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

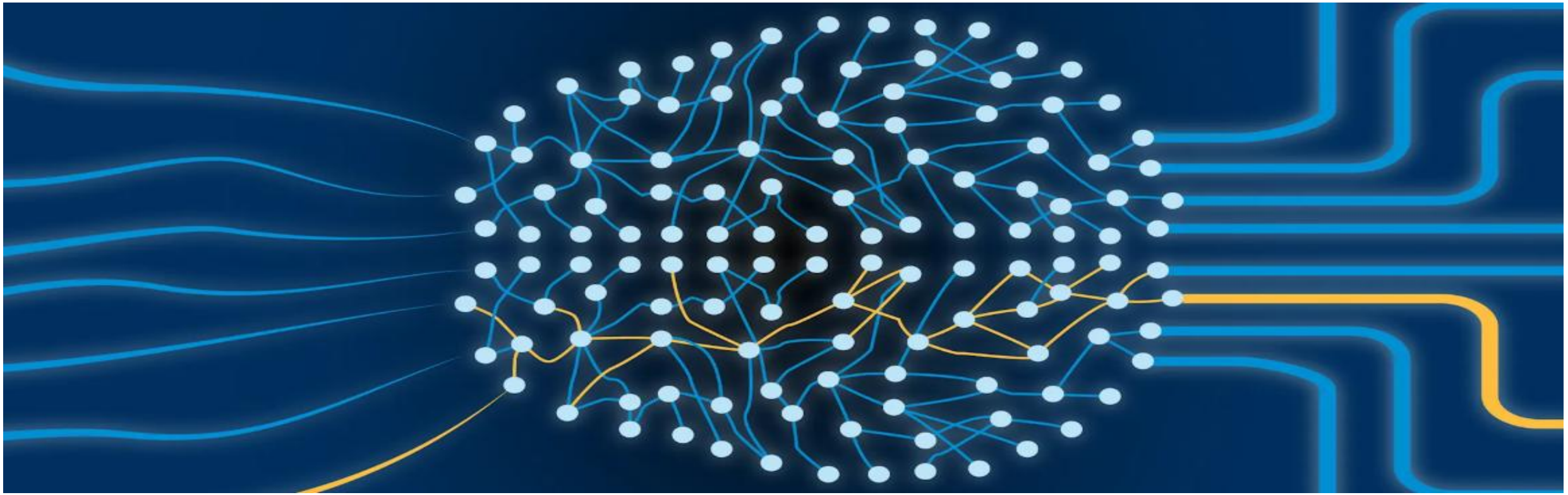
The Informatics-programme is a programme consisting of lectures, group work, visits and projects in the field of Business and Informatics. Evaluation follows the rules of the European Credit Transfer System (ECTS). Incoming students can select a programme of up to 30 ECTS credits per semester.

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	

Let’s Dive In!

1. What is deep learning?
2. What is a neural network?
3. Our first neural network
4. Tensors and tensor operations
5. Grandient-based optimization
6. Demo



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

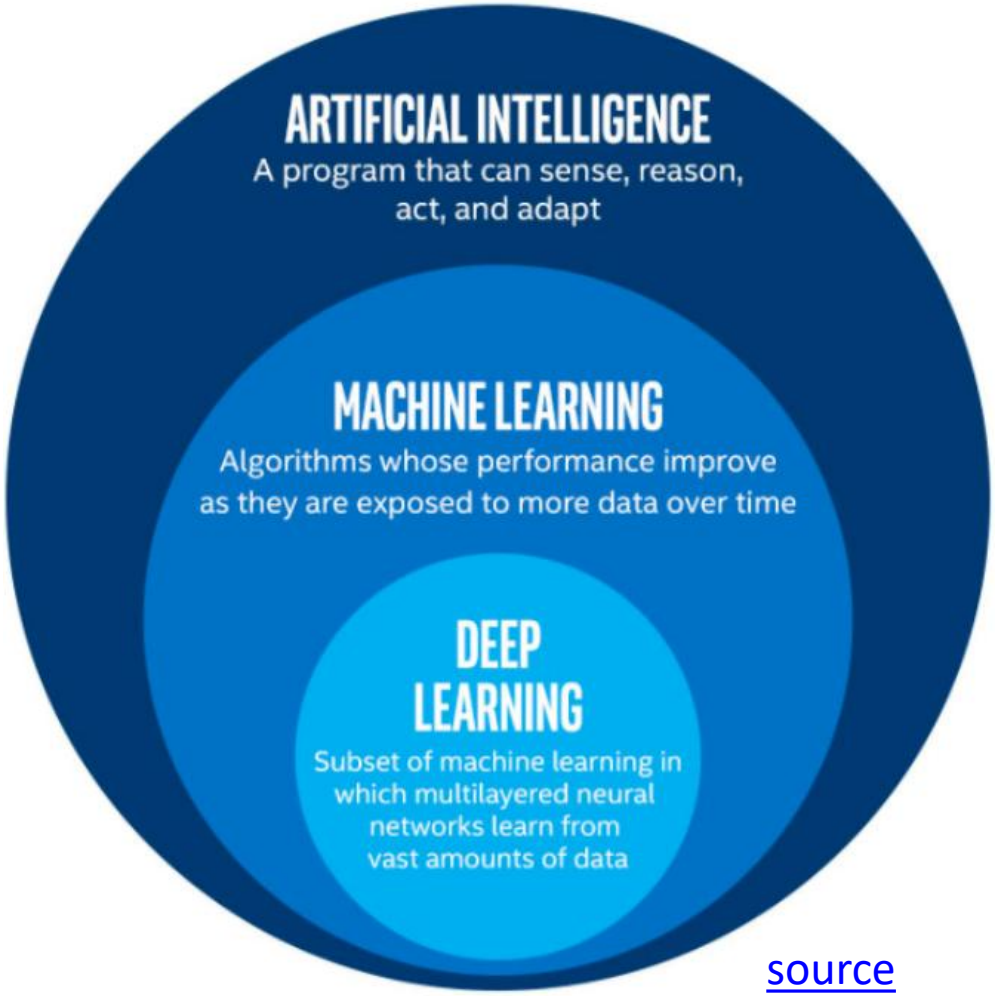
Introduction to Deep Learning

WHAT IS DEEP LEARNING?

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Deep Learning ⊂ Machine Learning ⊂ Artificial Intelligence



source

- **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*."

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

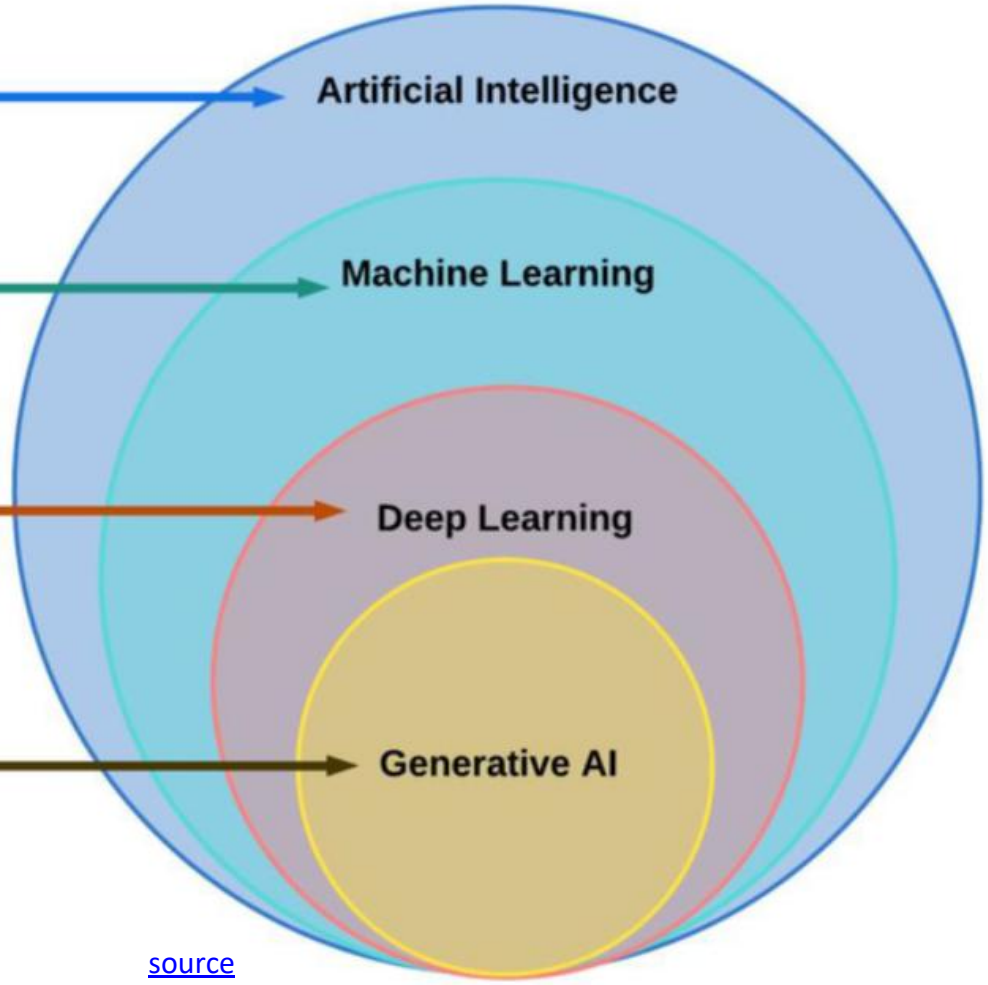
What About GenAI?

Artificial Intelligence
AI involves techniques that equip computers to emulate human behavior, enabling them to learn, make decisions, recognize patterns, and solve complex problems in a manner akin to human intelligence.

Machine Learning
ML is a subset of AI, uses advanced algorithms to detect patterns in large data sets, allowing machines to learn and adapt. ML algorithms use supervised or unsupervised learning methods.

Deep Learning
DL is a subset of ML which uses neural networks for in-depth data processing and analytical tasks. DL leverages multiple layers of artificial neural networks to extract high-level features from raw input data, simulating the way human brains perceive and understand the world.

Generative AI
Generative AI is a subset of DL models that generates content like text, images, or code based on provided input. Trained on vast data sets, these models detect patterns and create outputs without explicit instruction, using a mix of supervised and unsupervised learning.



source

Intuitive Explanation of Machine Learning

Example: buying a new car

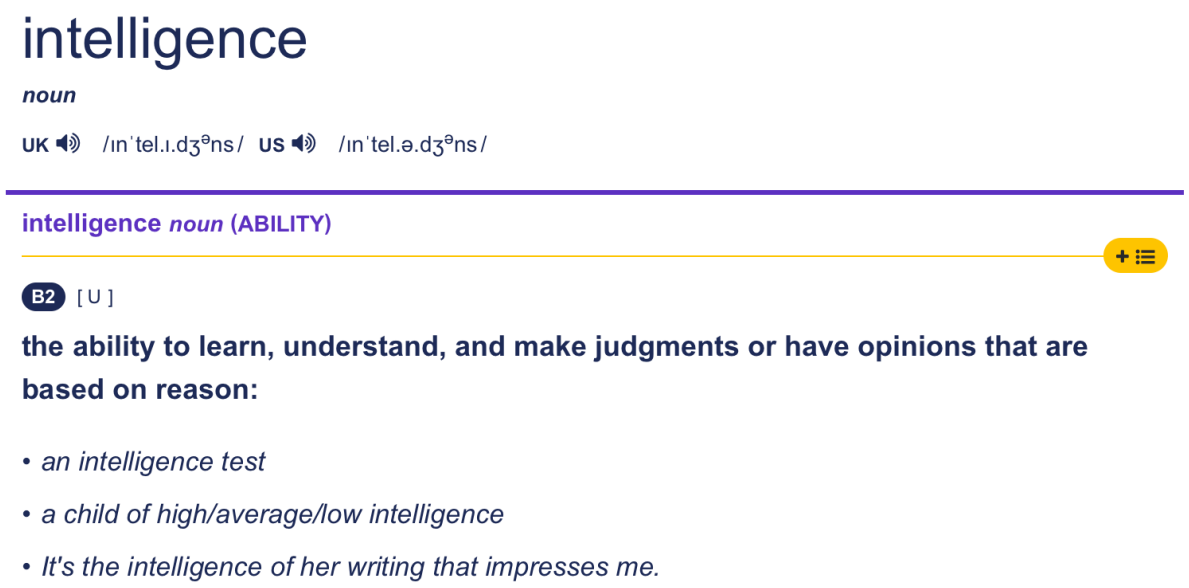
- How do we make decisions?
  - by logical **reasoning**
  - by relying on previous **experiences** (either our own or those of others)

- For a computer: **experiences = data**

"Machine learning is common sense, except done by a computer"

Formal Explanation of Machine Learning

- Core domain of AI, concerned with automatic learning



intelligence

noun

UK /ɪnˈtel.ɪ.dʒᵊns/ US /ɪnˈtel.ə.dʒᵊns/

intelligence *noun* (ABILITY)

B2 [U]

the ability to learn, understand, and make judgments or have opinions that are based on reason:

- *an intelligence test*
- *a child of high/average/low intelligence*
- *It's the intelligence of her writing that impresses me.*

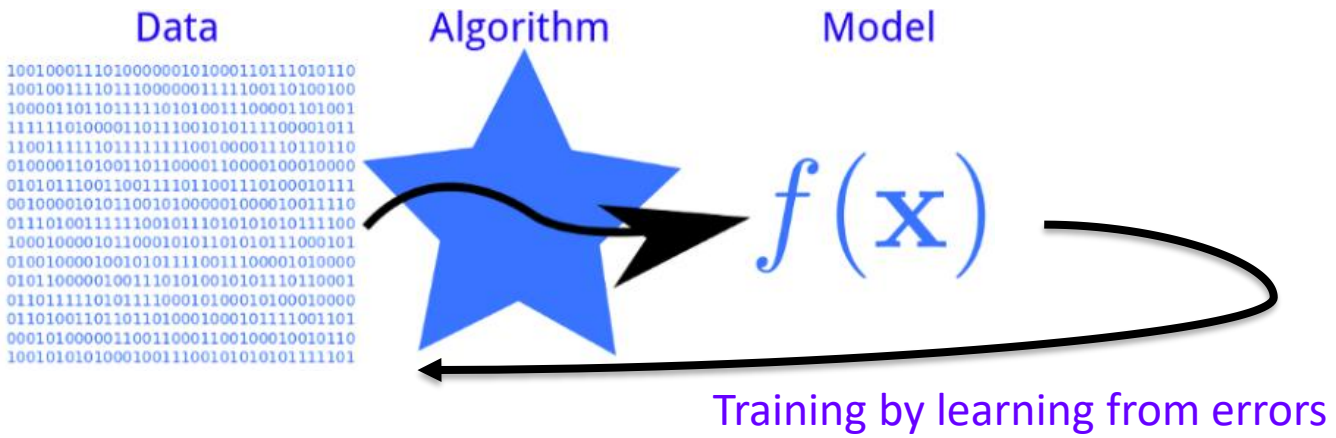
- A computer is said to be able to learn if its performance in solving some task improves with its experience

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Machine Learning: Algorithm versus Model

- **Algoritme**: A procedure, or a set of steps, used to build a model.
- **Model**: A set of rules that represent the data and can be used to make predictions.
- **Learning**: The process in which the algorithm improves the model by analyzing errors.

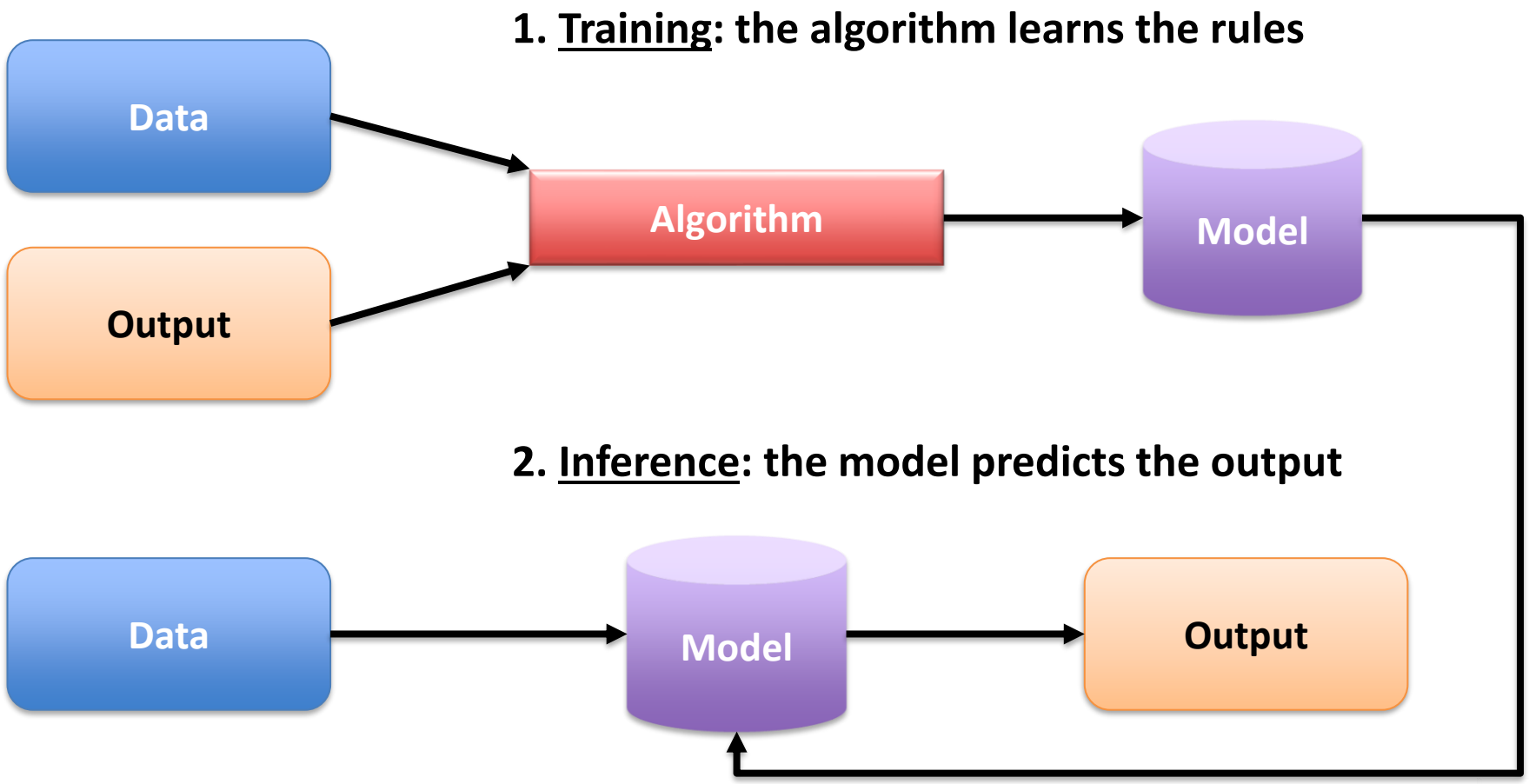


"An algorithm is run on the data to create a model.
By learning from errors, the model improves itself."

