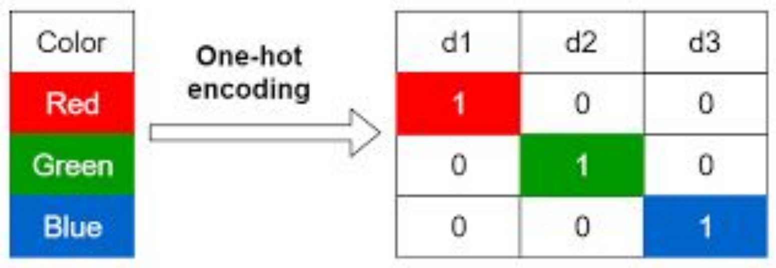
Efficient Estimation of Word Representations in Vector Space

Group 7
Liang Liu & Simei Yan & Zitai Wu & Bangguo Xu

Traditional Word Representation

One-hot encoding

- Each word has a unique position set to 1 in a high-dimensional space, with all other positions set to 0.
- The number of dimensions in this space is equal to the number of words in the vocabulary.

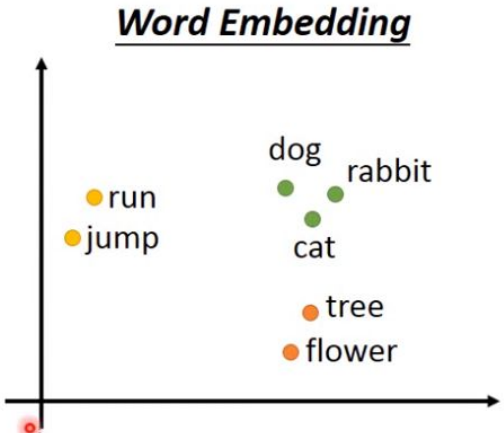


Limitations of One-hot Encoding

- **Lack of Semantic Association**
  - One-hot vectors cannot capture relationships or similarities between words.
- High-dimensional Space
  - The dimension of a one-hot vector is equal to the size of the vocabulary
- Low Storage and Computational Efficiency
  - Storing a large number of one-hot vectors, especially for extensive vocabularies, can take up a significant amount of storage space.

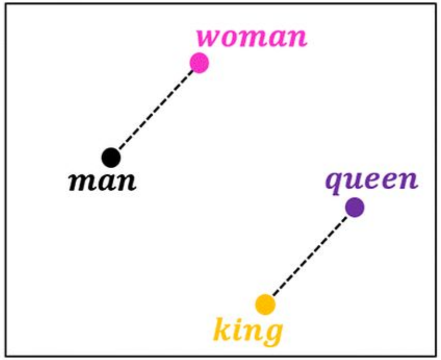
Word embedding

- Word Embedding is a technique that maps words into a vector space.
- It converts each word in the vocabulary into a fixed-size vector.



Benefits of Word Embeddings

- **Dimensionality Reduction**
  - Traditional methods may need up to millions of dimensions, but word embeddings use only 50-300 dimensions
- **Semantically Rich**
  - Word embeddings in vector space capture both semantic and syntactic information, unlike the spares nature of one-hot encoding.
- **Meaningful Computations**
  - Word embeddings enable meaningful calculation, e.g., the vector operation “King”- “Man” + “Woman” approximates “Queen”.

