Google Developer Groups

# Journal Club #1

Word2Vec: Efficient Estimation of Word Representations in Vector Space

## Why Word Vectorization?



How to we represent words as numbers?

How do we retain meaning of these words?

And what about context-awareness?

## Key Insights in IR and NLP



Text has **no** meaning!

빨간 차를 운전합니다

... but still an informative source!

파란색 자를 운전합니다 is more similar to the above than 버스를 운전합니다.

**term frequency (TF)**
**document frequency (DF)**
**TF-IDF**

... and **not** random!

*I drive a red car* is more probable than...

I drive a red horse.
red car a drive I.
I car a red drive.

**language models**
**uni-gram, bi-gram, n-gram**

... also the meaning is defined by usage and context!

I drive a truck / I drive a car / I drive the bus...

*truck/car/bus* similar in meaning!

**statistical semantics**
**word embeddings, deep learning**

Google Developer Groups

## Former NLP systems treat words as **atomic units**

- Words are represented as indices in a vocabulary
- **No notion of similarity** between words!
- ... but simple and robust!
- simple + lots of data **>** complex + little data

## Several Strategies based on that idea!

- Bag of Words
- Term Frequency-Inverse Document Frequency (TF-IDF)
- n-Grams

Google Developer Groups

**I love cats. I love dogs.**

## Bag of Words…?

Vocabulary: [“I”, “love”, “cats”, “dogs”]
“I love cats.” -> [1, 1, 1, 0]
“I love dogs.” -> [1, 1, 0, 1]

## TF-IDF…?

TF: "I" and "love" appear twice; "cats" or "dogs" appear once
IDF: *downweights* common words
“I” appears more often than “cats”, therefore its score is lower
“I love cats.” -> [0.1, 0.1, 0.8, 0]
“I love dogs.” -> [0.1, 0.1, 0, 0.8]

## n-grams…?

Example via *bigram* -> [“I love”, “love cats”]
Count of bigrams to capture short-range dependencies

## Word Vectorization!

Google Developer Groups

The idea is actually quite **old**!

*distributed word representations learned by neural networks* - emerged early 1990s

Such neural networks learn dense, distributed vectors for words

These can capture some form of semantic similarity!

... but become **very** computationally expensive, which made them **not** feasible :(

*Word2Vec* builds on these foundations!

Training is **much more efficient**!

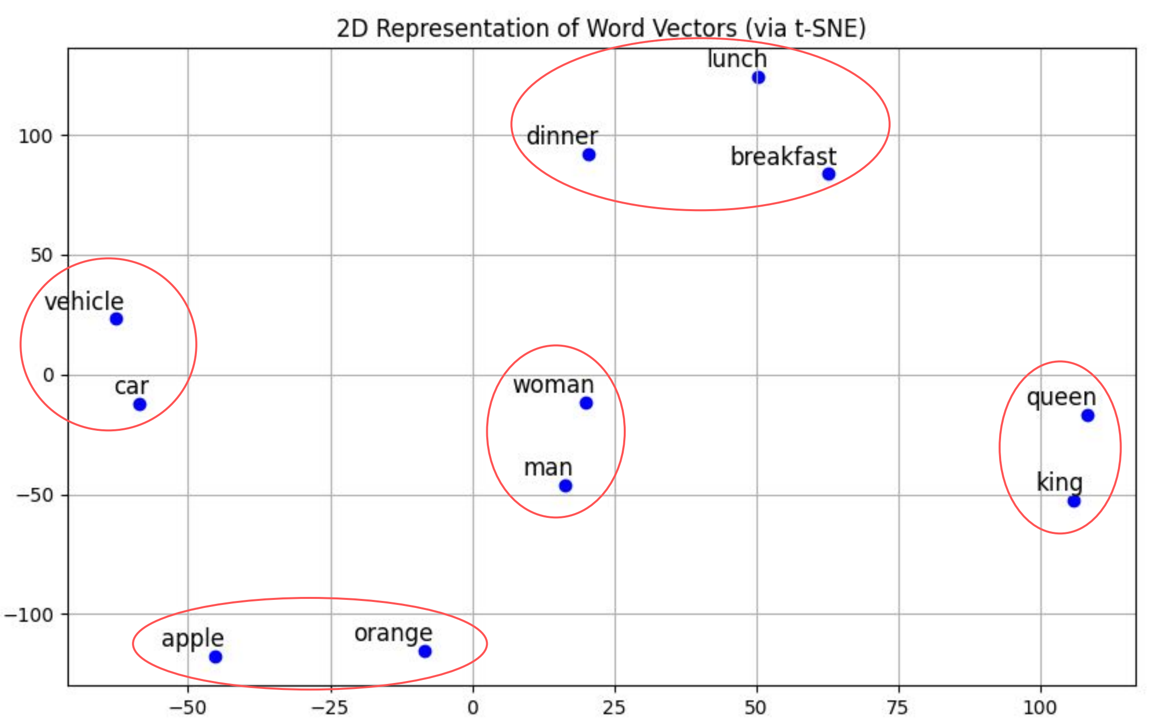
Embeddings are **general-purpose** that became widely adopted in NLP