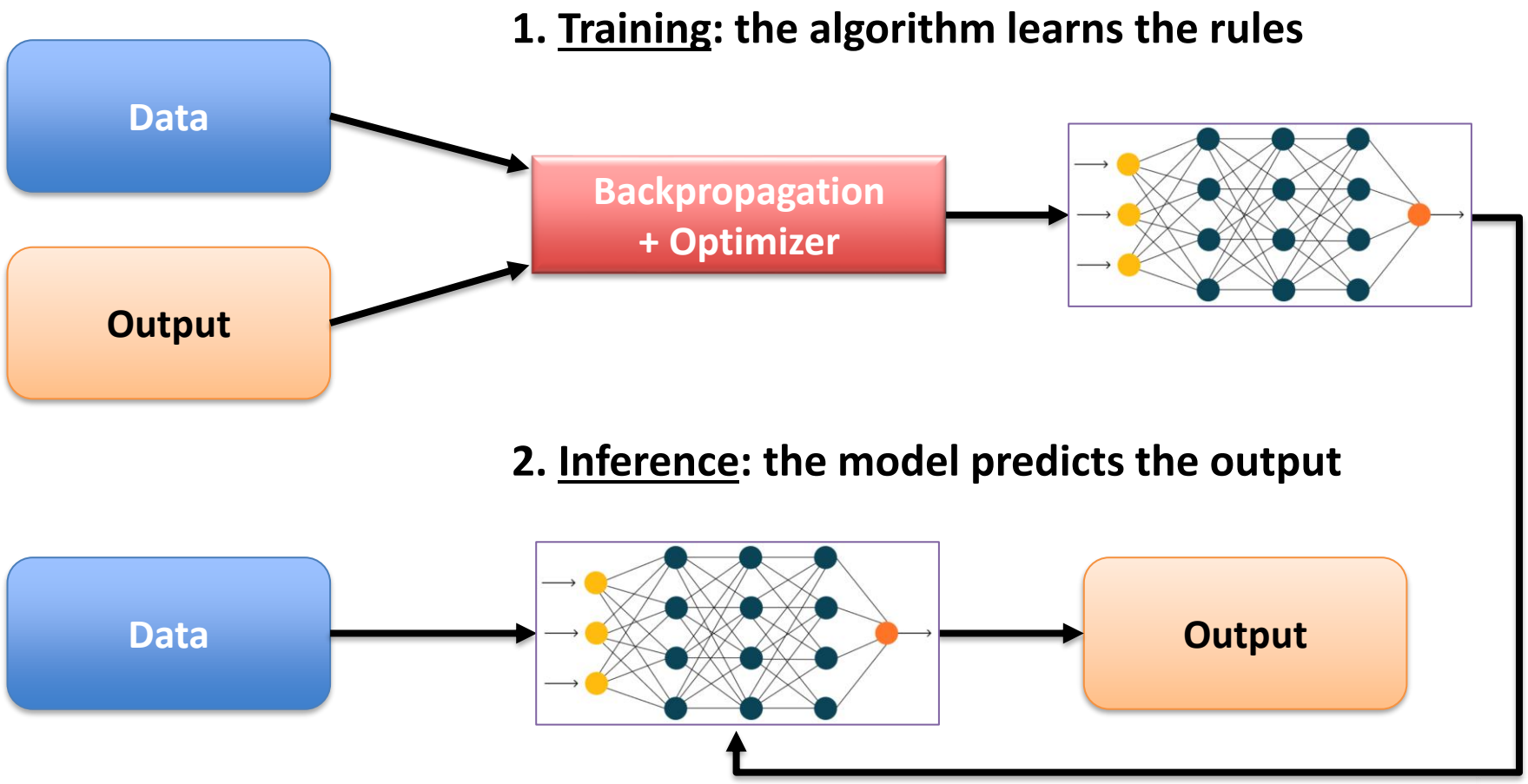
TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

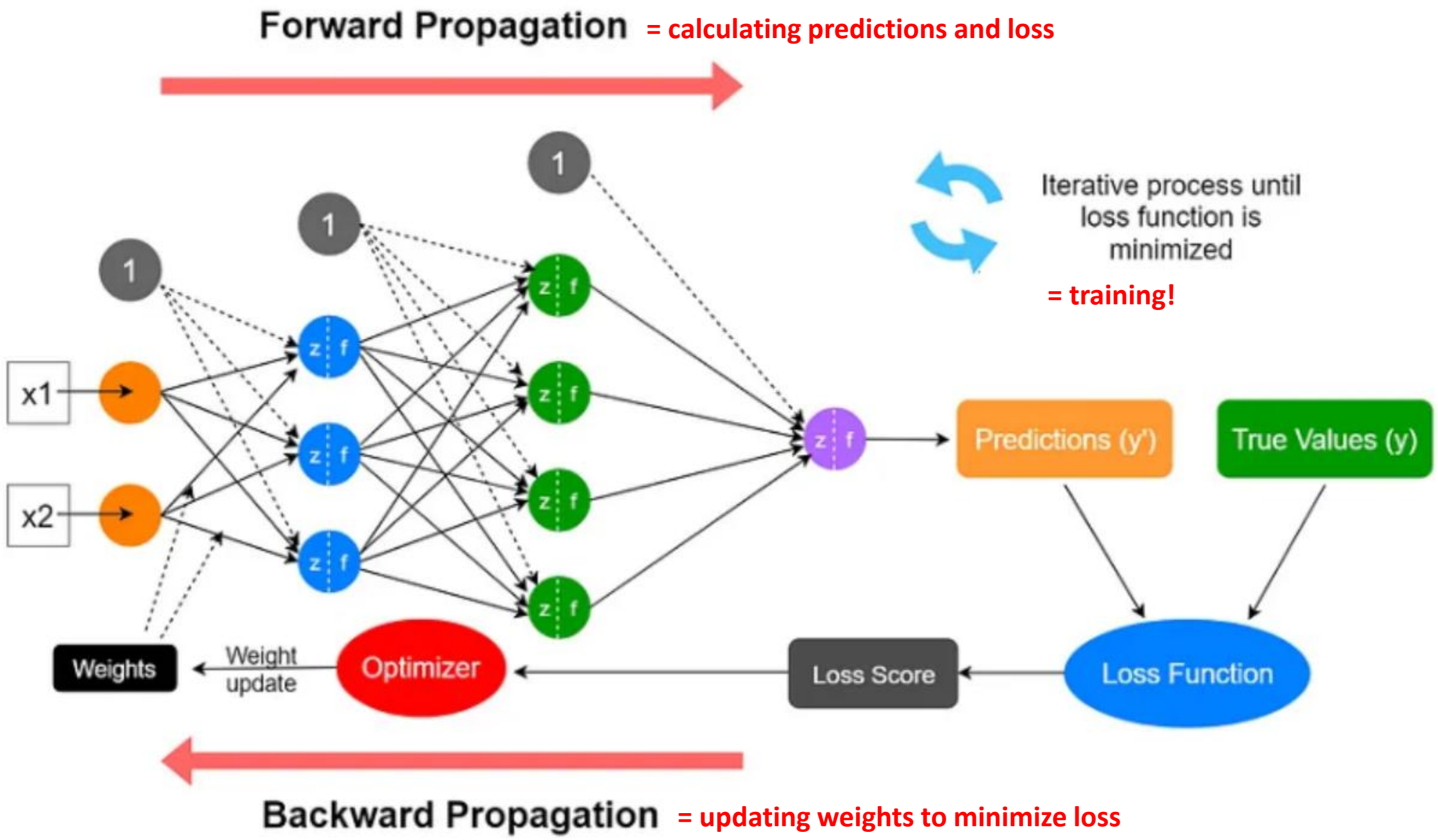
vives University of Applied Sciences

Machine Learning: Training versus Inference



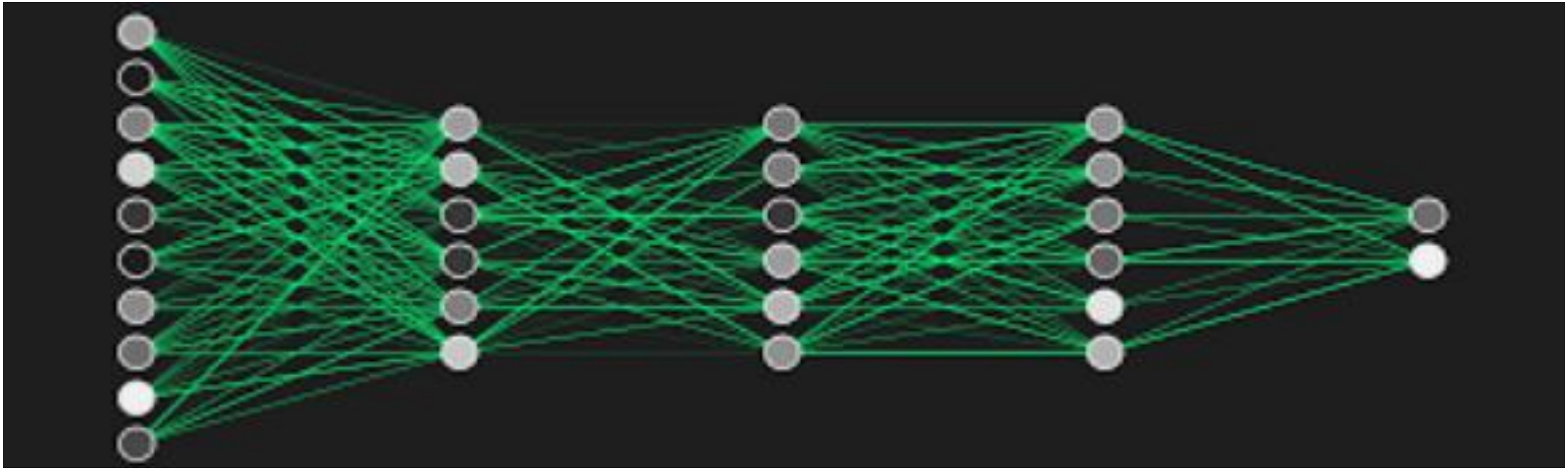
Deep Learning uses Artificial Neural Networks!




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

What is Deep Learning?

- Branch of machine learning that uses **artificial neural networks**
- **Inspired by** the functioning of the **human brain** (but not a simulation of it!)
- Requires **large amounts of data** and **high computational power**
- **Different types of neural networks**, each suitable for **specific applications**:
  - Speech recognition
  - Image recognition
  - Text translation
  - Object detection
  - Image generation
  - Text generation
  - …



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

Introduction to Deep Learning

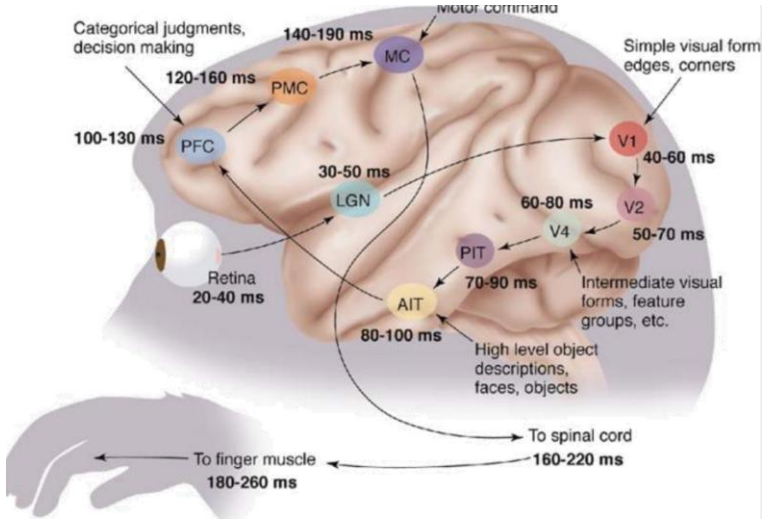
WHAT IS A NEURAL NETWORK?

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

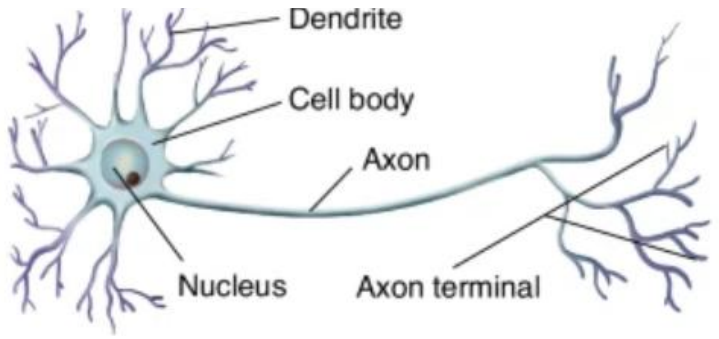
vives University of Applied Sciences

What is a Neural Network?

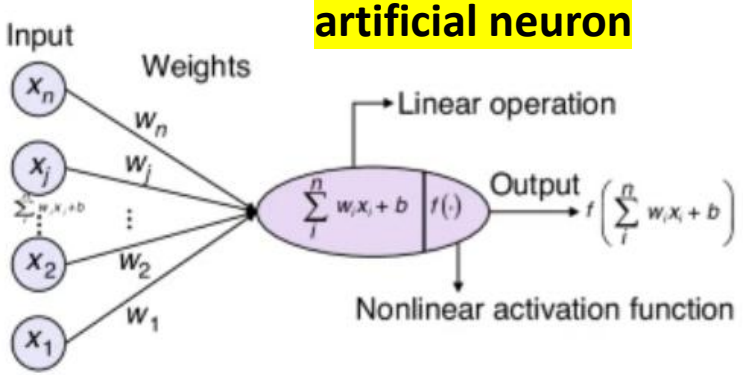
biological brain



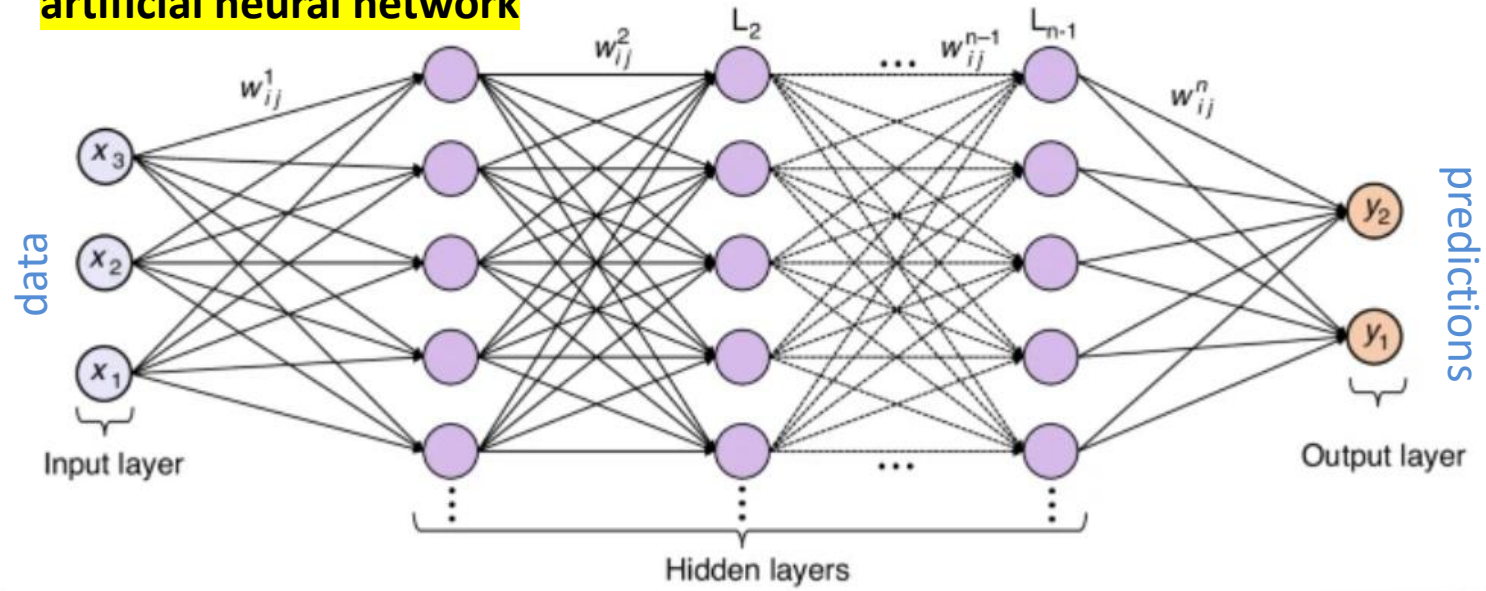
biological neuron



artificial neuron



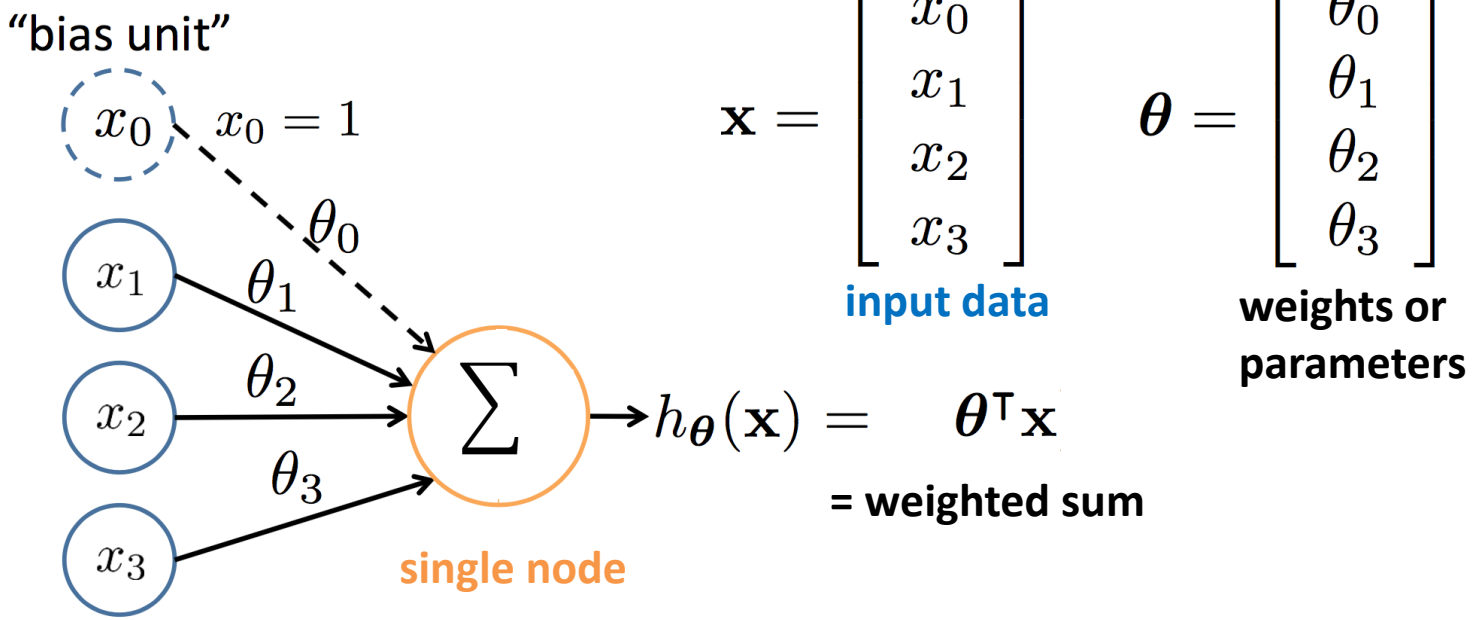
artificial neural network



TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

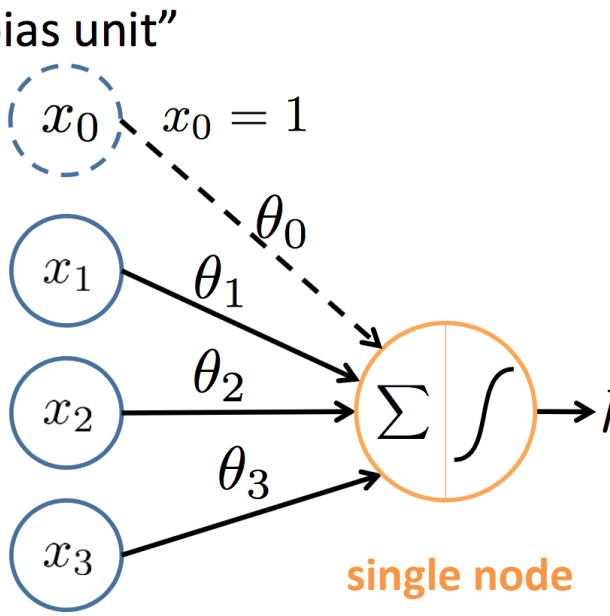
A Single Neuron

Neuron Model: Linear Unit



A Single Neuron with Activation Function

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**

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

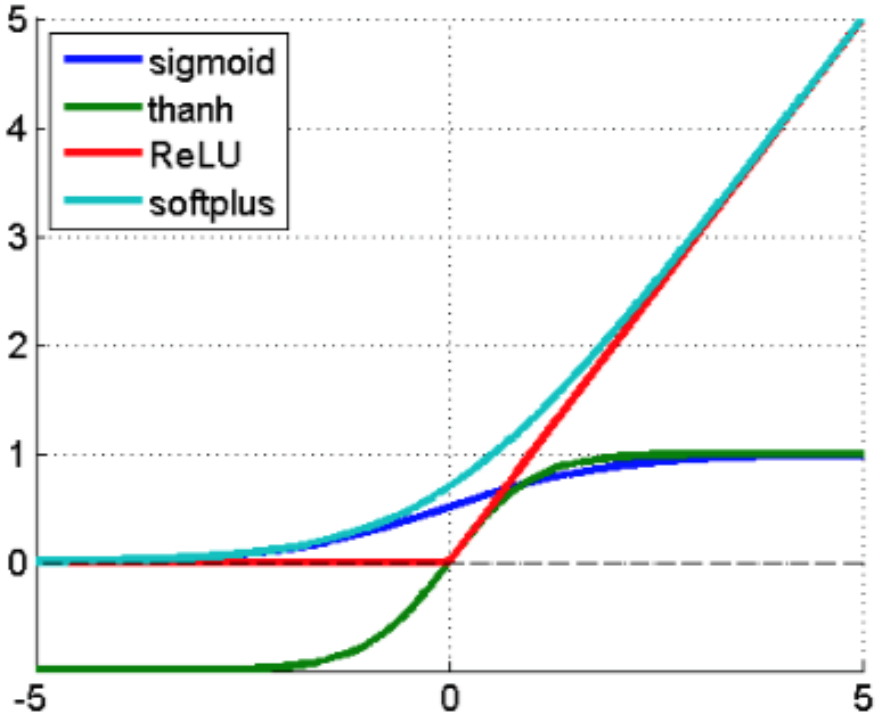
= activation of weighted sum

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

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

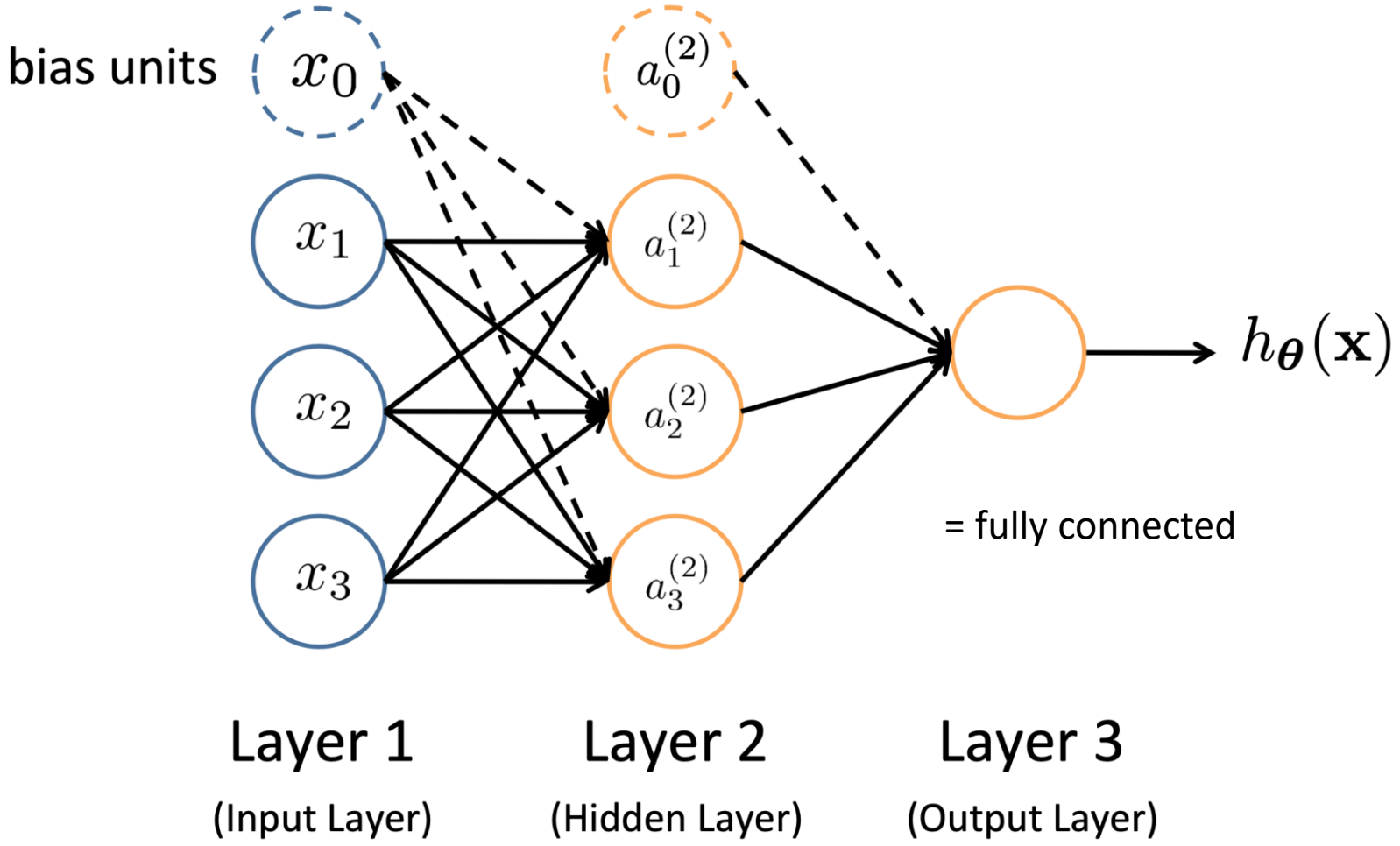
vives University of Applied Sciences

Different 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 Network with Multiple Neurons



DEEP learning = minimum 3 layers!

