

# NumPy

## Intro to Data Science



Jędrzej Ogrodowski

# What is NumPy

NumPy is a Python's library used to work with multidimensional arrays and advanced mathematical functions.

# What is NumPy

NumPy is a Python's library used to work with multidimensional arrays and advanced mathematical functions

Some math functions:

- shape manipulation
- sorting
- basic linear algebra
- basic statistical operations
- random simulation

**WHY NumPy???**

# Why NumPy?

All concepts used in Data Science, Machine Learning and AI in general are based on math.

# Why NumPy?

All concepts used in Data Science, Machine Learning and AI in general are based on math.

NumPy is used to apply math into our project EASIER and FASTER.

# Why NumPy?

All concepts used in Data Science, Machine Learning and AI in general are based on math.

NumPy is used to apply math into our project EASIER and FASTER.

NumPy is foundation of most Python libraries e.g pandas, SciPy, Matplotlib, OpenCV, scikit-learn.

# Why NumPy?

All concepts used in Data Science, Machine Learning and AI in general are based on math.

NumPy is used to apply math into our project EASIER and FASTER.

NumPy is foundation of most Python libraries e.g pandas, SciPy, Matplotlib, OpenCV, scikit-learn.

Well documented and a lot of tutorials on YouTube, Coursera...



# Why NumPy?

Arrays play a crucial role in data science as they form the foundation for handling and manipulating data. Whether you're working with raw data from sensors, financial time series, or images, arrays are the primary data structures used for numerical computing.

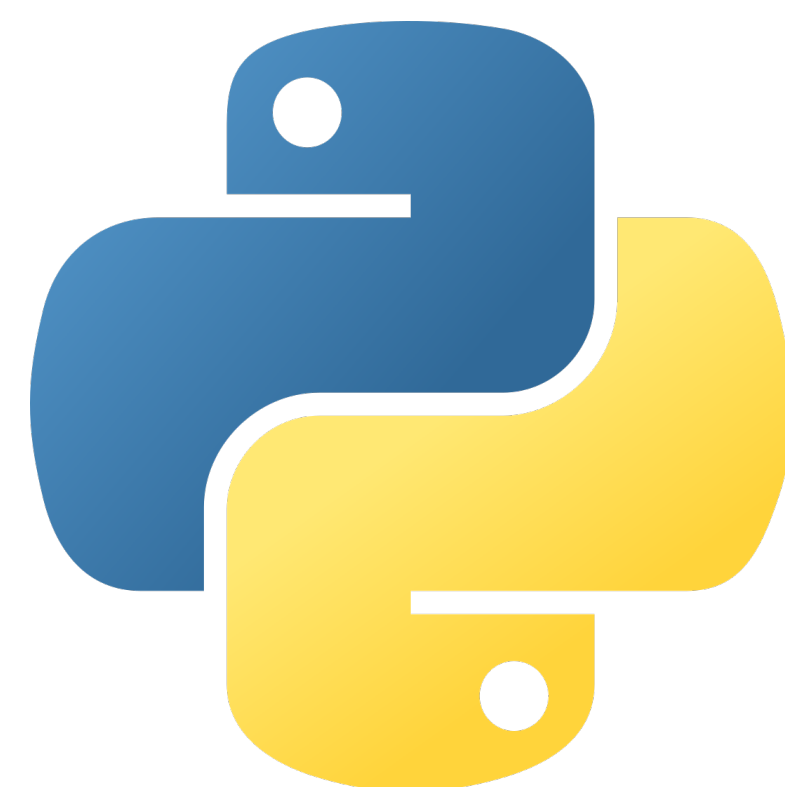
# Why NumPy?

Arrays play a crucial role in data science as they form the foundation for handling and manipulating data. Whether you're working with raw data from sensors, financial time series, or images, arrays are the primary data structures used for numerical computing.

NumPy arrays are optimized, faster and memory-efficient.



vs



# NumPy vs Python

NumPy take advantage of vectorization, contiguous memory blocks and low-level iterations.

# NumPy vs Python

NumPy take advantage of vectorization, contiguous memory blocks and low-level iterations.

