Introduction to Deep Learning

Guest Lectures at TTK University of Applied Sciences, Tallinn, Estonia

8 – 12 December 2025

Andy Louwyck & Dominique Stove

Vives University of Applied Sciences, Kortrijk, Belgium

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Who’s Teaching Today?

Andy Louwyck

- Master and Doctor in Science: Geology
- Associate Degree in Programming
- Micro Degree in AI & Data Science
- Lecturer in IT at Vives University of Applied Sciences
- AI Coordinator at Flanders Environment Agency

Dominique Stove

- Master in Applied Economics
- Teaching Master’s Degree in Economics, Mathematics, and Physics
- Lecturer in IT at Vives University of Applied Sciences
- IT Consultant in Infrastructure
- Founder and Business Owner of IqPro

vives University of Applied Sciences

Vives Campus in the City of Kortrijk

Informatics Program for Exchange Students

https://www.vives.be/en/commercial-sciences-business-management-and-informatics/informatics-kortrijk

The Informatics-programme is a programme consisting of lectures, group work, visits and projects in the field of Business and Informatics. Evaluation follows the rules of the European Credit Transfer System (ECTS). Incoming students can select a programme of up to 30 ECTS credits per semester.

New full-year program!

Title	ECTS	hours/week S1	hours/week S2	Semester
Introduction to Artificial Intellingence	5	3	0	1
Programming in Python	3	2	0	1
Digital Workplace	3	2	0	1
Android App Development	5	3	0	1
E-business en E-marketing	3	2	0	1
Introduction to linux	3	2	0	1
Cybersecurity	5	3	0	1
Professional and International Communication 3 (English)	3	2	0	1
	30	**19**	0	
Machine Learning - Fundamentals	6	0	4	2
IT-Project	5	0	3	2
Power Tools	3	0	3	2
Full-Stack Development in .NET	6	0	4	2
Mobile App Development iOS	5	0	4	2
Data Engineering	5	0	3	2
Node.js Development	3	0	2	2
	33	0	23	

Guest Lectures at TTK University of Applied Sciences, Tallinn, Estonia

Introduction to Deep Learning

DEEP LEARNING: OVERVIEW

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Deep Learning & Artificial Intelligence

- **Artificial Intelligence** (AI):

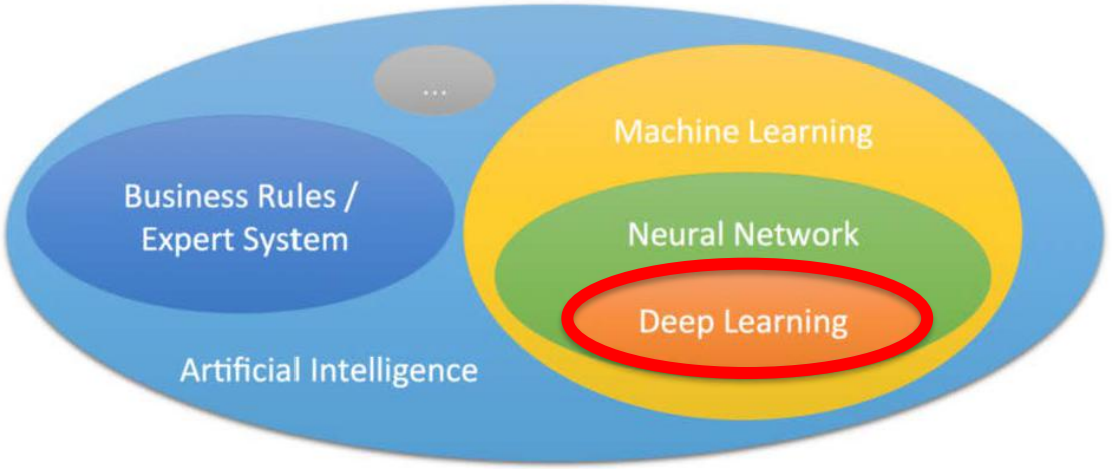
  "The set of all tasks in which a computer can make decisions."

- **Machine Learning** (ML):

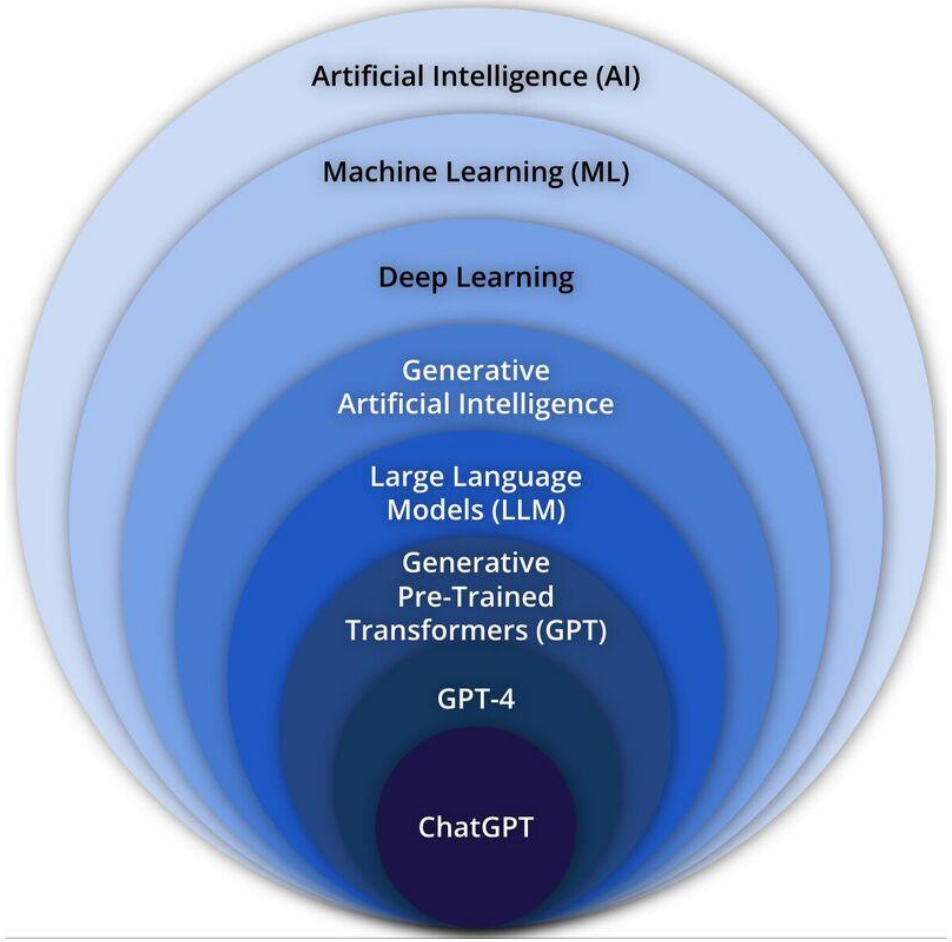
  "The set of all tasks in which a computer can make decisions *based on data*."

- **Deep Learning** (DL):

  "The field of machine learning that uses certain objects called *neural networks*."



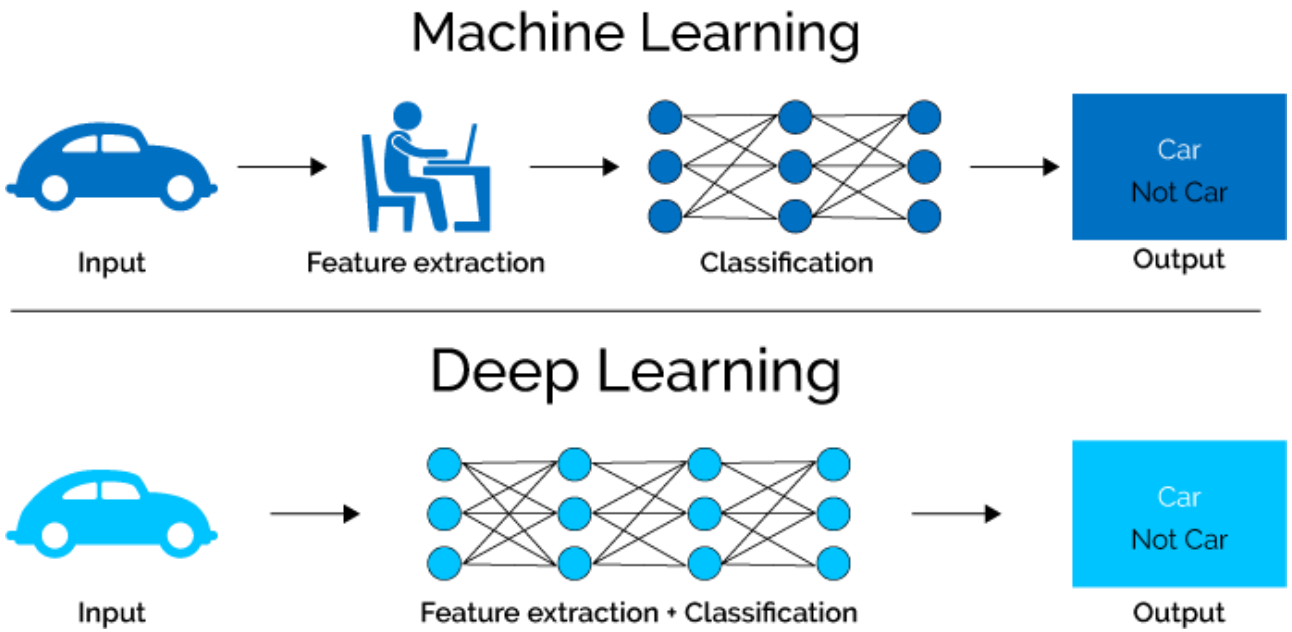
What about ChatGPT?



https://www.linkedin.com/posts/edoardoquercidellarovere_spesso-mi-capita-di-leggere-e-sentire-conversazioni-activity-7126095949738721280-amQI/

What is deep learning?

- Subfield of ML for **learning representations** of data
- Exceptional effective at **learning patterns**
- Utilizes learning algorithms that derive **meaning** out of data by using a **hierarchy of multiple layers**
- If you provide the system **large amounts of training data**, it begins to understand it and respond in useful ways
- It can learn **both unsupervised and supervised**
- It is highly suitable for **parallelization** on GPUs



https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

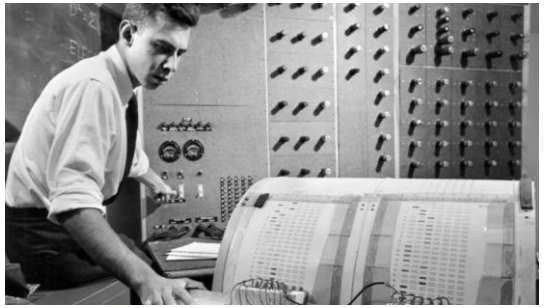
Deep learning: use cases

- Computer vision (CV)
- Face recognition
- Natural language processing (NLP)
- Speech recognition
- Generative AI
- Self-driving cars
- …

- Even in your mobile device
  - Siri
  - Google assistant

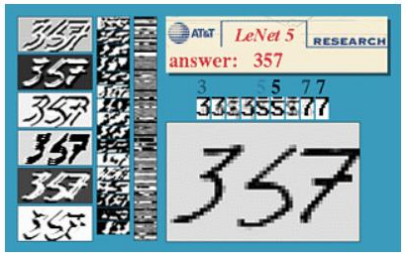
Deep learning: brief history

Frank Rosenblatt



1958 Perceptron

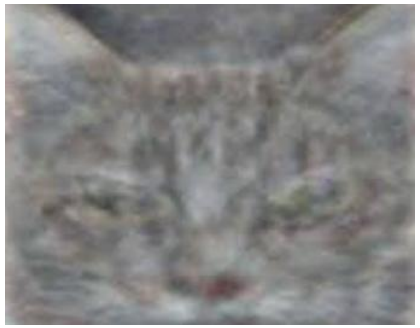
1974 Backpropagation



Yann LeCun

Convolution Neural Networks for Handwritten Recognition

1998



Google Brain Project on 16k Cores

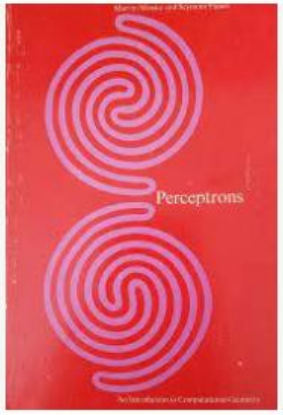
2012

I am ChatGPT, created by OpenAI.

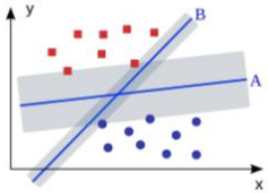
2022

awkward silence (AI Winter)