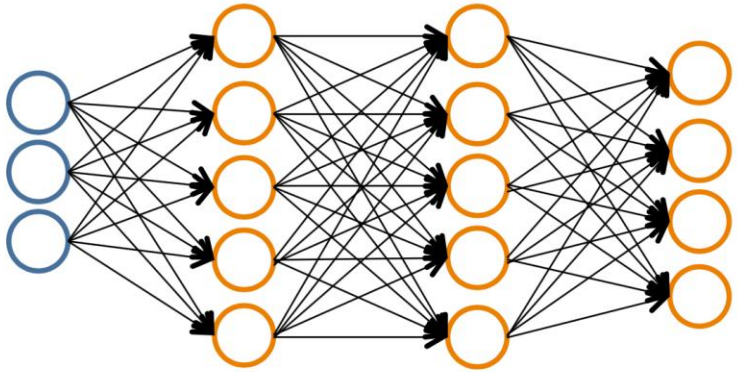
Image Classification: Example

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

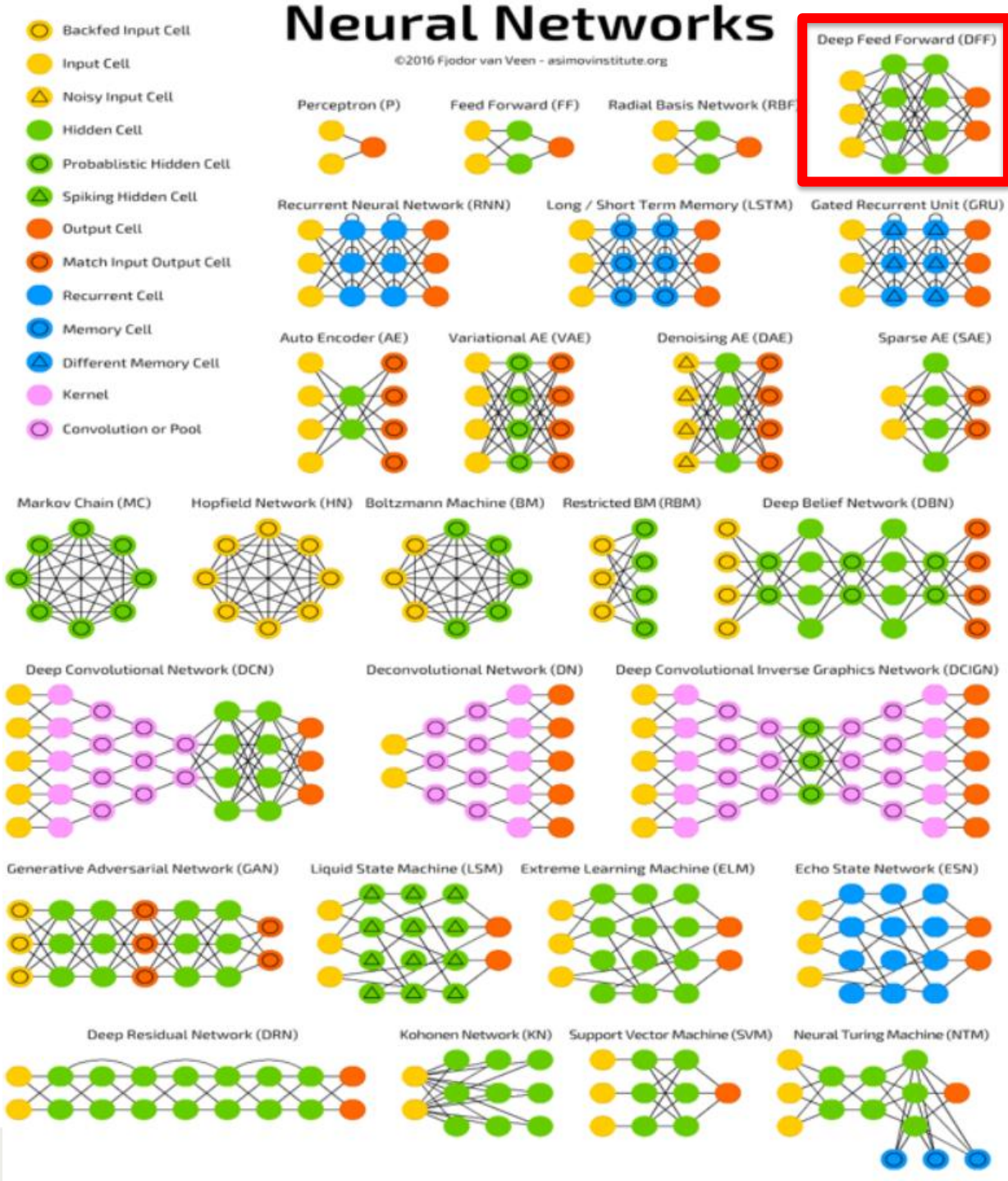
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

Network Architectures

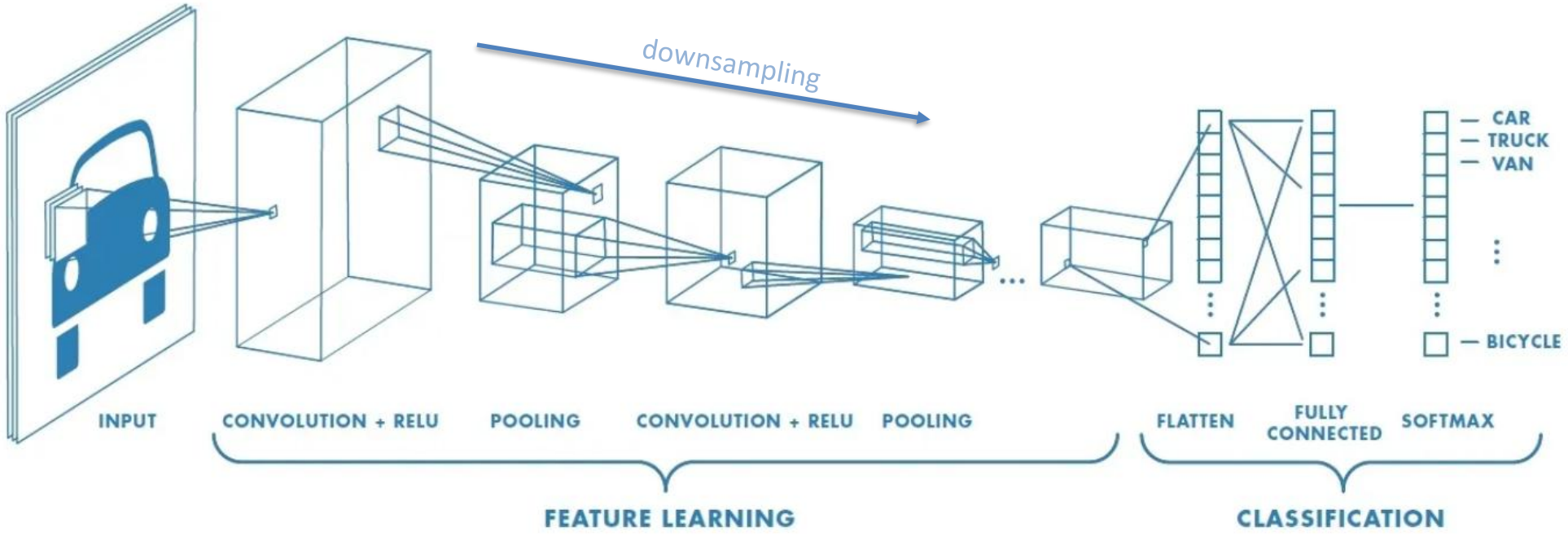
- So far we have only discussed
  (Deep) Feed Forward Networks (FFN)

- 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)
  - Diffusion models
  - …



Convolution Neural Network for Image Classification

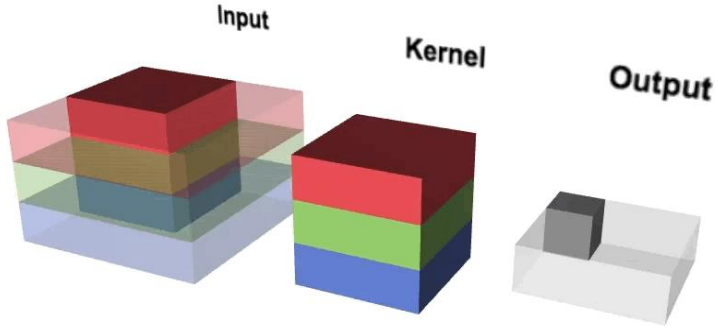
- Convnet = convolution base + classifier
- The **convolutional base** learns the features and consists of convolutional layers
- The **classifier** does the classification and consists of fully connected (dense) layers



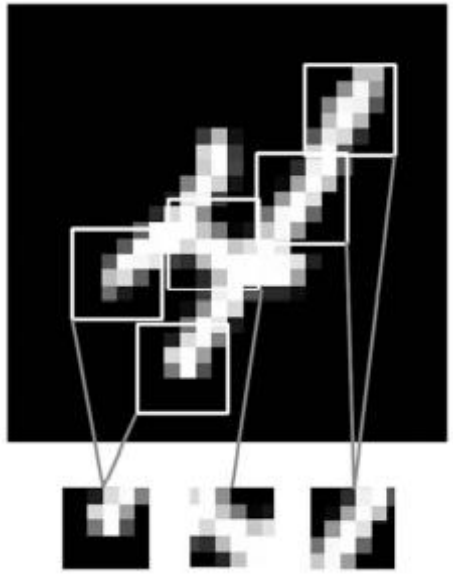
source

How Convolution Works

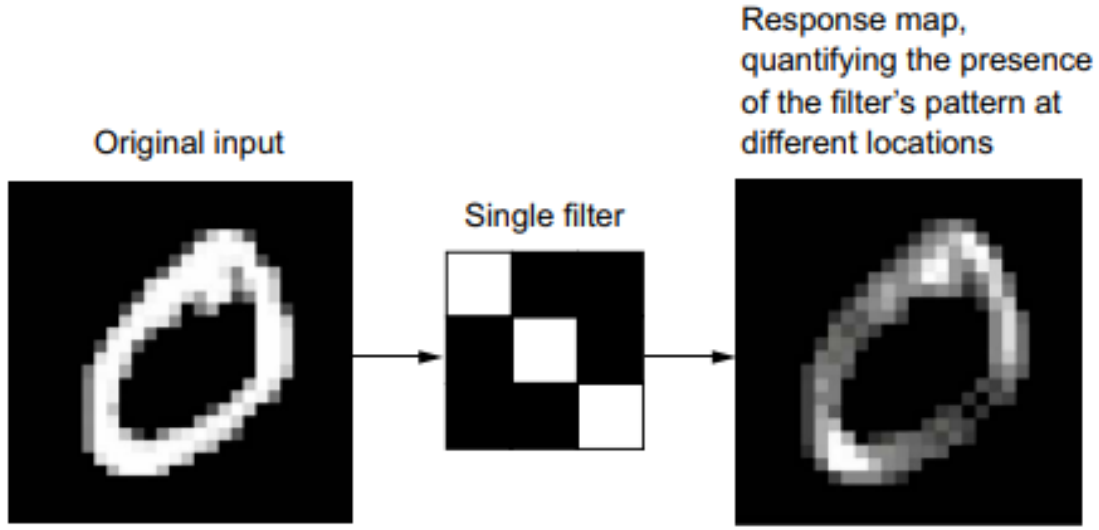
- A convolutional layer is a **feature detector**
- It ‘slides’ filters (= convolutional **kernels**) over the input image
- The result of ‘sliding’ a filter is a response **feature map**
- In this way convolutional layers learn **local patterns**
  by optimizing the filters during training



source

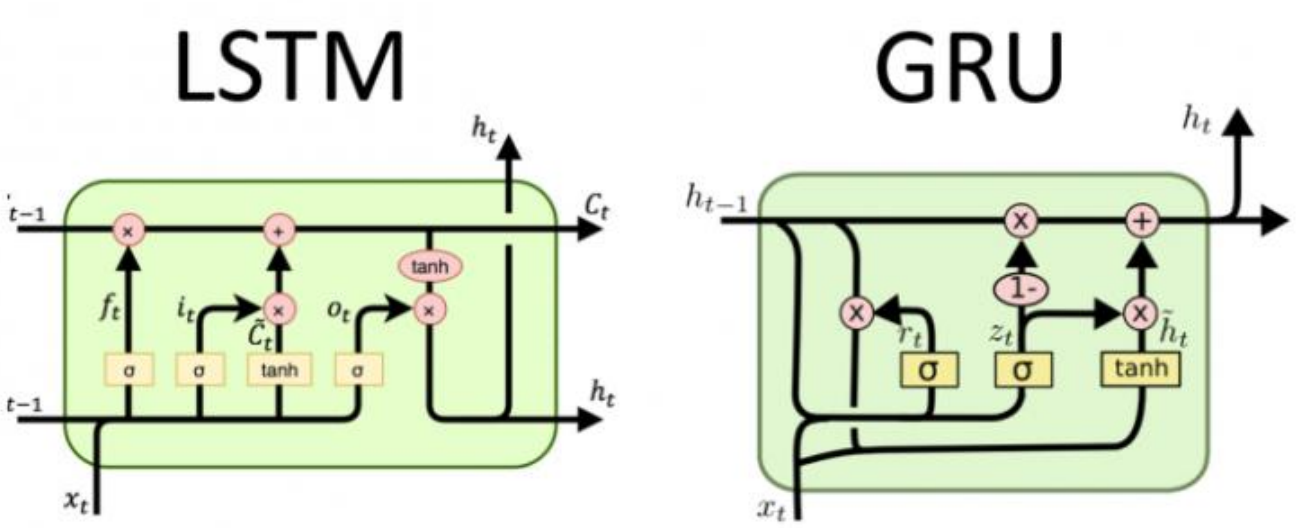


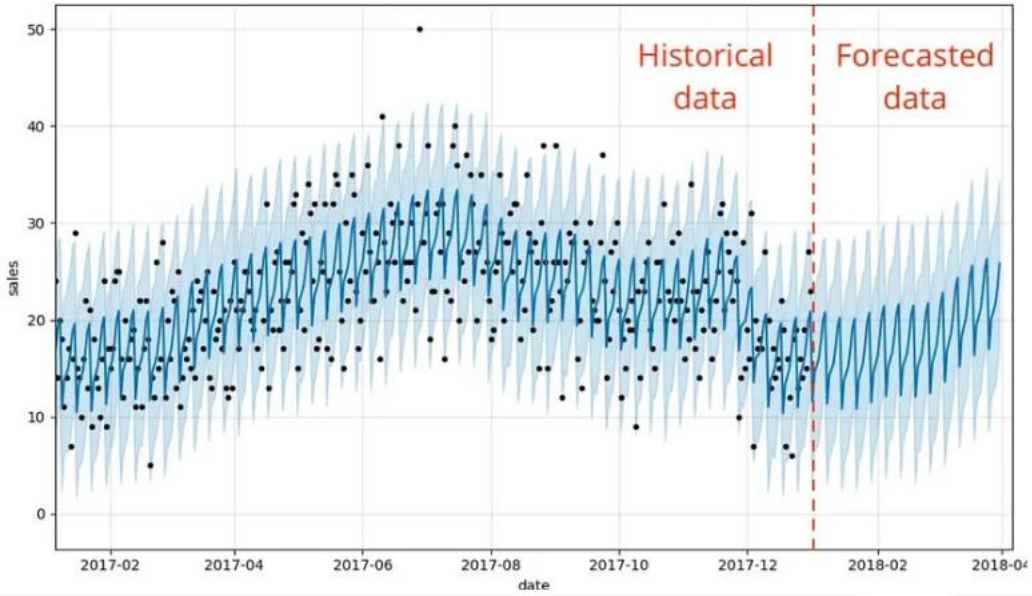
Learning local patterns



Neural Nets for Time Series Forecasting

- LSTM = Long Short-Term Memory
- GRU = Gated Recurrent Unit
- designed for **sequence data** (= data with strict order)
- suitable for tasks like **natural language processing** and **time series analysis**



