

Machine Learning (CE 40717)  
Fall 2025

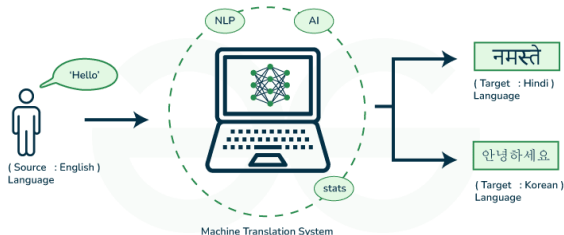
CE Department  
Sharif University of Technology



### 3 References



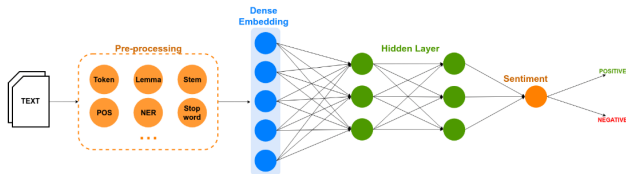
- Language is central to human interaction; many of our daily activities revolve around text and language.
- Natural Language Processing (NLP) enables computers to understand and generate human language.



- NLP helps translate text from one language to another.

Figure adapted from [geeksforgeeks.org/machine-translation-of-languages-in-artificial-intelligence/](https://www.geeksforgeeks.org/machine-translation-of-languages-in-artificial-intelligence/)

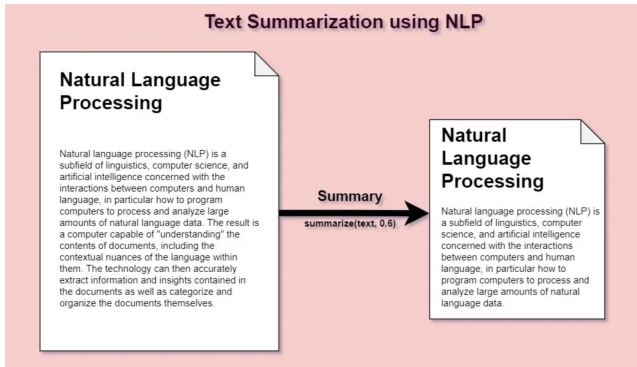
# Sentiment Analysis



- Determines the sentiment (e.g., positive or negative) expressed in a text.

Figure adapted from [www.mdpi.com/2079-9292/9/3/483](http://www.mdpi.com/2079-9292/9/3/483)

## Text Summarization using NLP



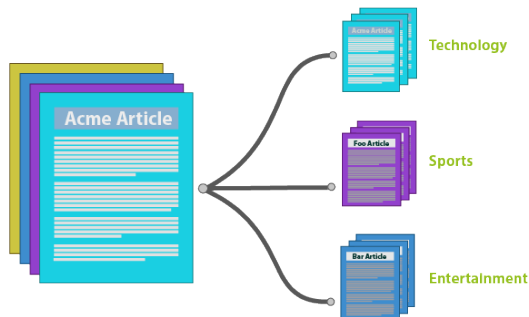
- Automatically generates a concise summary of longer text.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

\_\_\_\_\_

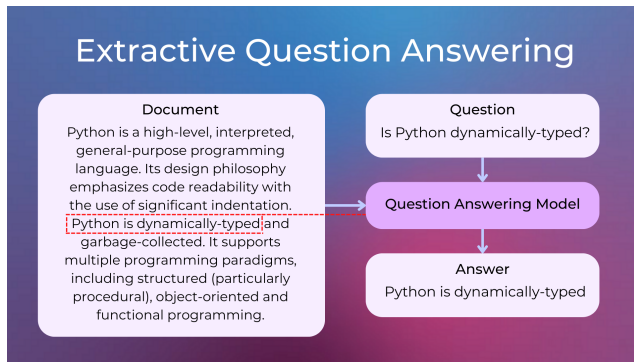






- Categorizes text into predefined groups or topics.

## Question Answering



- Answers questions based on a given text or dataset.

Figure adapted from [nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/17-question-answering](http://nlplanet.org/course-practical-nlp/02-practical-nlp-first-tasks/17-question-answering)



- To process text effectively, the first step is to represent words in a way that models can understand.
- We need to transform words into vectors or dense representations to capture their meaning and relationships.
- This is crucial for enabling machines to understand and use language as humans do.

