

---

# Word Embeddings

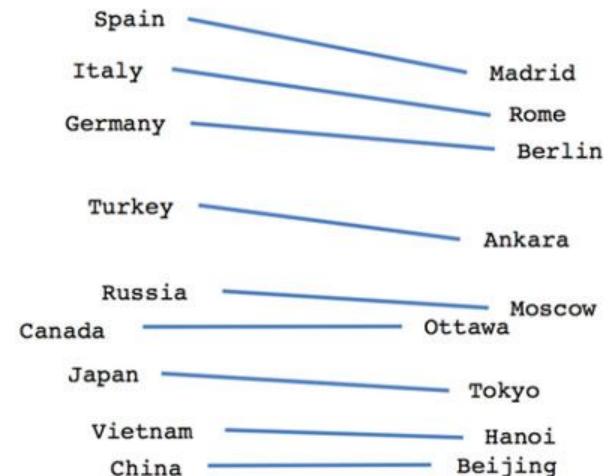
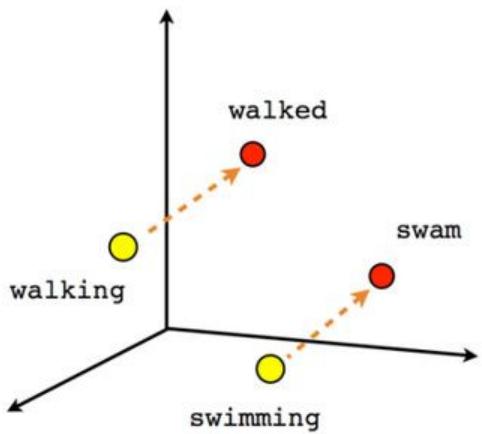
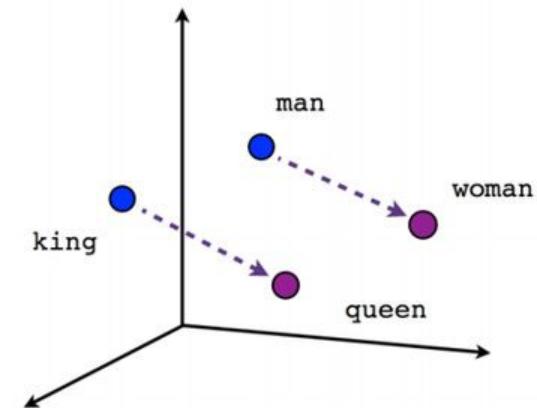
Chapter 4

---

# Word2Vec

Word embedding is a technique in natural language processing (NLP) that represents words as numerical vectors in a high-dimensional space, capturing semantic relationships between words so that similar words have similar vector representations.

Word2Vec is one of the most popular methods for generating these embeddings, developed by researchers at Google.



# Word embeddings

	battle	horse	king	man	queen	..	woman
authority	0	0.01	1	0.2	1	...	0.2
event	1	0	0	0	0	...	0
has tail?	0	1	0	0	0	...	0
rich	0	0.1	1	0.3	1	...	0.2
gender	0	1	-1	-1	1	...	1

<b>King</b>	-	<b>man</b>	+	<b>woman</b>	=	<b>result</b>	≈	<b>Queen</b>
1		0.2		0.2		1		1
0		0		0		0		0
0		0		0		0		0
1		0.3		0.2		0.9		1
-1		-1		1		1		1

# Word2Vec

It is a technique developed by Google (Mikolov et al., 2013) to convert words into dense vector representations such that similar words have similar vectors

"Words that appear in similar contexts tend to have similar meanings".

- semantic and syntactic relationships between words using shallow neural networks

learns a weight matrix (embedding matrix) where each word is represented as a dense vector in an N-dimensional space (e.g., 100D, 300D)

Words used in similar contexts will have vectors close to each other

Word2Vec is trained on the Google News dataset, which contains about 100 billion words.

embeddings using the library Gensim or spaCy.