1969
Perceptron criticized



1995
SVM reigns



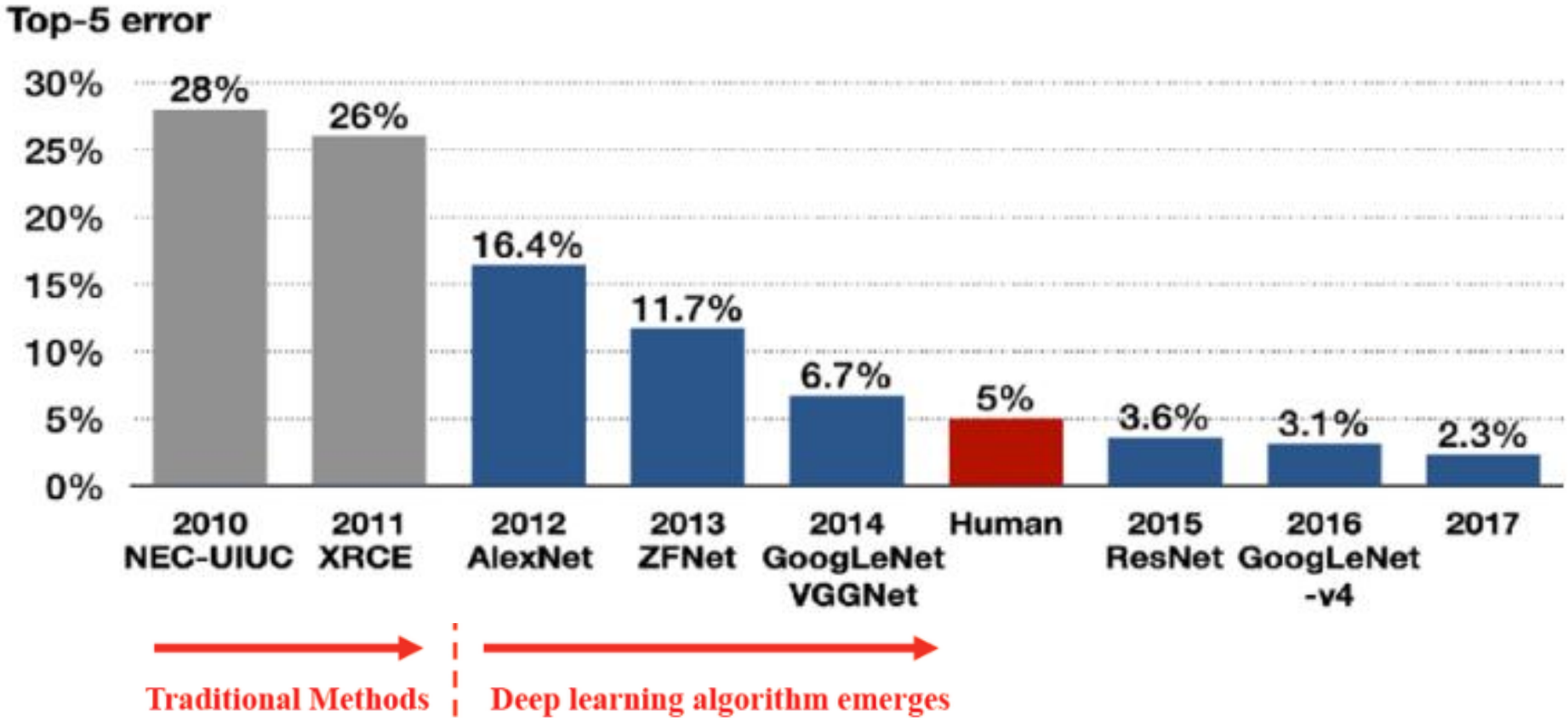
2006
Restricted Boltzmann Machine

Geoffrey Hinton

2012
AlexNet wins ImageNet

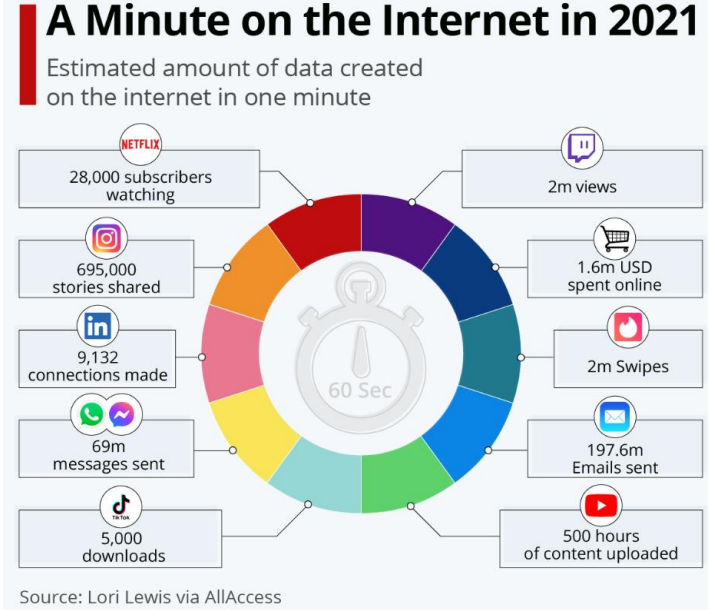
IMAGENET

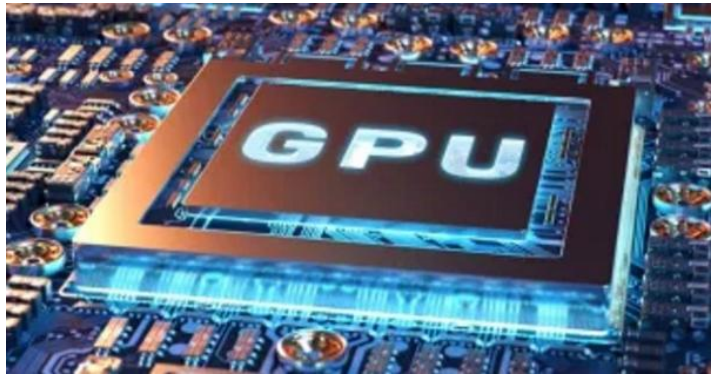
The ImageNet competition



Why did deep learning only take off after 2012?

- Deep learning algorithms already known in the 1990s
- Yet a breakthrough only occurred in 2012!
  - Bottleneck in the 1990s and 2000s: **data and hardware**
  - Much more data available through the internet
  - Gaming industry developed better hardware, e.g. GPUs





The democratization of deep learning

- Deep learning initially required knowledge of C/C++ and CUDA
- Now there are many accessible high-level (Python) libraries

Guest Lectures at TTK University of Applied Sciences, Tallinn, Estonia

Introduction to Deep Learning

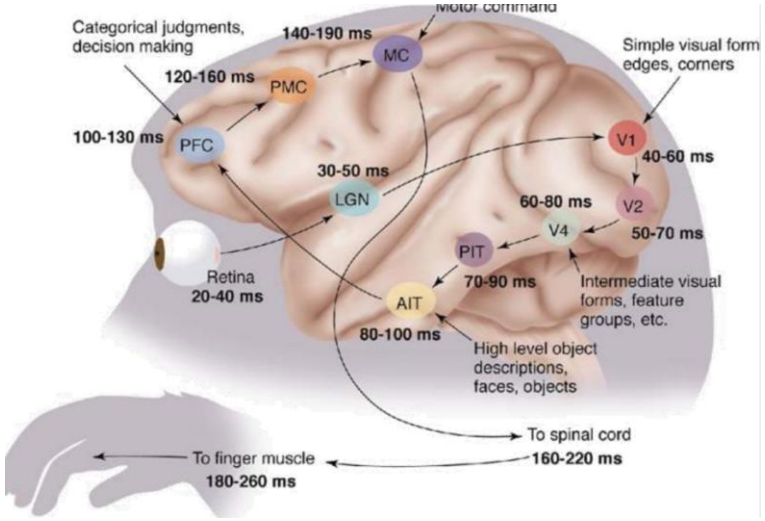
DEEP LEARNING: NEURAL NETWORKS

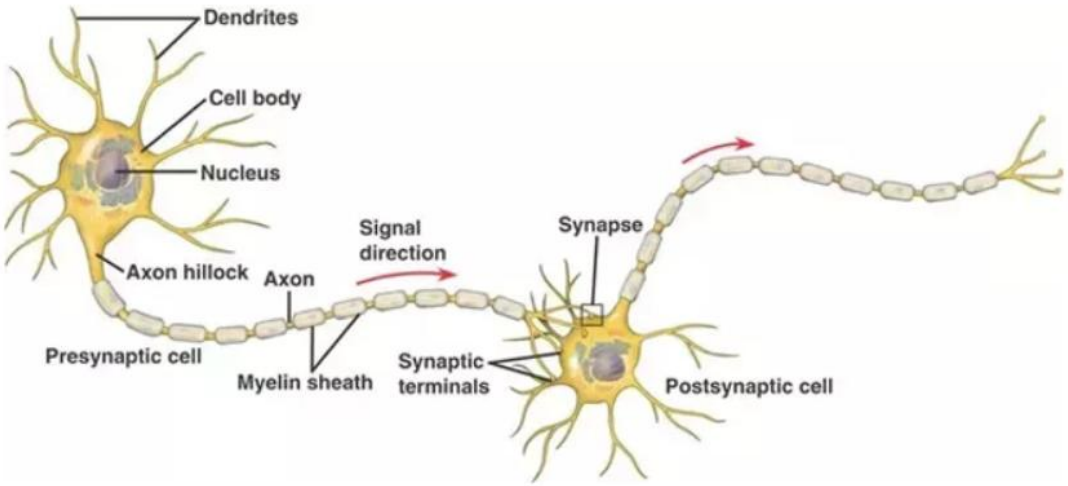
TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Deep learning: inspired by the brain

but not a model of the brain!

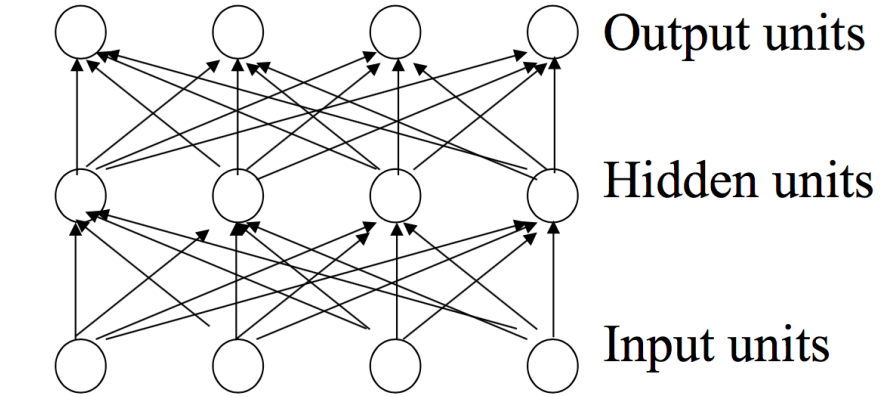




Our brain has lots of neurons connected together and the strength of the connections between neurons represents long term knowledge.

Artificial neural networks

Neural networks

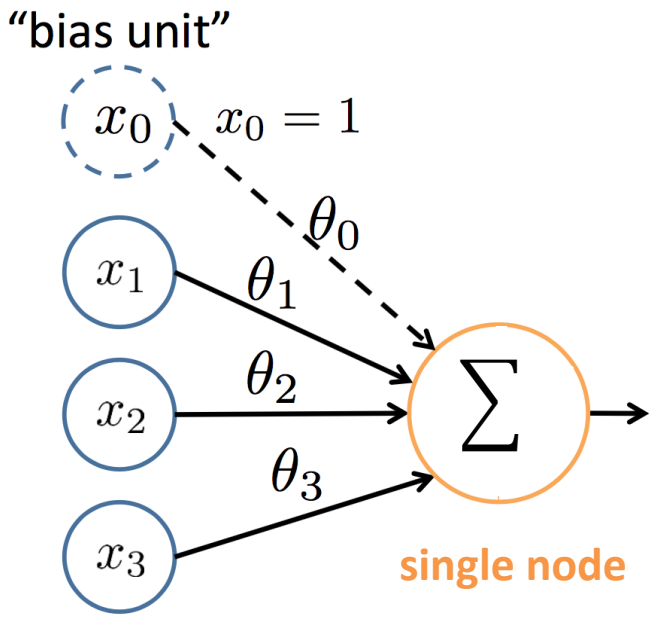


Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Linear regression

Neuron Model: Linear Unit



$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

input data

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

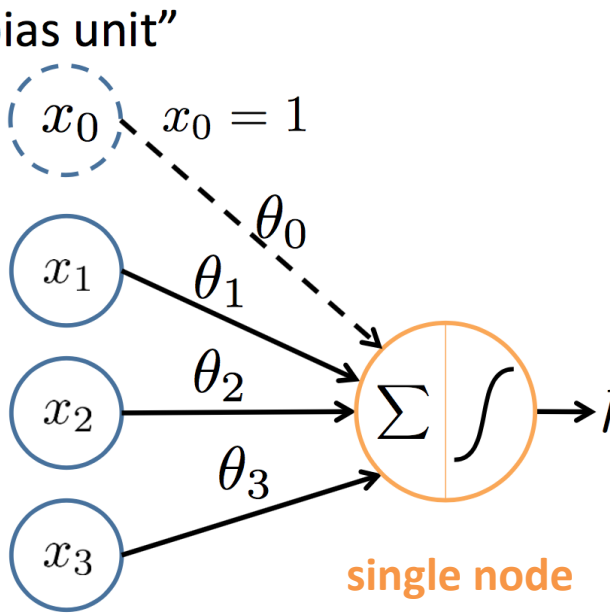
weights or parameters

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}$$

= weighted sum

Logistic regression

Neuron Model: Logistic Unit



$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

input data — **weights or parameters**

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

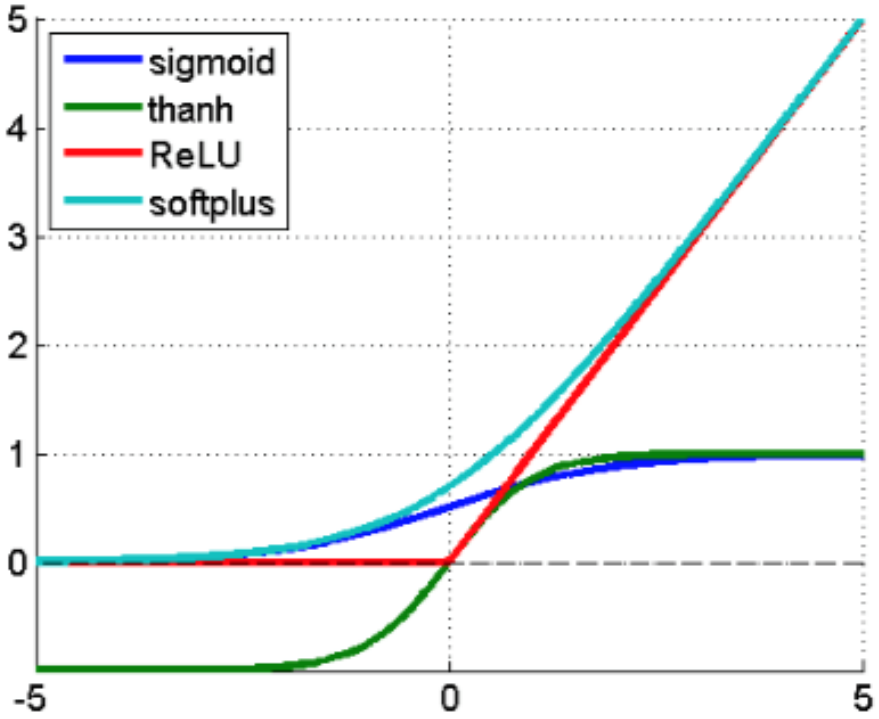
= activation of weighted sum

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

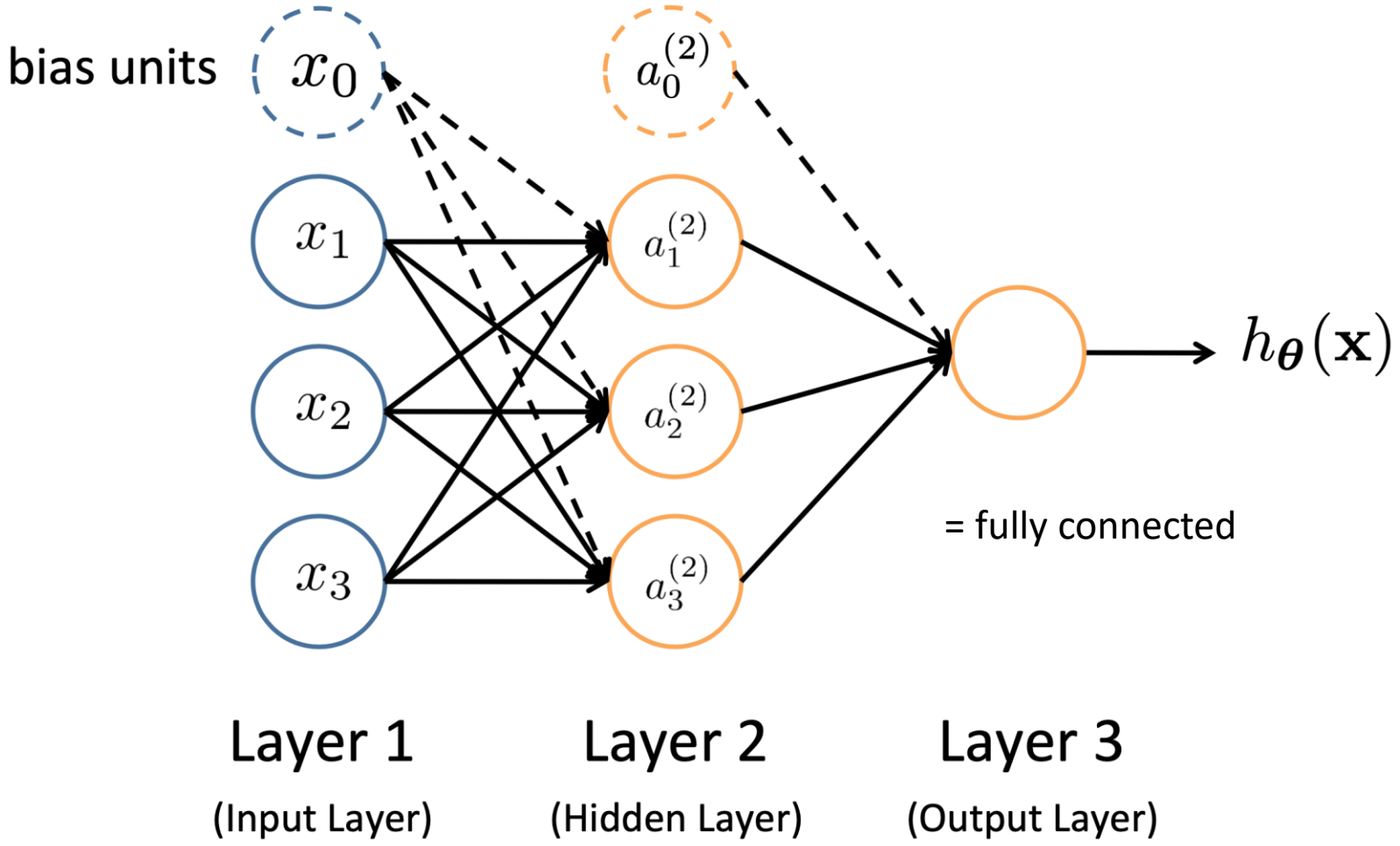
vives University of Applied Sciences

Activation functions



Most deep networks use ReLU - max(0,x) - nowadays for hidden layers, since it trains much faster, is more expressive than logistic function and prevents the gradient vanishing problem.

