WORD EMBEDDINGS

What You Need to Know

Hauke Licht and Lisa Wierer

荃者所以在鱼，得鱼而忘荃 Nets are for fish;
Once you get the fish, you can forget the net.

言者所以在意，得意而忘言 Words are for meaning;
Once you get the meaning, you can forget the words

庄子(Zhuangzi), Chapter 26

https://web.stanford.edu/~jurafsky/slp3/6.pdf

"The meaning of a word is its USE in the language.

(…)

One cannot guess how a word functions. One has to look at its use, and learn from that."

– Ludwig Wittgenstein, Philosophical Investigations

HARD FACTS ABOUT WORD EMBEDDINGS

Purpose: Word embeddings accurately estimate the **semantic proximity** of words.

Word Vectors: These models create **vectors** that represent word meanings, with **similar words** having similar vectors.

Vector Length: Word vectors typically range from 50 to 300 dimensions ($k = 50 \le k \le 300$).

Efficiency: Embeddings convert large, sparse matrices ($n*m$) into dense matrices ($k*m$), improving performance.

Linear Relationships: king - man ≈ queen - woman

DISTRIBUTIONAL HYPOTHESIS

Harris (1954) stated that “co-occurrence of words offers complete description of language without considering its *historical* or *psychological* aspects”.

- We do not need to treat words as symbols to understand them
- Words do not co-occur randomly
    - For a word, surrounding words are their environment (context)
    - Word cooccur in a certain environment because of semantic necessity
    - Words are synonyms if they occur in the same environment

DISTRIBUTIONAL SEMANTICS

Words with the same context have the same meaning according to the distributional hypothesis.

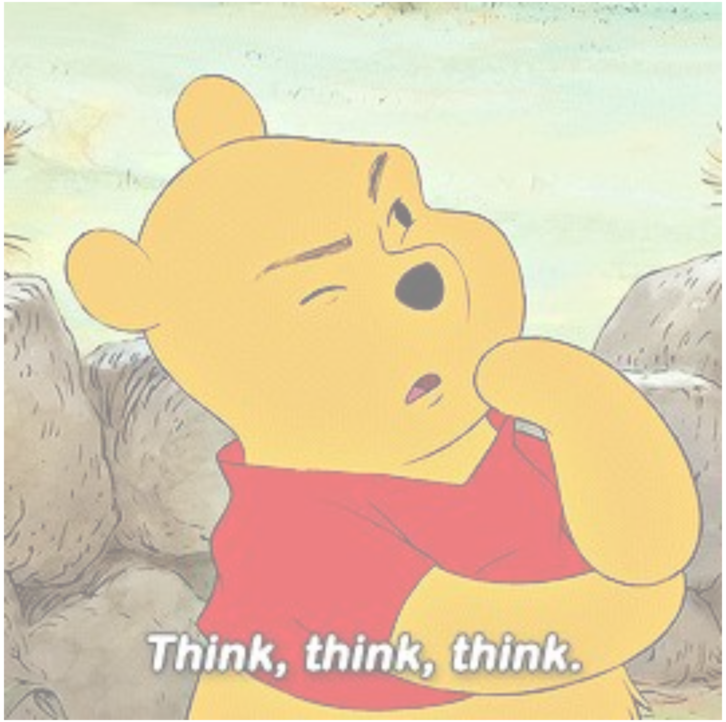
- Syntagmatic associates
    - Words that neighbor to each other (first order collocations)
        - e.g. “I will go to see a lawyer to seek legal advice”
        - “lawyer” and “legal” are words in the same topic
        - e.g. “I met a bad doctor yesterday”
        - “bad” is modifier of “doctor”
- Paradigmatic parallels
    - Words that have similar neighbors (second order collocations)
        - e.g. “I will go to see a lawyer/barrister to seek legal advice”
        - e.g. “I met a bad/terrible doctor yesterday”

How to learn a word's meaning

What does the word **tezgüino** mean?

Examples how it's used in a sentences:

1. A bottle of **tezgüino** is on the table.
2. Everyone likes **tezgüino**.
3. **Tezgüino** makes you drunk.
4. We make **tezgüino** out of corn.



How to learn a word's meaning

What does the word **tezgüino** mean?

Examples how it's used in a sentences:

1. A bottle of **tezgüino** is on the table.
2. Everyone likes **tezgüino**.
3. **Tezgüino** makes you drunk.
4. We make **tezgüino** out of corn.