# Word2Vec

It is a technique developed by Google (Mikolov et al., 2013) to convert words into dense vector representations such that similar words have similar vectors

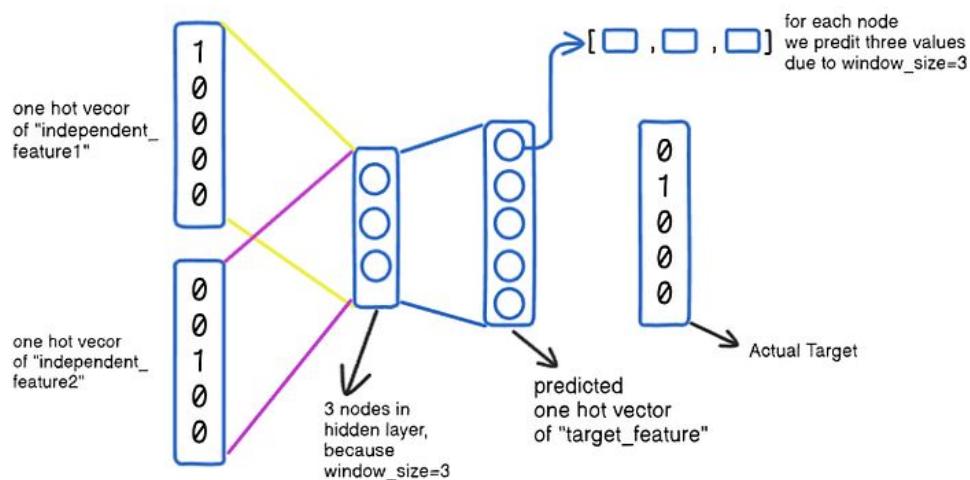
Word2Vec uses shallow neural networks with two main architectures: Continuous Bag of Words (CBOW) and Skip-gram.

CBOW predicts a target word from its surrounding context, while Skip-gram predicts the context words given a target word.

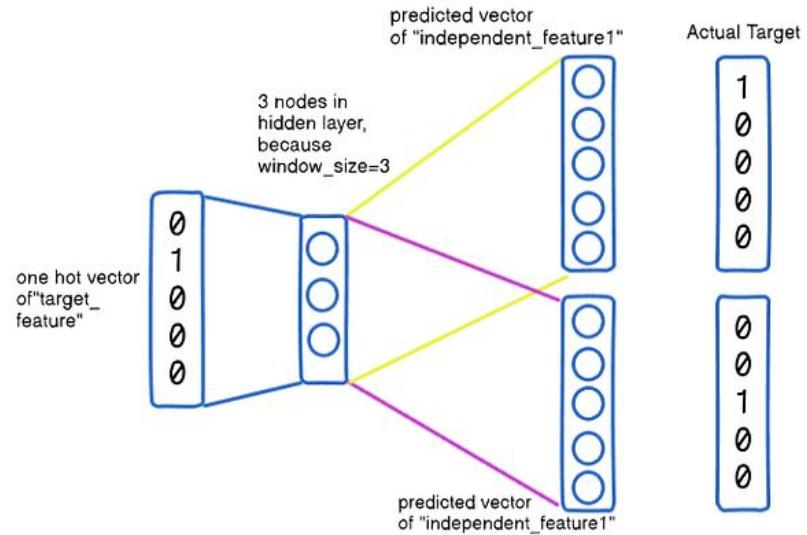
Both methods train on large text corpora to learn vector representations that place semantically similar words close together in the vector space

# Word2vec

## CBOW



## Skip-gram



# CBOW

CBOW takes multiple context words (each as a one-hot vector) around a missing word and feeds them into a hidden layer to predict the target word in the center.

In the diagram, several one-hot context vectors are combined into a hidden layer with as many units as the embedding size, then the output layer predicts a one-hot vector for the target word, which is compared with the actual target to train the weights.