## Motivation and One-Hot Encoding

- Traditional models like one-hot encoding lack semantic understanding and fail to capture word relationships.
- One-hot encoding represents words as binary vectors where only one position is 1, making the representation sparse and non-semantic.
- We need dense, semantic word representations to improve natural language processing tasks.

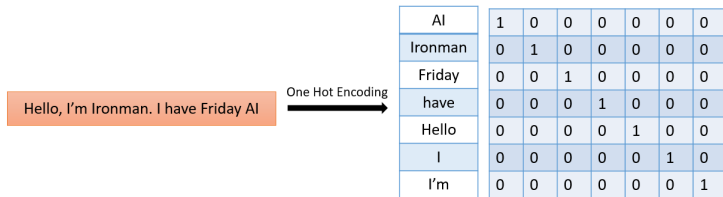


Figure adapted from [medium.com/@kalyan45/natural-language-processing-one-hot-encoding-5b31f76b09a0](https://medium.com/@kalyan45/natural-language-processing-one-hot-encoding-5b31f76b09a0)

1

- For example, given a vocabulary of 5 words:
  - apple = [1, 0, 0, 0, 0]
  - banana = [0, 1, 0, 0, 0]
  - cherry = [0, 0, 1, 0, 0]
  - date = [0, 0, 0, 1, 0]
  - elderberry = [0, 0, 0, 0, 1]
- The length of the one-hot vector depends on the number of unique words in the vocabulary.

- **Strengths:**
  - One-hot encoding is a simple and intuitive representation that can be effective in certain models, especially smaller neural networks.
  - It requires minimal computation and works well for small vocabularies or categorical features in simpler tasks.
- **Limitations:**
  - One-hot encoding does not capture semantic relationships, so similar words (e.g., `hotel` and `motel`) appear unrelated.
  - The vectors are sparse, containing mostly zeros, making it inefficient for large vocabularies and prone to overfitting due to the large number of parameters in downstream models.

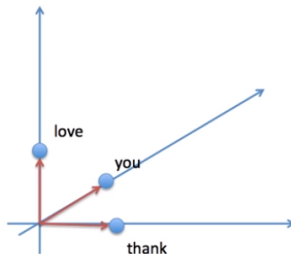


- **Cosine Similarity:**

$$\cos(\theta) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \|\mathbf{v}_2\|}$$

Downloaded from <http://ajphaphysocpharm.sagepub.com/> at 11:06 11 November 2014

unique word	encoding
thank	[1, 0, 0]
you	[0, 1, 0]
love	[0, 0, 1]



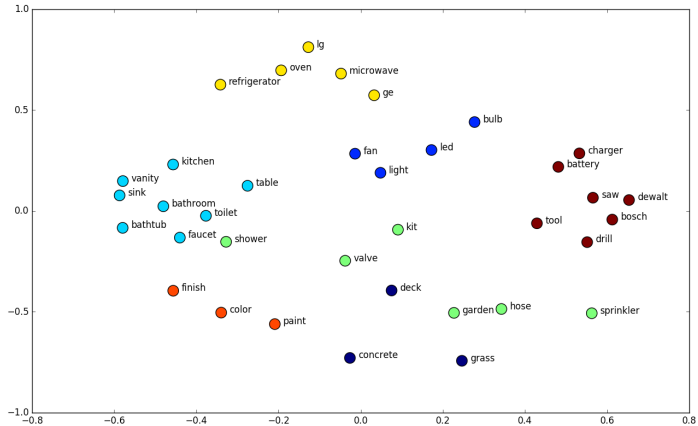
- Different words can be perpendicular to each other in space regardless of their meanings.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ≡ ↺ 🔍 ↻



- To process text data, we need to represent words in a form that a machine can understand—numerical vectors.
- Word2Vec uses a neural network to learn **word embeddings** that capture semantic similarities.
- These embeddings allow words with similar meanings to be represented by vectors close to each other in a high-dimensional space.

# Example: Word Embedding



- Word2Vec is a shallow neural network, with an input, hidden, and output layer.
- It takes in a target word and learns to predict either the surrounding context words or the target word from a set of context words.
- Through training, the network adjusts weights to create meaningful vector representations of words.