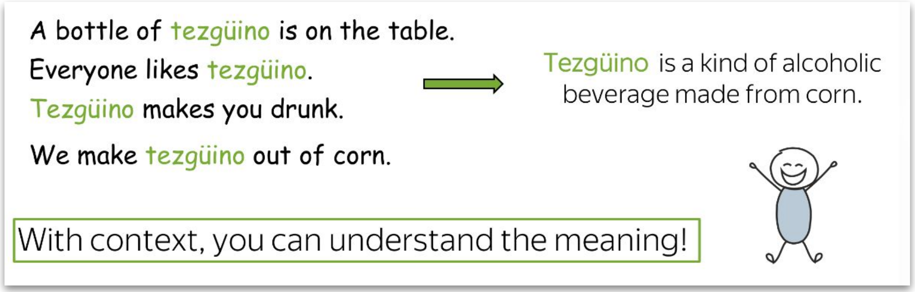
Implication

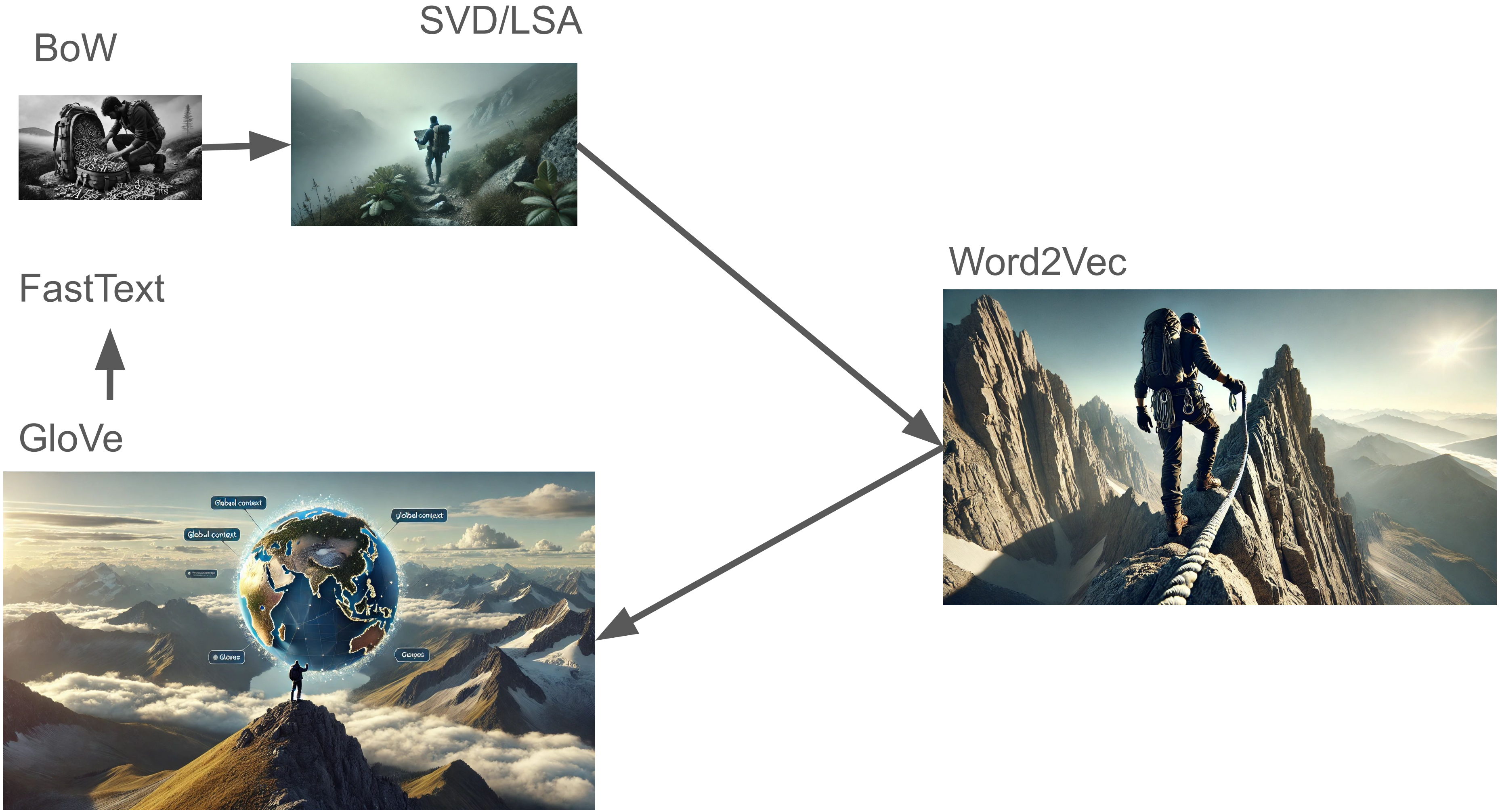
- Words with **similar meaning** appear in **similar context** (word windows or sentences)
- To **capture a word's meaning** with numbers, its *numeric representation* should summarize in which word contexts it occurs

Word embedding methods

Intuition

- co-occurrence patterns and/or word context information summarizes a word's meaning and functions
- embedding methods condense these distributional patterns into low-dimensional vectors




BoW
SVD/LSA
Word2Vec
FastText
GloVe
Global context

Word2Vec

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

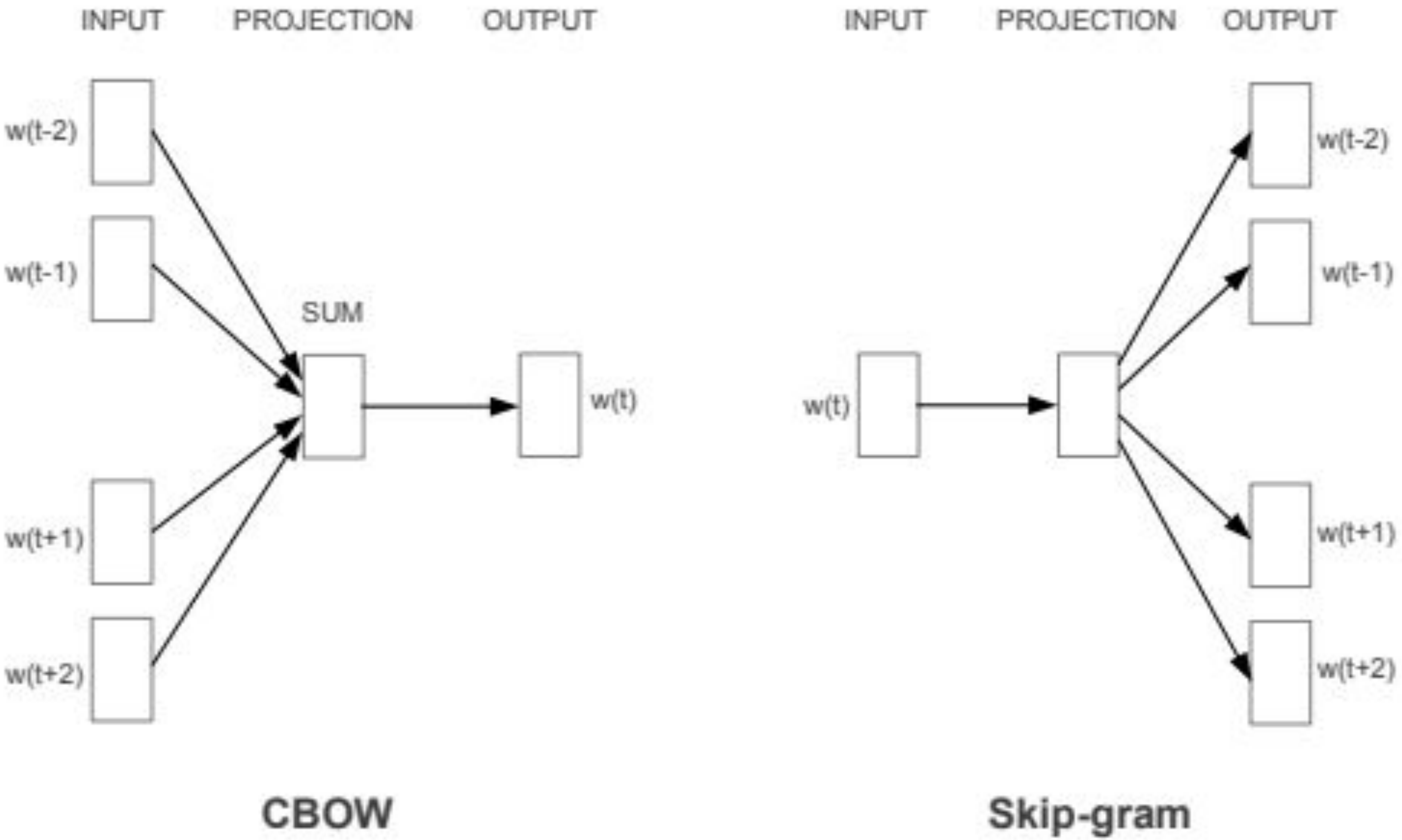
Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.


