Autumn 2024, Part 2

Text preprocessing, Word Embeddings

Eliza Vialykh, @elf_lesnoy

Content

1. Text preprocessing
2. Text Vectorization

Text preprocessing

Text preprocessing

- Sentence segmentation
- Tokenization
- Stemming
- Lemmatization

Start

Every NLP task starts with a piece of text, like the following made-up customer feedback about a certain online order.

```
text = """Dear Amazon, last week I ordered an Optimus Prime action figure
from your online store in Germany. Unfortunately, when I opened the package,
I discovered to my horror that I had been sent an action figure of Megatron
instead! As a lifelong enemy of the Decepticons, I hope you can understand my
dilemma. To resolve the issue, I demand an exchange of Megatron for the
Optimus Prime figure I ordered. Enclosed are copies of my records concerning
this purchase. I expect to hear from you soon. Sincerely, Bumblebee."""
```

Start

Every NLP task starts with a piece of text, like the following made-up customer feedback about a certain online order.

```
text = """Dear Amazon, last week I ordered an Optimus Prime action figure
from your online store in Germany. Unfortunately, when I opened the package,
I discovered to my horror that I had been sent an action figure of Megatron
instead! As a lifelong enemy of the Decepticons, I hope you can understand my
dilemma. To resolve the issue, I demand an exchange of Megatron for the
Optimus Prime figure I ordered. Enclosed are copies of my records concerning
this purchase. I expect to hear from you soon. Sincerely, Bumblebee."""
```

How to feed this text to the machine?

Tokenization

The man had been taken outside
a small holdfast in the hills.

Tokenization

The man had been taken outside a small holdfast in the hills.

Tokenization

The man had been taken outside
a small holdfast in the hills.

Sentence segmentation

HX: This 64 y/o RHM had had difficulty remembering names, phone numbers and events for 12 months prior to presentation, on 2/28/95. **Sentence** This had been called to his attention by the clerical staff at his parish--he was a Catholic priest. **Sentence** He had had no professional or social faux pas or mishaps due to his memory. **Sentence** He could not tell whether his problem was becoming worse, so he brought himself to the Neurology clinic on his own referral. **Sentence**

Tokenization

Token normalization

- Stemming
- Lemmatization

Stemming

Lemmatization algorithms can be complex. For this reason we sometimes make use of a simpler but cruder method, which mainly consists of chopping off word final affixes. This naive version of morphological analysis is called **stemming**.

complete → complete

exception → except

http://snowball.tartarus.org/algorithms/russian/stemmer.html

word	stem	word	stem
в	в	п	п
вавиловка	вавиловк	па	па
вагнера	вагнер	пава	пав
вагон	вагон	павел	павел
вагона	вагон	павильон	павильон
вагоне	вагон	павильонам	павильон
вагонов	вагон	павла	павл
вагоном	вагон	павлиний	павлин
вагоны	вагон	павлиньи	павлин
важная	важн	павлиньим	павлин
важнее	важн	павлович	павлович

Стеммер Портера

Материал из Википедии — свободной энциклопедии

[править | править код]

У этого термина существуют и другие значения, см. Портер.

Стеммер Портера — алгоритм стемминга, опубликованный Мартином Портером в 1980 году. Оригинальная версия стеммера была предназначена для английского языка и была написана на языке BCPL. Впоследствии Мартин создал проект «Snowball» и, используя основную идею алгоритма, написал стеммеры для распространённых индоевропейских языков, в том числе для русского[1].

Алгоритм не использует баз основ слов, а лишь, применяя последовательно ряд правил, отсекает окончания и суффиксы, основываясь на особенностях языка, в связи с чем работает быстро, но не всегда безошибочно.

Алгоритм был очень популярен и тиражируем, в него часто вносились изменения разными разработчиками, причём не всегда удачные. Примерно в 2000 году Портер принял решение «заморозить» проект и впредь распространять одну-единственную реализацию алгоритма (на нескольких популярных языках программирования) со своего сайта.

Lemmatization

POS → form

Lemmatization is the process of reducing a word to its base form, or lemma. Unlike Stemming, lemmatization takes into account the context and grammatical features of a word, such as part of speech.

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
words = ['пеликану', 'беспрецендентно', 'побежал', 'приводит', 'красоты']
for word in words:
    p = morph.parse(word)[0]
    print(p.normal_form)
```

