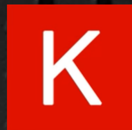
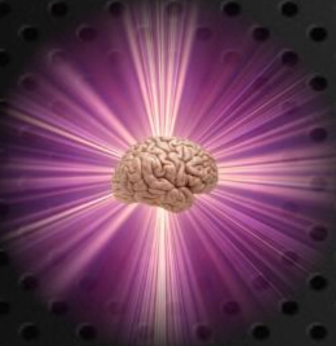
AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering
Department of Biocybernetics and Biomedical Engineering



Knowledge-based Computational Intelligence and Data Mining in Biomedicine



Introduction to Computational Intelligence and Neural Networks

Adrian Horzyk
horzyk@agh.edu.pl

Google: Adrian Horzyk

How will we work?

- We will start with a very short introduction.
- The theory will be presented together with practical examples.
- During the lectures, I will introduce new topics that will be next experienced by you in the subsequent laboratory classes.
- We will focus on the most important issues of machine learning and computational intelligence, starting from scratch.
- The theory necessary to accomplish assignments during lab classes will be presented during the lectures – don't miss them!
- You need to install Anaconda 3, Keras, and a few other libraries.
- We will implement everything in Python, so be ready for it!
- For labs, you get the link to download Jupyter notebooks with the presentations of examples of models, explanations of new implemented topics, and assignments for you to accomplish.

How will you be evaluated?

- During the laboratory classes, you always start with filling the active Quiz in MS Teams, where you sign up for the classes and answer a few questions that will check your understanding of the topics presented in the preceding lecture. Each Assignment you should finish and upload to MS Teams by Monday of the week following classes you got it! You will collect some points for the correct answers and for signing up for the laboratory classes!
- The points collected from all quizzes and assignments will be summed up to your final grade of the lab classes.
- Your work in the project classes will be evaluated separately. You will still need to accomplish active Quizzes in MS Teams, where you sign up for the classes and answer a few questions. Moreover, you will have to choose a topic for your final project and accomplish it by the end of the semester.
- At the last project classes, you will be asked to provide a 5 min presentation of your model, experiments, and achieved results to finish project classes.
- A final exam checking your knowledge and understanding will also take place.
- Your final grade of the course will be calculated as the average of the grades you got from the laboratory and project classes and the final exam.

The Course Schedule: Lab Classes

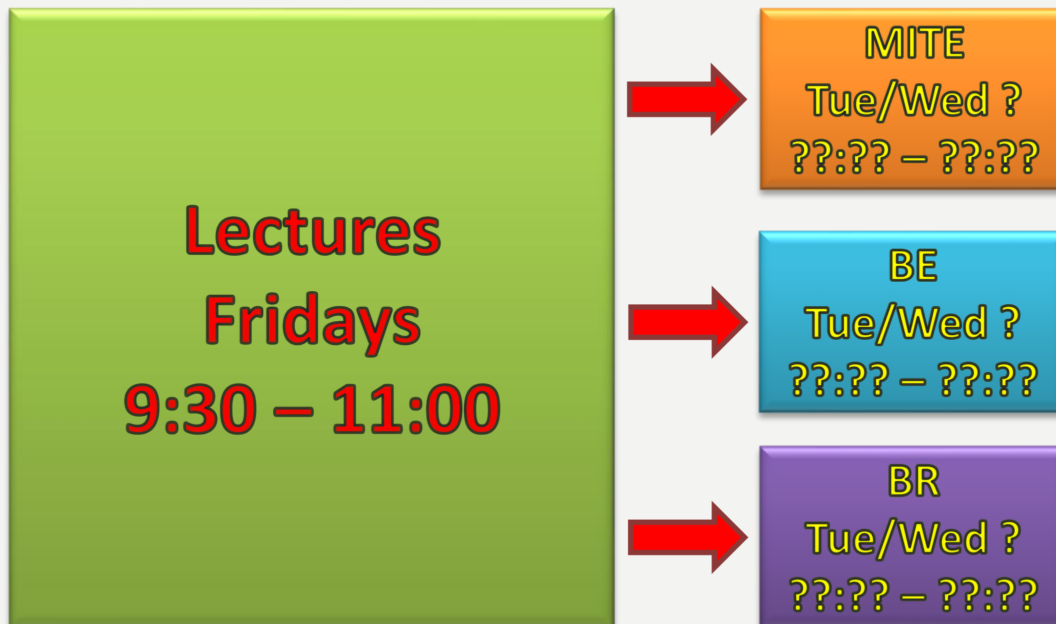
- In the first seven weeks of the semester, we will meet at the laboratory classes, to master the basic knowledge and abilities.
- The university system does not work correctly, so the right schedule is presented only in the first week of the semester and looks as follows:

- MITE - Medical Information Technology and Electronics
- BE - Biomaterials Engineering
- BR - Biomechanics and Robotics

✓ If you could not take part in your lab or project classes, choose a different group in the same week!

The Course Schedule: Project Classes

- In the second seven weeks of the semester, we will meet at the project classes, to accomplish a chosen project and present it at the last classes in MS Teams to all participants.
- When would you like to meet – I wait for your proposals:



- **MITE - Medical Information Technology and Electronics**
- **BE - Biomaterials Engineering**
- **BR - Biomechanics and Robotics**

✓ **If you could not take part in your lab or project classes, choose a different group in the same week!**

Final Projects in Project Classes

- Look at Kaggle.com and choose an interesting task to solve. You can choose any current or past competition task:
  - If you choose the current task, you can try to take part in the competitions and win quite a lot of money. You may not be able to compare your results to the other competitors (when the competitions do not finish during this semester), so then try to justify the quality of your results in any other way.
  - If you choose any past competition task (but not too old, please), you can be able to compare your results to the results achieved by the other competitors (recommended). Compare your results with those published in Kaggle in your final presentation.
  - You can also choose any other non-trivial and essential data and computational intelligence problem for you and describe this problem to me to agree on it.
- Before the end of the laboratory classes, I will announce the assignment to propose to me your chosen problem.
- You should finally choose your final project topic, having enough data to train the model that you will develop BEFORE the end of the laboratory classes!
- Interesting problems and solutions can be further developed to scientific papers or doctoral theses.

Machine Learning (ML)

What is the fundamental difference between classical programming and machine learning?

The machine learning system are trained (learns), not programmed!

ML is presented with a lot of data and responses to establish rules that allow the selected process to be automated without the need for *a priori* programming by the developer.