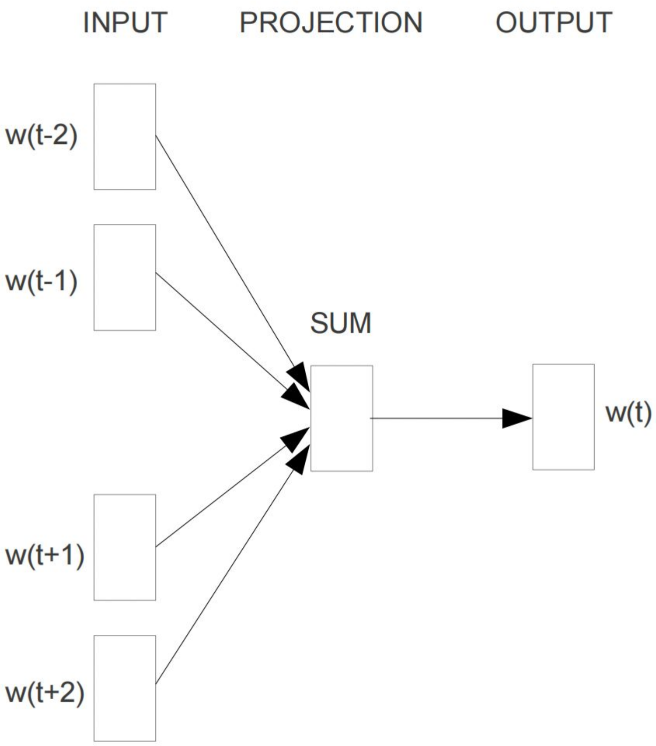


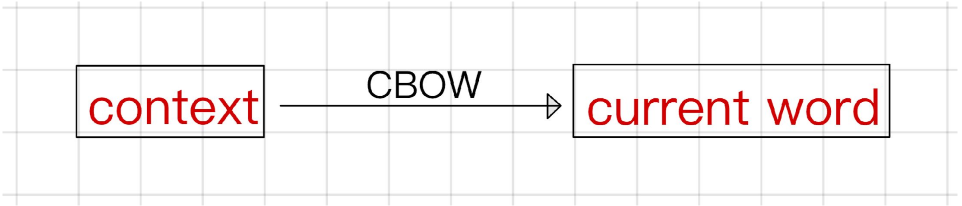
Word2Vec

Represent words in a dense vector format such that words with similar meanings are close in the vector space.

Word2Vec includes two primary training algorithms: Continuous Bag of Words (CBOW) and Skip-Gram.

Continuous Bag-of-Words Model (CBOW)





- use context to predict the current word
- As a product of CBOW model learning, we can obtain distributed representations of words.
- ignore the order of these input words in the sentence

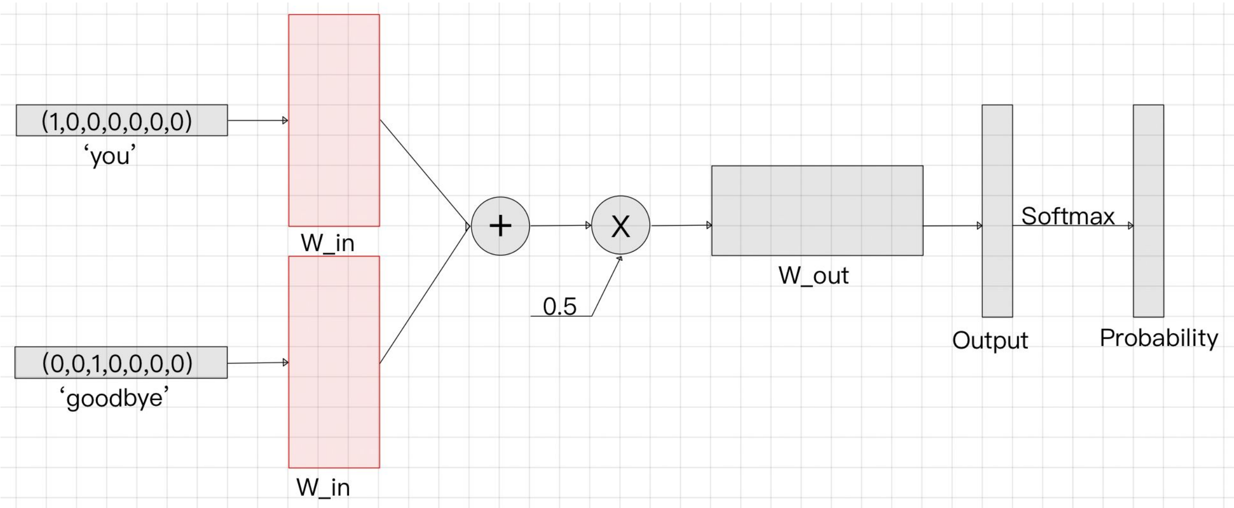
Continuous Bag-of-Words Model (CBOW)

you ? goodbye and i say hello.

- context: “you”, “goodbye”

current word: “say”

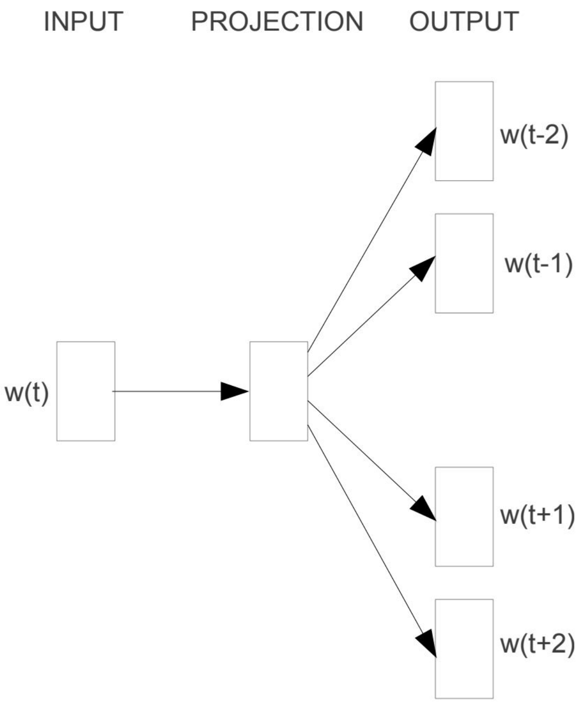
Word	Word ID	one–hot vector
you	0	(1,0,0,0,0,0,0)
goodbye	2	(0,0,1,0,0,0,0)