```
пеликан
беспрецендентный
побежать
приводить
красота
```

Lemmatization

Lemmatizer from NLTK:

- Based on **WordNet** database

Wordnet is a publicly available lexical database of over 200 languages that provides semantic relationships between its words.

PRINCETON UNIVERSITY

WordNet

A Lexical Database for English

https://wordnet.princeton.edu/

Text preprocessing

- Capital Letters
- Punctuation
- Contractions (e.g, etc.)
- Numbers (dates, ids, page numbers)
- Stop-words (“the”, “is”, etc.)
- Tags

Handful tools for preprocessing

- **NLTK**
  - nltk.stem.SnowballStemmer
  - nltk.stem.PorterStemmer
  - nltk.stem.WordNetLemmatizer
  - nltk.corpus.stopwords
- **BeautifulSoup** (for parsing HTML)
- **Regular Expressions** (import re)
- **Pymorphy2**

Text Vectorization

Learning Word Embedding

Goal: we need to transform the free-text words into numeric values

ARTICLE | FREE ACCESS

A vector space model for automatic indexing

Authors: G. Salton, A. Wong, and C. S. Yang | Authors Info & Claims

Communications of the ACM, Volume 18, Issue 11 • Pages 613 - 620
https://doi.org/10.1145/361219.361220

Published: 01 November 1975 Publication History

Check for updates

One-hot encoding

Each word is transformed into a binary vector where all elements are 0, except for one element, which is set to 1.

