

CS 489: Introduction to Natural Language Processing

Lecture 3: Distributional Semantics and Word Embeddings

Instructor: Freda Shi

September 17, 2026



**UNIVERSITY OF
WATERLOO**

Outline

1. Motivation: Why do we need word representations?
2. The distributional hypothesis
3. Count-based methods
 - Word-word co-occurrence matrices
 - TF-IDF
 - Pointwise mutual information (PMI / PPMI)
 - Dimensionality reduction with SVD
4. Prediction-based methods: Word2Vec
 - Skip-gram
 - CBOW
 - Negative sampling
5. Evaluating word embeddings

Representing Meaning by Hand: WordNet

First attempt: ask linguists to hand-enumerate each word's **features** and **relations** to other words.

WordNet (Miller, 1995): a large lexical database of English.

- Words grouped into **synsets** (synonym sets), linked by semantic relations.
- IS-A chain: dog → canine → carnivore → mammal → animal.

Representing Meaning by Hand: WordNet

First attempt: ask linguists to hand-enumerate each word's **features** and **relations** to other words.

WordNet (Miller, 1995): a large lexical database of English.

- Words grouped into **synsets** (synonym sets), linked by semantic relations.
- IS-A chain: dog → canine → carnivore → mammal → animal.

Manual enumeration is **very hard**:

- Labor-intensive and subjective: decades of expert effort, yet experts disagree.
- Never complete: new words and senses keep appearing.
- No graded similarity: is doctor closer to philosophy or to hospital?

[Miller (1995). WordNet: A lexical database for English. *CACM* 38(11).]

How Should We Represent Words?

Goal: We want a representation of words that captures their **meaning**.

- Words with similar meanings should have similar representations.
- The representation should capture relationships between words.
- It should be learnable from data (no manual feature engineering).
- It should be usable as input to downstream models.

One-Hot Representations

The simplest approach: represent each word as a one-hot vector.

One-Hot Representations

The simplest approach: represent each word as a one-hot vector.

Given vocabulary $V = \{w_1, w_2, \dots, w_{|V|}\}$, word w_i is represented as:

$$\mathbf{e}_i \in \{0, 1\}^{|V|}, \quad (\mathbf{e}_i)_j = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

One-Hot Representations

The simplest approach: represent each word as a one-hot vector.

Given vocabulary $V = \{w_1, w_2, \dots, w_{|V|}\}$, word w_i is represented as:

$$\mathbf{e}_i \in \{0, 1\}^{|V|}, \quad (\mathbf{e}_i)_j = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

Example ($|V| = 5$):

cat	[1, 0, 0, 0, 0]
dog	[0, 1, 0, 0, 0]
math	[0, 0, 1, 0, 0]

One-Hot Representations

The simplest approach: represent each word as a one-hot vector.

Given vocabulary $V = \{w_1, w_2, \dots, w_{|V|}\}$, word w_i is represented as:

$$\mathbf{e}_i \in \{0, 1\}^{|V|}, \quad (\mathbf{e}_i)_j = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

Example ($|V| = 5$):

cat	[1, 0, 0, 0, 0]
dog	[0, 1, 0, 0, 0]
math	[0, 0, 1, 0, 0]

Problem: $\mathbf{e}_i^\top \mathbf{e}_j = 0$ for all $i \neq j$. Every word is equally distant from every other word!

What We Really Want: Dense Representations

Instead of sparse, high-dimensional one-hot vectors, we want **dense** vectors in $\mathbb{R}^d$ (typically $d \in [50, 300]$).

What We Really Want: Dense Representations

Instead of sparse, high-dimensional one-hot vectors, we want **dense** vectors in $\mathbb{R}^d$ (typically $d \in [50, 300]$).

Dense vectors (a.k.a. **word embeddings**):

- Each dimension encodes some latent “feature” of meaning.
- Similar words $\rightarrow$ nearby vectors (high cosine similarity).
- Compact: $d \ll |V|$.
- Learnable from large text corpora.

What We Really Want: Dense Representations

Instead of sparse, high-dimensional one-hot vectors, we want **dense** vectors in $\mathbb{R}^d$ (typically $d \in [50, 300]$).

Dense vectors (a.k.a. **word embeddings**):

- Each dimension encodes some latent “feature” of meaning.
- Similar words $\rightarrow$ nearby vectors (high cosine similarity).
- Compact: $d \ll |V|$.
- Learnable from large text corpora.

