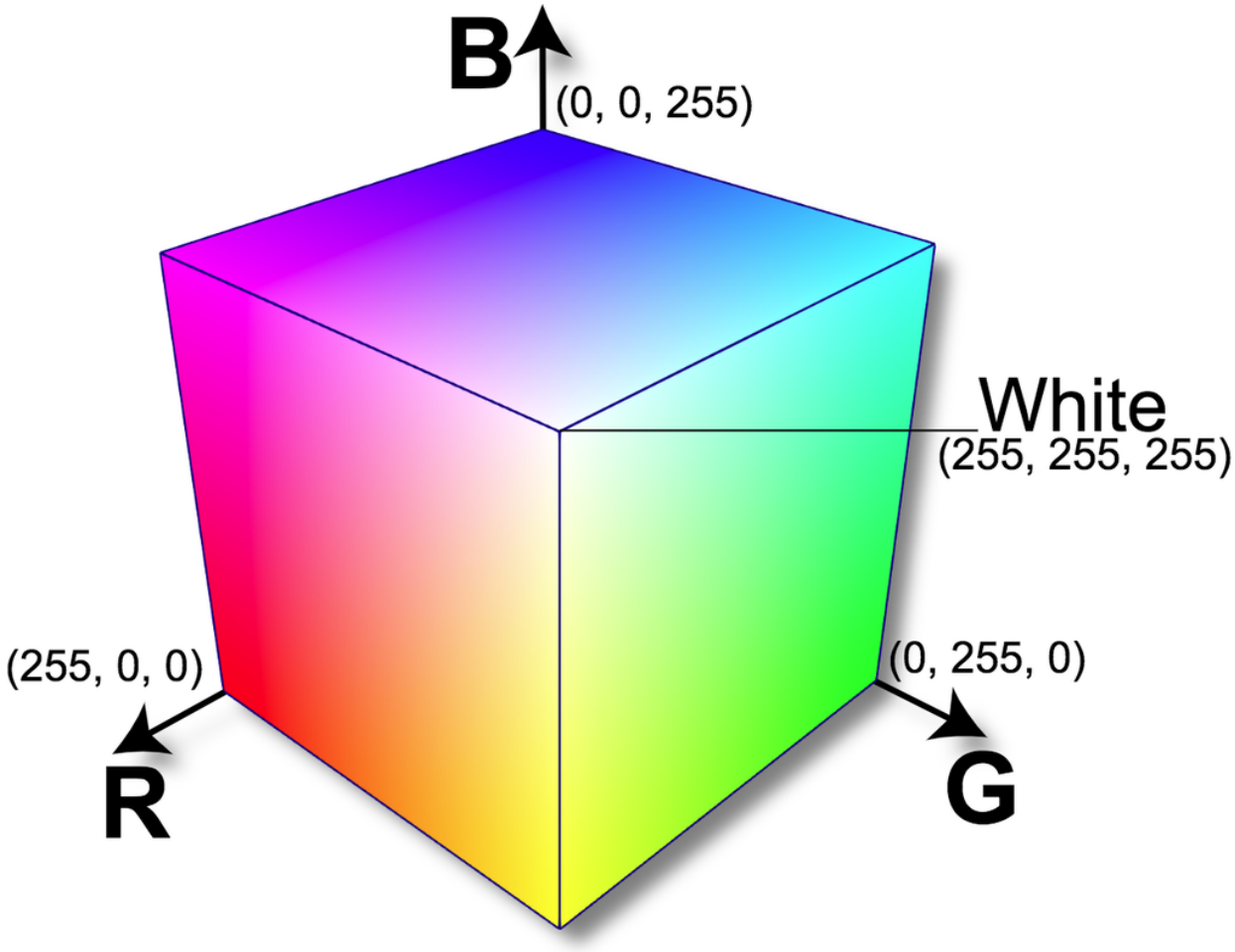
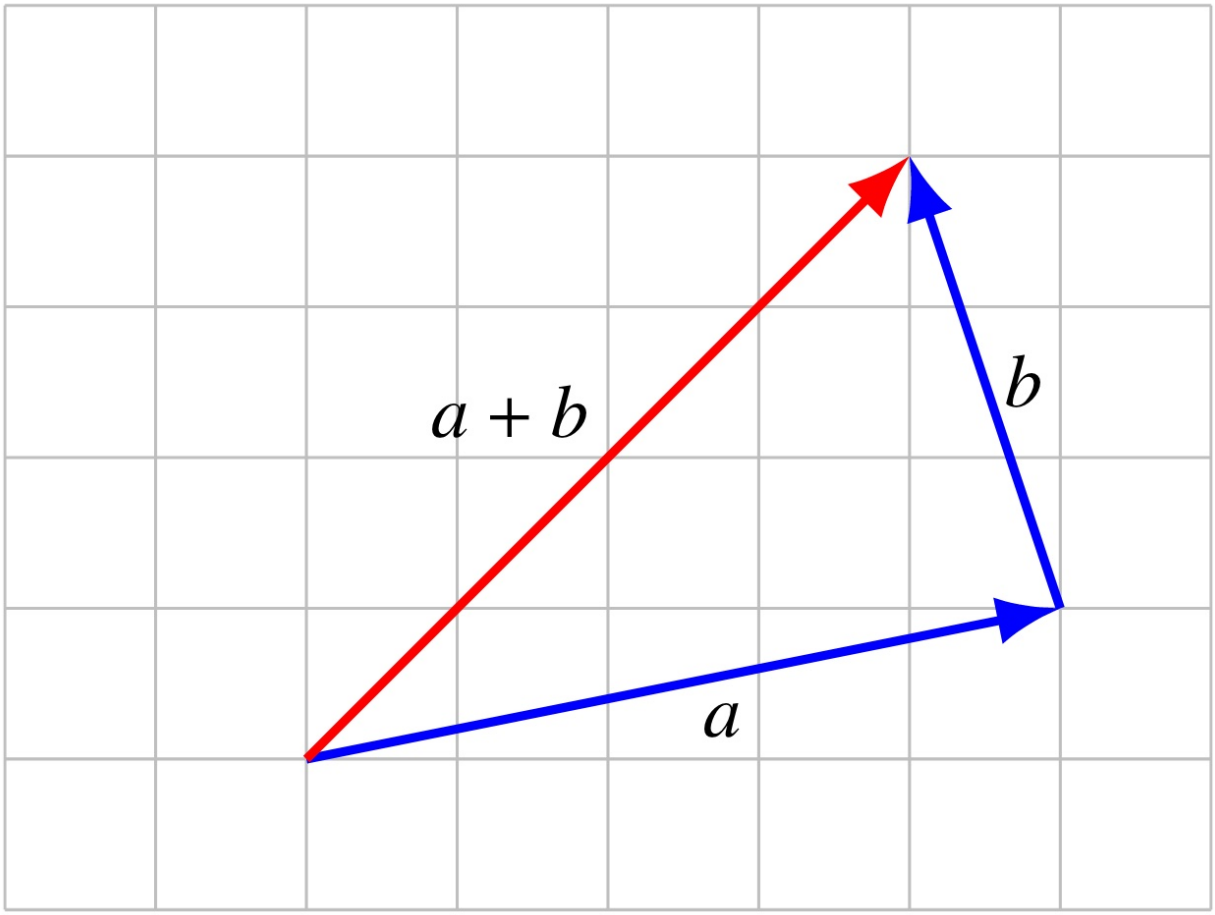
01 Vectors