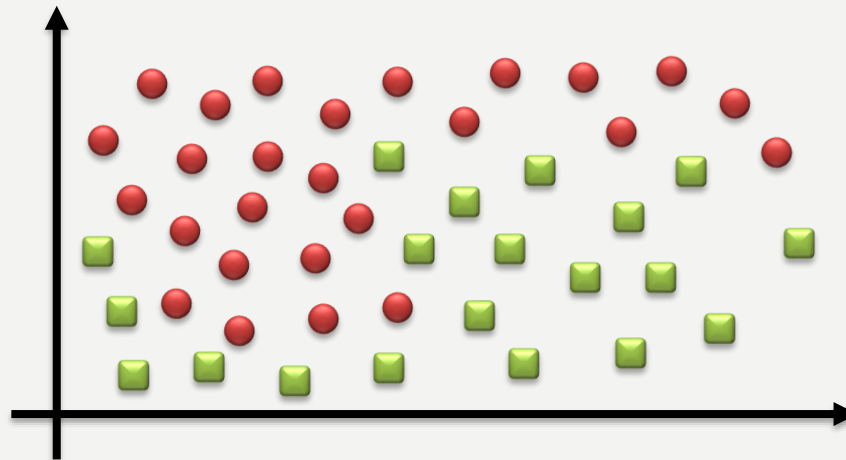
Machine Learning Tasks

In machine learning, we can distinguish a few groups of tasks:

- **Classification** assigns items into the defined target classes/categories.
- **Binary Classification** assigns items into two defined target classes/categories.
- **Multi-class Classification** assigns items exclusively into one of many defined target classes/categories.
- **Multi-label Classification** assigns items simultaneously to one or more defined target classes/categories.
- **Regression** predicts a range of numerical (continuous) values for a given dataset. For example, regression might be used to predict the cost, probability, size, growth, length, strength, height, width etc. of something of given variables/parameters.

Teaching / Training Problem

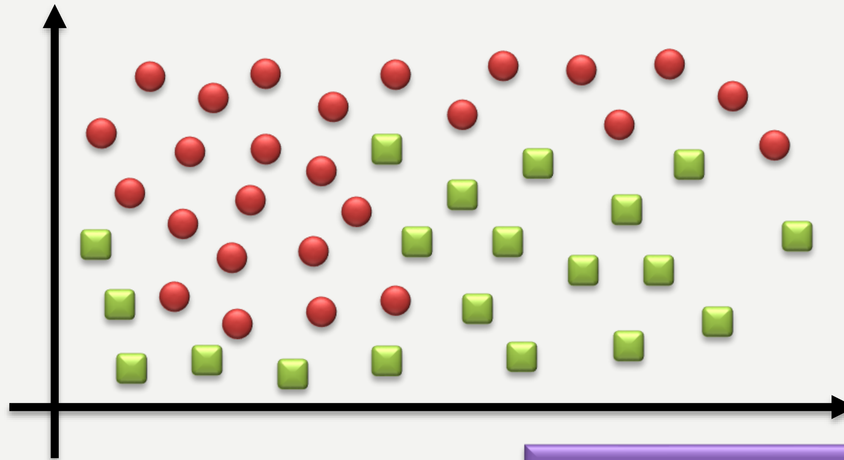
The problem of learning (training) consists in defining evaluation criteria to assess to what extent the model (training system, e.g., a neural network) fitted to the learning data and responses.



In the above figure, we have two types of objects (● and ■), which can be described by the coordinates (x, y).
Let's assume that we want to write an algorithm which, based on the coordinates of a point, will predict its color or shape, not only in the places where the above points occur, but also in other points in (data) space.

Measuring Correctness

For a given problem of classification of points on a plane:

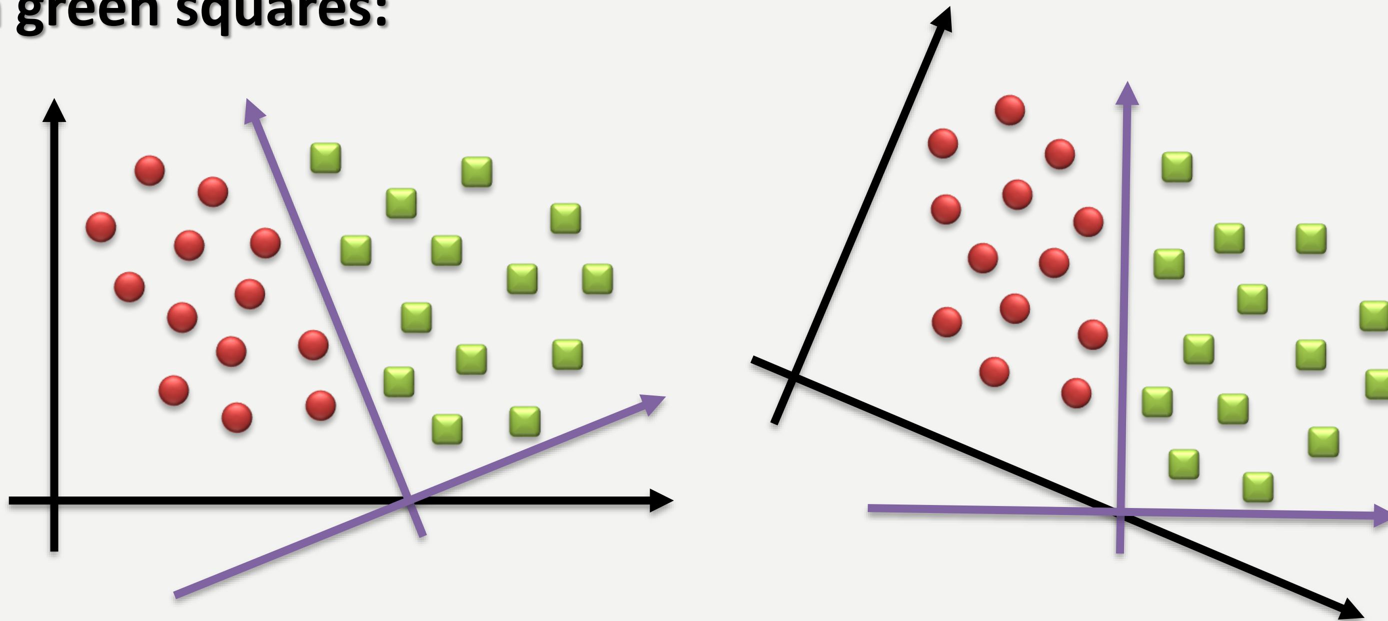


- the coordinates of the points are **Data (input data)**
- colors/shapes are expected **Answers (predictions/classes)**
- **Machine Learning** has to find the right **Rules/Models**

And we will evaluate these rules on the basis of the number of correctly classified points, i.e. consistent with the expected answers specified for the data.