Example (illustrative values):

cat	[0.7, -0.2, 0.1, ...]	
dog	[0.6, -0.1, 0.3, ...]	$\leftarrow$ close to cat!
math	[0.3, 0.8, -0.5, ...]	

The Distributional Hypothesis

*“You shall know a word
by the company it keeps.”*

– J. R. Firth (1957)



Key idea: Words that appear in **similar contexts** tend to have **similar meanings**.

The Distributional Hypothesis

*“You shall know a word
by the company it keeps.”*

– J. R. Firth (1957)



Key idea: Words that appear in **similar contexts** tend to have **similar meanings**.

Anticipated earlier by Wittgenstein:

“The meaning of a word is its use in the language.”

– Ludwig Wittgenstein, *Philosophical Investigations* (1953, §43)

Example: An Unknown Word in Context

Suppose you encounter an unknown word: **tezgüino**.

You observe it in these contexts:

- “A bottle of **tezgüino** is on the table.”
- “Everyone likes **tezgüino**.”
- “**Tezgüino** makes you drunk.”
- “We brew **tezgüino** from corn.”

Example: An Unknown Word in Context

Suppose you encounter an unknown word: **tezgüino**.

You observe it in these contexts:

- “A bottle of **tezgüino** is on the table.”
- “Everyone likes **tezgüino**.”
- “**Tezgüino** makes you drunk.”
- “We brew **tezgüino** from corn.”

From context alone, you can infer that **tezgüino** is **an alcoholic drink**.

(Tezgüino is indeed a corn beer brewed by the Tarahumara people of Mexico.)

⇒ Context provides strong signal about word meaning, even for unknown words.

Distributional Semantics: Which Words Fit **tezgüino**'s Contexts?

Contexts observed:

C1. “A bottle of ____ is on the table”

C2. “Everyone likes ____”

C3. “____ makes you drunk”

C4. “We brew ____ from corn”

Distributional Semantics: Which Words Fit **tezgüino**'s Contexts?

Contexts observed:

C1. "A bottle of ____ is on the table"

C2. "Everyone likes ____"

C3. "____ makes you drunk"

C4. "We brew ____ from corn"

wine and *tezgüino* keep almost the **same company** $\Rightarrow$ similar meanings.

loud fits nowhere $\Rightarrow$ unrelated.

	fits in context?				shared with <i>tezgüino</i>
	C1	C2	C3	C4	
tezgüino	1	1	1	1	4
loud	0	0	0	0	0
motor oil	1	0	0	1	2
tortillas	0	1	0	1	1
choices	0	1	0	0	1
wine	1	1	1	0	3

Roadmap: Count-Based vs. Prediction-Based

Two families of methods for building word vectors:

Count-based (this section):

- Build a co-occurrence matrix from a corpus.
- Apply weighting (TF-IDF, PMI).
- Reduce dimensionality (SVD).
- Rows of the reduced matrix = word vectors.

Prediction-based (next section):

- Train a model to predict context words from a target word (or vice versa).
- The learned parameters = word vectors.
- Word2Vec (Mikolov et al., 2013).

Word–Word Co-occurrence Matrix

Idea: Count how often words co-occur within a **context window**.

Context window of size k : the k words before and after a target word.

Word-Word Co-occurrence Matrix

Idea: Count how often words co-occur within a **context window**.

Context window of size k : the k words before and after a target word.

Example corpus: "I like deep learning. I like NLP. I enjoy NLP."

Window size $k = 1$:

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

Context Window Size

The choice of window size k affects what kind of similarity is captured:

Small window ($k = 1-3$):

- Captures **syntactic** similarity.
- Words in the same syntactic role are similar.
- E.g., “big” $\approx$ “large” $\approx$ “huge”.

Large window ($k = 5-10$):

- Captures **topical** similarity.
- Words in the same topic/domain are similar.
- E.g., “baseball” $\approx$ “pitcher” $\approx$ “stadium”.

In practice, $k \in [2, 10]$ is typical. Often $k = 5$.

Measuring Similarity: Dot Product and Cosine

Given two word vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$:

Dot product:

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^d u_i v_i$$

Measures similarity but is sensitive to vector magnitude.

Measuring Similarity: Dot Product and Cosine

Given two word vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$:

Dot product:

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^d u_i v_i$$

Measures similarity but is sensitive to vector magnitude.

Cosine similarity:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} \in [-1, 1]$$

Normalized: it captures direction, not magnitude.