Hello! I am pikachu. →

Hello	1	0	0	0
I	0	1	0	0
am	0	0	1	0
pikachu	0	0	0	1

One-hot encoding

Each word is transformed into a binary vector where all elements are 0, except for the one element, which is set to 1.

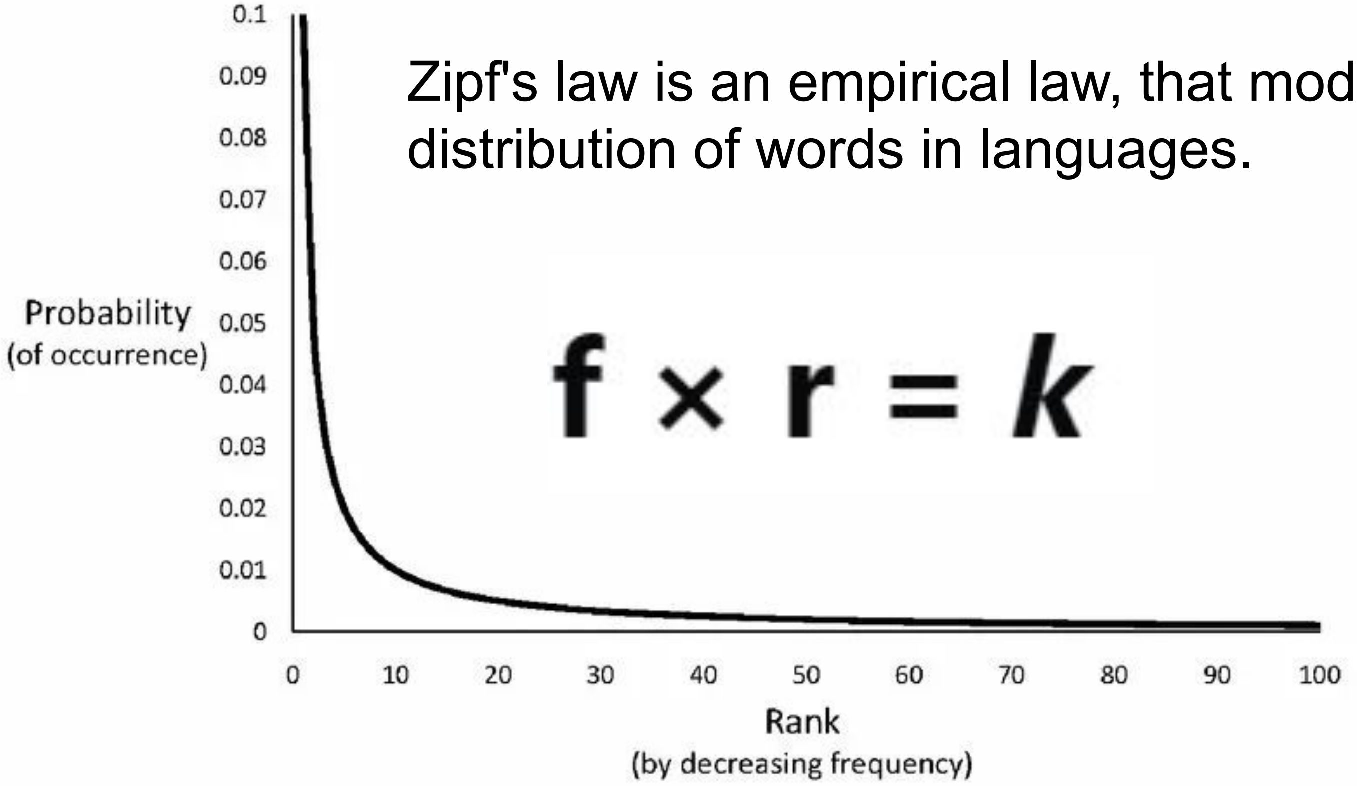
Hello! I am pikachu.

Hello	1	0	0	0
I	0	1	0	0
am	0	0	1	0
pikachu	0	0	0	1

- high dimensionality :(
- orthogonality
- lack of information about words

Zipf’s Law

Zipf's law is an empirical law, that models the frequency distribution of words in languages.



Approaches for learning word embedding

Count-based: based on the word frequency and co-occurrence matrix

Context-based: given a local context, we want to design a model to predict the target words

Bag of Words

With **BoW**, we break down sentences and paragraphs into individual words, then count how often each word appears.

	It	was	a	very	huge	bird
It was a very very very huge bird	1	1	1	3	1	1
A huge bird	0	0	1	0	1	1
Bird	0	0	0	0	0	1
Bird bird bird	0	0	0	0	0	3

TF-IDF

TF-IDF stands for term frequency-inverse document frequency,
where t denotes a single term; d a singe document, and D a collection of documents

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

$$\mathrm{tf}(t, d) = \frac{\sum_{t \in d} t}{|d|} \qquad \mathrm{idf}(t, D) = \frac{|D|}{\sum_{t \in D} t}$$