- The transformation from the input layer to the intermediate layer is completed by the same fully connected layer with weight W_in.
- The transformation from the intermediate layer to the output layer is completed by another fully connected layer with weight W_out.
- Each row of the weight matrix W_in stores each word vector.

Skip-gram

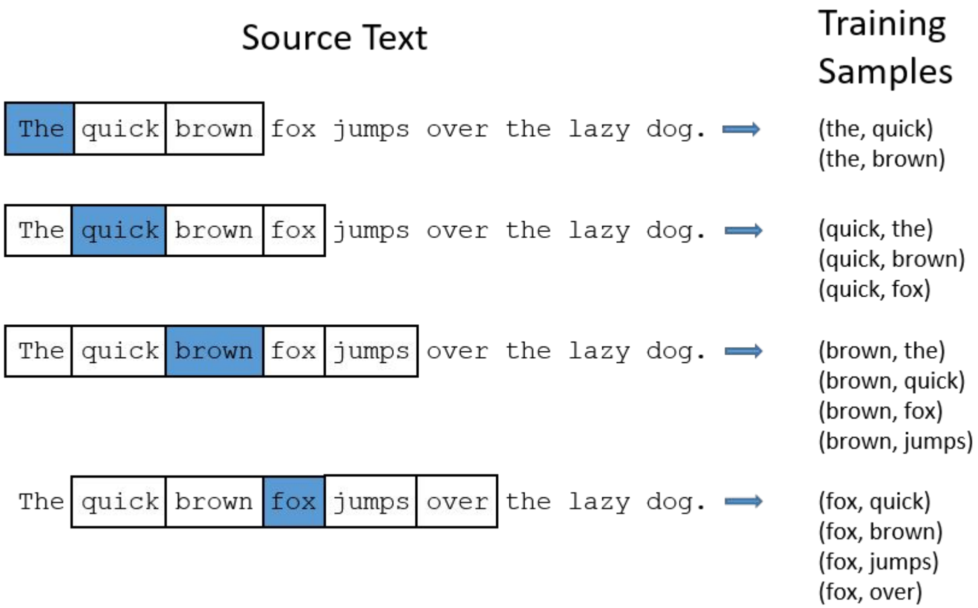
Skip-gram is an architecture for computing word embeddings. Instead of using surrounding words to predict the center word as with CBow, Skip-gram uses the central word to predict the surrounding words.



Skip-gram

Skip-gram

We'll train the neural network to do this by feeding it word pairs found in our training documents. The below example shows some of the training samples (word pairs) we would take from the sentence "The quick brown fox jumps over the lazy dog." We've used a small window size of 2 just for the example. The word highlighted in blue is the input word.