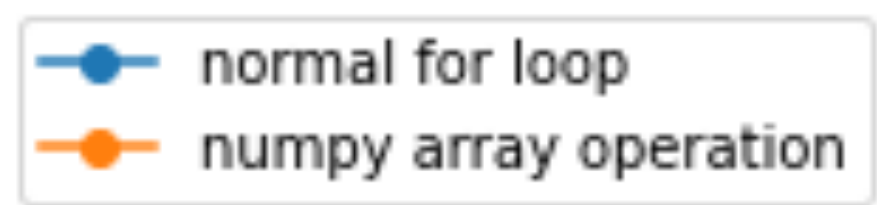
While operating on small data sets the difference may be unnoticeable, on big ones - NumPy will save your life (and time). Some sources say that NumPy's ndarray is up to 50x faster than python list.

# NumPy vs Python

NumPy take advantage of vectorization, contiguous memory blocks and low-level iterations.

While operating on small data sets the difference may be unnoticeable, on big ones - NumPy will save your life (and time). Some sources say that NumPy's ndarray is up to 50x faster than python list.

NumPy built-in mathematical functions speed up the project creation process. With one function you can achieve python's few lines but with better optimization.



Feature	Python List	NumPy Array
Storage	Dynamic type	Fixed type
Functionality	General-purpose	Mathematical and scientific operations
Speed	Slower due to type-checking and overhead	Faster due to vectorized operations and no type-checking
Memory	More due to overhead	Optimized and efficient storage



C when you use a C extension library in Python:



**All essential functions in notebook to this  
lesson on my Github.**

# Arrays

# Create array

We can create n-dimensional arrays.

```
arr1D = np.array([1, 2, 3])  
# [1 2 3]
```

```
arr2D = np.array([[1, 2, 3], [4, 5, 6]])  
# [[1 2 3]  
#    [4 5 6]]
```

As you can see, it's easy, but rarely done in real life projects.

# Generate

More often you will generate arrays.

```
arr = np.arange(start, stop, step)
```

```
arr = np.linspace(start, stop, numberOfElements)
```

```
zeroMatrix = np.zeros((2, 3), dtype=int)
```

```
onesMatrix = np.ones((2, 3), dtype=int)
```

```
fullMatrix = np.full((2, 3), 0.5)
```

# Generate random

It's really handy to generate sample data sets for our algorithms.

```
rng = np.random.default_rng()  
rng.random()
```

But what is more interesting, you can generate normally distributed data. (Quite often used). More about Normal Distribution on later lesson.

```
random.normal(loc=0.0, scale=1.0, size=None)
```

*loc: centre of distribution (mean)*

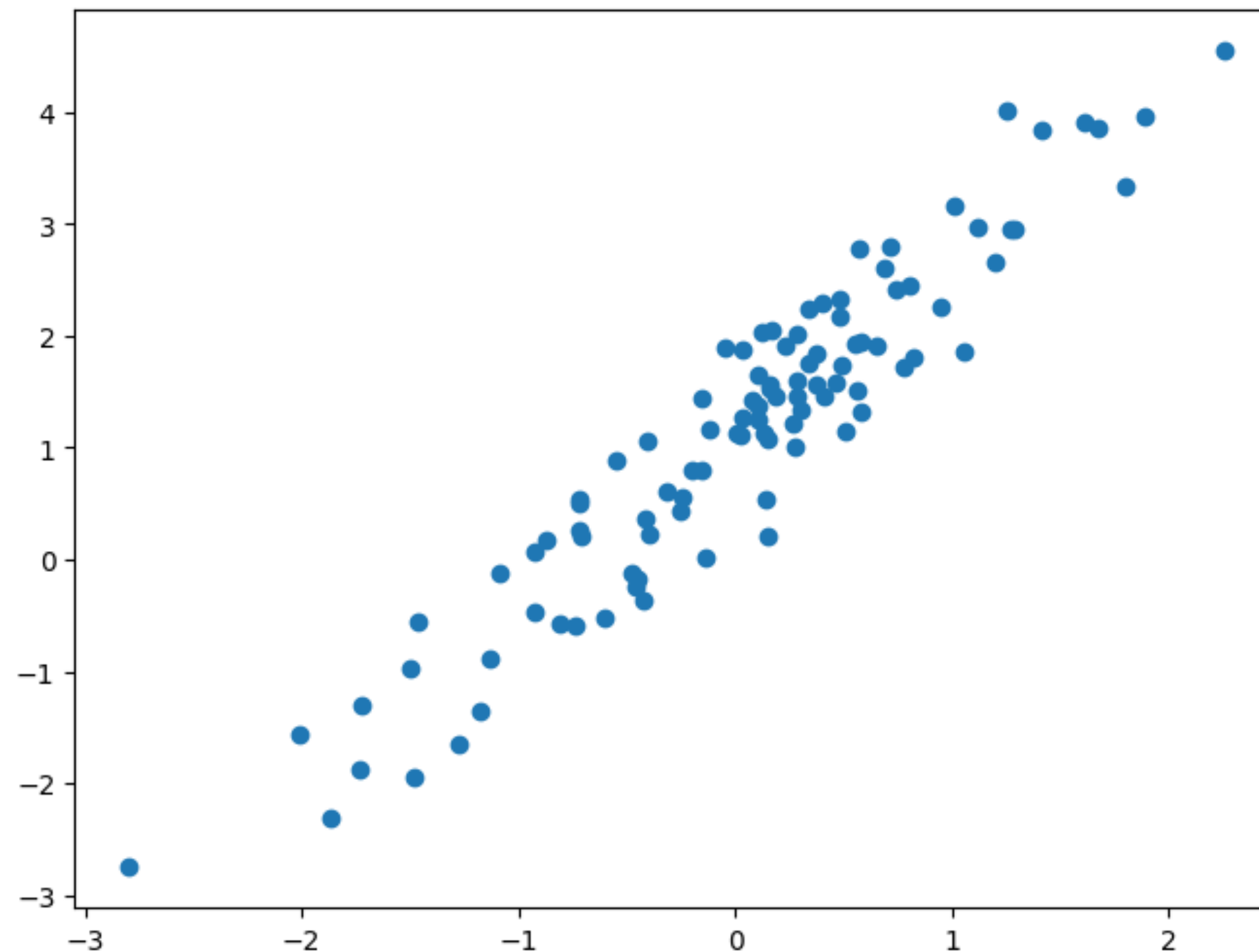
*scale: spread of distribution (standard deviation)*



