Algebra basics with Numpy/Scipy

Daniel García Hernández

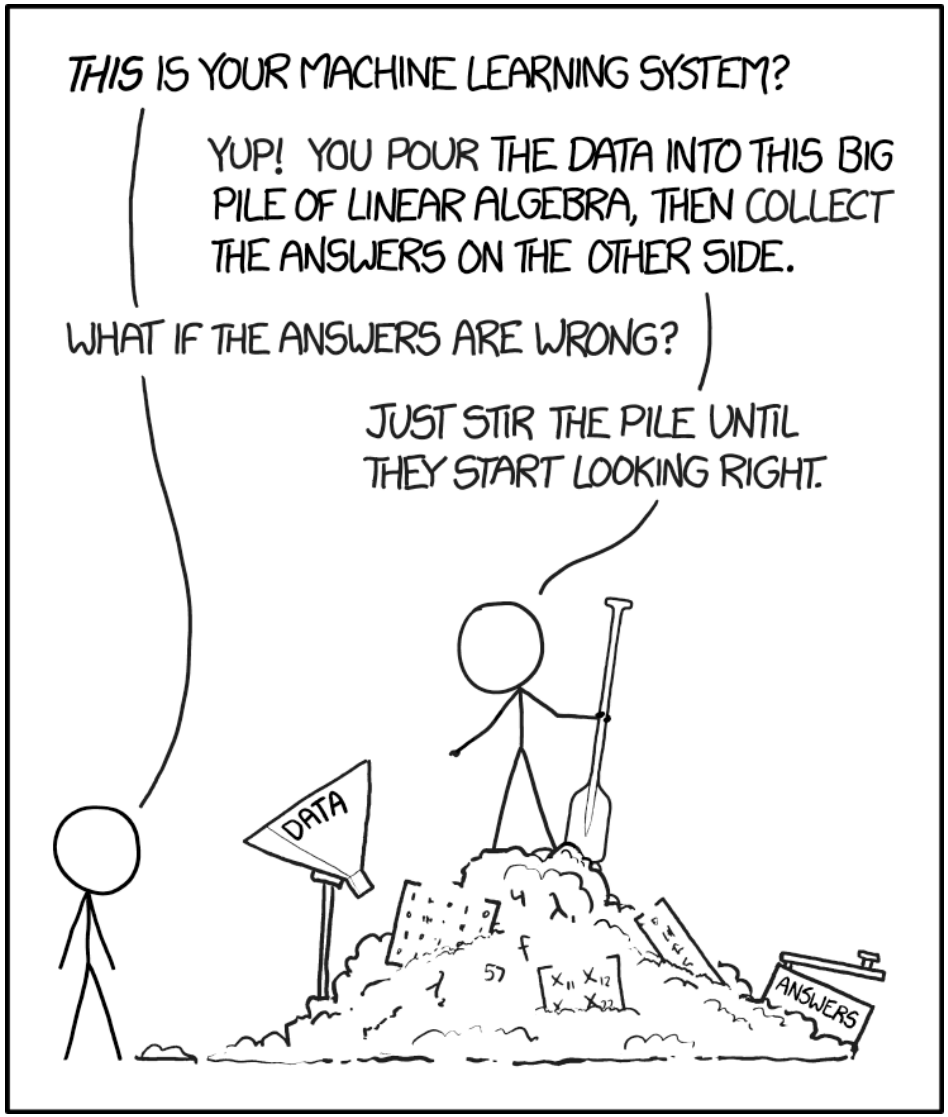
Master Big Data

February 20, 2020

: Overview

1. Why to learn algebra
2. Scalars, vectors, matrices, and tensors
3. Intro to NumPy
4. Operations with arrays

Why to learn algebra: xkcd comics



[1] https://xkcd.com/1838/

Why to learn algebra: Representing data

With algebra we can find the best representation of our data, and therefore storing, processing and handling it the best way possible:

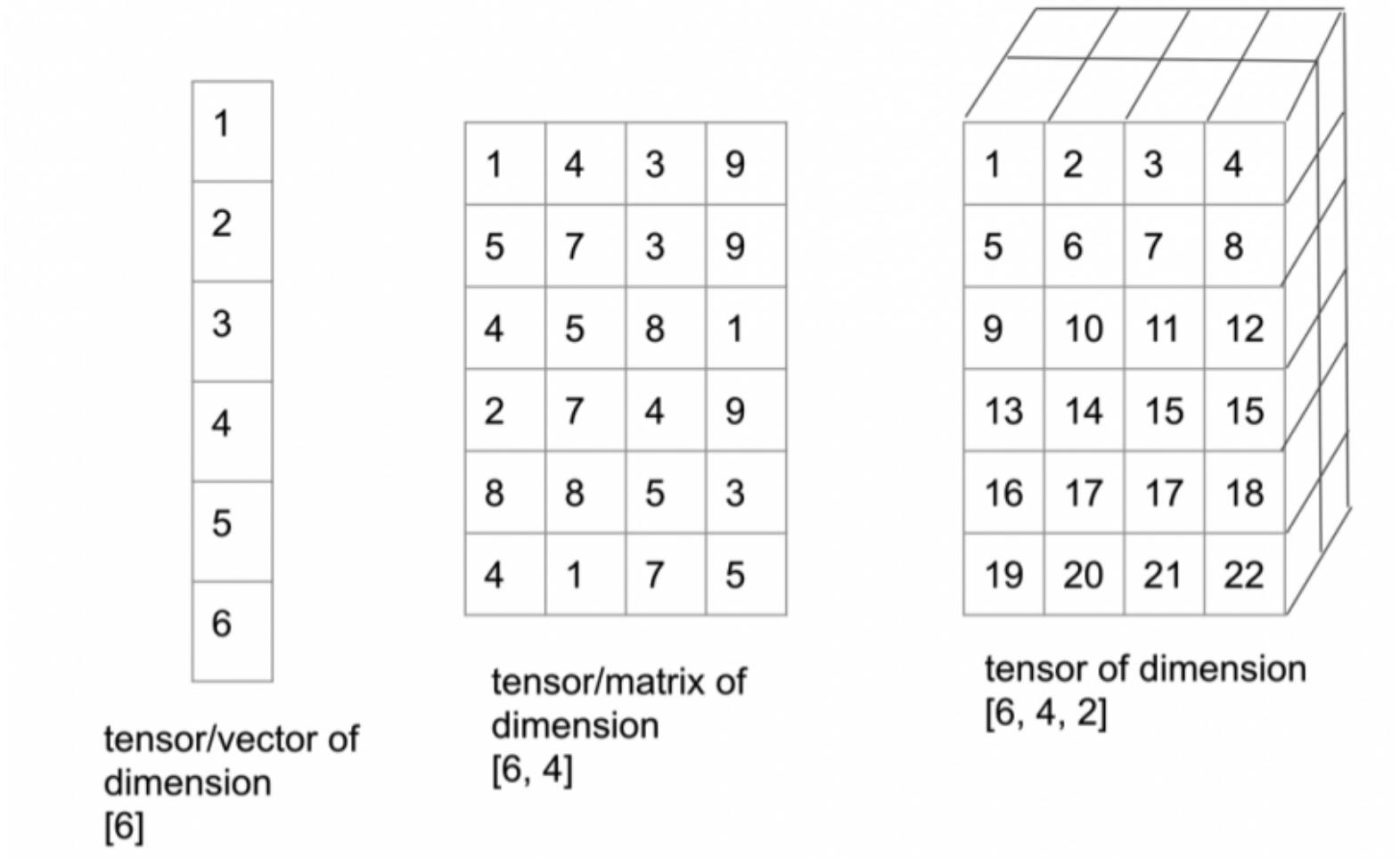
- Tabular datasets are matrices
- Pictures are matrices
- Time series are vectors
- Videos are tensors
- Sound is a tensor

Also, Deep Learning is all about neural networks and those are basically algebra and calculus on steroids.

Scalars, vectors, matrices, and tensors: Key algebra objects (1)

- Scalar: it's a single number -`int, float`.
- Vector: ordered 1D array of data. Can be a row or a column. They have a single index with which we can refer to every element in the vector
- Matrix: ordered 2D array of data, with double index. The first index refers to the row, the second refers to the column.
- Tensor: ordered nD array of data, with n indices.

Scalars, vectors, matrices, and tensors: Key algebra objects (2)



[1] https://builtin.com/data-science/basic-linear-algebra-deep-learning

Intro to NumPy: Description and elements

- `numpy`: numerical python
- Module to store and operate algebra objects
- The basic element is the array: `numpy.array`
- these arrays can be 0D (scalar), 1D (vector), 2D (matrix, or nD (tensors)
- `numpy` is what's under the hood of `pandas` and is the cornerstone of the machine learning and deep learning Python packages

Intro to NumPy: Creating arrays and its properties (1)

We can create an array from a python list, and it will inherit all the properties an array has.

- `arr = np.array([1, 2, 3])`

Once we have created an array, we can start getting its properties and passing `numpy` methods to it

- `arr.shape, arr.dtype, ...`

Intro to NumPy: Creating arrays and its properties (2)

- Shape: `shape`. Returns a tuple with as many elements as dimensions an array has, and each element represents the length across that dimension
  - Shape = (2, 3) means 2 rows and 3 columns
- Dtype: `dtype`. Data-type of the array's elements
  - `dtype('<U1')` means Unicode string
- Rank: `numpy.linalg.matrix_rank`. Returns the number of dimensions the array has:
  - Rank = 3 means the array has three dimensions (rows, columns, and depth, for example)