Representation of Learning Data

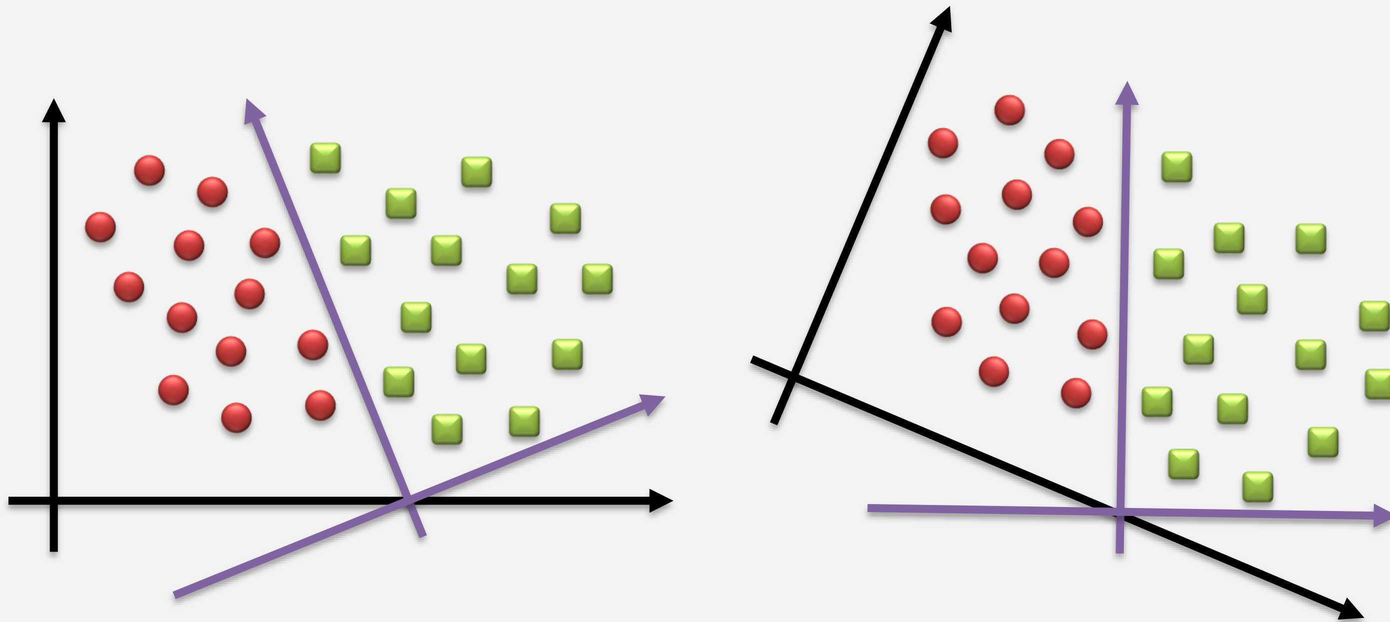
Data representation also plays an important role in machine learning, as we can illustrate in the example below. If we do a simple change – the transformation of the coordinate system, then it will be easier to find a rule that will distinguish (discriminate, separate) red circles from green squares:



IF x> 0 THEN it is a green square,

ELSE it is a red circle.

Machine Learning



In the example shown above, we changed the coordinate system manually to find the simple classification rule.

However, if we developed a method of automatic modification of the coordinate system that would use the percentage feedback of correctly classified points, then we would conduct machine learning that automates the process of searching for a better data representation for a specific purpose that is, the classification of points.

Machine Learning Algorithms

Machine learning algorithms contain mechanisms to automatically search for a method (operation/rules) of transforming data into such a representation that will facilitate the execution of a specific sentence, e.g. **classification, regression, or prediction**.

These operations can consist in changing the coordinate system, linear projection, shifts, but they can also be non-linear, performing non-linear, complex transformations.

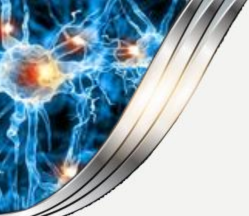
Machine learning algorithms, however, are not creative (intelligent) enough to independently invent such transformations, but only change the parameters of these transformations and use various combinations of them for a previously defined **set** of them, called the **hypothesis space.**

Symbolic Artificial Intelligence

In the 1950s and 1980s, the dominant approach to artificial intelligence was to define a sufficiently large number of knowledge processing rules that were used in the so-called expert systems.

Symbolic artificial intelligence has been successful in solving well-defined logical problems, i.e. playing chess, but it has proved unable to define clear rules for solving nondeterministic or more complex problems, i.e. classification and recognition of images, speech recognition, or automatic text translation.

Symbolic artificial intelligence has been replaced by various forms of machine learning, including deep learning.



Deep Learning

Deep learning is used to define learning on multilayer structures that gradually (hierarchically) transform the input data representation into a form that allows for the grouping of similar and frequent data patterns (i.e. certain combinations of them) present in the input data for use in classification or regression (e.g. prediction).

Deep structures consist of many (tens or even hundreds) layers that perform various transformations, selections and non-linear transformations, which distinguishes them from the so-called shallow structures, which usually consist of one, two or three layers of such transformations, and this learning is called shallow learning.

These structures are often referred to as artificial neural networks, but it is worth noting that they have little in common with biological neural networks and the way data is processed in the brain, although some ideas have been inspired by biological processes.

Structure of Deep Networks

The structure of the deep network consists of successive layers that perform various transformations and operations in different order:

- **adaptive filtration** (in convolution layers - conv),
- **selection** (e.g. maximum value - maxpooling),
- **calculations** (e.g. average value - avgpooling),
- **regularization** (e.g. using dropout),
- **normalization** (e.g. by balancing the results in the softmax layer).
- As a result, there is **a multi-stage process** of transforming the input data representation into a form that enables the achievement of the set goals.
- There is no real intelligence here, and the idea is quite simple, but due to the large number of such transformations in different configurations, incredible results can be achieved!

Operation of Deep Networks

The operation of deep networks is based on a multi-stage (hierarchical) mapping (transformation) of input data (e.g. images, signals) to the target output data (e.g. class labels or numerical values).

The mapping process is performed with a sequence of simple data transformations in the layers.

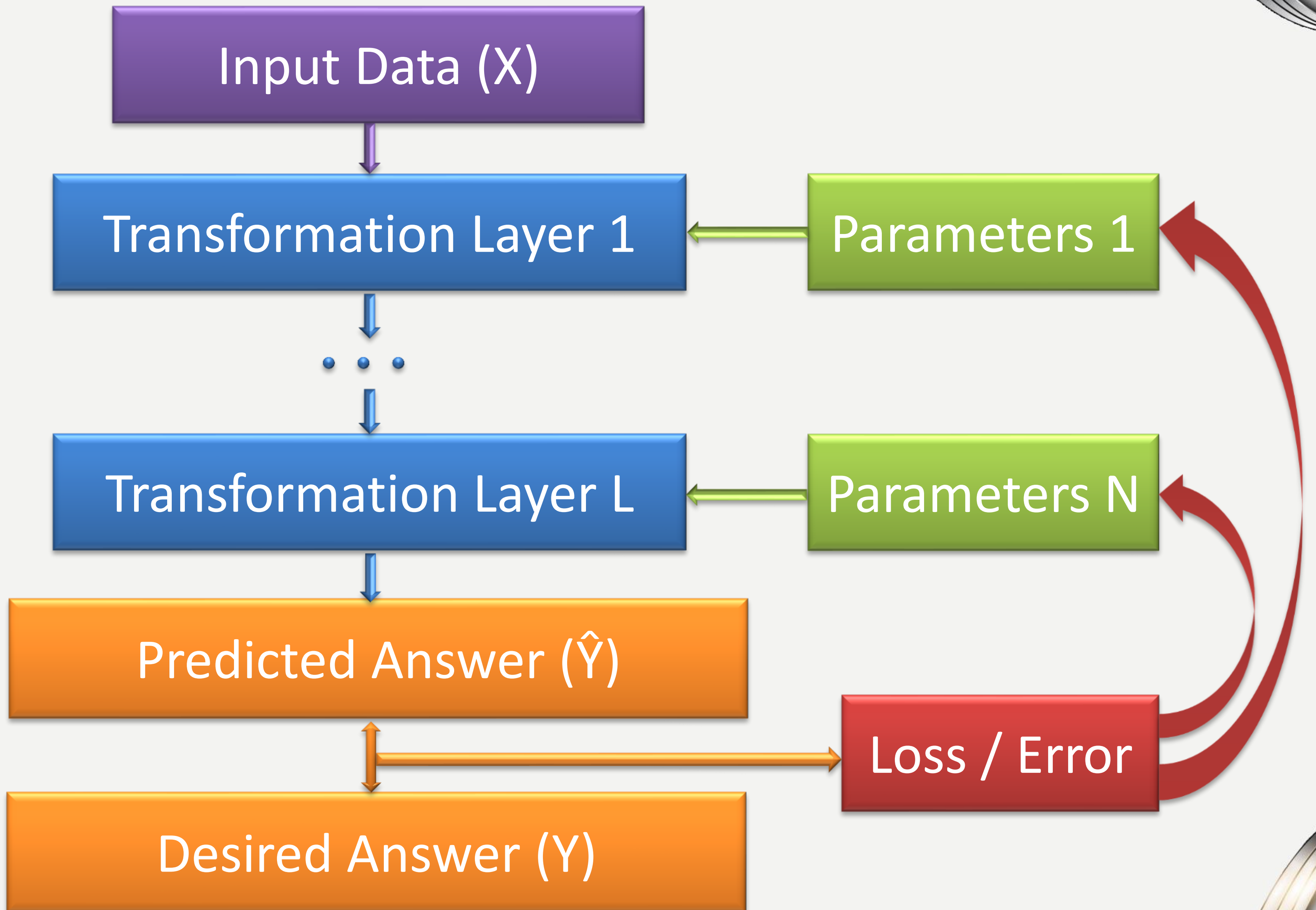
The type of transformation is usually determined by the network designer (based on experience or some algorithm, e.g. evolutionary), and the parameters (weights) of these transformations are determined in the learning process.

