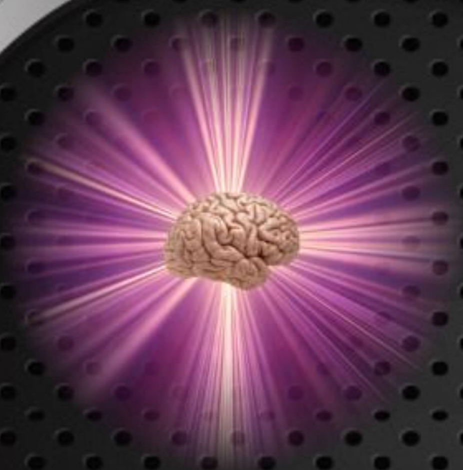
AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering
Department of Biocybernetics and Biomedical Engineering



Knowledge-based CI and DM in Biomedicine

Sequential Data, NLP, and Text Processing

Adrian Horzyk
horzyk@agh.edu.pl

Google: Adrian Horzyk

Sequential Data

What are sequential data?

Sequential Data and Domains

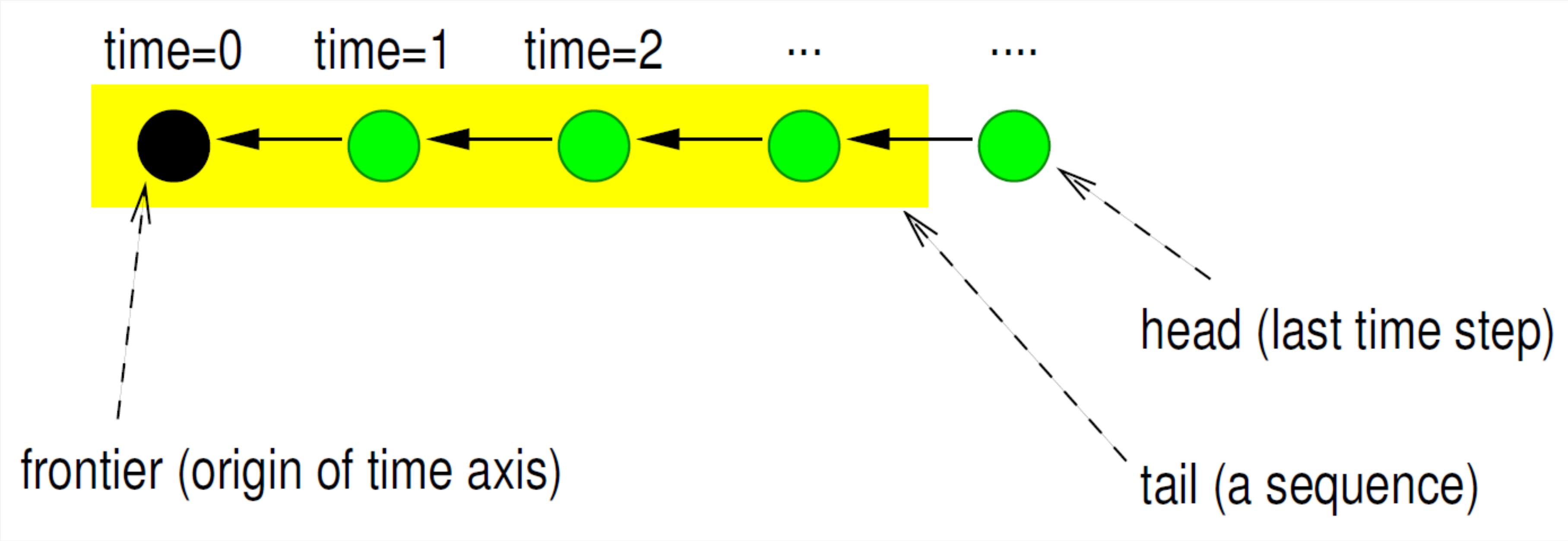
Sequential data are the second type of the most frequently used data in real-world applications because many processes, signals and input data are ordered in time or for any other reasons, so we need to use this special relationship between simple or object data that puts them in order. This order has a special meaning when classifying, predicting, translating, transforming or doing any other operation on such sequential data.

Sequential patterns differ from static patterns because:

- successive data (points) are strongly correlated,
- the succession of data is crucial from their recognition or classification point of view.