Layered networks with multiple neurons



DEEP learning = minimum 3 layers!

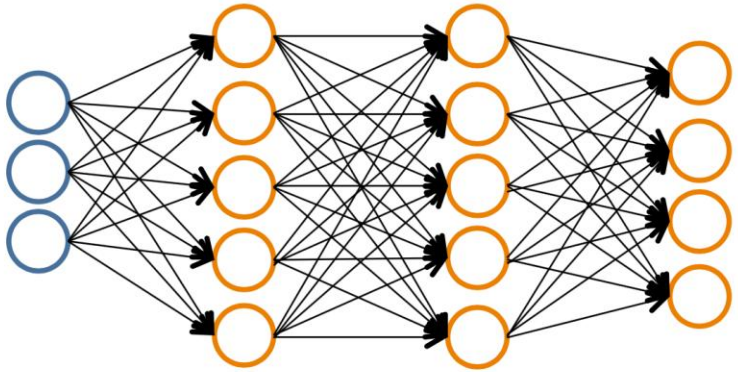
Example: image classification

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

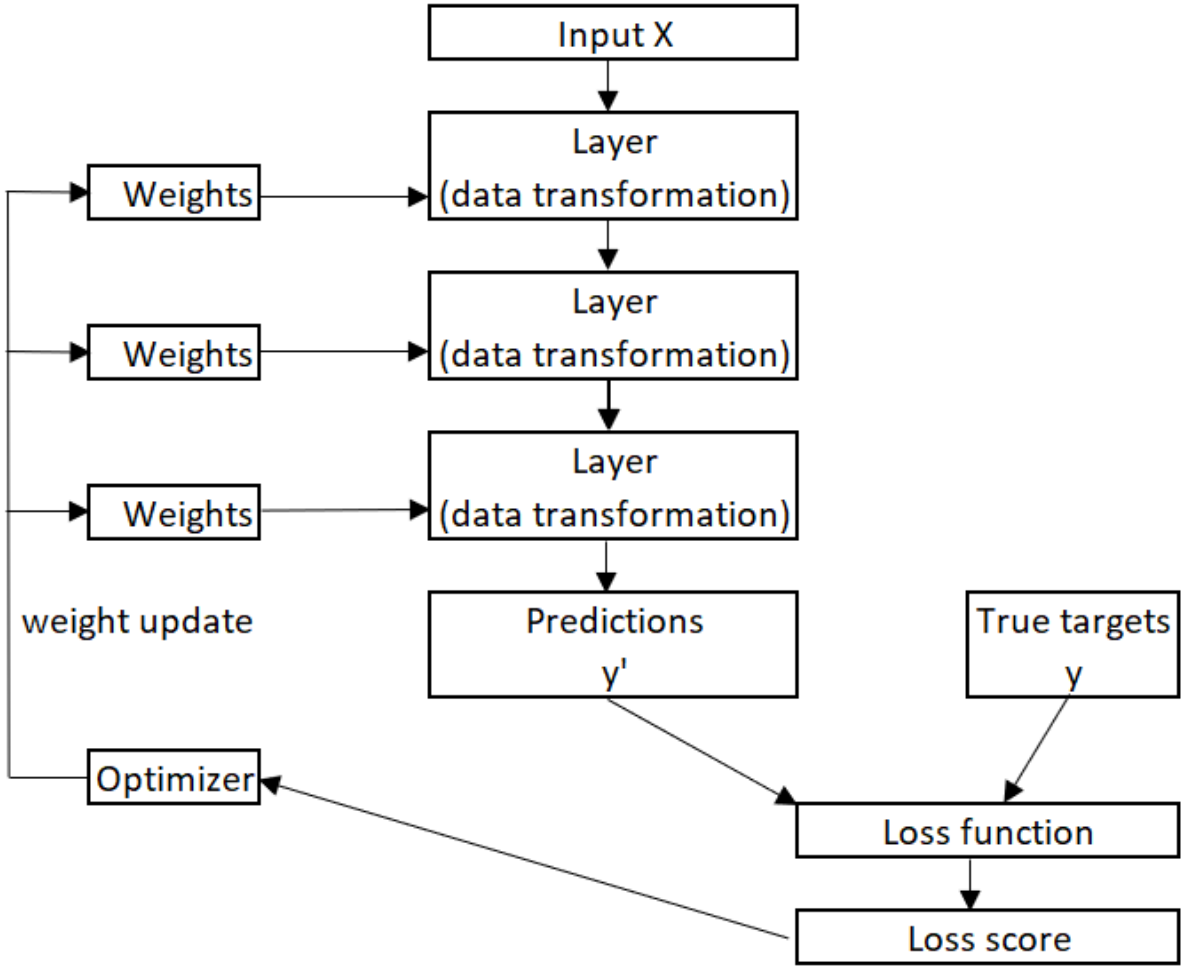
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

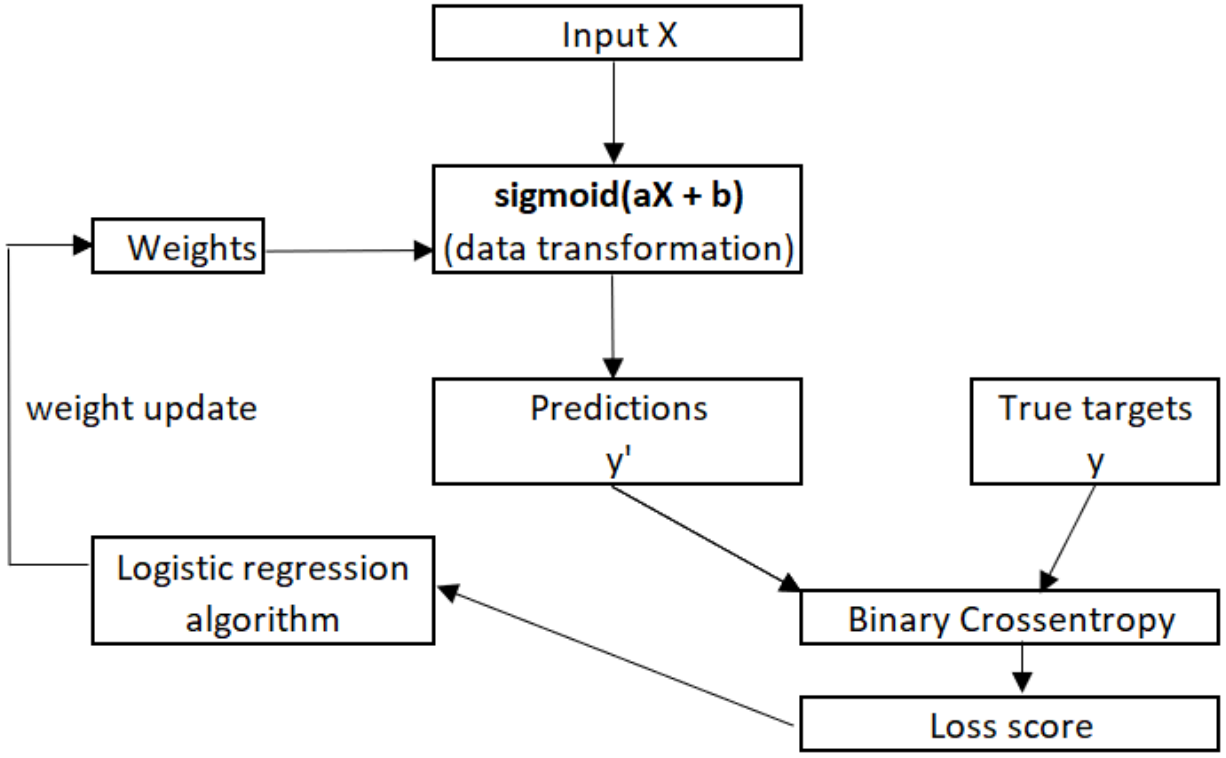
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

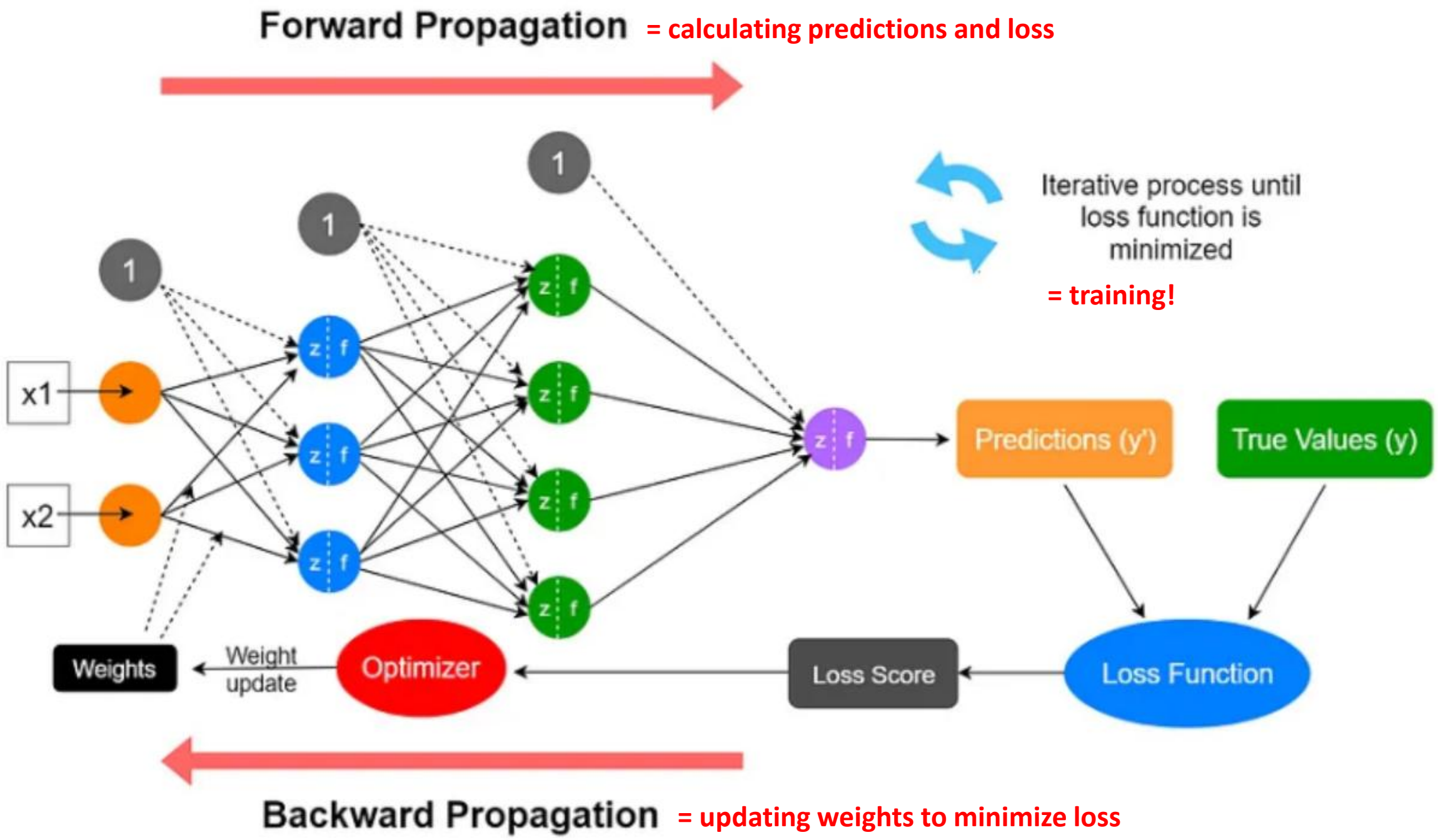
Understanding how neural networks learn



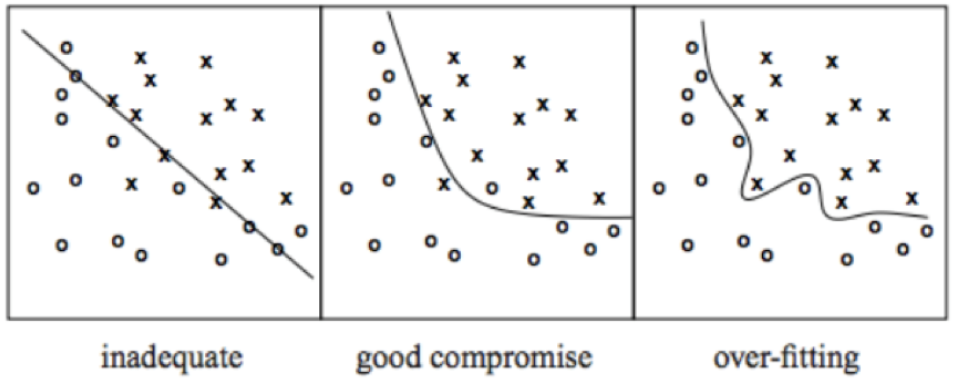
NEURAL NETWORK



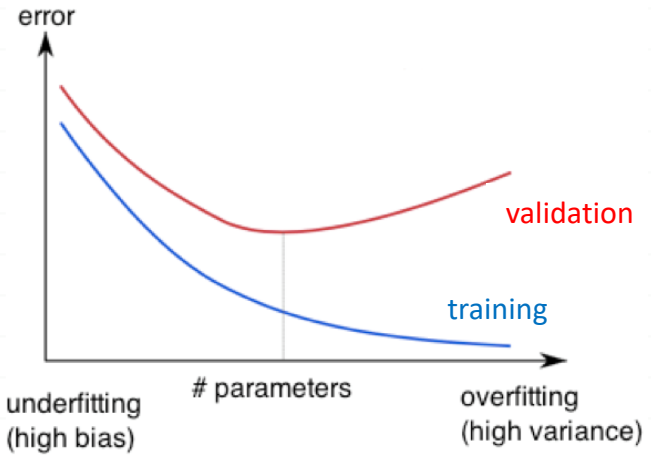
LOGISTIC REGRESSION


Forward Propagation
= calculating predictions and loss
1
1
1
x1
x2
z f
z f
z f
z f
z f
z f
z f
Iterative process until loss function is minimized
= training!
Predictions (y')
True Values (y)
Loss Function
Loss Score
Optimizer
Weight update
Weights
Backward Propagation
= updating weights to minimize loss

Overfitting is an issue!

