UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

Natural Language Processing

Text Representation

LESSON 6

prof. Antonino Staiano

M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples

NLP from symbols to numbers

- Natural language is inherently a discrete symbolic representation of human knowledge
- Sound is transformed into letters or ideograms and these discrete symbols are composed to obtain words, then

Words $\xrightarrow{\text{form}}$ Sentences $\xrightarrow{\text{form}}$ Texts, discourses, dialogs $\xrightarrow{\text{convey}}$ Knowledge, Emotions, …

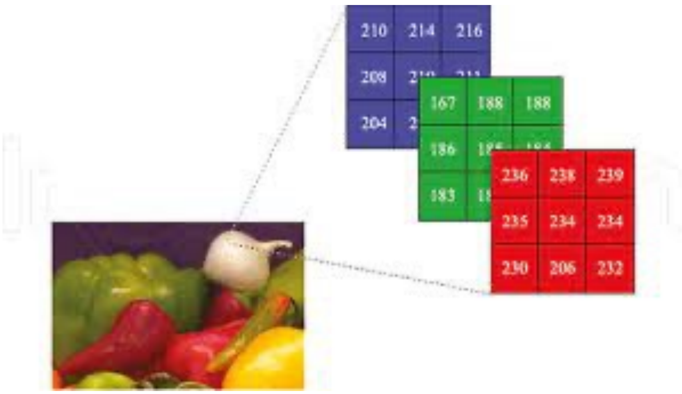
- The composition of symbols in words and of words in sentences follow rules that both the listener and the speaker know

Text representation

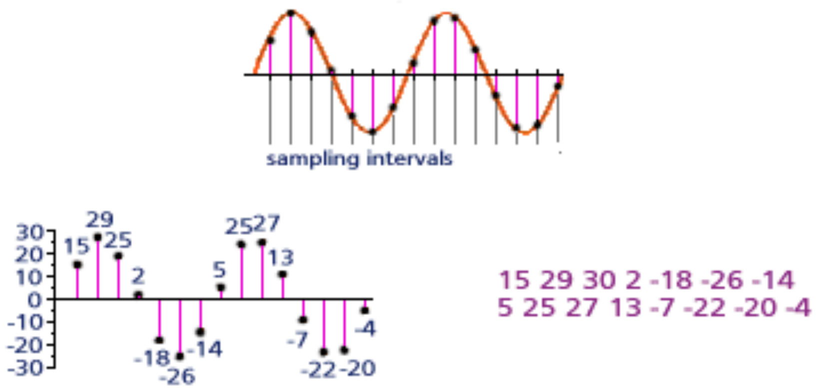
- In NLP and ML, it is mandatory to encode text data into a suitable numerical form
  - The encoding is fundamental for good-quality results
    - Trash in, Trash out!
- How do we transform a given text into a numerical form to feed it into an NLP or ML algorithm?
  - This conversion from raw text to a suitable numerical form is called text representation

Feature representation

- A common step in any ML task, whether the data is text, images, video or speech
  - Nonetheless, feature representation is much more involved for text as compared to other data formats
- For image and speech is straightforward



An RGB image and its encoding



Speech signal representation

Words and Meaning

- What is the **meaning** of a word?
  - The linguistic study of word meaning is called **lexical semantics**
- A model of word meaning should allow us to relate different words and draw inferences to address meaning-related tasks

Words and meaning

- A word form is associated with a single lemma, the citation form used in dictionaries
- Example
  - Word forms **sing**, **sang**, **sung** are associated with the lemma **sing**
- A word form can have multiple meanings; each meaning is called a word sense, or sometimes a synset
- Example
  - The word form **mouse** can refer to the rodent or the cursor control device

Words and meaning

- Lexical semantic relationship between words are important components of word meaning
- Two words are synonyms if they have a common word sense
- Example
  - car and automobile
- Two words are similar if they have similar meanings
- Example
  - car and bicycle
- Two words are related if they refer to related concepts.
- Example
  - car and gasoline

Words and meaning

- Two words are antonyms if they define a binary opposition
  - Example: hot and cold
- One word is a hyponym of another if the first has a more specific sense. Notions of hypernym or hyperonym are defined symmetrically
  - Example: car and vehicle
- Words can have affective meanings, implying positive or negative connotations / evaluation
  - Example: happy and sad; great and terrible

How do we represent meaning in a computer?