Measuring Similarity: Dot Product and Cosine

Given two word vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$:

Dot product:

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^d u_i v_i$$

Measures similarity but is sensitive to vector magnitude.

Cosine similarity:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} \in [-1, 1]$$

Normalized: it captures direction, not magnitude.

For word vectors, **cosine similarity** is almost always preferred.

- Words like “the” have large count vectors $\rightarrow$ high dot product with everything.
- Cosine corrects for this frequency effect.

Problems with Raw Counts

Raw co-occurrence counts are problematic:

- **High-frequency function words** (“the”, “a”, “of”) dominate counts.
- **Sparsity:** most entries are zero.
- **Dimensionality:** vectors have $|V|$ dimensions (can be 50k+).
- Raw counts don't distinguish **informative** from **uninformative** co-occurrences.

Problems with Raw Counts

Raw co-occurrence counts are problematic:

- **High-frequency function words** (“the”, “a”, “of”) dominate counts.
- **Sparsity**: most entries are zero.
- **Dimensionality**: vectors have $|V|$ dimensions (can be 50k+).
- Raw counts don't distinguish **informative** from **uninformative** co-occurrences.

Solutions:

1. Re-weight counts → **TF-IDF** (word–document matrices) or **PMI** (word–word matrices).
2. Reduce dimensionality → **SVD**.

TF-IDF (Term Frequency–Inverse Document Frequency)

Originally designed for **information retrieval** (word–document matrices), but the intuition applies broadly.

Term frequency of word w in document d :

$$\text{tf}(w, d) = \log(1 + \text{count}(w, d))$$

TF-IDF (Term Frequency–Inverse Document Frequency)

Originally designed for **information retrieval** (word–document matrices), but the intuition applies broadly.

Term frequency of word w in document d :

$$\text{tf}(w, d) = \log(1 + \text{count}(w, d))$$

Inverse document frequency of word w :

$$\text{idf}(w) = \log \frac{N}{|\{d : w \in d\}|}$$

where N is the total number of documents.

TF-IDF (Term Frequency–Inverse Document Frequency)

Originally designed for **information retrieval** (word–document matrices), but the intuition applies broadly.

Term frequency of word w in document d :

$$\text{tf}(w, d) = \log(1 + \text{count}(w, d))$$

Inverse document frequency of word w :

$$\text{idf}(w) = \log \frac{N}{|\{d : w \in d\}|}$$

where N is the total number of documents.

TF-IDF weight:

$$\text{tf-idf}(w, d) = \text{tf}(w, d) \times \text{idf}(w)$$

TF-IDF (Term Frequency–Inverse Document Frequency)

Originally designed for **information retrieval** (word–document matrices), but the intuition applies broadly.

Term frequency of word w in document d :

$$\text{tf}(w, d) = \log(1 + \text{count}(w, d))$$

Inverse document frequency of word w :

$$\text{idf}(w) = \log \frac{N}{|\{d : w \in d\}|}$$

where N is the total number of documents.

TF-IDF weight:

$$\text{tf-idf}(w, d) = \text{tf}(w, d) \times \text{idf}(w)$$

Effect: Downweights words that appear in every document (“the”, “is”) and upweights distinctive words.

Pointwise Mutual Information (PMI)

For word–word co-occurrence, **PMI** is the standard reweighting method.

PMI measures whether two words co-occur more than expected by chance:

$$\text{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w) P(c)}$$

Pointwise Mutual Information (PMI)

For word–word co-occurrence, **PMI** is the standard reweighting method.

PMI measures whether two words co-occur more than expected by chance:

$$\text{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w) P(c)}$$

Let $N = \sum_{w', c'} \text{count}(w', c')$ be the total co-occurrence count. Then:

- $P(w, c) = \text{count}(w, c) / N$ (joint probability)
- $P(w) = \sum_{c'} \text{count}(w, c') / N$, $P(c) = \sum_{w'} \text{count}(w', c) / N$ (marginals)

Pointwise Mutual Information (PMI)

For word–word co-occurrence, **PMI** is the standard reweighting method.

PMI measures whether two words co-occur more than expected by chance:

$$\text{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w)P(c)}$$

Let $N = \sum_{w', c'} \text{count}(w', c')$ be the total co-occurrence count. Then:

- $P(w, c) = \text{count}(w, c) / N$ (joint probability)
- $P(w) = \sum_{c'} \text{count}(w, c') / N$, $P(c) = \sum_{w'} \text{count}(w', c) / N$ (marginals)
- $\text{PMI} > 0$: words co-occur *more* than expected.
- $\text{PMI} = 0$: words are independent.
- $\text{PMI} < 0$: words co-occur *less* than expected.

