Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean

Presented by,

Anagha M B, Gokul P V, and Chandu Dadi

Introduction

What is **Word representations in vector space/ Word Embeddings**?

Word representation in vector space refers to the way words are represented as numerical vectors in a high-dimensional space, where similar words are closer together and dissimilar words are farther apart.

Example: "The students of the MAI 2024 batch are cool!"

The → [0.1, 0.2, 0.3]

Students → [0.7, 0.8, 0.5]

Of → [0.3, 0.4, 0.2]

MAI → [0.8, 0.7, 0.3]

2024 → [0.9, 0.6, 0.2]

Batch → [0.5, 0.3, 0.6]

Are → [0.2, 0.5, 0.9]

Cool → [0.8, 0.9, 0.7]

Introduction

The → [0.1, 0.2, 0.3]

Students → [0.7, 0.8, 0.5]

Of → [0.3, 0.4, 0.2]

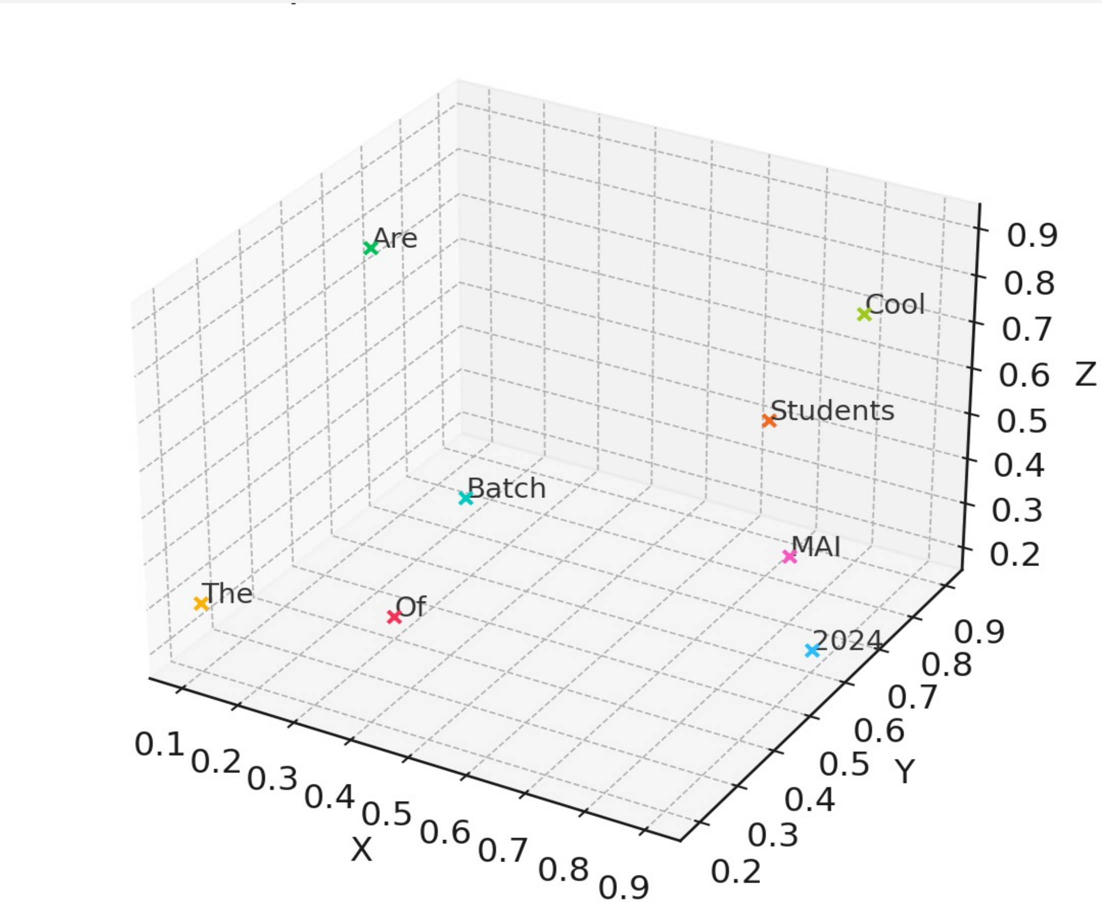
MAI → [0.8, 0.7, 0.3]

2024 → [0.9, 0.6, 0.2]

Batch → [0.5, 0.3, 0.6]

Are → [0.2, 0.5, 0.9]

Cool → [0.8, 0.9, 0.7]



Introduction

- **Why** Word representations in vector space?

