Vector Symbolic Architectures

Iwo Błądek

Poznan University of Technology

Poznań, 2018

Syntax vs. Semantics

Syntax and semantics in reasoning

Example: Find x if you know that:

$$\frac{10}{x} = 2$$

Syntax and semantics in reasoning

Example: Find x if you know that:

$$\frac{10}{x} = 2$$

Solution by using syntax (structural transformations):

$$\frac{10}{x} = 2 \iff 10 = 2x \iff \frac{10}{2} = x \iff x = \frac{10}{2} \iff x = 5$$

semantics

Syntax and semantics in reasoning

Example: Find x if you know that:

$$\frac{10}{x} = 2$$

Solution by using syntax (structural transformations):

$$\frac{10}{x} = 2 \iff 10 = 2x \iff \frac{10}{2} = x \iff x = \frac{10}{2} \iff x = 5$$

~~semantics~~ or maybe not?!

Syntax and semantics in reasoning

Example: Find x if you know that:

$$\frac{10}{x} = 2$$

Solution by using syntax (structural transformations):

$$\frac{10}{x} = 2 \iff 10 = 2x \iff \frac{10}{2} = x \iff x = \frac{10}{2} \iff x = 5$$

~~semantics~~ or maybe not?!

Solution by using semantics (properties of objects):

Division is defined as finding such a number, which when added n times (where n is the divisor) gives the original number.
How many 2's we need to add to obtain 10?
Answer: 5. So $x = 5$.

Formalization

One of the greatest achievements of mathematics and logic, often so much taken for granted that not properly realized, is moving the reasoning about the mathematical objects from the *properties and relations* between them to the *syntactical manipulations on their structure*.

If for a given domain of knowledge such a syntactic encoding of knowledge is possible, we call it a **formalization** of this domain of knowledge.

Formalization

Syntactic manipulations can be performed by a person not knowing anything about what they concern. They make the process of computations, mental or not, much easier and faster ("automatic").

Formalization

Formalization allows machines to perform domain-reasoning on the level similar to that of humans (but usually much faster):

- automatic theorem provers (e.g. Z3).
- proof-assistants (e.g. Coq).
- analysis of computer programs in programming environments (e.g. PyCharm, Eclipse).

What would be necessary for a computer to be able to reason semantically?

Is there really a fundamental difference between those two types of reasoning?

Formalization

What does it really mean to understand something?

What do we want to achieve

The goal

We want to somehow connect syntactic transformation with distributed representations.

Example: We want to represent:

$$\text{chases(dog, boy)}$$

as a vector of numbers (e.g. activations of neurons). Additionally, we later want to be able to automatically answer, who was chasing whom:

$$\text{chases(dog, boy)} \circledast \text{“who is chasing”} = \text{dog}$$
$$\text{chases(dog, boy)} \circledast \text{“who is being chasen”} = \text{boy}$$

The goal

$$dog = \begin{pmatrix} 1.9 \\ 0.9 \\ 1.1 \\ 0.2 \end{pmatrix}, \quad boy = \begin{pmatrix} 0.2 \\ 0.6 \\ 0.6 \\ 0.1 \end{pmatrix}, \quad chase = \begin{pmatrix} 1.3 \\ 0.9 \\ 1.4 \\ 0.5 \end{pmatrix}$$

$$chases(dog, boy) = \begin{pmatrix} 0.5 \\ 1.2 \\ 0.6 \\ 0.4 \end{pmatrix}$$

$$\begin{pmatrix} 0.5 \\ 1.2 \\ 0.6 \\ 0.4 \end{pmatrix} \circledast \text{“who is being chasen”} \approx \begin{pmatrix} 0.2 \\ 0.6 \\ 0.6 \\ 0.1 \end{pmatrix}$$

The goal

- Is such a system possible?
- How $\circledast$ should be implemented?
- How "who is being chasen" (the question) should be implemented?
- Is a similar system used in the brain? (open question)

Basics of linear algebra

Linear algebra

Linear algebra

Branch of mathematics concerned with linear dependencies, that is dependencies in which a value depends on a "weighted" (by a scalar, i.e. constant number) sum of other values.

An example of a linear function:

$$f(x, y) = 2 \cdot x - 5 \cdot y$$

Matrix (pol. *Macierz*)

- Data structure for storing numbers, equipped with a certain set of operations.