Model Details

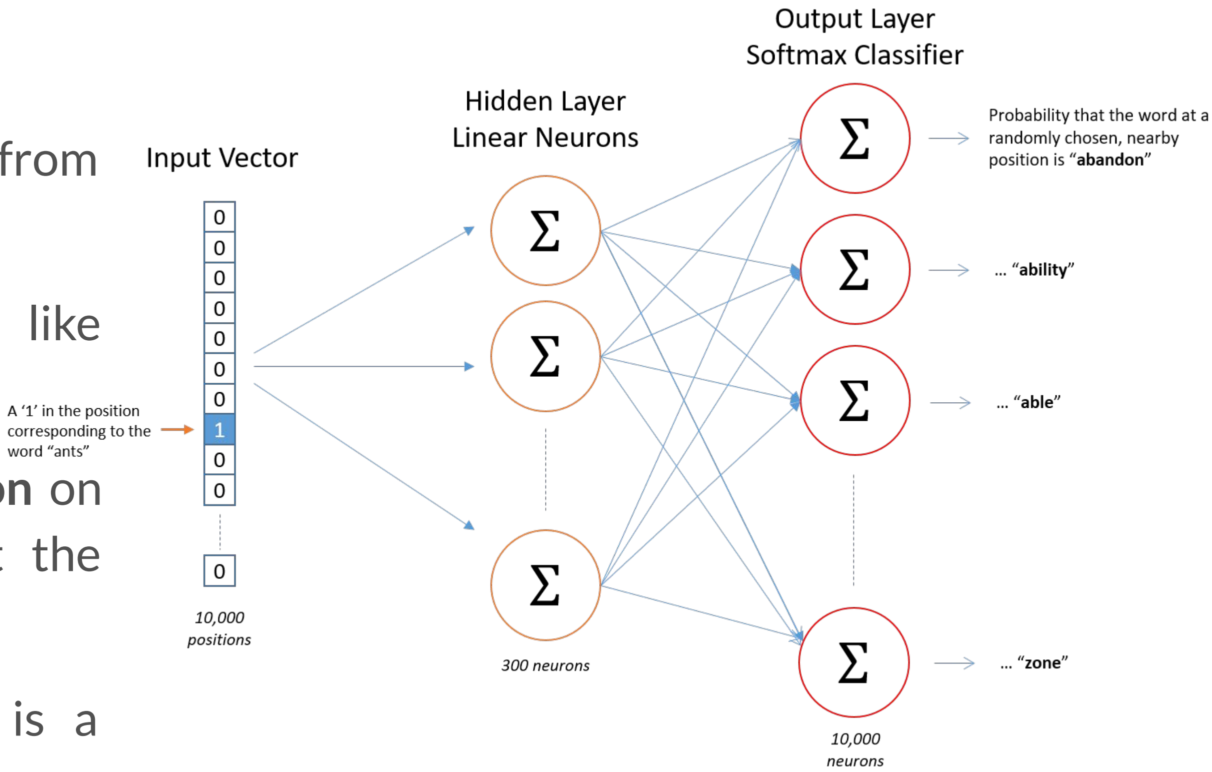
So how is this all represented?

1. Build a vocabulary of words from the training documents.

2. Represent an input word like "ants" as a one-hot vector.

3. There is **no activation function** on the hidden layer neurons, but the output neurons use softmax.

4. The output of the network is a single vector.

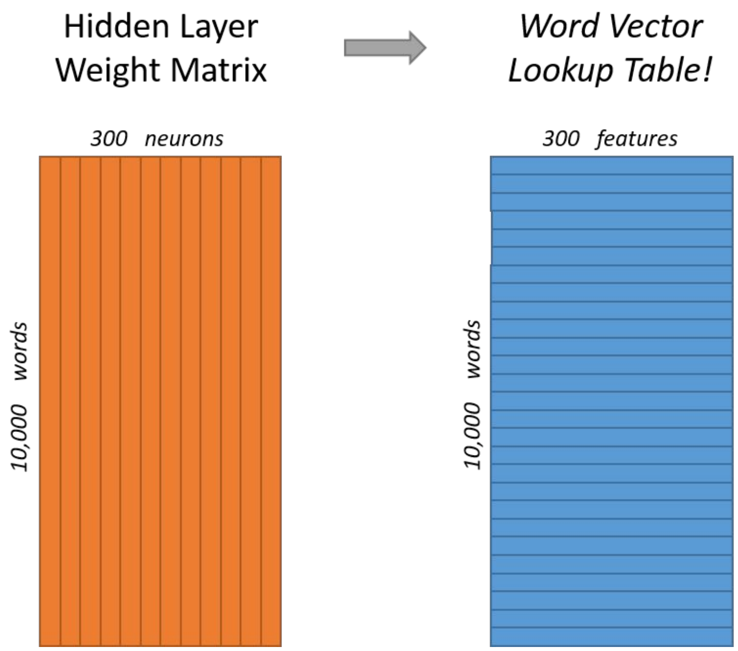


The Hidden Layer

For example, we're going to say that we're learning word vectors with 300 features. So the hidden layer is going to be represented by a weight matrix with 10,000 rows (one for every word in our vocabulary) and 300 columns (one for every hidden neuron).

If we look at the *rows* of this weight matrix, these are actually what will be our word vectors!

So the end goal of all of this is really just to learn this hidden layer weight matrix!