TU WIEN DS

Motivation

# What's the rationale for word vectorization?

Google Developer Groups



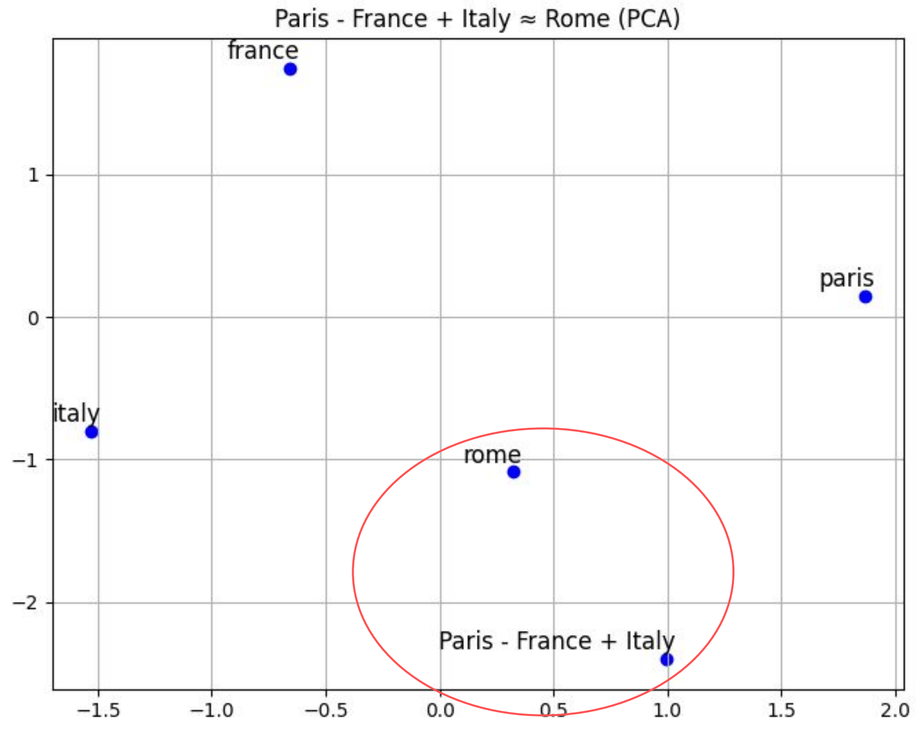

```python
import gensim.downloader as api
model = api.load("word2vec-google-news-300")

words = ['king', 'queen',
         'man', 'woman',
         'car', 'vehicle',
         'apple', 'orange',
         'breakfast', 'lunch', 'dinner']
```