INPUT
PROJECTION
OUTPUT
w(t-2)
w(t-1)
SUM
w(t)
w(t+1)
w(t+2)
CBOW
INPUT
PROJECTION
OUTPUT
w(t)
w(t-2)
w(t-1)
w(t+1)
w(t+2)
Skip-gram

Skip-gram:

- **Focus**: Predicts the surrounding context words given a target word.
- **Strength**:
  - Better for learning representations of **rare words** because it trains on each target word's context individually.
  - Performs well when you want detailed, high-quality word embeddings for smaller datasets or infrequent words.
  - Captures more precise semantic relationships between words.

USED MORE OFTEN

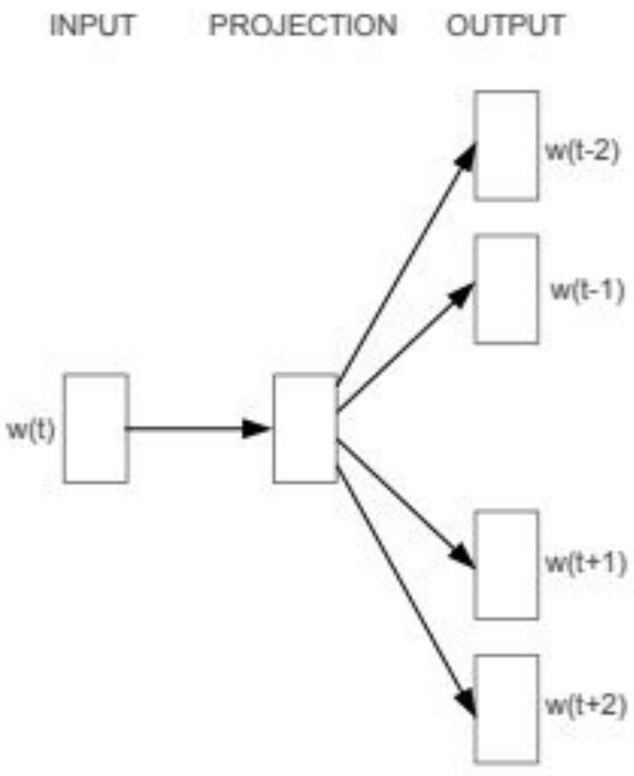
CBOW:

- **Focus**: Predicts a target word based on the surrounding context.
- **Strength**:
  - Faster and more computationally efficient because it averages context word embeddings.
  - Better for dealing with **larger datasets** and common words.

Word2Vec Skip-Gram

Implication of *fill-the-blank* example

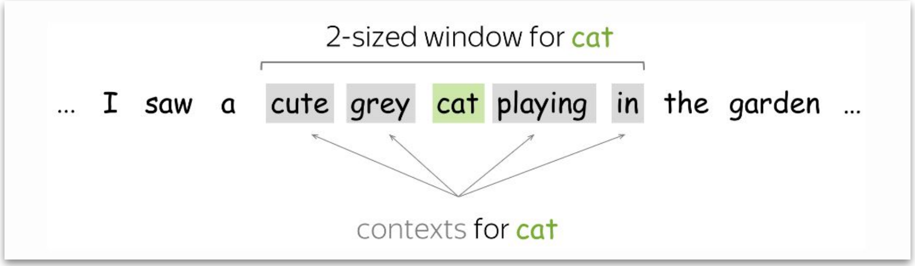
> *To **capture a word's meaning** with numbers, its* numeric representation *should summarize in which word contexts it occurs.*



Skip-gram

Approach

Use a word's embedding to predict which words occur in its context
⇒ *incentive* to learn about its usage



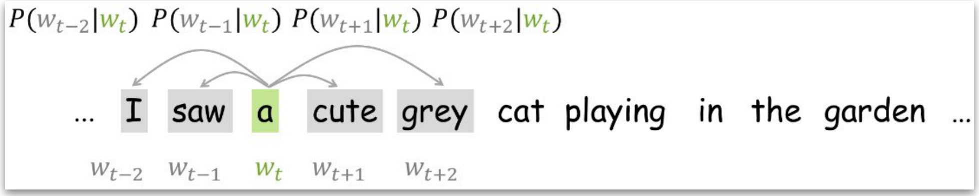
Notation

- **focus word**: word whose meaning we want to capture
- **context word**: word occurring in a window around the focus word's