Addition:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x & v \\ y & z \end{bmatrix} = \begin{bmatrix} a+x & b+v \\ c+y & d+z \end{bmatrix}.$$

Multiplication:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x & v \\ y & z \end{bmatrix} = \begin{bmatrix} ax+by & av+bz \\ cx+dy & cv+dz \end{bmatrix}.$$

Linear algebra

Special matrices:

- zero matrix (pol. *macierz zerowa*)
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- unit matrix (pol. *macierz jednostkowa*)
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Linear algebra

For arbitrary $x_{i,j}$ holds:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}$$

Linear algebra

For arbitrary $x_{i,j}$ holds:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Linear algebra

For arbitrary $x_{i,j}$ holds:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_{1,1}+1 & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2}+1 & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3}+1 \end{bmatrix}$$

Linear algebra

For arbitrary $x_{i,j}$ holds:

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}$$

Vectors

- A special matrices with one of the dimensions equal to 1.
- Can be either a column or a row.

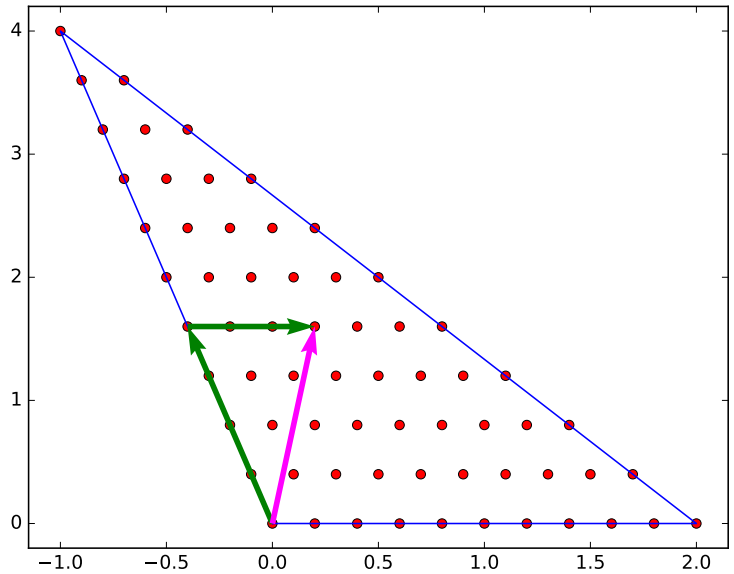
$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \quad \text{vs.} \quad \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

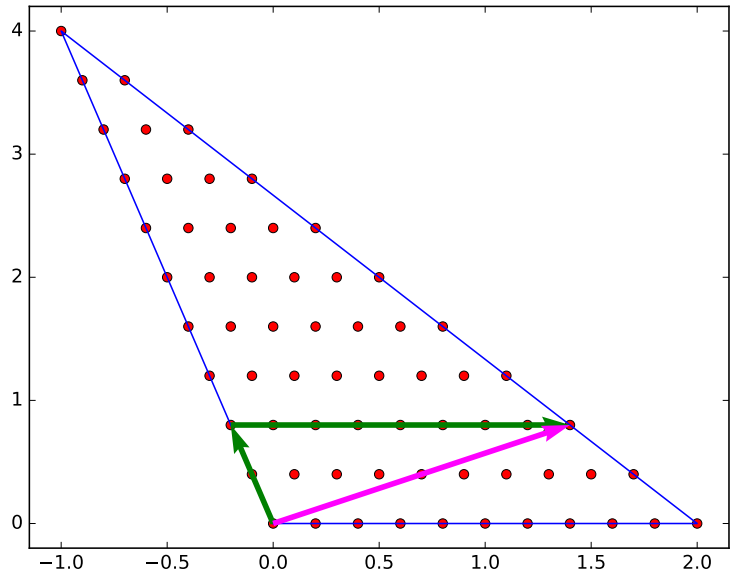
The vectors above are two different vectors!

Vectors

What are vectors good for apart from specifying relative movement in physics?

Representing points in space! For example, point $(2, 1)$ in the Euclidean space may be represented by a vector $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$.