Operations with arrays: Element-wise operations

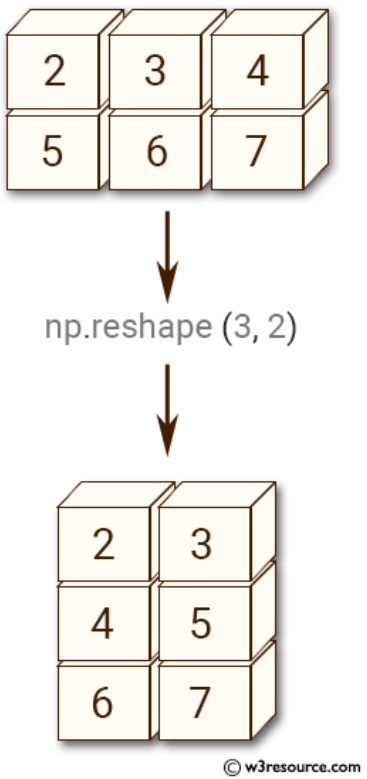
We can perform element-wise (for each element of the array) operations in an array and the result will change in values but not in shape. Arrays must have the same shape when operating two of them:

- sum/substract:
  `np.array([1, 2]) + np.array([5, 6]) = array([6, 8])`
- multiply/divide:
  `np.array([1, 2]) * np.array([5, 6]) = array([5, 12])`
- multiply by scalar: `np.array([1, 2]) * 2 = array([2, 4])`
- power: `np.array([1, 2]) ** 3 = array([1, 8])`
- logical ops: `array1[condition]`, `np.where()`(vectors only)

Operations with arrays: Rearranging elements (1)

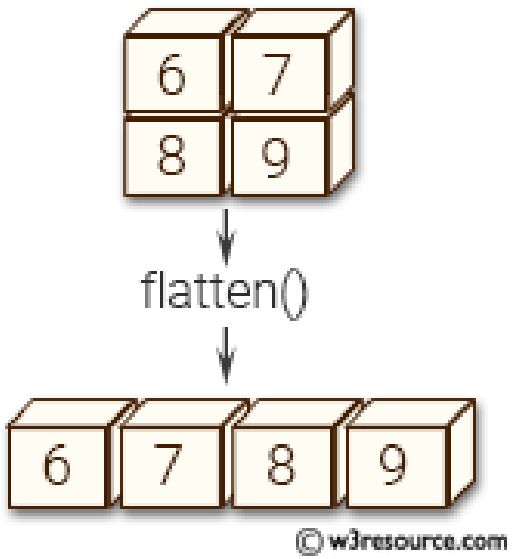
We can also slice, reshape, flatten and transpose (square matrices) and this will make the resulting array to change in shape without changing the individual values:

- `np.reshape(array)`



Operations with arrays: Rearranging elements (2)

- `np.flatten(array)`



Operations with arrays: Rearranging elements (3)

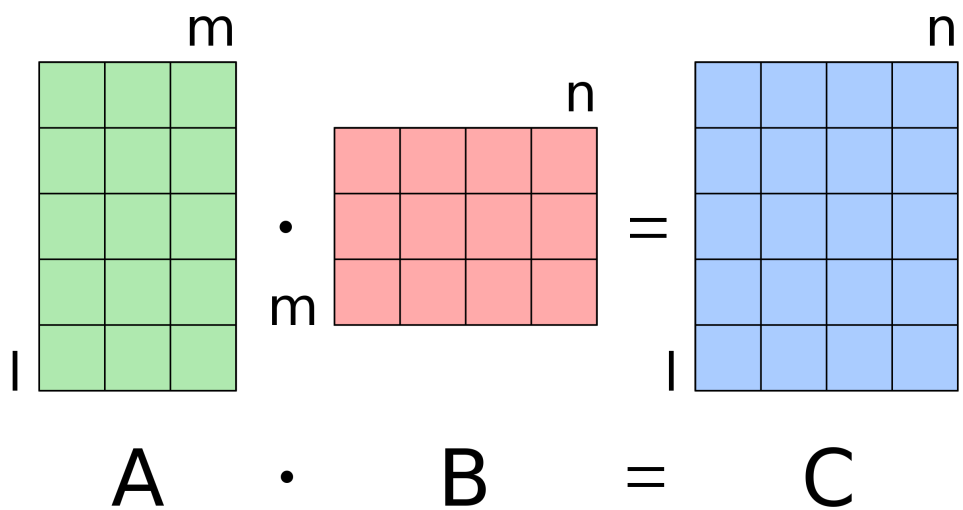
- `transpose()`: swapping rows for columns
- Only for square matrices: `mtx.shape = (n, n)`
- `array[i, j] -> array[j, i]`

Operations with arrays: Basic linear algebra operations (1)

- Dot product: `np.dot(v1, v2)`

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

- Matrix multiplication: `np.dot(mtx1, mtx2)`, `mtx1 @ mtx2` (preferred)



[1] https://en.wikipedia.org/wiki/Dot_product

[2] By Quartl - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=27634996

Operations with arrays: Basic linear algebra operations (2)

- Inverse of a matrix: `np.linalg.inv(mtx_sq)`
  Inverse matrices are such that, when multiplied by their original matrix, the result is the identity matrix of that size. Only can be applied to (some) square matrices.
  - A * A' = I

The End