- Previously commonest NLP solution
  - Use, e.g., Wordnet, a thesaurus containing lists of synonym sets and hypernyms ("is a" relationships)

Synonym set containing "good"

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
          ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

Hypernyms of "panda"

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

WordNet

- WordNet (English) is a hand-built resource containing 117,000 synsets
  - sets of synonymous words (See *http://wordnet.princeton.edu/*)
- Synsets are connected by relations such as
  - *hyponym/hypernym* (IS-A: chair-furniture)
  - *meronym* (PART-WHOLE: leg-chair)
  - *antonym* (OPPOSITES: good-bad)
- *globalwordnet.org* now lists wordnets in over 50 languages (but variable size/quality/licensing)

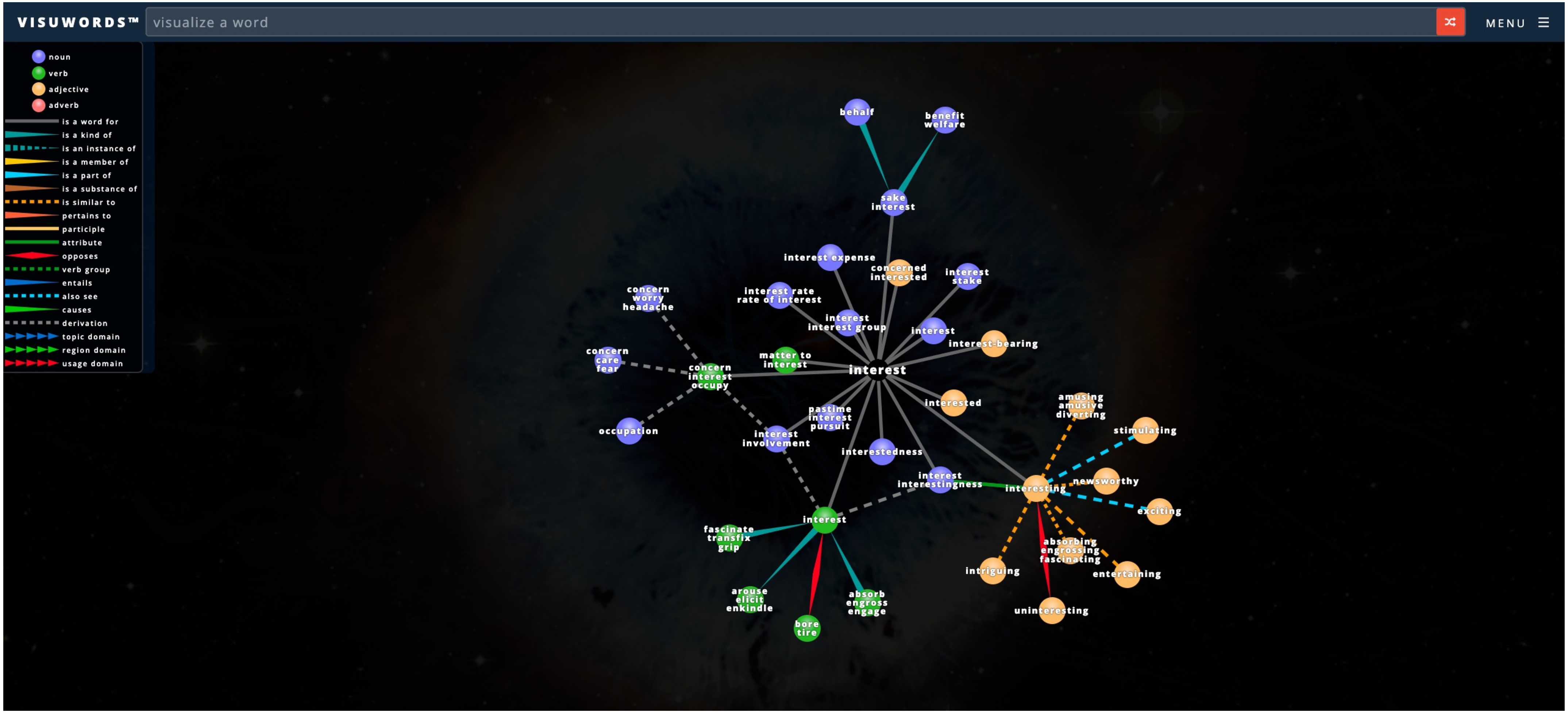
NLTK and WordNet

- NLTK provides an excellent API for looking things up in WordNet

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('car')
[Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'),
Synset('car.n.04'), Synset('cable_car.n.01')]
>>> wn.synset('car.n.01').definition()
u'a motor vehicle with four wheels; usually propelled by an
internal combustion engine'
>>> wn.synset('car.n.01').hypernyms()
[Synset('motor_vehicle.n.01')]
```

- Note: WordNet uses an obscure file format, so reading the files directly is not recommended!

Visualizing WordNet



Problems with resources like Wordnet

- A useful resource but missing nuance
  - e.g., "proficient" is listed as a synonym for "good"
    - This is only correct in some contexts
  - Also, WordNet list offensive synonyms in some synonym set without any coverage of the connotations or appropriateness of words
- Missing new meanings of words:
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can't be used to accurately compute word similarity

Distributional semantics

- It is very difficult to define the notion of word sense in a way that can be understood by computers
  We take a radically different approach, already foreseen in the following works:
  - "*The meaning of a word is its use in the language*"
    - *Ludwig Wittgenstein Philosophical Investigations, 1953*
  - "*You shall know a word by the company it keeps*"
    - *John Rupert Firth Selected papers, 1957*

Distributional semantics

- Distributional semantics develops methods to quantify semantic similarities between words based on their distributional properties, i.e., neighboring words
- The basic idea lies in the so-called distributional hypothesis
  - language elements with similar distributions have similar meanings
  - the meaning of a word is defined by its distribution in language use

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent banking

- The basic approach is to collect distributional information in high-dimensional vectors, and to define distributional/semantic similarity in terms of vector similarity

Text representation

- There are a variety of approaches, depending both by the task to be addressed and the model to be employed
  - Basic vectorization approaches
  - Distributed representation
- Here, we'll overview basic approaches, and just introduce distributed representation deferring its details when needed

An introducing scenario

- We're given a labeled text corpus and asked to to build a sentiment analysis model
- The model needs to understand the meaning of the sentence
  - The crucial points are