Matrices as transformations

When we multiply matrix and vector, we get certain another vector. For example:

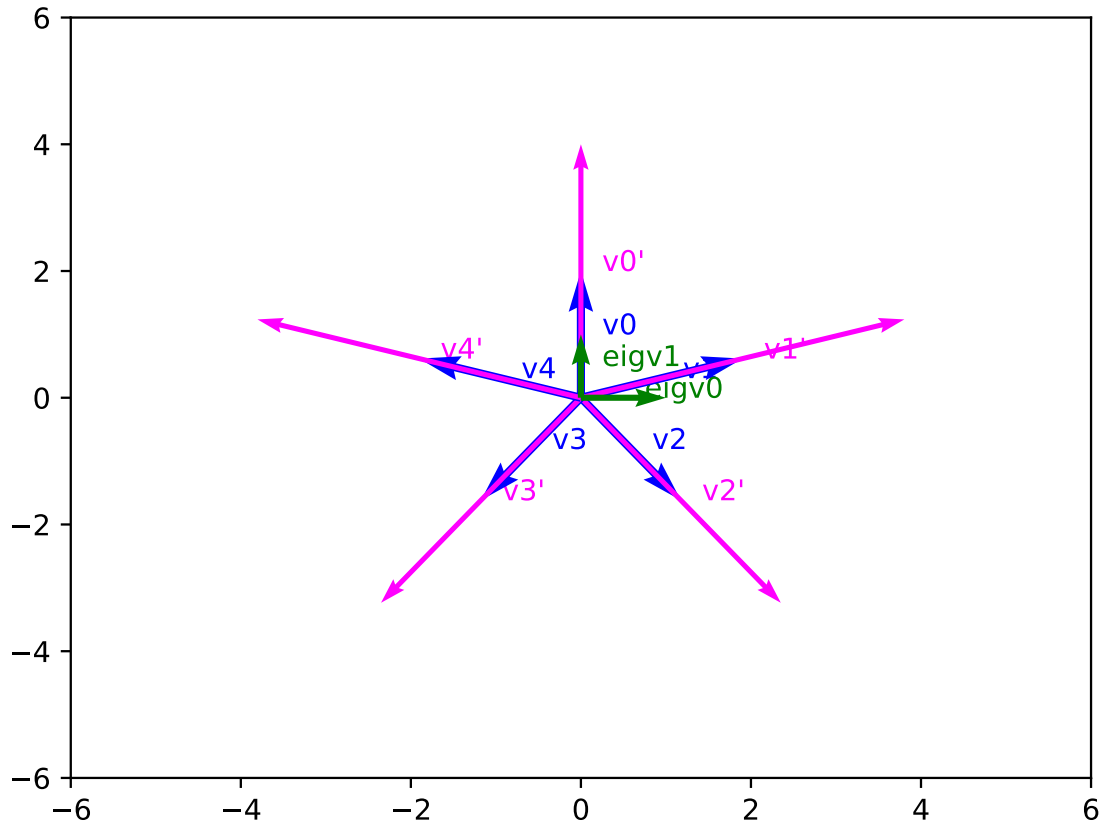
$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

In general:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = x \cdot \begin{bmatrix} a \\ c \end{bmatrix} + y \cdot \begin{bmatrix} c \\ d \end{bmatrix}$$

