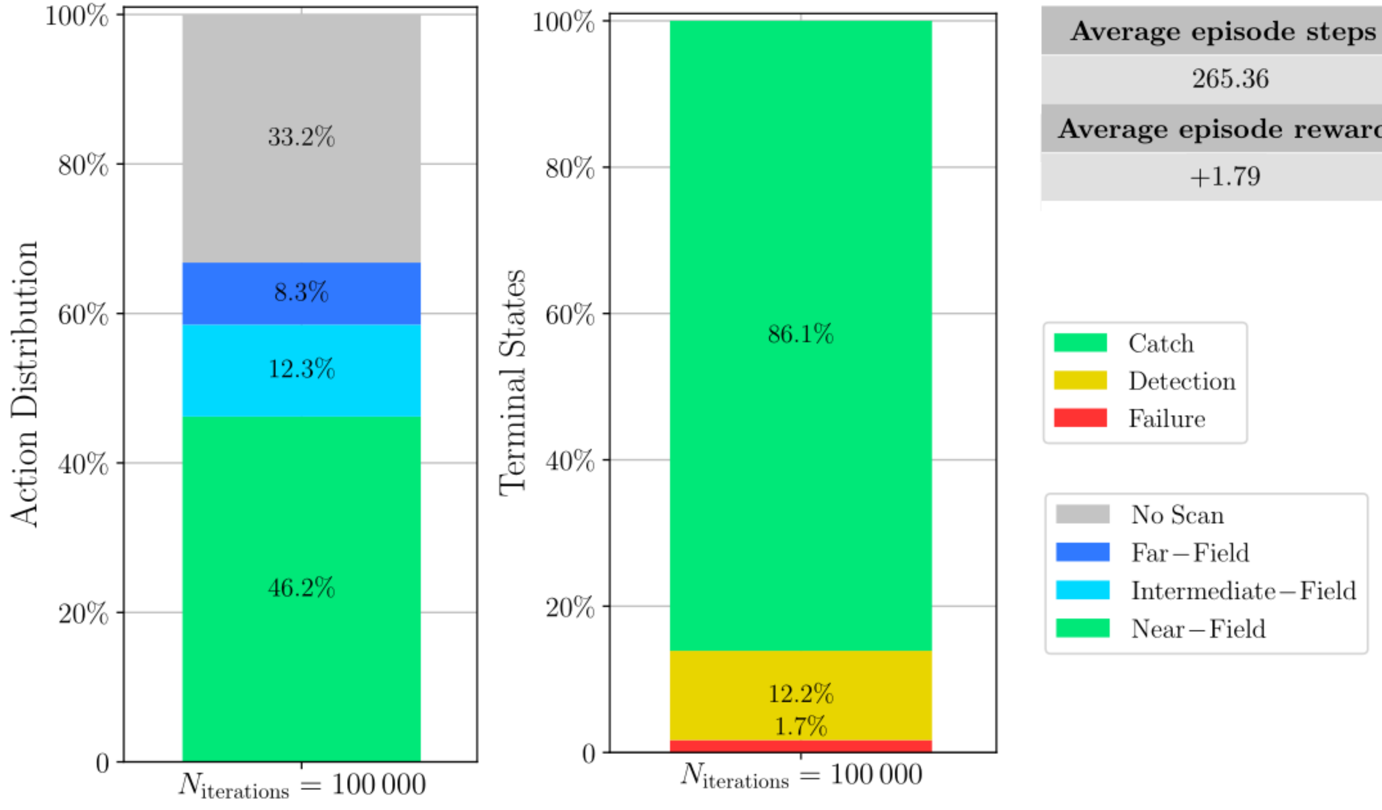
Basic Monitoring Strategy



Deep Reinforcement Learning

Training Results

- ***100.000 training iterations***
  - Utilization of all scan modes
    - ***Improved detection rate***
  - Use of standby mode for low risk situations
    - ***Improved energy consumption***
  - Agent ***learned reliable detection strategy***

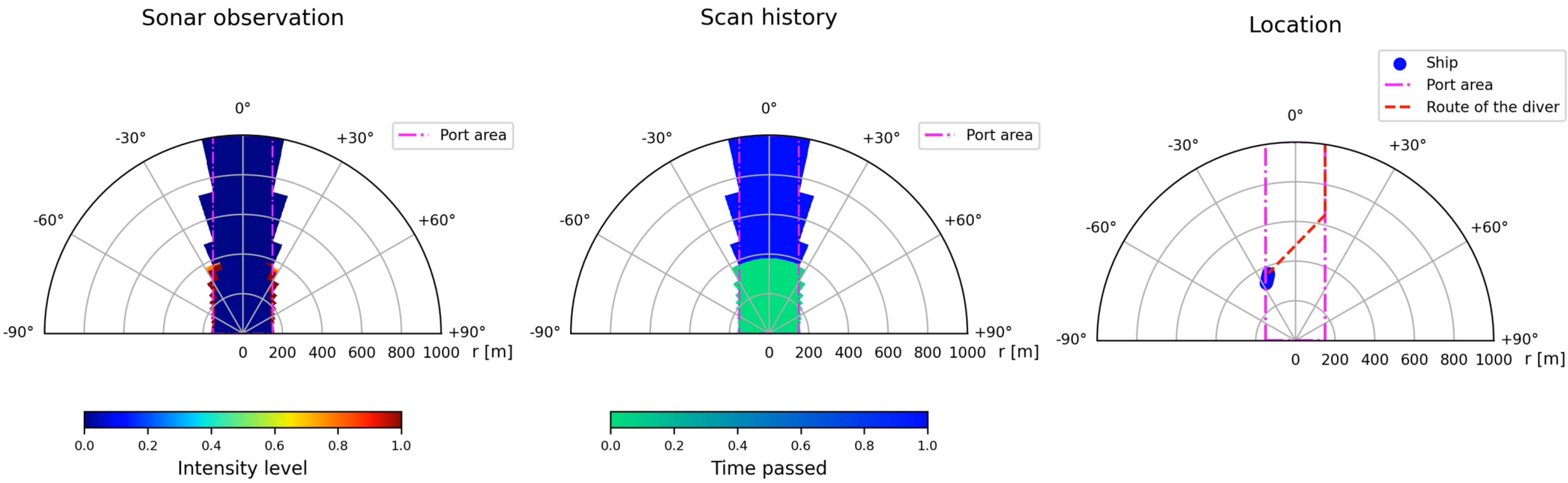


Average episode steps
265.36
Average episode reward
+1.79

Action choices and evaluation statistics

Deep Reinforcement Learning

Advanced Monitoring Strategy



Literature:

- R. S. Sutton, A. G. Barto: ***Reinforcement learning: An introduction***, MIT press, 2018
- V. Mnih et al: ***Playing Atari with deep reinforcement learning***, arXiv:1312.5602, 2013
- M.G. Bellemare et al: ***A distributional perspective on reinforcement learning***, International Conference on Machine Learning, PMLR, 2017

Summary and Outlook

Summary:

- Motivation
- Structure of a (basic) neural network
- Applications of neural networks
- Types of neural networks
- Basic training of neural networks

Next part:

- Hidden Markov Models (HMMs)