    1. Break the sentence into lexical units (i.e., lexemes, words or phrases)
    2. Derive the meaning for each lexical units
    3. Understand the syntactic (grammatical) structure of the sentence
    4. Understand the context in which the sentence appears
  - The semantics (meaning) of the sentence is the combination of the above points
  - Any good text representation scheme reflect the linguistic properties of the text in the best possible way

Vector Space Models

- Text units, i.e., characters, phonemes, words, phrases, sentences, paragraphs, and documents, are represented with vectors of numbers
- Simplest form
  - Vectors of identifiers, e.g., index numbers in a corpus vocabulary
- The most common way to measure the similarity between two text blobs is the cosine similarity $\mathrm{sim}_{\cos}(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\|_2 \|\boldsymbol{v}\|_2} = \frac{\sum_i \boldsymbol{u}_{[i]} \cdot \boldsymbol{v}_{[i]}}{\sqrt{\sum_i (\boldsymbol{u}_{[i]})^2} \sqrt{\sum_i (\boldsymbol{v}_{[i]})^2}}.$
  - Sometimes, Euclidean distance is also used
- The difference between representation schemes consists in how well the resulting vector captures the linguistic properties of the text it represents

Basic approaches

- Map each word in the vocabulary V of the text corpus to a unique ID (integer)
- Each sentence or document in the corpus is a V-dimensional vector
- Example
  - Let's consider a toy corpus

D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

  - Lowercasing text and ignoring punctuation the vocabulary is comprised of six words, V= [dog, bites, man, eats, meat, food]
  - Every document in this corpus can be represented with a six-dimensional vector

One-hot encoding

- Each word w in the vocabulary is given a unique integer ID, $w_{id} \in \{1, \dots, |V|\}$
- Each word is represented by a |V|-dimensional binary vector, filled with all 0s barring the *index* $= w_{id}$,where we put a 1
- The representation for individual words is then combined to for a sentence representation
- Example
  - V= [dog, bites, man, eats, meat, food]
  - ID: dog =1, bites = 2, man =3, meat = 4, food = 5, eats = 6
    - dog = [1 0 0 0 0 0] or man = [0 0 1 0 0 0], etc.
  - Document D1=[[1 0 0 0 0 0] [0 1 0 0 0 0] [0 0 1 0 0 0]]
  - Similarly, for D2, D3, and D4

D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

One-hot encoding cons

- The size of a one-hot vector is proportional to the size of V
  - Many real-world corpora have large vocabularies
  - Sparse representation (i.e., most of entries are 0)
- Not fixed-length representation
  - A text with 10 words gets a longer representation than a text with 5 words
  - Most learning algorithm work with feature vectors of the same length
- If words are atomic units, there's no notion of similarity
  - Consider run, ran, and apple. Run and ran have similar meanings as opposed to run and apple, but they're all equally apart
  - Semantically, very poor at capturing the meaning of the word in relation to other words
- Not capable of handling the *out-of-vocabulary* (OOV) problem
  - There is no way to represent new words from test sets, not present in the training corpus

Problem with words as discrete symbols

- **Example:** in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"
- But:

$$\text{motel} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$$