## Word2Vec as a Neural Network

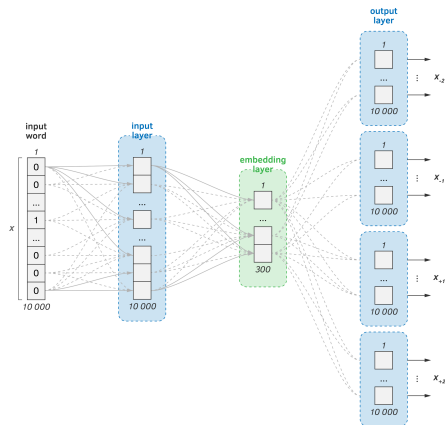


Figure adapted from <https://www.mlebook.com/wiki> chapter 4 draft

## Expected Outcome of Word2Vec

- Word2Vec aims to create a vector space where words with similar meanings or contexts are located close to each other.
- **Expected Result:** Semantically related words—such as “king” and “queen” or “dog” and “puppy”—should have similar vector representations.
- This proximity allows for various NLP tasks, such as:
  - **Synonym detection:** Identifying words with similar meanings.
  - **Analogy tasks:** Solving analogies by vector arithmetic (e.g., “king” - “man” + “woman”  $\approx$  “queen”).
  - **Clustering of concepts:** Grouping related concepts together in the embedding space.
- By representing words in this way, Word2Vec enables models to make use of semantic relationships between words.



# Expected Outcome of Word2Vec



Hypothetical features to understand word embeddings

# Word2Vec: Contextual Word Representation

- The core idea is based on distributional semantics: "You shall know a word by the company it keeps."
- Word2Vec uses two main algorithms for learning word vectors:
  - Continuous Bag of Words (CBOW)
  - Skip-gram Model

## ② Word2Vec

## Continuous Bag of Words (CBOW)

## Skip-gram Model

## Word Embedding Visualization

## Word Analogy

### 3 References

# CBOW: How It Works

- CBOW predicts the target word using the context (surrounding words) in a fixed window.
- For each word in the corpus, CBOW takes a set of context words and predicts the center word.
- Example: Given the context words {"the", "brown", "fox", "over"}, CBOW predicts the center word "jumps."
- CBOW tends to perform better on smaller datasets and is computationally more efficient.

## Example: Continuous Bag of Words

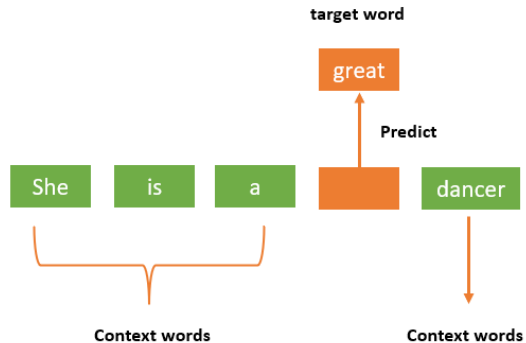
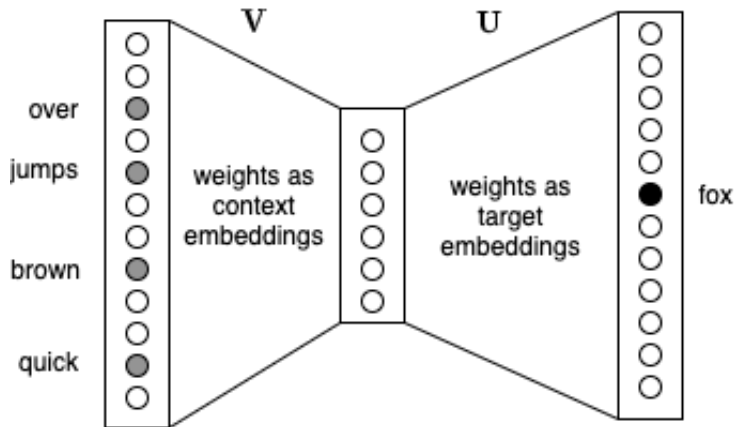


Figure adapted from [geeksforgeeks.org/continuous-bag-of-words-cbow-in-nlp/](https://www.geeksforgeeks.org/continuous-bag-of-words-cbow-in-nlp/)

# Continuous Bag of Words as a Neural Network



## ① Introduction

## ② Word2Vec

Continuous Bag of Words (CBOW)

**Skip-gram Model**

Word Embedding Visualization

Word Analogy

## ③ References

# Skip-gram: How It Works

- Skip-gram is the reverse of CBOW. It predicts the surrounding context words given a target word.
- For each word  $w_t$ , the model predicts the words in the window of size  $m$  around it (e.g., words  $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$ ).
- Example: If the center word is “jumps,” Skip-gram predicts the context words “the,” “brown,” “fox,” and “over.”
- Skip-gram is better suited for larger datasets and can capture rare words more effectively.