# Generate random

Example of sample data set generated for Linear Regression algorithm.

```
X = np.random.normal(size=100)
y = X * 1.5 + 1 + np.random.normal(size=100, scale=0.5)
```



# Array element type

Arrays in NumPy has fixed type, thus they can contain only one type of data. We can check type, create array with specified type and change type.

```
typeOfArr = arr1D.dtype  
# int64
```

```
arrayWithFloats = np.array([1,2,3], dtype=float)  
# float64
```

```
newDataTypeArr = arrayWithFloats.astype(int)  
# int64
```



**Built in functions**

# Built in functions

```
np.sum(array)
```

```
np.mean(array)
```

```
np.max(array), np.min(array)
```

```
np.std(array)
```

```
np.var(array)
```

```
np.isnan(array)
```

```
np.power(array)
```

# Linear algebra

# Oh no maths again? YES.

## Linear algebra is most important math skill in Machine Learning.

Data sets are often represented as matrices, where every row corresponds to an observation and every column represents a function. This matrix illustration permits efficient manipulation and data analysis

Linear algebra is the base of linear regression, a widely used technique for modeling relationships between variables and depicting predictions.

Optimization: Linear algebra is important for optimization algorithms used in machine learning, including **gradient descent**, based on calculating gradients.

# Linear Algebra in NumPy

We can perform all important matrix operations.

```
rankOfMatrix = np.linalg.matrix_rank(matrix)
```

```
traceOfMatrix = np.trace(matrix)
```

```
detOfMatrix = np.linalg.det(matrix)
```

```
inversedMatrix = np.linalg.inv(matrix)
```

```
dotProduct = np.dot(vector1, vector2)
```

```
arrTransposed = array.T
```

And more...

# Rank of matrix

The rank of a matrix is the number of independent rows or columns it has.

```
rankOfMatrix = np.linalg.matrix_rank(matrix)
```

One useful application of calculating the rank of a matrix is the computation of the number of solutions of a system of linear equations.

# Trace of matrix

The trace of a matrix is the sum of the diagonal elements of a square matrix.

```
traceOfMatrix = np.linalg.trace(matrix)
```

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 3 \\ 11 & 5 & 2 \\ 6 & 12 & -5 \end{pmatrix}$$

$$\text{trace}(\mathbf{A}) = \sum_{i=1}^3 a_{ii} = a_{11} + a_{22} + a_{33} = 1 + 5 + (-5) = 1$$

# Determinant of matrix

The determinant is special number that provides information such as whether matrix is invertible.

```
detOfMatrix = np.linalg.det(matrix)
```

In particular, the determinant is nonzero if and only if the matrix is invertible. The determinant is completely determined by the two following properties: the determinant of a product of matrices is the product of their determinants, and the determinant of a triangular matrix is the product of its diagonal entries.