Outline: 01 Vectors

- First definitions and notations
- Examples
- Addition, subtraction and scalar multiplication
- Inner product
- Complexity

Outline: 01 Vectors

- **First definitions and notations**
- Examples
- Addition, subtraction and scalar multiplication
- Inner product

- Complexity

Vector: Definition

Definition A *vector* is an ordered list of numbers, written as:

$$\begin{pmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{pmatrix} \text{ or } \begin{bmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{bmatrix} \text{ or } (-1.1, 0.0, 3.6, -7.2).$$

- The numbers in the list are called: *components, elements, entries, or coefficients* of the vector.
- The number n of elements in the list is called: *size, dimension, or length* of the vector.
- If a vector has n elements, it is called a n-vector.

Exercise: What are the components of the vector above? What is its dimension?

Definition: In contrast to vectors, numbers are just called *scalars*. For example, 3.4 is a scalar.

Vector: Notations

Notation:

- We use symbols to denote vectors, e.g., $a, X, p, \beta, E^{\text{aut}}$.
- Other possible notations: bold $\mathbf{g}$, $\overrightarrow{a}$.
- The ith element of n-vector a is denoted a_i.
- In a_i, the i is the index.

Remarks: What is really "a_i"?

- in Math: for an n-vector, indexes run from $i = 1$ to $i = n$,
- 😱 in Python: for an n-vector, indexes run from $i = 0$ to $i = n - 1$,
- 😱 Sometimes, a_i refers to the ith vector in a list of vectors.

Definition: Two vectors a and b of the same size n are equal if:

$$a_i = b_i \text{ for all } i \text{ in } \{1, \ldots, n\}.$$

Vector: Python

In Python, vectors can be represented as:

- a list of numbers, using `[]`,
- a tuple of numbers, using `()`,

- an array of numbers with the package NumPy. and their size/length/dimension is computed with `len` .

In [1]:
```
a = [-1.1, 0.0, 3.6, -7.2]; print(a)
b = (-1.1, 0.0, 3.5, -7.2); print(b)
```

```
[-1.1, 0.0, 3.6, -7.2]
(-1.1, 0.0, 3.5, -7.2)
```

In [2]:
```
import numpy as np
c = np.array([-1.1, 0.0, 3.6, -7.2]); c
```

Out[2]:
```
array([-1.1,  0. ,  3.6, -7.2])
```

In [3]:
```
len(a), len(b), len(c)
```

Out[3]:
```
(4, 4, 4)
```

Vector components: Python

In Python, we can access the components of a vector with the following syntax.

In [4]:
```
a[0], b[0], c[0]
```

Out[4]:
```
(-1.1, -1.1, -1.1)
```

😱- Be careful that the first element is indexed 0, and the last is indexed n - 1.

In [5]:
```
a[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/var/folders/dz/k1hb2xr94k558sjs416njdp40000gn/T/ipykernel_70763/3944406842.py i
n <module>
----> 1 a[4]

IndexError: list index out of range
```

Vector components: Python

In Python, the components of a vector can be assigned (except for the tuple 😱).