# Why Skip-gram?

- Skip-gram is preferred for large datasets because it handles rare words more effectively than CBOW.
- It captures detailed information about the surrounding words, leading to better word representations in the vector space.
- Word2Vec embeddings from Skip-gram have been widely adopted in various NLP tasks such as machine translation, sentiment analysis, and document classification.

# Skip-gram Example

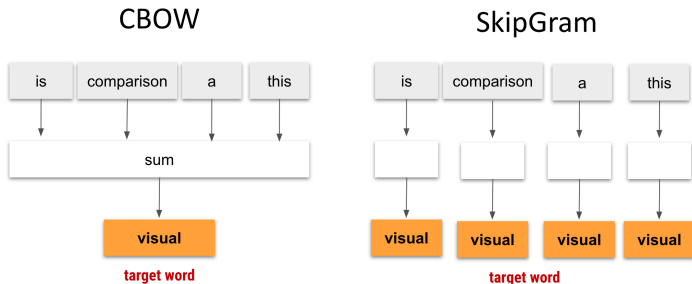
Window Size	Text	Skip-grams
2	[ The <b>wide</b> road shimmered ] in the hot sun.	wide, the wide, road wide, shimmered
	The [ wide road <b>shimmered</b> in the ] hot sun.	shimmered, wide shimmered, road shimmered, in shimmered, the
	The wide road shimmered in [ the hot <b>sun</b> ].	sun, the sun, hot
3	[ The <b>wide</b> road shimmered in ] the hot sun.	wide, the wide, road wide, shimmered wide, in
	[ The wide road <b>shimmered</b> in the hot ] sun.	shimmered, the shimmered, wide shimmered, road shimmered, in shimmered, the shimmered, hot
	The wide road shimmered [ in the hot <b>sun</b> ].	sun, in sun, the sun, hot

Different window sizes and samples drawn from context words and their target

# CBOW vs. Skip-gram

- CBOW and Skip-gram are the two primary architectures for Word2Vec.
- **CBOW** predicts a target word given its surrounding context, making it efficient and effective for smaller datasets.
- **Skip-gram**, on the other hand, predicts the surrounding words for a given target word. It is well-suited for larger datasets and can handle rare words more effectively.
- In essence, the Skip-gram model captures more detailed word relationships and is robust in large vocabularies.

# CBOW vs. Skip-gram



By: Kavita Ganesan

This is a visual comparison

## Difference between CBOW and Skip-gram

# Skip-gram: Objective Function

- The Skip-gram model learns word representations by predicting context words  $w_o$  given a center word  $w_c$ .
- Each word has two embeddings:
  - **Input (center) embedding:**  $\mathbf{v}_w$
  - **Output (context) embedding:**  $\mathbf{u}_w$
- The conditional probability of a context word is:

$$P(w_o | w_c) = \frac{\exp(\mathbf{u}_{w_o}^\top \mathbf{v}_{w_c})}{\sum_{x \in V} \exp(\mathbf{u}_x^\top \mathbf{v}_{w_c})}$$

- Intuition: words that co-occur should have large dot products.

# Skip-gram as a 2-Layer Neural Network

- Word2Vec is a shallow neural network:

$$\text{one-hot}(w_c) \rightarrow \mathbf{W}_{\text{in}} \rightarrow \mathbf{h} \rightarrow \mathbf{W}_{\text{out}} \rightarrow \text{softmax}$$

- Weight matrices:

$$\mathbf{W}_{\text{in}} \in \mathbb{R}^{|V| \times d}, \quad \mathbf{W}_{\text{out}} \in \mathbb{R}^{d \times |V|}$$

- Because the input is one-hot:

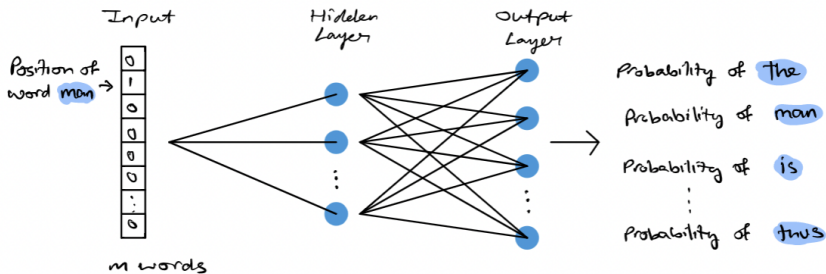
$$\mathbf{h} = \mathbf{W}_{\text{in}}^{\top} \mathbf{x} = \mathbf{v}_{w_c}$$

- **Key point:**

- $\mathbf{v}_w$  = row of  $\mathbf{W}_{\text{in}}$
- $\mathbf{u}_w$  = column of  $\mathbf{W}_{\text{out}}$

- Updating embeddings = updating network weights.