$$\text{hotel} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

- These two vectors are orthogonal
  - There is no natural notion of **similarity** for one-hot vectors!

Bag of Words

- Bag of Words (BoW) represents the text as a bag (collection) of *words* while ignoring the order and the context
- The intuition is that a text is characterized by a unique set of words
  - If two text pieces have the same words, then they are similar
- BoW maps words to unique integer IDs between 1 and $|V|$
  - Each document in the corpus is converted into a $|V|$-dimensional vector where the i-th component of the vector $i=w_{id}$ is the number of times the word w occurs in the document
  - *Obs.*: sometimes we don't care about the frequency of occurrence of words, but just want to represent whether the word exists or not in the text

Bag of Words

- Example

D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

- ID: dog =1, bites = 2, man =3, meat = 4, food = 5, eats = 6
  - D1 = [1 1 1 0 0 0]
  - D2 = [1 1 1 0 0 0]
  - D3 = [1 0 0 1 0 1]
  - D4 = [0 0 1 0 1 1]

BoW pros and cons

- Simple to understand and implement
- Documents having the same words will have their vector representation similar in Euclidean space
- Fixed-length encoding for any sentence of arbitrary length
- The size of the vector increases with the size of the vocabulary
  - Sparsity continues to be a problem
- It does not capture the similarity between different words that mean the same thing
  - "I run", "I ran", and "I ate"
    - The three BoW vectors are all equally apart
- No way to handle out-of-vocabulary words
- Word order information is lost
  - D1 and D2 have the same representation in the example

Bag of N-grams

- All the representation schemes seen so far treat words as independent units
  - There's no notion of phrases or word ordering
- The bag of n-grams breaks texts into chunks of n contiguous words
  - Each chunk is called n-gram
- The corpus vocabulary, V, is a collection of all unique n-grams across the text corpus
  - Each document is represented by a |V|-sized vector that contains the frequency counts of n-grams present in the document

Bag of N-grams

- Example: 2-gram (bigram) model
  - V = {*dog bites, bites man, man bites, bites dog, dog eats, eats meat, man eats, eats food*}
  - |V| = 8
  - D1 = [1 1 0 0 0 0 0 0]
  - D2 = [0 0 1 1 0 0 0 0]
  - D3 = [0 0 0 0 1 1 0 0]
  - D4 = [0 0 0 0 0 0 1 1]

D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

- Obs.: increasing the value of n a larger context is incorporated
  - However, also the sparsity increases

Bag of N-grams pros and cons

- Some context and word-order information is captured
  - The vector space can capture some semantic similarity
- As n increases, dimensionality (and therefore sparsity) quickly increases
- No way to address the OOV problem

TF-IDF

- Term Frequency-inverse Document Frequency (TF-IDF) introduces the notion of importance of words in a document
  - Commonly used representation scheme for information-retrieval systems
- Intuition
  - If a word *w* appears many times in a document d_i but does not occur much in the rest of the documents d_j in the corpus, then *w* must be of great importance for d_i
  - The importance of *w* should increase in proportion to its frequency in d_i (TF), but at the same time its importance should decrease in proportion to the word's frequency in other documents d_j (IDF) in the corpus