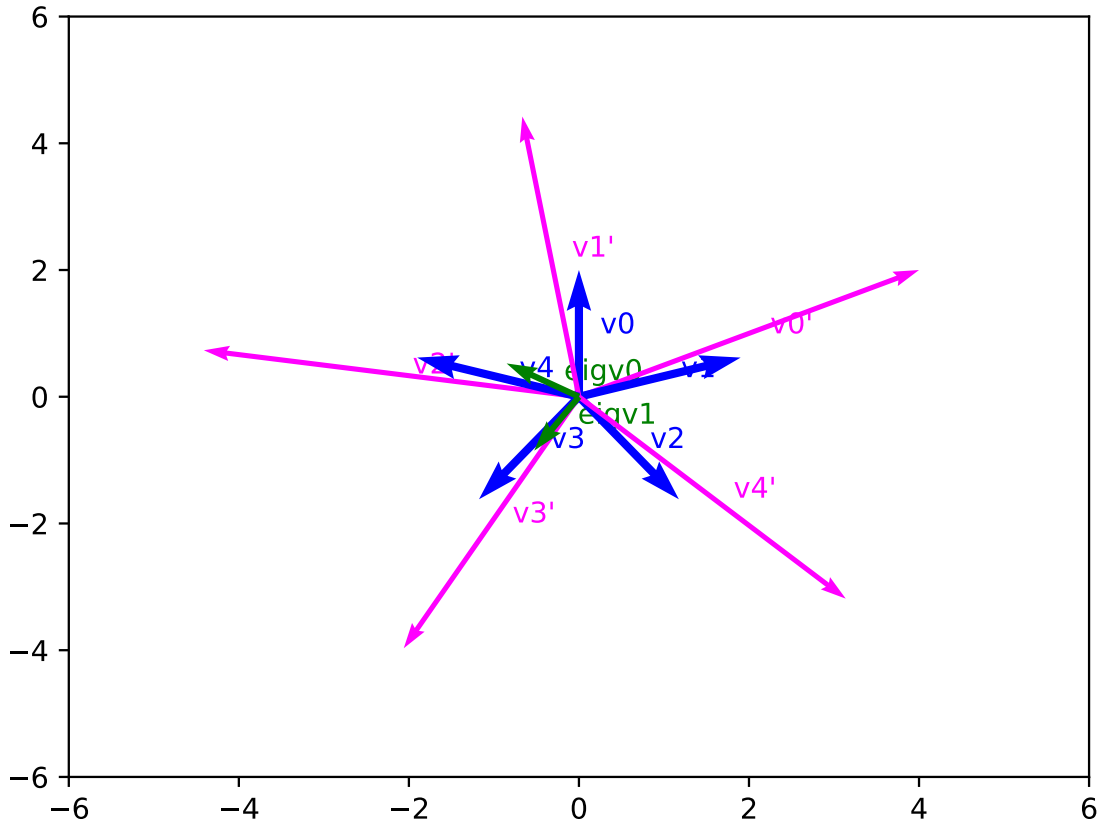
Matrices as transformations

Matrix used for transformation: $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$