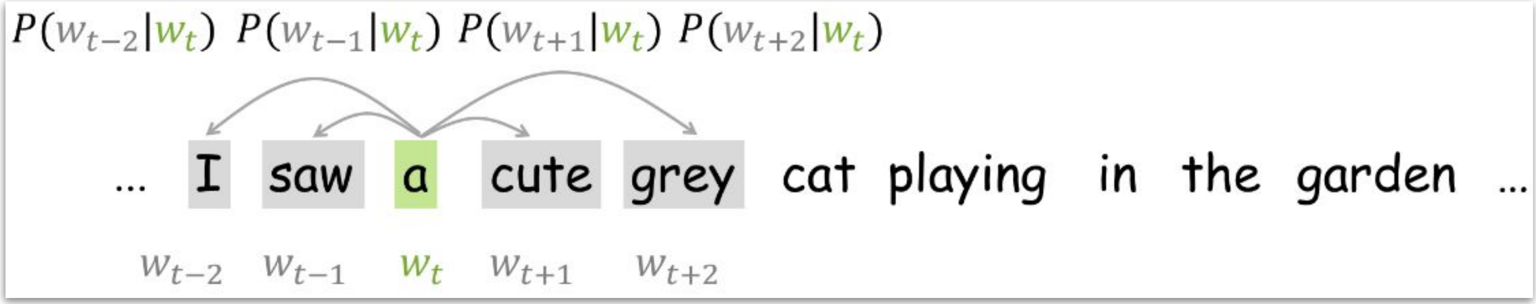
The skip-gram algorithm

Use a word's embedding to predict which words occur in its context

- w_i is the word *i*'s embedding
- ***t*** indicates a word's position relative to the focus word
- *P*(*a* | *b*) is the ***conditional probability*** that *a* occurs given *b*

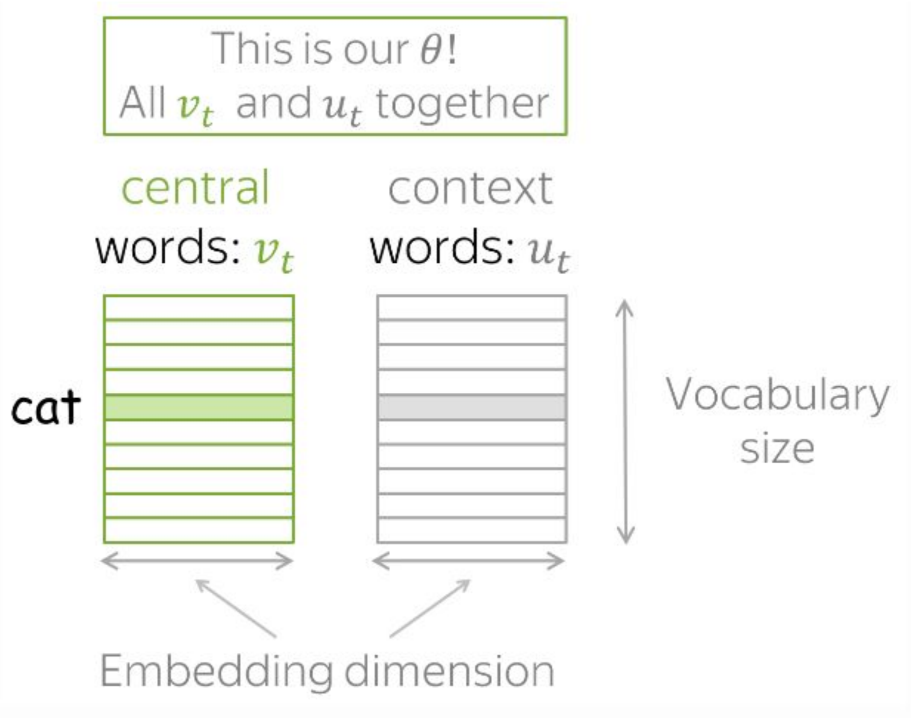


The skip-gram algorithm



- w_i is the word *i*'s embedding
- *t* indicates a word's position relative to the focus word
- *P*(*a* | *b*) is the ***conditional probability*** that *a* occurs given *b*

using different vectors depending on whether the word is a focus or context word



Why Two Different Sets of Vectors?

1. **Input vector (target)**: Represents the word when it is treated as the focus word. This vector captures what the word **means** in different contexts.
2. **Output vector (context)**: Represents the word when it is part of the context for another word. This vector captures how the word **appears in relation** to other words.

The skip-gram algorithm

Overall, we want to ***maximize***

focus word

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \ne 0} P(w_{t+j} | w_t, \theta)$$

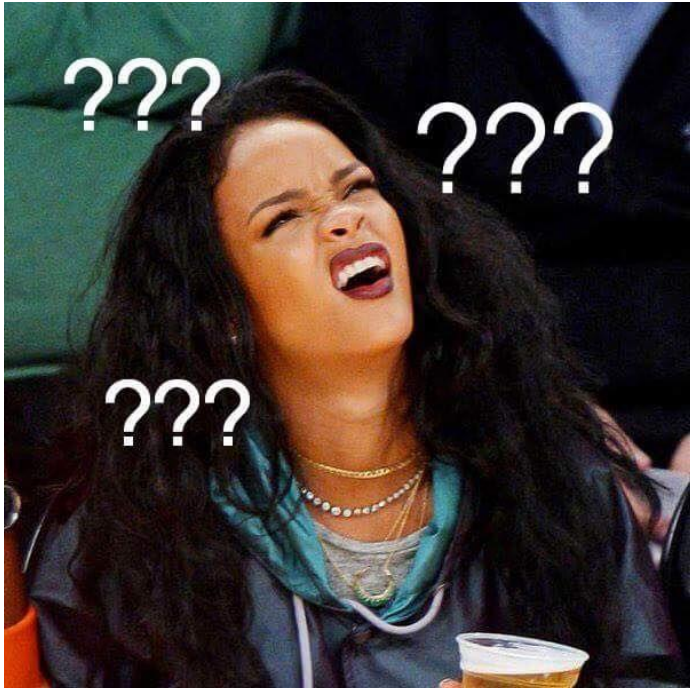
positions in the text corpus

relative position to focus word

the “embeddings” we optimize

One step at a time

How do we get P(context word|focus word)?



INITIALIZE EMBEDDINGS RANDOMLY for all words (two vectors per word: focus and context)

CALCULATE SIMILARITY between embedding vector of focus and context word = PREDICTION OF CONTEXT WORDS.

CALCULATE ERROR: how well the current embeddings predict the actual context words compared to what it should have predicted

UPDATE INPUT VECTOR to better predict true context word.


OH
REALLY?!?