Motivation

## What's the rationale for word vectorization?

Google Developer Groups




```
model = api.load("word2vec-google-news-300")
words = ['italy', 'france', 'paris', 'rome']

def get_word_vectors(words):
    vectors = [model[word] for word in words]
    return np.array(vectors)

word_vectors = get_word_vectors(words)

pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(word_vectors)

def add_vector_arithmetic(pca_model,
                          original_vectors,
                          words):
    arithmetic_result = model['paris'] -
                        model['france'] +
                        model['italy']

    result_vector = pca_model.transform(
        [arithmetic_result])[0]

    return np.vstack([reduced_vectors, result_vector]),
        words + ['Paris - France + Italy']

reduced_vectors, words = add_vector_arithmetic(pca,
                                               word_vectors,
                                               words)
```

Google Developer Groups

# Cool and all…
# but what does the paper tell us?

Word2Vec - making word vectorization accessible

## Feed Forward Neural Net Language Model (NNLM)



Remember the encoder-only GPT architecture?

... used to predict the next word in a sentence based on previous words

**This model does just the same!**

Input is a few previous words

... represented as **one-hot-encoded vectors**, according to position in the vocabulary

Projection Layer transforms words into **continuous dense vector representation**

... also has a **hidden layer** for learning complex patterns

Output is the probabilities for all words in the vocabulary

... the most likely word comes next, of course

Wayyyyy too complex for lots of vocabulary! 😔

$N \times D \times H$

## Recurrent Neural Net Language Model (RNNLM)

Google Developer Groups

Addresses some problems of the NNLM

... **no** need to specify context length

... RNNs have better ability to capture **more complex patterns**

No projection layer!

... only input, hidden and output layer

Most complexity still comes from the hidden layer

H x H

... sooo still not feasible! 😔

**And how does Word2Vec address these issues?**

TU WIEN DS

Google Developer Groups

We saw that the complexity is caused by non-linear hidden layers

... **but this makes them attractive, right?**

Just use simpler models!

... might not be able to represent data as precisely

... but trained on much more data more efficiently!

**Resulted in extremely efficient log-linear models!**

## Continuous Bag-of-Words Model

Google Developer Groups

Same architecture as NNLM, but **no hidden layer**

Projection layer is shared for all words

... instead of each word independently

... all words projected into same space, embeddings averaged

Predicts a word in dependence of **surrounding** words

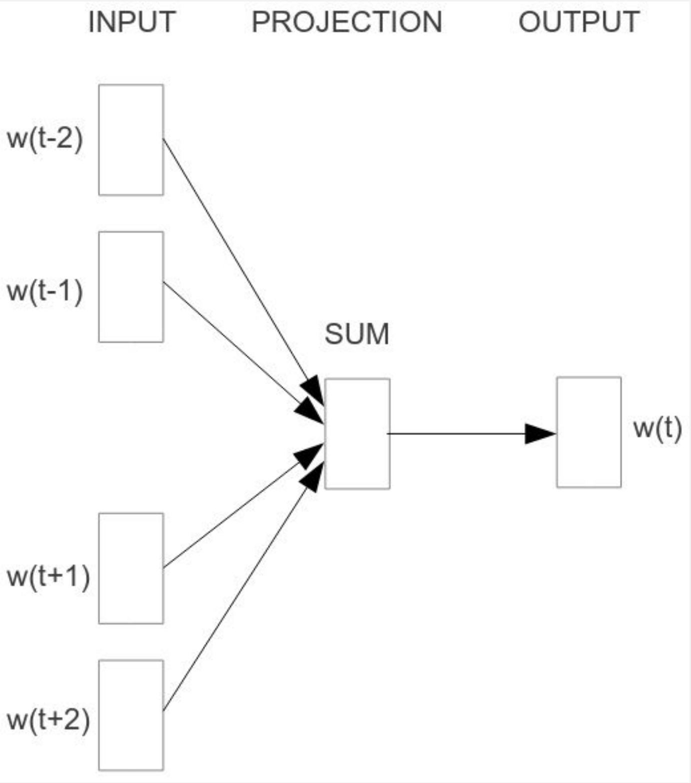
... instead of only previous words!