Human-Computer Interaction,

Search and Information Retrieval,

Language Translation,

Sentiment analysis,

Content classification,

Generating Text,

and so more.

Background

- **One-hot encoding** is used as a basic method to represent words as vectors, where each word in the vocabulary is assigned a unique vector with all elements being 0 except for one position, which is 1.

The → [1, 0, 0, 0, 0, 0, 0, 0, 0]

students → [0, 1, 0, 0, 0, 0, 0, 0, 0]

of → [0, 0, 1, 0, 0, 0, 0, 0, 0]

the → [0, 0, 0, 1, 0, 0, 0, 0, 0]

MAI → [0, 0, 0, 0, 1, 0, 0, 0, 0]

2024 → [0, 0, 0, 0, 0, 1, 0, 0, 0]

batch → [0, 0, 0, 0, 0, 0, 1, 0, 0]

are → [0, 0, 0, 0, 0, 0, 0, 1, 0]

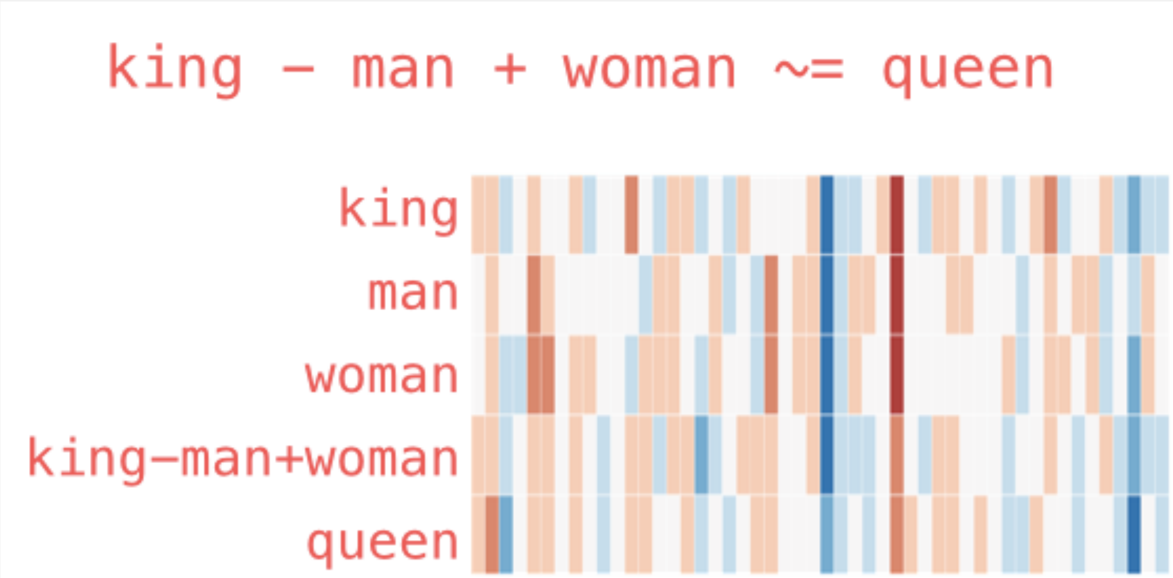
cool → [0, 0, 0, 0, 0, 0, 0, 0, 1]

Limitation: Outputs are high dimensional vectors and sparse.

TF-IDF - **sparse** and **high-dimensional**, especially when applied to large corpora of text