INITIALIZE EMBEDDINGS RANDOMLY for all words (two vectors per word: focus and context)

CALCULATE SIMILARITY between embedding vector of focus and context word = PREDICTION OF CONTEXT WORDS.

CALCULATE ERROR: how well the current embeddings predict the actual context words compared to what it should have predicted

UPDATE INPUT VECTOR to better predict the context word.

TOO MUCH WORK

SHORTCUT = NEGATIVE SAMPLING

The skip-gram algorithm

focus	***context***	***label***
cute	saw	“hit”
cute	a	“hit”
cute	grey	“hit”
cute	cat	“hit”

Solution ⇒ “negative sampling”

1. take actual target words (label = “hit”)

The skip-gram algorithm

focus	***context***	***label***
cute	saw	“hit”
cute	a	“hit”
cute	grey	“hit”
cute	cat	“hit”
cute	do	“miss”
cute	melon	“miss”
cute	tezgüino	“miss”
…	…	…

Solution ⇒ “negative sampling”

1. take actual target words (label = “hit”)
2. take a random sample of words from the vocabulary (label = “miss”)

The skip-gram algorithm

focus	*context*	*label*	*sim*
cute	saw	“hit”	0.23
cute	a	“hit”	0.41
cute	grey	“hit”	0.33
cute	cat	“hit”	0.68
cute	do	“miss”	0.10
cute	melon	“miss”	-0.12
cute	tezgüino	“miss”	-0.43
…	…	…	

Solution ⇒ “negative sampling”

1. take actual target words (label = “hit”)
2. take a random sample of words from the vocabulary (label = “miss”)
3. compute $\mathbf{sim}(w_c, w_v)$ for “hits” and “misses” with focus word

The skip-gram algorithm

focus	*context*	*label*	*sim*
cute	saw	“hit”	0.23
cute	a	“hit”	0.41
cute	grey	“hit”	0.33
cute	cat	“hit”	0.68
cute	do	“miss”	0.10
cute	melon	“miss”	-0.12
cute	tezgüino	“miss”	-0.43
…	…	…	

Solution ⇒ “negative sampling”

1. take actual target words (label = “hit”)
2. take a random sample of words from the vocabulary (label = “miss”)
3. compute $\mathbf{sim}(w_c, w_v)$ for “hits” and “misses” with focus word
4. **classify** if context word is “hit” or “miss”

The skip-gram algorithm

focus	***context***	***label***	***sim***
cute	saw	“hit”	☝️ 0.23
cute	a	“hit”	☝️ 0.41
cute	grey	“hit”	☝️ 0.33
cute	cat	“hit”	☝️ 0.68
cute	do	“miss”	👇 0.10
cute	melon	“miss”	👇 -0.12
cute	tezgüino	“miss”	👇 -0.43
…	…	…	

Solution ⇒ “negative sampling”

1. take actual target words (“hits”)
2. take a random sample of words from the vocabulary (“misses”)
3. compute $\mathbf{sim}(w_c, w_v)$ for “hits” and “misses” with focus word
4. classify if context word is “hit” or “miss”
5. **update** model parameters $\boldsymbol{\theta}$ (our word embeddings) to
   - increase probability of “hits” ☝️
   - decrease probability of “misses” 👇

   using “**gradient descent**”

The skip-gram algorithm

focus	***context***	***label***	***sim***
cute	saw	“hit”	0.23
cute	a	“hit”	0.41
cute	grey	“hit”	0.33
cute	cat	“hit”	0.68
cute	do	“miss”	0.10
cute	a	“miss”	0.41
cute	tezgüino	“miss”	-0.43
…	…	…	

Solution ⇒ “negative sampling”

1. take actual target words (“hits”)
2. take a random sample of words from the vocabulary (“misses”)
   tiny chance that this random sampling of negative words causes “**label noise**”
   ⇒ but inconsequential overall ✔️

Hands-on → Word2Vec Tasks

- Word Similarity
- Finding most similar terms
- Centroids
- Analogy problem

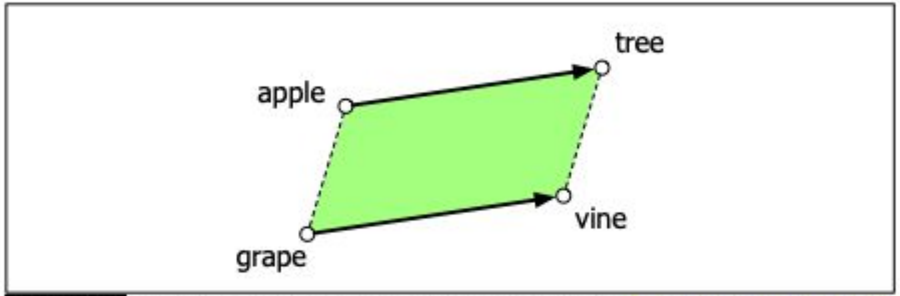


Figure 6.15 The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of $\vec{vine}$ can be found by subtracting $\vec{apple}$ from $\vec{tree}$ and adding $\vec{grape}$.

GloVe

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
jpennin@stanford.edu, richard@socher.org, manning@stanford.edu

Abstract

Recent methods for learning vector space representations of words have succeeded in capturing fine-grained semantic and syntactic regularities using vector arithmetic, but the origin of these regularities has remained opaque. We analyze and make explicit the model properties needed for such regularities to emerge in word vectors. The result is a new global log-bilinear regression model that combines the advantages of the two major model families in the literature: global matrix factorization and local context window methods. Our model efficiently leverages

the finer structure of the word vector space by examining not the scalar distance between word vectors, but rather their various dimensions of difference. For example, the analogy "king is to queen as man is to woman" should be encoded in the vector space by the vector equation *king* – *queen* = *man* – *woman*. This evaluation scheme favors models that produce dimensions of meaning, thereby capturing the multi-clustering idea of distributed representations (Bengio, 2009).

The two main model families for learning word vectors are: 1) global matrix factorization methods, such as latent semantic analysis (LSA) (Deerwester et al., 1990) and 2) local context window methods, such as the skip-gram model of Mikolov

FIRST: OBSERVE CO-OCCURENCE OF WORDS

The model learns by looking at how often words appear near each other in a sentence. For example, the word "dog" might frequently appear near words like "bark" or "pet."