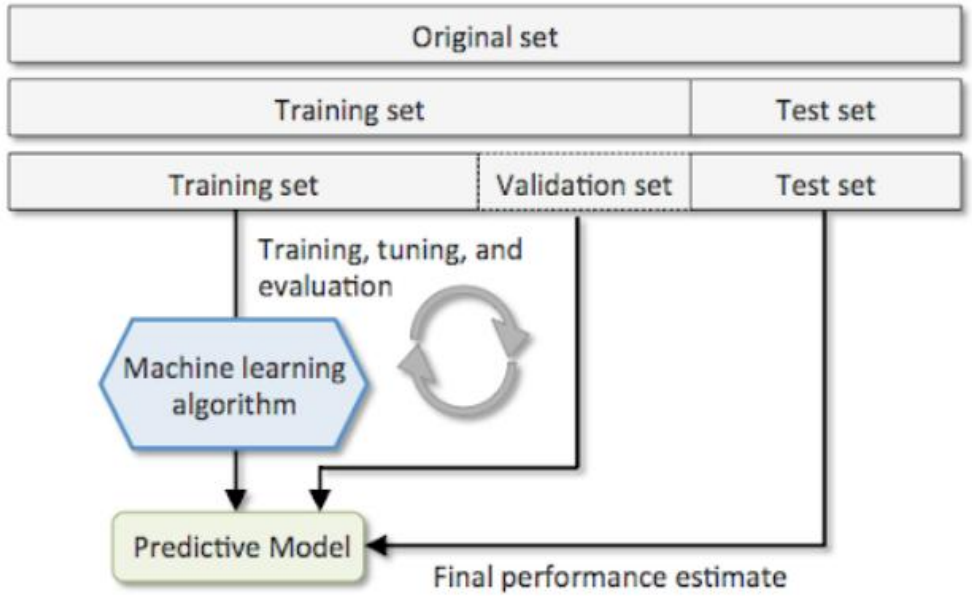


http://wiki.bethanycrane.com/overfitting-of-data



https://www.neuraldesigner.com/images/learning/selection_error.svg

Learned hypothesis may **fit** the training data very well, even outliers (**noise**) but fail to **generalize** to new examples



Splitting the dataset!

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Guest Lectures at TTK University of Applied Sciences, Tallinn, Estonia

Introduction to Deep Learning

DEEP LEARNING:
A FIRST NEURAL NETWORK

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

A first neural network

- MNIST dataset
  - **Classifying** handwritten digits = 10 classes (0 to 9)
  - 60000 training images, 10000 test images

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

  - First important step: exploring the data!
    - Images are 2D arrays of integers between 0 and 255

Note on classes and labels

In machine learning, a category in a classification problem is called a class. Data points are called samples. The class associated with a specific sample is called a label.

A first neural network

- **Preparing the data**
  - Reshape to 1D vectors
  - Convert to floats between 0 and 1

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255
```

A first neural network

- An artificial neural network consists of **layers**
- A layer = a *filter* that distills meaningful **representations** from the data

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

= the network architecture

A first neural network

- Compiling the model:
  - **Optimizer**
  - **Loss** function
  - Evaluation **metrics**

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

A first neural network

- Fitting the data = **training** the model

```
>>> model.fit(train_images, train_labels, epochs=5, batch_size=128)
Epoch 1/5
60000/60000 [==============================] - 5s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [========================>.....] - ETA: 1s - loss: 0.1035 - acc: 0.9692
```

- **Epochs** = the number of times the entire dataset is run through by the algorithm
- **Batch size** = the number of training examples that are grouped during one iteration

- Usually the model is also being **validated** during training:
  - to detect and overcome **overfitting**
  - to finetune the hyperparameters

A first neural network

- Making **predictions**:

```
>>> test_digits = test_images[0:10]
>>> predictions = model.predict(test_digits)
>>> predictions[0]
array([1.0726176e-10, 1.6918376e-10, 6.1314843e-08, 8.4106023e-06,
       2.9967067e-11, 3.0331331e-09, 8.3651971e-14, 9.9999106e-01,
       2.6657624e-08, 3.8127661e-07], dtype=float32)
```

Method `predict` returns probabilities!

A first neural network

- Maximum probability = class to which a sample “probably” belongs

```
>>> predictions[0].argmax()
7
>>> predictions[0][7]
0.99999106
```

```
>>> test_labels[0]
7
```

The predicted class is correct in this example!

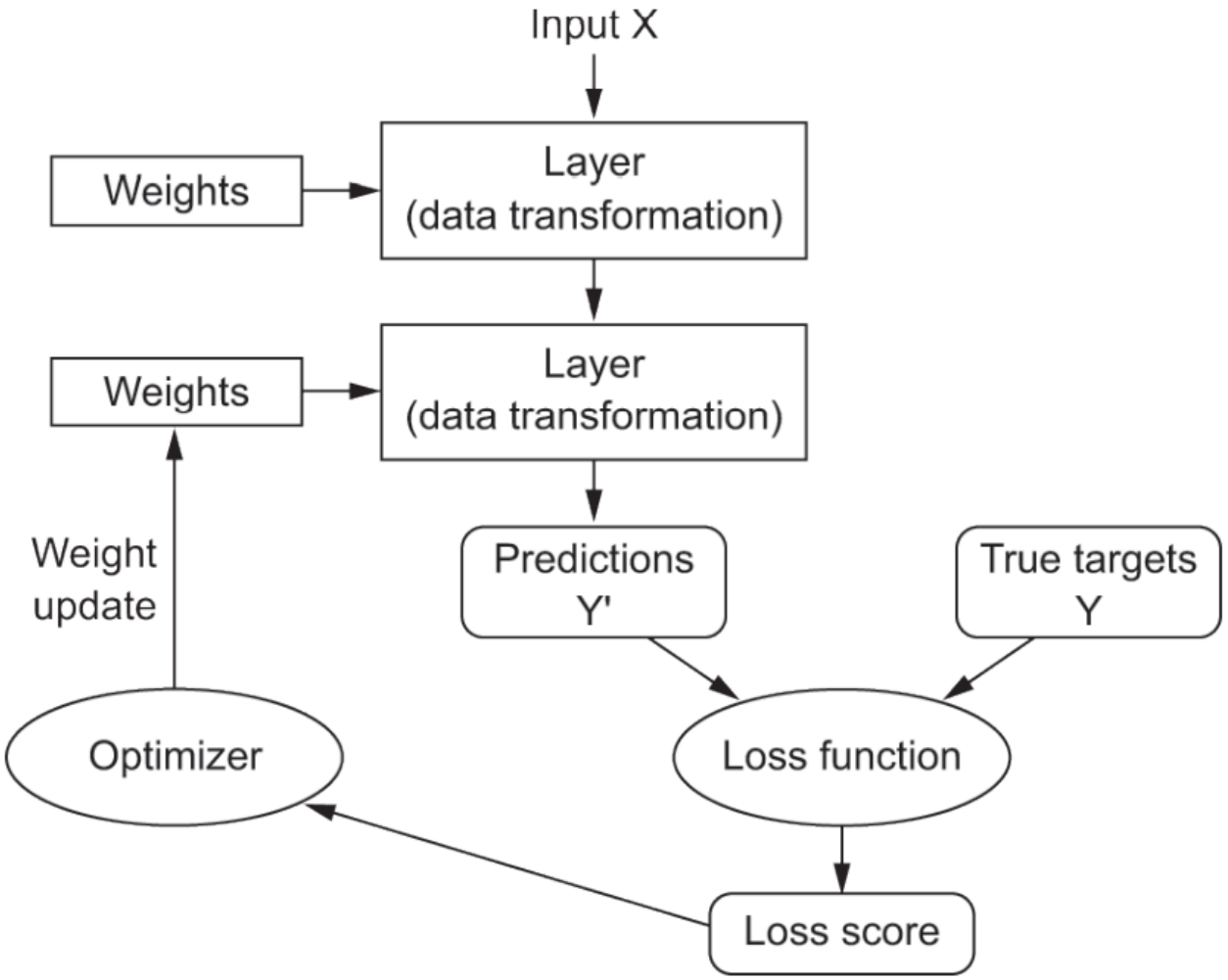
A first neural network

- **Evaluating** the model using the **testset**

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"test_acc: {test_acc}")
test_acc: 0.9785
```

Overfitting?

A first neural network




```
model = models.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

```
model.compile(
    optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
```

```
model.fit(train_images, train_labels,
          epochs=5, batch_size=128);
```

Guest Lectures at TTK University of Applied Sciences, Tallinn, Estonia

Introduction to Deep Learning

DEEP LEARNING: TENSORS

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Tensors

Tensor = multi-dimensional array

= **basic data structure** for representing and manipulating data in deep learning

= NDarray in NumPy

- **Scalar** = rank-0 tensor (0 axes)

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

- **Vector** = rank-1 tensor (1 axis)

```
>>> x = np.array([12, 3, 6, 14, 7])
>>> x
array([12, 3, 6, 14, 7])
>>> x.ndim
1
```

Tensors

- **Matrix** = rank-2 tensor (2 axes)

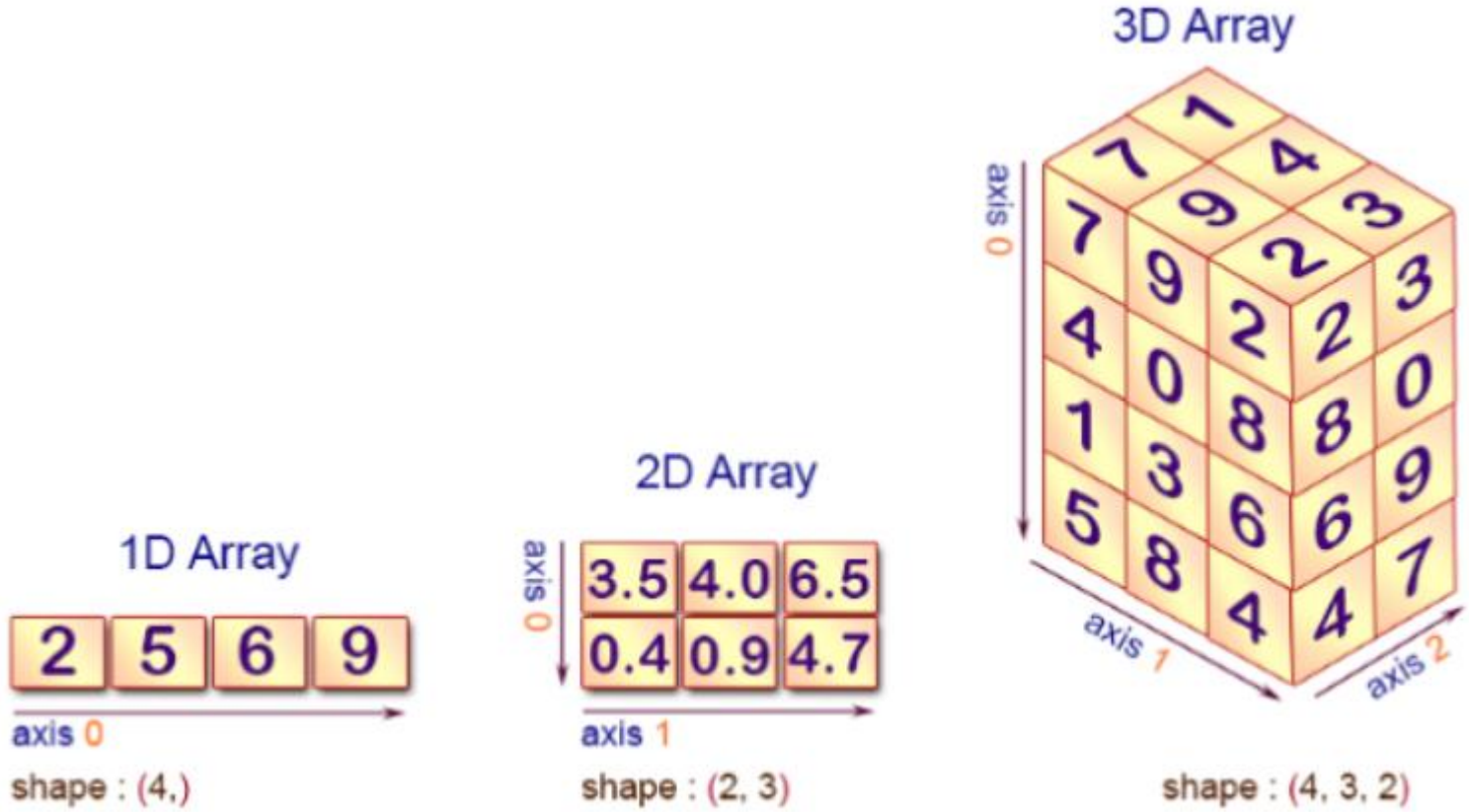
```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

- **Multi-dimensional array** = Rank-3 or higher tensor (N axes with N >= 3)

```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

Tensors

NumPy arrays: **axes**



Tensors

A tensor is defined by three key attributes:

- *Number of axes (rank)*—For instance, a rank-3 tensor has three axes, and a matrix has two axes. This is also called the tensor's `ndim` in Python libraries such as NumPy or TensorFlow.
- *Shape*—This is a tuple of integers that describes how many dimensions the tensor has along each axis. For instance, the previous matrix example has shape `(3, 5)`, and the rank-3 tensor example has shape `(3, 3, 5)`. A vector has a shape with a single element, such as `(5,)`, whereas a scalar has an empty shape, `()`.
- *Data type* (usually called `dtype` in Python libraries)—This is the type of the data contained in the tensor; for instance, a tensor's type could be `float16`, `float32`, `float64`, `uint8`, and so on. In TensorFlow, you are also likely to come across `string` tensors.

Tensors

- The MNIST dataset is also a tensor!

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Next, we display the number of axes of the tensor `train_images`, the `ndim` attribute:

```
>>> train_images.ndim
3
```

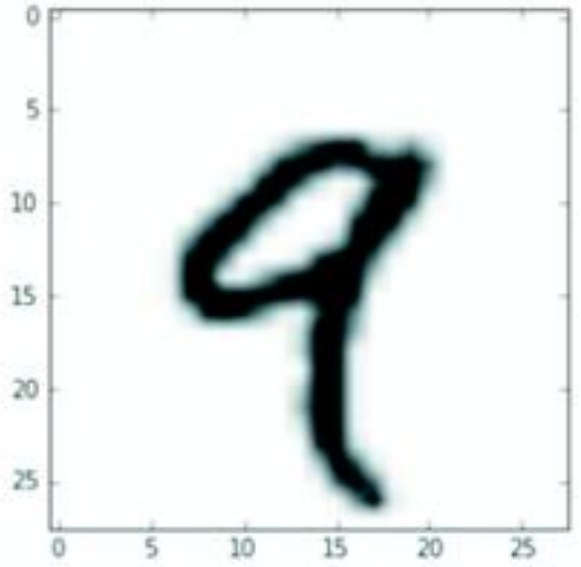
Here's its shape:

```
>>> train_images.shape
(60000, 28, 28)
```

And this is its data type, the `dtype` attribute:

```
>>> train_images.dtype
uint8
```

```
import matplotlib.pyplot as plt
digit = train_images[4]
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```




```
>>> train_labels[4]
9
```

Tensors

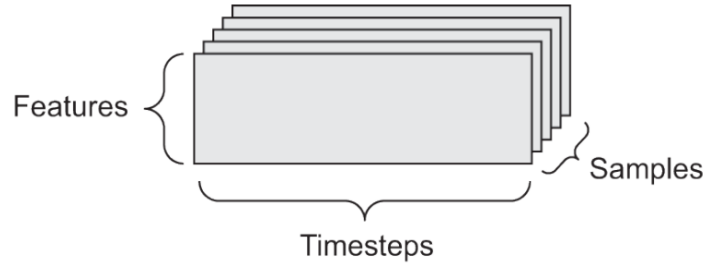
- **Vector data**: (samples, feature)
  - Tables and matrices (e.g. csv-files)
- **Time series**: (samples, timesteps, features)
  - E.g. dataset of stock prices, tweets, …
- **Images**: (samples, width, height, channels)
  - E.g. MNIST dataset
- **Videos**: (samples, frames, height, width, channels)
  - E.g. YouTube filmpjes