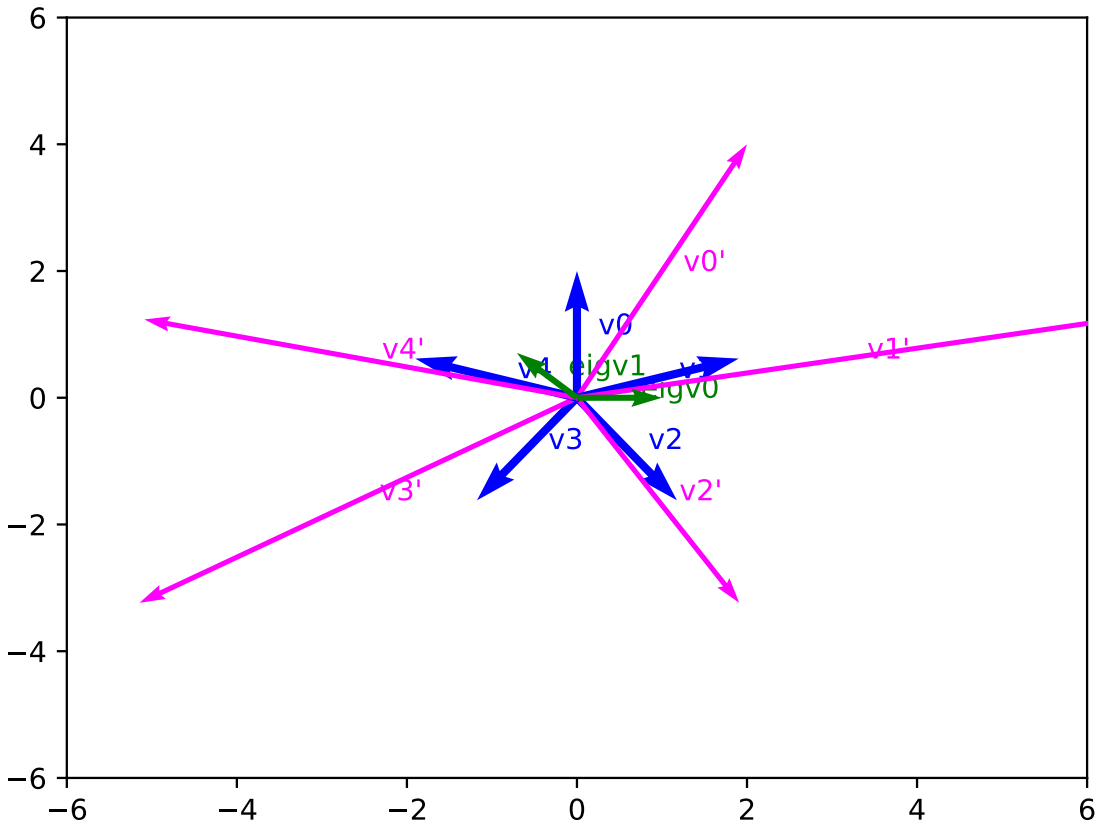
Matrices as transformations

Matrix used for transformation: $\begin{bmatrix} -1 & 2 \\ 2 & 1 \end{bmatrix}$



Matrices as transformations

Matrix used for transformation: $\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$



Convolution

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 2 & -1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{Convolution}$$

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} 20 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 40 & -20 & 20 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{Convolution}$$

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} a & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 2a & -a & a & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{Convolution}$$

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 2 & -1 & 1 & 2 & -1 & 1 & 0 \end{bmatrix} \quad \text{Convolution}$$

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 2 & -1 & 1 & 0 & 2 & -1 & 1 \end{bmatrix} \quad \text{Convolution}$$

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 2 & -1 & 1 & 0 & 0 & 0 & 2 \end{bmatrix} ? \quad ? \quad \text{Convolution}$$

What if we don't have enough space for new elements to add?

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 2 & -1 & 1 & 0 & 0 & 0 & 2 & -1 & 1 \end{bmatrix} \quad \text{Convolution}$$

What if we don't have enough space for new elements to add?
We add them anyway!

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 2 & ? & ? & ? & ? & ? & ? \end{bmatrix} \quad \text{Convolution}$$

Convolution

- **Convolution** (pol. *splot*) is used for example in the signal processing and image processing.
- Convolution is an operation between two signals (mathematically speaking: two functions with time as an argument).
- The second of them is often called **mask** or **kernel**.

Example:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{Original signal}$$

$$M = \begin{bmatrix} 2 & -1 & 1 \end{bmatrix} \quad \text{Mask/Kernel}$$

$$A * B = \begin{bmatrix} 2 & (-1+2) & (1+(-1)) & 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{Convolution}$$

$$A * B = \begin{bmatrix} 2 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Convolution

The other way of representing what happens:

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad M = \begin{bmatrix} 3 & 1 & 2 \end{bmatrix}$$

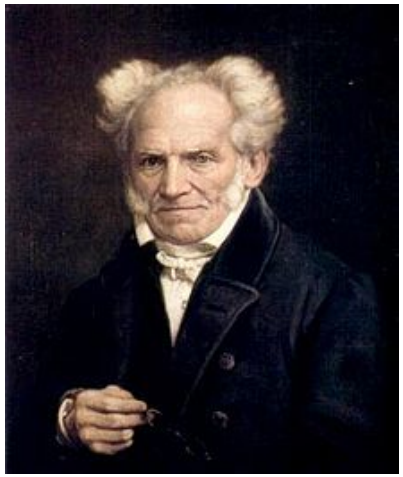
		1	2	1	0	0	0	0 (Original signal)
		$\underline{3}$	1	2				
			$\underline{6}$	2	4			
				$\underline{3}$	1	2		
					$\underline{0}$	0	0	
						.....		
3	7	7	5	2	0	0	(Convolution)	

Convolution

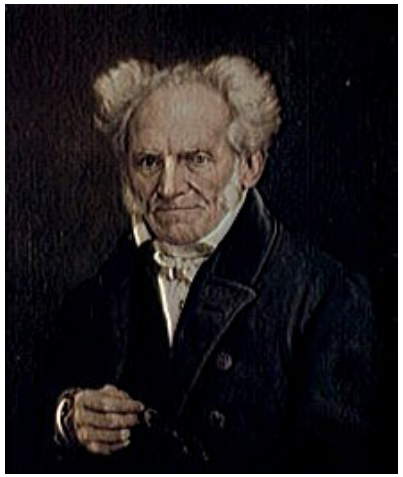
- **Interpretation 1:** a response of the system over time for signals incoming in the subsequent time moments. (perspective used in this presentation)
- **Interpretation 2:** a weighted average of the neighborhood. (in this case mask is centered on the element)
- Mask/kernel describes system's response for a single spike (value of 1 in A). If a value is different, then the system's response is appropriately scaled (multiplication).
- Convolution is **commutative**, so $A * M = M * A$.