how frequently a word appeared in a given document

simply the inverse of document frequency

TF-IDF

Document_1	
Term	**Count**
girl	1
the	1
buy	2
car	1

Document_2	
Term	**Count**
the	1
cat	1
milk	2
dog	3

tf("the",d1) = 1/5 = 0.2
tf("the",d2) = 1/7 = 0.14
idf("the", D) = log(2/2) = 0

tf-idf = 0

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{\, d_i \in D \mid t \in d_i \,\}|}$$

TF-IDF

Document_1	
Term	**Count**
girl	1
the	1
buy	2
car	1

Document_2	
Term	**Count**
the	1
cat	1
milk	2
dog	3

Why is there a logarithm?

tf("the",d1) = 1/5 = 0.2
tf("the",d2) = 1/7 = 0.14
idf("the", D) = log(2/2) = 0

tf-idf = 0

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{\, d_i \in D \mid t \in d_i \,\}|}$$

TF-IDF

However, the TF-IDF matrix usually has a high dimension and sparsity, as many words are rare or not found at all in some documents.

TF-IDF

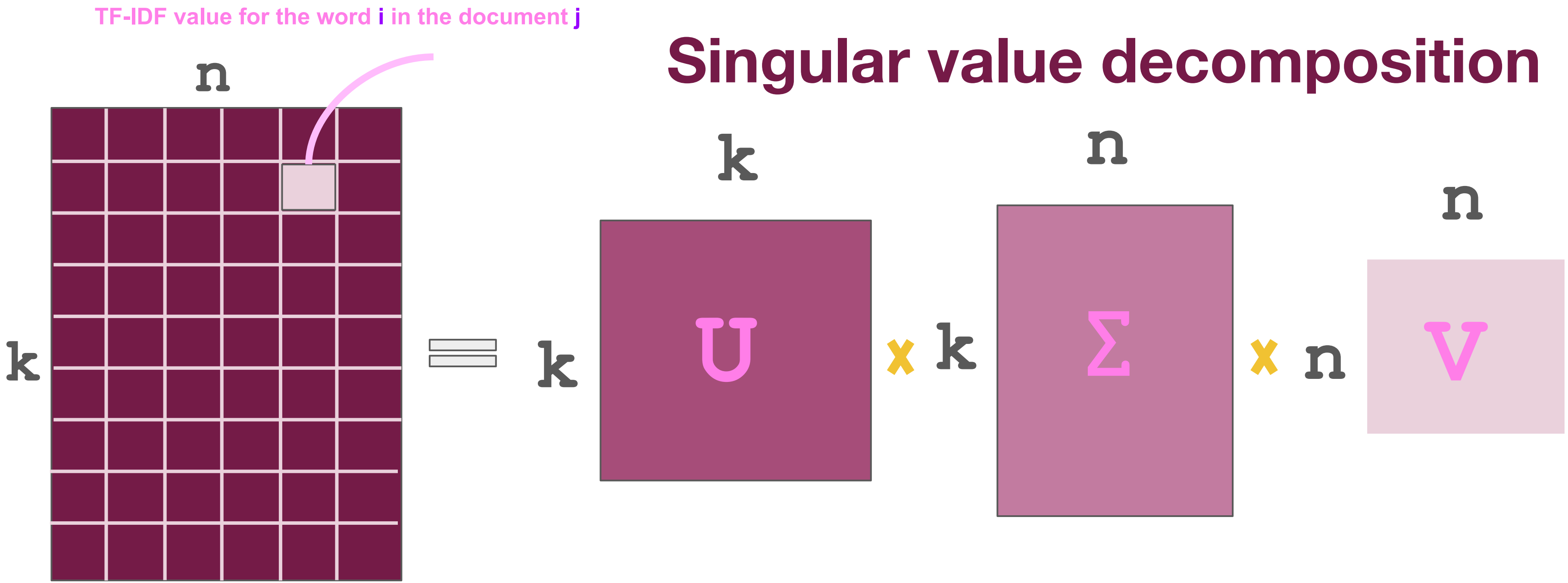
However, the TF-IDF matrix usually has a high dimension and sparsity, as many words are rare or not found at all in some documents.

LSA (Latent Semantic Analysis)

LSA (Latent Semantic Analysis)

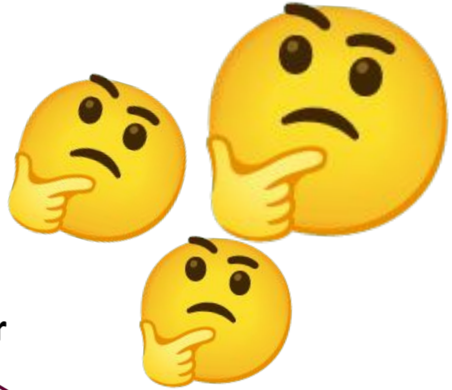
$$M = U\Sigma V^T$$

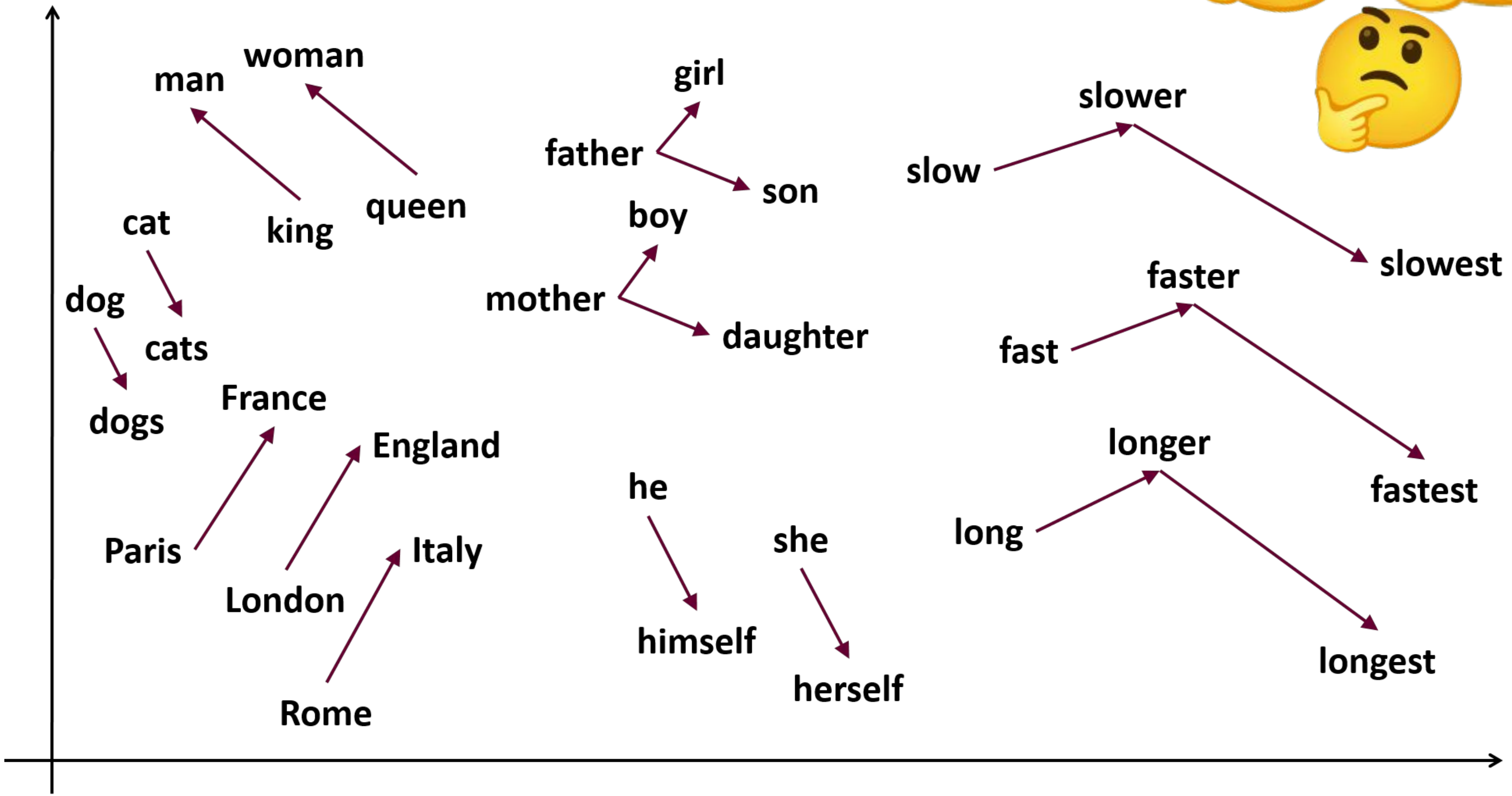
```
n = words
k = documents
M = k X n
```