# Inversion of matrix

```
inverseMatrix = np.linalg.inverse(matrix)
```

An  $n$  by  $n$  square matrix  $A$  is called invertible if there exists an  $n$  by  $n$  square matrix  $B$  such that:

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$$

Where  $\mathbf{I}_n$  denotes  $n$  by  $n$  identity matrix.

$$\mathbf{A} = \begin{pmatrix} -1 & \frac{3}{2} \\ 1 & -1 \end{pmatrix} \rightarrow \mathbf{A}^{-1} = \begin{pmatrix} 2 & 3 \\ 2 & 2 \end{pmatrix}.$$

# Dot product of matrix

The dot product is a way to multiply two vectors that results in a single number.

```
dotProduct = np.dot(vector1, vector2)
```

For two vectors:

$$\mathbf{a} = [a_1, a_2, \dots, a_n], \quad \mathbf{b} = [b_1, b_2, \dots, b_n]$$

Dot product is defined as:

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

# Transpose of matrix

Transpose „flips“ matrix over its diagonal.

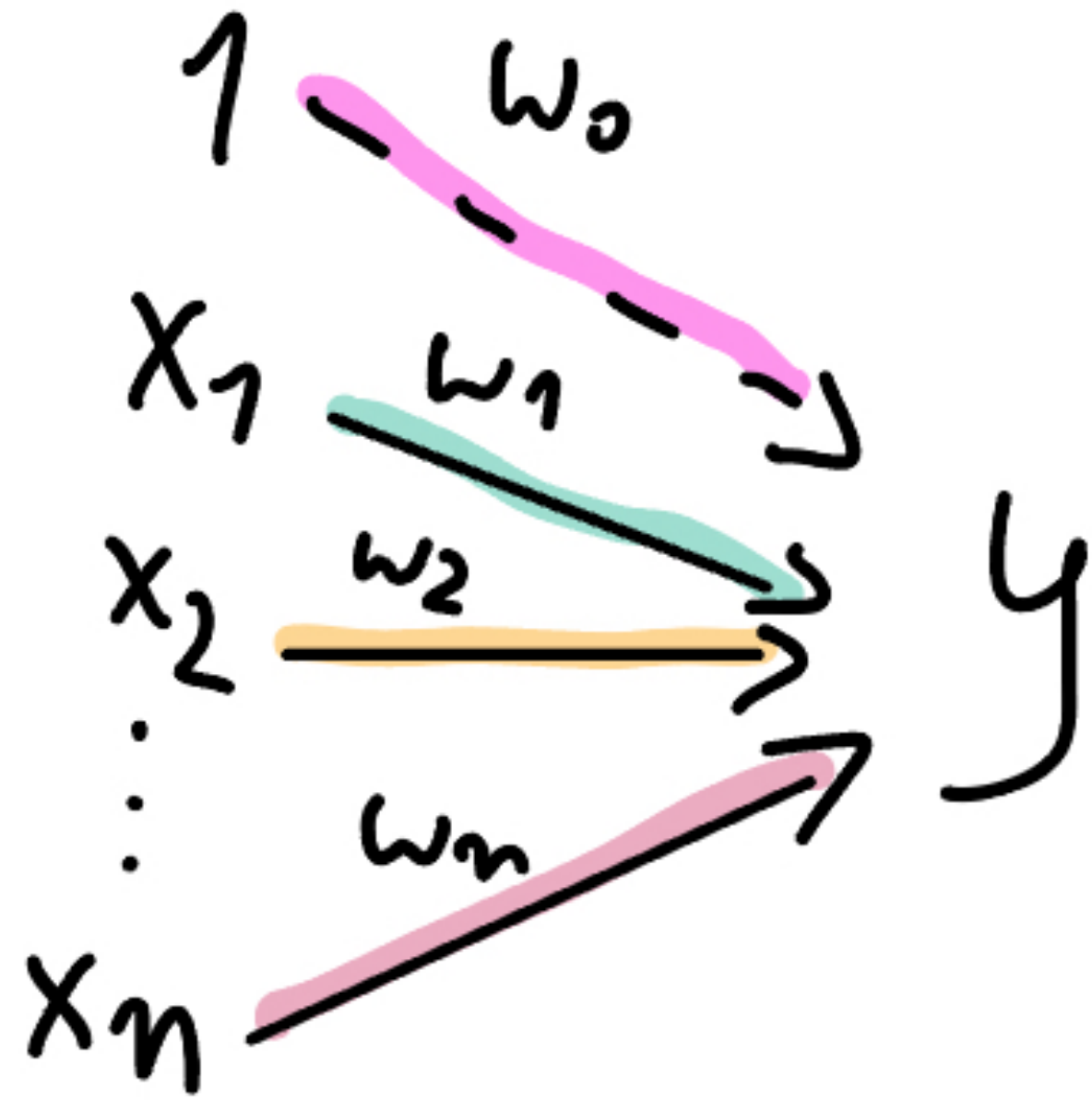
```
arrTransposed = arr2D.T
```

Sometimes we want to „match“ size of matrix to make it possible to multiply it.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