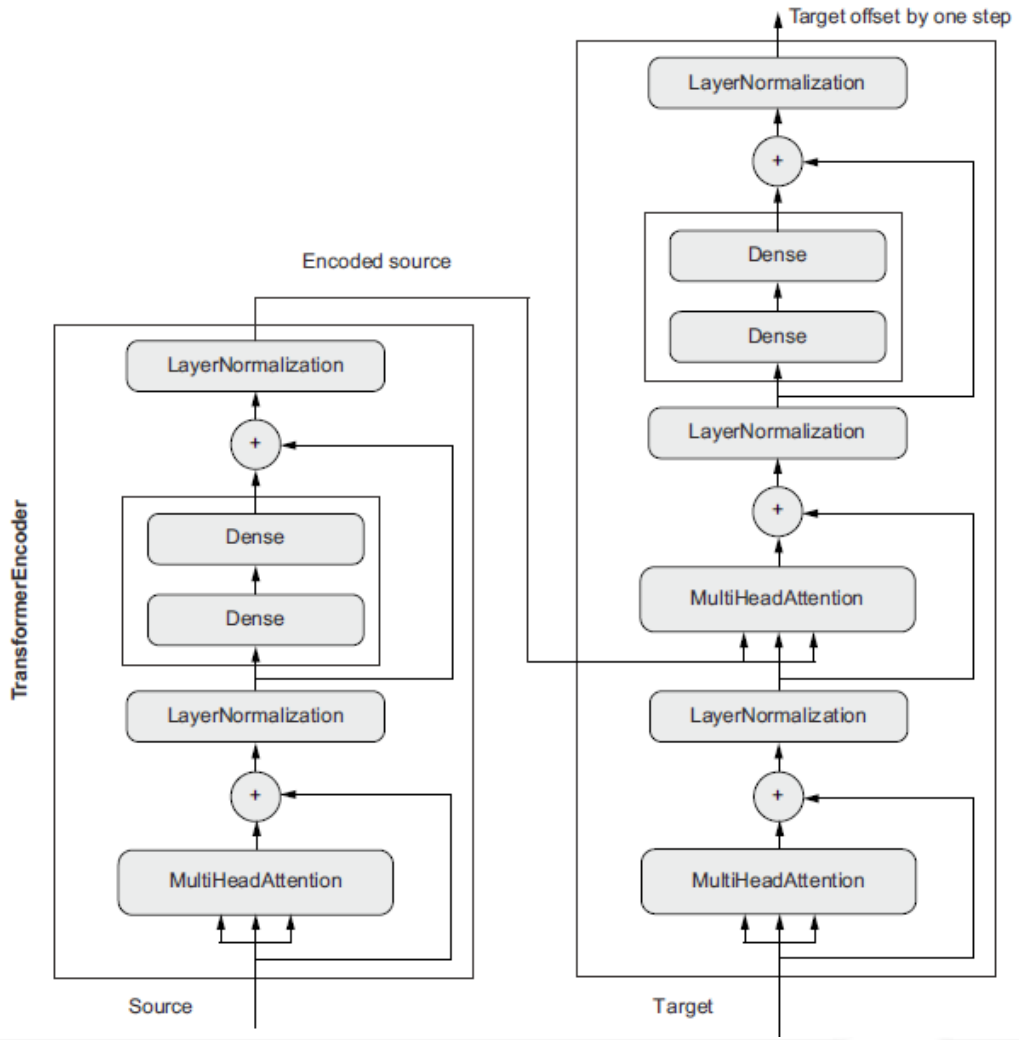


Transformers

Hugging Face

- 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

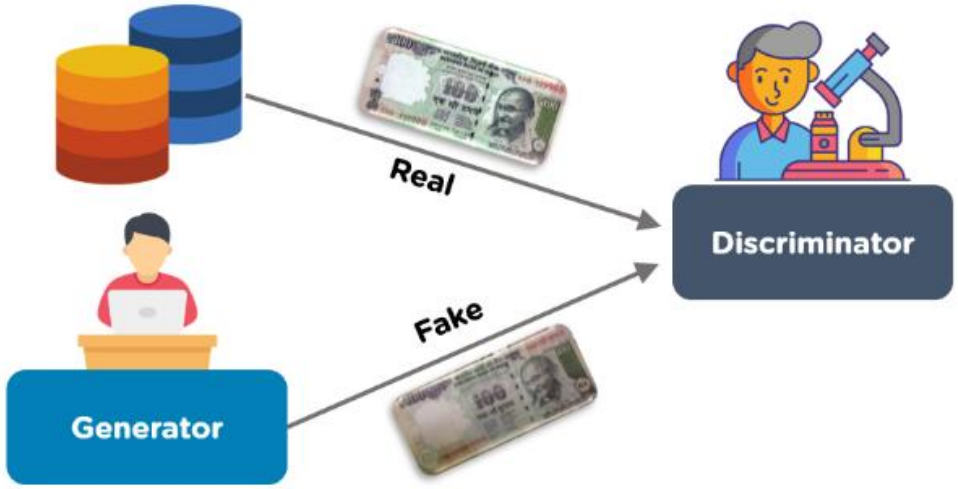


Neural Nets for Generating Images

- **Generative Adversarial Networks**

  = Discriminator + Generator

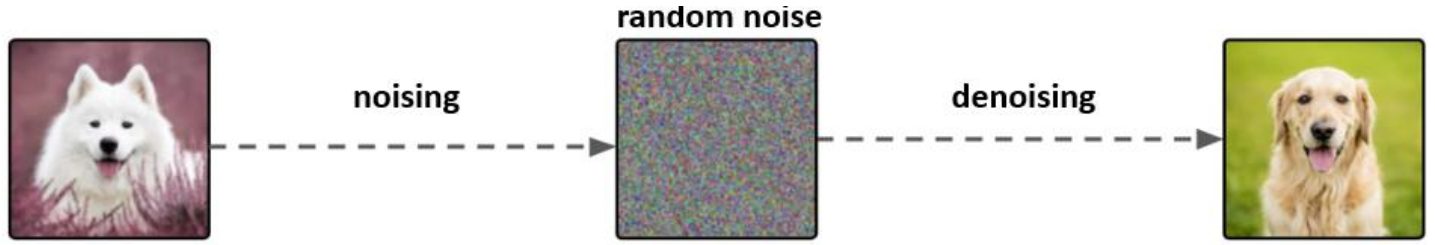
  e.g. StyleGAN, CycleGAN, …

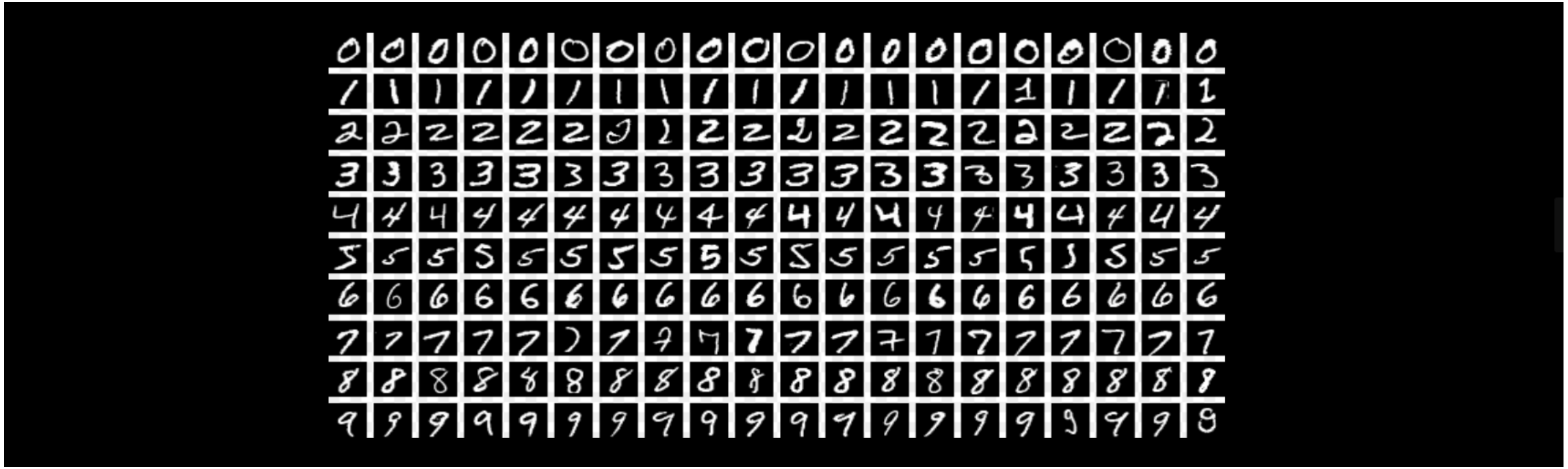


- **Diffusion models**

  = Noising + Denoising

  e.g. DALL-E, Stable Diffusion, …





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

Introduction to Deep Learning

OUR FIRST NEURAL NETWORK

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Our 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.

Our 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
```

Our 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

Our first neural network

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

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

Our 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

Our 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!

Our 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!

Our first neural network

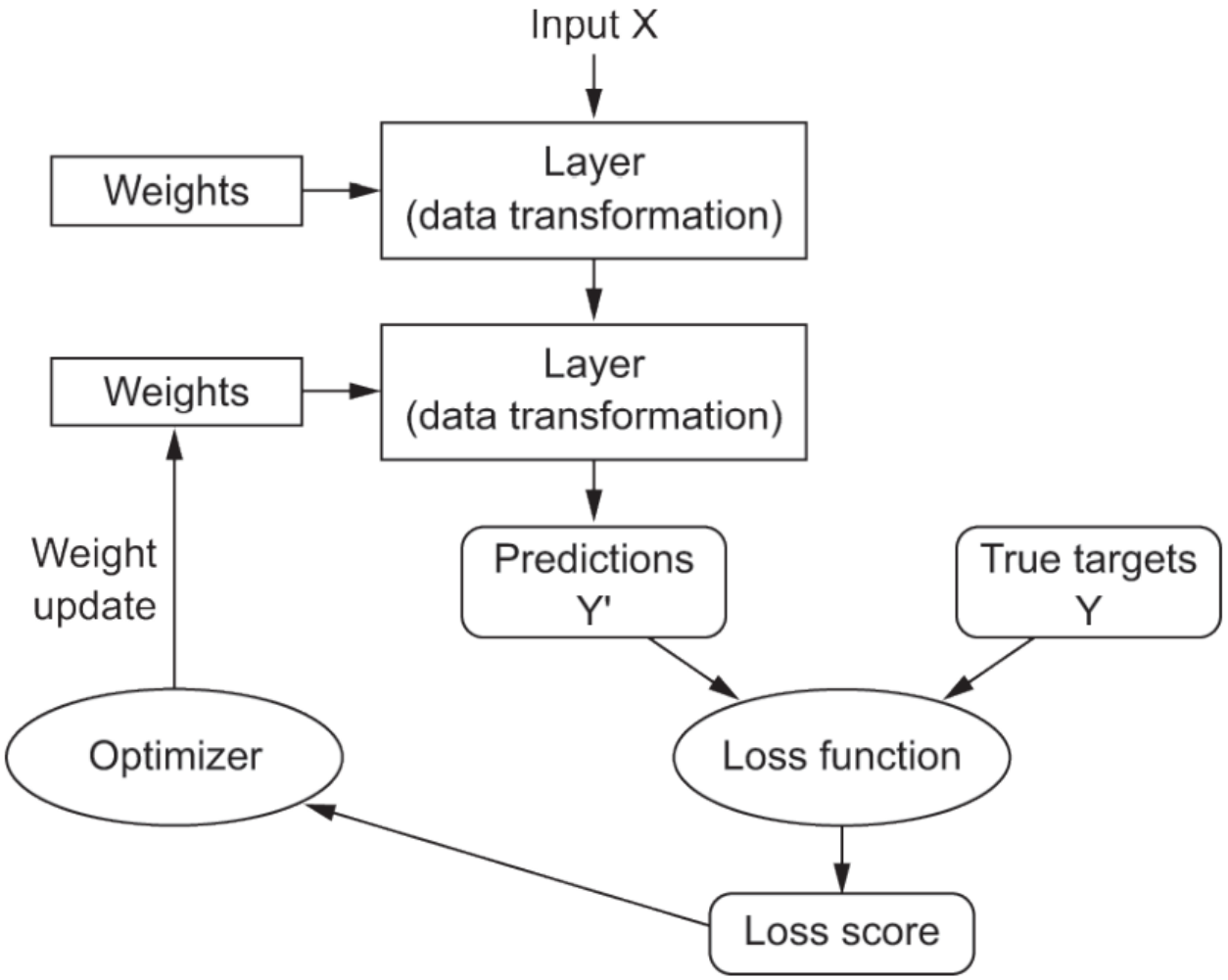
- **Evaluating** the model using the **testset**

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

Overfitting?

Overfitting happens when a machine learning model performs significantly better on the training data than on the validation or test data.

Our 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

TENSORS AND TENSOR OPERATIONS

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
```

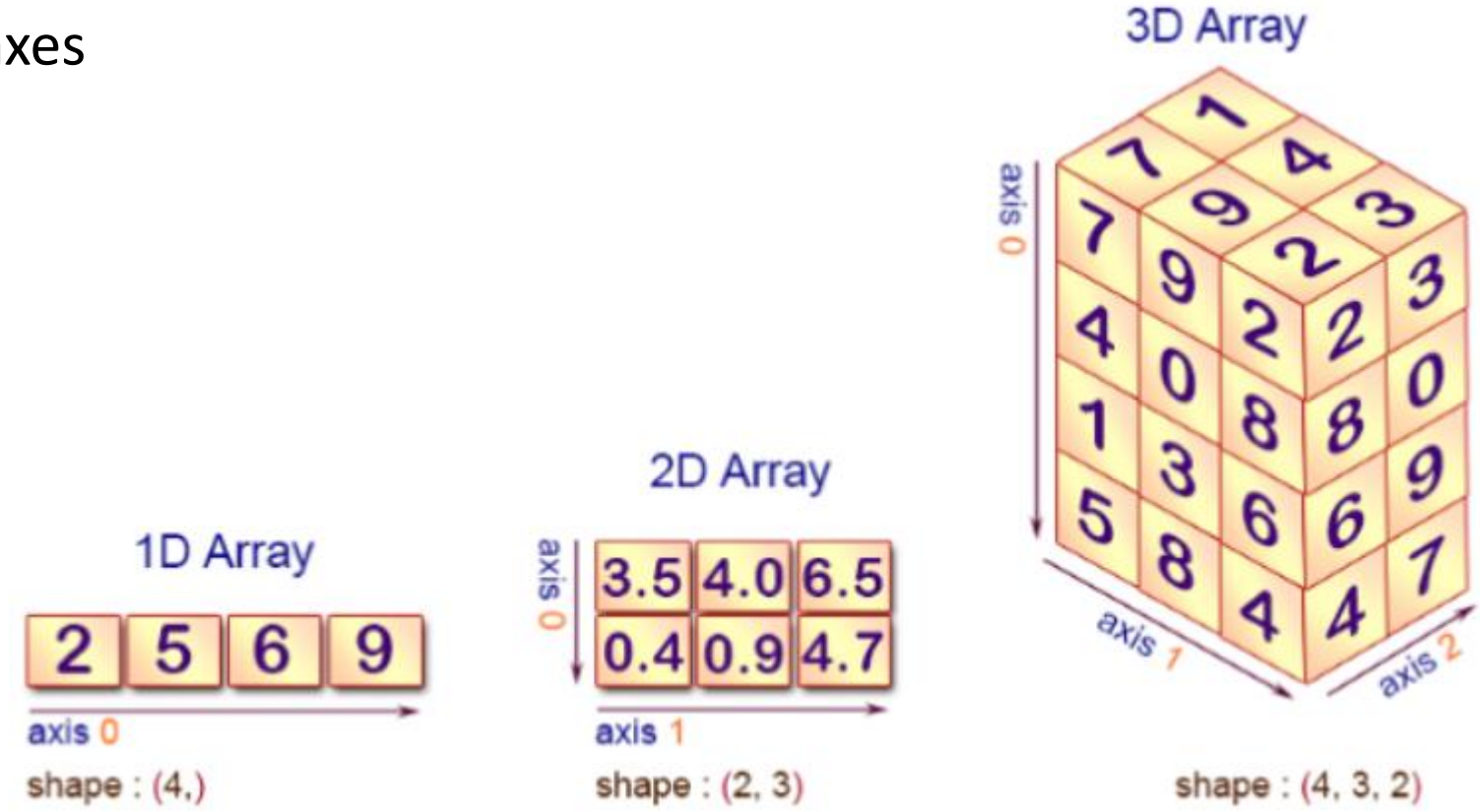
- **Multi-dimensional array** = Rank-3 or higher tensor (N axes with N >= 3)

```
>>> 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 = Numerical Python = Library for fast numerical computing with arrays

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
```

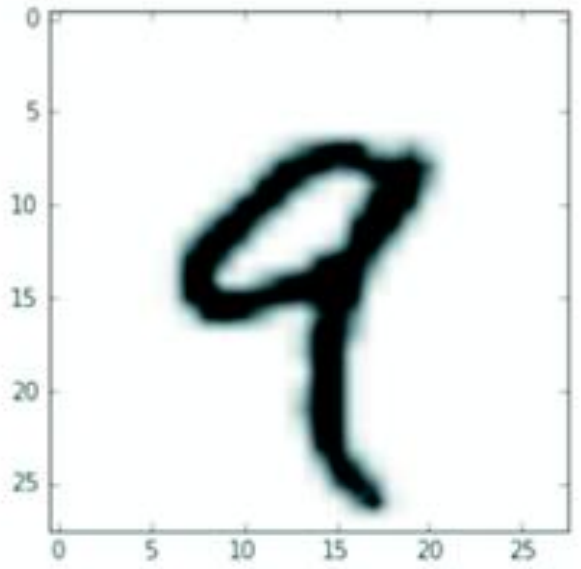
Here's its shape:

```
>>> 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

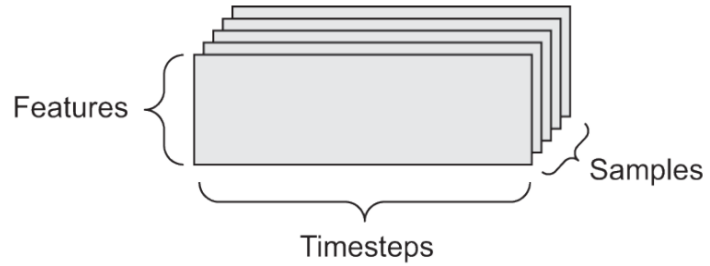
- **Vector data**: (samples, feature)
  - Tables and matrices (e.g. csv-files)
- **Time series**: (samples, timesteps, features)
  - E.g. dataset of stock prices, tweets, …
- **Images**: (samples, width, height, channels)
  - E.g. MNIST dataset
- **Videos**: (samples, frames, height, width, channels)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

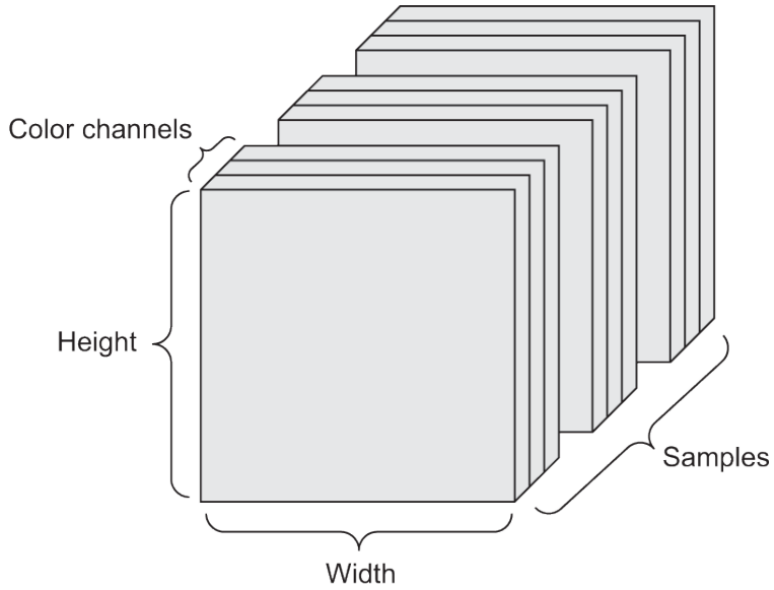
data(set)

	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

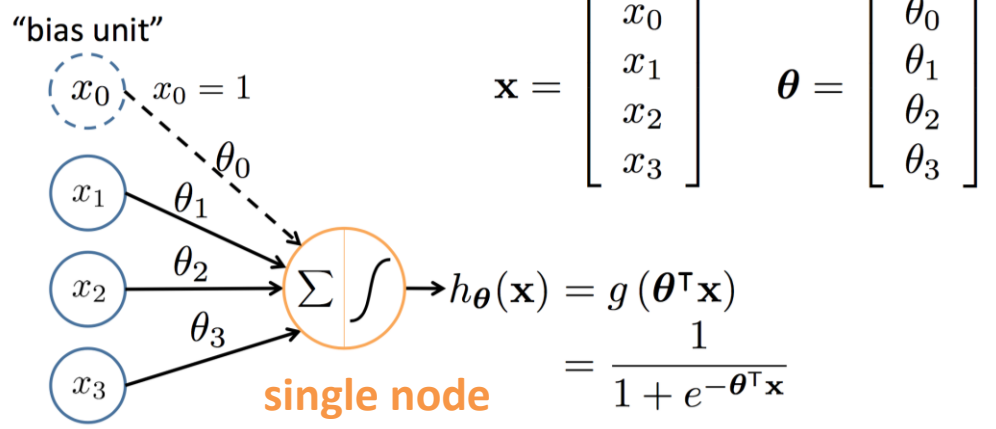




Tensor operations

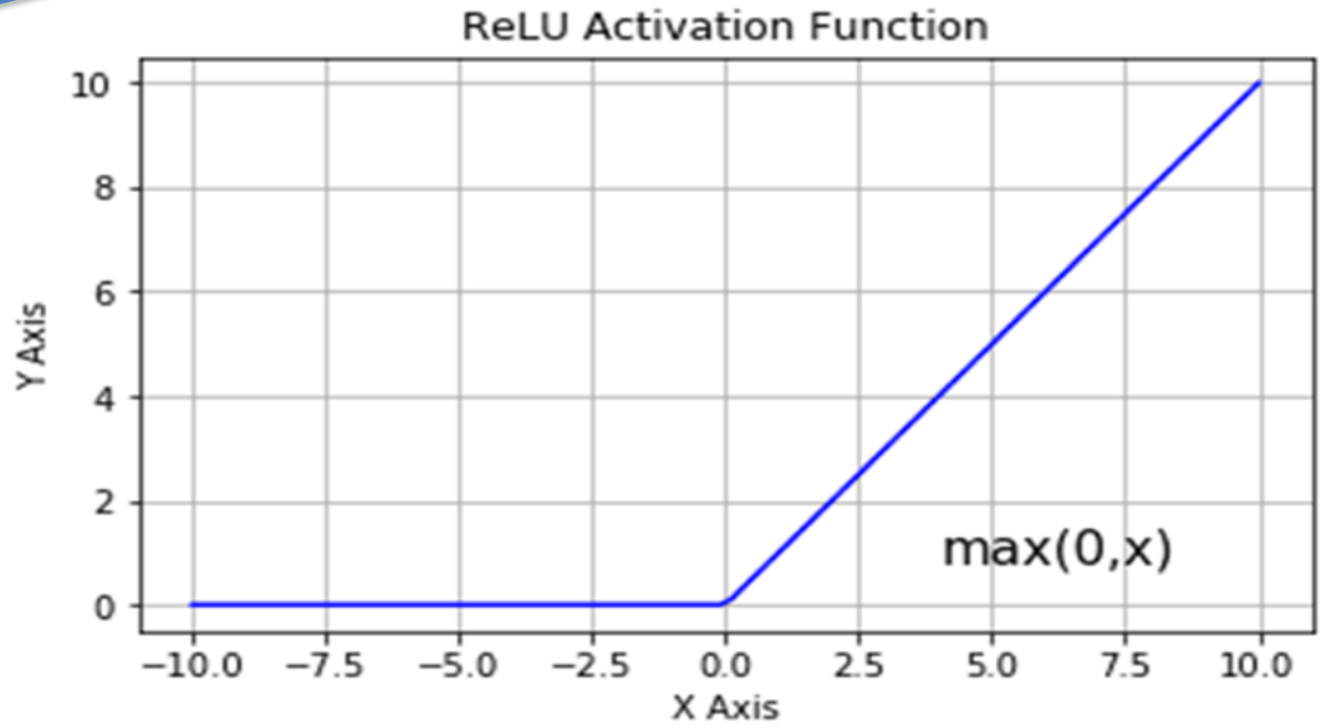
```
keras.layers.Dense(512, activation="relu")
```

- "fully-connected" or "dense" layer
- essentially represents tensor computations




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

- dot = matrix multiplication between tensor **W** en tensor **input**
- plus: sum of tensors
- ReLU(x) = max(x, 0)



TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

Tensor operations: element-wise operations

= Operations that are performed on each element individually

- Can be highly parallelized: vectorization, GPU!
- Example: ReLU and addition

```
def naive_relu(x):
    assert len(x.shape) == 2
    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] = max(x[i, j], 0)
    return x
```

```
def naive_add(x, y):
    assert len(x.shape) == 2
    assert x.shape == y.shape
    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] += y[i, j]
    return x
```

- Corresponding vectorized expressions in NumPy:

```
import numpy as np
z = x + y
z = np.maximum(z, 0.)
```

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Tensor operations: element-wise operations

Vectorized NumPy expressions are much faster!