Due to the multi-stage nature of this learning, we call it "deep".

The deep learning process, therefore, is to gradually find a set of weights (of which there may be millions) for all layers of the network so that the network correctly maps input to output.

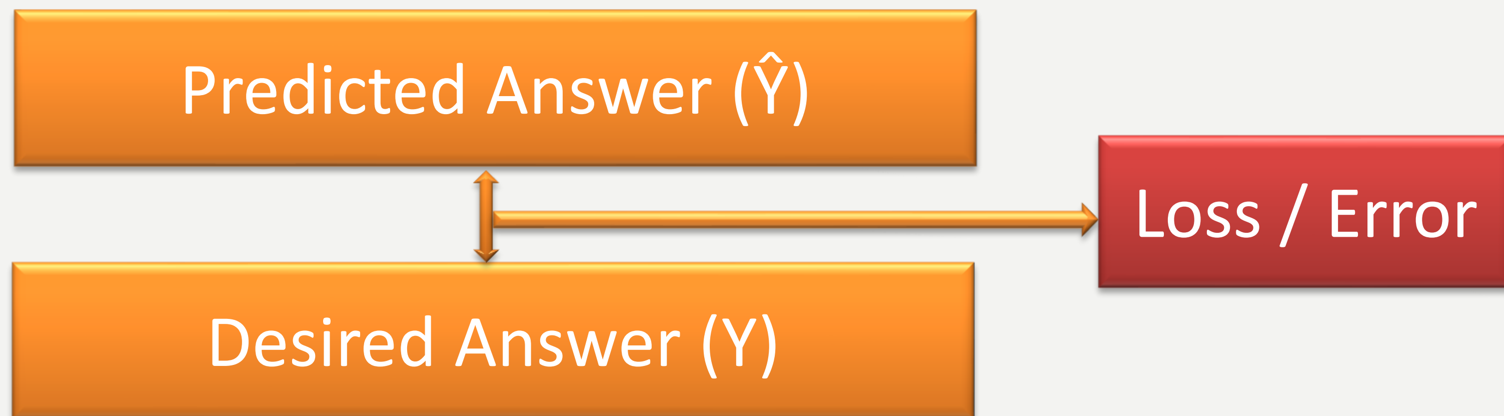
This task is not easy, because the change of weights affects all mapping operations in subsequent layers of such networks!

Adaptable Network Parameters



Network Learning Process Control

The supervised learning process of the deep network is controlled by comparing the response (prediction) received Ŷ and the desired answer Y, which determines the error for each network input (X).



Based on these errors, the loss (goal) function is defined, which is then minimized, which indirectly aims to reduce errors (differences | Y - Ŷ |) for individual X inputs.

For this purpose, we most often use the backpropagation algorithm, which is derived from the optimization theory.

Learning Mechanism

Initially, network weights are assigned small random values for which the network performs a series of random transformations, resulting in output Ŷ that significantly deviates from the desired Y.

Based on the calculated loss/error, the network parameters (including weights) are adjusted, which is called the weight tuning or network training process, and as a result, the loss/error value usually decreases with each interaction of the network learning process.

The learning process consists of many such weight-tuning phases for the calculated output values. One-time use of all training data is called a learning stage/training cycle/epoch.

The learning process therefore takes place in a training loop (epoch), and this loop is repeated many times until a satisfactory minimization of the loss function (error/goal) is obtained for the training data.

Applications of Deep Learning

Deep Learning has led to breakthroughs in many areas:

- image classification close to the human level,
- speech recognition at a level similar to human capabilities,
- handwriting transcription at a level similar to human capabilities,
- improvement of machine translation,
- improved text-to-speech conversion,
- the advent of assistants such as Google Now and Amazon Alexa,
- precise control of autonomous robots,
- construction of driving, walking and flying autonomous vehicles,
- improving the selection of advertisements in Google, Baidu and Bing services,
- building recommendation systems based on large data sets,
- improving the results returned by search engines,
- the ability to answer questions asked in natural language,
- defeating grandmasters in games like Chess or Go.

Advantages of Deep Learning

Deep learning typically provides:

- greater learning accuracy for many learning problems;
- better generalizing properties of trained models;
- simplicity and automation of features processing, often not requiring pre-processing of data, required by many other methods;
- deep convolutional models also make the recognition of features independent of their location in the input space (usually a matrix or sometimes an input vector), which greatly facilitates network training and reducing the number of necessary model parameters;
- joint processing of all layers of feature hierarchy representation without the need for designer intervention, usually giving better solutions than shallow model stacks;
- scalability of parallel computing to many GPUs with many TPU processors (tensor processor unit);
- the possibility of training on new data, as well as transferring large trained models to other similar tasks (transfer learning), which gives the possibility of teaching also on small data sets.

Kaggle.com

Competitions organized on Kaggle.com, in which thousands of analysts take part (due to prizes reaching thousands or even millions of dollars), allow you to find and choose the methods that work best.

Recent competitions have been dominated by gradient-enhanced machines and deep learning on various types of networks:

- gradient-enhanced machines are used for well-structured data (XGBoost library);
- deep learning is used in the case of perceptual problems, i.e. image classification (Keras library).

The example of the Kaggle website shows that organizing public competitions is a great way to motivate researchers and scientists to develop their projects, compare the results of different algorithms, which also contributed to the development of deep learning theory.

NVIDIA and CUDA

In 2007, NVIDIA launched the CUDA project and provided the programming interface for its GPUs:

https://developer.nvidia.com/about-cuda.

In parallel computations, several graphics chips could then replace an entire cluster of general purpose CPUs.