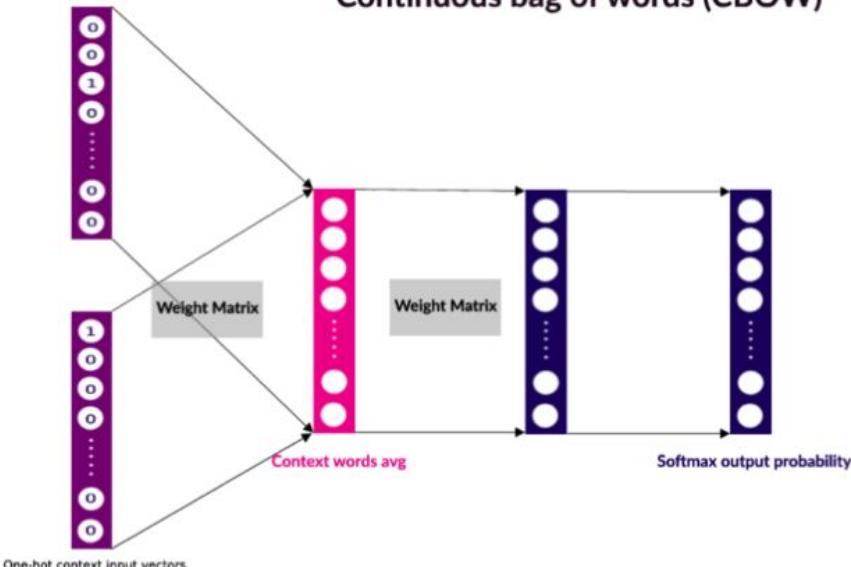
# CBOW

CBOW uses surrounding context words as input to predict the target word

The context words are encoded using one-hot vectors, which are averaged to more accurately predict the target word

# CBOW

Continuous bag of words (CBOW)



The      Brown      Fox      jumped      over      the      lazy      dog

Context Words      Target Word      Context Words

## 1. Vocabulary & One-Hot Vectors

Let the vocabulary size be  $V$  which is the total number of unique words in your dataset

- Given a sentence like: "The cat sat on the mat"
  - ["the", "cat", "sat", "on", "mat"]  $\rightarrow V = 5$

Every word is represented as a one-hot vector of size  $V$

- For example: "cat" = [0, 1, 0, 0, ..., 0] (1 at the index corresponding to "cat")

Each input word is:  $x_i \in R^V$

- $x_i$  is a one-hot vector for the  $i$ th context word
- window size of 2, for the target word "sat", the context words are
  - ["The", "cat", "on", "the"]

## 2. Input to Hidden Layer

use a weight matrix  $W \in RV \times N$  to transform one-hot vectors into dense embeddings of size  $N$  (say, 50 or 100 dimensions)

For one-hot vector  $x_i$ , the embedding is:

$$W^T x_i = e_i \in R^N$$

To combine multiple context embeddings, CBOW averages them:

$$h = \frac{1}{C} \sum_{i=1}^C W^T x_i$$

where  $h \in R^N$

### 3. Hidden to Output Layer (Prediction)

To predict the center word using the context vector  $h$

Weight matrix  $U \in \mathbb{R}^{N \times V}$ , where each column  $u_w$  is a vector representing a word in the output layer

Compute a score for each possible word  $w$  using dot product:

$$score(w) = h^T u_w$$

Apply softmax over all vocabulary words to get probabilities:

$$P(w|context) = \frac{e^{h^T u_w}}{\sum_{v=1}^V e^{h^T u_v}}$$

Word with the highest probability is selected as the predicted center word

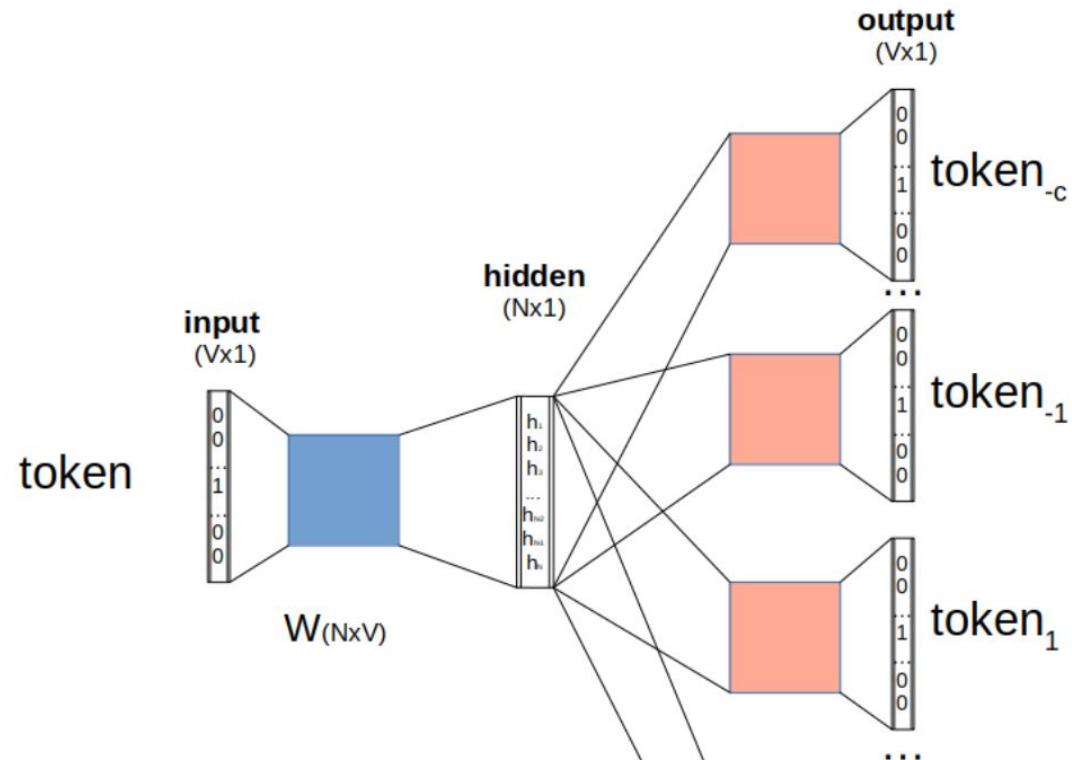
# Skipgram

Skip-gram does the reverse: it takes a single target word (one-hot) as input and predicts the surrounding context words.

In the diagram, the target word is mapped through the hidden layer to multiple output vectors, each trying to match one of the context words' one-hot vectors within the window size, and training adjusts weights so the model better predicts these neighbors

# Skipgram

model begins with the target word and predicts the context words that should surround it



## 1. Vocabulary & One-Hot Vectors

Let the vocabulary size be  $V$  which is the total number of unique words in your dataset

Every word is represented as a one-hot vector of size  $V$

Center word=  $w_t$

$$x = \text{one-hot}(w_t)$$

## 2. Input to Hidden Layer

use a weight matrix  $W \in \mathbb{R}^{V \times N}$  to transform one-hot vectors into dense embeddings of size  $N$  (say, 50 or 100 dimensions)

Embedding of the center word is:

$$h = W^T x \in \mathbb{R}^N$$

### 3. Hidden to Output Layer (Prediction)

To predict each context word from the hidden vector  $h$

Weight matrix  $U \in \mathbb{R}^{N \times V}$ , where each column  $u_{wc}$  is a vector representing a word in the output layer

For a context word  $w_c$ , the predicted score is:

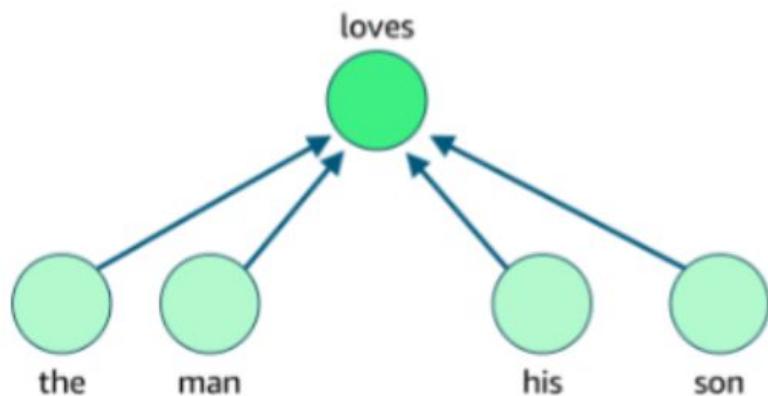
$$\text{score}(w_c) = h^T u_{wc}$$

Apply softmax to get the probability distribution over all words:

$$P(w_c | w_t) = \frac{e^{h^T u_{wc}}}{\sum_{v=1}^V e^{h^T u_v}}$$

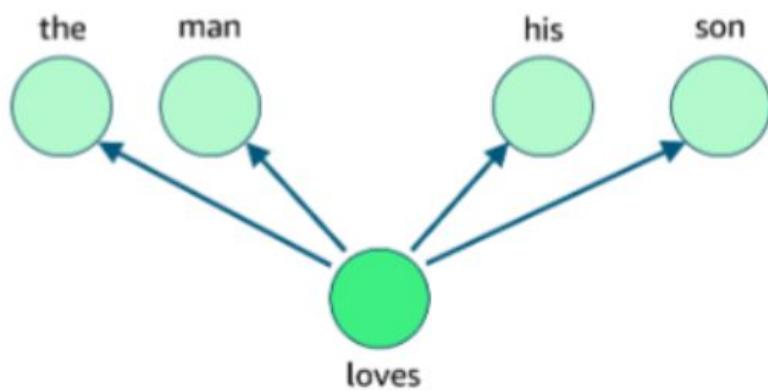
## Continuous bag-of-words (CBOW)

Generate center words from context words



## Skip-Gram

A word can be used to generate the words that surround it



# Word2Vec

In both architectures, the hidden layer weights become the learned word embeddings, so words that appear in similar contexts get similar vectors in this shared space.

The window size (shown as 3 nodes in the hidden layer label) controls how many neighboring words are considered, and after training these embeddings can be used for downstream NLP tasks such as similarity or classification

# GloVe

GloVe (Global Vectors for Word Representation) provides dense vector representations of words that capture semantic and syntactic relationships based on global co-occurrence statistics from a large corpus.

GloVe embedding improves Word-2-Vec's by computing the co-occurrence of corpus words once and deriving vector representations from it.

GloVe also takes a descriptive approach towards deriving its cost function by first examining the expected outcome of co-occurrence ratios.

# GloVe

GloVe is an unsupervised learning algorithm that creates word embeddings by analyzing word co-occurrence statistics across an entire corpus.

Combines count-based methods with predictive models

Global word-word co-occurrence matrix from corpus

Dense vector representations capturing semantics

# GloVe

In Word-2-Vec we tend to re-visit neighboring words multiple times as a new center word vector is being trained.

Aware of this duplication of effort, GloVe training starts by forming a co-occurrence matrix  $X$  where the  $ij$ -th entry is the number of times words on  $i$ -th row and  $j$ -th column appeared together

# Co-Occurrence Count Approaches

## a. Window Count

Where a window length is decided in advance, e.g. 5 or 10. This approach tends to capture both syntactic, e.g. Part of Speech, and semantic information.

## b. Document Count

Where co-occurrences are across documents, per document in the corpus. This approach tends to extract general topics information and leads to Latent Semantic Analysis.

In both approaches, the co-occurrence matrix is calculated once across the entire corpus and later processed to train the word vectors.

This proves to enhance the training time significantly in comparison to Word-2-Vec.

# Example

Following example depicts one such co-occurrence matrix for a window size of 1 when corpus consists of only the following three sentences.

D1: I like deep learning.

D2: I like NLP.