```
import numpy as np
from time import time

n = 1_000
x = np.random.randn(n, n)
y = np.random.randn(n, n)
```

```
print("RELU")

start = time()
naive_relu(x)
print("loops:", time() - start, "sec")

start = time()
np.maximum(x, 0.)
print("vectorized:", time() - start, "sec")
```

```
print("ADD")

start = time()
naive_add(x, y)
print("loops:", time() - start, "sec")

start = time()
x + y
print("vectorized:", time() - start, "sec")
```

```
RELU
loops: 0.7734863758087158 sec
vectorized: 0.0076906681060791016 sec
ADD
loops: 0.5160806179046631 sec
vectorized: 0.0032694339752197266 sec
```

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

Tensor operations: dot product between two vectors

The dot product of two vectors **a** = $[a_1, a_2, \cdots, a_n]$ and **b** = $[b_1, b_2, \cdots, b_n]$ is defined as

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

```
x = np.random.random((32,))
y = np.random.random((32,))
z = np.dot(x, y)
```

```
def naive_vector_dot(x, y):
    assert len(x.shape) == 1
    assert len(y.shape) == 1
    assert x.shape[0] == y.shape[0]
    z = 0.
    for i in range(x.shape[0]):
        z += x[i] * y[i]
    return z
```

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Tensor operations: dot product between matrix and vector

```
def naive_matrix_vector_dot(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 1
    assert x.shape[1] == y.shape[0]
    z = np.zeros(x.shape[0])
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            z[i] += x[i, j] * y[j]
    return z
```

```
def naive_matrix_vector_dot(x, y):
    z = np.zeros(x.shape[0])
    for i in range(x.shape[0]):
        z[i] = naive_vector_dot(x[i, :], y)
    return z
```

= Dot product between each row of the matrix and the vector

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

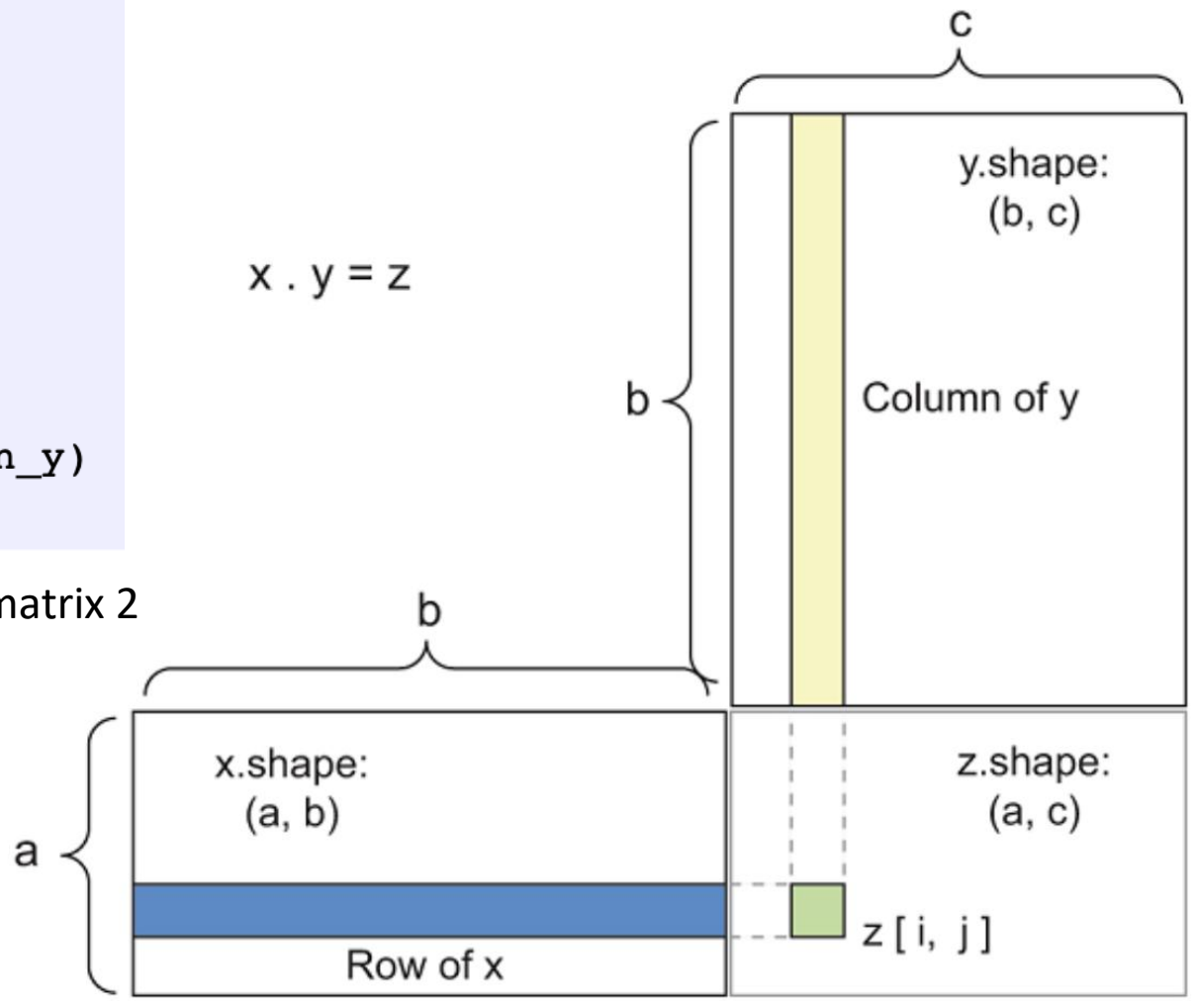
vives University of Applied Sciences

Tensor operations: dot product between two matrices

```
def naive_matrix_dot(x, y):
    assert len(x.shape) == 2                          1
    assert len(y.shape) == 2                          1
    assert x.shape[1] == y.shape[0]                   2
    z = np.zeros((x.shape[0], y.shape[1]))            3
    for i in range(x.shape[0]):                       4
        for j in range(y.shape[1]):                   5
            row_x = x[i, :]
            column_y = y[:, j]
            z[i, j] = naive_vector_dot(row_x, column_y)
    return z
```

= Dot product between each row of matrix 1 and each column of matrix 2

= matrix multiplication!



TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

Tensor reshaping

A third type of tensor operation that's essential to understand is *tensor reshaping*. Although it wasn't used in the `Dense` layers in our first neural network example, we used it when we preprocessed the digits data before feeding it into our model:

```
train_images = train_images.reshape((60000, 28 * 28))
```

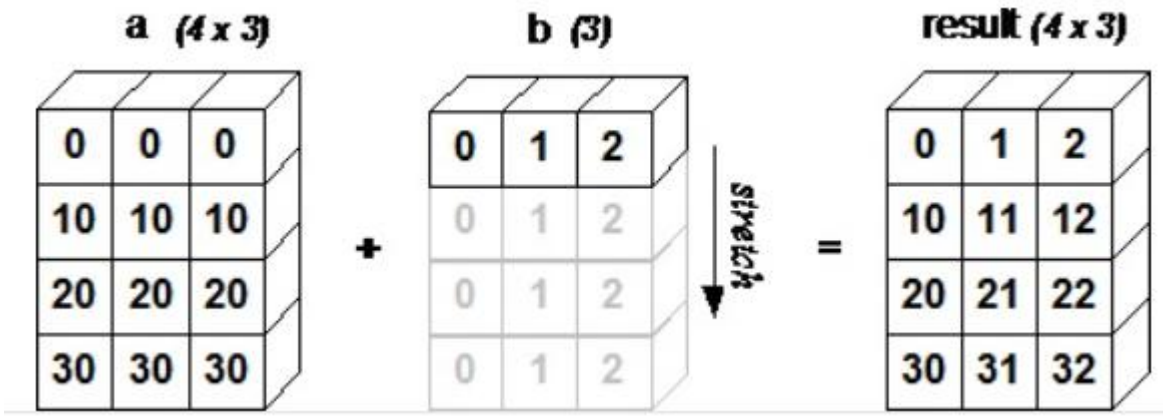
```
>>> x = np.array([[0., 1.],
                  [2., 3.],
                  [4., 5.]])
>>> print(x.shape)
(3, 2)
>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])
 >>> x = x.reshape((2, 3))
 >>> x
 array([[ 0.,  1.,  2.],
        [ 3.,  4.,  5.]])
```

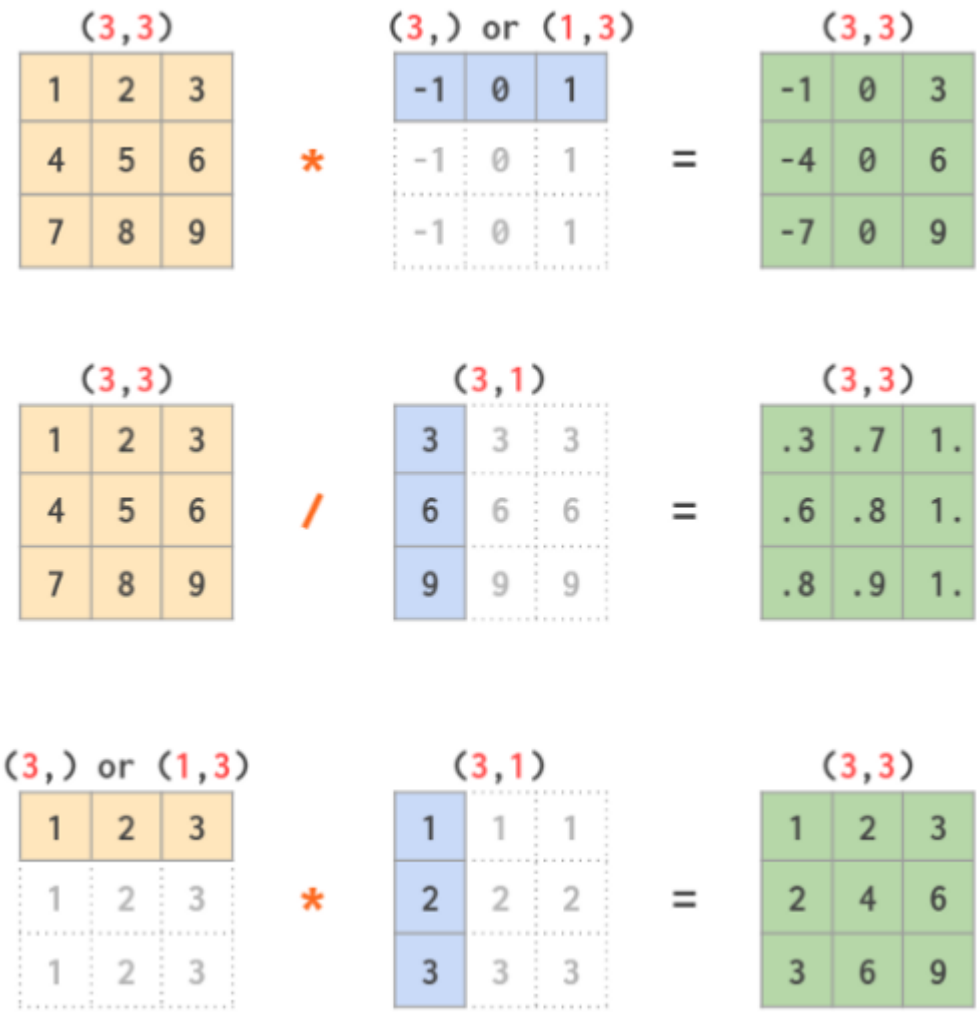
```
>>> x = np.zeros((300, 20))
>>> x = np.transpose(x)
>>> print(x.shape)
(20, 300)
```

TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Broadcasting

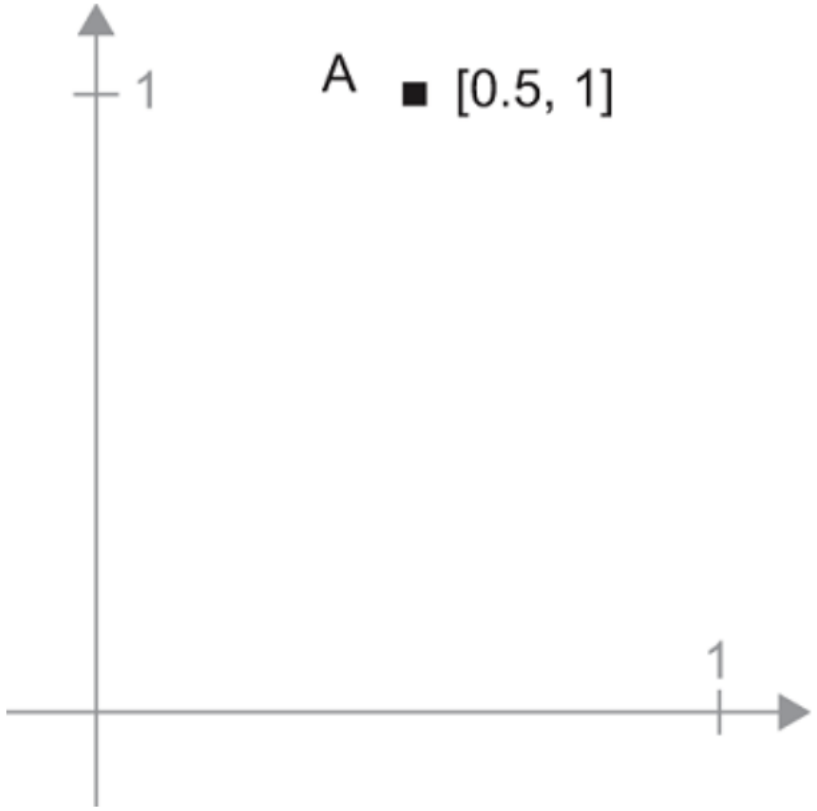


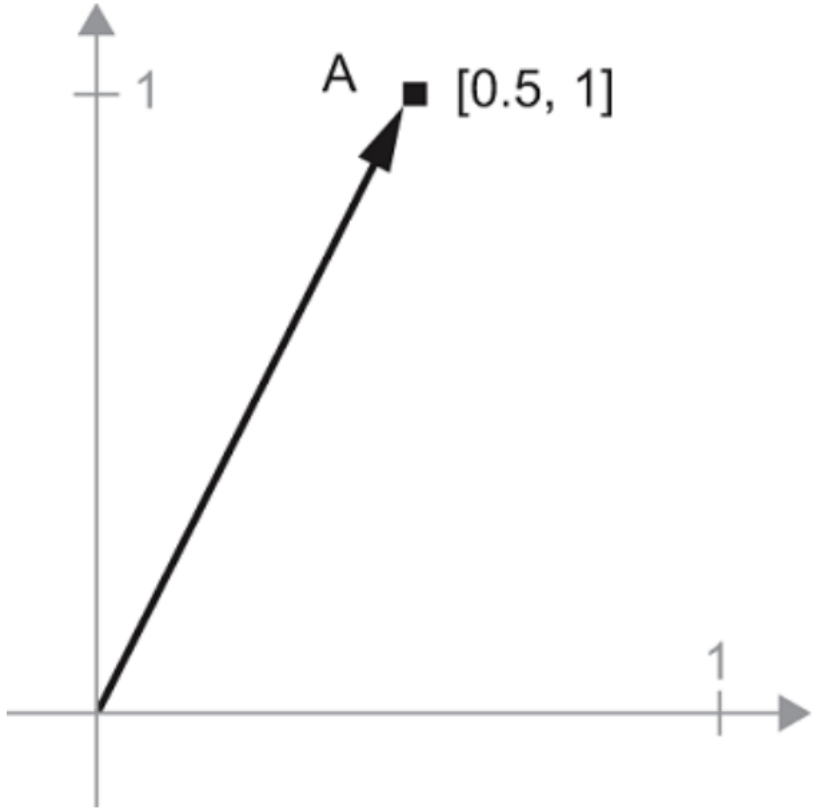


TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

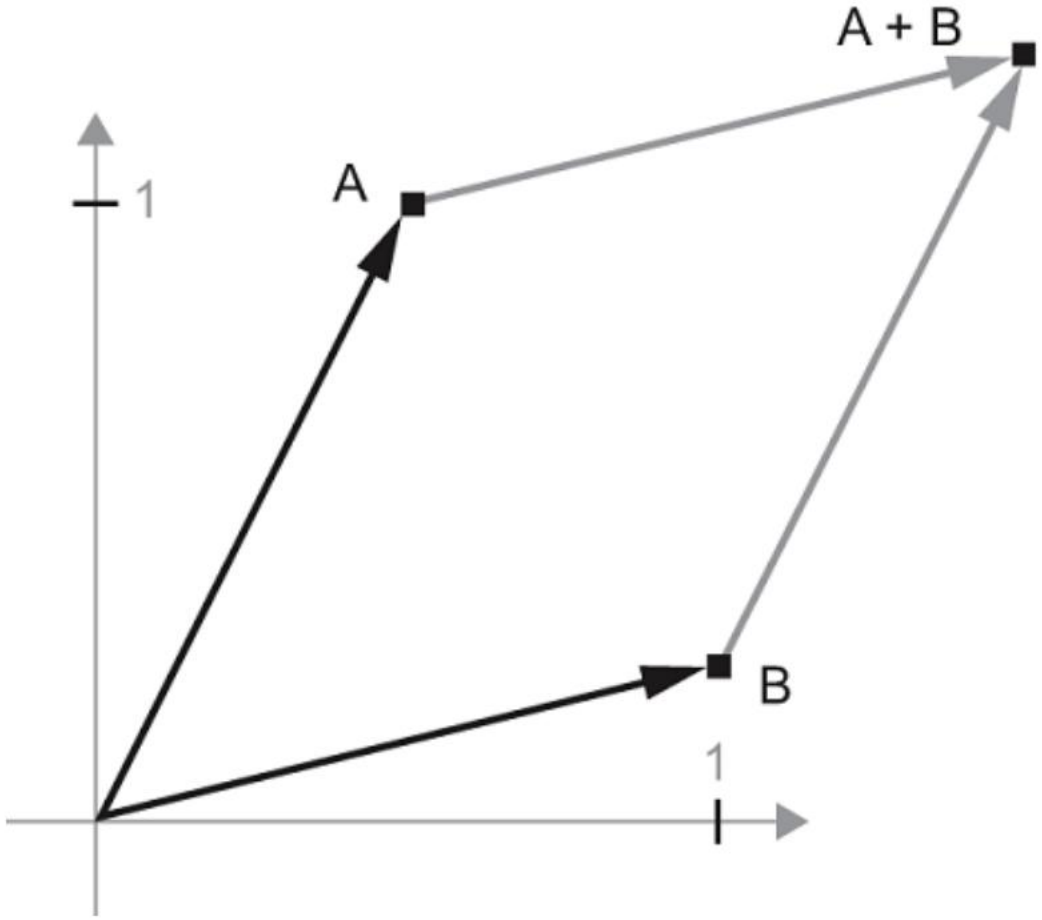
Tensor operations: geometric interpretation