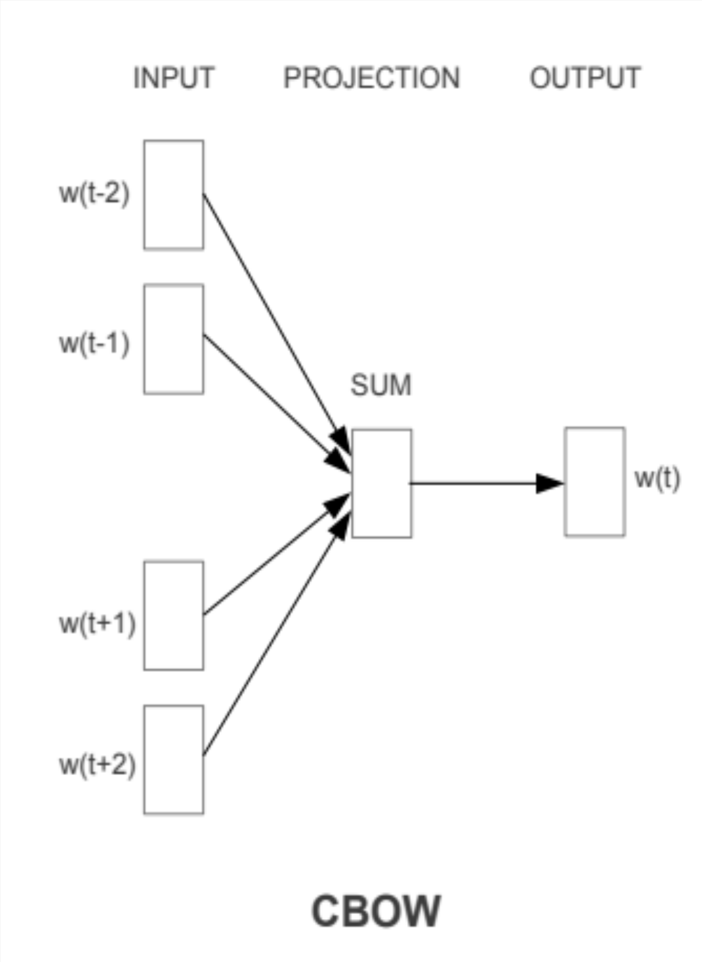
Objective of the Paper

The paper introduces **Word2Vec** models (Skip-gram & CBOW) that learn efficient **word representations** from large text corpora and captures both **syntactic** and **semantic** word relationships.