Order of words in the history does **not** influence projection

... therefore called Bag-of-Words!

... “continuous”, since it uses a distributed representation of the context

**Achieves a log-linear runtime!** 🤯



## Continuous Skip-Gram Model

Google Developer Groups

Similar to CBOW

... but predicts **surrounding** words based on current word

Range C is adjustable

... increased C improves quality of embeddings

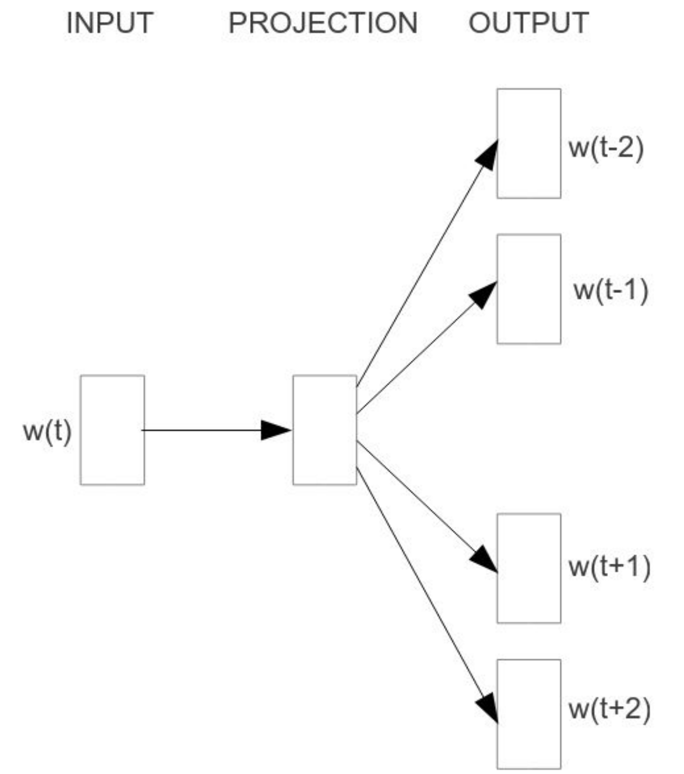
... but also increases the computational complexity 😢

Distant words are usually less related

... give less weight to the distant words

... sample less from those words in the training examples

**Also achieves a log-linear runtime!** 🥳



Google Developer Groups

Trained on the **Google News Corpus**

... **6 billion** tokens (i.e. words)
... vocabulary size restricted to **1 million** of most common words
... this helps manage the computational cost!

Experimented a lot with vector dimensionalities

... adding more dimensions/data improves accuracy
... **but** only to a certain point! 😧
... except if you **increase both simultaneously**!
... how sustainable is that anyways?! 🤨

| Model Architecture | Semantic-Syntactic Word Relationship test set | | MSR Word Relatedness Test Set [20] |
|---|---|---|---|
| | Semantic Accuracy [%] | Syntactic Accuracy [%] | |
| RNNLM | 9 | 36 | 35 |
| NNLM | 23 | 53 | 47 |
| CBOW | 24 | 64 | 61 |
| Skip-gram | 55 | 59 | 56 |

TU WIEN DS

Google Developer Groups

| Relationship | Example 1 | Example 2 | Example 3 |
|---|---|---|---|
| France - Paris | Italy: Rome | Japan: Tokyo | Florida: Tallahassee |
| big - bigger | small: larger | cold: colder | quick: quicker |
| Miami - Florida | Baltimore: Maryland | Dallas: Texas | Kona: Hawaii |
| Einstein - scientist | Messi: midfielder | Mozart: violinist | Picasso: painter |
| Sarkozy - France | Berlusconi: Italy | Merkel: Germany | Koizumi: Japan |
| copper - Cu | zinc: Zn | gold: Au | uranium: plutonium |
| Berlusconi - Silvio | Sarkozy: Nicolas | Putin: Medvedev | Obama: Barack |
| Microsoft - Windows | Google: Android | IBM: Linux | Apple: iPhone |
| Microsoft - Ballmer | Google: Yahoo | IBM: McNealy | Apple: Jobs |
| Japan - sushi | Germany: bratwurst | France: tapas | USA: pizza |

Google Developer Groups

**Testing**

How well do they perform to state-of-the-art BERT embeddings?