D3: I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0

## Co-occurrence ratios

After counting the co-occurrences, GloVe argues that for any group of three corpus words,  $w_1$ ,  $w_2$ ,  $w_3$ ; one of the following three scenarios applies:

1.  $w_3$  is relevant to both  $w_1$  and  $w_2$ :

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \simeq 1$$

# Co-occurrence ratios

For example, let  $w1=$  “*gymnastics*”,  $w2=$  “*javelin*” and  $w3=$  “*olympics*”:

$$P(\text{olympics}|\text{gymnastics}) = \frac{\#(\text{olympics}, \text{gymnastics})}{\sum_{w \in \text{corpus}} \#(w, \text{gymnastics})}$$

$$P(\text{olympics}|\text{javelin}) = \frac{\#(\text{olympics}, \text{javelin})}{\sum_{w \in \text{corpus}} \#(w, \text{javelin})}$$

where  $\#(v, z)$  denotes the respective entry in the co-occurrence matrix representing words  $v$  and  $z$ . It's not hard to see Equation (2) holds:

$$\frac{P(\text{olympics}|\text{gymnastics})}{P(\text{olympics}|\text{javelin})} \approx 1$$

# Co-occurrence ratios

2.  $w_3$  is only relevant to one of  $w_1$  or  $w_2$ , not both:

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \gg 1 \quad (3)$$

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \ll 1 \quad (4)$$

Following the previous example, let  $w_3 = \text{"vault"}$  or  $w_3 = \text{"spear"}$ . Take a moment to see how the ratio this time will be significantly smaller or larger than 1.

$$\frac{P(\text{vault}|\text{gymnastics})}{P(\text{vault}|\text{javelin})} \gg 1$$

$$\frac{P(\text{spear}|\text{gymnastics})}{P(\text{spear}|\text{javelin})} \ll 1$$

# Co-occurrence ratios

3.  $w_3$  is irrelevant to both  $w_1$  and  $w_2$ :

$$\frac{P(w_3|w_1)}{P(w_3|w_2)} \approx 1 \quad (5)$$

Continuing on the same example, let  $w_3 = \text{"croissant"}$ . Once again the ratio tends to be close to 1, as in case 1.

$$\frac{P(\text{croissant}|\text{gymnastics})}{P(\text{croissant}|\text{javelin})} \approx 1$$

# Co-occurrence ratios

GloVe uses co-occurrence ratios to learn word relationships, where cases 1 and 3 both give ratio  $\approx 1$  but case 2 gives extreme ratios—allowing the model to focus only on discriminative words.

GloVe ignores ratio=1 cases (1 & 3) via low weighting  $f(X_{ij})$ , they add noise but don't mislead.

Only case 2 ratios ( $>>1$  or  $<<1$ ) force vectors to separate meaningfully, like pushing "vault" closer to gymnastics than javelin.

# FastText

FastText is a library for efficient text classification and word representation learning in NLP, extending Word2Vec by incorporating subword information through character n-grams.

This subword approach allows handling of out-of-vocabulary (OOV) words, morphological variations, and rare terms by representing words as sums of their subword embeddings.

# FastText

## Core Mechanism

FastText breaks words into character n-grams, typically 3 to 6 characters long, with special boundary markers like < and >.

For "running", example 3-grams include <ru, run, unn, nni, nin, ing>, and the word vector is the sum (or average) of these n-gram vectors plus the full word vector if known

# Key advantages

Subword handling improves robustness in morphologically rich languages and noisy text like social media.

Generates embeddings for unseen words by combining subword parts, solving OOV issues in traditional models.

Captures internal word structure, aiding tasks like language identification and classification.

Trained efficiently with techniques like hierarchical softmax and negative sampling.

# N gram extraction

FastText builds word embeddings by representing each word as a sum (or average) of embeddings from its character n-grams, extending traditional word2vec models.

This subword approach uses overlapping character sequences of lengths typically from 3 to 6, with special boundary markers like "<" at the start and ">" at the end to denote word edges.

For a word like "running" with minn=3 and maxn=6, FastText extracts all substrings in that range, such as <ru, run, unn, nni, nin, ing, ng>, runn, unnn, and longer ones up to ning>. These n-grams capture morphological patterns and internal structure, allowing shared subwords (e.g., "ing" in running/jumping) to contribute similar semantic signals.