While LSA effectively captures latent semantic structures, it provides more abstract relationships between terms and documents. We are aiming for more explicit and nuanced word associations…

Context vectors





Softmax

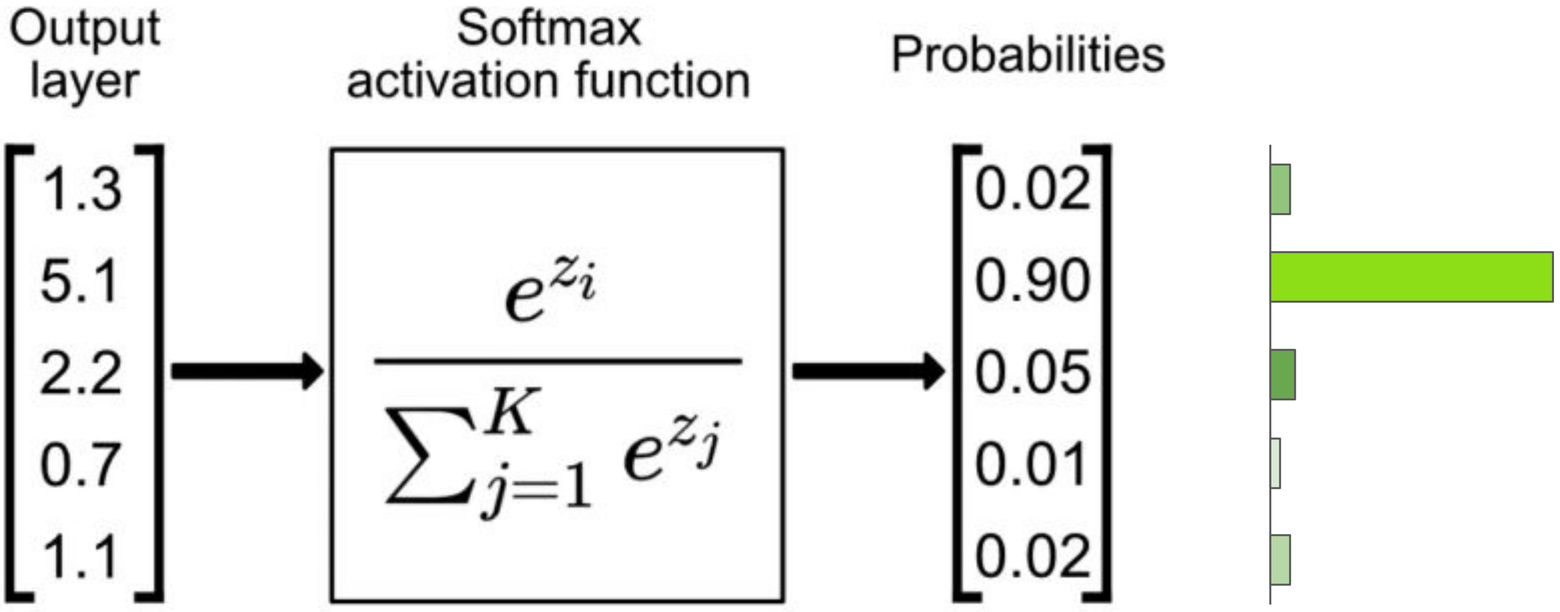
At the output **we want to get the probability distribution** for the next token

$$\underbrace{\begin{pmatrix} \\ \\ \cdots\cdots\cdots\cdot \\ \\ \\ \end{pmatrix}}_{\text{weights}} \underbrace{\begin{bmatrix} \\ \cdots\cdots \\ \cdots\cdot \\ \\ \end{bmatrix}}_{\text{data}} = \begin{bmatrix} 0.5 \\ 1.8 \\ 2.9 \\ -1.2 \\ -4 \\ 3.8 \\ -4.2 \\ 3.2 \\ 5.1 \\ -5.4 \\ -1.2 \\ 4.4 \end{bmatrix}$$

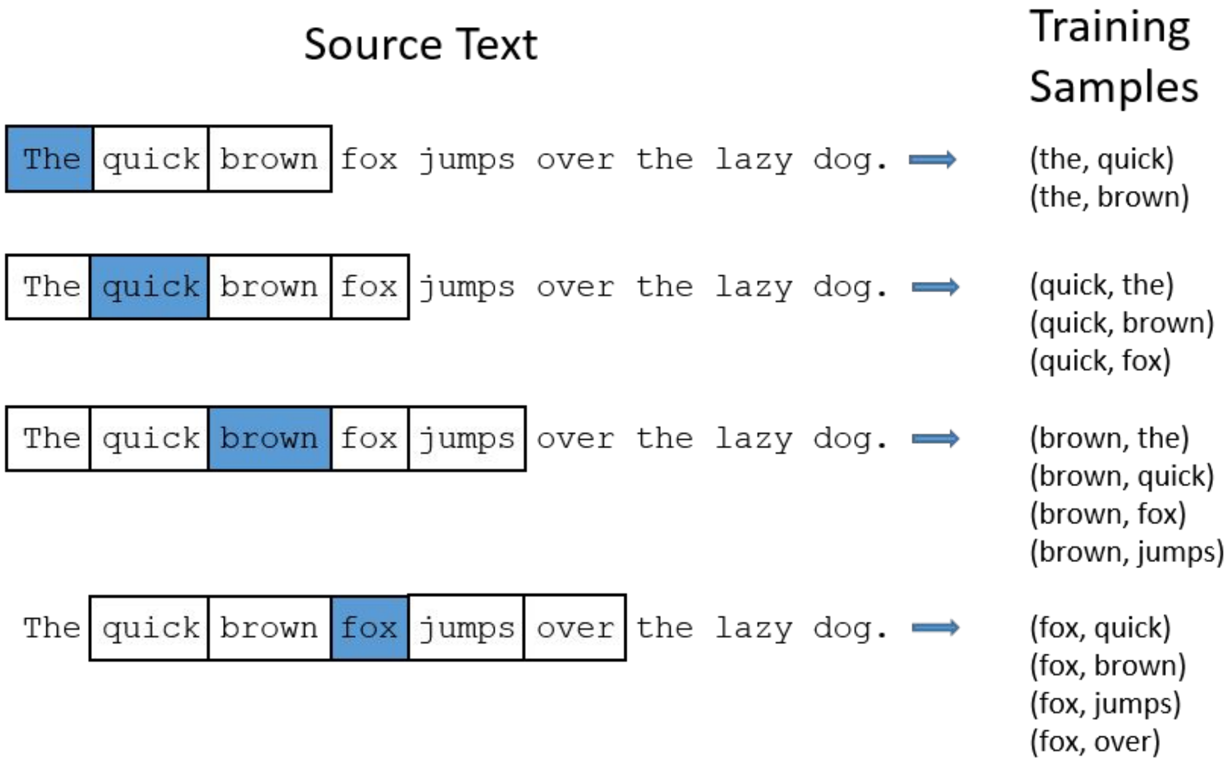
but probability is 0≤X≤1

Softmax

At the output we want to get the probability distribution for the next token



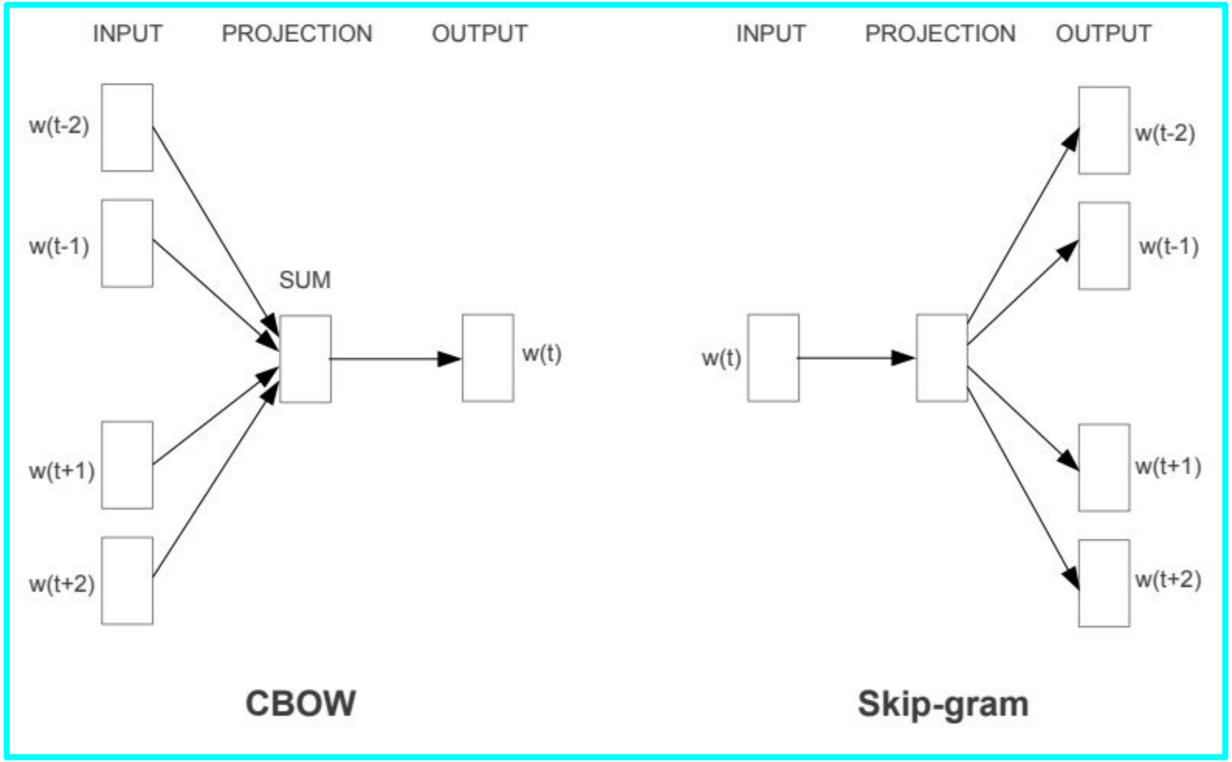
Word2Vec



https://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

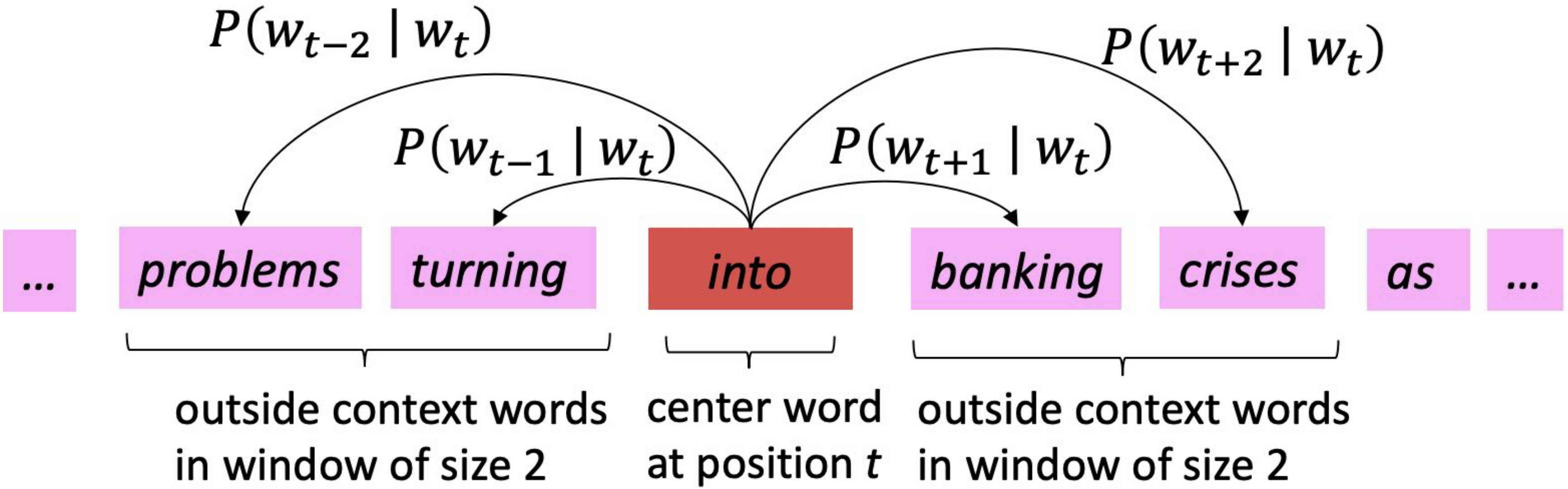
Word2Vec

The technique works by training a neural network **to forecast a word based on its neighboring words.** The neural network may be constructed utilizing one of two architectures: **Continuous Bag of Words (CBOW)** or **Skip-Gram**.



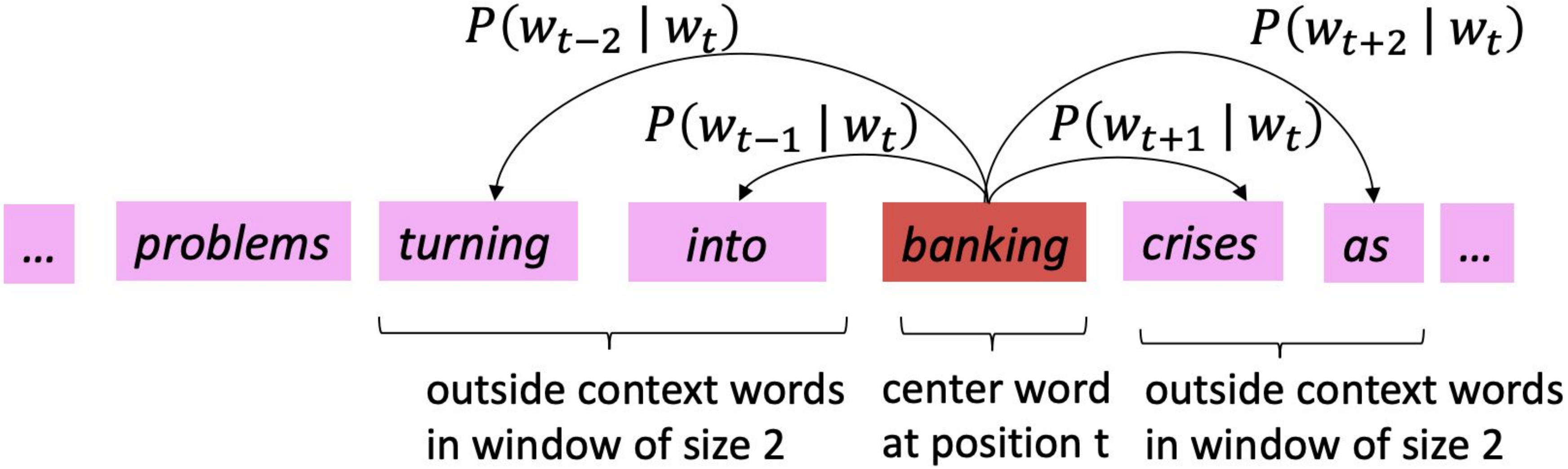
https://arxiv.org/pdf/1301.3781

Skip-Gram