Word2Vec vs. BERT

## Benchmarking Dataset

Google Developer Groups

**Microsoft Research Paraphrase Corpus** (MRPC)

... is a corpus of sentence pairs
... is automatically extracted from online news sources
... check whether the sentences in the pair are **semantically equivalent**

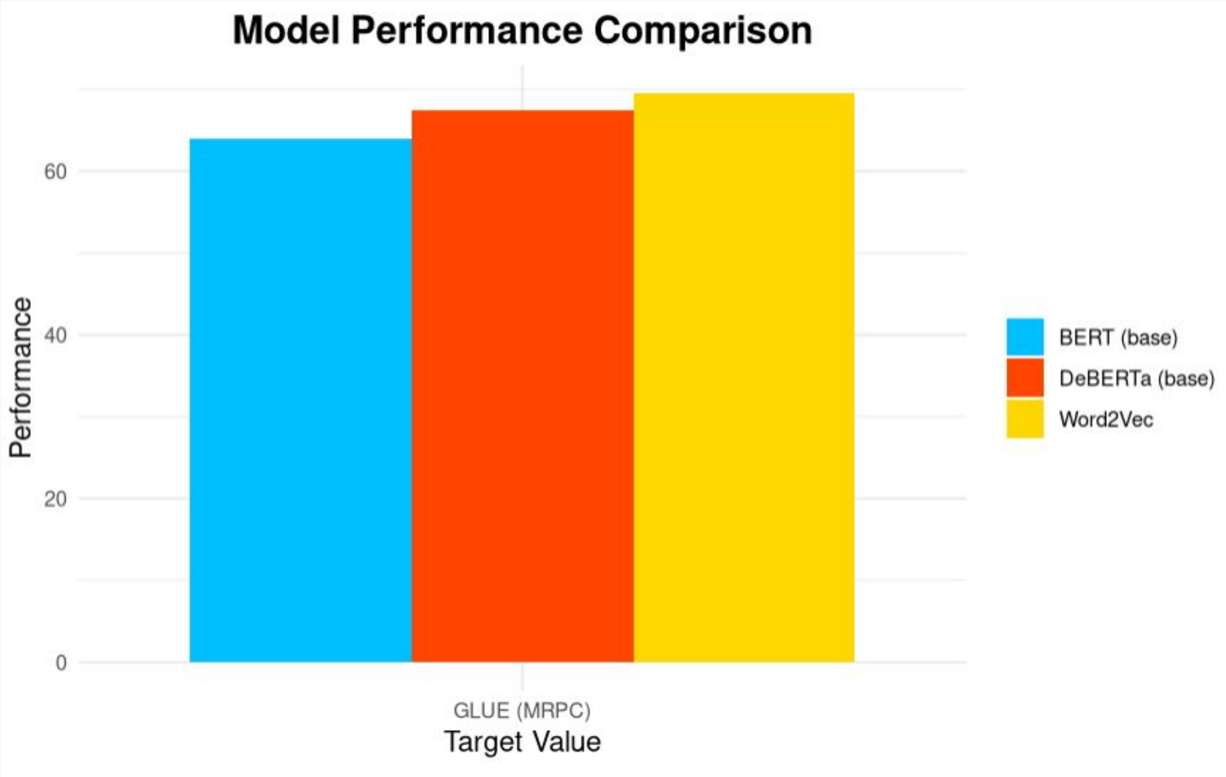
We will use it to check for semantic similarity, and therefore benchmark the embeddings of Word2Vec vs. BERT!

For that, we will train a logistic regression model on Word2Vec and BERT embeddings...