```
A = [0.5, 1]
```





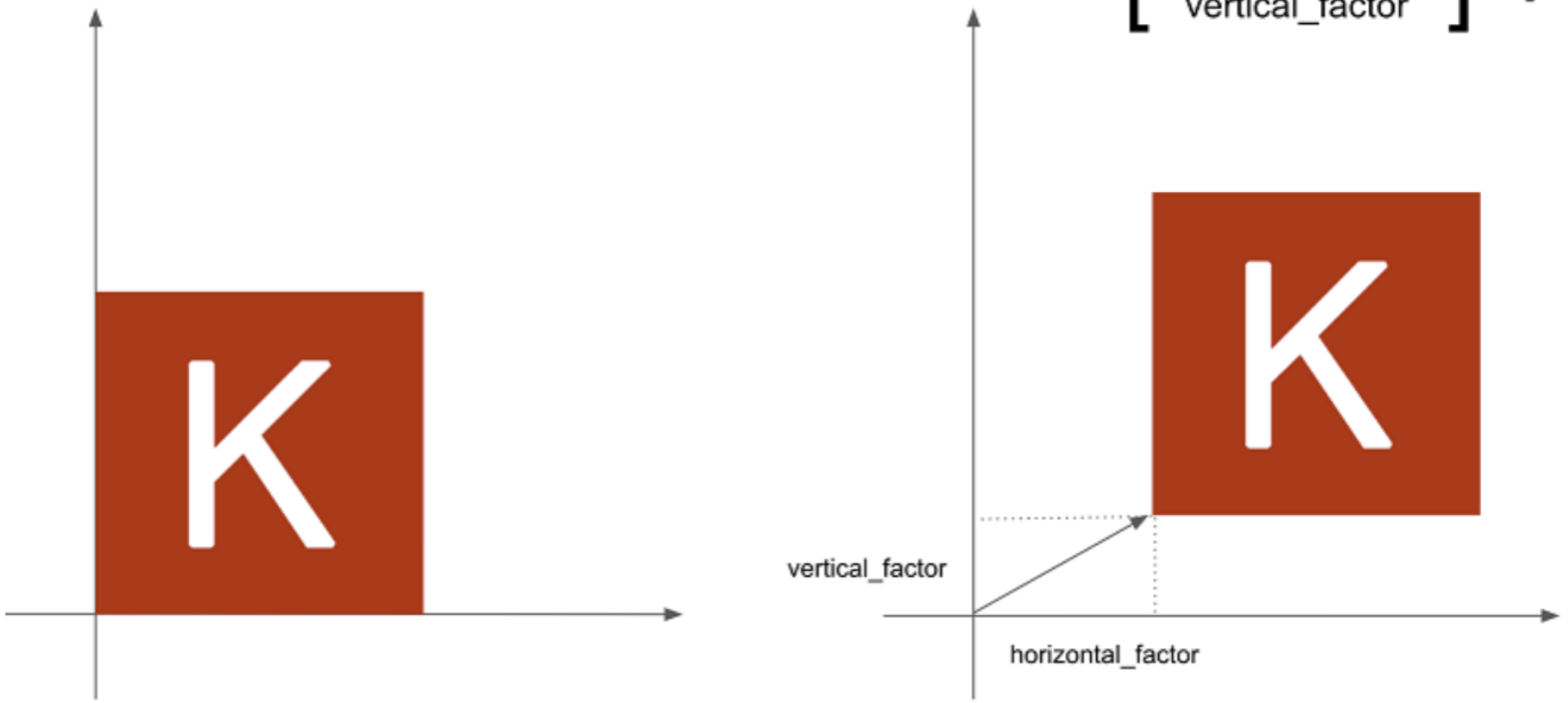
Tensor operations: geometric interpretation



Tensor operations: geometric interpretation

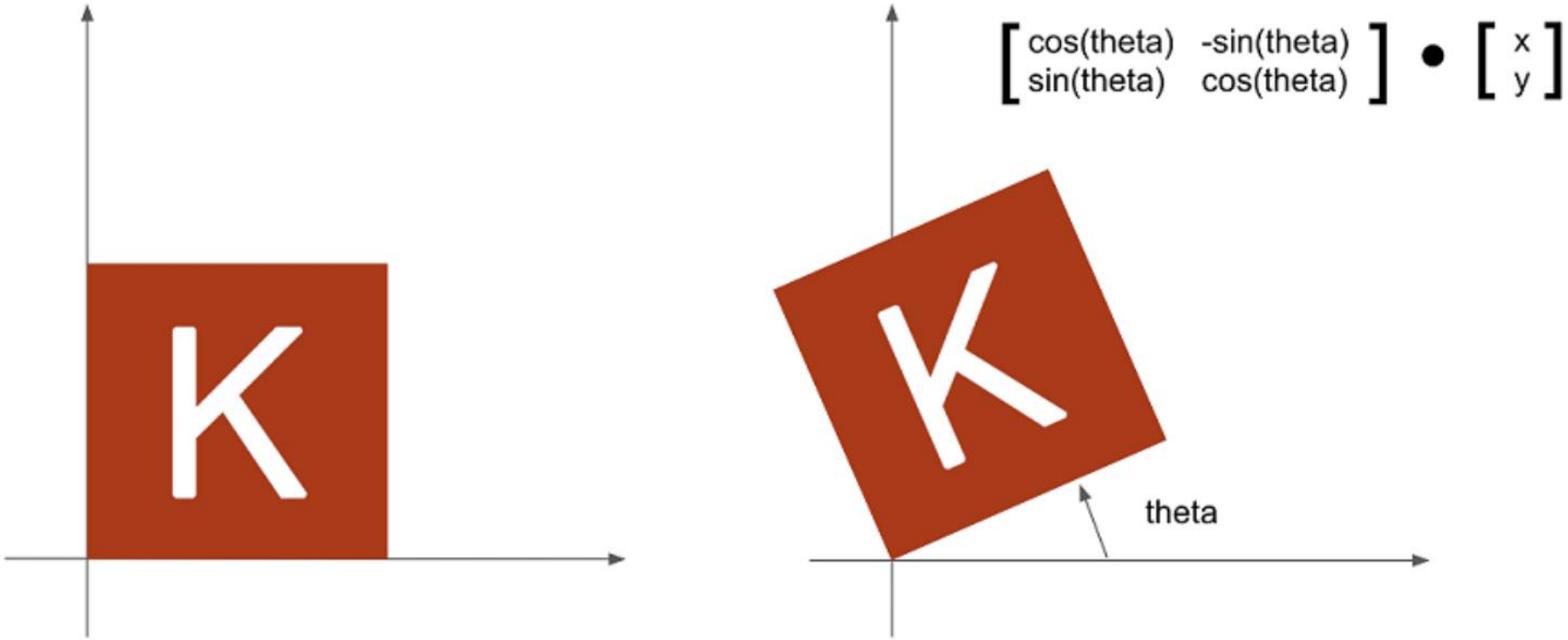
Translation: As you just saw, adding a vector to a point will move this points by a fixed amount in a fixed direction. Applied to a set of points (such as a 2D object), this is called a "translation" (see figure TODO).

$$\begin{bmatrix} \text{horizontal\_factor} \\ \text{vertical\_factor} \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$



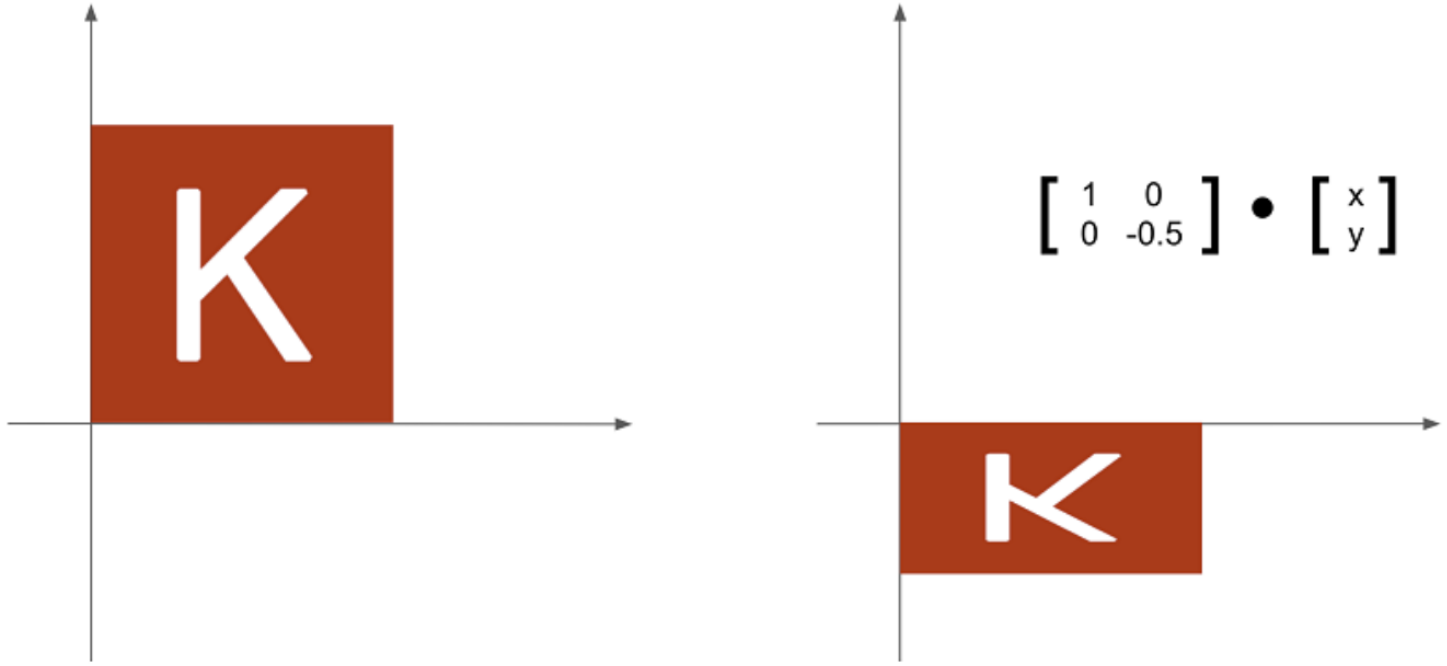
Tensor operations: geometric interpretation

Rotation: A rotation of a 2D vector by an angle theta (see figure TODO) can be achieved via a dot product with a 2 × 2 matrix `R = [[cos(theta), sin(theta)], [-sin(theta), cos(theta)]]`.



Tensor operations: geometric interpretation

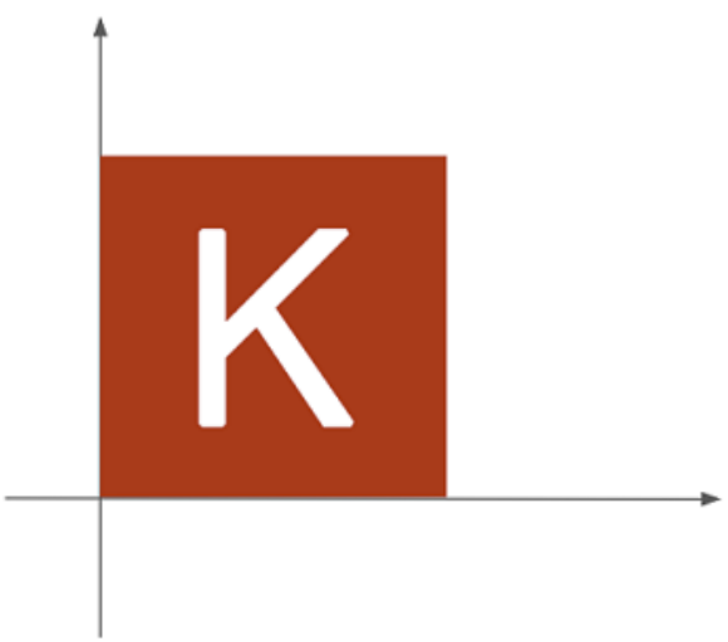
Scaling: A vertical and horizontal scaling of the image (see figure TODO) can be achieved via a dot product with a 2 x 2 matrix s = (note that such a matrix is called a "diagonal matrix", because it only has non-zero coefficients in its "diagonal", going from the top left to the bottom right).

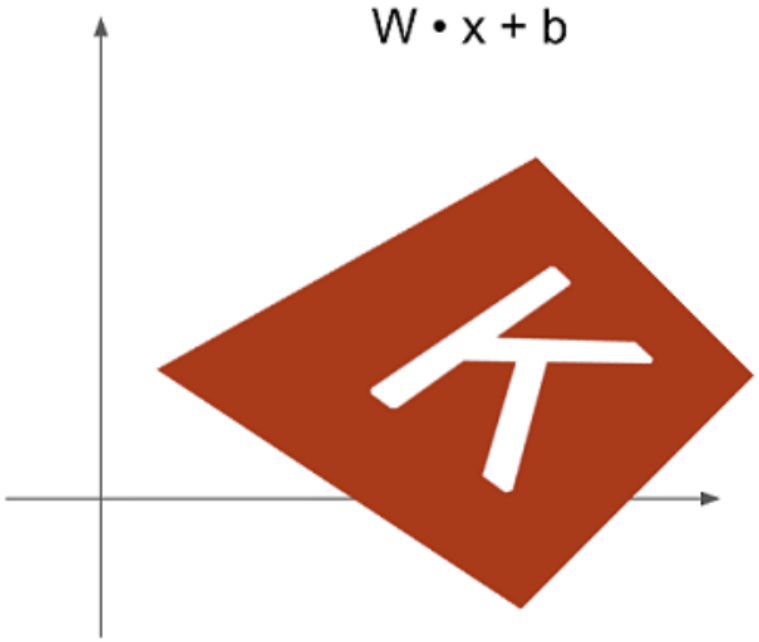


Tensor operations: geometric interpretation

Linear transform: A dot product with an arbitrary matrix implements a linear transform. Note that *scaling* and *rotation*, seen above, are by definition linear transforms.

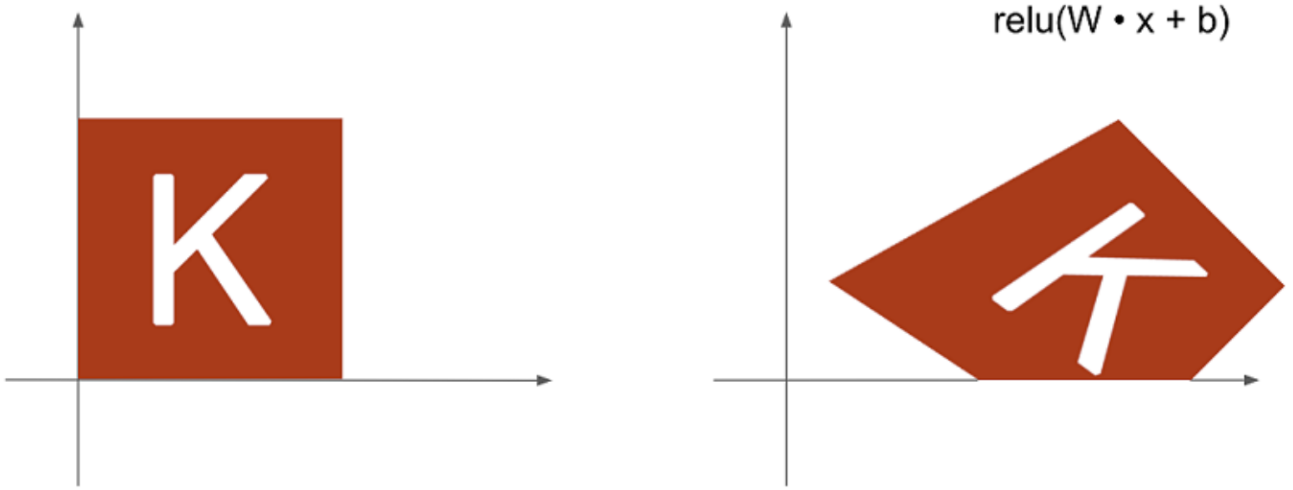
Affine transform: An affine transform (see figure TODO) is the combination of a linear transform (achieved via a dot product some matrix) and a translation (achieved via a vector addition). As you have probably recognized, that's exactly the `y = W • x + b` computation implemented by the `Dense` layer! A `Dense` layer without an activation function is an affine layer.





Tensor operations: geometric interpretation

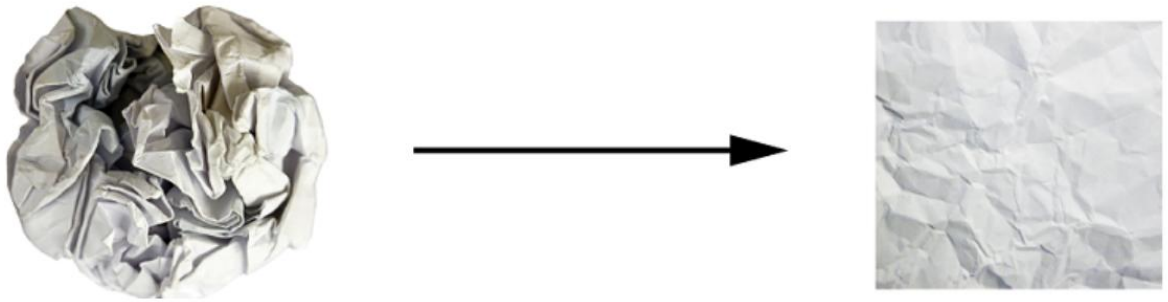
`Dense` layer with `relu` activation: An important observation about affine transforms is that if you apply many of them repeatedly, you still end up with an affine transform (so you could just have applied that one affine transform in the first place). Let's try it with two: `affine2(affine1(x)) = W2 • (W1 • x + b1) + b2 = (W2 • Wa) • x + (W2 • b1 + b2)`. That's an affine transform where the linear part is the matrix `W2 • W1` and the translation part is the vector `W2 • b1 + b2`. As a consequence, a multi-layer neural network made entirely of `Dense` layers without activations would be equivalent to a single `Dense` layer. This "deep" neural network would just be a linear model in disguise! This is why we need activation functions, like `relu` (seen in action in figure TODO). Thanks to activation functions, a chain of `Dense` layer can be made to implement very complex, non-linear geometric transformation, resulting in very rich hypothesis spaces for your deep neural networks. We cover this idea in more detail in the next chapter.



Deep Learning: geometric interpretation

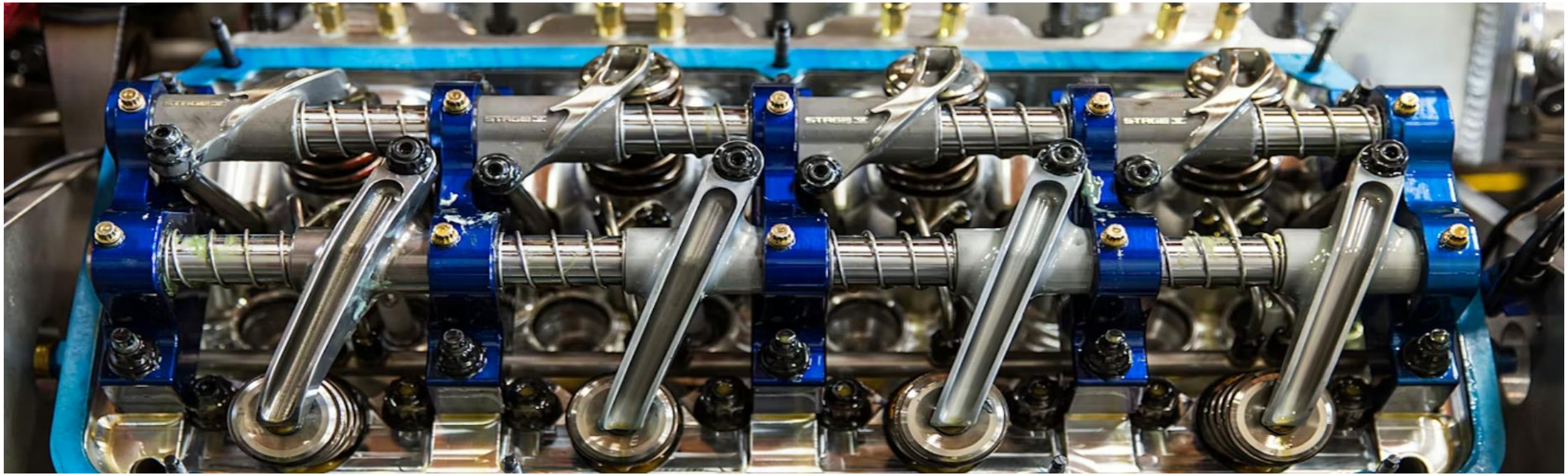
- **Each layer** in the neural network performs **tensor operations** on the data
- These tensor operations **transform the data** into meaningful representations
- Tensor operations can be **interpreted geometrically**…
- … and so can neural networks:

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

GRADIENT-BASED OPTIMIZATION

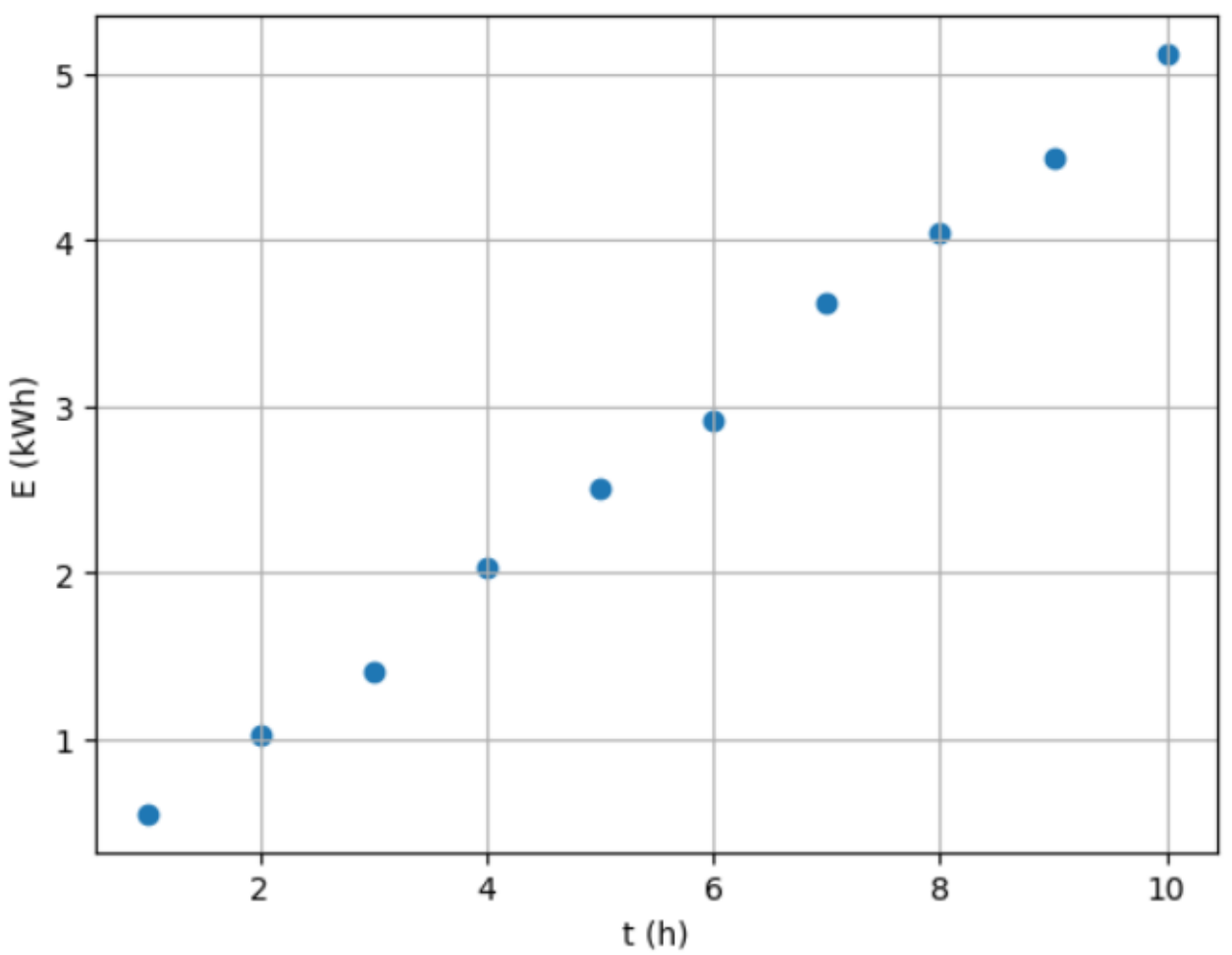
TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

vives University of Applied Sciences

Example: Energy usage

Time (hours)	Energy Usage (kWh)
1	0.55
2	1.03
3	1.41
4	2.03
5	2.51
6	2.92
7	3.62
8	4.05
9	4.49
10	5.12

```
t = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
E = np.array([0.55, 1.03, 1.41, 2.03, 2.51, 2.92, 3.62, 4.05, 4.49, 5.12])

plt.scatter(t, E);
plt.grid();
plt.xlabel('t (h)');
plt.ylabel('E (kWh)');
```



Example: Energy usage