PMI in Action: What Co-occurs with “chicken”?

Top context words for “chicken” in a large web corpus:

By raw count:

Context word	Count
the	7853
and	6276
a	5213
of	3963
in	3282
to	2520
is	2339
with	2127

Dominated by high-frequency function words.

By PMI:

Context word	PMI
fried	10.2
pork	9.3
beef	9.0
soup	8.7
sauce	7.8
curry	7.7
cooked	7.6
lamb	7.5

Semantically related content words.

PMI surfaces **informative** associations that raw counts hide.

PMI at the Extremes

PMI values for “chicken” with selected context words (same corpus as before):

Largest PMI ($\gg 0$):

fried	10.2
pork	9.3
beef	9.0
soup	8.7
sauce	7.8
curry	7.7

PMI ≈ 0 :

climbed	0.003
detailing	0.003
turkish	0.002
productivity	0.001
swing	0.000
structures	-0.001

Smallest PMI (< 0):

users	-4.6
data	-4.6
discussion	-4.7
editors	-4.8
edits	-5.3
deletion	-6.1

- PMI > 0 : co-occur *more* than chance. PMI ≈ 0 : roughly *independent*.
- PMI < 0 : co-occur *less* than chance (*anticorrelated*), but unreliable with finite data.

Positive PMI (PPMI)

Problem: negative PMI values are unreliable for rare pairs (small counts → noisy estimates). **Fix:** clip at zero, giving **Positive PMI (PPMI)**:

$$\text{PPMI}(w, c) = \max(0, \text{PMI}(w, c))$$

PPMI is the **most common** weighting for word–word co-occurrence matrices.

PMI: Worked Example

Corpus (two sentences, counted individually):

I like deep learning.

I like NLP.

Window size $k = 1$, respecting sentence boundaries (as before). Total co-occurrence count = 10.

Compute PMI(like, deep):

$$P(\text{like, deep}) = \frac{1}{10}$$

$$P(\text{like}) = \frac{2 + 1 + 1}{10} = \frac{4}{10} = \frac{2}{5}$$

$$P(\text{deep}) = \frac{1 + 1}{10} = \frac{2}{10} = \frac{1}{5}$$

$$\text{PMI}(\text{like, deep}) = \log_2 \frac{1/10}{(2/5)(1/5)} = \log_2 \frac{1/10}{2/25} = \log_2 1.25 \approx 0.32$$

PMI > 0: “like” and “deep” co-occur slightly more than chance.

Dimensionality Reduction: Why?

Even after PPMI reweighting, the co-occurrence matrix $\mathbf{M} \in \mathbb{R}^{|V| \times |V|}$ is:

- **Very large:** $|V|$ can be 50,000+ $\rightarrow$ vectors of 50,000+ dimensions.
- **Very sparse:** most entries are zero.
- **Noisy:** rare co-occurrences may be unreliable.

Dimensionality Reduction: Why?

Even after PPMI reweighting, the co-occurrence matrix $\mathbf{M} \in \mathbb{R}^{|V| \times |V|}$ is:

- **Very large:** $|V|$ can be 50,000+ $\rightarrow$ vectors of 50,000+ dimensions.
- **Very sparse:** most entries are zero.
- **Noisy:** rare co-occurrences may be unreliable.

Solution: Reduce $|V|$ dimensions to d dimensions ($d \in [50, 300]$) while preserving the most important structure.

$\Rightarrow$ **Singular Value Decomposition (SVD).**

Singular Value Decomposition (SVD)

Any matrix $M \in \mathbb{R}^{m \times n}$ can be factored as:

$$M = U\Sigma V^T$$

where:

- $U \in \mathbb{R}^{m \times m}$: its columns are the **left singular vectors**; each row corresponds to a word.
- $\Sigma \in \mathbb{R}^{m \times n}$: diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq \dots \geq 0$.
- $V \in \mathbb{R}^{n \times n}$: its columns are the **right singular vectors**; each row corresponds to a context word.

The singular values σ_i tell us how much “variance” each dimension captures.