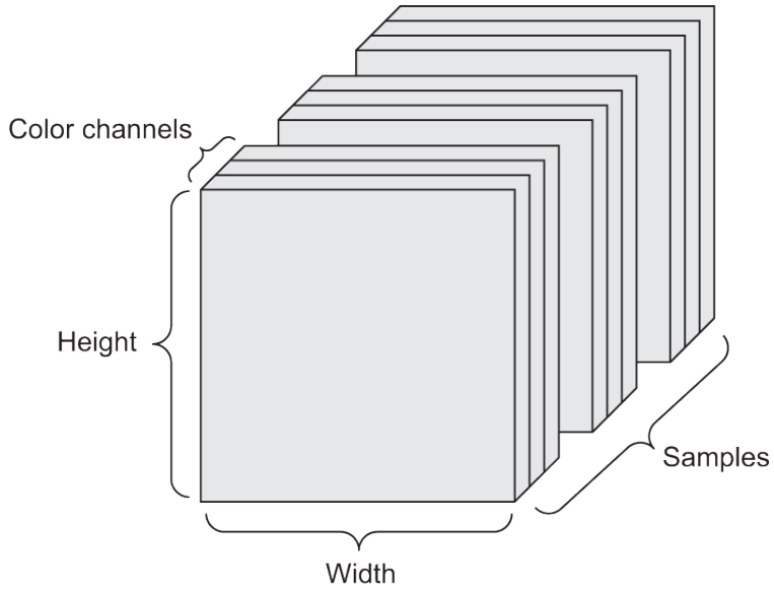
data(set)

	A	B	C	D	E	F
1	id	date	size	typos	recipients	spam
2	0	12/01/2021	2.5	0	1	False
3	1	13/01/2021	1.3	0	2	False
4	2	14/01/2021	12.1	3	15	True
5	3	15/01/2021	7.8	2	19	True
6	4	16/01/2021	4.6	1	5	False
7	5	17/01/2021	9.8	5	1	True
8	6	18/01/2021	11.6	3	63	True

example

feature target

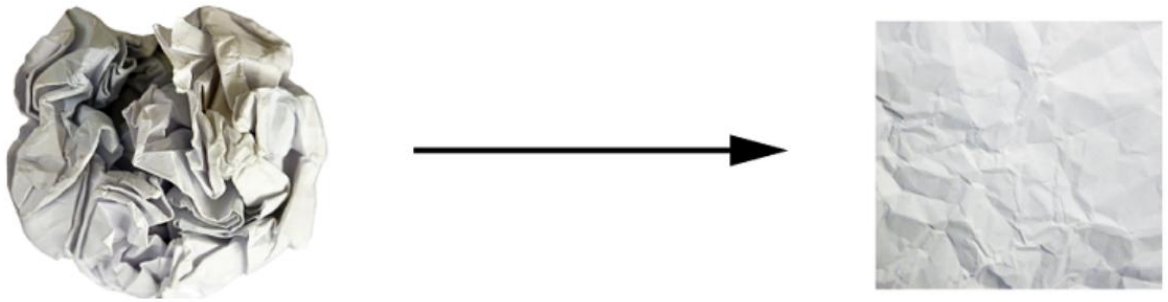




Tensors

- **Each layer** in the neural network performs **tensor operations** on the data
- These tensor operations **transform the data** into meaningful representations
- Tensor operations can be **interpreted geometrically**…
- … and so can neural networks:

Uncrumpling a complicated manifold of data



TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Guest Lectures at TTK University of Applied Sciences, Tallinn, Estonia

Introduction to Deep Learning

DEEP LEARNING: GRADIENT-BASED OPTIMIZATION

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Gradient-based optimisation

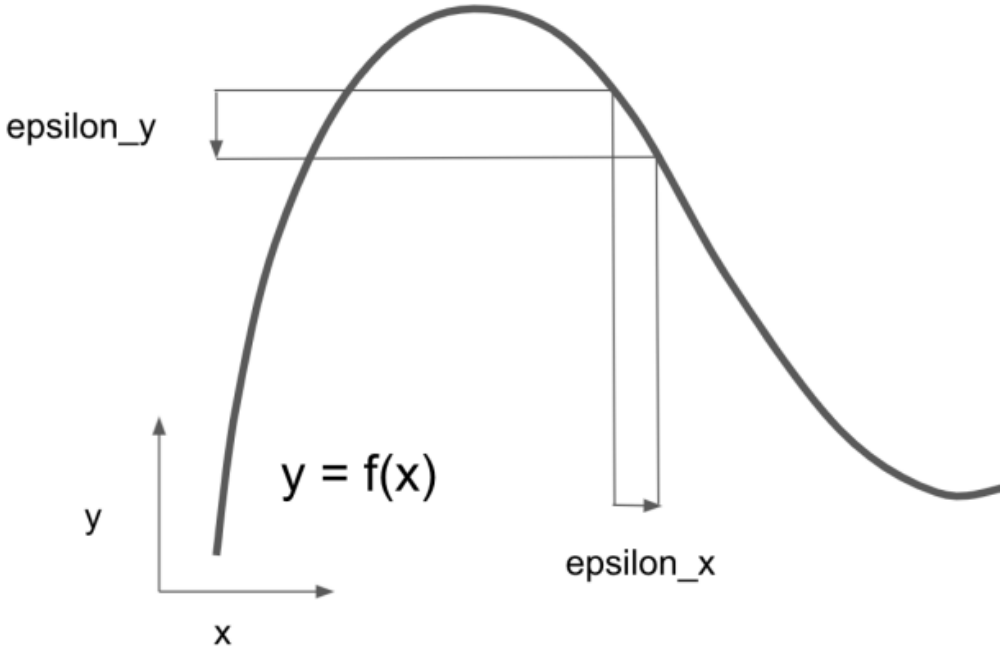
```
output = relu(dot(input, W) + b)
```

- **Random initialization** of weights W (+ biasvector b is initialized as 0)
- How to adjust parameters W and b to improve the model?
- **Training:** gradually adjust these parameters based on a feedback signal
- Repeat these steps, until the loss seems sufficiently low (= **training loop**):

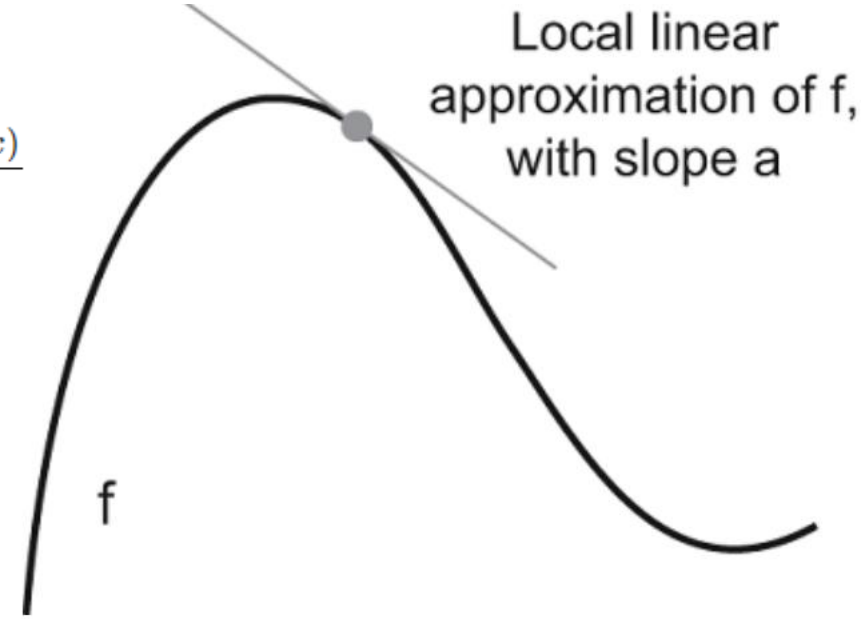
1. Draw a batch of training samples `x` and corresponding targets `y_true`.
2. Run the model on `x` (a step called the *forward pass*) to obtain predictions `y_pred`.
3. Compute the loss of the model on the batch, a measure of the mismatch between `y_pred` and `y_true`.
4. Update all weights of the model in a way that slightly reduces the loss on this batch.

→ **Gradient Descent**

Derivative and gradient



$$a = \frac{f(x+\epsilon) - f(x)}{\epsilon}$$



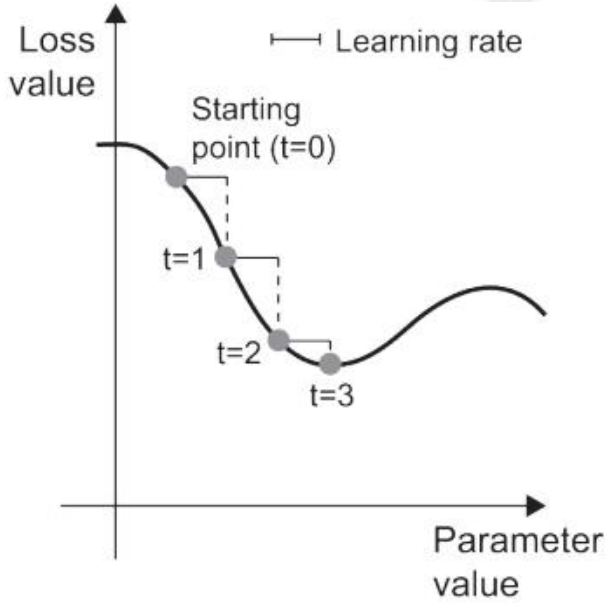
- De **afgeleide** is de richtingscoëfficiënt van de raaklijn in een punt
- De **gradiënt** kan je zien als de afgeleide in meerdere dimensies
- De afgeleide van een tensoroperatie is dus een gradiënt

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

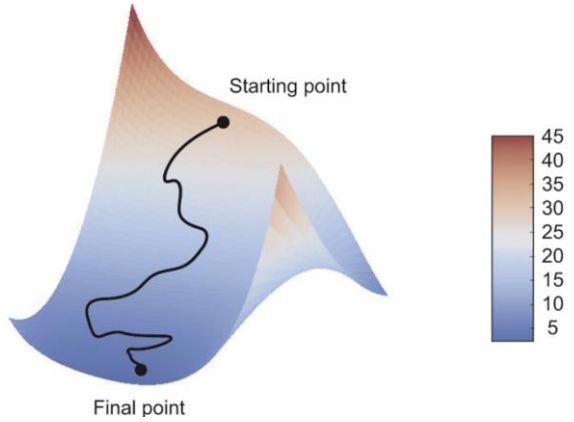
vives University of Applied Sciences

Stochastic Gradient Descent (SGD)

1. Draw a batch of training samples `x` and corresponding targets `y_true`.
2. Run the model on x to obtain predictions `y_pred` (this is called the *forward pass*).
3. Compute the loss of the model on the batch, a measure of the mismatch between `y_pred` and `y_true`.
4. Compute the gradient of the loss with regard to the model's parameters (this is called the *backward pass*).
5. Move the parameters a little in the opposite direction from the gradient — for example `W -= learning_rate * gradient` — thus reducing the loss on the batch a bit. The *learning rate* (`learning_rate` here) would be a scalar factor modulating the "speed" of the gradient descent process.



- **True SGD**: batchsize == 1
- **Mini-batch SGD**: 1 < batchsize < nsamples
- **(Full) Batch SGD**: batchsize == nsamples

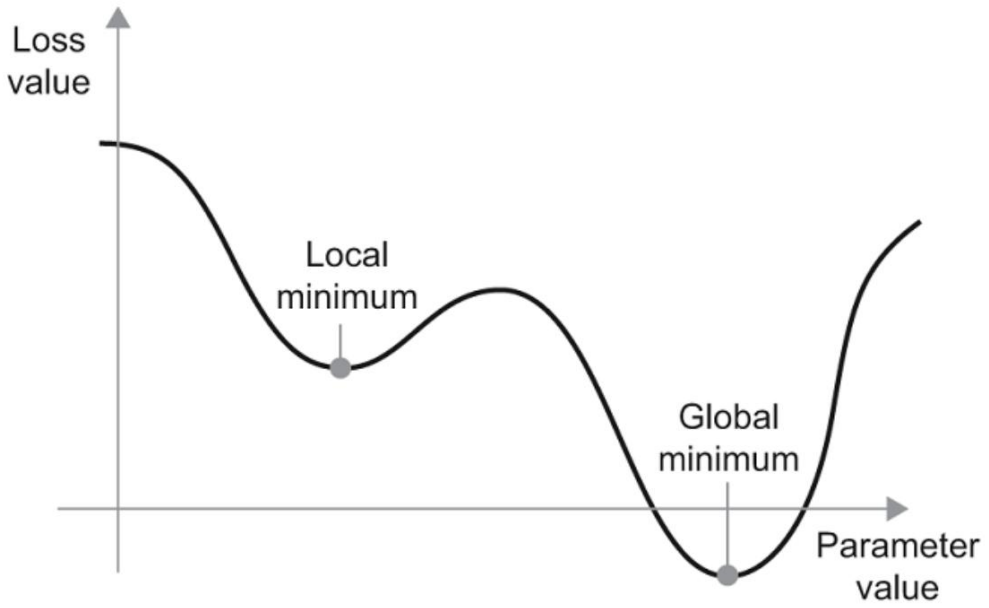


TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

Stochastic Gradient Descent: variants

- Probleem: lokale minima!
- Oplossing: **Momentum**

```
past_velocity = 0.
momentum = 0.1                                    ❶
while loss > 0.01:                                ❷
    w, loss, gradient = get_current_parameters()
    velocity = past_velocity * momentum - learning_rate * gradient
    w = w + momentum * velocity - learning_rate * gradient
    past_velocity = velocity
    update_parameter(w)
```



- Bijv. RMSProp, AdaGrad, …

Chain rule

- **Lossfunctie** van neuraal netwerk = samengestelde functie

```
loss_value = loss(y_true, softmax(dot(relu(dot(inputs, W1) + b1), W2) + b2))
```

- Gradiënt van lossfunctie nodig om parameters te updaten

```
W1 = W0 - alpha * grad(loss_value, W0)

b1 = b0 - alpha * grad(loss_value, b0)
```

- **Kettingregel**: gradiënt van samengestelde functie = product van gradiënten van functies

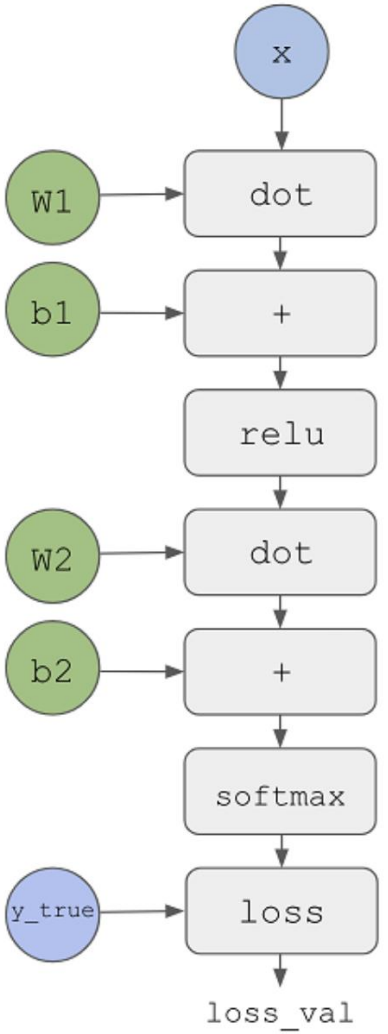
```
def fghj(x):
    x1 = j(x)
    x2 = h(x1)
    x3 = g(x2)
    y = f(x3)
    return y

grad(y, x) == grad(y, x3) * grad(x3, x2) * grad(x2, x1) * grad(x1, x)
```