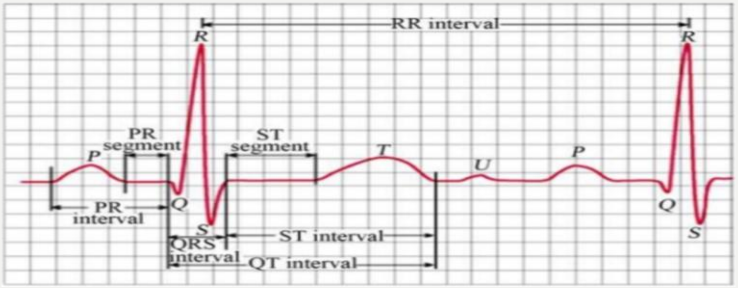
Sequential Data and Domains

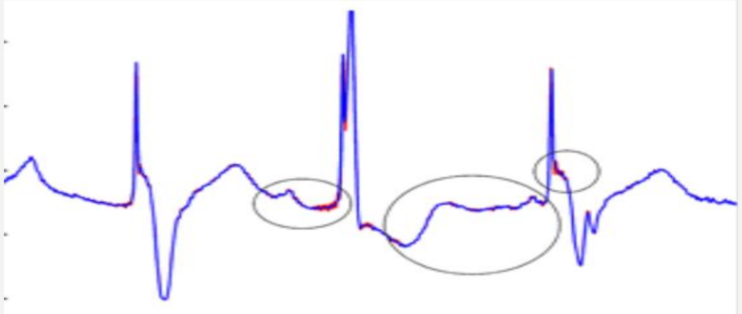
A sequence can be defined using mathematical induction as an external vertex or an ordered pair (t,h) where the head h is a vertex and the tail t is a sequence:

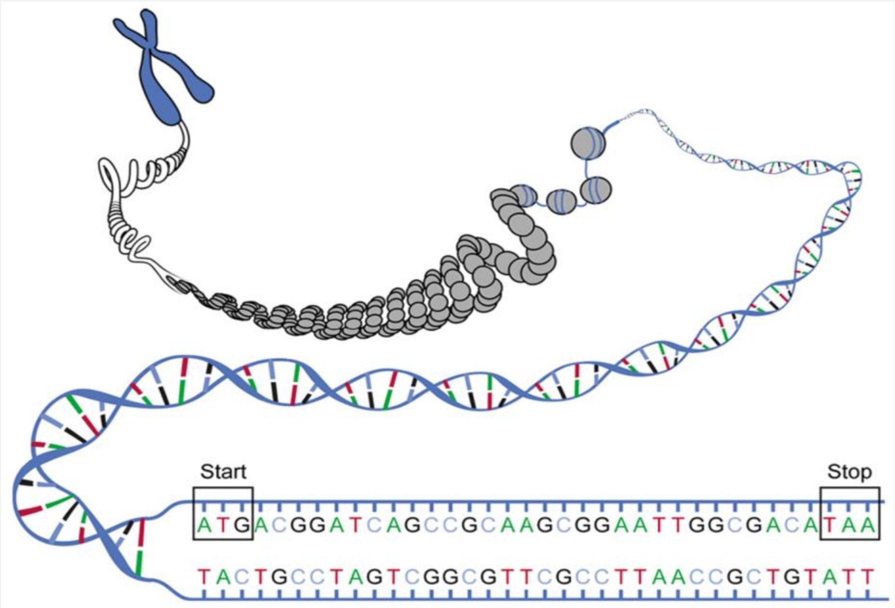


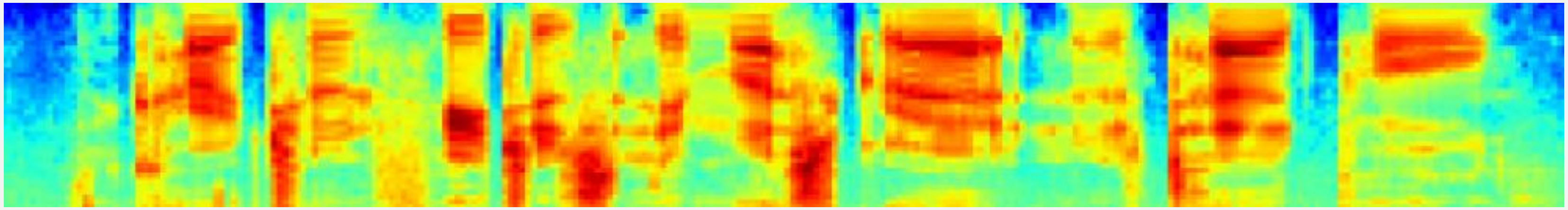
Sequential Data and Domains

Sequential data are used in speech recognition, automatic proof-reading and text correction, DNA sequence analysis, ECG and EEG signals classification, sentiment classification, machine translation, video activity recognition , music generation, and many others:







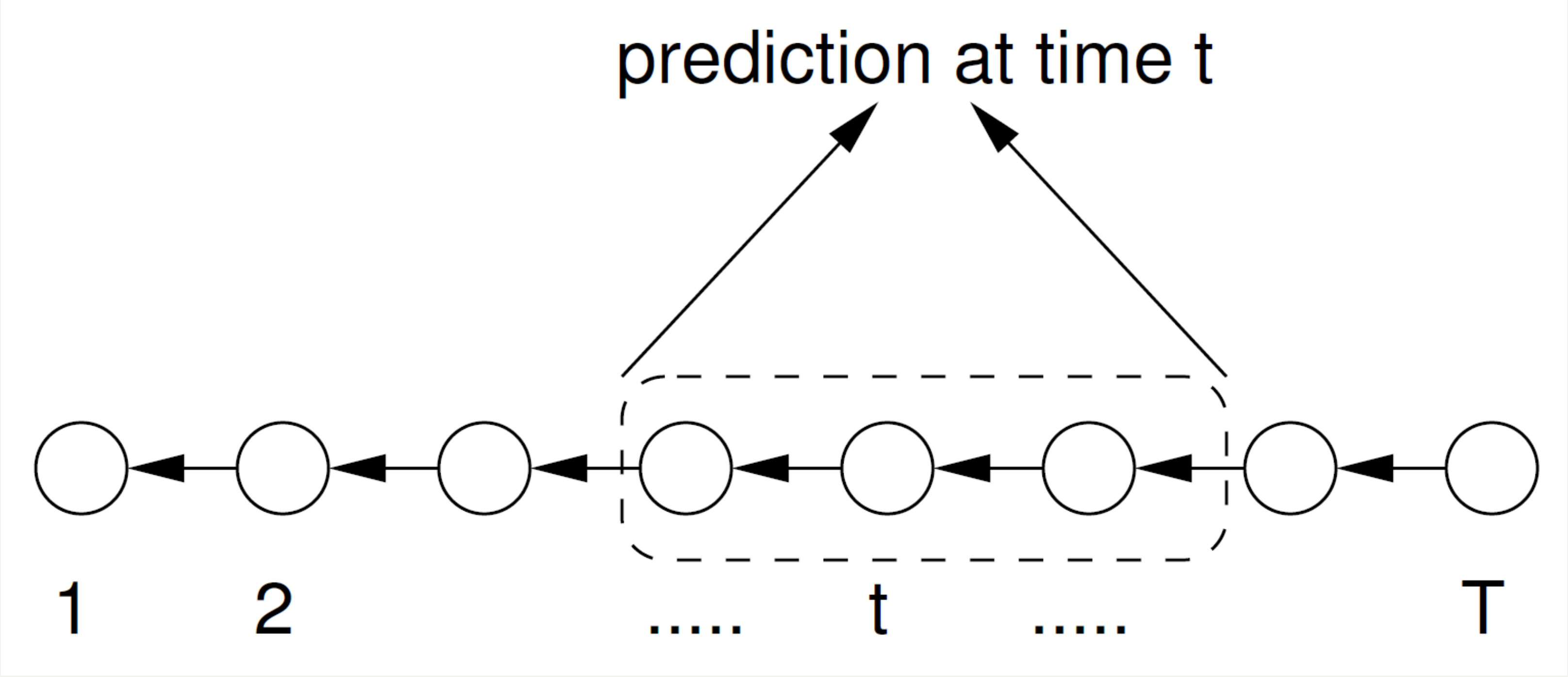


time or a sequence of presented elements

Learning Sequences

Sequences usually **model processes in time (actions, movements)** and are sequentially processed in time to predict next data (conclusions, reactions).