OBSERVE

SECOND: PREDICTING NEIGHBORS

The model focuses on predicting a word's neighbors (context) based on the word itself. So, it tries to learn what words are likely to show up next to the word "dog" by seeing many examples of sentences with that word.

PREDICT

Objective: Predict Ratios of Co-occurrences:

- The GloVe model learns embeddings by focusing on the **ratios** of co-occurrence probabilities rather than directly predicting context words.
- Specifically, it assumes that the relationship between words can be modelled based on the **ratio of their co-occurrence probabilities**:

$$P(j|i) = \frac{X_{ij}}{\sum_k X_{ik}}$$

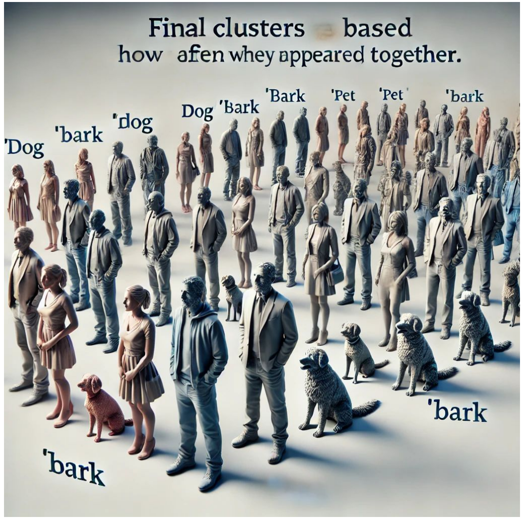
  - Where P(j|i) is the probability that word j appears in the context of word i.
  - Where X_ij is the co-occurence of the word i and j and X_ik is the co-occurence of the word i and k.

Why Ratios?

- The idea is that **word pairs** that have similar meanings (e.g., "ice" and "snow") should have similar ratios of co-occurrences with a third word (e.g., "cold"), compared to word pairs with different meanings (e.g., "ice" and "steam").

SECOND: OPTIMIZATION SO THAT SIMILAR WORDS HAVE SIMILAR VECTORS

Word2Vec defines a goal (cost function) that helps the model learn. It tries to adjust the word representations (called vectors) so that similar words end up with similar vectors. The cost function tells the model how wrong its predictions are, and the model adjusts its learning to make better predictions.



OPTIMIZE

LOGARITHMIC MODEL of the co-occurences

$$w_i^T w_j + b_i + b_j = \log(X_{ij})$$

- w_i and w_j are the word vectors (embeddings) for words i and j.
- b_i and b_j are bias terms for each word.
- X_ij is the co-occurrence count for word i and word j.

MINIMIZE cost function

$$J = \sum_{i,j} f(X_{ij}) \left(w_i^T w_j + b_i + b_j - \log(X_{ij})\right)^2$$

f(X_ij) is a weighting function to reduce the importance of very frequent co-occurrences and handle rare words

Word Vectors as Output:

- Once training is complete, we end up with two sets of word vectors:
  - **Word vectors** (focus word embeddings).
  - **Context vectors** (context word embeddings).
- GloVe typically combines these two vectors into one final embedding for each word by averaging or summing them
- These final word vectors capture both local context and global co-occurrence information.

FastText

"apple"⇒{<ap,app,ppl,ple,le>}

"apple"⇒{<ap,app,ppl,ple,le>}

Embedding = Sum of embeddings of n-grams

"apple"⇒{<ap,app,ppl,ple,le>}

trains similar to Skip & CBOW

Embedding = Sum of embeddings of n-grams

NOW

YOU HAVE TO MAKE SOME DECISIONS

CONTEXT SIZE/WINDOW SIZE

1, 2, 3, 4, 5, 6, …, 12, …, 24, …, 48, …

MISS SUBTLE/DISTANT RELATIONSHIPS

NOISE

“CAT” & “SAT”
“KING” & “QUEEN”

“GOVERNMENT” & “POLICY”
“PRESIDENT” & “POWER”

EMBEDDING DIMENSIONS

20, …, 50, …, 100, …, 200, …, 300, …, 450, …

STRIP MEANING AND SUBTLENESS? NOISE?