- **Backpropagation** algoritme: kettingregel toepassen om gradiënt van lossfunctie te berekenen

Automatic differentation

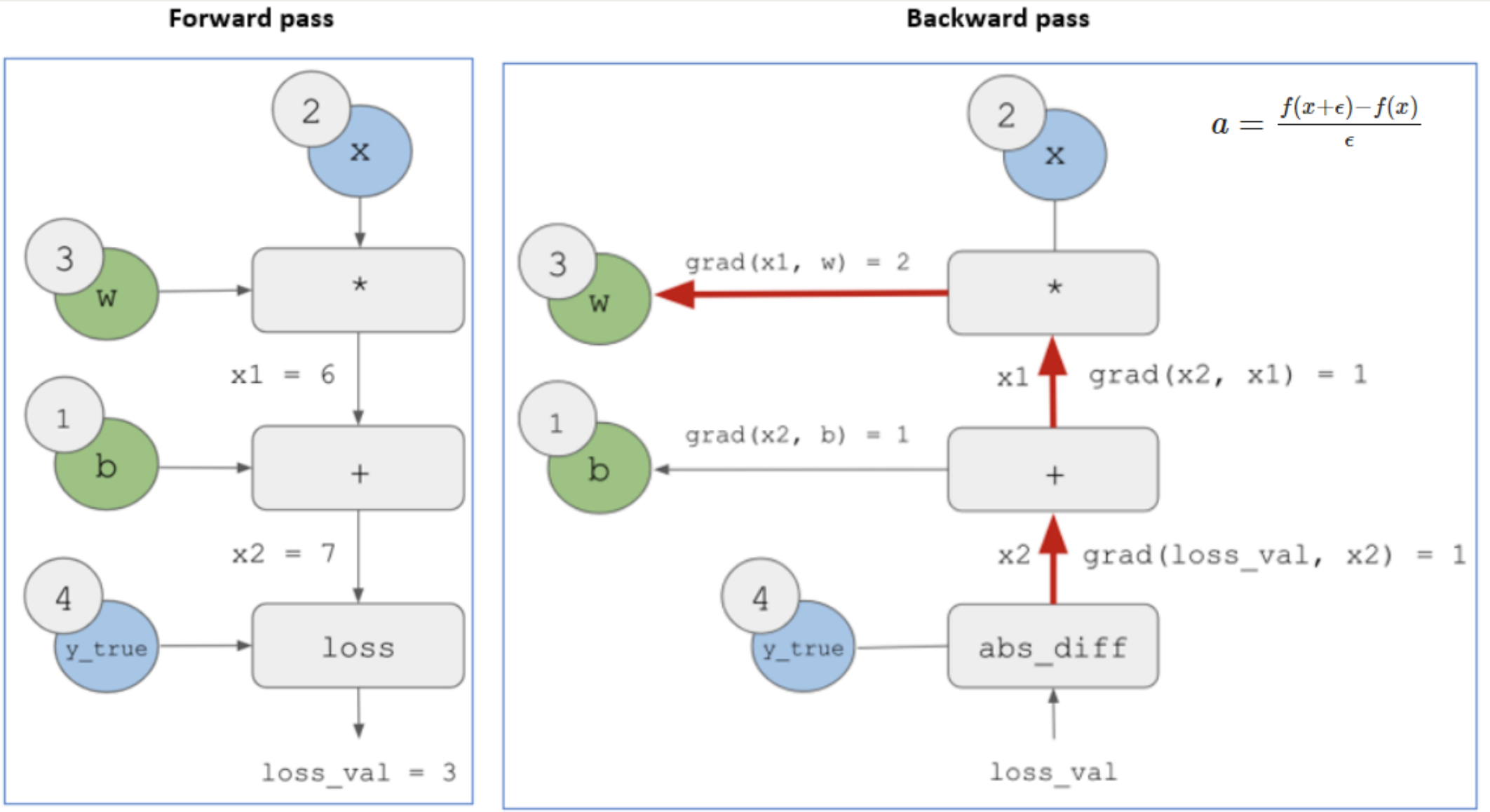
AUTOMATIC DIFFERENTIATION WITH COMPUTATION GRAPHS

A useful way to think about backpropagation is in terms of *computation graphs*. A computation graph is the data structure at the heart of TensorFlow and the deep learning revolution in general. It's a directed acyclic graph of operations — in our case, tensor operations. For instance, this is the graph representation of our first model:



TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

Backpropagation



Applying the chain rule

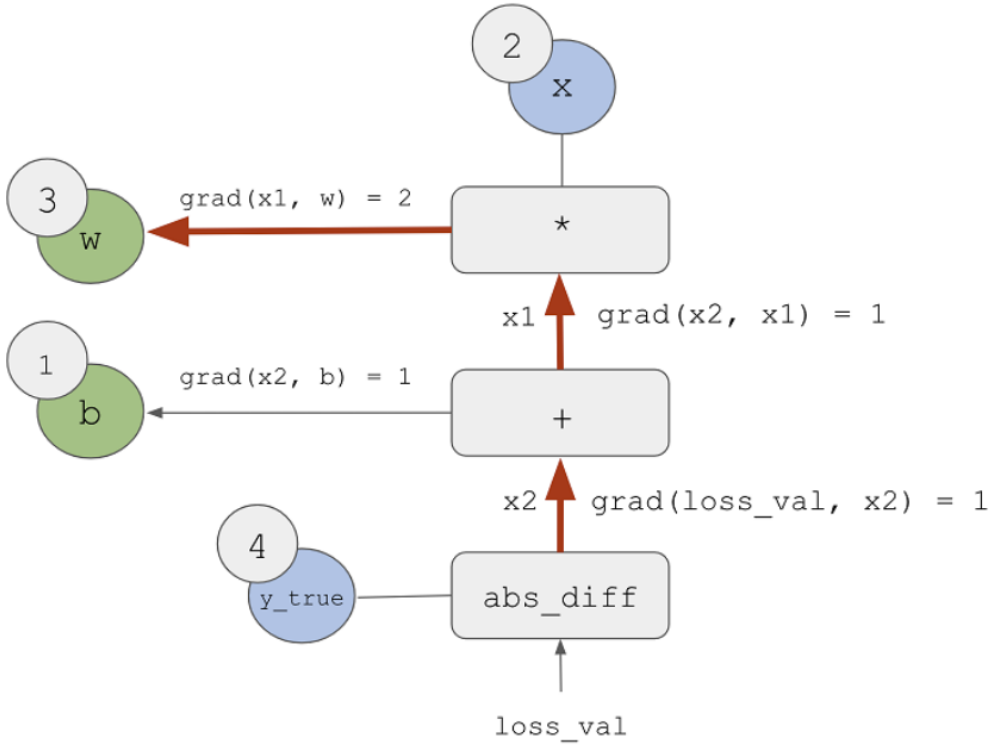


Figure 2.23 Path from loss_val to w in the backward graph

By applying the chain rule to our graph, we obtain what we were looking for:

- `grad(loss_val, w) = 1 * 1 * 2 = 2`
- `grad(loss_val, b) = 1 * 1 = 1`

Backpropagation in Excel

x	w	x1 = x*w	b	x2 = x1+b	y_true	loss_val = abs(x2-y_true)	eps
2	3	6	1	7	4	3	0,1
x	w	x1	b	x2 + eps	y_true	loss_val_eps	grad = (loss_val_eps - loss_val) / (x2 + eps - x2)
2	3	6	1	7,1	4	3,1	1
x	w	x1 + eps	b	x2	y_true	loss_val_eps	grad = (loss_val_eps - loss_val) / (x1 + eps - x1)
2	3	6,1	1	7,1	4	3,1	1
x	w + eps	x1	b	x2	y_true	loss_val_eps	grad = (loss_val_eps - loss_val) / (w + eps - w)
2	3,1	6,2	1	7,2	4	3,2	2
x	w	x1	b + eps	x2	y_true	loss_val_eps	grad = (loss_val_eps - loss_val) / (b + eps - b)
2	3	6	1,1	7,1	4	3,1	1

$$a = \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Tensorflow: automatic differentiation

THE GRADIENT TAPE IN TENSORFLOW

The API through which you can leverage TensorFlow's powerful automatic differentiation capabilities is the `GradientTape`. It's a Python scope that will "record" the tensor operations that run inside it, in the form of a computation graph (sometimes called a "tape"). This graph can then be used to retrieve the gradient of any output with respect to any variable or set of variables (instances of the `tf.Variable` class). A `tf.Variable` is a specific kind of tensor mean to hold mutable state — for instance, the weights of a neural network are always `tf.Variable` instances.

```
import tensorflow as tf
x = tf.Variable(0.)                                    ❶
with tf.GradientTape() as tape:                        ❷
  y = 2 * x + 3                                        ❸
grad_of_y_wrt_x = tape.gradient(y, x)                  ❹
```

Guest Lectures at TTK University of Applied Sciences, Tallinn, Estonia

Introduction to Deep Learning

DEEP LEARNING: NETWORK TYPES

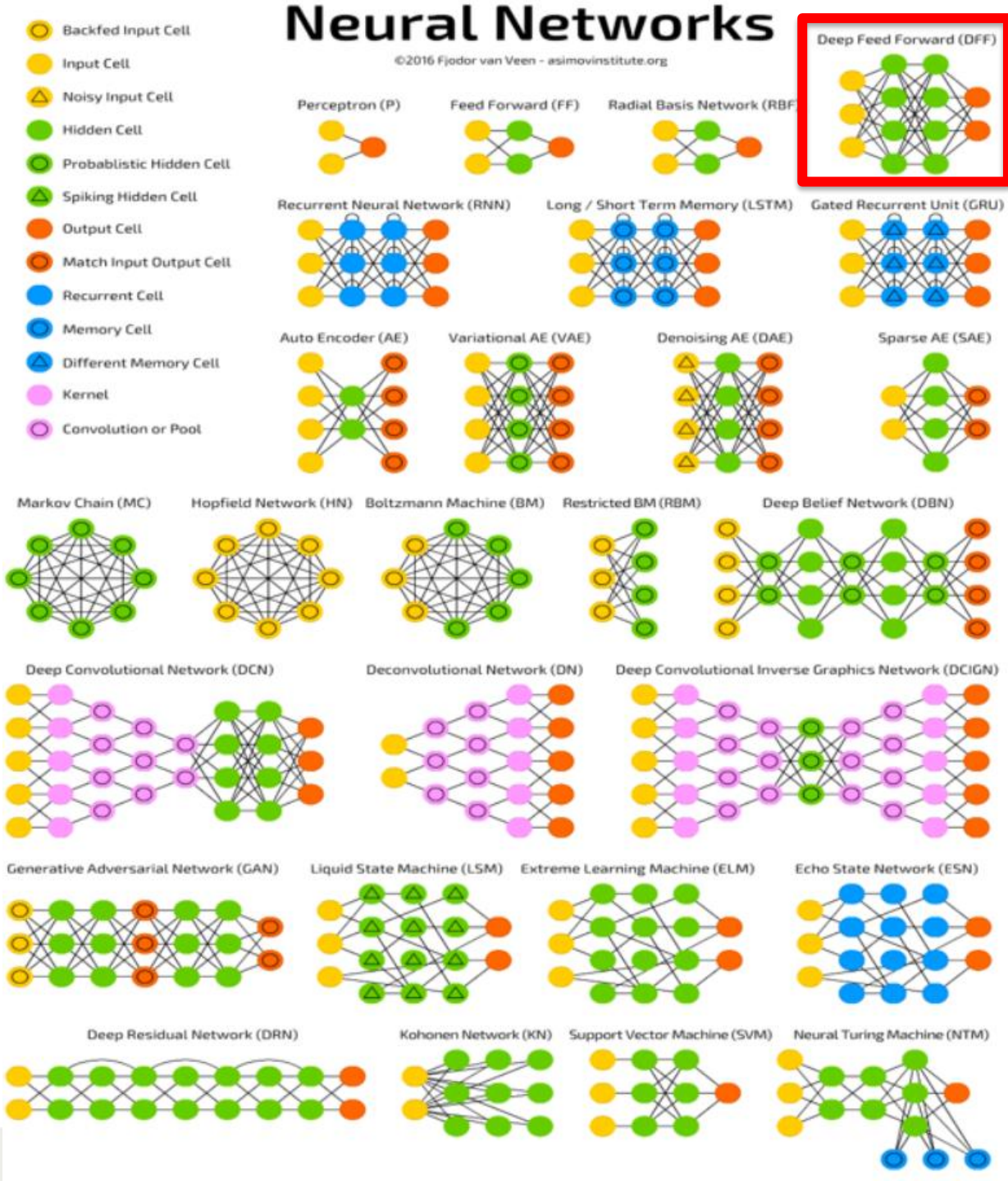
TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Network architectures

- So far we have only discussed
  (Deep) Feed Forward Networks (FFN)

- But there are many other network types:
  - Convolutional Neural Networks (CNN)
  - Recurrent Neural Networks (RNN)
  - Long Short-Term Memory Networks (LSTM)
  - Transformers
  - Variational Autoencoders (VAE)
  - Generative Adversarial Networks (GAN)
  - …



Convolution Neural Networks

- Most often used for **computer vision** tasks:
  - Image classification
  - Image segmentation
  - Object detection
  - …

- Consist of **convolutional layers** that:
  - apply **filters** to capture features such as edges, colors, textures, patterns, … (think of Instagram filters)
  - are usually combined with **pooling layers** to downsample the feature maps