In [6]:
```
a[2] = 4.0
a
```

Out[6]:
```
[-1.1, 0.0, 4.0, -7.2]
```

In [7]:
```
b[2] = 4.0
```

```
b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/dz/k1hb2xr94k558sjs416njdp40000gn/T/ipykernel_70763/1212393371.py i
n <module>
----> 1 b[2] = 4.0
      2 b

TypeError: 'tuple' object does not support item assignment
```

In [8]:
```
c[2] = 4.0
c
```

Out[8]:
```
array([-1.1,  0. ,  4. , -7.2])
```

😱 Special Warning with NumPy Arrays

If we:

- assign a vector a to another new vector y,
- change the components of a, Then:
- the components of y will be changed.

The assignment of *arrays* does not copy the original array to a new one, it creates a *reference* to the same values.

In [9]:
```
y = a; print(a, y)
a[3] = 2021; print(a, y)
```

```
[-1.1, 0.0, 4.0, -7.2] [-1.1, 0.0, 4.0, -7.2]
[-1.1, 0.0, 4.0, 2021] [-1.1, 0.0, 4.0, 2021]
```

To avoid this problem, use the `.copy` syntax:

In [28]:
```
y = a.copy(); print(a, y)
a[3] = 2022; print(a, y)
```

```
[-1.1, 0.0, 4.0, 2021] [-1.1, 0.0, 4.0, 2021]
[-1.1, 0.0, 4.0, 2022] [-1.1, 0.0, 4.0, 2021]
```

Block or Stacked Vectors

Definition: Suppose a, b, c are vectors with sizes m, n, p. We can create a new vector d as:

$$d = \begin{bmatrix} a \\ b \\ c \end{bmatrix}.$$

The vector d is called a *block vector or a stacked vector with entries* a, b, c, or simply the concatenation of a, b, c. d has size $m + n + p$ with the following components:

$$d = (a_1, \ldots, a_m, b_1, \ldots, b_n, d_1, \ldots, d_p).$$

In [36]:
```
a = np.array([1, -1]); b = np.array([2, -2, -2.2]); c = np.array([3, 3.3])
d = np.concatenate([a, b, c]); d
```

Out[36]:
```
array([ 1. , -1. ,  2. , -2. , -2.2,  3. ,  3.3])
```

Zero, Ones and One-Hot Vectors

Definition: The n-vector with all entries 0 is denoted 0_n or just 0 and is called a zero vector. The n-vector with all entries 1 is denoted 1_n or just 1 and is called a ones-vector.

Definition: A one-hot vector is a vector which has one entry 1 and all others 0. If i is the index of the non-zero entry, we denote it e_i.

Exercise: What are all the one-hot vectors of length 3?

In Python:

In [87]:
```
zeros_vec = np.zeros(3); display(zeros_vec)
ones_vec = np.ones(4); display(ones_vec)
```

```
array([0., 0., 0.])
array([1., 1., 1., 1.])
```

In [40]:
```
i = 1; n = 3
ei = np.zeros(n); ei[i] = 1
ei
```

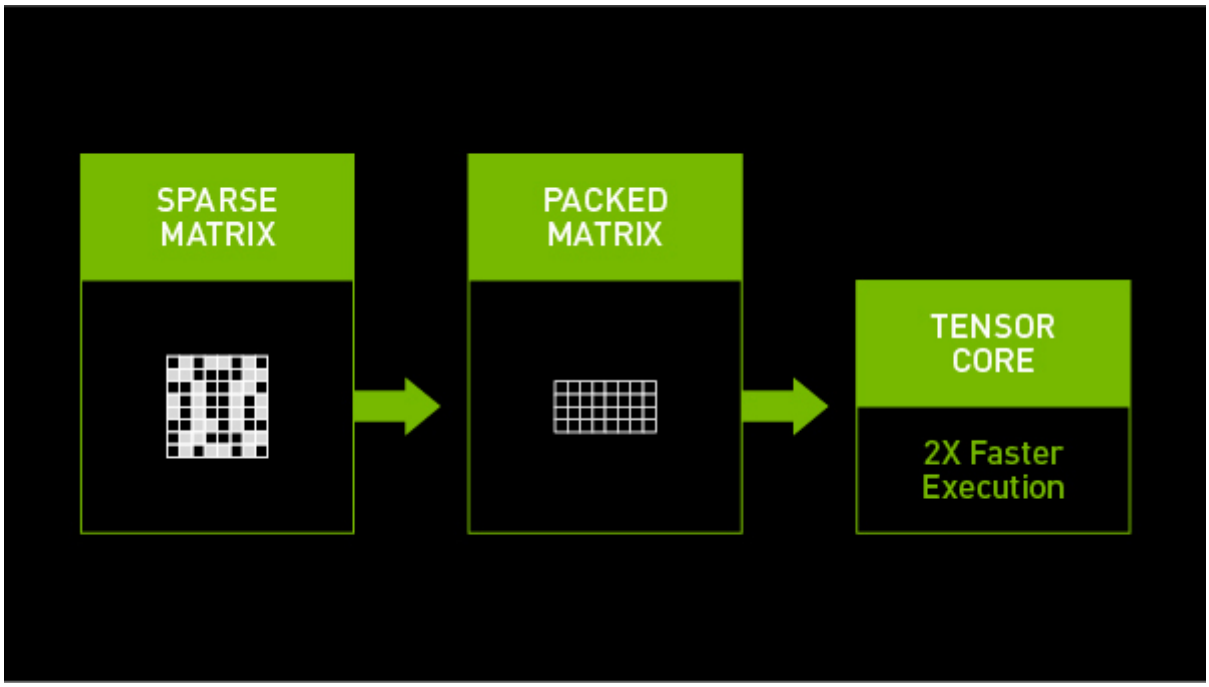
Out[40]:
```
array([0., 1., 0.])
```

Sparsity

Definition: A vector is sparse if "many" of its entries are 0.