Truncated SVD

Key idea: Keep only the top d singular values/vectors:

$$\mathbf{M} \approx \mathbf{U}_d \mathbf{\Sigma}_d \mathbf{V}_d^\top$$

where $\mathbf{U}_d \in \mathbb{R}^{m \times d}$, $\mathbf{\Sigma}_d \in \mathbb{R}^{d \times d}$, $\mathbf{V}_d \in \mathbb{R}^{n \times d}$.

This is the **best rank- d approximation** to $\mathbf{M}$ (minimizes Frobenius norm of the error).

Truncated SVD

Key idea: Keep only the top d singular values/vectors:

$$M \approx U_d \Sigma_d V_d^T$$

where $U_d \in \mathbb{R}^{m \times d}$, $\Sigma_d \in \mathbb{R}^{d \times d}$, $V_d \in \mathbb{R}^{n \times d}$.

This is the **best rank- d approximation** to M (minimizes Frobenius norm of the error).

Word vectors: Take row i of U_d (or $U_d \Sigma_d^{1/2}$) as the vector for word i .

Truncated SVD

Key idea: Keep only the top d singular values/vectors:

$$\mathbf{M} \approx \mathbf{U}_d \mathbf{\Sigma}_d \mathbf{V}_d^\top$$

where $\mathbf{U}_d \in \mathbb{R}^{m \times d}$, $\mathbf{\Sigma}_d \in \mathbb{R}^{d \times d}$, $\mathbf{V}_d \in \mathbb{R}^{n \times d}$.

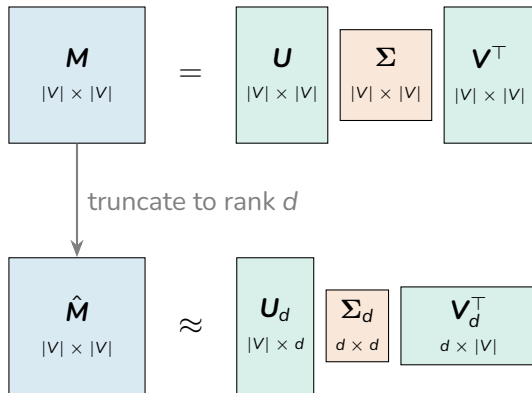
This is the **best rank- d approximation** to $\mathbf{M}$ (minimizes Frobenius norm of the error).

Word vectors: Take row i of $\mathbf{U}_d$ (or $\mathbf{U}_d \mathbf{\Sigma}_d^{1/2}$) as the vector for word i .

Benefits:

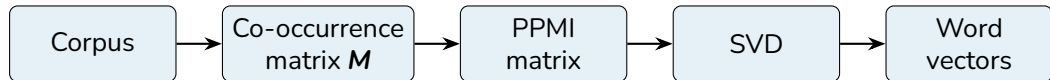
- Dense, low-dimensional vectors ($d \in [50, 300]$).
- Denoising: removes low-variance (noisy) dimensions.
- Captures latent semantic structure (“Latent Semantic Analysis”).

SVD Illustrated



Row i of U_d gives the d -dimensional embedding for word i .

The Full Pipeline: Count-Based Word Vectors



Steps:

1. Collect word–word co-occurrence counts (window size k).
2. Compute PPMI to reweight counts.
3. Apply truncated SVD to get d -dimensional vectors.
4. Use the resulting vectors for downstream tasks.

From Counting to Prediction

Count-based methods:

- Collect co-occurrence statistics over entire corpus.
- Apply matrix factorization.

Prediction-based methods (Word2Vec, Mikolov et al., 2013):

- Train a model to **predict context words** from a target word.
- The model parameters *are* the word embeddings.
- Never explicitly builds a co-occurrence matrix.

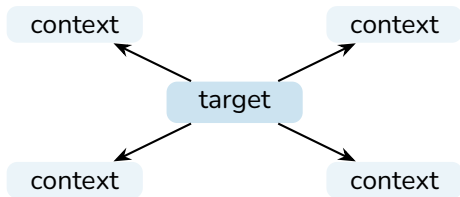
Prediction-based methods scale better and often produce higher-quality embeddings than count-based methods.

(Levy & Goldberg (2014) later showed the two are closely connected.)

Word2Vec: Two Architectures

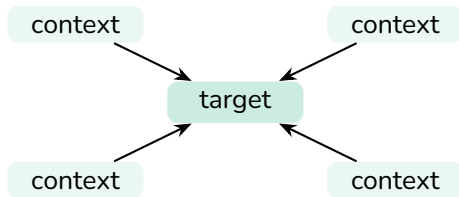
Skip-gram:

- Given the **target** word, predict the **context** words.
- Input: center word.
- Output: surrounding words.