Sequences can have **variable length**, but typical machine learning models use a fixed number of inputs (fixed-size window) as a prediction context:

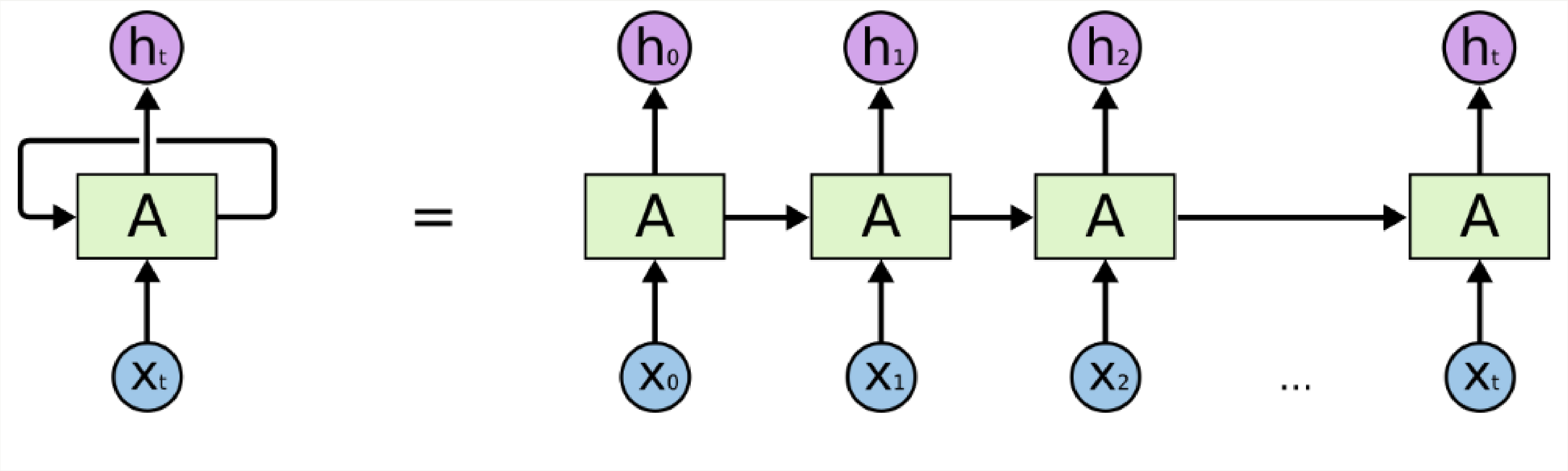


Sequential Nature of Thinking

- **Human thinking process** does not start from scratch every second for each pattern as is usually processed in CNNs and classic artificial neural networks – what is their major shortcoming between others.
- We always take into account **previous words, situations, and states** of our brains, not throwing away all previous thoughts during e.g. speech recognition, machine translation, entity names recognition, sentiment classification, music generation, or image captioning.
- Our intelligence works so well because it is not started again and again for every new situation but incorporates the **knowledge** that is gradually formed **in time**. Thanks to it, all next intelligent processes take into account our **previous experiences**.

Recurrent Neural Networks

- **Recurrent neural networks** address this issue, implementing various loops, allowing information to persist, and gradually processing data in time (following time steps).
- We can into account previous state of the network, previous inputs and/or previous outputs during computations.
- This **chain-like nature** reveals that recurrent neural networks are intimately related to **sequences** and **lists**, and are the natural neural network architecture for such data.



Natural Language Processing (NLP) and Representation of Words

How to represent and work with a language?

Natural Language Processing

Natural Language Processing (NLP) includes various tasks of language analysis, understanding, translation, generation, classification and clustering, so we need to operate on words, parts of words, sequences of words, and sentences.

To work with words, we need:

- an efficient way to represent words and their meaning,
- a possibility to associate the words with their meaning (semantics),
- to be able to order and process words in sequences and contexts

because each sequence of words cares the meaning of important dependences between words put in given orders, e.g.

He likes to take part in computational intelligence exercises.

He exercises computational intelligence to take part in likes.