Deep neural networks mainly consist of many matrix multiplication operations, so they can be effectively parallelized.

Around 2011, Dan Ciresan and Alex Krizhevsky (pioneers) began work on writing a neural network implementation using CUDA technology.

Today we can use powerfull GPU processors:

- NVIDIA TITAN X - the graphics processor has a computing power of 6.6 TFLOPS, replacing several hundred laptops.
- NVIDIA TESLA V100 - the graphics processor has a computing power of 7 TFLOPS and 112 TFLOPS on tensors!

Development of Deep Learning Methods

In deep learning, the biggest problem was with gradient propagation through deep layer stacks, causing the problem of vanishing or exploding gradients.

Currently, we can teach hundreds or even thousands of layers due to the introduction of simple but effective tweaks to the training algorithm, enabling better propagation of gradients, i.e.:

- better functions of activating the layers of neurons;
- better weight (Xavier's) initiation schemes for individual layers;
- better optimization schemes, i.e. RMSProp and Adam;
- methods and structures for better regularization of the learning process, dropout, residual connections, shortcuts (in ResNet networks), inception blocks, etc.;
- convolutions separated depending on the depth.

Tensors in Machine Learning

Tensor – is a numerical data structure with any number of dimensions, so it is a generalization of a **matrix**, which is a two-dimensional structure. **Dimensions** are also called axes and the number of **axes** (`ndim`) is called **a rank**.

A scalar is **a zero-dimensional tensor (0D)**, containing only one number (scalar tensor, rank 0 tensor), i.e. (`ndim = 0`), e.g.:

```
>>> import numpy as np
>>> x = np.array(28)
>>> x.ndim
0
```

A vector is **a one-dimensional (1D) tensor** with exactly one axis (rank 1 tensor), e.g.:

```
>>> x = np.array([4, 28, 16, 5, 8])
>>> x.ndim
1
```

Tensors in Machine Learning

A matrix is a two-dimensional (2D) tensor that has two axes, called rows and columns (rank 2 tensor), e.g.:

```
>>> x = np.array([[4, 28, 16, 5],
                  [32, 2, 15, 4],
                  [1, 9, 12, 18]])
>>> x.ndim
2
```

The first axis of the matrix is rows, the second axis of the matrix is columns.

We can also create multidimensional tensors (tensors of N rank), which are used, for example, to record color images or training sets.

Tensors in Machine Learning

Three-dimensional (3D) tensor has three axes called rows, columns and depth / filters (rank 3 tensor), e.g.:

```
>>> x = np.array([[[4, 28, 16, 5],
                   [32, 2, 15, 4],
                   [1, 9, 12, 18]],
                  [[6, 7, 36, 25],
                   [4, 20, 21, 7],
                   [11, 17, 6, 2]]])
>>> x.ndim
3
```

In deep learning, we deal with **0 to 5 dimensional tensors**.

Tensor Attributes

Tensors are defined by three attributes:

Rank is the number of axes (dimension) of the tensor **(ndim).**

Shape is a tuple of integer values that define the number of dimensions of individual tensor axes (shape), e.g.:

- for the presented matrix x it was (4, 3)
- for the vector x shown it was (5,)
- and for a scalar it would be an empty tuple, i.e. ()

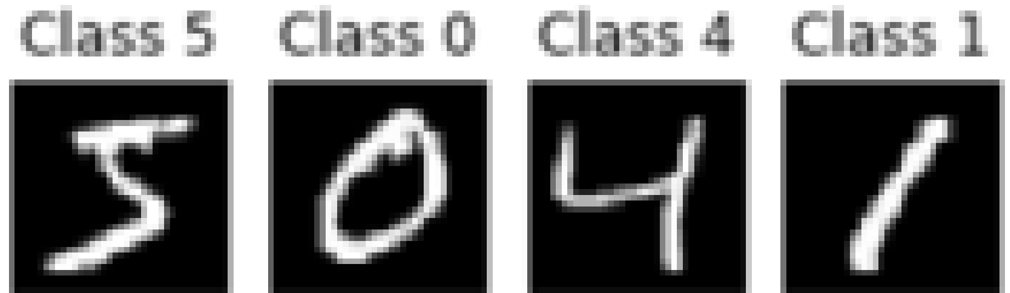
Data type (`dtype`), which in the case of tensors is usually a numeric type, i.e.: `float32, uint8, float64`, sometimes also `char` containing strings, but the Numpy library and most other libraries one do not support string tensors!

Tensor Attributes

Examples of tensor attributes display for the MNIST set:

```
import numpy as np
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
print(train_images.ndim)
print(train_images.shape)
print(train_images.dtype)
```

```
3
(60000, 28, 28)
uint8
```



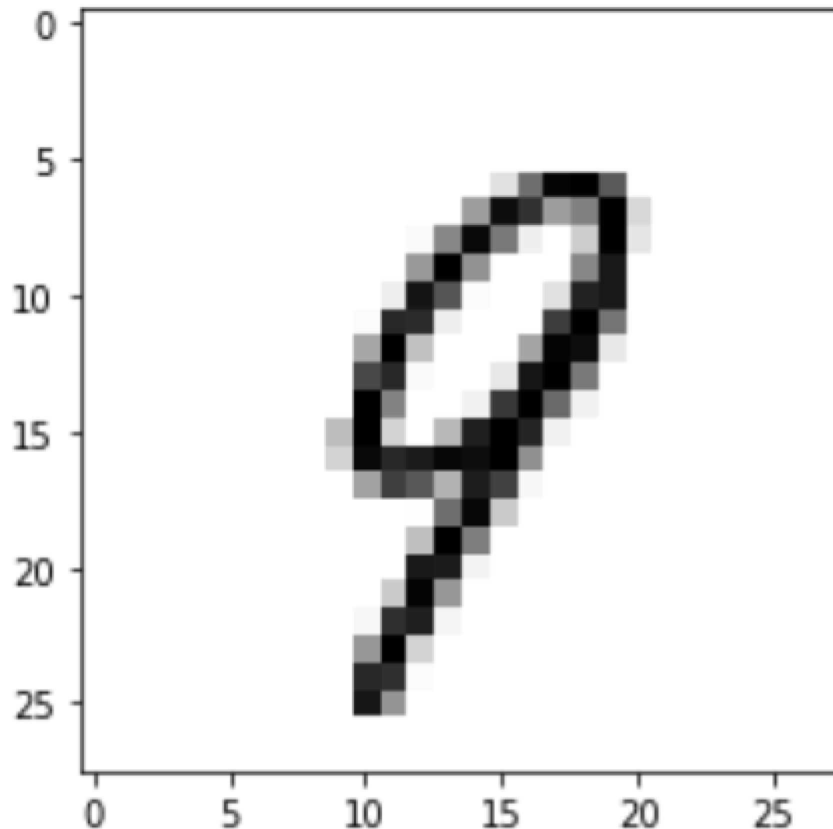
So we are dealing with a 60,000 x 28 x 28 three-dimensional (shape) training data tensor with eight-bit integers ranging from 0 to 255, where 60000 is the number of black and white images with dimensions of 28 x 28.