TU WIEN DS

## Zero-Shot Training

Google Developer Groups




TU WIEN DS

Word2Vec vs. BERT
# Zero-Shot Training

Google Developer Groups

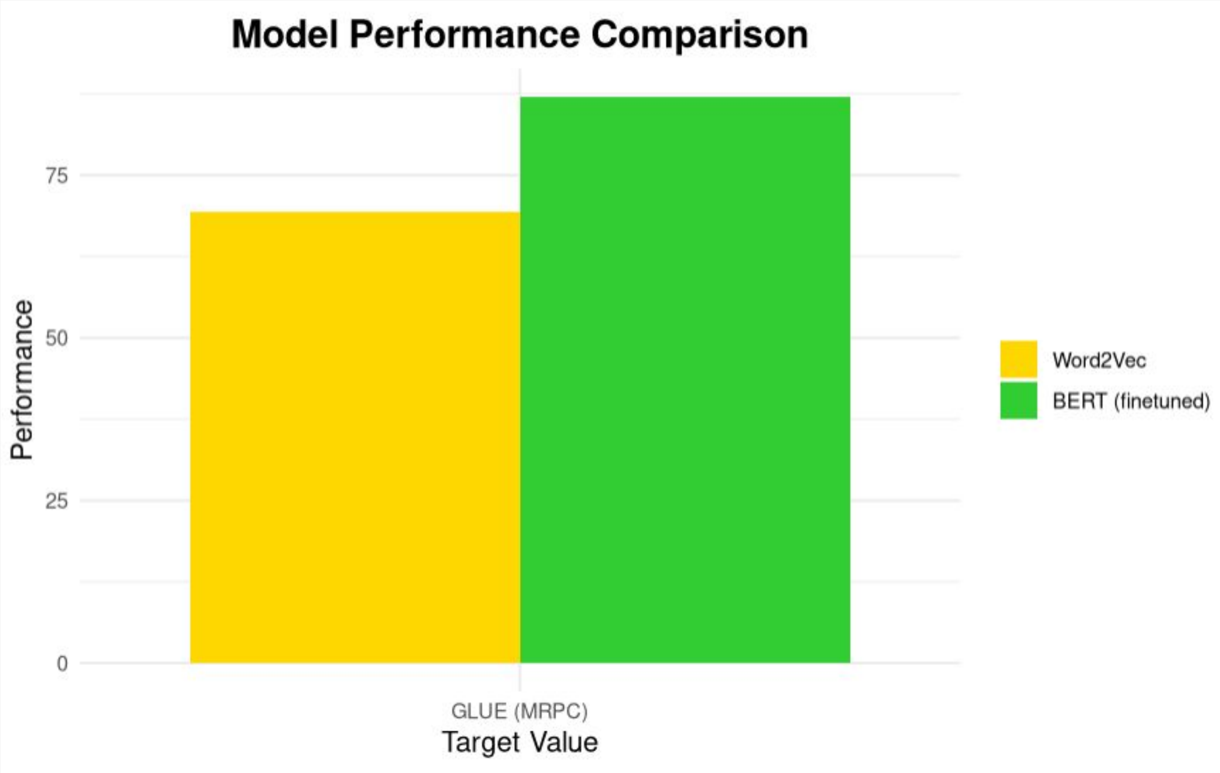
**Discuss:** When Word2Vec is better, then why do we use BERT?

**Key-Feature:** BERT is a contextual model that can be fine-tuned, while Word2Vec is a **static, pre-trained** word embedding model; these vectors don't change based on new input contexts!
Therefore, we will pre-train BERT via text classification...

**Word2Vec vs. BERT ⇔ static vs. contextual embeddings**

**Fine-Tuning**

Google Developer Groups



# Discussion

Google Developer Groups

In a world of transformer models, is there still any need for Word2Vec? How might Word2Vec still be relevant in smaller, less resource-intensive settings?

Recall the differences between Continuous Bag of Words (CBOW) and Skip-gram. Why does CBOW generally have worse performance for semantic accuracy?

About CBOW and Skip-gram, in which cases might one be preferred over the other?