

## Lecture 1 - Intro and Word Vectors

Language has been a major factor in human advancement. It allows us to think, communicate, and plan in complex ways. As Herb Clark puts it, “The common misconception is that language use primarily concerns words and what they mean. It doesn't. It has primarily to do with **people and what they mean.**”

Today's search engines have evolved beyond just retrieving documents. For example, if you ask, “When did Kendrick Lamar's first album come out?” the system doesn't just return links—it finds relevant documents, reads them, and extracts a direct answer. This makes it more of an *answer engine* than a search engine.

Modern systems often use multiple neural networks in this process:

1. A **retrieval model** finds passages related to your query.
2. A **reranking model** sorts them by relevance.
3. A **reader model** analyzes the top passages and synthesizes an accurate answer.

You Only Need One Model for Open-domain Question Answering Paper

<https://arxiv.org/pdf/2112.07381.pdf>.

**Foundation models** are a generalized form of large language models that use the same underlying technology across different types of data—such as images, audio, bioinformatics (like DNA and RNA), seismic waves, and other signals. The idea is to *build large, versatile models that can handle various modalities using a common approach.*

### How do we represent the meaning of a word?

Definition: **meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

**Commonest linguistic way of thinking of meaning:**




signifier (symbol)  $\Leftrightarrow$  signified (idea or thing)

= denotational semantics

tree  $\Leftrightarrow$  {, , , ...}

In linguistics, **meaning** is commonly understood as a connection between two parts:

- the **signifier** (the symbol or word), and
- the **signified** (the idea or object the word refers to).

This relationship is known as **denotational semantics**—where a word *denotes* or directly refers to a specific concept or thing. For example, the word "**tree**" (signifier) represents the idea or image of various types of trees (signified), like , , , etc.

This same principle of **denotation** is also applied in programming languages. In that context, symbols like `while`, `if`, or variable names have specific meanings or behaviors, which are considered their **denotations**—i.e., the exact function or role they represent in code.

*(Note: Denotation is the literal meaning of a word, as opposed to any emotional or cultural connotations it might carry.)*

## **Words as Discrete Symbols (One-Hot Encoding):**

### **Localist Representation**

- Words treated as **discrete, independent symbols** (e.g., *hotel*, *motel*).
- Represented using **one-hot vectors**:

### **Problems with One-Hot Vectors**

- **No semantic similarity:**
  - *motel* and *hotel* are similar in meaning but represented by **orthogonal vectors** (no overlap).
- **Cannot capture word relationships.**
  - Makes tasks like **web search, translation, and QA** less effective.

### **Solution: Word Vectors**

- Idea: Replace one-hot vectors with **dense, learned representations** that encode **semantic similarity**.
- Leads to development of models like **Word2Vec**.

**Distributional semantics:** A word's meaning is given by the words that frequently appear close by.

"You shall know a word by the company it keeps." – J.R. Firth

### **Core Idea**

- A word's **meaning** is derived from the **words that frequently appear near it**.
- Meaning is captured by analyzing a word's **context** (neighboring words in a fixed-size window).

## How It Works

- For each occurrence of a word **w** in text:
  - Its **context** = nearby words within a window (e.g.,  $\pm 2$  words).
- We collect **statistics** of these contexts across the corpus.
- These are used to create a **dense vector representation** for the word.

## Goal

- Learn **dense, low-dimensional vectors** (vs. sparse one-hot).
- Vectors reflect **semantic similarity** and **contextual usage**.

## Word Vectors:

### Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector **dot** (scalar) **product**

$$\text{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \text{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

Note: **word vectors** are also called **(word) embeddings** or **(neural) word representations**  
 They are a **distributed** representation

## Core Idea

- **Words that appear in similar contexts** have **similar meanings** and hence **similar vectors**.
- Similarity is measured using the **dot product** of vectors.

## Word Embeddings

- Each word is represented as a **dense vector** in a **high-dimensional space**.