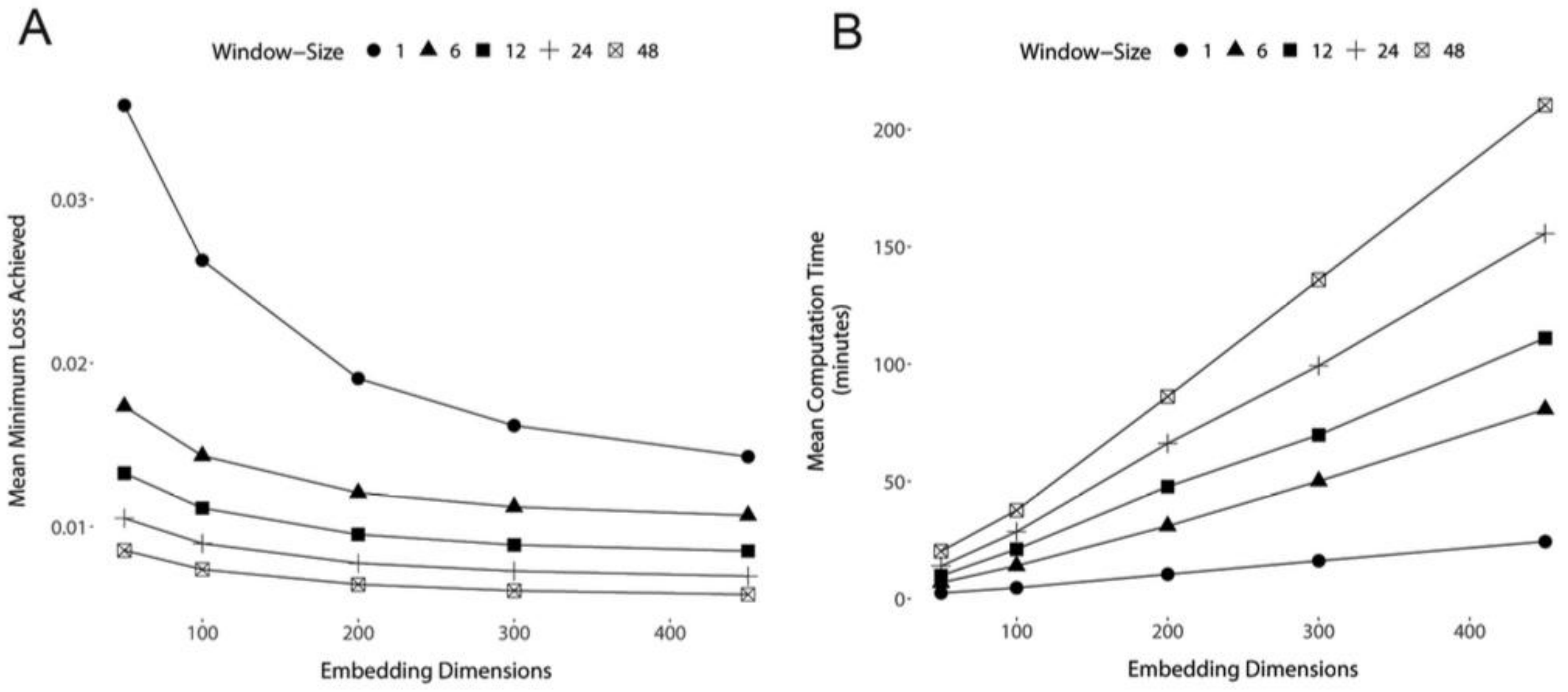


Figure 4. Technical criteria: larger models fit better but take longer to compute. *A*, Mean minimum loss achieved. *B*, Computation time (minutes)

Rodriguez and Spirling (2021)

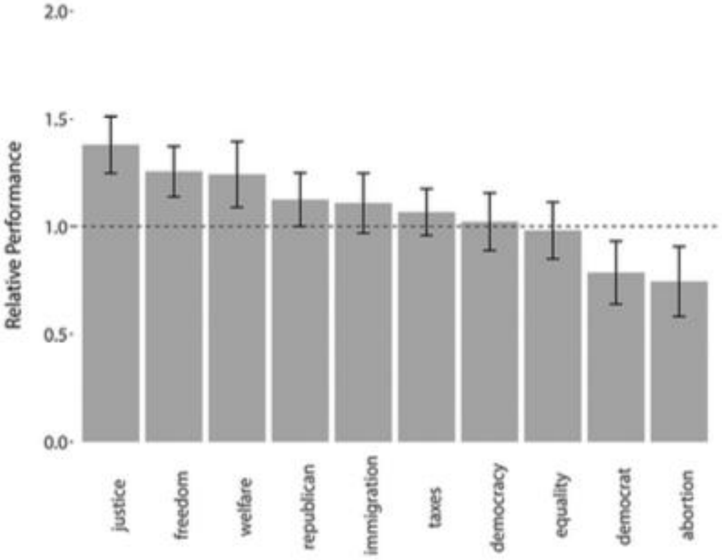
OFF-THE SHELF vs CUSTOM MODELS

CUSTOM MODEL .... OFF-THE SHELF

TYPOS AND SMALL SIZE LOWER MEANING

BIAS?
TRUTHFULNESSS?

C Glove over local



D GloVe over Human

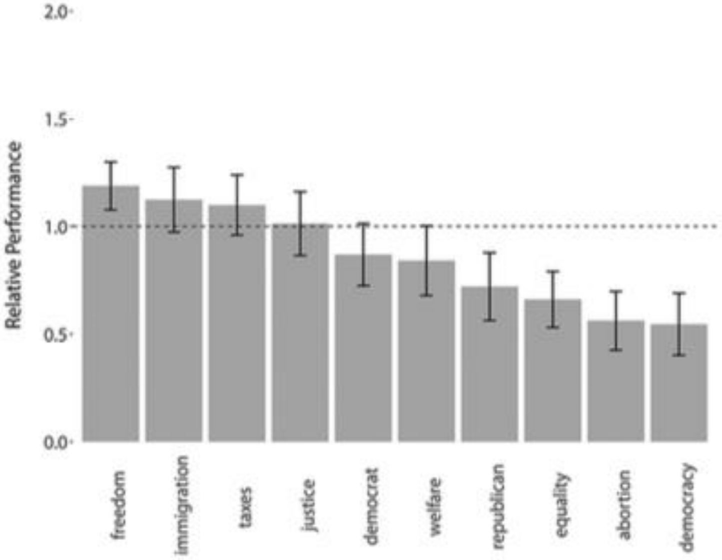
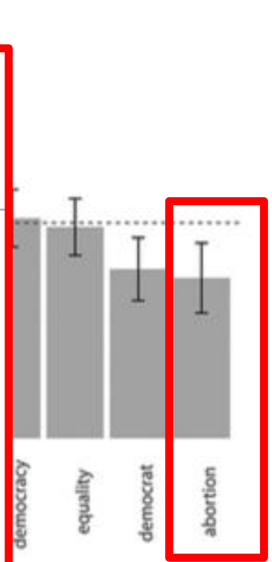


Figure 2. Human preferences: Turing assessment. *A*, Candidate: local 48-300; baseline: local 6-300. *B*, Candidate: local 6-300; baseline: human. *C*, Candidate: GloVe; baseline: local 6-300. *D*, Candidate: GloVe; baseline: human.