CBOW (Continuous Bag of Words):

- Given the **context** words, predict the **target** word.
- Input: surrounding words.
- Output: center word.



Skip-Gram: Setup

Two embedding matrices:

- $\mathbf{W} \in \mathbb{R}^{|V| \times d}$: target (center) word embeddings.
- $\mathbf{U} \in \mathbb{R}^{|V| \times d}$: context word embeddings.

For target word w and context word c :

- $\mathbf{w} = \mathbf{W}[w] \in \mathbb{R}^d$ (row of $\mathbf{W}$)
- $\mathbf{u}_c = \mathbf{U}[c] \in \mathbb{R}^d$ (row of $\mathbf{U}$)

Skip-Gram: Setup

Two embedding matrices:

- $\mathbf{W} \in \mathbb{R}^{|V| \times d}$: target (center) word embeddings.
- $\mathbf{U} \in \mathbb{R}^{|V| \times d}$: context word embeddings.

For target word w and context word c :

- $\mathbf{w} = \mathbf{W}[w] \in \mathbb{R}^d$ (row of $\mathbf{W}$)
- $\mathbf{u}_c = \mathbf{U}[c] \in \mathbb{R}^d$ (row of $\mathbf{U}$)

Objective: For each position t in the corpus with center word w_t and context window $\{w_{t-k}, \dots, w_{t+k}\} \setminus \{w_t\}$:

$$\max \sum_{t=1}^T \sum_{\substack{-k \leq j \leq k \\ j \neq 0}} \log P(w_{t+j} \mid w_t)$$

Skip-Gram: Softmax Probability

The probability of context word c given target word w :

$$P(c \mid w) = \frac{\exp(\mathbf{u}_c^\top \mathbf{w})}{\sum_{c'=1}^{|V|} \exp(\mathbf{u}_{c'}^\top \mathbf{w})}$$

This is a **softmax** over the entire vocabulary.

Skip-Gram: Softmax Probability

The probability of context word c given target word w :

$$P(c \mid w) = \frac{\exp(\mathbf{u}_c^\top \mathbf{w})}{\sum_{c'=1}^{|V|} \exp(\mathbf{u}_{c'}^\top \mathbf{w})}$$

This is a **softmax** over the entire vocabulary.

Problem: The denominator sums over *all* $|V|$ words!

- $|V|$ can be 10^5 to 10^6 .
- Computing this for every training example is extremely expensive.
- $\mathcal{O}(|V|)$ per training step $\rightarrow$ impractical.

$\Rightarrow$ We need a more efficient training objective.

Skip-Gram: Training Walkthrough

Sentence: “The cat sat on the mat” (window $k = 2$)

Center word $w_t = \text{sat}$ (position 3):



Training pairs generated from this position:

Target (input)	Context (output)
sat	The
sat	cat
sat	on
sat	the

The window slides across every position in the corpus.

Negative Sampling

Key idea: Instead of softmax over all $|V|$ words, reformulate as a **binary classification** problem.

For each observed (target, context) pair (w, c) :

- **Positive example:** (w, c) is a real co-occurrence $\rightarrow$ label = 1.
- **Negative examples:** Sample K random words $c_1^-, \dots, c_K^-$ from the vocabulary $\rightarrow$ label = 0.

Negative Sampling

Key idea: Instead of softmax over all $|V|$ words, reformulate as a **binary classification** problem.

For each observed (target, context) pair (w, c) :

- **Positive example:** (w, c) is a real co-occurrence $\rightarrow$ label = 1.
- **Negative examples:** Sample K random words $c_1^-, \dots, c_K^-$ from the vocabulary $\rightarrow$ label = 0.

Objective (Skip-gram with Negative Sampling, SGNS):

$$\mathcal{L} = \log \sigma(\mathbf{u}_c^\top \mathbf{w}) + \sum_{i=1}^K \mathbb{E}_{c_i^- \sim P_n} \left[\log \sigma(-\mathbf{u}_{c_i^-}^\top \mathbf{w}) \right]$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

Cost: $\mathcal{O}(K)$ per training step instead of $\mathcal{O}(|V|)$. Typically $K = 5-20$.

Negative Sampling: Noise Distribution

How to sample negative examples?