Lookup Table

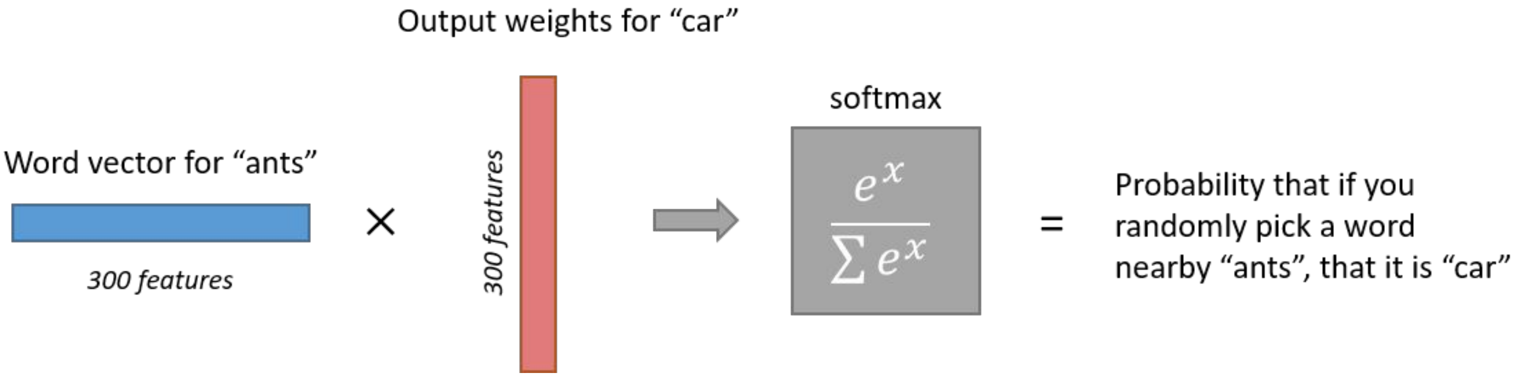
“The one-hot vector is almost all zeros… what’s the effect of that?”

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

If you multiply a 1 x 10,000 one-hot vector by a 10,000 x 300 matrix, it will effectively just *select* the matrix row corresponding to the “1”

This means that the hidden layer of this model is really just operating as a lookup table. The output of the hidden layer is just the “word vector” for the input word.

The Output Layer



The 1 x 300 word vector for “ants” then gets fed to the output layer. The output layer is a softmax regression classifier.

Short Summary

1. Data Preprocessing: Select and cleanse text corpus, create a vocabulary, and perform basic processing such as tokenization on the text content.
2. Model Definition: The dimensions of the input and output layers correspond to the vocabulary size, while the hidden layer dimension is the predefined vector space dimension.
3. Context Selection: For each target word in the corpus, identify its context words within a specific window.
4. Model Training: Use the target words to predict context words, and adjust model weights by calculating the error between predicted outputs and actual outputs (using algorithms like gradient descent and backpropagation).
5. Vector Extraction: After the model training, obtain vector representations for each word using the weights of the hidden layer.

Higher quality of word vectors

Traditional Methods

example words	similar words
China	Italy, Spain, Germany, Belgium
Apple	Orange, Banana, Pear
King	Queen, Prince, Monarch, Ruler

It's challenging to capture more complex relationships.

different types of similarities

Example 1

big	bigger
small	smaller

Example 2

big – biggest	small – smallest

Example 3

Equation: X = vector("biggest") - vector("big") + vector("small")

Example 4

"France is to Paris as Germany is to Berlin."

Test the quality of the word vector

To measure quality of the word vectors, we define a comprehensive test set that contains semantic questions and syntactic questions.

two steps:

① Create a list of similar word pairs

② Choose two word pairs randomly, a large list of questions is formed by connecting two word pairs

Example

city	state
München	Bayern
Düsseldorf	Nordrhein-Westfalen
Stuttgart	Baden-Württemberg

Question: München is to Bayern as Stuttgart is to what?

Test in corpus

Corpus: Google News(1 million most frequent words)

first evaluated models: use the most frequent 30k words

Result in CBOW model

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Test in corpus

Corpus: Google News(1 million most frequent words)

first evaluated models: use the most frequent 30k words

Result in CBOW model

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Comparison of Model Architectures

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	$\boxed{64}$	61
Skip-gram	$\boxed{55}$	59	56

Compare with other models

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Comparison with different epochs and data size

Training a model with double the data for one cycle can be as good or better than going through the same data three times, and also makes the process a bit faster.

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Large Scale Parallel Training of Models

When using the **distributed framework**, the CBOW model and the Skip-gram model are much closer to each other than their **single-machine** implementations.

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Conclusion

The most important contribution of this paper is that it introduces **two new model** architectures (CBOW, Skip-gram) for computing continuous vector representations of words from very large datasets.

Advantage

1. Perform well in word similarity tasks
2. Have low computational costs

Thank you!