- Can be stored and manipulated efficiently on a computer.

Exercise: Give examples of sparse and non-sparse vectors.

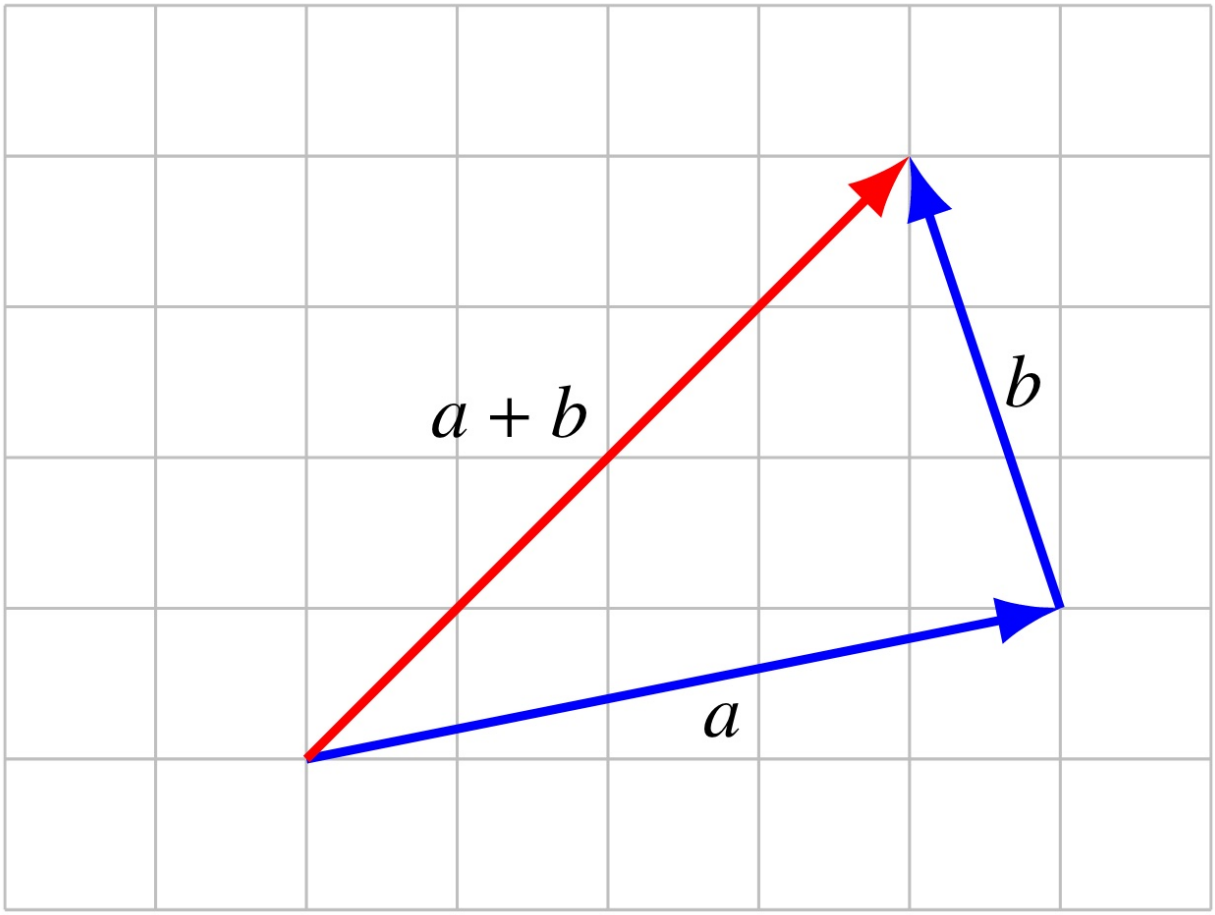


The A100 packs sparse matrices to accelerate AI inference tasks
Source: https://blogs.nvidia.com/blog/2020/05/14/sparsity-ai-inference/

Outline: 01 Vectors

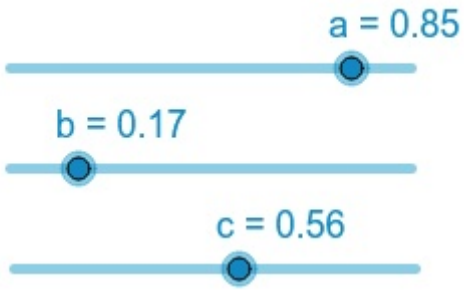
Example: Location or displacement in 2D or 3D

The 2-vector (x1, x2) can represent a location or a displacement in 2-D



Examples in ECE and beyond

- vector of color: (R, G, B)
- grades of n different homework problems
- audio: xi is the acoustic pressure at sample time i (sample times are spaced 1/44100 seconds apart)
- features: xi is the value of ith feature or attribute of an entity
- customer purchase: xi is the total $ purchase of product i by a customer over some period
- word count: xi is the number of times word i appears in a document (see next slide)



red: $\vec{u} = (255, 0, 0)$
green: $\vec{v} = (0, 255, 0)$
blue: $\vec{w} = (0, 0, 255)$

Example: Word count vectors

In a short document:

Word count vectors are used in computer based document analysis. Each entry of the word count vector is the number of times the associated dictionary word appears in the document.