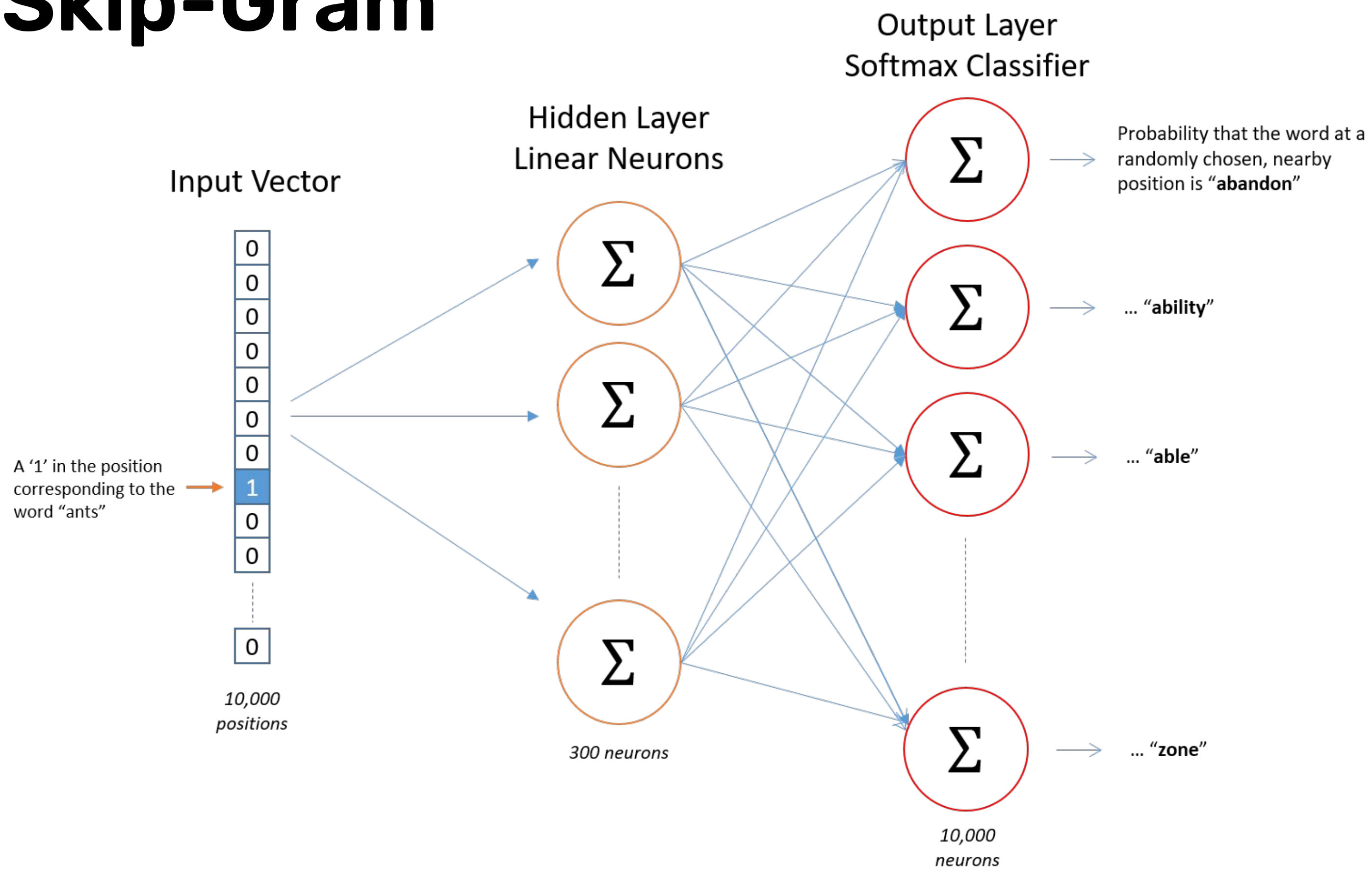
https://web.stanford.edu/class/cs224n/

Skip-Gram



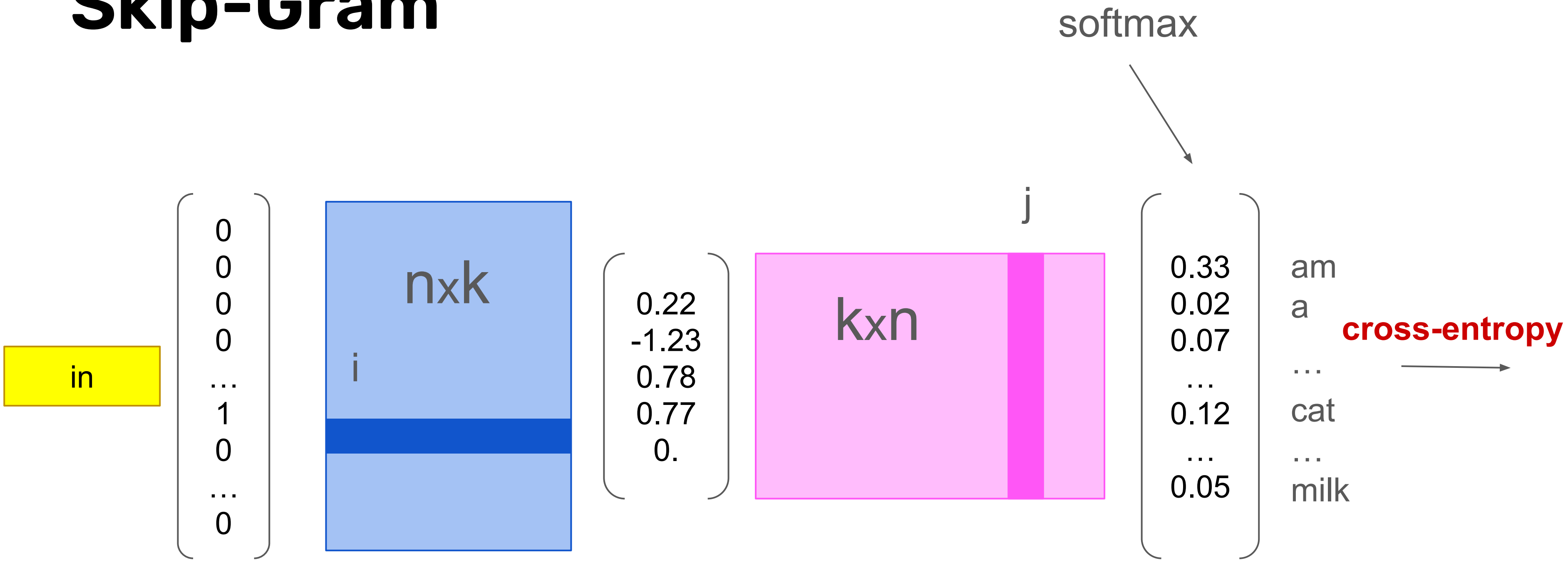
https://web.stanford.edu/class/cs224n/

Skip-Gram



https://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

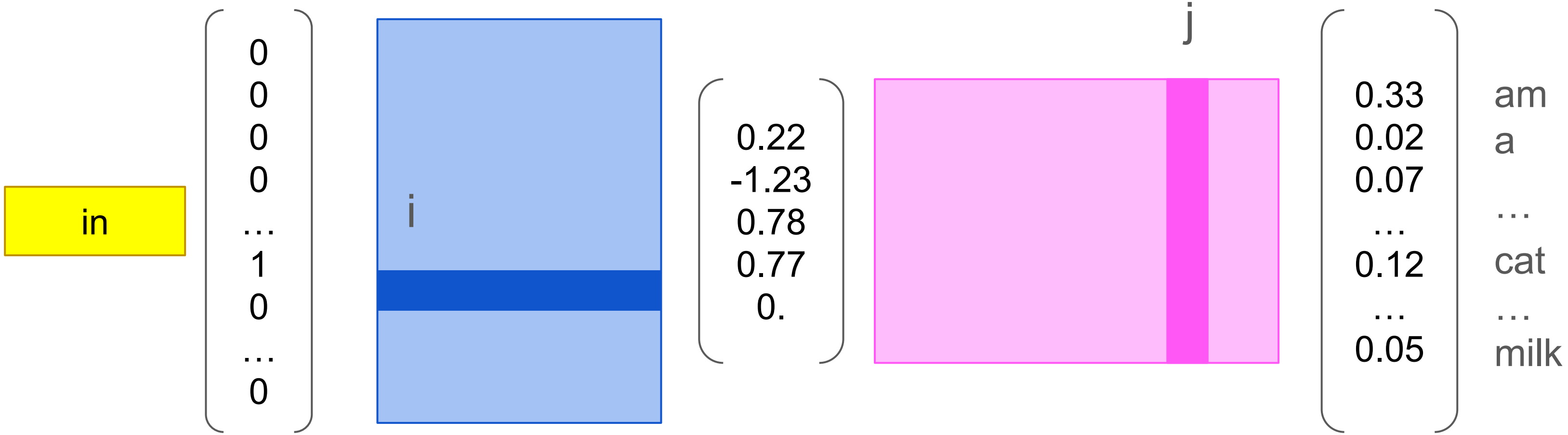
Skip-Gram



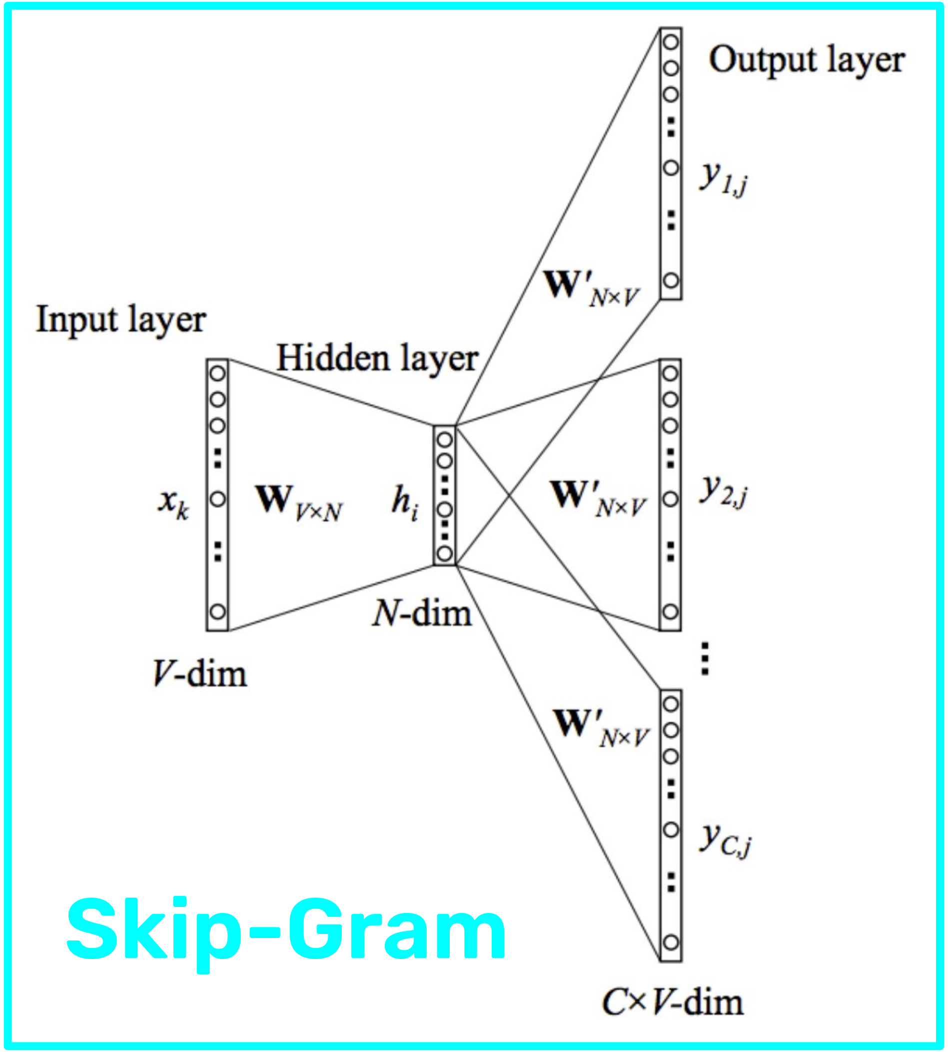
https://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

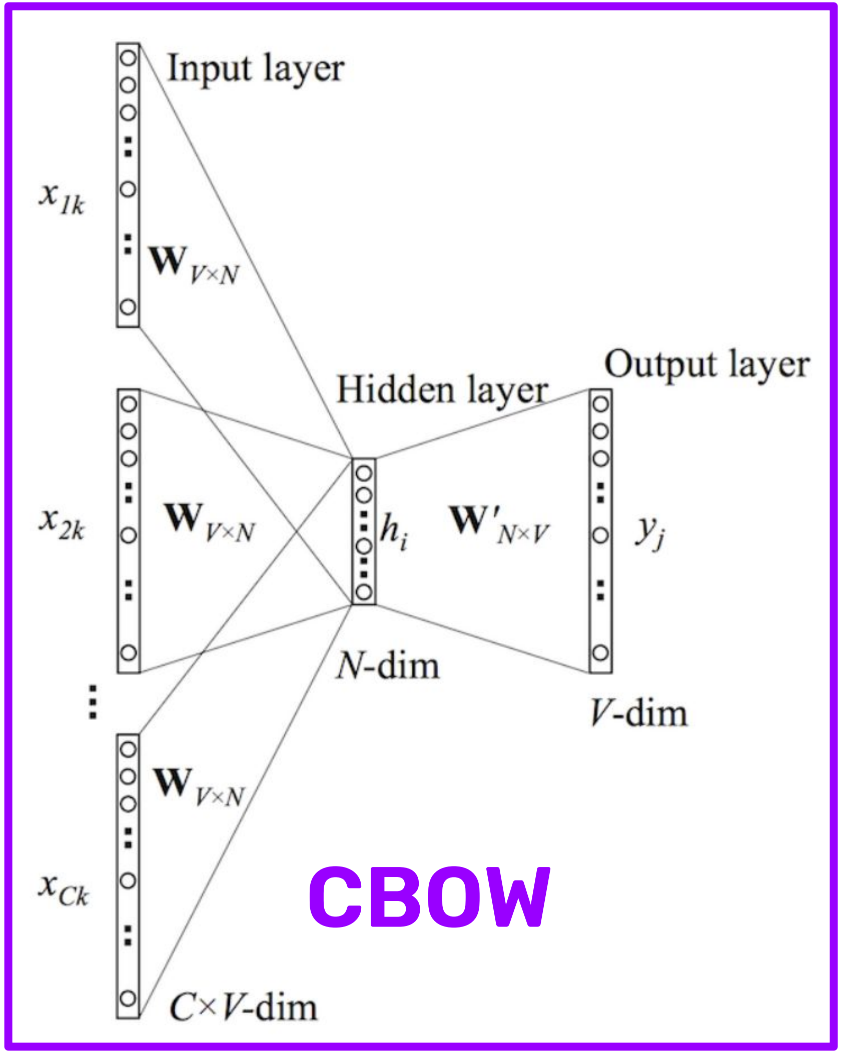
Skip-Gram

$$\begin{bmatrix}0 & 0 & 0 & \boxed{1} & 0\end{bmatrix} \times \begin{bmatrix}17 & 24 & 1\\ 23 & 5 & 7\\ 4 & 6 & 13\\ \boxed{10} & \boxed{12} & \boxed{19}\\ 11 & 18 & 25\end{bmatrix} = \begin{bmatrix}10 & 12 & 19\end{bmatrix}$$

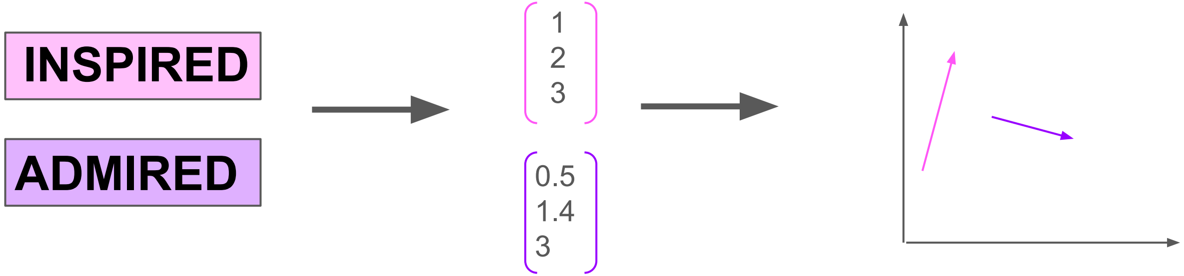


https://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/


Output layer
$y_{1,j}$
$\mathbf{W'}_{N\times V}$
Input layer
Hidden layer
x_k
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W'}_{N\times V}$
$y_{2,j}$
N-dim
V-dim
$\mathbf{W'}_{N\times V}$
$y_{C,j}$
Skip-Gram
C×V-dim


Input layer
x_{1k}
$\mathbf{W}_{V\times N}$
Hidden layer
Output layer
x_{2k}
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W'}_{N\times V}$
y_j
N-dim
V-dim
$\mathbf{W}_{V\times N}$
x_{Ck}
CBOW
C×V-dim

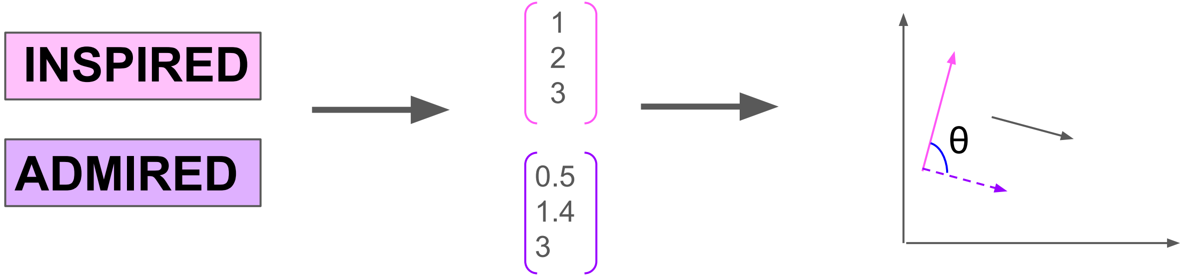
Learning Word Embedding