Convolution in 2D

- Nobody said that we can only convolve in one dimension!
- Example:



$$* \begin{bmatrix} -0.2 & -0.1 & 0.3 \\ -0.2 & -0.1 & 0.9 \\ -0.2 & -0.1 & 0.3 \end{bmatrix} =$$



You can apply arbitrary masks on an image in Gimp under the `Filtry→Ogólne→Zniekształcenia macierzowe`.

Circular convolution

In the **circular convolution** mask wraps on the end and starts from the beginning.

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}, \quad M = \begin{bmatrix} 3 & 4 & 5 & 6 \end{bmatrix}$$

	1	1	1	1	(Original signal)
	3	4	5	6	
	6	3	4	5	
	5	6	3	4	
	4	5	6	3	
	18	18	18	18	(Circular convolution)

Why values are the same?

Because the original signal and the mask have both the same length and the original signal is uniform.

Circular convolution

In the **circular convolution** mask wraps on the end and starts from the beginning.

$$A = \begin{bmatrix} 1 & 0 & 2 & 3 \end{bmatrix}, \quad M = \begin{bmatrix} 3 & 1 & 2 & 1 \end{bmatrix}$$

	1	0	2	3	(Original signal)
	$\underline{3}$	1	2	1	
	0	$\underline{0}$	0	0	
	4	2	$\underline{6}$	2	
	3	6	3	$\underline{9}$	
	10	9	11	12	(Circular convolution)

Circular convolution

In the **circular convolution** mask wraps on the end and starts from the beginning.

$$A = \begin{bmatrix} 2 & 1 & 1 & 1 \end{bmatrix}, \quad M = \begin{bmatrix} 3 & 4 & 5 & 6 \end{bmatrix}$$

	2	1	1	1	(Original signal)
	6	8	10	12	
	6	3	4	5	
	5	6	3	4	
	4	5	6	3	
	21	22	23	24	(Circular convolution)

Circular convolution

In the **circular convolution** mask wraps on the end and starts from the beginning.

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad M = \begin{bmatrix} 3 & 4 & 5 \end{bmatrix}$$

	1	1	1	1	1	(Original signal)
	$\underline{3}$	4	5			
		$\underline{3}$	4	5		
			$\underline{3}$	4	5	
	5			$\underline{3}$	4	
	4	5			$\underline{3}$	
	12	12	12	12	12	(Circular convolution)

Length doesn't matter – sequence of 1s will generate the same result.

Vector Symbolic Architectures

Vector Symbolic Architectures

Vector Symbolic Architectures

Utilize structural manipulations while not representing information in the form of symbols (as traditionally understood), but instead as the vectors of real numbers.

- Information is encoded in the form of vectors of real numbers.
- VSA define **composition**, **binding**, and **unbinding** operators.

Composition operator

- Used to **store** several pieces of information in a single vector.
- Realized by a simple addition of vectors.
- **Question:** How to retrieve later individual stored elements?

$$\begin{pmatrix} 1.2 \\ 1.8 \\ 1.6 \\ 0.7 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0.6 \\ 0.6 \\ 0.1 \end{pmatrix} + \begin{pmatrix} 0.4 \\ 0.2 \\ 0.3 \\ 0.2 \end{pmatrix} + \begin{pmatrix} 0.6 \\ 1.0 \\ 0.7 \\ 0.4 \end{pmatrix}$$

$$\mathbf{P} = \mathbf{car} + \mathbf{train} + \mathbf{airplane}$$

Binding operator

- Used to **combine** different pieces of information (e.g. **verb** $\circledast$ **learn**).
- Usually some information is lost in the process, but if the vectors used are remembered elsewhere, then they can be reconstructed.
- Realized by a circular convolution ($\circledast$) of vectors.