Exercise: Give the word count vector associated to the text in italic above using the following dictionnary:

$$D = \{\text{word}, \text{in}, \text{number}, \text{horse}, \text{the}, \text{document}\}.$$

Outline: 01 Vectors

- First definitions and notations
- Examples
- **Addition, subtraction and scalar multiplication**
- Inner product
- Complexity

Vector Addition and Subtraction

Definition: Two (or more) n-vectors a and b can be added, with sum denoted a + b.

- The sum is computed by adding corresponding entries.

- Similary, a and b can be subtracted, with subtraction denoted a - b, and subtracting entries.

Exercise: Compute:

$$\begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

In Python:

In [88]:
```
a = np.array([0, 7, 3]) + np.array([1, 2, 0]); display(a)
a = np.array([1, 9]) - np.array([1, 1]); display(a)
```

```
array([1, 9, 3])
array([0, 8])
```

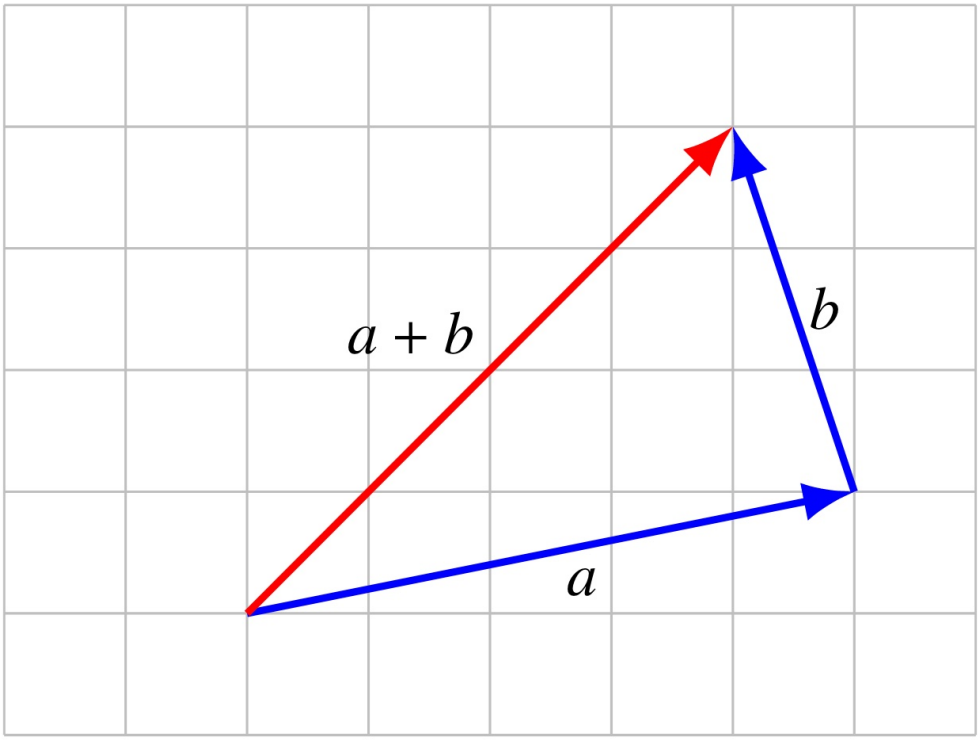
Properties of Vector Addition

Properties:

- commutative: $a + b = b + a$
- associative: $(a + b) + c = a + (b + c) = a + b + c$
- $a + 0 = a$
- $a - a = 0$