# Practical applications

# Linear regression

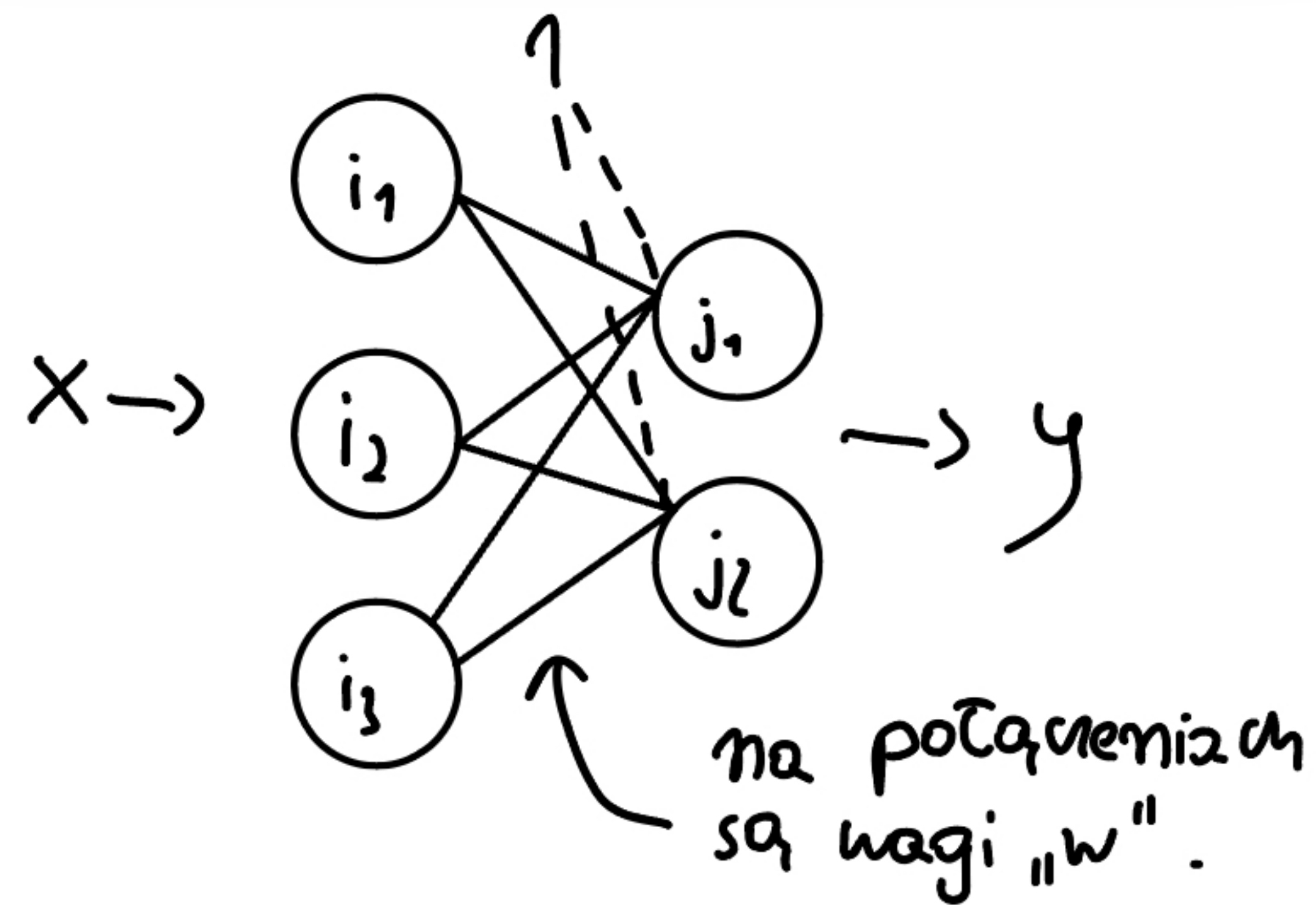


$$y = w_0 + x_1 w_1 + x_2 w_2 + \dots + x_n w_n$$

$$[1, x_1, x_2, \dots, x_n] \cdot \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = [y]$$

$$y = 1 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2 + \dots + x_n \cdot w_n$$

# Simple NN



$$[1, i_1, i_2, i_3] \cdot \begin{bmatrix} w_{01} & w_{02} \\ w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} = [j_1, j_2]$$

$$j_1 = w_{01} + i_1 w_{11} + i_2 w_{21} + i_3 w_{31}$$

$$j_2 = w_{02} + i_1 w_{12} + i_2 w_{22} + i_3 w_{32}$$

# CUDA and parallel computing

Feature	CPU	GPU
Cores	4,6,10,14,...,92,96	Thousands
Efficiency	Better at single-threaded tasks	Better at parallel computing
Computing power	Less effective when parallelized	Very effective in large-scale computations



# GPU in Data Science and ML

The use of CUDA (Compute Unified Device Architecture) is extremely important for data science and machine learning for several reasons, mainly due to the optimization of computational performance.

CUDA allows you to use the processing power of graphics cards (GPU) to perform complex parallel calculations.

Deep learning models such as neural networks require iterative computations during the training phase. Without a GPU, this time can be days or weeks, especially for large data sets.

**But where is the catch?**




smicro.pl

47 153,00 zł

NVIDIA Tesla V100 32GB CoWoS HBM2 PCIe 3.0 -...

**W magazynie, online**

 Bezpłatna dostawa

Google



gigaserwer.pl

31 904,80 zł

NVIDIA TESLA V100 Modul 32GB

**W magazynie, online**

Google



gigaserwer.pl

38 424,17 zł

NVIDIA TESLA V100 Modul 16GB

**W magazynie, online**

Google