Noise distribution $P_n(w)$:

$$P_n(w) = \frac{f(w)^{3/4}}{\sum_{w'} f(w')^{3/4}}$$

where $f(w)$ is the unigram frequency of word w .

Why the 3/4 exponent?

- Using raw unigram frequencies $f(w)$ oversamples “the”, “a”, “of”.
- The 3/4 power **flattens** the distribution: rare words get sampled more often relative to their frequency.
- Empirically found to work well (Mikolov et al., 2013).

Example: $f(\text{the}) = 0.01$ and $f(\text{platypus}) = 10^{-7}$: raw ratio 10^5 ; after the 3/4 power, ≈ 5623 .

SGNS and PMI: A Surprising Connection

Levy & Goldberg (2014) showed that Skip-gram with Negative Sampling (SGNS) is **implicitly factorizing** a PMI matrix!

Specifically, as the embedding dimension $d \rightarrow \infty$ and with sufficient training data:

$$\mathbf{w}^\top \mathbf{u}_c \approx \text{PMI}(w, c) - \log K$$

where K is the number of negative samples.

SGNS and PMI: A Surprising Connection

Levy & Goldberg (2014) showed that Skip-gram with Negative Sampling (SGNS) is **implicitly factorizing** a PMI matrix!

Specifically, as the embedding dimension $d \rightarrow \infty$ and with sufficient training data:

$$\mathbf{w}^\top \mathbf{u}_c \approx \text{PMI}(w, c) - \log K$$

where K is the number of negative samples.

Implication:

- Count-based (PPMI + SVD) and prediction-based (Word2Vec) methods are **closely related**.
- Word2Vec is not magic; it is (approximately) doing matrix factorization.
- SGNS has practical advantages: it scales to huge corpora and supports efficient online training.

[Levy, O. & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. *NeurIPS*.]

CBOW (Continuous Bag of Words)

CBOW is the “reverse” of Skip-gram:

- **Input:** context words (averaged or summed).
- **Output:** predict the center word.

Given context words $c_1, \dots, c_{2k}$ around target w , average the context (input) vectors and compute the softmax over the target's output vectors ($\mathbf{W}$ and $\mathbf{U}$ roles swapped vs. Skip-gram):

$$\hat{\mathbf{c}} = \frac{1}{2k} \sum_{i=1}^{2k} \mathbf{u}_{c_i}, \quad P(w \mid c_1, \dots, c_{2k}) = \frac{\exp(\mathbf{w}^\top \hat{\mathbf{c}})}{\sum_{w'} \exp(\mathbf{w}'^\top \hat{\mathbf{c}})}$$

CBOW (Continuous Bag of Words)

CBOW is the “reverse” of Skip-gram:

- **Input:** context words (averaged or summed).
- **Output:** predict the center word.

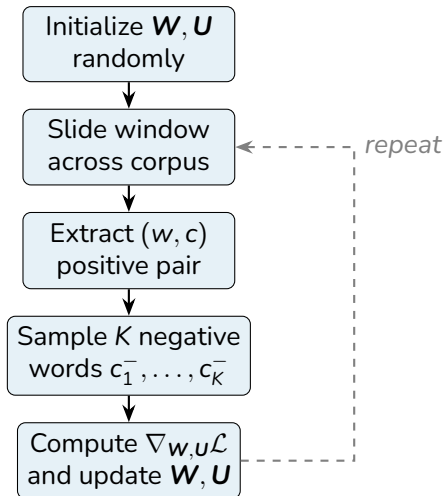
Given context words $c_1, \dots, c_{2k}$ around target w , average the context (input) vectors and compute the softmax over the target's output vectors ($\mathbf{W}$ and $\mathbf{U}$ roles swapped vs. Skip-gram):

$$\hat{\mathbf{c}} = \frac{1}{2k} \sum_{i=1}^{2k} \mathbf{u}_{c_i}, \quad P(w \mid c_1, \dots, c_{2k}) = \frac{\exp(\mathbf{w}^\top \hat{\mathbf{c}})}{\sum_{w'} \exp(\mathbf{w}'^\top \hat{\mathbf{c}})}$$

Skip-gram vs. CBOW:

- Skip-gram: better for **rare words** (each word is a separate training example).
- CBOW: faster training, slightly better for **frequent words**.
- In practice, Skip-gram (with negative sampling) is more commonly used.