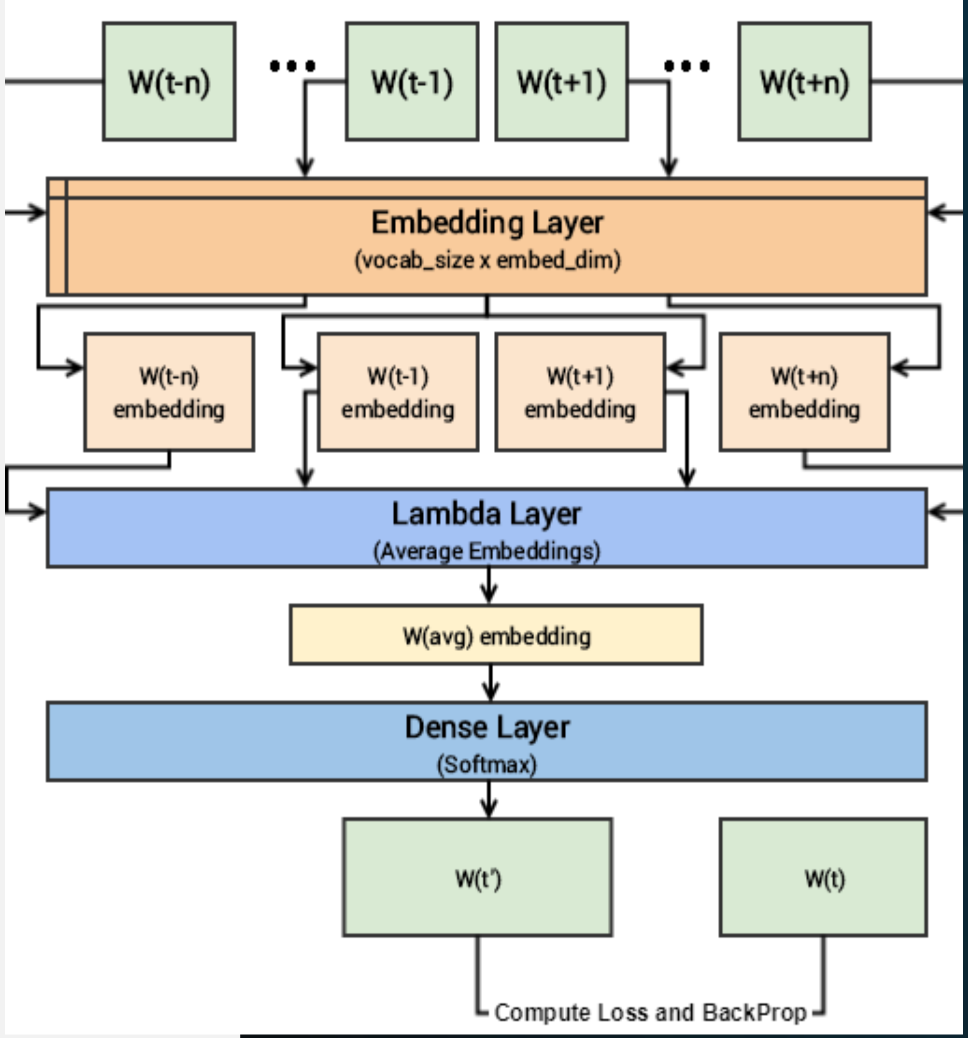
Continuous Bag-of-Words Model

- **How it works**: The CBOW model predicts the **target word** by using the **surrounding context words**.
- **Key Idea**: It averages the vectors of the context words and then predicts the word that is most likely to fit in the middle (ignores the word order).
- **Training Complexity**: $Q = N \times D + D \times \log_2(V)$
  - N : Number of context words
  - D : Dimensionality of word vectors
  - V : Vocabulary size
- **Example**: Predicts "eat" from the context "I will ___ dinner tonight."



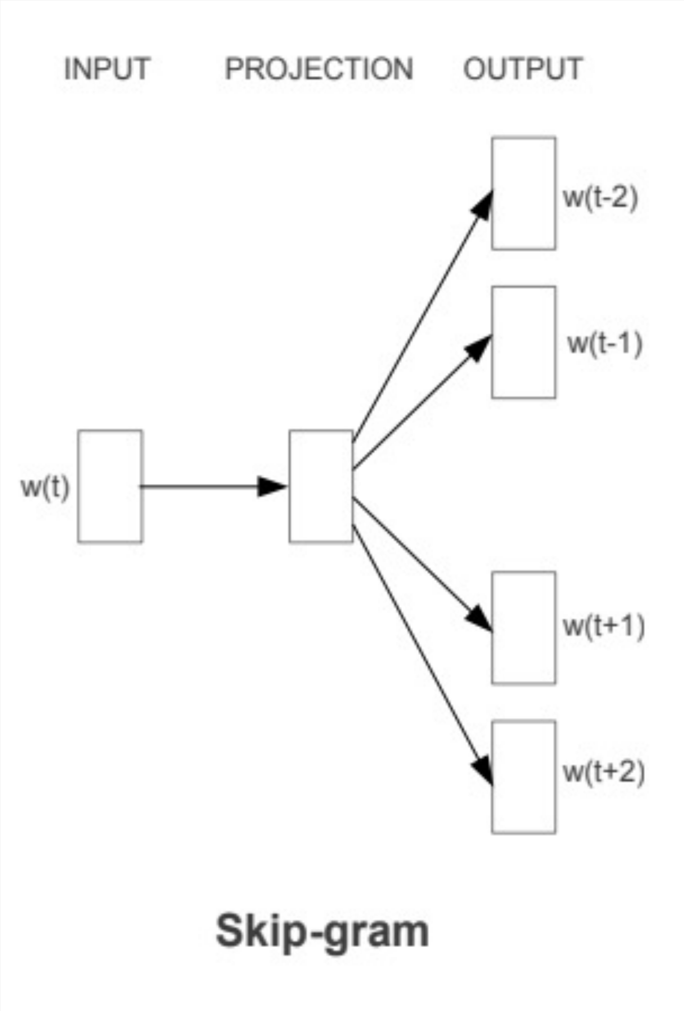
Training of CBOW

- The context words are first passed as an input to an embedding layer (initialized with some random weights)
- The word embeddings are then passed to a lambda layer where we average out the word embeddings.
- Pass these embeddings to a dense SoftMax layer that predicts target word. Compute the loss and perform backpropagation to update the embedding layer.