  - TF and IDF are combined to form the TF-IDF score

TF-IDF

- *TF* (term frequency) measures how often a term or word occurs in a given document
- The quantity is normalized by the length of the document
  - $TF(w,d) = \frac{\# \, of \, occurences \, of \, word \, w \, in \, document \, d}{Total \, \# \, of \, words \, in \, document \, d}$
- *IDF* (inverse document frequency) measures the importance of the term across a corpus
  - $IDF(w) = \log_e \frac{Total \, \# \, of \, documents \, in \, the \, corpus}{\# \, of \, documents \, with \, w \, in \, them}$
- *TF-IDF score = TF*IDF*

TF-IDF

- Example
  - Size of corpus N =4

D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

- The TD-IDF vector representation for D1 is

Word	IDF score	TF-IDF score
dog	$\log_2(4/3)=0.4114$	0.4114x0.33 = 0.136
bites	$\log_2(4/2)=1$	1x0.33=0.33
man	0.4114	0.4114x0.33 = 0.136
eats	1	1x0=0
meat	$\log_2(4/1)=2$	2x0=0
food	2	2x0=0

TF-IDF

- There are several variations of the basic TF-IDF formula used in practice
  - Avoiding possible zero divisions
  - Do not entirely ignore terms that appear in all documents
- TF-IDF could be used to compute the similarity between two texts using Euclidean distance or cosine similarity
- it still suffers from the curse of dimensionality as the previous vectorization methods

Distributed representation

- The basic approaches to vector representation share key drawbacks
  - To overcome these limitations, methods to learn low-dimensional representation were devised
  - They use neural network architectures to create dense, low-dimensional representations of words and texts
  - Distributed representation schemes significantly compress the dimensionality
    - This results in vectors that are compact and dense
  - Based on the distributional hypothesis from linguistics
    - Words that occur in similar contexts have similar meanings -> the corresponding representation vectors must be close to each other

Word vectors

- We will build a dense vector for each word, chosen so that it is similar to word vectors that appear in similar contexts, measuring similarity as the vector dot (scalar) product

$$
\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}
$$

- The obtained vectors are called word embeddings
  - Each discrete word is embedded in a continuous vector space
- They are a distributed representation

Visualizing word embeddings

- Word embeddings can be used to visualize the meaning of a word ***w***
- The most commonly used methods
  - listing the words in a vocabulary V with the highest cosine similarity with *w*
    - Locality-sensitive hashing (LSH) can be used, which hashes similar input items into the same buckets with a high probability
  - project the *d* dimensions of a word embedding down into 2 dimensions
    - *t*-distributed stochastic neighbor embedding (t-SNE) is used, preserving metric properties

Visualizing word embeddings

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word2vec

- Word2vec (Mikolov et al. 2013) is a framework for learning word vectors
- Idea:
  - We have a large corpus of text: a long list of words
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
  - Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
  - Keep adjusting the word vectors to maximize this probability
- Two different algorithms for learning word embeddings:
  - skip-gram with negative sampling (SGNS)
  - continuous bag-of-words (CBOW)