## Properties

- Vectors that are **close** (high dot product) represent **semantically similar** words.
- These vectors are called:
  - **Word vectors** or **Word embeddings** or **Neural word representations**
- They are a form of **distributed representation** (vs. localist/one-hot).

## Advantages

- Able to represent **semantic similarity**.
- Can encode **multiple senses** of a word based on context.
- Supports **generalization and reasoning** in NLP models.

**Q: Why do the entries in word vectors seem to be between -1 and 1? Are they bounded?**

❓ **No, they are not inherently bounded** between -1 and 1.

❓ The values in word vectors are **learned parameters** — during training (e.g., via Word2Vec), **there are no strict bounds**.

❓ However:

- **Regularization** (like L2) is often used during training to **prevent values from growing too large**.
- **Length normalization** (scaling vectors to have unit length) is sometimes applied, especially when comparing vectors using **cosine similarity**.

❓ The reason many values look small is because:

- Regularization tends to keep them small.
- We're training embeddings in a way that produces **meaningful semantic relationships**.

The goal isn't about bounding values but **capturing semantics** — i.e., words with **similar meanings are close** in vector space.

## Word2Vec:

**Word2Vec** is a method developed by Mikolov et al. (2013) to learn **dense vector representations of words**, also known as **word embeddings**.

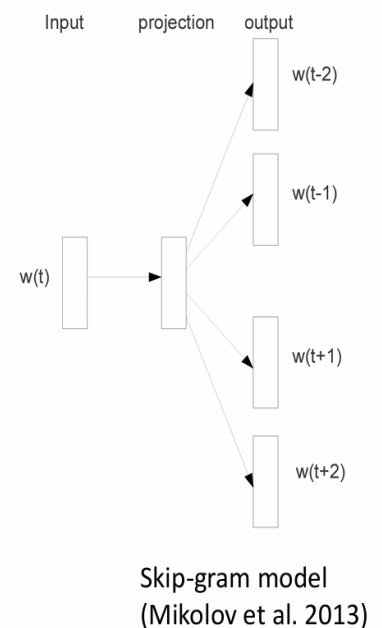
- These vectors capture **semantic meaning** of words.
- Words with **similar meanings** end up having **similar vectors**.
- Word2Vec learns this from a large corpus by examining **how words appear in context**.

### 3. Word2vec: Overview

**Word2vec** is a framework for learning word vectors (Mikolov et al. 2013)

Idea:

- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a **vector**
- Go through each position  $t$  in the text, which has a center word  $c$  and context ("outside") words  $o$
- Use the **similarity of the word vectors** for  $c$  and  $o$  to **calculate the probability** of  $o$  given  $c$  (or vice versa)
- **Keep adjusting the word vectors** to maximize this probability



### Key Idea: *Distributional Semantics*

"You shall know a word by the company it keeps." — J.R. Firth

Instead of one-hot vectors (which are sparse and don't encode any relationships), Word2Vec uses the **contexts around words** to learn dense embeddings. For example:

- "banking" might appear with words like "money", "loan", "regulation".
- These contexts help Word2Vec learn the *meaning* of "banking".

## How It Works: Word2Vec Models

### 1. Skip-gram Model

- Goal: **Given a center word, predict its context words.**
- Input: center word → Output: surrounding words (within a window).

### Example:

Sentence: *"The cat sits on the mat."*

If **"sits"** is the center word and window size is 2, the context is:  
["The", "cat", "on", "the"]

We train the model to predict these context words from "sits".

Mathematically, the objective is to **maximize the probability** of context words given the center word:

$$P(W_{\text{context}} / W_{\text{center}})$$

This is done using a **softmax function** over the dot product of the word vectors.

## 2. CBOW (Continuous Bag of Words) Model

- Goal: **Given context words, predict the center word.**
- Input: context words → Output: center word.

### Example:

Sentence: *"The cat sits on the mat."*

If context = ["The", "cat", "on", "the"], CBOW tries to predict "sits".