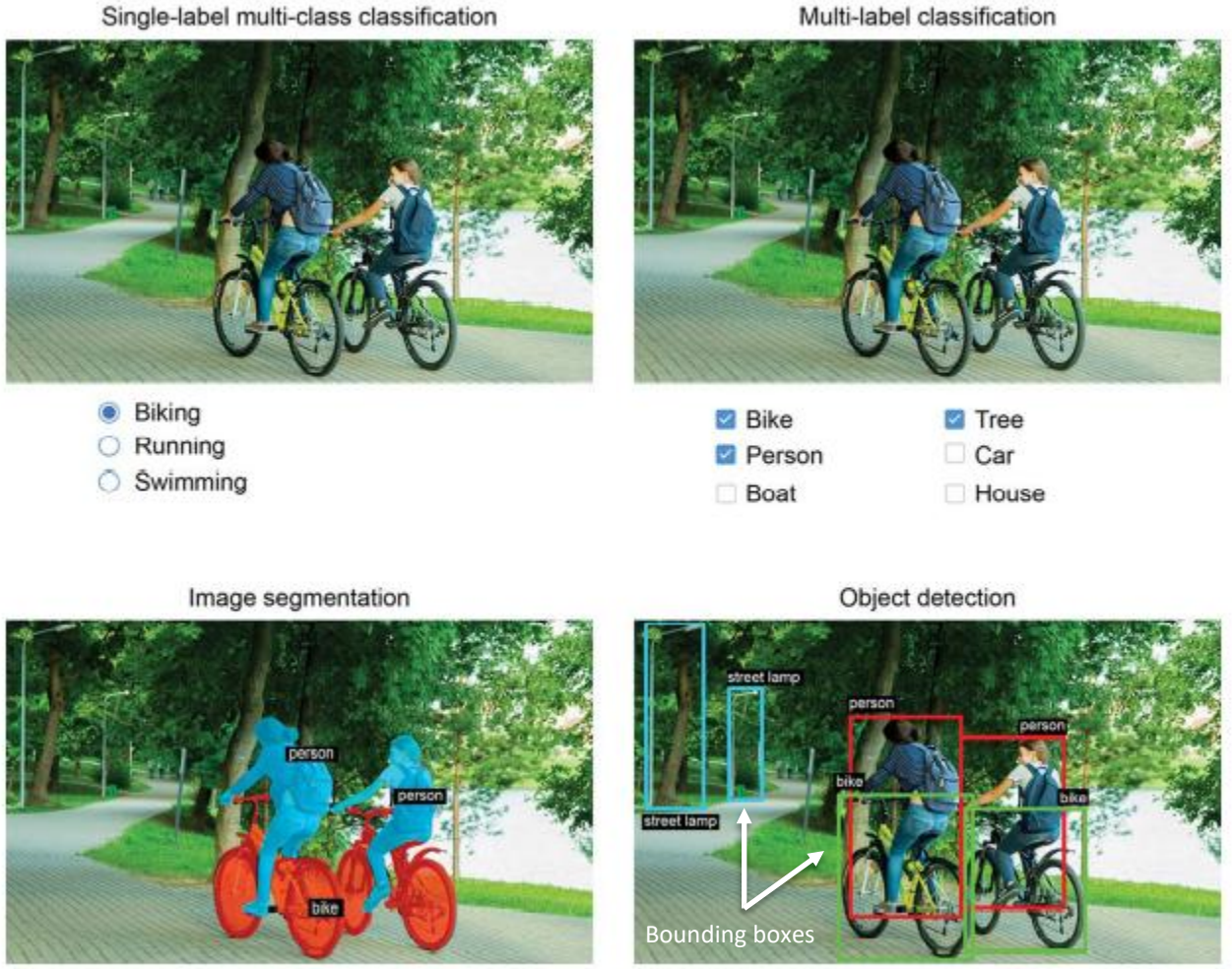
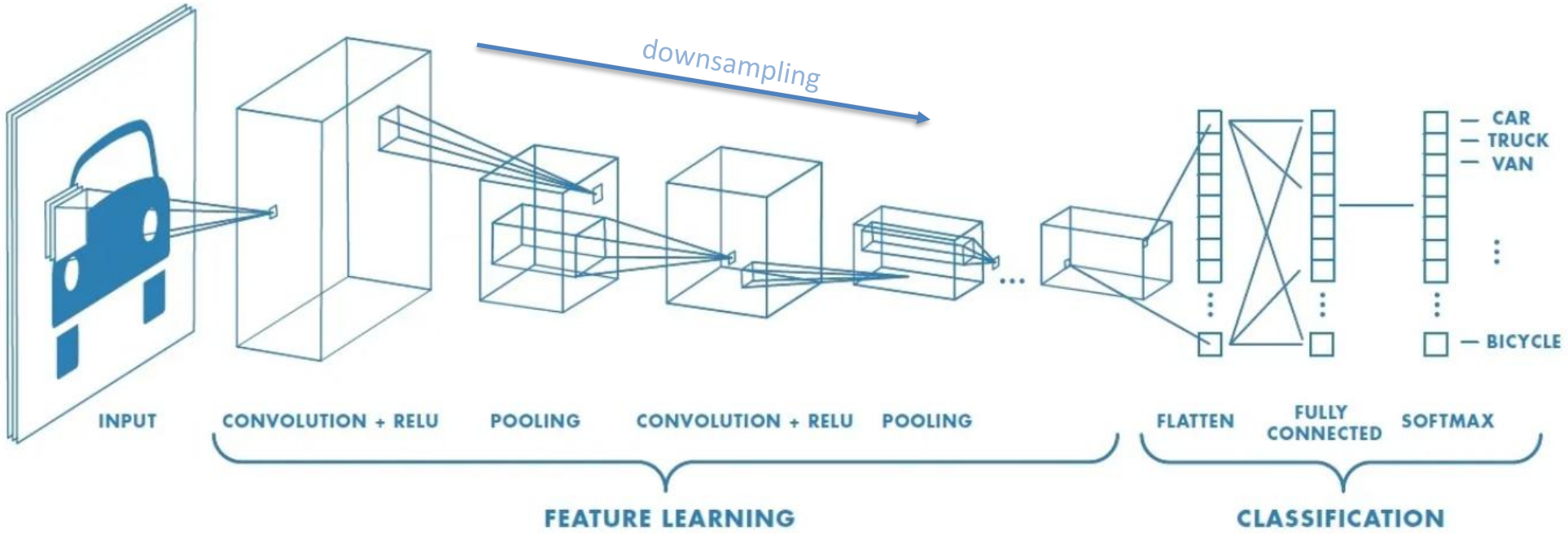


Image classification

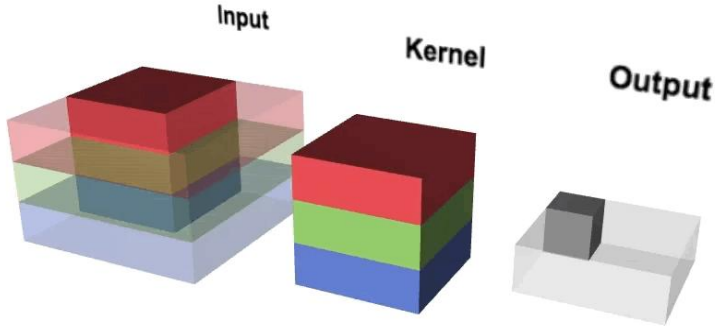
- Convnet = convolution base + classifier
- The **convolutional base** learns the features and consists of convolutional layers
- The **classifier** does the classification and consists of fully connected (dense) layers



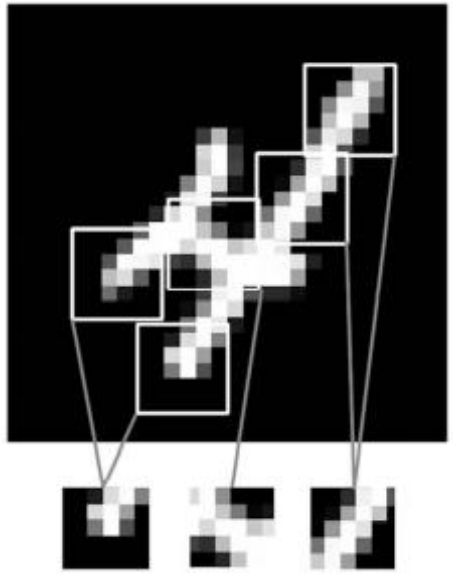
https://towardsdatascience.com/deep-dive-into-convolutional-networks-48db75969fdf

How convolution works

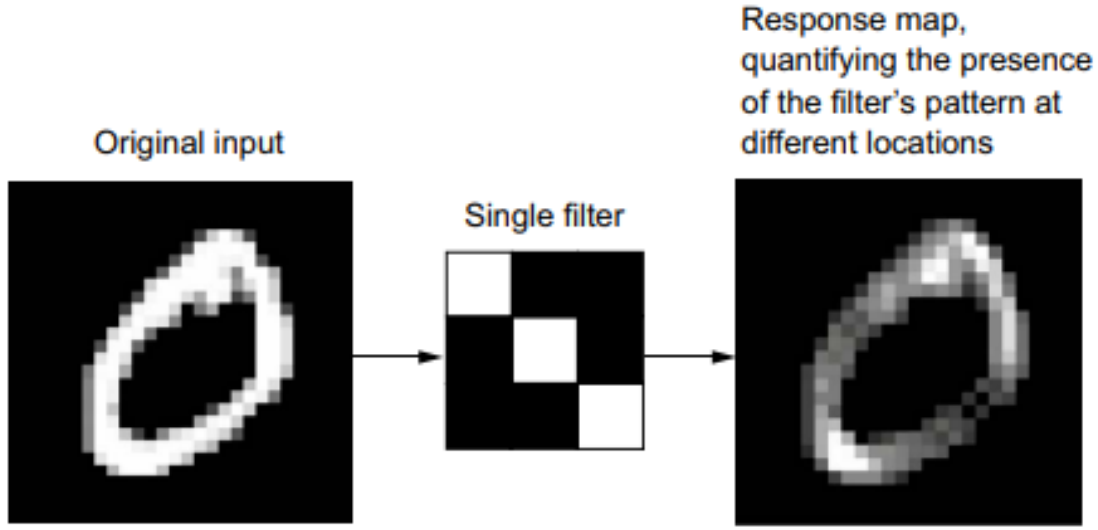
- A convolutional layer is a **feature detector**
- It ‘slides’ filters (= convolutional **kernels**) over the input image
- The result of ‘sliding’ a filter is a response **feature map**
- In this way convolutional layers learn **local patterns**
  by optimizing the filters during training



https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108

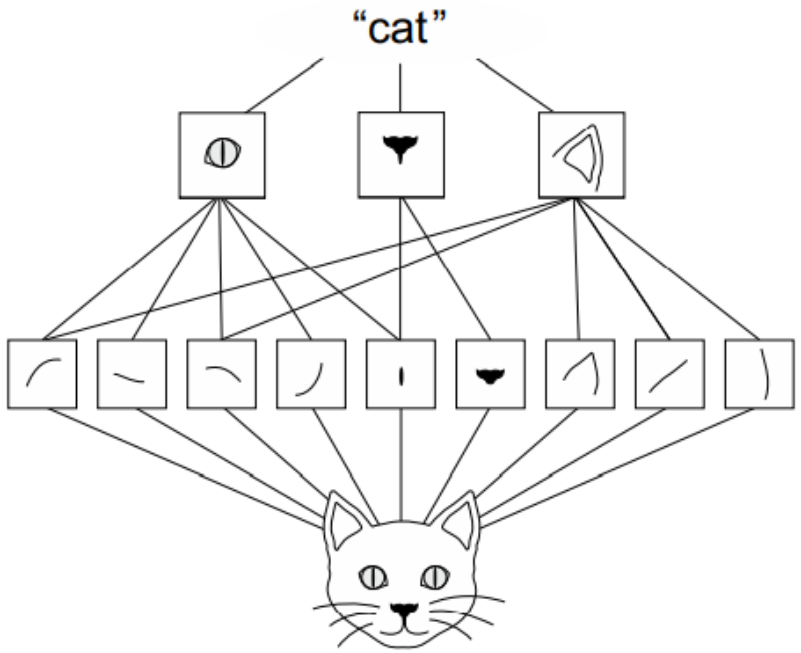


Learning local patterns

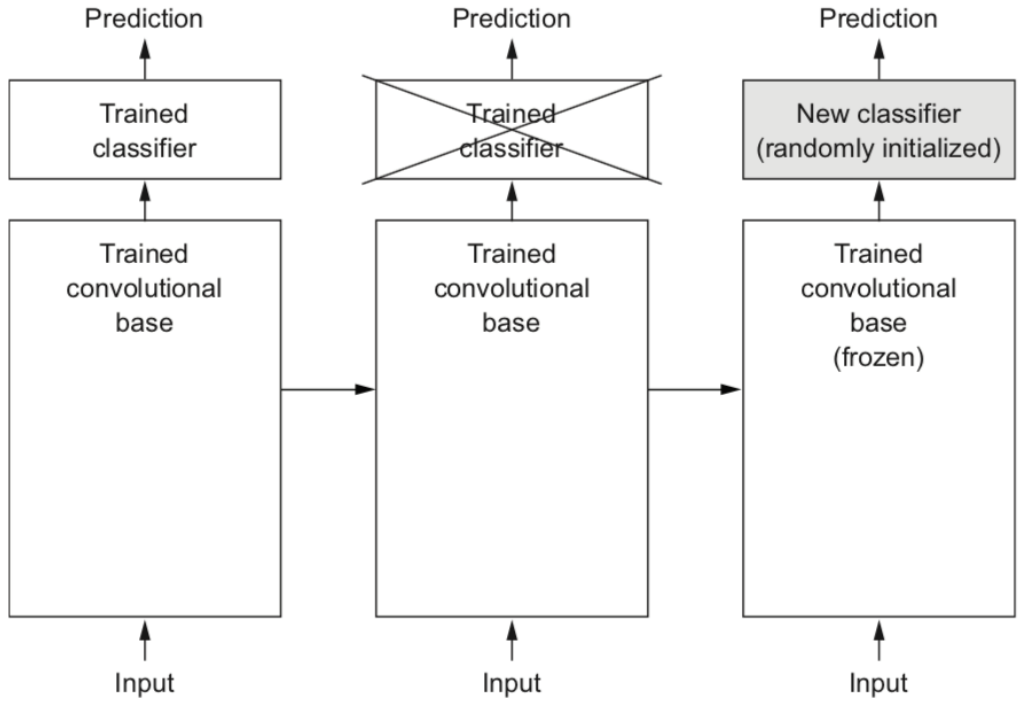


Interesting properties of CNNs

- The learned patterns are **translation-invariant**
  (= the location of a pattern or object on an image is irrelevant)
- By downsampling the feature maps, **spatial hierarchies** of patterns are learned
- The convbase can be reused (= **transfer learning**)