Rodriguez and Spirling (2021)

C Glove over local



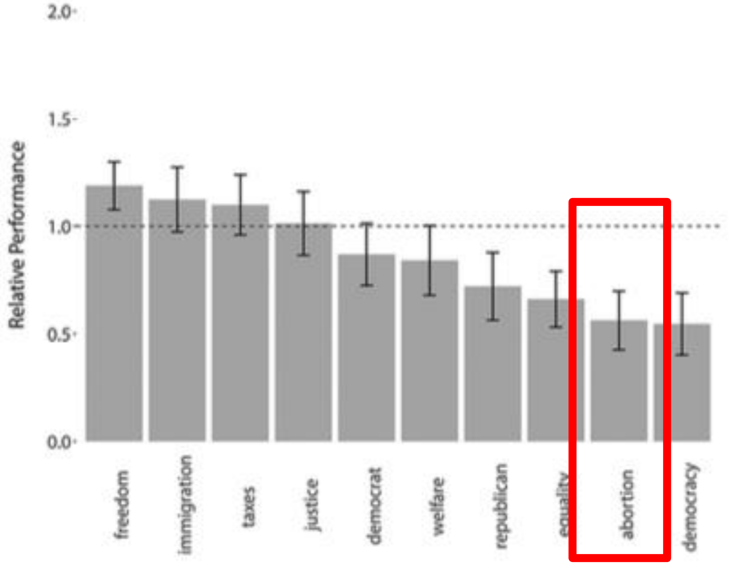
"abortion"

pretrained	local
abortions	abortions
contraception	partialbirth
euthanasia	lateterm
antiabortion	procedure
legalized	ban
homosexuality	partial
opposes	clincs
prolife	sterilization
pregnancy	clinic
advocates	birth

Note: First two groups: words ("abortion," "democr
model ones (left columns) for GloVe. Last group ("

BIAS???

D GloVe over Human



Figu... Candidate: local 48-300; baseline: local 6-300. *B*, Candidate: local 6-300; baseline: human.
C, C... te: GloVe; baseline: human.

Rodriguez and Spirling (2021)

Pretrained models

Google News (Word2Vec)

- **Description**: Word2Vec is one of the most popular word embedding models. It creates embeddings based on context using either **Skip-gram** or **CBOW** (Continuous Bag of Words) models.
- **Trained on**: Google News corpus (100 billion words).
- **Vector size**: 300 dimensions.

```python
import gensim.downloader as api

# Load pre-trained Word2Vec embeddings from Google News
model = api.load("word2vec-google-news-300")
vector = model['king']  # Example: Get vector for 'king'
```

Pretrained models

Common Crawl, Wikipedia + Gigaword (GloVe)

Description: GloVe is another widely-used embedding model developed by Stanford. It captures word co-occurrence in a global context, and like Word2Vec, represents words as vectors in a continuous space.

Trained on:

- Common Crawl (840 billion tokens)
- Wikipedia + Gigaword (6 billion tokens)

Vector size: 50, 100, 200, 300 dimensions.

```python
import numpy as np

# Load GloVe embeddings (download and extract first)
def load_glove_model(file_path):
    glove_model = {}
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            split_line = line.split()
            word = split_line[0]
            embedding = np.array(split_line[1:], dtype='float32')
            glove_model[word] = embedding
    return glove_model

glove_model = load_glove_model('glove.6B.300d.txt')
vector = glove_model['king']  # Example: Get vector for 'king'
```

Pretrained models

Wikipedia, Common Crawl (FastText)

Description: FastText is an extension of Word2Vec that takes subword information into account, making it robust for rare words and capable of generating embeddings for out-of-vocabulary words by averaging subword embeddings.
Trained on:

- Wikipedia
- Common Crawl

Vector size: 300 dimensions.

```
import fasttext.util

# Download and load pre-trained FastText embeddings
fasttext.util.download_model('en', if_exists='ignore')  # English embeddings
model = fasttext.load_model('cc.en.300.bin')
vector = model.get_word_vector('king')  # Example: Get vector for 'king'
```

LET’S EXPLORE THIS…

LIMITATIONS OF EMBEDDINGS

WRITE DOWN AT LEAST 3 WINS OF STATIC WORD EMBEDDINGS OVER BoW and 3 LIMITATIONS OF STATIC WORD EMBEDDINGS (Be creative)

OUT-OF VOCAB WORDS ARE IGNORED

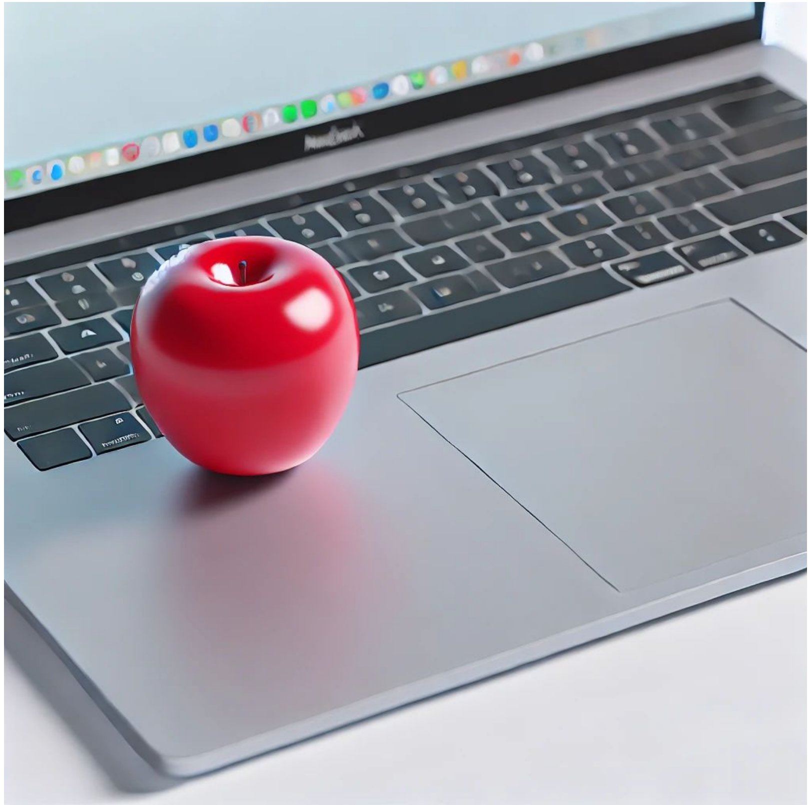
OOV-WORDS:

- **Slang** (e.g., "selfie" or "lit").
- **Domain-specific terms** (e.g., technical jargon, medical terms).
- **Proper nouns** (e.g., new names of people, places, or brands).
- **Typos or misspellings**.

NO VECTOR REPRESENTATION

FastText uses substructures of words to mitigate this problem, but still does not capture new terms accuratly

UNABLE TO HANDLE POLYSEMY



TRANSFORMERS