Given two vectors, A and B, each with n components, the similarity between them is computed as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||_2 ||\mathbf{B}||_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Learning Word Embedding



Given two vectors, A and B, each with n components, the similarity between them is computed as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

How to improve word2vec?🤓

- Negative Sampling
- Hierarchical Softmax

Hierarchical Softmax

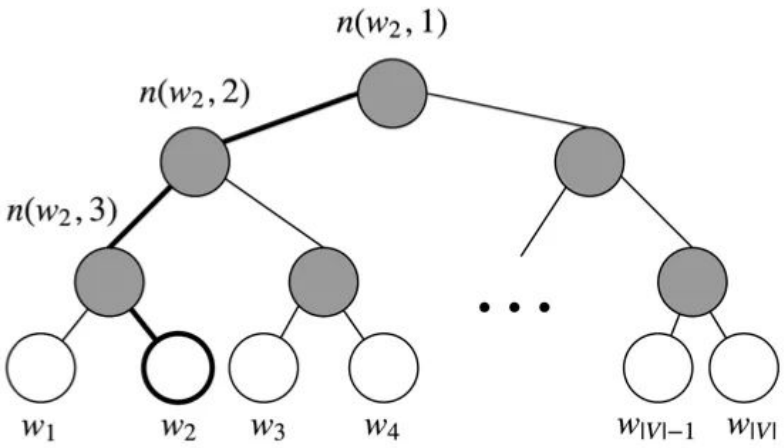
For a large dictionary, calculating the denominator requires the sum of all the words

$$P(w_o|w_c) = \frac{\exp(\mathbf{v}_o \cdot \mathbf{v}_c)}{\sum_{w=1}^{W} \exp(\mathbf{v}_w \cdot \mathbf{v}_c)}$$

Hierarchical Softmax

For a large dictionary, calculating the denominator requires the sum of all the words

$$P(w_o|w_c) = \frac{\exp(\mathbf{v}_o \cdot \mathbf{v}_c)}{\sum_{w=1}^{W} \exp(\mathbf{v}_w \cdot \mathbf{v}_c)}$$



$$P(n_i \mid w_I) = \sigma(\mathbf{v}_{n_i} \cdot \mathbf{v}_{w_I})$$

$$P(w_O \mid w_I) = \prod_{i=1}^{L(w_O)} P(n_i \mid w_I)$$

$$\boxed{-\log P(w_O \mid w_I) = -\sum_{i=1}^{L(w_O)} \log P(n_i \mid w_I)}$$

reduces the number of calculations from O(N) to O(logN)

Hierarchical Softmax

{"cat" — 5, "dog" — 9, "fish" — 12, "bird" — 3}