One-hot Encoding of Words

We usually use any kind of a word dictionary (a vocabulary of the processed language) in which each word is represented as a one-hot vector that consists of zeros except to the single position representing a given word that equals to 1. Each represented word has an exact position in this vector and is represented by 1 in this vector that contains zeros in all the other positions:

No	Dictionary	a	and	word	zyzzyva
1	a	1	0	0	0
	…	…	…	…	…
982	and	0	1	0	0
	…	…	…	…	…
132847	word	0	0	1	0
	…	…	…	…	…
171476	zyzzyva	0	0	0	1

One-hot vectors are often used to represent a sequence of words on the inputs of the recurrent neural networks (RNNs) as well as the other types of neural networks.

One-hot Encoding of Words

The words are usually limited to a given number of the most frequently used words in a given language (taken after the chosen frequency dictionary), and next, they are sorted in their alphabetical order to accelerate search algorithms:

WORD FREQUENCY DICTIONARY

TOP_NO	WORD	FREQUENCY
1	a	21166065
...	...	...
446	apple	61047
...	...	...
743	banana	7722
...	...	...
910	bike	30788
...	...	...
1247	car	244367
...	...	...
1443	child	242214
...	...	...
4875	king	101534
...	...	...
5328	man	741815
...	...	...
6087	orange	23088
...	...	...
6761	prince	30606
...	...	...
6762	princess	17992
...	...	...
7016	queen	34815
...	...	...
9300	truck	50566
...	...	...
9885	woman	316749
...	...	...
10000	zoo	10229

ONE-HOT VECTORS FOR THE SELECTED WORDS OF THIS DICTIONARY

	a	apple	banana	bike	car	child	king	man	orange	prince	princess	queen	truck	woman	zoo
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
446	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
743	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
910	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1247	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1443	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4875	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5328	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6087	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6761	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6762	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7016	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9300	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9885	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

One-hot vectors can be easily encoded using Keras Tokenizer:

```
from keras.preprocessing.text import Tokenizer
```

Tokenizer for One-hot Vectors

We should use the Tokenizer to create one-hot vectors for given samples:

```python
from keras.preprocessing.text import Tokenizer

samples = ['The mouse came out of the hole and ate my breakfast.',
           'The dog sat on the mat and ate my lunch.',
           'The cat lipped through the window and ate my dinner.']

# We create a tokenizer, configured to only take into account the top-1000 most common words
tokenizer = Tokenizer(num_words=1000)
# This builds the word index
tokenizer.fit_on_texts(samples)

# This turns strings into lists of integer indices.
sequences = tokenizer.texts_to_sequences(samples)
print ("sequences")
print (sequences)

# You could also directly get the one-hot binary representations.
# Note that other vectorization modes than one-hot encoding are also supported if necessary.
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')
print ("one_hot_results")
print (one_hot_results)

# This is how you can recover the word index that was computed
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
sequences
[[1, 5, 6, 7, 8, 1, 9, 2, 3, 4, 10], [1, 11, 12, 13, 1, 14, 2, 3, 4, 15], [1, 16, 17, 18, 1, 19,
2, 3, 4, 20]]
one_hot_results
[[0. 1. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]
 [0. 1. 1. ... 0. 0. 0.]]
Found 20 unique tokens.
```