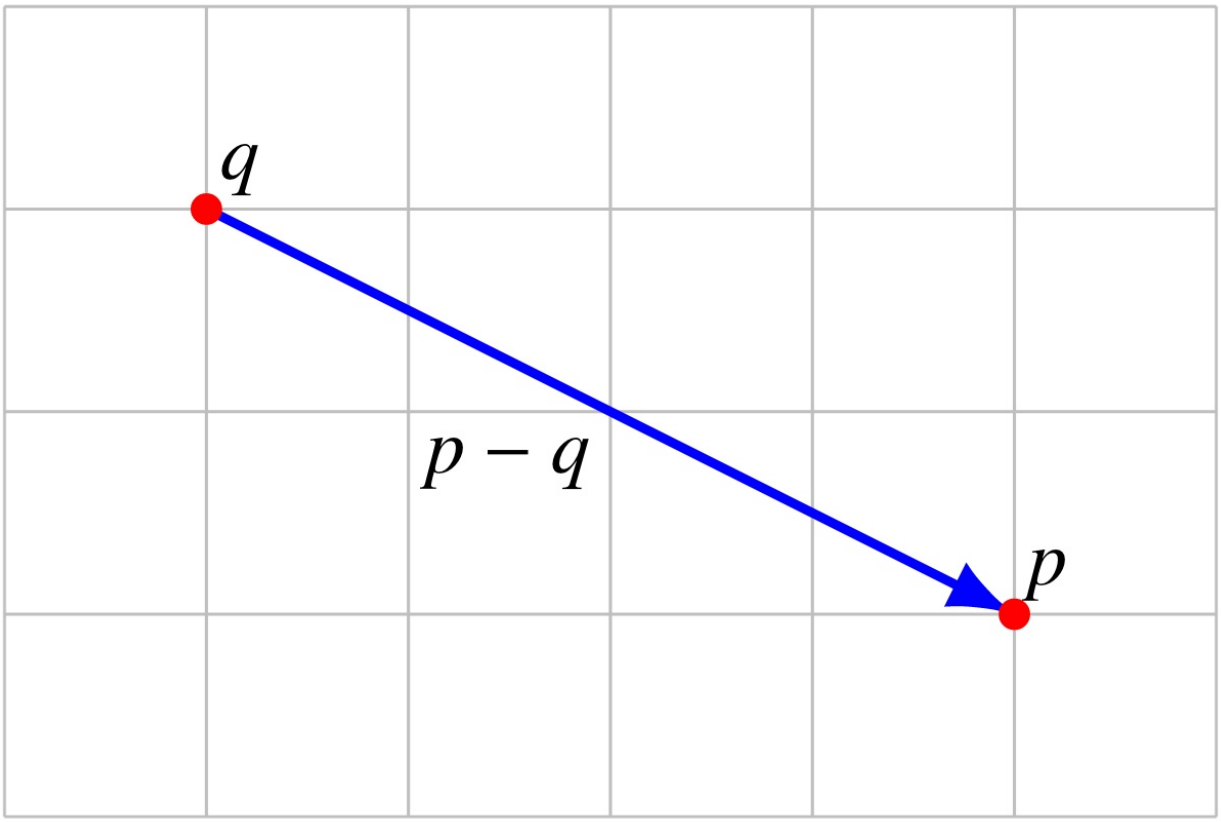
Adding displacement vectors

If vectors a and b are displacements, a + b is the sum displacement



Displacement from one point to another

Displacement from point q to point p is p – q



Scalar-vector multiplication

Definition: A scalar β and a n-vector a can be multiplied: $\beta a = (\beta a_1, \ldots, \beta a_n)$.

Exercise: Compute:

$$(-2)\begin{bmatrix}1\\9\\6\end{bmatrix}.$$

In Python:

In [67]:
```
x = np.array([0, 2, -1])
2.2 * x
```

Out[67]:
```
array([ 0. ,  4.4, -2.2])
```

Properties of scalar-vector multiplication

Properties:

- associative: $(\beta\gamma)a = \beta(\gamma a)$
- left distributive: $(\beta + \gamma)a = \beta a + \gamma a$
- right distributive: $\beta(a + b) = \beta a + \beta b$

In Python:

In [68]:
```
a = np.array([1, 2]); b = np.array([3, 4]); beta = 0.5

lhs = beta * (a + b)
rhs = beta * a + beta * b
lhs, rhs
```

Out[68]:
```
(array([2., 3.]), array([2., 3.]))
```

Linear combinations

Definition: For n-vectors $a_1, \ldots, a_m$ and scalars $\beta_1, \ldots, \beta_m$:

$$\beta_1 a_1 + \cdots + \beta_m a_m$$

is a linear combination of the vectors.

- $\beta_1, \ldots, \beta_m$ are the coefficients.

Exercise: Write a n-vector $b = (b_1, \ldots, b_n)$ as a linear combination of the one-hot n-vectors $e_1, \ldots, e_n$.

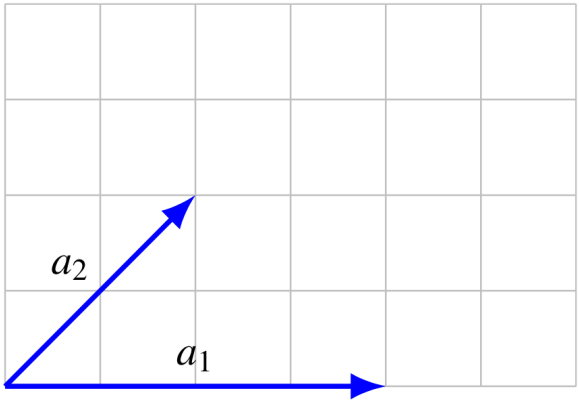
In Python:

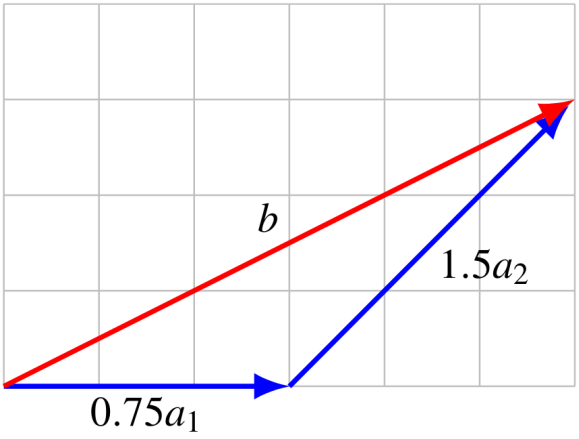
In [69]:
```
a, b = np.array([1, 2]), np.array([3, 4])
alpha, beta = -.5, 1.5
alpha * a + beta * b
```

Out[69]:
```
array([4., 5.])
```

Displacements and Linear Combination

Two vectors a_1 and a_2, and linear combination $b = 0.75a_1 + 1.5a_2$





Outline: 01 Vectors

Inner Product

Definition: The inner product (or dot product) of n-vectors a and b is

$$a^T b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n.$$

- Other notations: $\langle a, b \rangle$, $\langle a | b \rangle$, (a, b), $a \cdot b$.