# Skip-gram as a Neural Network



Skip-gram Model as Neural Network

Figure adapted from [towardsdatascience.com/word2vec-explained-49c52b4ccb71](https://towardsdatascience.com/word2vec-explained-49c52b4ccb71)

# Skip-gram: Loss Function

- Training maximizes likelihood of observed center–context pairs.
- Equivalent objective (negative log-likelihood):

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t)$$

- $T$ : corpus size,  $m$ : window size.



# Skip-gram: Gradient Calculation

- For one training pair  $(w_c, w_o)$ :

$$\log P(w_o | w_c) = \mathbf{u}_{w_o}^\top \mathbf{v}_{w_c} - \log \sum_x \exp(\mathbf{u}_x^\top \mathbf{v}_{w_c})$$

- Gradient w.r.t. center embedding:

$$\frac{\partial \log P(w_o | w_c)}{\partial \mathbf{v}_{w_c}} = \mathbf{u}_{w_o} - \sum_x P(w_x | w_c) \mathbf{u}_x$$

- Same result as backprop through  $\mathbf{W}_{\text{in}}$ .

# Skip-gram: Gradient Update (Intuition)

- Gradient ascent update:

$$\mathbf{v}_{w_c} \leftarrow \mathbf{v}_{w_c} + \eta \left( \mathbf{u}_{w_o} - \sum_x P(w_x | w_c) \mathbf{u}_x \right)$$

- Interpretation:
  - Pull center word toward true context word
  - Push it away from other words
- Output embeddings  $\mathbf{u}_x$  are updated analogously.

# Skip-gram Example with Gradient Intuition

- Sentence: “Cats chase mice in the dark night.”
- Center word:  $w_c = \text{“chase”}$
- Context word:  $w_o = \text{“cats”}$
- Example embeddings (2D for illustration):

$$\mathbf{v}_{\text{chase}} = [0.1, 0.2], \quad \mathbf{u}_{\text{cats}} = [0.3, 0.4]$$

- Dot product:

$$\mathbf{u}_{\text{cats}}^\top \mathbf{v}_{\text{chase}} = 0.11$$

- If  $P(\text{cats} | \text{chase}) = 0.493$ , the error is:

$$1 - 0.493 = 0.507$$

- The update moves  $\mathbf{v}_{\text{chase}}$  closer to  $\mathbf{u}_{\text{cats}}$ .

# Skip-gram Pseudocode

---

## Algorithm 1 Skip-gram Model

---

**Require:** Corpus  $D$ , window size  $w$ , embedding dimension  $d$ , learning rate  $\eta$ , epochs  $n$

Initialize input embeddings  $\mathbf{V}$  and output embeddings  $\mathbf{U}$  randomly

**for** epoch = 1 to  $n$  **do**

**for** each sentence  $S$  in  $D$  **do**

**for** each center word  $w_c$  in  $S$  **do**

**for** each context word  $w_o$  within window  $w$  **do**

                Compute score  $\mathbf{u}_{w_o}^\top \mathbf{v}_{w_c}$

                Compute  $P(w_o | w_c)$  using softmax

                Update  $\mathbf{v}_{w_c}$  and  $\mathbf{u}_{w_o}$  using gradients

**end for**

**end for**

**end for**

**end for**

**return** input embeddings  $\mathbf{V}$

---

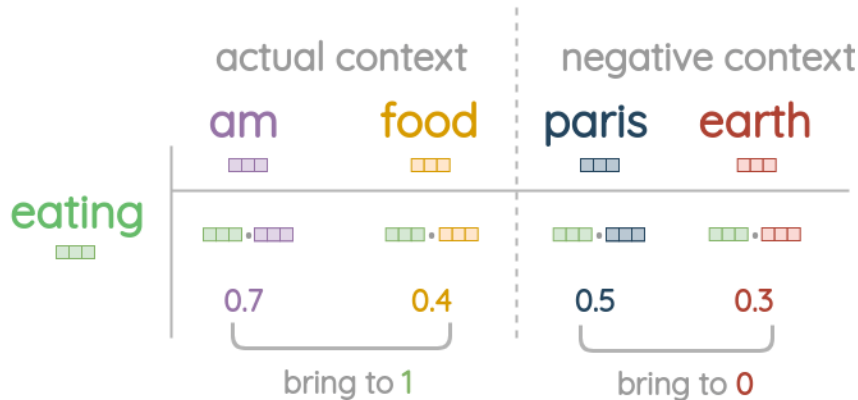
# Why Do We Need Negative Sampling?