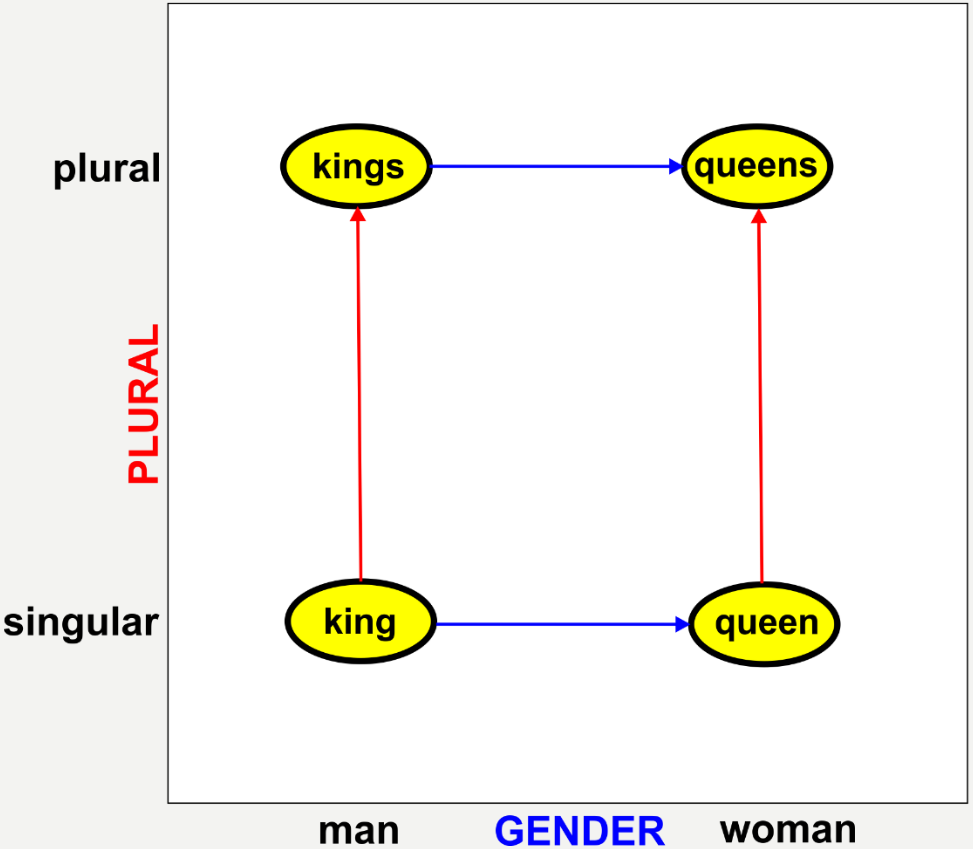
Word Meaning and Embeddings

How to represent the meaning of the words?

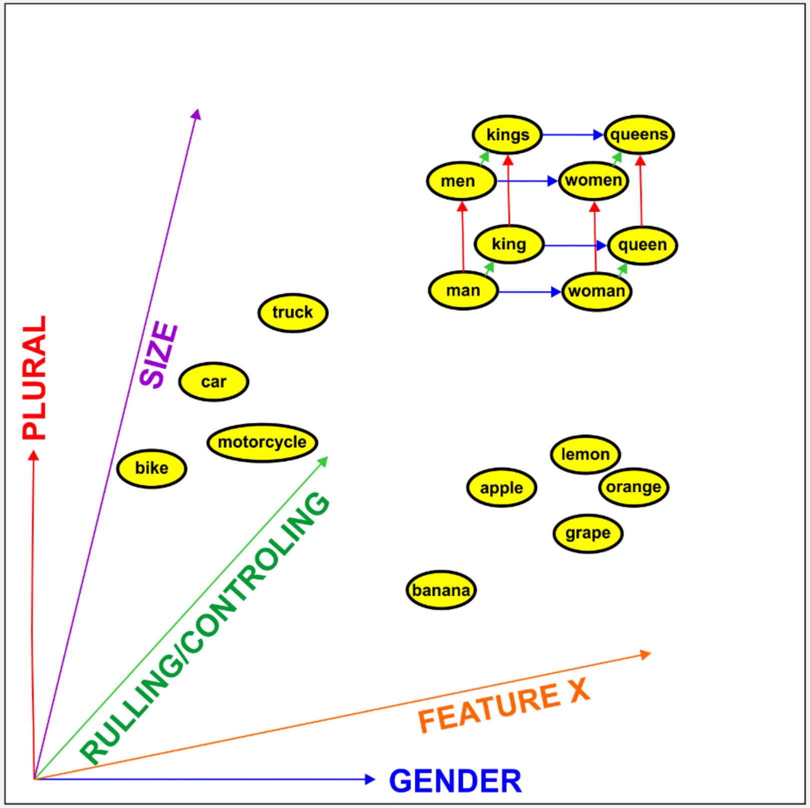
Word Meaning and Embeddings

The word embeddings use multidimensional structure, which puts similar-meaning words close in the embedding space:

WORD EMBEDDINGS



WORD EMBEDDINGS



For instance, the words “king" and “queen" are close in many aspects, so they should be close in the embedding space.