Exercise: Compute the inner product of $a = (1, 0, 2)$ and $b = (-1, 1, 2)$.

In Python:

In [70]:
```
x = np.array([-1, 2, 2])
y = np.array([1, 0, -3])
np.inner(x,y)
```

Out[70]:
```
-7
```

Properties of inner product

Properties:

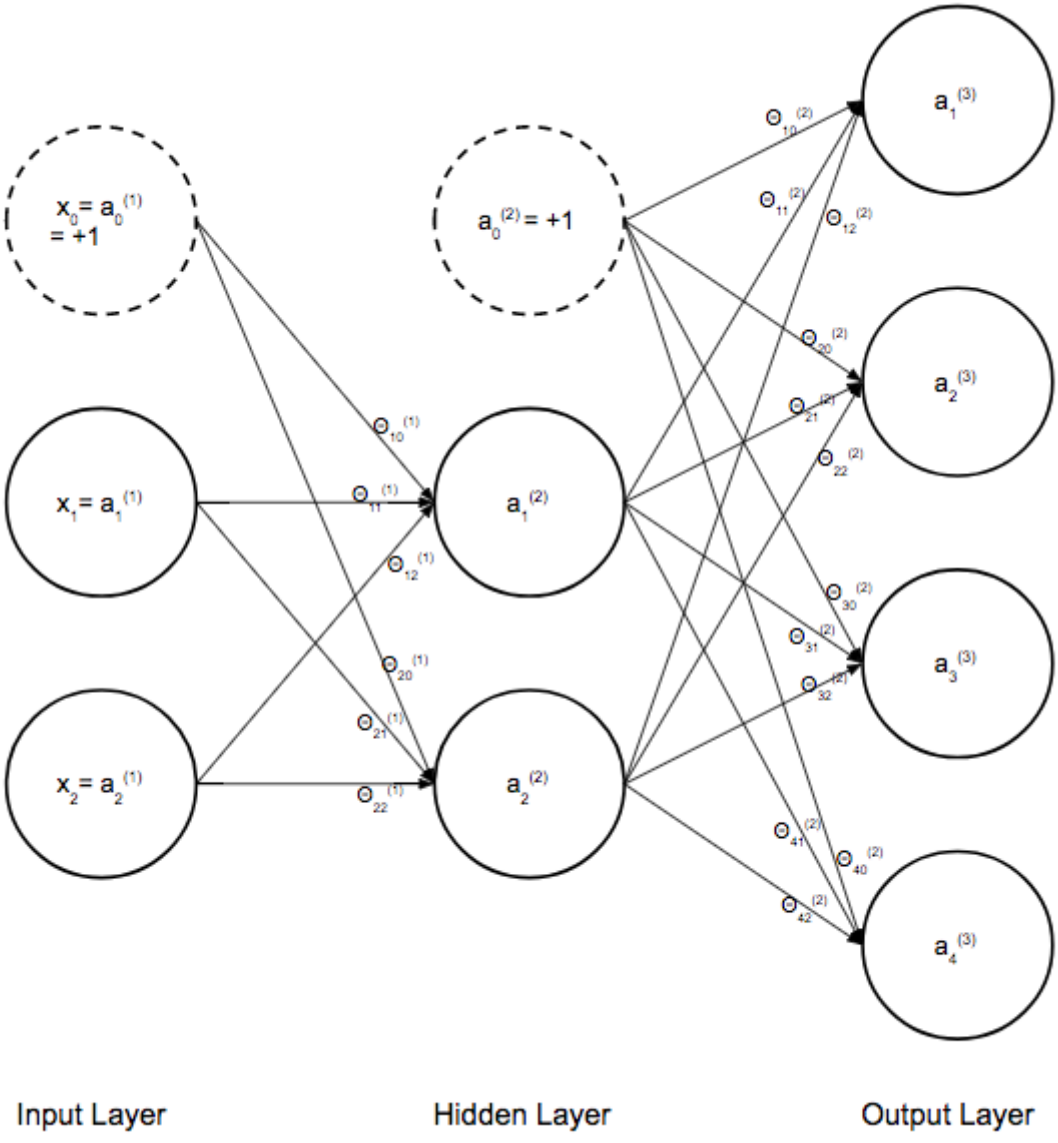
- $a^T b = b^T a$
- $(\gamma a)^T b = \gamma (a^T b)$
- $(a+b)^T c = a^T c + b^T c$

Exercise:

- Show that: $(a+b)^T (c+d) = a^T c + a^T d + b^T c + b^T d$
- Given $a = (a_1, \ldots, a_n)$: compute $e_i^T a$, $1^T a$, $a^T a$ using the entries of a.

Examples in ECE and beyond

- w is weight vector, f is feature vector; $w^T f$ is score
- p is vector of grades, q is vector of weights; $p^T q$ is the total grade



Inner product in neural networks.

Outline: 01 Vectors

Complexity = Flop Counts

Computers store (real) numbers in floating-point format.