Word2Vec: The Skip-Gram Training Loop



Evaluating Word Embeddings

How do we know if our word vectors are good?

Intrinsic evaluation:

- **Word similarity:** Compare cosine similarity of word pairs against human judgments (e.g., SimLex-999, WordSim-353).
- **Word analogy:** “*a* is to *b* as *c* is to *_*”.

Extrinsic evaluation:

- Use word vectors as features in a downstream task (NER, sentiment analysis, parsing).
- Measure task performance.
- Better embeddings → better downstream accuracy.

Word Analogy Task

The classic test:

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

More precisely, find w^* such that:

$$w^* = \arg \max_{w \in V \setminus \{a, b, c\}} \cos(\mathbf{v}_w, \mathbf{v}_b - \mathbf{v}_a + \mathbf{v}_c)$$

Examples:

a	b	c	Expected w^*
man	king	woman	queen
Paris	France	Tokyo	Japan
big	bigger	small	smaller
slow	slowly	quick	quickly

Word Analogy Task

The classic test:

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

More precisely, find w^* such that:

$$w^* = \arg \max_{w \in V \setminus \{a, b, c\}} \cos(\mathbf{v}_w, \mathbf{v}_b - \mathbf{v}_a + \mathbf{v}_c)$$

Examples:

a	b	c	Expected w^*
man	king	woman	queen
Paris	France	Tokyo	Japan
big	bigger	small	smaller
slow	slowly	quick	quickly

Caveat: Analogy tasks test a narrow slice of meaning: good analogy performance $\nRightarrow$ good embeddings for all tasks.

Biases in Word Embeddings

Word embeddings learn from text data, including its **biases**.

Example (Bolukbasi et al., 2016):

- $\mathbf{v}_{\text{man}} - \mathbf{v}_{\text{woman}} \approx \mathbf{v}_{\text{computer programmer}} - \mathbf{v}_{\text{homemaker}}$
- $\mathbf{v}_{\text{father}} - \mathbf{v}_{\text{mother}} \approx \mathbf{v}_{\text{doctor}} - \mathbf{v}_{\text{nurse}}$

These analogies reflect **societal stereotypes** in the training data.

Why this matters:

- Embeddings are used in real systems (search, hiring, recommendations).
- Biased embeddings $\rightarrow$ biased downstream decisions.
- Debiasing and fairness evaluation remain open problems.

[Bolukbasi, T. et al. (2016). Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. *NeurIPS*.]

Beyond Word2Vec

Word2Vec opened the door to many extensions:

- **GloVe** (Pennington et al., 2014): Combines count-based and prediction-based approaches. Directly factorizes a log co-occurrence matrix.
- **FastText** (Bojanowski et al., 2017): Extends Word2Vec with **subword** information. Represents each word as a bag of character n -grams → handles rare/unseen words.
- **Contextual embeddings** (coming later in the course): ELMo, BERT: word vectors that change based on context.

The “embed everything” idea also extends beyond words:

- **Node2Vec**: graph nodes.
- **Doc2Vec**: documents.
- **Item2Vec**: items in recommendation systems.

Summary

- **Distributional hypothesis:** A word's meaning is defined by the contexts in which it appears.
- **One-hot vectors** cannot capture similarity; we need **dense embeddings**.
- **Count-based methods:**
 - Build co-occurrence matrices, reweight with **PPMI**, reduce with **SVD**.
- **Word2Vec** (prediction-based):
 - **Skip-gram:** predict context from target.
 - **CBOW:** predict target from context.
 - **Negative sampling:** efficient training via binary classification.
- SGNS $\approx$ implicit matrix factorization of a shifted PMI matrix.
- Word embeddings capture analogies but also **biases**.

Readings

Required:

- Jurafsky & Martin, *SLP 3*, [Chapter 5: Embeddings](#).

Optional / further reading:

- Mikolov, T. et al. (2013). [Efficient estimation of word representations in vector space](#). *arXiv:1301.3781*.
- Mikolov, T. et al. (2013). [Distributed representations of words and phrases and their compositionality](#). *NeurIPS*.
- Levy, O. & Goldberg, Y. (2014). [Neural word embedding as implicit matrix factorization](#). *NeurIPS*.
- Pennington, J. et al. (2014). [GloVe: Global vectors for word representation](#). *EMNLP*.

Next Lecture

Supervised Learning Foundations

- From embeddings to predictions.
- Linear classifiers and logistic regression.
- Multi-class classification and softmax.