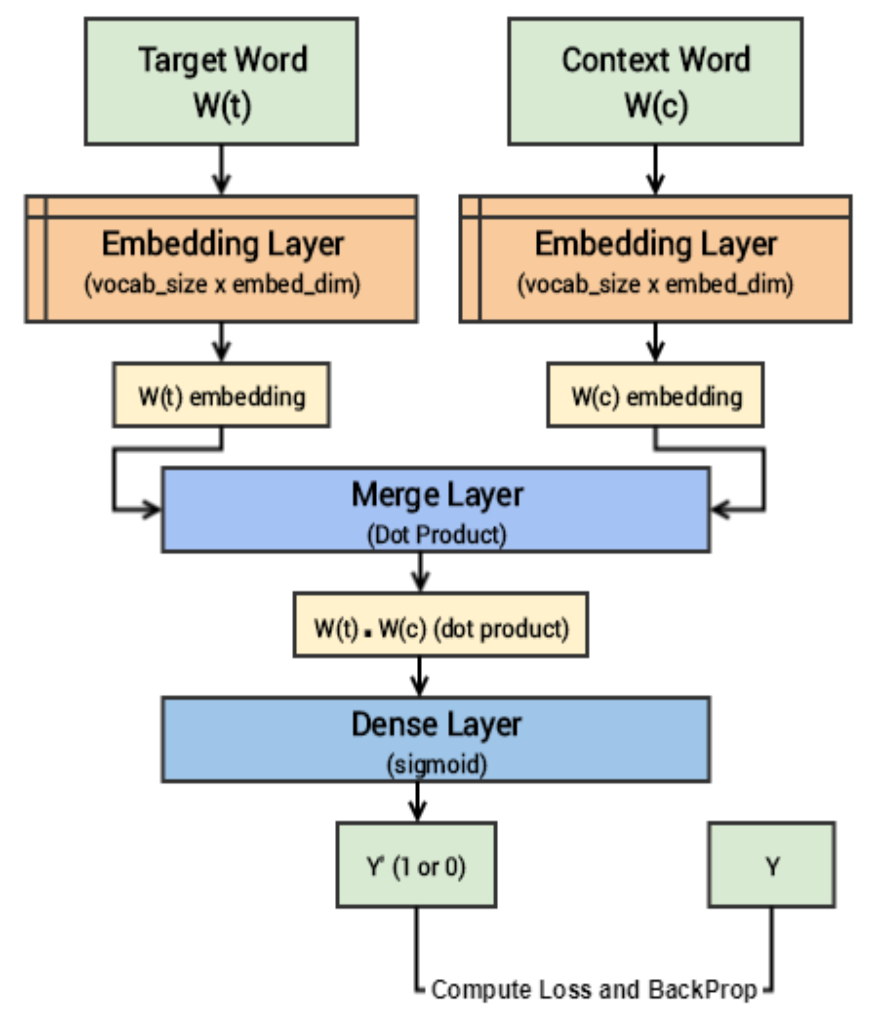
Continuous Skip-gram Model

- **How it works**: The Skip-gram model does the opposite of CBOW: it uses a **central word** to predict the surrounding **context words.**
- **Key Idea**: Given a word in the sentence, it predicts which words are likely to occur around it.
- **Training Complexity**: $Q = C \times (D + D \times \log_2(V))$
  - C : Maximum distance between words
  - D : Dimensionality of word vectors
  - V : Vocabulary size
- **Example**: Given "France", Skip-gram predicts related words like "Paris," "Europe," and "Eiffel Tower."



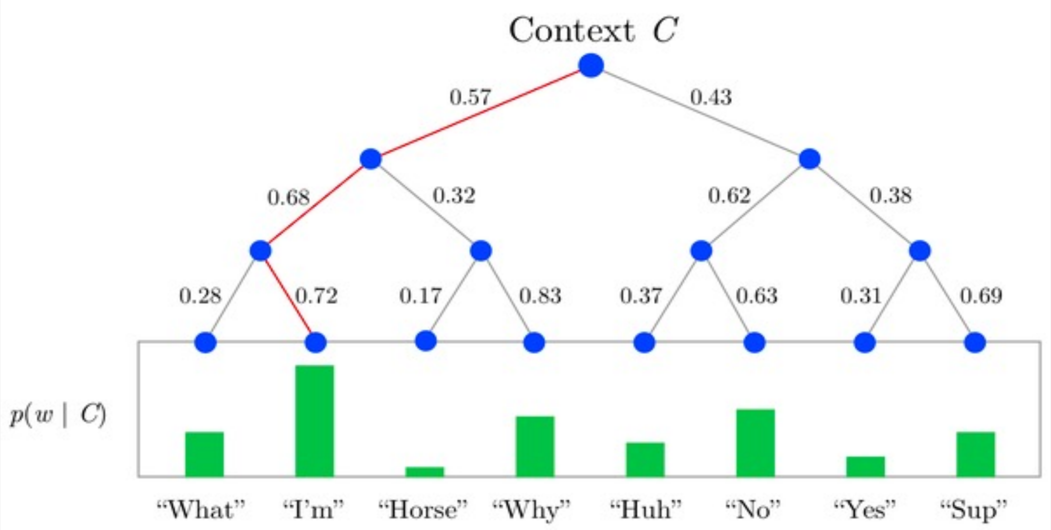
Training of Skip-gram Model

- Both the target and context word pairs are passed to individual embedding layers from which dense word embeddings for each of these words are obtained.
- Compute the dot product of these two embeddings using ‘merge layer’.
- This dot product value is then sent to a sigmoid layer that outputs either 0 or 1.
- The output is compared with the actual label and the loss is computed followed by backpropagation with each epoch to update the embedding layer in the process