MNIST Data Display

Data display:

```
import matplotlib.pyplot as plt
digit = train_images[22]
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```

We use the popular matplotlib library:



Tensor Slicing

We can cut a smaller tensor from a bigger tensor, e.g .:

```
my_slice = train_images[10:30]
print(my_slice.shape)
```

```
(20, 28, 28)
```

The resulting tensor thus contains images 10 to 30 from the original train_images tensor.

We can also cut out only image clippings from the entire collection, for example:

lower right corners:

```
my_slice_corners = train_images[:, 14:, 14:]
```

image center parts:

```
my_slice_centers = train_images[:, 7:-7, 7:-7]
```

Tensor Axes in Deep Learning

In deep learning, we use certain standards for recording data in tensors:

- axis 0 - is the axis of samples / batch (batch size),
- axis 1 - this is the timeline if present, e.g., if we are considering sequential samples,
- consecutive axes are data axes of samples depending on their dimensions 1D, 2D, or 3D.
- In addition, during training, we usually use certain slices of the data tensor (of some input or output data), which we call batches (or mini-batches) with a certain defined size, e.g. 256. Then the nth batch we get using:

```
batch = train_images[256 * n:256 * (n + 1)]
```

Examples of Data Tensors

Data tensors, depending on their type, usually have defined axes as follows:

- vector data (2D tensors): (samples, features)
- sequence data, e.g., time series, signals (3D tensors): (samples, time, features)
- color and monochrome images (4D tensors): (samples, height, width, channels) [TensorFlow] or (samples, channels, height, width) [Theano] where the number of channels for monochrome images = 1
- video materials (5D tensors): (samples, frames, height, width, channels) [TensorFlow] or (samples, frames, channels, height, width) [Theano]
- Keras supports both of these formats: TensorFlow and Theano.

Tensor Operations

Tensors can be added, multiplied etc., e.g.:

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

The * and + operations process each element of the tensor independently (the-wise element), so they are perfect for parallelization!

Numpy uses well-optimized, low-level BLAS numerical routines.

The dot (x, y) operation is a tensor product (dot product), i.e. an dot product of two vectors, a matrix and a vector or two matrices. It can be calculated when the result is a matrix with dimensions, i.e. the matrix x must have as many rows as the matrix y has columns:

```
x.shape[1] == y.shape[0]
```

```
(x.shape[0], y.shape[1])
```

Projection of Tensors

When adding tensors with different dimensions (e.g. vector and matrix), the smaller tensor (vector) is broadcasting virtually around the projection axis so that it takes the same shape as the larger tensor (z such as x):

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

```
x = np.random.random((32, 4, 16, 10))
y = np.random.random((16, 10))
z = np.maximum(x, y)
print(x.shape)
print(y.shape)
print(z.shape)
```

```
(32, 4, 16, 10)
(16, 10)
(32, 4, 16, 10)
```

Tensor Shape Change

Tensors can reshape as needed:

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

```
(60000, 28, 28)
(60000, 784)
```

We can also transpose the tensor:

```
x = np.random.random((10, 5))
print(x.shape)
x = np.transpose(x)
print(x.shape)
```

```
(10, 5)
(5, 10)
```

Geometric Interpretation

Operations on tensors can be imagined as geometrical operations on vectors in a multidimensional space.

If you took two or more colored pages, fold them together and crumple them, then a crushed ball would be created.

The operation of grading the points on the sheets would be like the operation (of that crushed ball of sheets of paper) that would have to be done to straighten the pages and easily separate the pages from each other.

Mini-batch Stochastic Learning

By mini-batch we define a certain subset of the training set, which is used in one step of training the network to adapt its parameters.

The size of the mini-batches is usually constant during training, and its size is often chosen as the power of 2, e.g. 64, 128, 256, 512, 1024.

The size of the mini-batches significantly affects the learning speed and the possibility of getting stuck or getting out of local minima or saddle points. So it is worth changing this value sometimes during the learning process, adjusting it to the rate of decrease of the learning error.

In the initial stage of learning and when the error decreases too slowly, it is worth reducing the size of the mini-batches, and in the final stages of learning, to stabilize the learning result and avoid fluctuations around the minimum, it is worth increasing the number of training patterns in the mini-batches, and even covering the entire training set with one big batch.

Anatomy of Neural Networks

Neural networks are made up of neurons, which are most often grouped into **layers**, in which these neurons can determine their values in parallel, and the layers are connected sequentially or form a more complex flow structure.

We stimulate neural networks with **input data** (usually affecting neurons in the input layer), for which neurons in subsequent layers make calculations until we obtain **results** (in the output layer).

The **result** of the network operation (**classifications** or **predictions**) for the input data **for supervised learning** is compared with the **desired labels or values**, calculating the error (loss) using the defined **loss function**, and then using the selected **optimizer**, we correct the errors, adapting network parameters (weights and others) to teach it the correct transformation of input data into output data.

Layers of Neural Networks

Layer:

- is a basic data structure used in neural networks;
- takes input from one or more tensors and computes the output as one or more tensors;
- some layers have no states (parameters/weights, e.g. max-pooling layers);
- can be fully (densely) connected (each of the neurons of the previous layer with each neuron of the current layer) or rarely connected (not all-to-all) to the previous layer.

Weights:

- are adaptable (free) parameters of the tensors, which are set using the stochastic slope algorithm along the gradient of the loss function.
- store the network knowledge about training data in the context of a defined output.

Layers

Now let's create a dense layer (connecting the network inputs to the 64 neurons of this layer on a all-to-all basis). It accepts only two-dimensional (2D) tensors as input, and the first dimension of the tensor must be 256 long, and the axis 0 - dimension sample - is undefined, so the layer will accept any size.

This layer will return a tensor, which first dimension will have a length of 64:

```
from keras import layers
layer = layers.Dense(64, input_shape=(256,))
```

Keras will take care of the automatic adjustment of the number of inputs in subsequent layers, so we do not have to provide an argument that defines the shape of the input tensor:

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, input_shape(256,)))
model.add(layers.Dense(128))
```

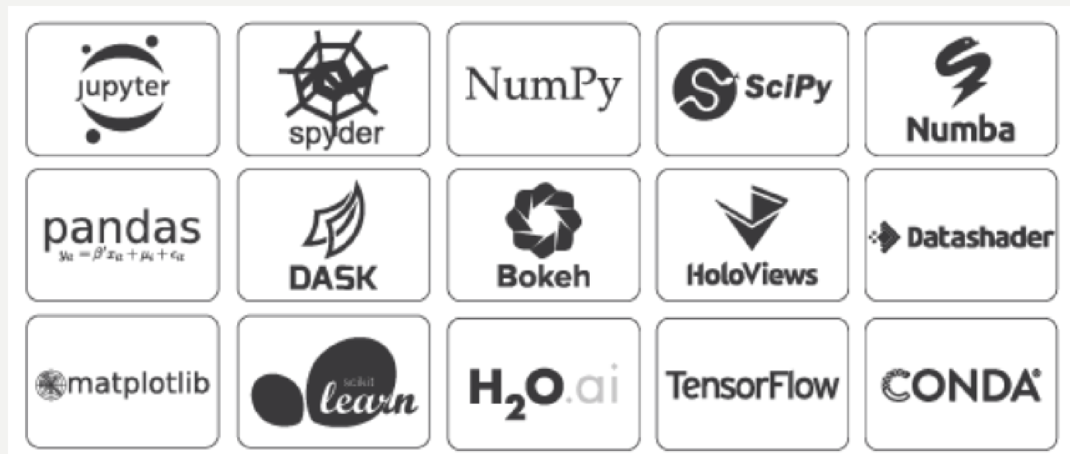
Install Anaconda & Jupyter Notebook

Install Jupyter using Anaconda with built in Python 3.7+

- It includes many other commonly used packages for scientific computing, data science, machine learning, and computational intelligence libraries.
- It manages libraries, dependencies, and environments with Conda.
- It allows developing and training various machine learning and deep learning models with scikit-learn, TensorFlow, Keras, Theano etc.
- It supplies us with data analysis including scalability and performance with Dask, NumPy, pandas, and Numba.
- It quickly visualizes results with Matplotlib, Bokeh, Datashader, and Holoviews.