```
      [29]
     /    \
  "fish" [17]
         /   \
      [8]    "dog"
     /   \
 "bird"  "cat"
```

1. **Each word is assigned a node with a weight** equal to its frequency of occurrence
2. The two nodes with the lowest weights are **combined into a new internal node**
3. Repeat

Hierarchical Softmax

{"cat" — 5, "dog" — 9, "fish" — 12, "bird" — 3}

```
      [29]
     /    \
"fish"  [17]
        /   \
     [8]    "dog"
    /   \
"bird"  "cat"
```

1. **Each word is assigned a node with a weight** equal to its frequency of occurrence
2. The two nodes with the lowest weights are **combined into a new internal node**
3. Repeat

Hierarchical Softmax

{"cat" — 5, "dog" — 9, "fish" — 12, "bird" — 3}

```
       [29]
      /    \
  "fish" [17]
          /  \
       [8]    "dog"
      /   \
  "bird"  "cat"
```

1. **Each word is assigned a node with a weight** equal to its frequency of occurrence
2. The two nodes with the lowest weights are **combined into a new internal node**
3. Repeat

Negative Sampling

With negative sampling, we are instead going to randomly select just a small number of “negative” words to update the weights for. (In this context, a “negative” word is one for which we want the network to output a 0 for).

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c \cdot v_w)$$

Suitable noise distribution defined as:

$$(w, c) \sim p_{words}(w) \frac{p_{contexts}(c)^{3/4}}{Z}$$

constant

$$p_{context}(x) = p_{words}(x) = \frac{count(x)}{|Text|}$$

https://arxiv.org/pdf/1402.3722

Negative Sampling

4 Why does this produce good word representations?

Good question. We don't really know.

GloVe …because the global corpus statistics are captured directly by the model.

Matrix Factorization + Window-Based Methods

$$X_i = \sum_k X_{ik} \qquad P_{ij} = P(j|i) = X_{ij}/X_i \qquad F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

https://nlp.stanford.edu/pubs/glove.pdf

GloVe

$$P(j|i) = \frac{X_{ij}}{X_i} \rightarrow \log\left(\frac{P(j|i)}{P(k|i)}\right) \approx w_j^T \cdot (w_i - w_k)$$

$$\boxed{\log(X_{ij}) \approx w_i^T \cdot \tilde{w}_j + b_i + \tilde{b}_j}$$

$$J = \sum_{i,j} \boxed{f(X_{ij})} \left(\boxed{w_i^T \cdot \tilde{w}_j + b_i + \tilde{b}_j} - \log(X_{ij}) \right)^2$$

$$\boxed{f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{X_{\max}}\right)^{\alpha} & \text{если } X_{ij} \leq X_{\max} \\ 1 & \text{если } X_{ij} > X_{\max} \end{cases}}$$

FastText

<ap, app, ppl, ple, le> **N-gram**

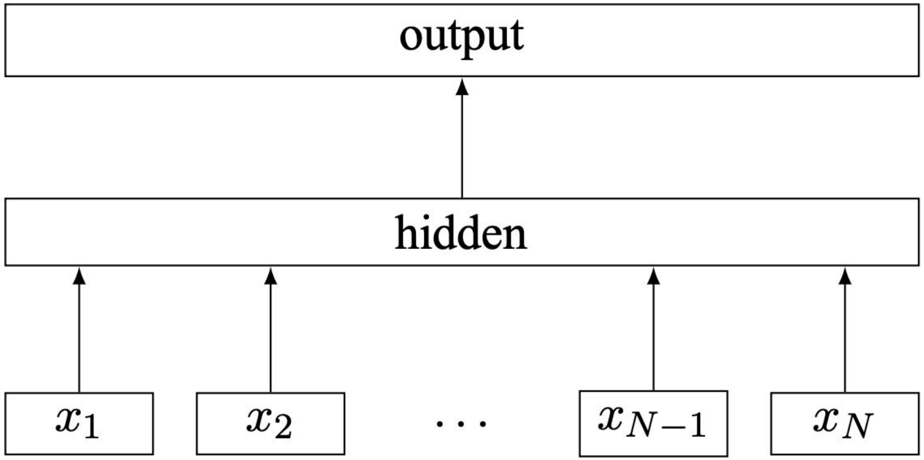


Figure 1: Model architecture of `fastText` for a sentence with N ngram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

fastText

https://arxiv.org/pdf/1607.01759

One-hot encoding

```
import numpy as np

corpus = [
    "The quick brown fox jumped over the lazy dog.",
    "She sells seashells by the seashore.",
    "Peter Piper picked a peck of pickled peppers."
]

# Create a set of unique words in the corpus
unique_words = set()
for sentence in corpus:
    for word in sentence.split():
        unique_words.add(word.lower())
```

One-hot encoding

```
# Create a dictionary to map each
# unique word to an index
word_to_index = {}
for i, word in enumerate(unique_words):
    word_to_index[word] = i

# Create one-hot encoded vectors for
# each word in the corpus
one_hot_vectors = []
for sentence in corpus:
    sentence_vectors = []
    for word in sentence.split():
        vector = np.zeros(len(unique_words))
        vector[word_to_index[word.lower()]] = 1
        sentence_vectors.append(vector)
    one_hot_vectors.append(sentence_vectors)
```

One-hot encoding

```
for vector in one_hot_vectors[0]:
    print(vector)
```

```
One-hot encoded vectors for the first sentence:
[0. 0. 0. 0. 0. 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. 1. 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.]
[0. 0. 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. 0. 0. 0. 1. 0. 0. 0. 0. 0. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 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. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```