```
P_guess = 0.5
E_guess = P_guess * t

plt.scatter(t, E, label='data');
plt.plot(t, E_guess, 'k:', label='initial guess');
plt.grid();
plt.xlabel('t (h)');
plt.ylabel('E (kWh)');
plt.legend();
```

- What is the energy usage after 11 hours?
- We construct a **model** to predict:

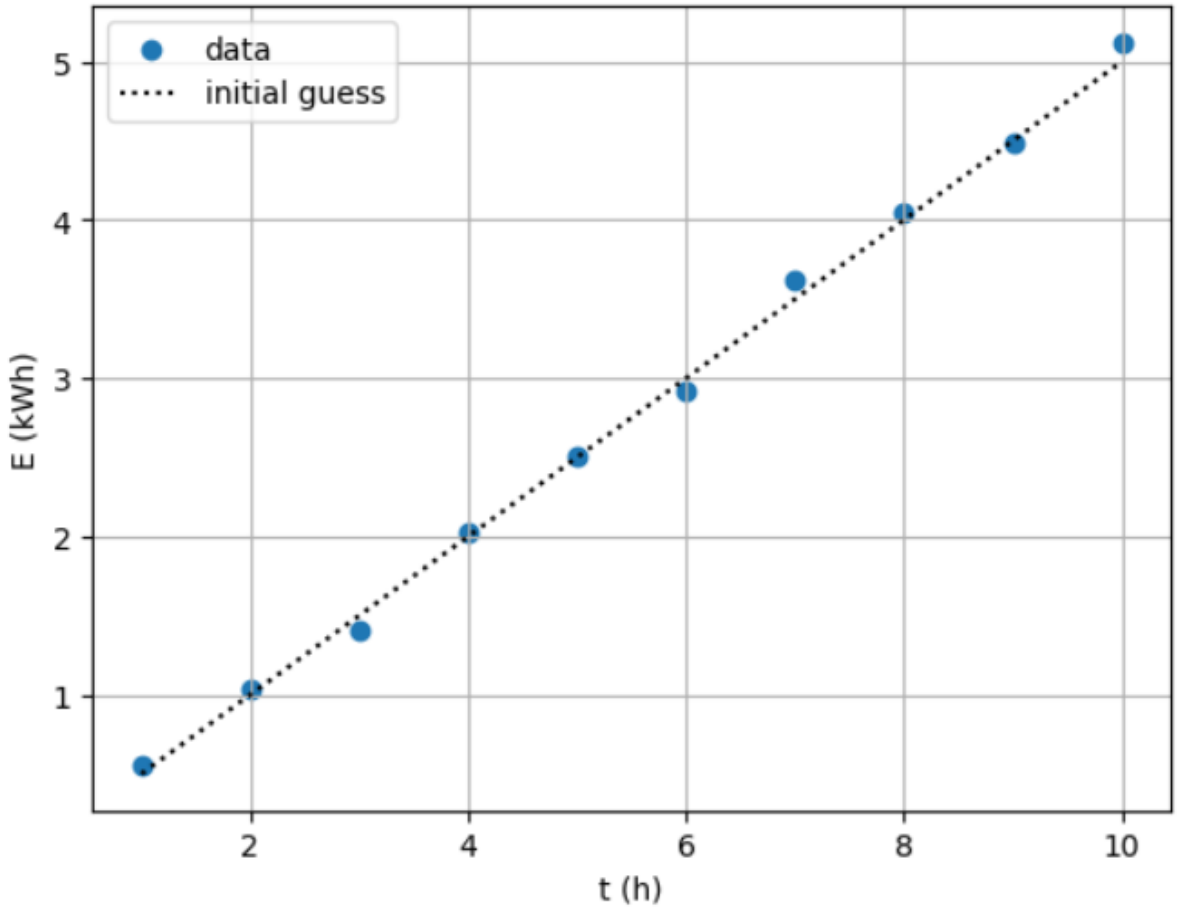
$$\widehat{E}(t) = Pt$$

- Based on visual inspection:

$$P = 0.5\ \mathbf{kW}$$

- Therefore:

$$\widehat{E}(11) = 5.5\ \mathbf{kWh}$$



Example: Energy usage

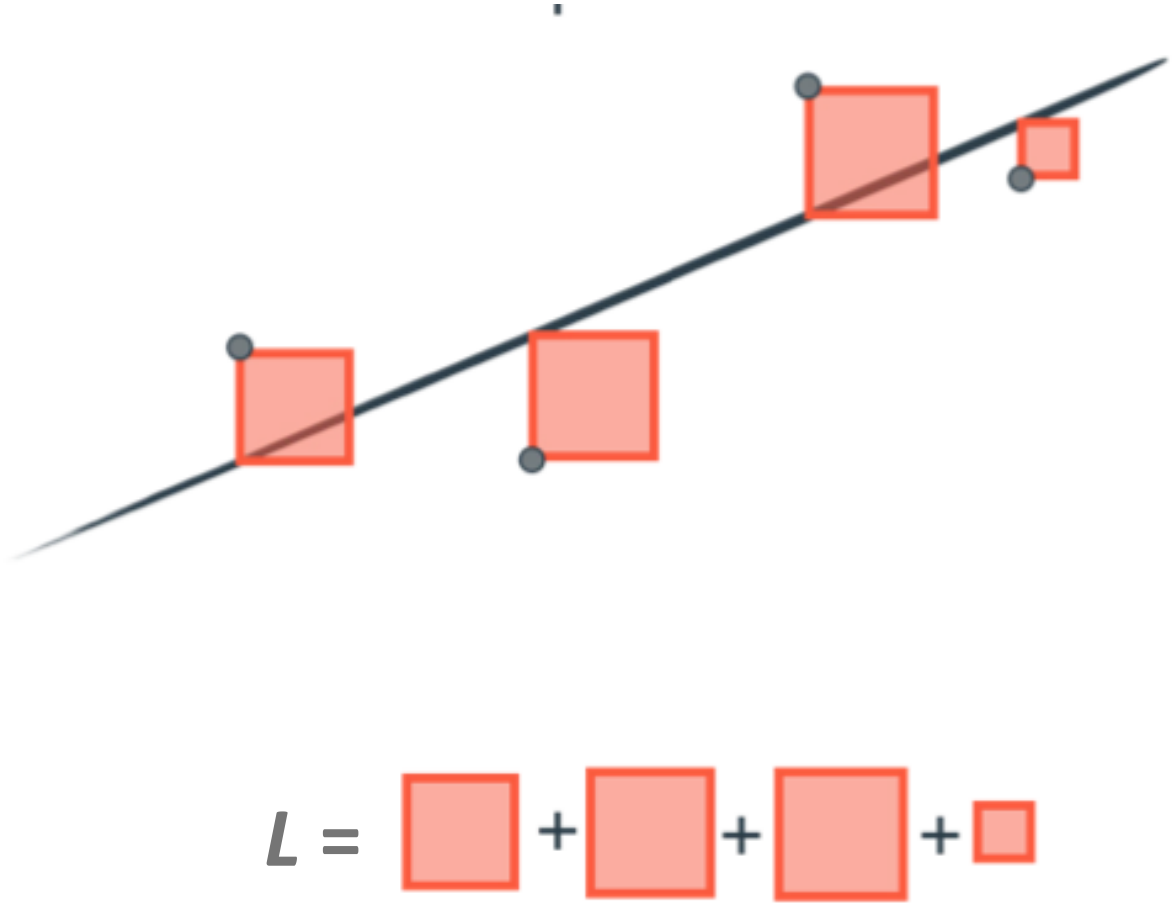
- Can we have a better model?
- We define the error or **residual**:

$$\epsilon_i = E_i - \widehat{E}_i = E_i - Pt_i$$

- We calculate the **sum or squared errors**:

$$L = \sum_i \epsilon_i^2$$

- L is called the **loss function**



Example: Energy usage

- We **minimize** the **loss function *L***
- In statistics: we use the **least squares method**
- We set the derivative of ***L*** wrt ***P*** equal to zero:

$$\frac{\partial L}{\partial P} = 0$$

- We solve the equation to find:

$$P_{\text{optimal}} = \frac{\sum_i E_i t_i}{\sum_i t_i^2} = 0.505\ \text{kW}$$

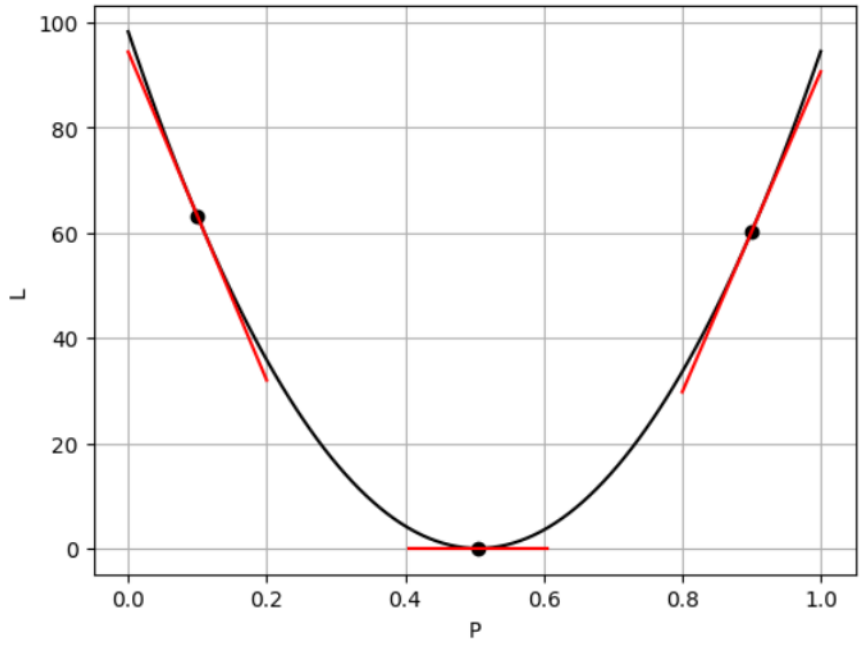
$$\begin{aligned}
&\frac{\delta L}{\delta P} \\
&= \sum_i 2\epsilon_i \cdot -t_i \\
&= 2\sum_i (E_i - Pt_i) \cdot -t_i \\
&= 2P\sum_i t_i^2 - 2\sum_i E_i t_i \\
&= 0
\end{aligned}$$

```
P_optimal = np.sum(E * t) / np.sum(t ** 2)
print('The optimal value for P is', P_optimal)
```

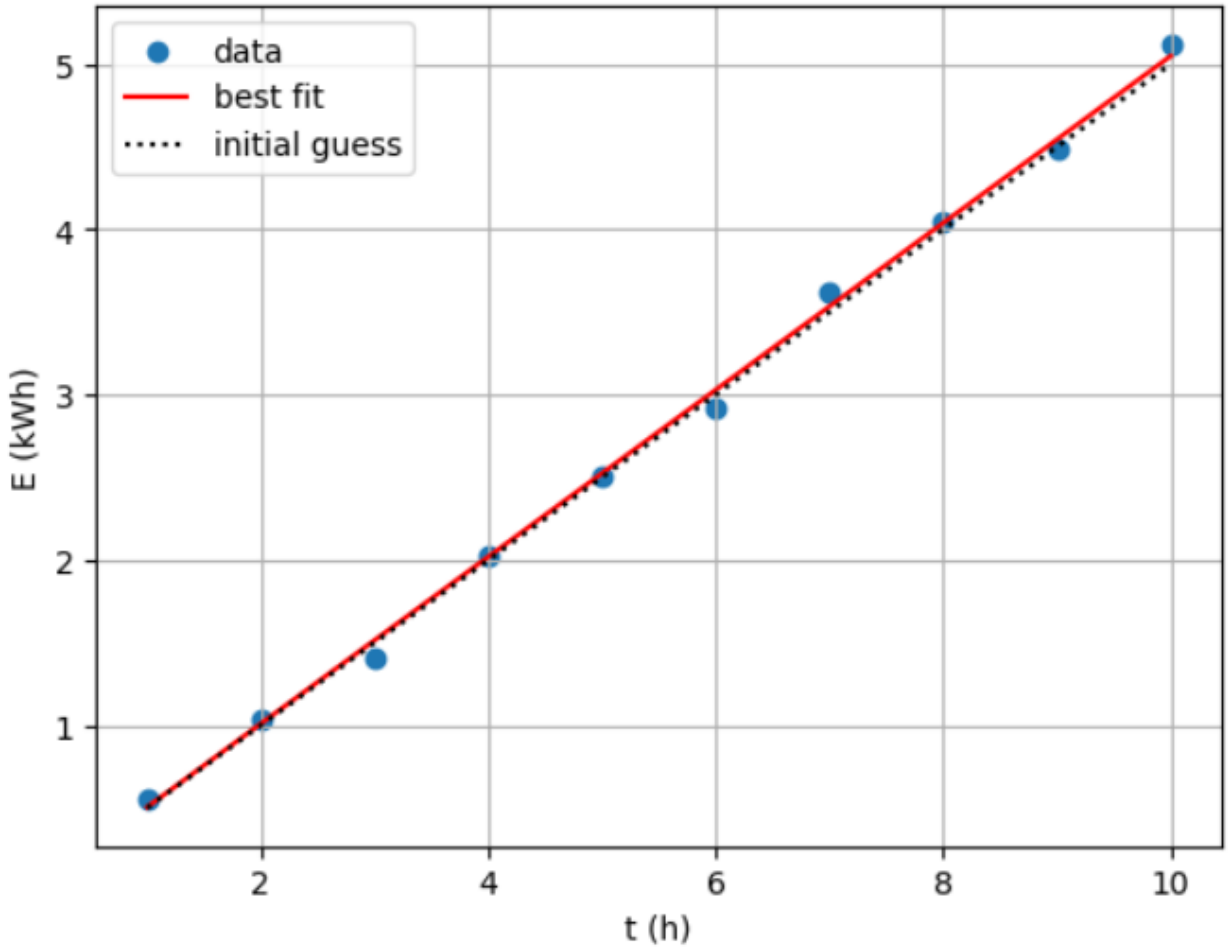
```
The optimal value for P is 0.5048831168831168
```

Example: Energy usage

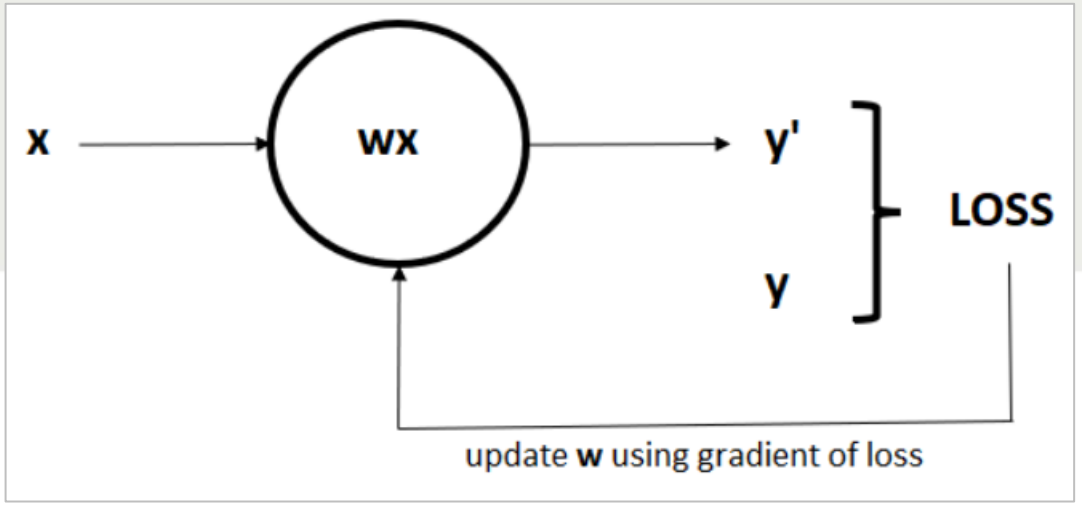
- Why set the derivative to zero?
- The derivative shows the slope of the curve
- **At the minimum, the slope is zero**



- The minimum loss gives **the best fitting line**!



Example: Energy usage



Deep learning:

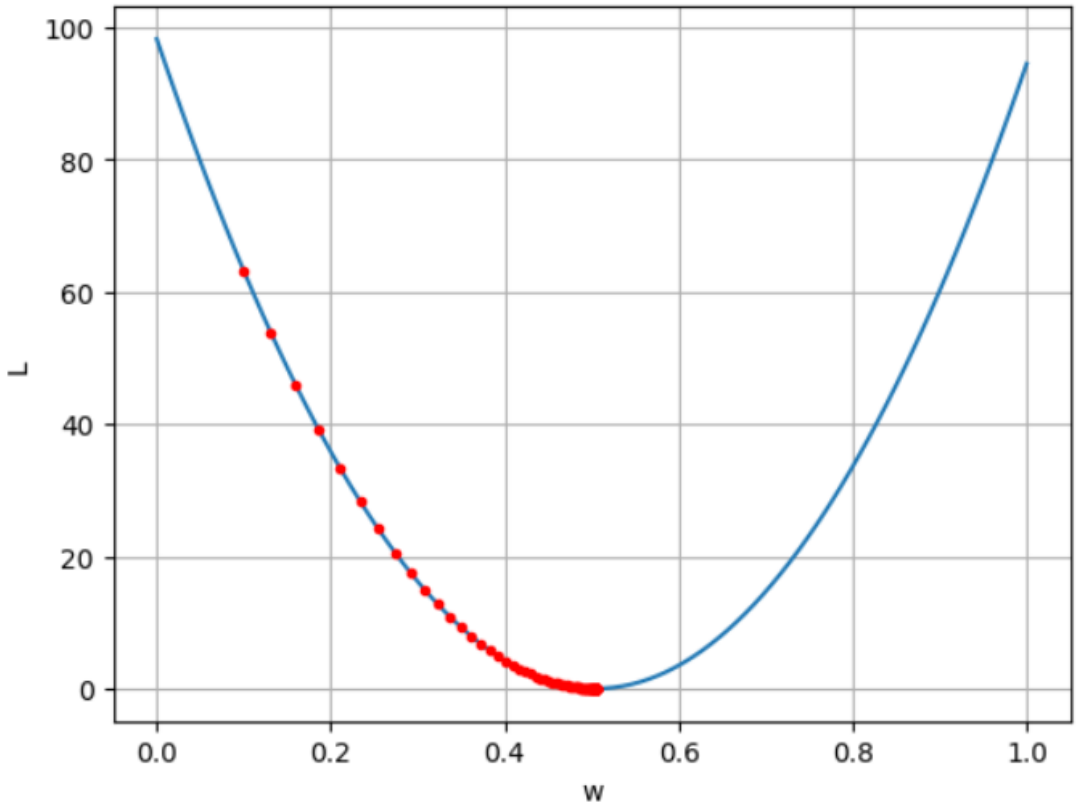
- Models are much more complex and nonlinear
- **Finding an exact solution is impossible**

Solution method:

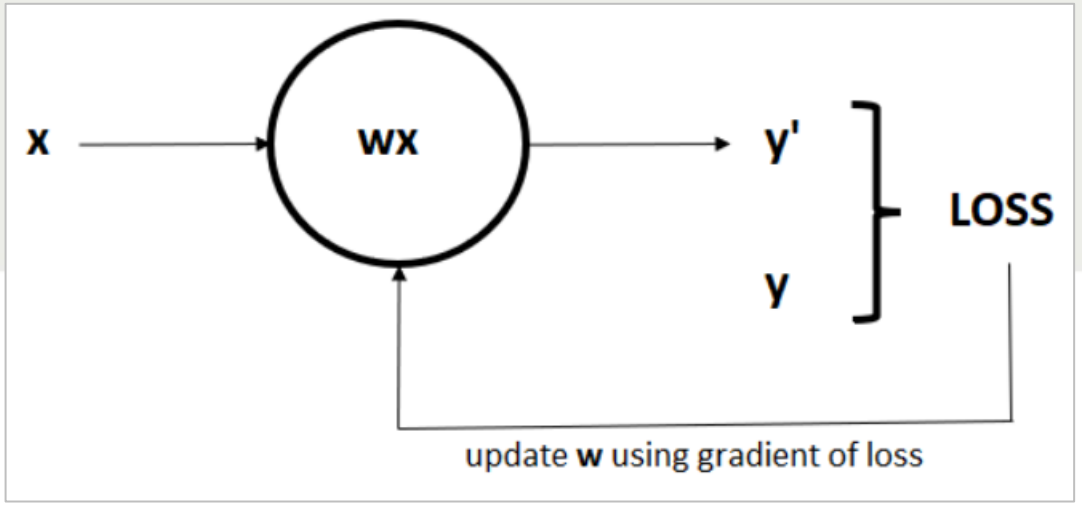
- Build a neural net with a **single linear node**
- **Gradient Descent** finds an approximate solution
  1. Start with a random value for ***w***
  2. Calculate the gradient (derivative) of ***L*** at ***w***
  3. Take a small step in the direction of the negative slope

$$w_{k+1} = w_k - \alpha \frac{\delta L}{\delta w}$$

  4. Repeat steps 2 and 3 until the minimum is reached closely enough



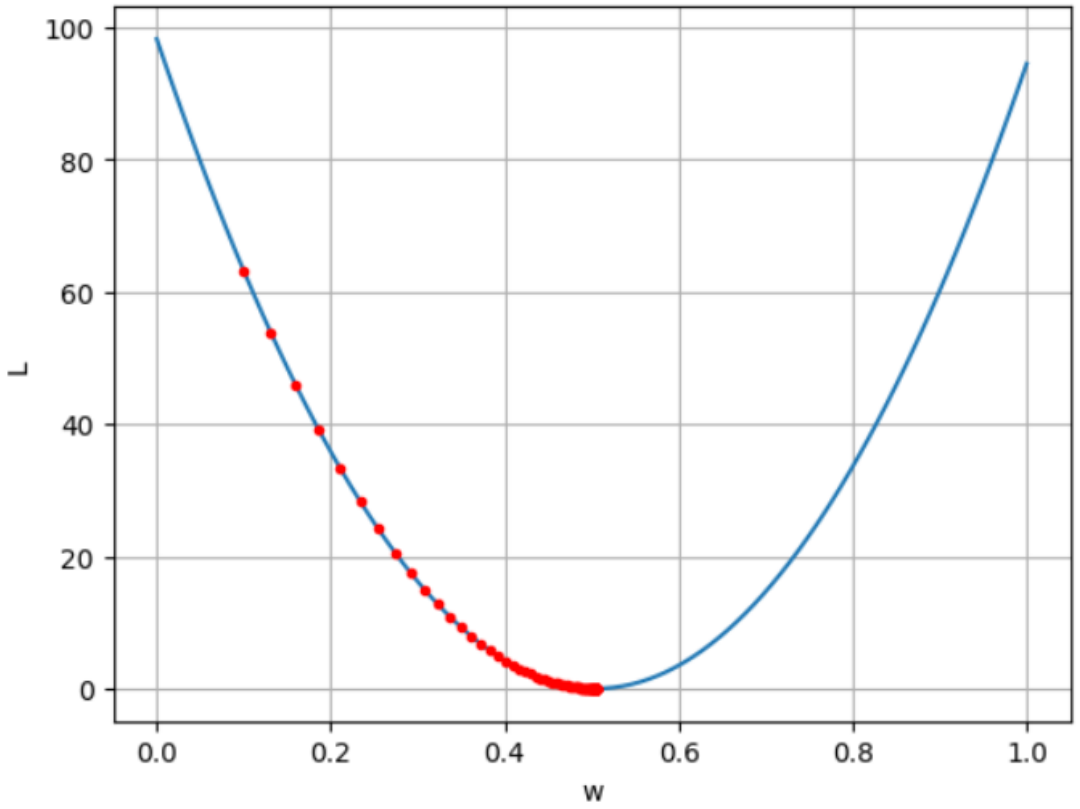
Example: Energy usage