And run it at the Terminal (Mac/Linux) or Command Prompt (Windows):

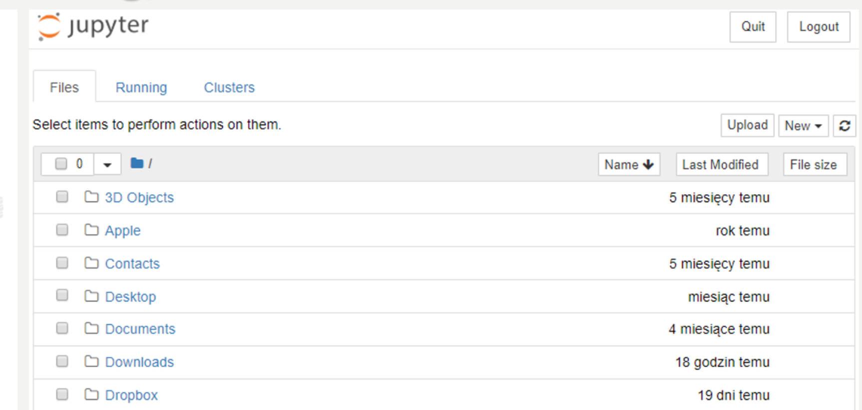
```
jupyter notebook
```



Jupyter Notebook

The Jupyter Notebook:

- is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative texts;
- includes data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.



We will use it to demonstrate various algorithms, so you are asked to install it.

Jupyter in your browser

Install a Jupyter Notebook

Anaconda Cloud

ANACONDA CLOUD

Search Anaconda Cloud

View ▾ Help ▾ adrianhorz... ▾

My Anaconda Landscape

Packages View all (0)

Get more information on how to upload a Package.

Notebooks View all (0)

Get more information on how to upload a Notebook.

Environments View all (0)

Get more information on how to upload an Environment.

Projects View all (0)

No projects yet, upload one here.

Favorites View all (0)

Favorite some packages, notebooks, and environments to get started!

Activity Feed View more

Welcome to Anaconda Cloud! 10 months and 22 hours ago

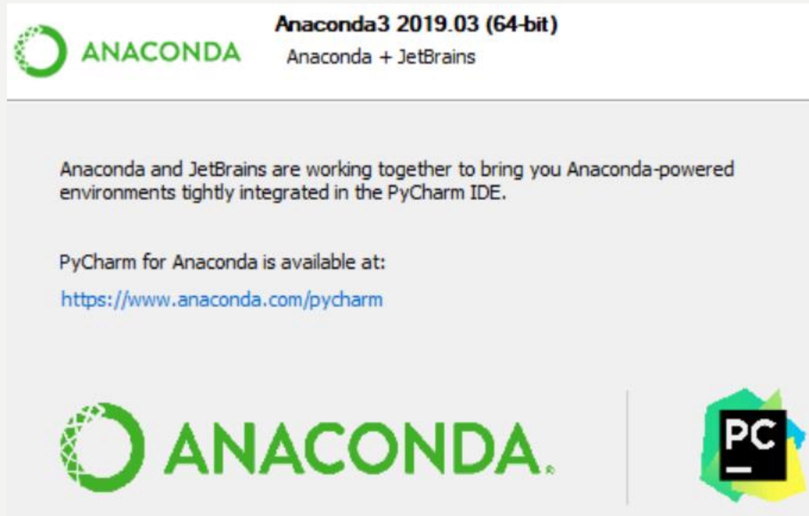
Anaconda Cloud allows you to create or distribute software packages.

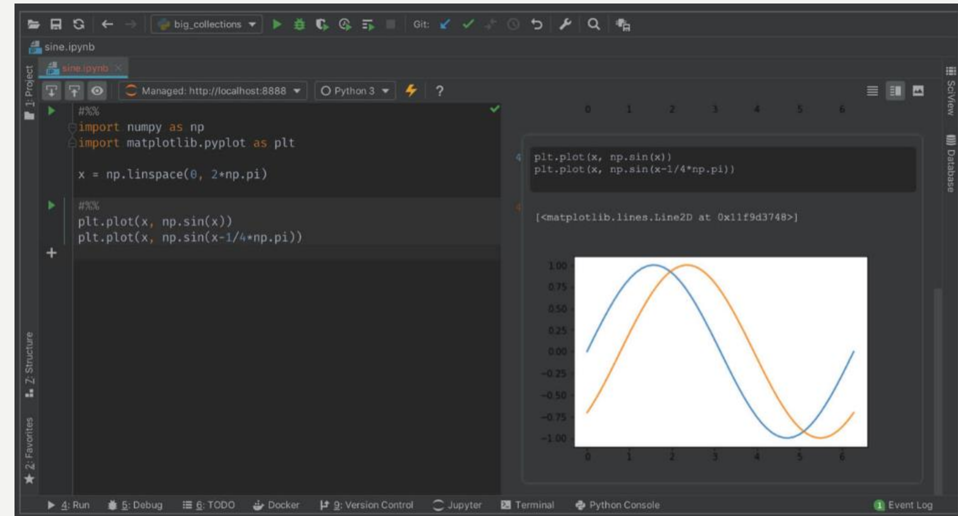
Getting started: Installing your first package

Getting started: Distributing your first package

PyCharm or Spider for Production

It is recommended to install PyCharm for Anaconda:





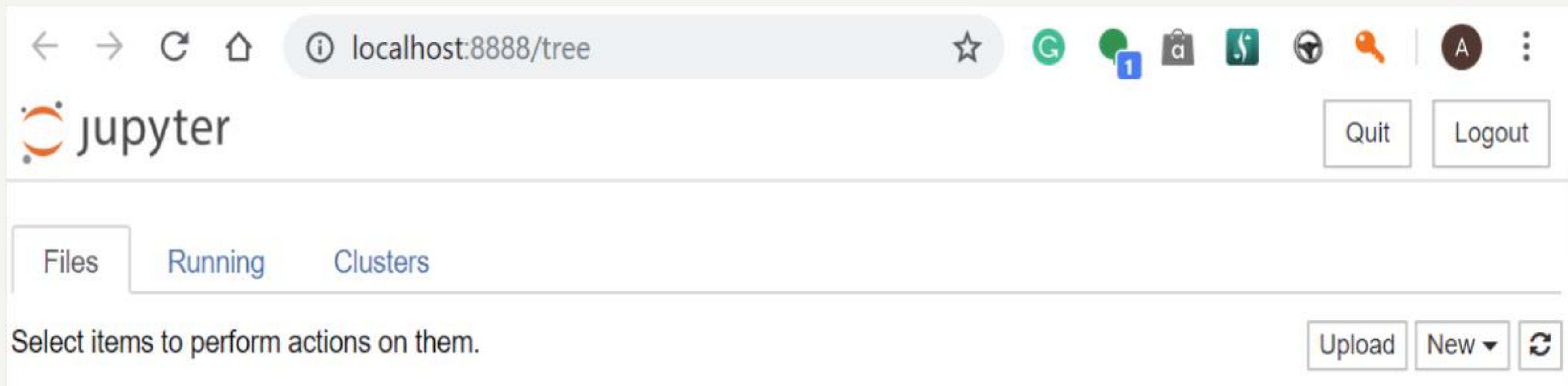
PyCharm is a python IDE for Professional Developers