Optimization Techniques - Hierarchical SoftMax

- **Problem**: Full SoftMax computation is **too costly** for large vocabularies.
- **Solution**: Use a **binary tree structure** for words.
  - **Efficient traversal**: Predict probabilities along the tree path.
  - **Reduces computation**: From O(V) to O(log V), where V is the vocabulary size.
- **Benefit**: **Scalable** for large datasets.
- Improves efficiency for **large vocabularies.**



Optimization Techniques
Negative Sampling

- **Problem**: Computing the full SoftMax is inefficient for frequent updates.
- **Solution**: Approximate SoftMax by updating weights for:
  - **Target word** (positive example).
  - A few randomly selected **negative examples**.
- **Benefit**: Significantly reduces training time by focusing on a subset of words.
- Optimizes training with **fewer updates**.

Skipgram

shalt	not	make	a	machine

input	output
make	shalt
make	not
make	a
make	machine

Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Some Interesting Results

[Embedding projector - visualization of high-dimensional data]

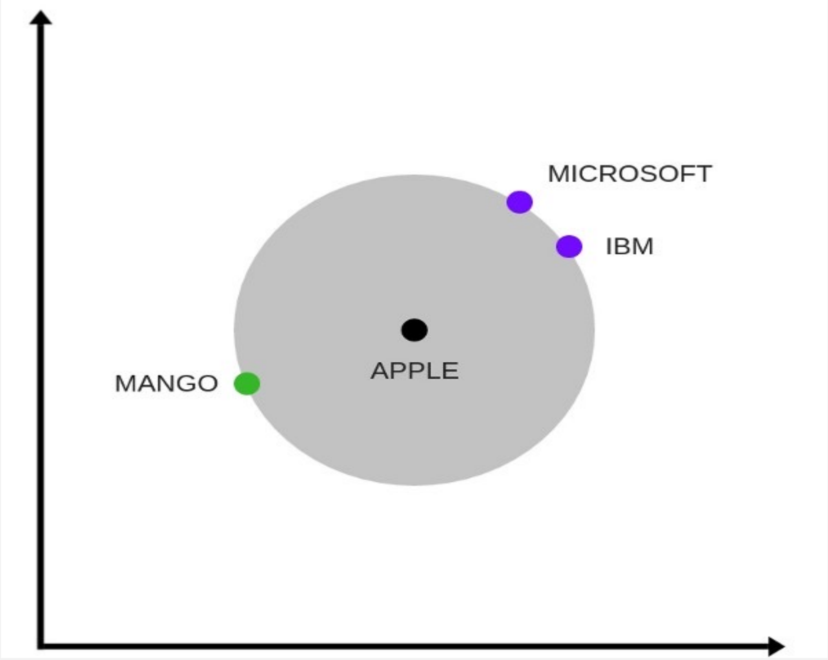


Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

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

Limitations

- **No Sub word Information**:
  - Word2Vec treats words as atomic units, failing to handle **morphologically rich** languages.
  - Struggles with **rare words** and **out-of-vocabulary** (OOV) words.
- **Context Independence**:
  - Word2Vec generates **static embeddings**—the same word has the same vector regardless of context (e.g., "bank" in finance vs. river context).
- **Lack of Global Co-occurrence Information**:
  - Focuses on local context, missing **global word statistics** that can capture more nuanced relationships.

Conclusion

- **Word2Vec** introduces efficient methods (Skip-gram and CBOW) for learning word embeddings.
- Models can capture both **syntactic** and **semantic** relationships between words.
- Optimization techniques like **Hierarchical Softmax** and **Negative Sampling** enable scalability to large datasets.
- Word2Vec revolutionized NLP by enabling more **effective word representations**.
- It laid the groundwork for more advanced models like **GloVe**, **FastText**, and **contextual embeddings** (e.g., BERT).

References

- Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. 2013 Jan 16.
- https://jalammar.github.io/illustrated-word2vec/
- https://www.analyticsvidhya.com/blog/2021/07/word2vec-for-word-embeddings-a-beginners-guide/

Thank you!