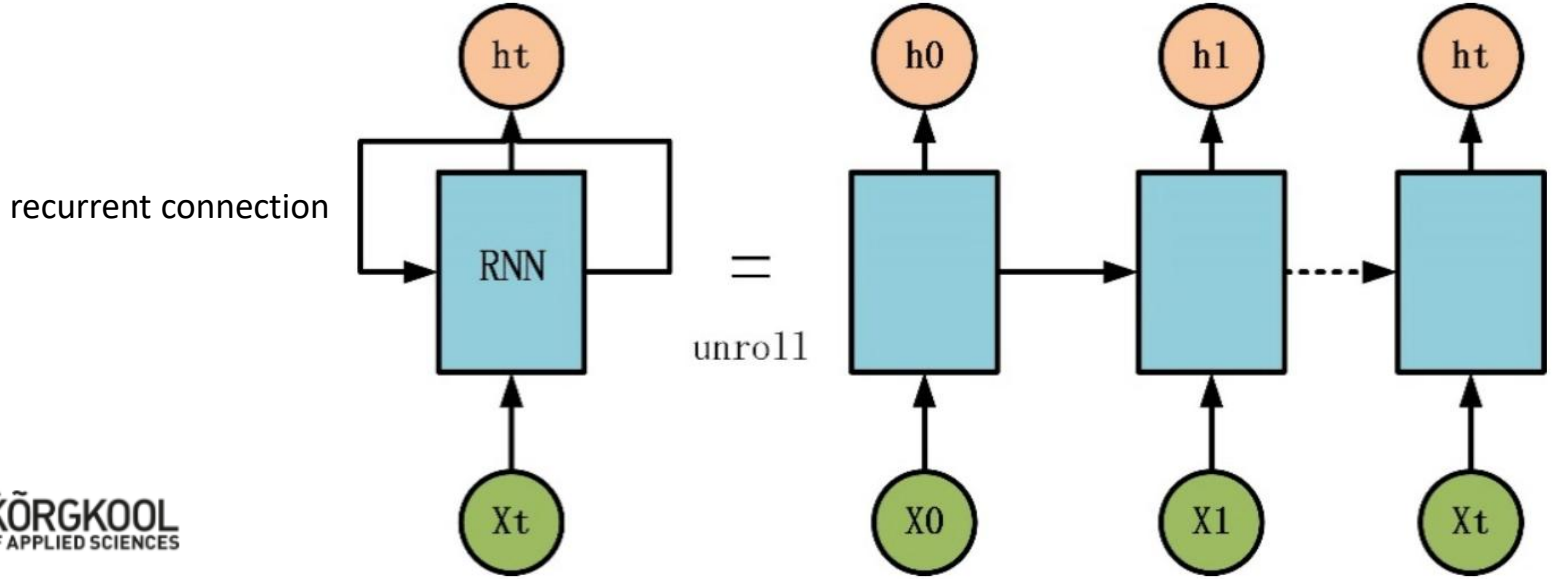
Learning spatial hierarchies



Transfer learning

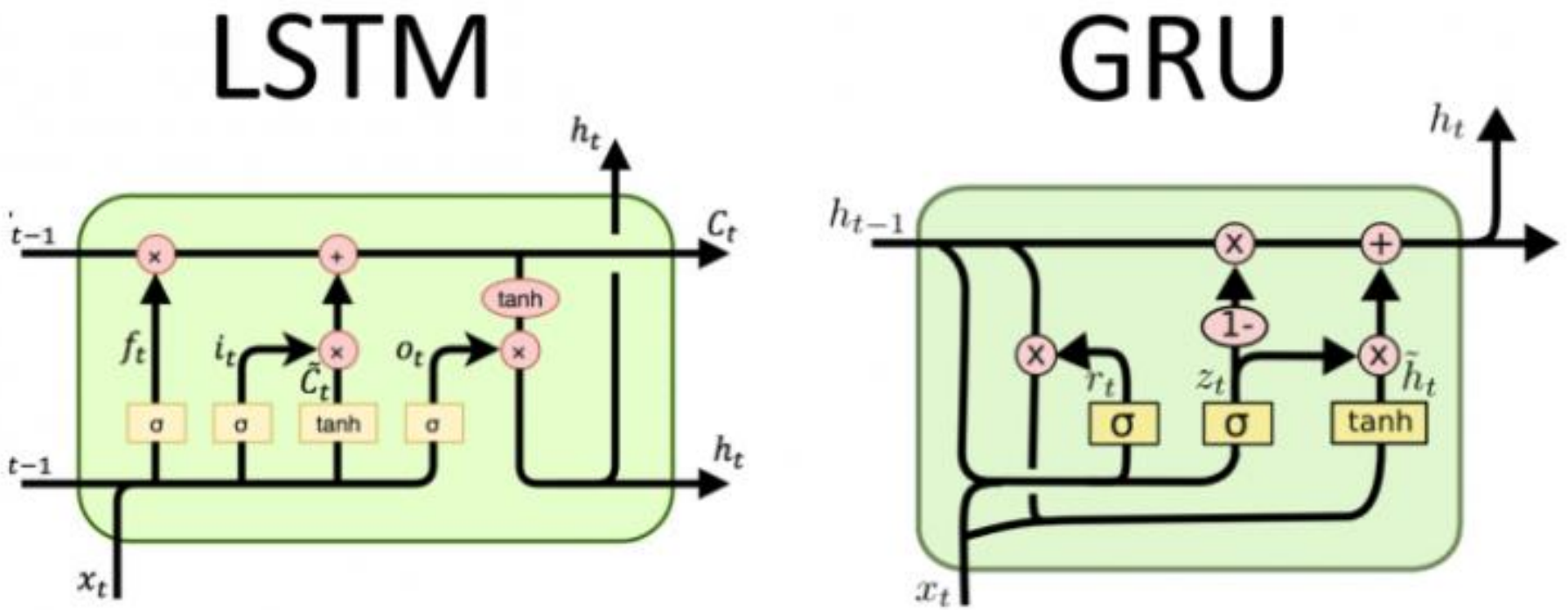
Recurrent Neural Networks

- RNNs are designed for **sequence data** (= data with strict order)
- RNNs allow information to be retained and processed over time.
- RNNs use recurrent connections to maintain a **memory** of previous inputs
- RNNs are suitable for tasks like **natural language processing** and **time series analysis**



Examples of RNNs

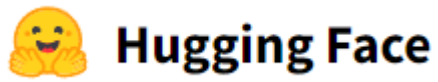
- LSTM = Long Short-Term Memory
- GRU = Gated Recurrent Unit

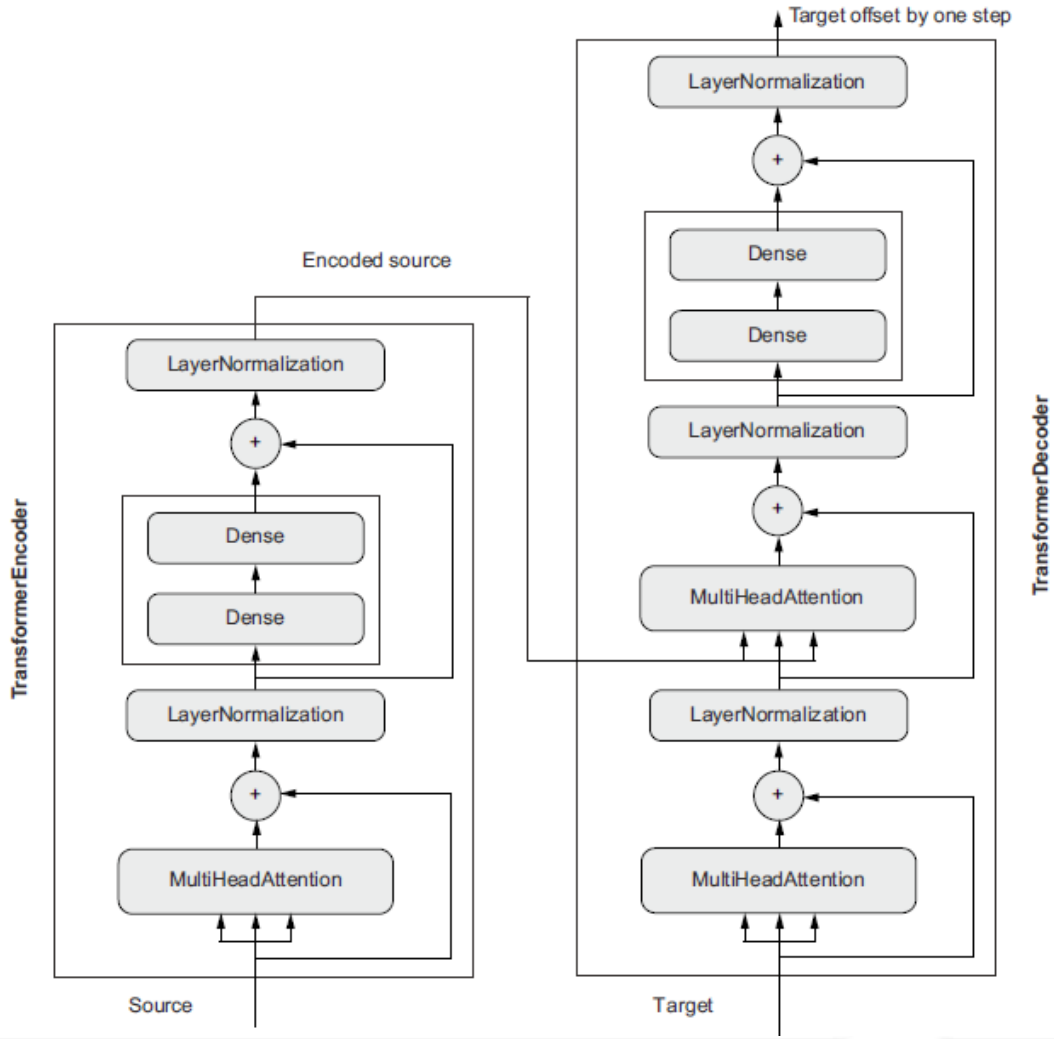


Transformers

- Designed for **sequence data**
  but process the entire input at once
  hence, more suitable for parallel computing than RNNs
- Apply the mechanism of **self-attention**
  + consider order in the data, but not as strict as RNNs do
  hence, more suitable for **natural language processing**
- Also used in computer vision

- Developed by Google Brain in 2017
- Examples: GPT, BERT, XLNet, RoBERTa, …
- See also: Hugging Face Transformers library





Generative deep learning models

- **Generative Adversarial Networks**

  = Discriminator + Generator

  e.g. StyleGAN, CycleGAN, …

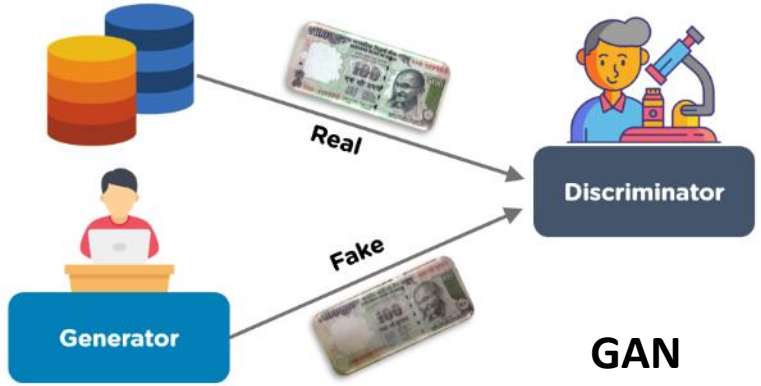
- **Variational Autoencoders**

  = Encoder + Decoder

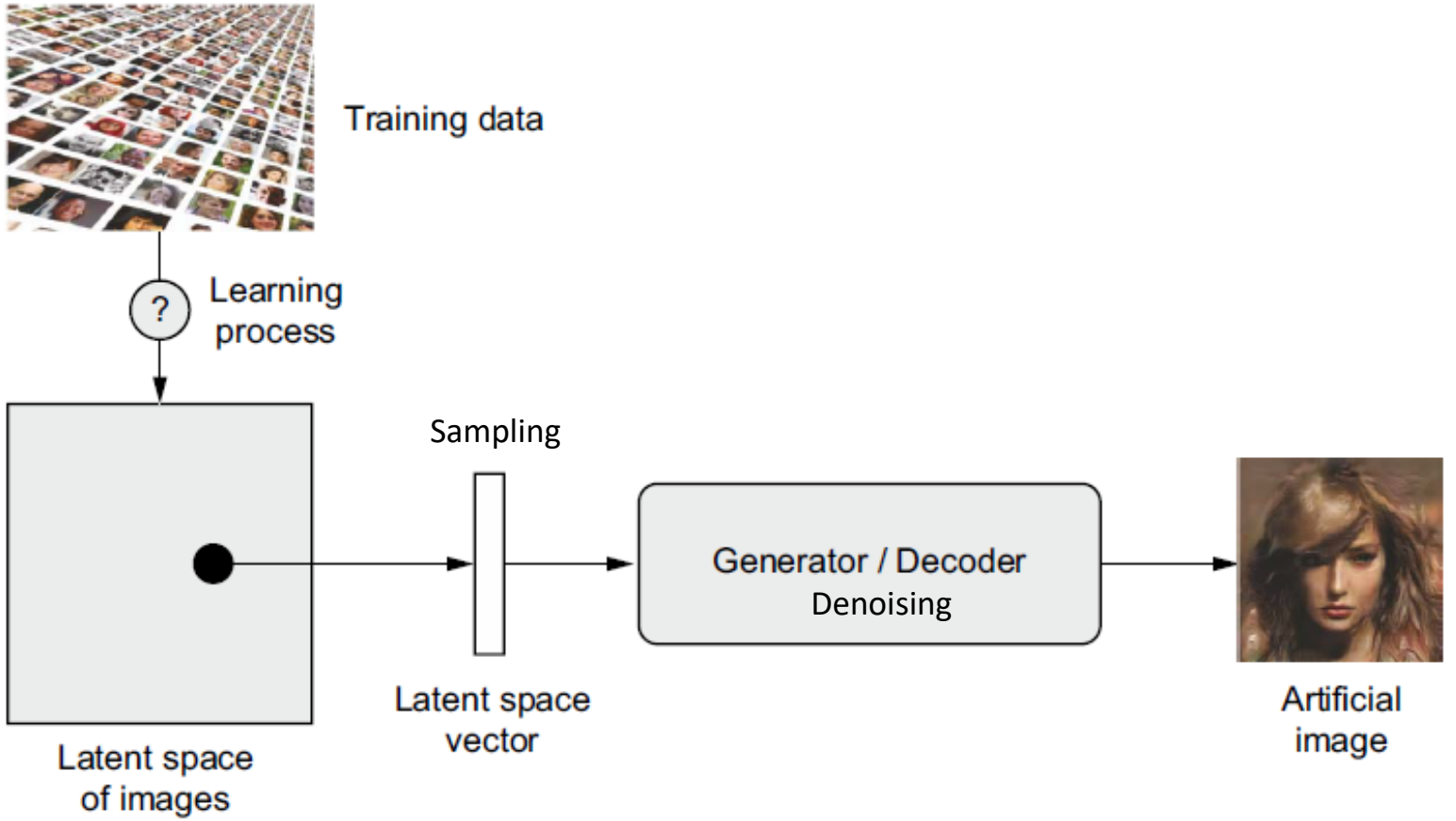
- **Diffusion models**

  = Noising + Denoising

  e.g. DALL-E, Stable Diffusion, …

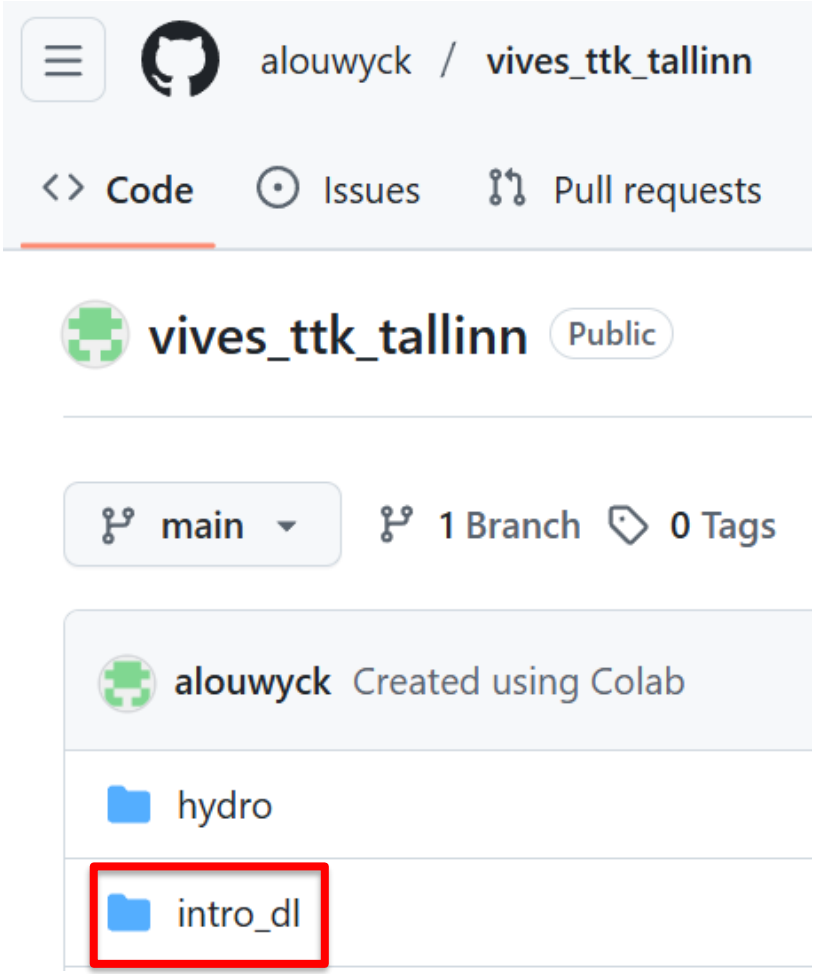


GAN



GitHub Repo

https://github.com/alouwyck/vives_ttk_tallinn



TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Sources

- Most slides are based on the book “Deep Learning with Python (2nd edtion)” by François Chollet (2021).
- Some slides are adopted from the presentation on deep learning that was part of the course “Introduction to Artificial Intelligence” given by Dr. Stefaan Haspeslagh at the Vives University of Applied Sciences during the academic year 2019-2020.
- Other sources are mentioned on the slides.