$$\begin{pmatrix} 1.69 \\ 1.69 \\ 1.73 \\ 1.77 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0.6 \\ 0.6 \\ 0.1 \end{pmatrix} \circledast \begin{pmatrix} 0.3 \\ 0.9 \\ 0.1 \\ 0.5 \end{pmatrix} + \cdots + \begin{pmatrix} 0.4 \\ 0.2 \\ 0.3 \\ 0.2 \end{pmatrix} \circledast \begin{pmatrix} 0.6 \\ 1.0 \\ 0.7 \\ 0.4 \end{pmatrix}$$

$$\mathbf{P} = \mathbf{agent} \circledast \mathbf{student} + \mathbf{verb} \circledast \mathbf{learn} + \mathbf{what} \circledast \mathbf{kogni}$$

Unbinding operator

- Used to **retrieve** information from the binded representation.
- Usually requires some postprocessing to clear up the noise.
- Also realized by a circular convolution ($\circledast$), but the second vector is *inverted* (this is described on the next slide).

$$\mathbf{P} = \mathbf{agent} \circledast \mathbf{student} + \mathbf{verb} \circledast \mathbf{learn} + \mathbf{what} \circledast \mathbf{kogni}$$

$$\begin{aligned} \mathbf{P} \circledast \mathbf{agent}^{-1} &= \mathbf{student} + \textit{noise} \approx \mathbf{student} \\ \mathbf{P} \circledast \mathbf{verb}^{-1} &= \mathbf{learn} + \textit{noise} \approx \mathbf{learn} \\ \mathbf{P} \circledast \mathbf{what}^{-1} &= \mathbf{kogni} + \textit{noise} \approx \mathbf{kogni} \end{aligned}$$

Inverting vectors

- Exact procedure is rather complicated, but it can be quite well *approximated* with a very simple algorithm.
- Complex systems does not necessarily need to compute everything exactly. In fact, it can be very beneficial to approximate (what is better: 98% in 1 s, 99% in 1 day, or 100% in 1 year?).

Algorithm:
Assume that we want to invert the vector:

$$x = [x_1, x_2, x_3, x_4, x_5]$$

The approximated inversion will have the "tail" of the list reversed, while the first element of the list remains unchanged:

$$x^{-1} = [x_1, x_5, x_4, x_3, x_2]$$

Example

Example

- 50-dimensional unit vectors (of length 1), seed=1
- Similarity of vectors is computed as a *dot product*. Interpretation rules: 1.0 – the same vector, 0.0 – orthogonal, -1.0 – in the opposite direction. Closer to 1.0 means more similar.

Vocabulary (basic "symbols"):
agent, **student**, **verb**, **learn**, **what**, **kogni**

Sentence:
$\mathbf{P} = \mathbf{agent} \circledast \mathbf{student} + \mathbf{verb} \circledast \mathbf{learn} + \mathbf{what} \circledast \mathbf{kogni}$

General form of questions:
$\mathbf{answer} = \mathbf{P} \circledast \mathbf{question}$

Example

```
SIMILARITY of 'what' with vocabulary:
-0.103877851423          agent
0.0690717296125          student
0.0595131957944          verb
0.219254946056           learn
1.0                      what
-0.0478595262065         kogni
```

Example

```
SIMILARITY of 'kogni' with vocabulary:
0.084252963404          agent
0.0105059822009         student
0.131548409036          verb
0.0685833841815         learn
-0.0478595262065        what
1.0                     kogni
```

Example

$$\textbf{answer} = \textbf{P} \circledast \textbf{agent}^{-1}$$

```
QUESTION: Who is the agent?
SIMILARITY of 'answer' with vocabulary:
0.348150364868          agent
0.701648021437          student
-0.00336734074461       verb
0.0928581135105         learn
0.0221546642779         what
-0.0495422116263        kogni
```

Example

$$\textbf{answer} = \textbf{P} \circledast \textbf{verb}^{-1}$$

```
QUESTION: What is the verb?
SIMILARITY of 'answer' with vocabulary:
-0.00334961185176          agent
-0.0705556986752           student
0.125815701745             verb
0.473234691606             learn
0.135686477099             what
0.0953053451415            kogni
```

Example

$$\textbf{answer} = \textbf{P} \circledast \textbf{what}^{-1}$$

```
QUESTION: What is the target of the action?
SIMILARITY of 'answer' with vocabulary:
0.0218402616746          agent
-0.128131779576          student
0.134468887466           verb
0.16382221557            learn
0.198250358035           what
0.530717820654           kogni
```

The end of math (for now!)

Thank you for your attention.