# GPU in Data Science and ML

We can use online solutions. It is handy if we often use laptops.

One of such services is Google Colab, hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Free version is limited, if we want use more powerful GPUs, there is Colab Pro.

<https://www.datageeks.pl/geeks-blog/77-google-colaboratory>

<https://colab.google/>



```
from google.colab import drive  
drive.mount('/content/drive')
```

Mounted at /content/drive

```
[ ] import pandas as pd
```

```
dataset = pd.read_csv('/content/  
dataset.head()
```

	sepal.length	sepal.width	
0	5.1	3.5	
1	4.9	3.0	
2	4.7	3.2	
3	4.6	3.1	
4	5.0	3.6	

## Change runtime type

### Runtime type

Python 3

### Hardware accelerator



CPU



T4 GPU



A100 GPU



V100 GPU



TPU

Want access to premium GPUs? [Purchase additional compute units](#)

# CuPy

We can improve our NumPy and SciPy work by using GPU too. CuPy is an open-source library that provides NumPy/SciPy-like capabilities but runs on NVIDIA GPUs. It accelerates numerical computations by leveraging CUDA, a parallel computing platform, for better performance on large datasets and complex operations.

CuPy is an efficient choice for anyone looking to speed up large-scale data processing. It enables seamless transition for those familiar with NumPy/SciPy due to its similar API, making it a "drop-in" replacement with minimal code changes.

# CuPy vs NumPy

Using NumPy:

```
import numpy as np
x = np.random.rand(1000, 1000)
y = np.dot(x, x.T)
```

Using CuPy:

```
import cupy as cp
x_gpu = cp.random.rand(1000, 1000)
y_gpu = cp.dot(x_gpu, x_gpu.T)
```

# Summary



- Consider using NumPy in your projects.
- Think about how data is stored and how to take advantage of it.
- Think about using GPU on some extensive projects.