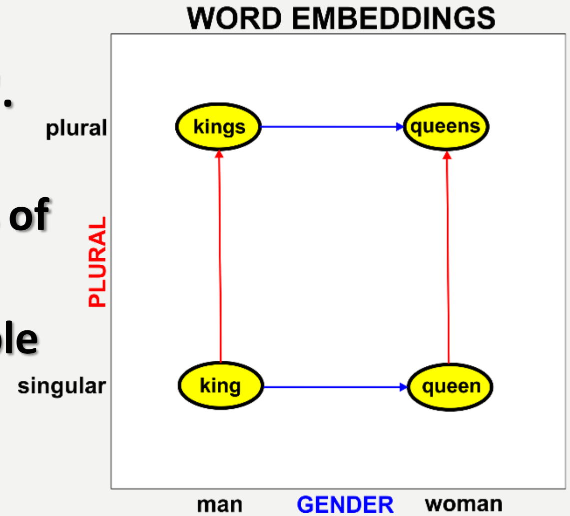
Word Embeddings

Word embeddings are meant to map human language into a geometric space. In a reasonable embedding space, we would expect:

- Synonyms to be embedded into similar word vectors.
- The geometric distance (e.g. L2 distance, also known as the Euclidean norm) between any two-word vectors to relate to the semantic distance of the associated words.
- Words meaning very different things would be embedded to points far away from each other, while related words would be close.
- Specific directions in the embedding space would be meaningful.

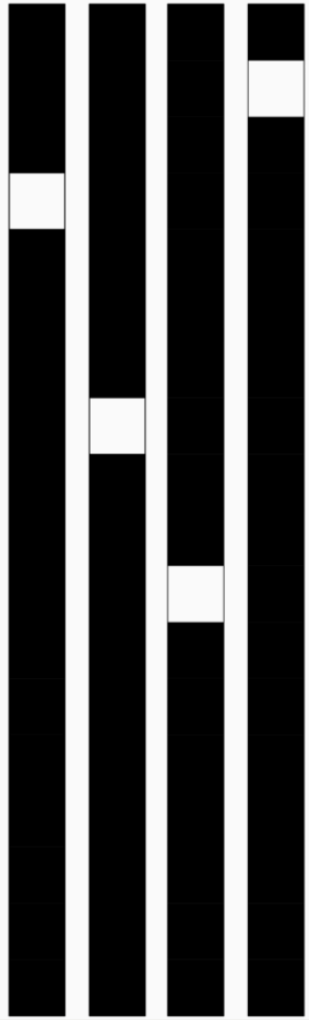
The geometric relationships between word vectors should reflect the semantic relationships between these words and all others. Common examples of meaningful geometric transformations are, e.g., "gender vectors" and "plural vector":

- For instance, by adding a "female vector" to the vector "king", one obtains the vector "queen".
- By adding a "plural vector", one obtains "kings". Word embedding spaces typically feature thousands of such interpretable and potentially useful vectors.
- The words "accurate" and "exact" are interchangeable in most sentences, so they should be close in the embedding space.

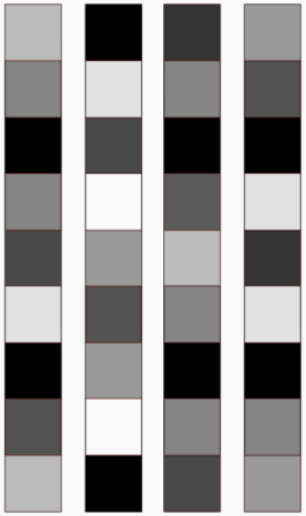


Comparison of Word Representations

One-hot word vectors:
- Sparse
- High-dimensional
- Hard-coded



Word embeddings:
- Dense
- Lower-dimensional
- Learned from data



These two approaches deliver two different solutions!

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/ahdydci.php
4. Nikola K. Kasabov, Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence, In Springer Series on Bio- and Neurosystems, Vol 7., Springer, 2019.
5. Holk Cruse, Neural Networks as Cybernetic Systems, 2nd and revised edition
6. R. Rojas, Neural Networks, Springer-Verlag, Berlin, 1996.
7. Convolutional Neural Network (Stanford)
8. Visualizing and Understanding Convolutional Networks, Zeiler, Fergus, ECCV 2014.

BIBLIOGRAPY

10. Home page for this course: http://home.agh.edu.pl/~horzyk/lectures/ahdydkbcidmb.php
11. LSTM cells from scratch and the code
12. Understanding LSTM
13. Francois Chollet, Deep Learning with Python, Manning, Nov. 2017, ISBN 9781617294433
14. Understanding GRU