- The softmax denominator sums over the entire vocabulary  $V$ .
- For large vocabularies, this is computationally expensive.
- **Key idea:** update only a few words instead of all words.
- **Negative sampling** provides an efficient approximation.

# Skip-gram: Negative Sampling

- For each positive pair  $(w_c, w_o)$ , sample  $k$  negative words.
- The objective becomes binary classification:
  - Maximize similarity for true context words
  - Minimize similarity for randomly sampled words
- Example:
  - Center word: “cat”
  - Positive context: “cute”
  - Negative samples: “table”, “sky”, “computer”

# Skip-gram: Negative Sampling



Negative sampling

## ① Introduction

## ② Word2Vec

Continuous Bag of Words (CBOW)

Skip-gram Model

**Word Embedding Visualization**

Word Analogy

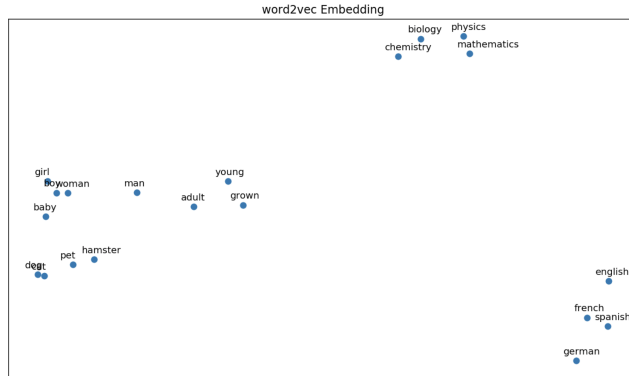
## ③ References



# Visualizing Words in 2D

- After training the model, words are mapped into a high-dimensional vector space.
- Using techniques like PCA or t-SNE, these vectors can be reduced to 2D for visualization, where similar words appear closer together.

# Visualizing Words in 2D



### Words represented in a 2D space after dimensionality reduction

## ① Introduction

## ② Word2Vec

Continuous Bag of Words (CBOW)

Skip-gram Model

Word Embedding Visualization

Word Analogy

## ③ References

# Word Analogy: Vector Arithmetic in Word2Vec

- Word2Vec embeddings can solve analogy tasks by performing vector arithmetic.
- The analogy task takes the form:

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

- The analogy is solved by finding the word vector closest to  $\mathbf{v}_{king} - \mathbf{v}_{man} + \mathbf{v}_{woman}$ .

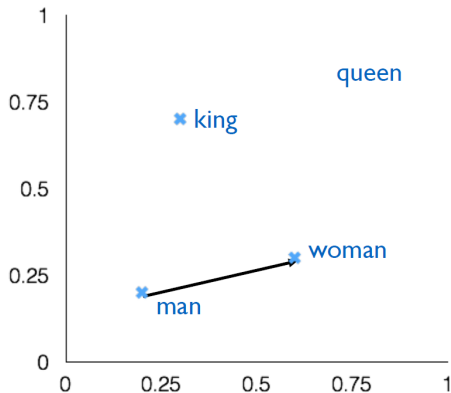
# Word Analogy Formula

- The formal formula for solving word analogies is:

$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

- Definitions:
  - $x_a$ : Vector for the first word (e.g., "man").
  - $x_b$ : Vector for the second word (e.g., "woman").
  - $x_c$ : Vector for the third word (e.g., "king").
  - $x_i$ : Candidate word vectors in the vocabulary.
- Example: To solve "man is to woman as king is to ?", the formula computes a vector closest to  $x_b - x_a + x_c$  (e.g., "queen").





### Word analogy example

### 3 References



- **These slides were prepared with contribution from:**
  - Omid Daliran
  - Armin Geramirad

- [1] M. Soleymani, "Machine learning." Sharif University of Technology.
- [2] M. Soleymani, "Modern information retrieval." Sharif University of Technology.
- [3] C. Manning, "Natural language processing with deep learning." Stanford.
- [4] E. Asgari, "Natural language processing." Sharif University of Technology.