Here, the model **averages** the vectors for context words and predicts the center word.

## Training Objective

Word2Vec uses the **negative log likelihood** as its loss function.

In simple terms:

- We want to **maximize** the probability of correct predictions.
- This is equivalent to **minimizing** the negative log likelihood:

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

Where:

- $T$  is the number of words in the corpus.
- $m$  is the context window size.
- $\theta$  are the model parameters (word vectors).

### Technical Notes:

The learned word vectors **are not bounded**, but regularization can help keep them small.

Some approaches **normalize** the vectors (e.g., length = 1).

Regularization techniques like L2 norm are commonly used.

### Outcome:

Word2Vec gives us **meaningful embeddings**, such that:

- $\text{vec}(\text{"king"}) - \text{vec}(\text{"man"}) + \text{vec}(\text{"woman"}) \approx \text{vec}(\text{"queen"})$
- Words like "money", "cash", "finance" end up close in vector space.

### Word2vec: objective function

For each position  $t = 1, \dots, T$ , predict context words within a window of fixed size  $m$ , given center word  $w_t$ . Data likelihood:

Likelihood =  $L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$

$\theta$  is all variables to be optimized

sometimes called a *cost* or *loss* function

The **objective function**  $J(\theta)$  is the (average) negative log likelihood:

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

Minimizing objective function  $\Leftrightarrow$  Maximizing predictive accuracy

## Word2vec: objective function

- We want to minimize the objective function:

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

- Question:** How to calculate  $P(w_{t+j} | w_t; \theta)$ ?
- Answer:** We will use two vectors per word  $w$ :
  - $v_w$  when  $w$  is a center word
  - $u_w$  when  $w$  is a context word
- Then for a center word  $c$  and a context word  $o$ :

} These word vectors are subparts of the big vector of all parameters  $\theta$

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

## Word2vec: prediction function

- ② Exponentiation makes anything positive

① Dot product compares similarity of  $o$  and  $c$ .  
 $u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$   
Larger dot product = larger probability

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function**  $\mathbb{R}^n \rightarrow (0,1)^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values  $x_i$  to a probability distribution  $p_i$

- “max” because amplifies probability of largest  $x_i$
- “soft” because still assigns some probability to smaller  $x_i$
- Frequently used in Deep Learning

But sort of a weird name because it returns a distribution!

Here, we have considered we have vectors for each word. Using these vectors, we can then calculate probabilities. But **where do the vectors come from?**

Answer:

The word vectors are learned by framing it as an **optimization problem**: we start with random vectors and adjust them using a large text corpus to maximize the



probability of actual context words, so words appearing in similar contexts end up with similar vectors.

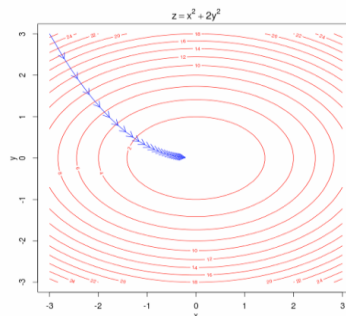
Word vectors (embeddings) in **Word2Vec** are **not pre-defined** — they are **learned** through an **optimization process** during training on a large text corpus.

## To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall:  $\theta$  represents **all** the model parameters, in one long vector
- In our case, with  $d$ -dimensional vectors and  $V$ -many words, we have  $\rightarrow$
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute **all** vector gradients!

31

Here, our parameters for our model are the concatenation of all the word vectors.

For each word, we assume two vectors. We assume one vector when they're the center word and one vector when they're the outside word.

So if we say had 100 dimensional vectors, we'll have 100 parameters for 'aardvarks' and an outside word.

100 parameters for "a" as an outside word, all the way through to 100 parameters for zebras and outside word. Then we'd have 100 parameters for aardvark as a center word continuing down. So if we had a vocabulary of 400,000 words and 100 dimensional word vectors that means we'd have 400,000 times 2 are 800,000 times 100. We'd have 80 million parameters.