```
alpha = 0.0001  # learning rate
niter = 100     # number of iterations

w_steps = np.zeros(niter) # array containing all intermediate values of w
w_steps[0] = 0.1          # initial value for w

# gradient descent algorithm
for k in range(niter-1):
    w_steps[k+1] = w_steps[k] - alpha * dLdw(w_steps[k])

print("the optimal value for w found by gradient descent is", w_steps[-1])
```

$$w_{k+1} = w_k - \alpha \frac{\delta L}{\delta w}$$

```
the optimal value for w found by gradient descent is 0.5047378060244245
```



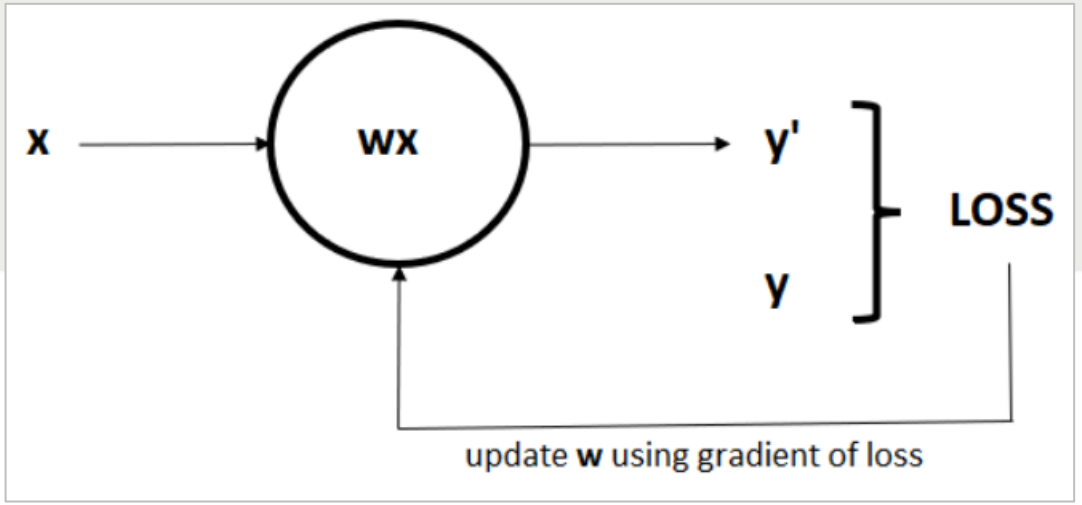
Note: The **learning rate** α determines the size of each step taken during optimization, balancing speed and accuracy.

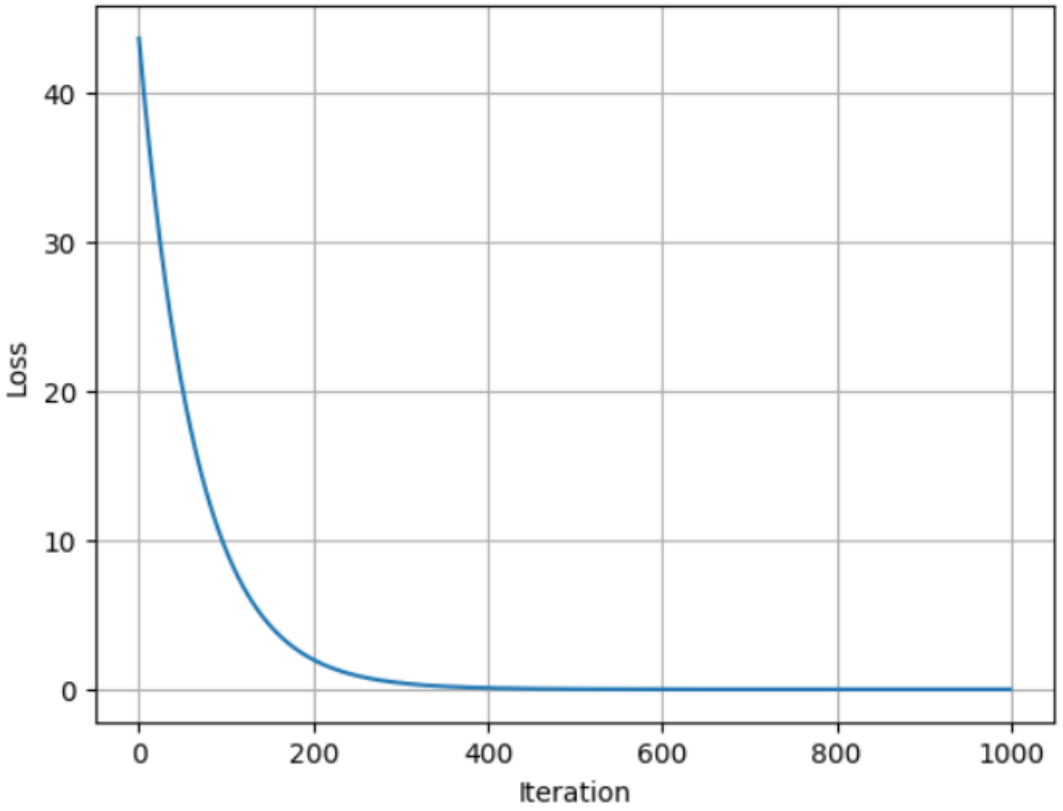
Example: Energy usage

With **Keras** we define a neural net with one layer containing a **single linear unit** without bias (= intercept):

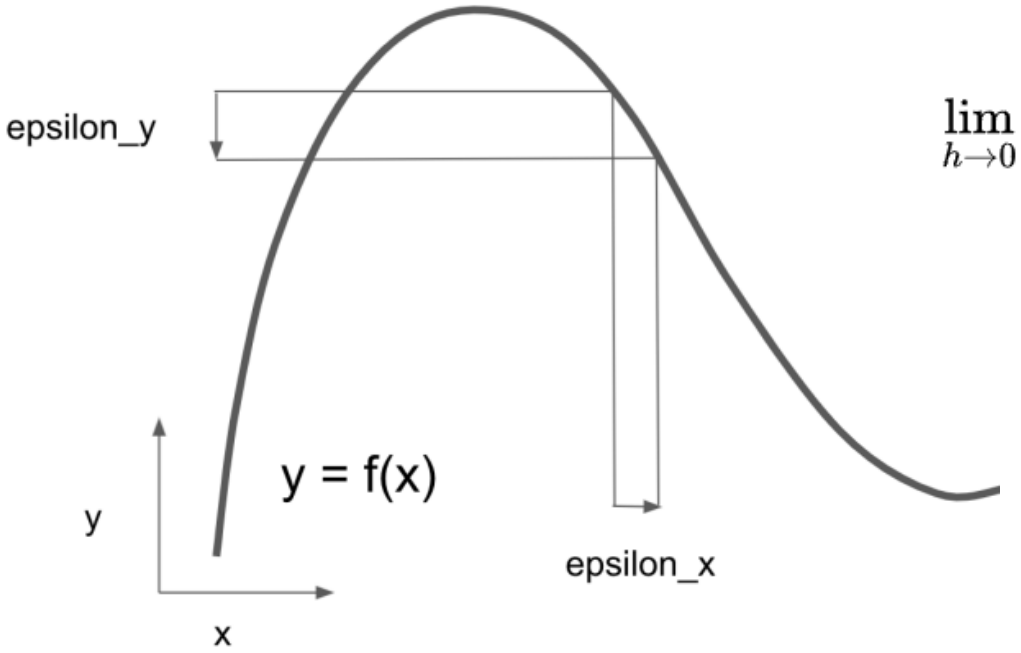
```
# building the model
single_node = keras.Sequential()
single_node.add(keras.layers.Dense(1, use_bias=0))
# compiling the model
single_node.compile(
    optimizer=keras.optimizers.SGD(learning_rate=alpha),
    loss='mse'
)
# training the model
result = single_node.fit(
    x[:, np.newaxis], y[:, np.newaxis],
    epochs=1000, batch_size=len(x)
);

w_opt = single_node.get_weights()
print("the optimal value for w found by Keras is", w_opt[0][0].item())
```

```
the optimal value for w found by Keras is 0.5053511261940002
```

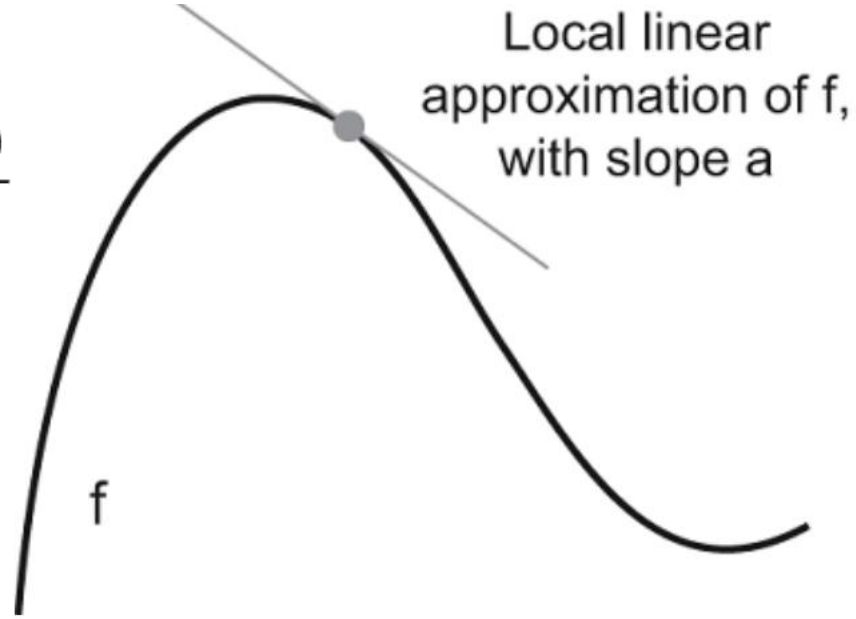




Derivative and gradient



$$\lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$



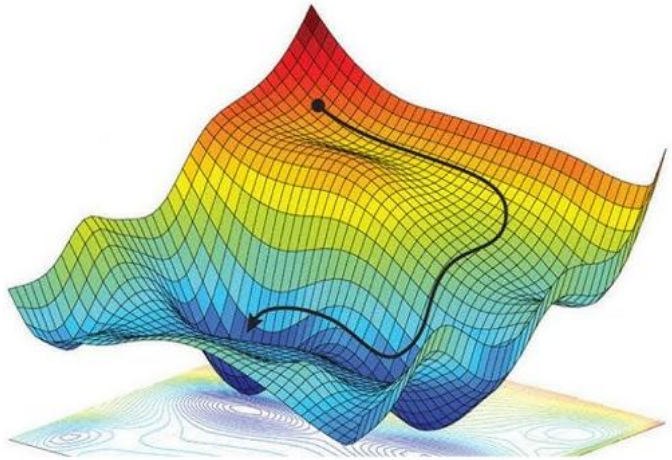
- The **derivative** is the slope of the tangent line at a point
- The **gradient** can be seen as the derivative in multiple dimensions

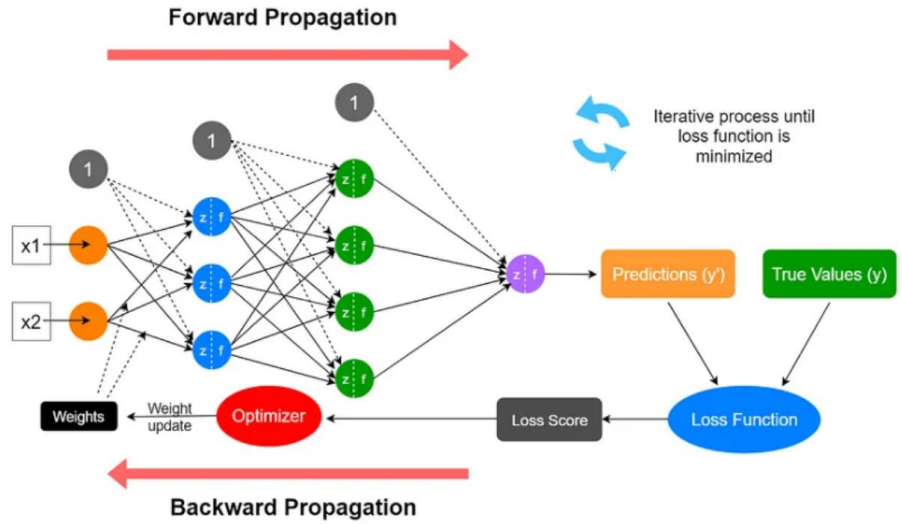
TALLINNA TEHNIKAKÕRGKOOL
TTK UNIVERSITY OF APPLIED SCIENCES

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.



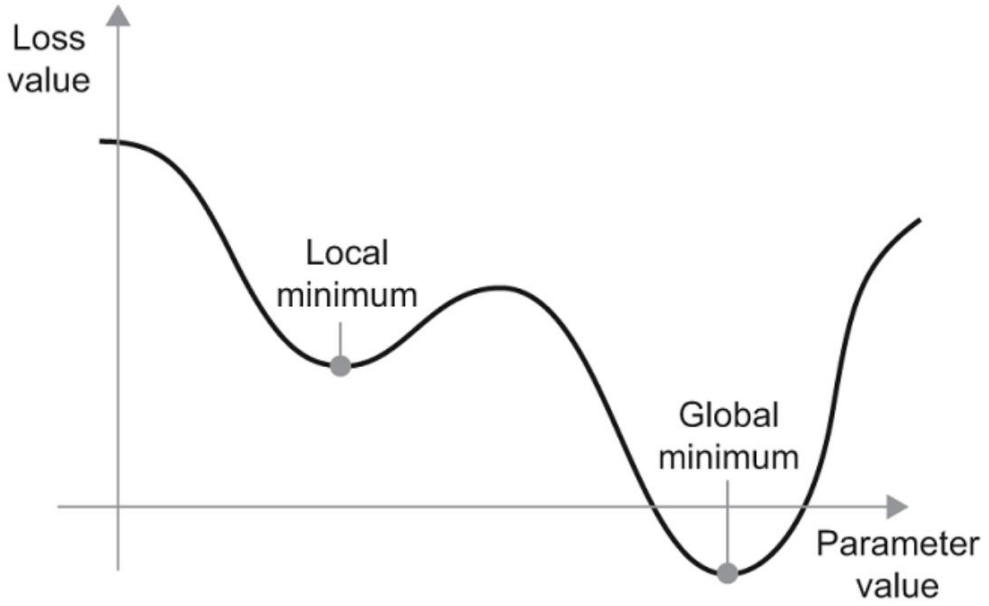


- **True SGD**: batchsize == 1
- **Mini-batch SGD**: 1 < batchsize < nsamples
- **(Full) Batch SGD**: batchsize == nsamples

Stochastic Gradient Descent: variants

- Problem: local minima!
- Solution: **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)
```



- E.g., RMSProp, AdaGrad, …

Backpropagation and the chain rule

- **Loss function** of neural network is a composite function

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

- Gradient of loss function is needed to **update parameters**

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

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

- **Chain rule**: gradient of composite function = product of gradients of functions

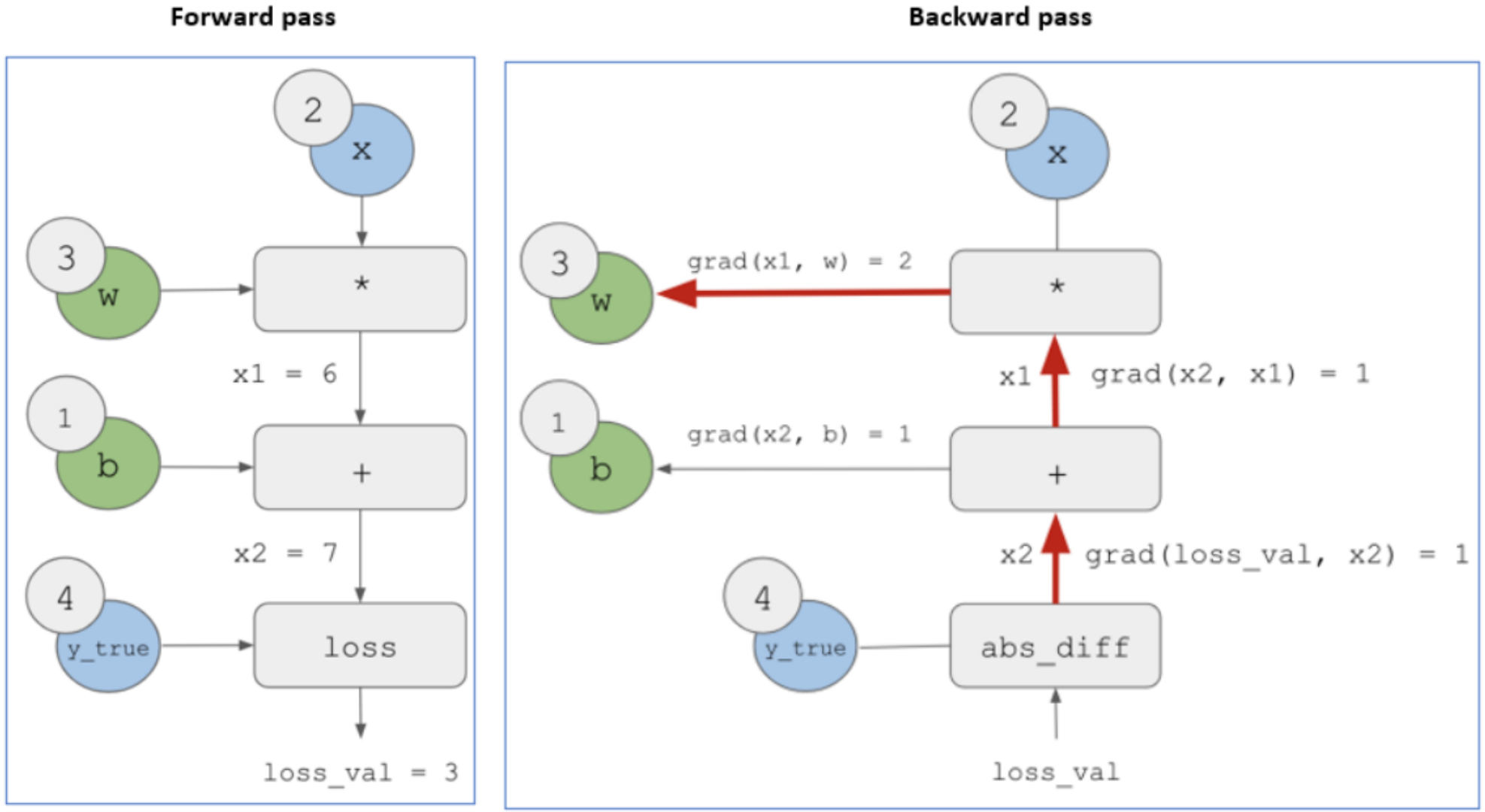
```
def fghj(x):
    x1 = j(x)
    x2 = h(x1)
    x3 = g(x2)
    y = f(x3)
    return y

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

- **Backpropagation** algorithm applies the chain rule to calculate the gradient of the loss function with respect to the parameters of the neural network

Autodiff

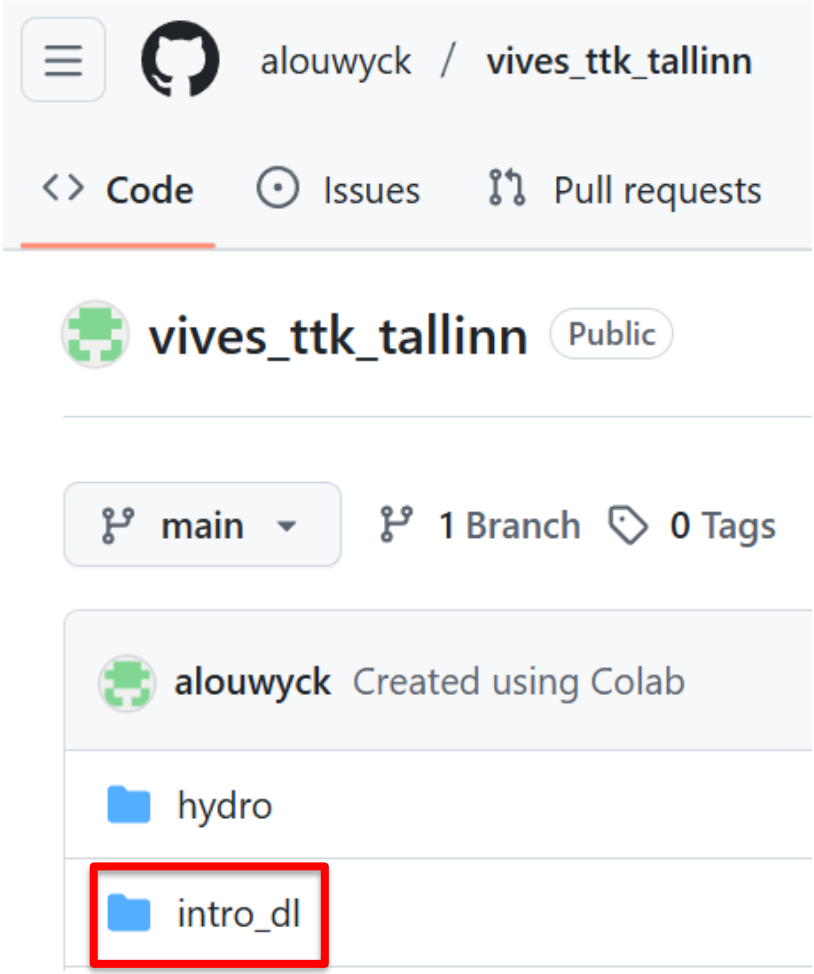
AUTOMATIC DIFFERENTIATION WITH COMPUTATION GRAPHS

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.



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.
- Copilot and perplexity.ai were used to generate some of the content