- It includes scientific mode to interactively analyze your data

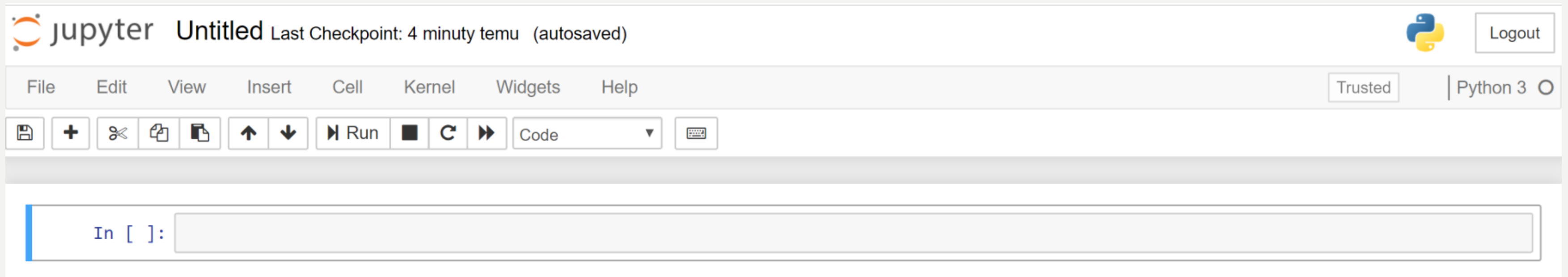
Uploading or Creating New Jupyter Notebook

Start a new Python notebook or use an existing one:

- Clicking **New** → Python 3 OR **Upload** an existing notebook

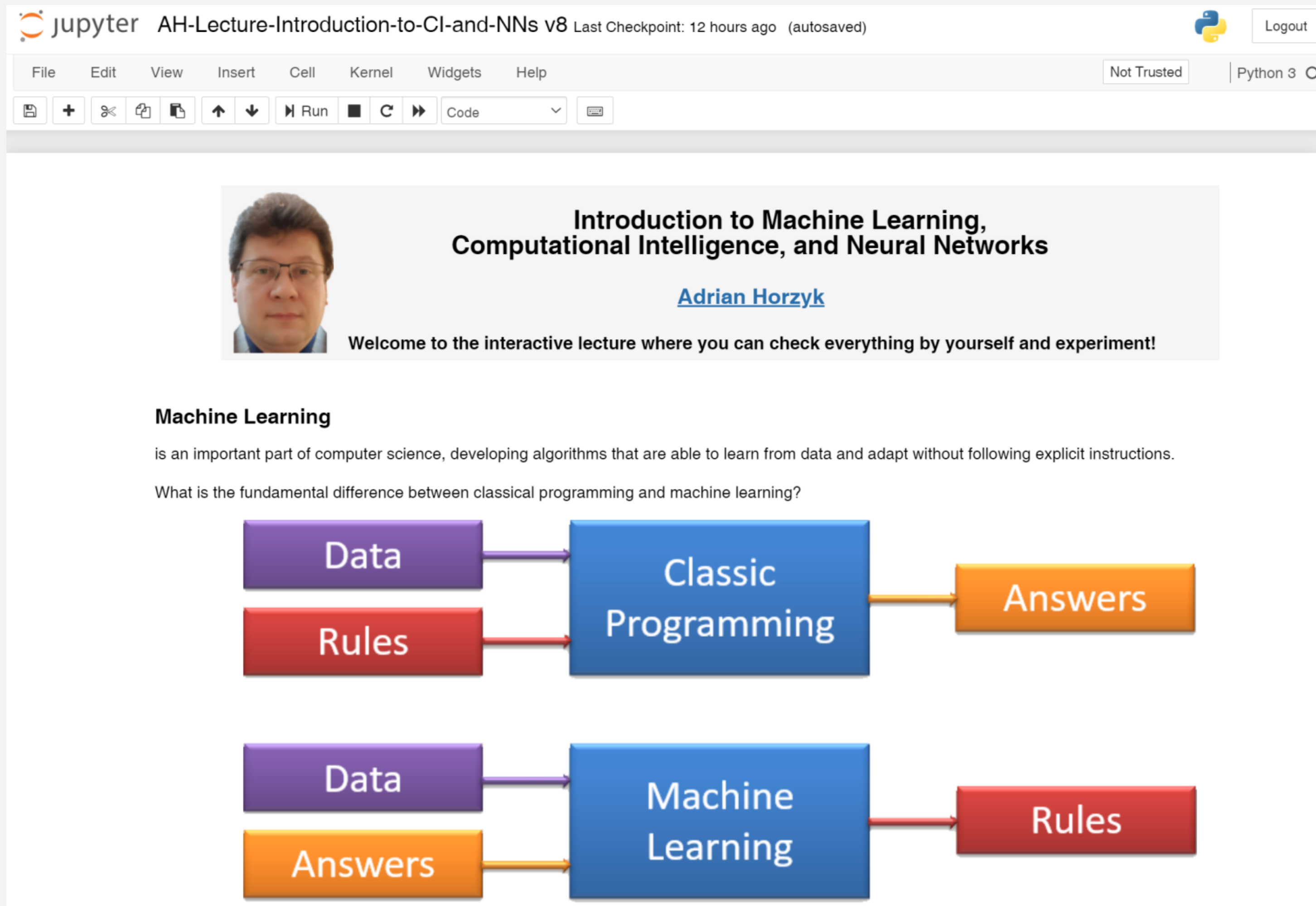


- A new Python project in the Jupyter Notebook looks as follows:



Let's go to the first Jupyter notebook

Now let's try to see it in practice and experiment with
our first Jupyter notebook:



BIBLIOGRAPY

1. Francois Chollet, "Deep learning with Python", Manning Publications Co., 2018.
2. Ian Goodfellow, Yoshua Bengio, Aaron Courville, "Deep Learning", MIT Press, 2016, ISBN 978-1-59327-741-3.
3. Home page for this course: http://home.agh.edu.pl/~horzyk/lectures/ahdydkbcidmb.php
4. JUPYTER: https://jupyter.org/
5. Holk Cruse, Neural Networks as Cybernetic Systems, 2nd and revised edition.
6. R. Rojas, Neural Networks, Springer-Verlag, Berlin, 1996.