In [83]:
```
a = 1.2
type(a)
```

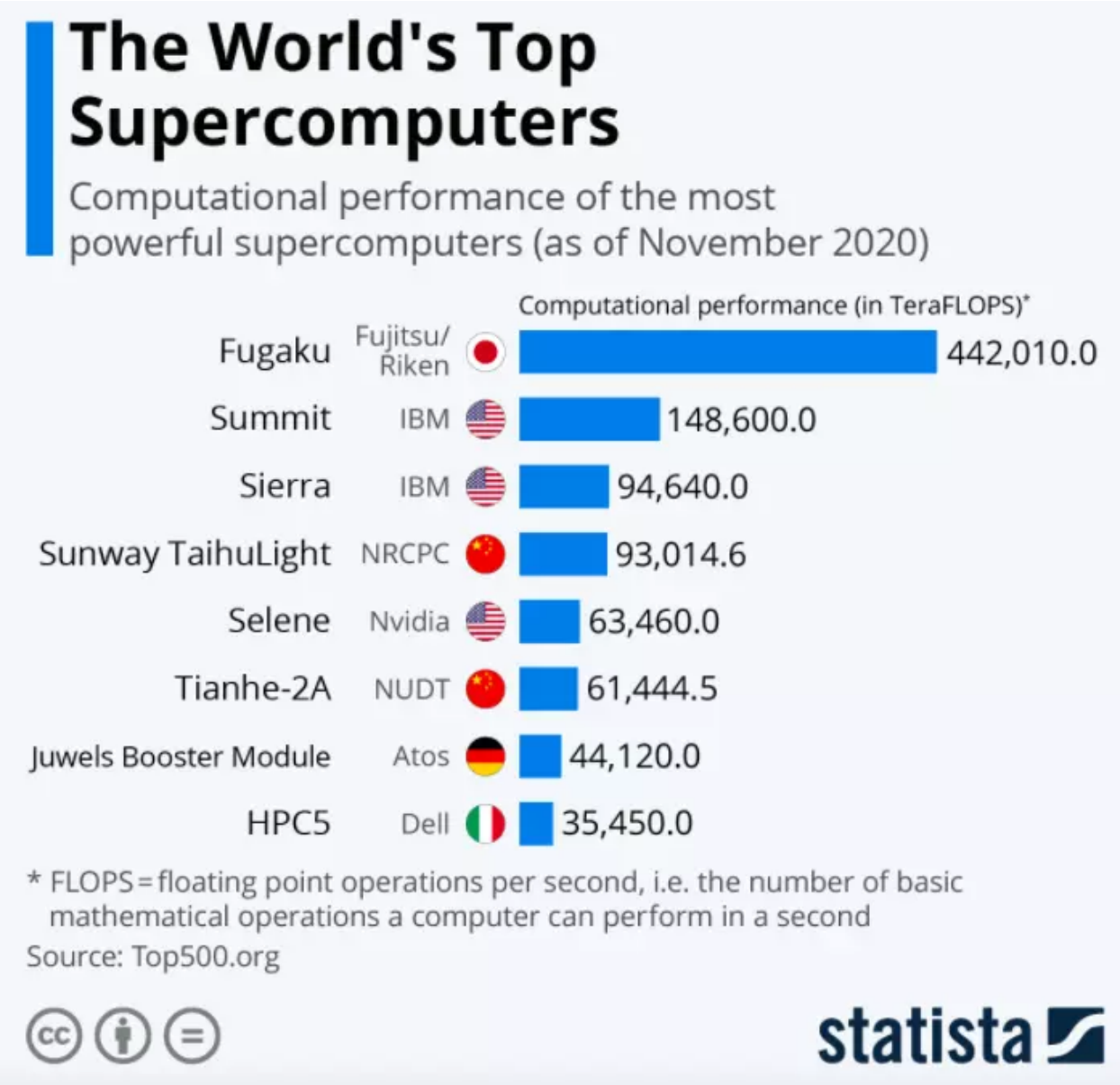
Out[83]:
```
float
```

Definition: Basic arithmetic operations (addition, multiplication, . . .) are called floating point operations or flops.

Definition: The complexity of an algorithm or operation is the total number of flops needed, as function of the input dimension(s).

- Complexity can be very grossly approximated
- Complexity allows to estimate the execution time: time to execute $\simeq$ (flops needed)/(computer speed)

Current computers are around 1Gflop/sec (109 flops/sec), but this can vary by factor of 100



Source: https://www.weforum.org/agenda/2021/01/supercomputer-world-technology-computer-japan-fugaku/

Complexity of vector addition, inner product

- $x + y$ needs n additions, so: n flops
- $x^T y$ needs n multiplications and $n - 1$ additions so: $2n - 1$ flops --> approximated to $2n$ (or even n)

🤓 Much less flops are needed when x or y is sparse.

In Python, we can time our operations (does *not* give the complexity, only the time of execution):

In [89]:
```
import timeit

a, b = np.random.randn(10 ** 4), np.random.randn(10 ** 4)
```

In [90]:
```
%timeit a + b
%timeit np.sum(a * b)
```

```
3.87 µs ± 121 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
10.5 µs ± 106 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Outline: 01 Vectors

- First definitions and notations
- Examples
- Addition, subtraction and scalar multiplication
- Inner product
- Complexity

Resources

- Ch. 1 of VMSL

In [1]:
```
from IPython.display import Audio,Image, YouTubeVideo
id='fNk_zzaMoSs'
YouTubeVideo(id=id,width=600,height=300)
```

Out[1]: