Language Models

Lecture 7

Adarsh Kumar
Department of Industrial Mining Engineering and ICT (EMIT),
Manresa School of Engineering (EPSEM),
Polytechnic University of Catalonia, Manresa, Barcelona, Spain
adarsh.kumar@upc.edu

May 5, 2026

What is an N-gram Model?

Idea: Predict the next word using only the last $(n-1)$ words.

$$P(w_t \mid w_1, ..., w_{t-1}) \approx P(w_t \mid w_{t-n+1}, ..., w_{t-1})$$

Examples:

- Unigram $(n = 1)$: $P(w_t)$
- Bigram $(n = 2)$: $P(w_t \mid w_{t-1})$
- Trigram $(n = 3)$: $P(w_t \mid w_{t-2}, w_{t-1})$

Training Example (Bigram)

Training Corpus:

I love NLP
I love AI
I love ML

Bigram Counts:

Bigram	Count
(I, love)	3
(love, NLP)	1
(love, AI)	1
(love, ML)	1

Computing Probabilities

Bigram probability:

$$P(w_t \mid w_{t-1}) = \frac{\text{Count}(w_{t-1}, w_t)}{\text{Count}(w_{t-1})}$$

Since:

$$\text{Count}(I) = 3$$

$$P(love \mid I) = \frac{3}{3} = 1.0$$

$$P(NLP \mid love) = \frac{1}{3}$$

Sentence Probability Example

Find probability of:

"I love NLP"

Using bigram model:

$$P(I) \times P(love \mid I) \times P(NLP \mid love)$$

Assume:

$$P(I) = \frac{3}{9}$$

$$P(love \mid I) = 1.0$$

$$P(NLP \mid love) = \frac{1}{3}$$

$$P(\text{sentence}) = \frac{3}{9} \times 1 \times \frac{1}{3}$$

Limitation: Cannot handle unseen bigrams (zero probability problem).

The Bigram Model

The Markov Assumption

We approximate the probability of a word by looking only at the **previous word**:

$$P(w_i|w_1 w_2 \dots w_{i-1}) \approx P(w_i|w_{i-1})$$

- This is a **1st-order Markov Process**.
- It assumes that the "history" is captured sufficiently by the single preceding word.
- Total sentence probability becomes:

$$P(w_1^n) \approx \prod_{i=1}^{n} P(w_i|w_{i-1})$$

Example: Training on Shakespeare

If our training text is: *"To be or not to be"*

Previous (w_{i-1})	Current (w_i)	Count
To	be	2
be	or	1
or	not	1
not	to	1

- $P(\text{be}|\text{to}) = \frac{Count(\text{to be})}{Count(\text{to})} = \frac{2}{2} = 1.0$
- The model learns "local" grammar but lacks "global" context.

Local vs. Global Context

- **Local Grammar:** The model ensures w_i fits with w_{i-1}.
  - Example: "Eat **the apple**" (High Probability)
  - Example: "Eat **the sky**" (Low Probability)
- **Global Failure:** The model "forgets" the subject of the sentence.

The "Forgetful" Bigram

"The cake I bought for the party yesterday **is delicious**."

A Bigram model only sees "... yesterday [?]" and might predict "yesterday **went**" because it forgot about the cake.

Why does it "Forget"?

- **Limited Context Window:** The Bigram model has a memory of $N - 1 = 1$.
- **Distance Problem:**

The Cake Example

Sentence: *"The **cake** that I bought yesterday . . . "*

Human Logic: Knows the subject is *cake*.
Bigram Logic: Only sees $P(w_{next}|\text{yesterday})$.

The Result

The model might generate: *"The cake that I bought yesterday **afternoon was** . . . "*
By sheer luck, "afternoon" fits "yesterday." But if the sentence were longer, the model would eventually drift into a completely different topic (e.g., *". . . yesterday morning rain is wet"*).

Expanding the Window: The Trigram Model

- A **Trigram** looks at the **two** preceding words.
- **Formula:**

$$P(w_i|w_1 \dots w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$$

Comparison: Predicting after "New York"

- **Bigram:** Sees "York." Might predict *York* ***Post*** or *York* ***Peppermint***.
- **Trigram:** Sees "New York." Much more likely to predict *New York* ***City*** or *New York* ***State***.

Trigram vs. Bigram: The "Cake" Example

Sentence Context: "The cake I bought yesterday . . ."

Bigram Model

Context Window: 1 word
Sees only: "yesterday"

- $P(w_i|\text{yesterday})$
- **Result:** "yesterday **morning**" or "yesterday **went**."
- **Failure:** Has no idea a cake exists.

Trigram Model

Context Window: 2 words
Sees: "bought yesterday"

- $P(w_i|\text{bought, yesterday})$
- **Result:** "yesterday **is**" or "yesterday **was**."
- **Success:** Captures the **syntax** of the phrase.

- **Key Takeaway:** By seeing "bought yesterday," the Trigram model knows a verb *must* follow the action, making the sentence more stable than the Bigram.

The Long-Distance Dependency Problem

- **Concept:** A Trigram model has a "memory" of only two words.
- **The Challenge:** Language often has connections (dependencies) that span many words.

Example: Subject-Verb Agreement

Sentence: "The **books** that I left on the dusty **old table** . . ."

Human Logic	Trigram Logic
• Connects books to the verb.	• Sees only old table.
• Result: **are** (Plural)	• Result: **is** (Singular)

The Failure

$P(w_i \mid \text{old, table})$ predicts *is* because the model cannot "see" the subject (**books**) situated 7 words back. The "bridge" is too long for its 2-word memory.

The Sparsity Problem (Zero Probability)

- **The Dilemma:** Increasing n improves context but makes the model "fragile."
- **Definition:** Most valid word sequences never appear in a training corpus, leading to $P = 0$.

Example: The Training Data

Suppose our corpus only contains:

1. "The **cat sat**"
2. "The **dog ran**"

The Trigram Calculation

To calculate the probability of: *"The cat **ran**"*
The model looks for: $P(\text{ran} \mid \text{the, cat}) = \frac{Count(\text{the cat ran})}{Count(\text{the cat})}$

The "Technical Nightmare"

Since *"the cat ran"* never occurred together, **Count = 0**.
The model assigns **zero probability** to a perfectly valid English sentence and "breaks."

The 4-gram Model: Precision and Idioms

- **Mathematical Approximation:** The context window expands to the three previous words ($N - 1 = 3$).

$$P(w_i \mid w_1 \dots w_{i-1}) \approx P(w_i \mid w_{i-3}, w_{i-2}, w_{i-1})$$

Example 1: Fixed Entities (Idioms)

Predicting the word after: "*The United States of . . .*"

- **Bigram** (*of*): High entropy (*of the, of course, of mice*)
- **Trigram** (*States of*): Better (*States of America, States of mind*)
- **4-gram** (**United States of**): Near 100% probability of **America**.

Example 2: Syntactic Stability

Sentence: "The cake I **bought yesterday was** . . ."

- **4-gram Context:** Captures the verb "was."
- **Result:** Correctly expects an **adjective** (*delicious, gone*) rather than a random noun, providing much smoother flow than a Bigram.

The Downside of 4-grams: Data Hunger

- **The Dilemma:** As N increases, accuracy improves, but the model becomes **statistically fragile**.
- **The Aggressive Sparsity Problem:** As the window grows, the likelihood of seeing a specific 4-word sequence in training drops to near zero.

Exponential Complexity

If your vocabulary V is **10,000** words:

- **Bigrams (V^2):** 100 million possible pairs.
- **Trigrams (V^3):** 1 trillion possible triplets.
- **4-grams (V^4): 10 quadrillion (10^{16})** possible sequences.

The Reality Check

Most 4-word combinations have **never been written** in human history! To train a reliable 4-gram model, you need a massive, near-infinite amount of text just to avoid getting "Zero Probabilities" for perfectly normal sentences.

Statistical Fragility: The N-gram Trade-off

- **Definition:** The model's inability to handle sequences that are slightly different from the training data.
- **The Rigidness of Higher N:** As n increases, the requirements for a "match" become too specific.

Example: The Sushi Problem

Training Data: "I want to eat **pizza**."
User Input: "I want to eat **sushi**."

- **Bigram:** Sees *eat* $\rightarrow$ predicts *sushi* (Success).
- **4-gram:** Sees *want to eat* $\rightarrow$ looks for *sushi*. If it's not there, probability becomes **Zero** (Failure).

The Fragility Paradox

By trying to be **more accurate** (larger N), the model becomes **less useful** because it cannot generalize to new, valid sentences. It becomes a "parrot" of its training data.

Memory vs. Complexity

Model	Context Window	Quality
Unigram	0 words (Random)	Very Poor
Bigram	1 word	Local Grammar only
Trigram	2 words	Short-phrase Coherence
LLM (GPT)	Thousands of words	Human-like Global Context

- As N increases, model accuracy improves.
- **The Cost:** Higher N requires exponentially more training data (The Sparsity Problem).

N-Gram Problems: Long-distance Dependencies in Bigram Models

Key Idea: Bigram models cannot maintain long-range context.

The Problem: In complex sentences, the grammatical form of a word often depends on a subject that appeared much earlier.

Example:

> *"The soups that I made from that new cookbook I bought yesterday were amazingly delicious."*

Human Logic:

- We know **"were"** is correct.
- Subject is plural: **"soups"**.

Bigram Model Logic:

- Model only sees the previous word: "yesterday".
- Might predict: **"yesterday was"**.
- Ignores long-distance subject "soups".

N-Gram Problems: Failure to Model Meaning (Fragility)

Key Idea: Bigram models are **statistically fragile** — they rely on exact word matches, not meaning.

The Problem: Bigram models struggle with new sequences, even if the sequences have similar meanings to ones seen before.

Example:

Trained on: "The cat sat"
Encountered: "The feline sat"

Human Logic:

- Understand that **"cat"** and **"feline"** are synonyms.
- Predicts **"sat"** correctly.

Bigram Model Logic:

- Sees **"feline"** as new/unseen.
- Probability of **"sat"** is low.
- Ignores semantic similarity.

Why Study N-gram Models?

N-gram models: the "fruit fly of NLP"

- Simple models that introduce core NLP concepts.
- Useful to understand how large language models work at a fundamental level.
- Fast, interpretable, and educationally valuable.

Key Topics Introduced: Training/test sets, perplexity, sampling, interpolation, backoff.

Next-Word Prediction with N-grams

Bigram Model:

$$P(w_n \mid w_{n-1}) = \frac{\text{Count}(w_{n-1}, w_n)}{\text{Count}(w_{n-1})}$$

Example: Corpus: "I love NLP", "I love AI"

- $P(\text{love} \mid I) = 1.0$
- $P(\text{NLP} \mid \text{love}) = 0.5$
- $P(\text{AI} \mid \text{love}) = 0.5$

Illustrates basic **next-word prediction**, the foundation of language models.

Training and Test Sets

Purpose: Evaluate how well the model generalizes to unseen text.
Example:

- Train: "I love NLP", "AI is fun"
- Test: "I love AI"
- Challenge: "love AI" is new $\rightarrow$ smoothing needed

Lesson: Introduces the concept of generalization, key in all NLP models.

Perplexity: Evaluating N-gram Models

Perplexity measures "surprise":

$$\text{Perplexity} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_{i-1},\dots)}$$

Example: Bigram probabilities for "I love AI":

- $P(I \mid < s >) = 0.6$
- $P(love \mid I) = 1.0$
- $P(AI \mid love) = 0.5$

Interpretation: Lower perplexity = model predicts better.

Sampling Sentences with N-grams

Generate text probabilistically:

- Start with $< s >$ (start token)
- Sample next word using learned probabilities
- Repeat until $< /s >$ (end token)

Example (Bigram Sampling):

1. Start: $< s > \rightarrow$ "I" (P=0.6)
2. Next: "love" (P=1.0)
3. Next: "AI" (P=0.5)

Result: "I love AI"

Interpolation and Backoff

Handling unseen sequences:

- **Backoff:** Use lower-order n-grams if higher-order is unseen
- **Interpolation:** Weighted combination of unigram, bigram, trigram

Example:

$$P_{\text{interp}}(AI \mid I, love) = \lambda_3 P_{\text{tri}} + \lambda_2 P_{\text{bi}} + \lambda_1 P_{\text{uni}}$$

Purpose: Introduces smoothing concepts used in all modern language models.

Efficiency and Practical Use

- N-gram models are small, fast, and interpretable.
- Useful in low-resource or lightweight scenarios.
- Example: Mobile app predicting next word in a tiny corpus offline.

Summary:

- Introduces next-word prediction, perplexity, sampling.
- Teaches smoothing and generalization concepts.
- Efficient for practical applications when LLMs are overkill.

Maximum Likelihood Estimation (MLE) for Bigram Probabilities

Definition: MLE estimates the probability of a word given the previous word based on observed counts:

$$P(w_n \mid w_{n-1}) = \frac{\text{Count}(w_{n-1}, w_n)}{\text{Count}(w_{n-1})}$$

Interpretation: - Numerator: how many times the bigram occurs - Denominator: how many times the previous word occurs - Meaning: "How often does this word follow the previous word?"

Step 1: Example Corpus

Corpus:

- I love NLP
- I love AI
- I enjoy AI

Step 1: Count Bigrams

Bigram	Count
I love	2
love NLP	1
love AI	1
I enjoy	1
enjoy AI	1

Step 2: Count Previous Words (Unigrams)

Word	Count
I	3
love	2
enjoy	1

Step 3: Apply MLE Formula

$$P(w_n \mid w_{n-1}) = \frac{\text{Count}(w_{n-1}, w_n)}{\text{Count}(w_{n-1})}$$

Bigram Probabilities:

- $P(\text{love} \mid I) = \frac{2}{3} \approx 0.667$
- $P(\text{enjoy} \mid I) = \frac{1}{3} \approx 0.333$
- $P(\text{NLP} \mid \text{love}) = \frac{1}{2} = 0.5$
- $P(\text{AI} \mid \text{love}) = \frac{1}{2} = 0.5$

Interpretation: Given "I", 66.7% of the time the next word is "love", 33.3% it is "enjoy".

Step 4: Problem of Zero Probabilities

- What if the model sees an unseen bigram, e.g., "I hate AI"? - MLE assigns: $P(\text{hate} \mid I) = 0$

Solution: Smoothing - Laplace (Add-one) smoothing:

$$P_{\text{Laplace}}(w_n \mid w_{n-1}) = \frac{\text{Count}(w_{n-1}, w_n) + 1}{\text{Count}(w_{n-1}) + V}$$

where V = vocabulary size

Example: $V = 4$ (I, love, NLP, AI)

$$P(\text{I} \mid \text{love}) = \frac{0+1}{2+4} = \frac{1}{6} \approx 0.167$$

Large Language Models (LLMs)

- Neural network based
- Use long context
- Learn distributed word representations
- Trained on massive datasets

Examples:

- GPT-style models
- Transformer-based architectures

These models capture long-range dependencies and semantic meaning.

Comparison

Feature	N-gram LM	LLM
Context Size	Limited	Large
Memory	Fixed window	Long-range
Method	Counting	Neural Networks
Data Requirement	Small	Very Large

Word Embeddings?

- Dense vector representations of words in a **continuous vector space**.
- Semantically similar words are **closer together**.
- Overcomes limitations of one-hot encoding:
  - Sparse and high-dimensional
  - Cannot capture semantic similarity

Example

Word	**One-hot vector**
king	[1,0,0,...,0]
queen	[0,1,0,...,0]

Vectors are orthogonal, so similarity = 0.

Why Use Word Embeddings?

- Capture **semantic similarity**: "dog" $\approx$ "puppy"
- Reduce dimensionality of representations
- Improve performance in NLP tasks:
  - Text classification
  - Machine translation
  - Named entity recognition
  - Sentiment analysis

How Are Word Embeddings Learned?
(A) Prediction-based (Neural Embeddings)

- **Word2Vec (Mikolov et al., 2013)**:
  - CBOW: Predict target word from context
  - Skip-gram: Predict context from target word
  - Captures analogies: king $-$ man $+$ woman $\approx$ queen

(B) Count-based (Matrix Factorization)

- **GloVe (2014)**:
  - Constructs word-context co-occurrence matrix across the corpus
  - Combines global statistics with local context
  - Applies weighted factorization to produce dense embeddings
- Other methods: LSA (Latent Semantic Analysis)

Overview

- GloVe (Global Vectors for Word Representation) is an unsupervised word-embedding algorithm.
- It maps words to dense numeric vectors that capture meaning and relationships.
- Widely used as a fast, effective feature representation in real-time NLP systems.

What is GloVe?

- Learns word vectors from global word–word co-occurrence statistics.
- Uses a context window over a large corpus to build a co-occurrence matrix.
- Trains embeddings so that vector geometry reflects semantic similarity and analogies.

Our Tiny Corpus

Training Data

Three simple sentences:

1. "ice is cold"
2. "steam is hot"
3. "ice and steam are both water"

Vocabulary (9 words):

$$V = \{\text{ice, steam, cold, hot, water, is, are, both, and}\}$$

Step 1: Building Co-occurrence Matrix

Using context window of size 2 (2 words before and after target):

For "ice":

- Sent 1: "ice is cold" → is, cold
- Sent 3: "ice and steam are both water" → and, steam

For "steam":

- Sent 2: "steam is hot" → is, hot
- Sent 3: "ice and steam are both " → and, are, both

Step 2: Co-occurrence Matrix

	ice	steam	cold	hot	water	is	are	both	and
ice	0	1	1	0	1	1	0	0	1
steam	1	0	0	1	1	1	1	1	1
cold	1	0	0	0	0	1	0	0	0
hot	0	1	0	0	0	0	1	0	0
water	1	1	0	0	0	0	1	1	1
is	1	1	1	0	0	0	0	0	0
are	0	1	0	1	1	0	0	0	1
both	0	1	0	0	1	0	1	0	0
and	1	1	0	0	1	0	1	0	0

Table: Word-word co-occurrence counts

Step 3: Raw Co-occurrence Vectors

Each word is represented by its row in the co-occurrence matrix:

Word "ice" vector (9-dimensional)

$$\text{ice} = [0, 1, 1, 0, 1, 1, 0, 0, 1]$$

`[ice, steam, cold, hot, water, is, are, both, and]`

Word "steam" vector

$$\text{steam} = [1, 0, 0, 1, 1, 1, 1, 1, 1]$$

Dimension = 9 (vocabulary size)

Step 4: Computing Similarity

Cosine Similarity between ice and steam:

$$\text{cosine}(a, b) = \frac{a \cdot b}{||a|| \times ||b||}$$

$$\begin{aligned} a \cdot b &= (0 \times 1) + (1 \times 0) + (1 \times 0) + (0 \times 1) + (1 \times 1) + (1 \times 1) \\ &\quad + (0 \times 1) + (0 \times 1) + (1 \times 1) \\ &= 0 + 0 + 0 + 0 + 1 + 1 + 0 + 0 + 1 = \mathbf{3} \end{aligned}$$

Other Similarity Methods: Cosine similarity (Standard for embeddings, ignores vector magnitude), Euclidean distance, Manhattan (L1) distance, Pearson correlation, KL-divergence.

Step 5: Computing Magnitudes

$$
\begin{aligned}
||\text{ice}|| &= \sqrt{0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2 + 0^2 + 0^2 + 1^2} \\
&= \sqrt{0 + 1 + 1 + 0 + 1 + 1 + 0 + 0 + 1} \\
&= \sqrt{5} = \mathbf{2.236}
\end{aligned}
$$

$$
\begin{aligned}
||\text{steam}|| &= \sqrt{1^2 + 0^2 + 0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2} \\
&= \sqrt{1 + 0 + 0 + 1 + 1 + 1 + 1 + 1 + 1} \\
&= \sqrt{7} = \mathbf{2.646}
\end{aligned}
$$

Step 6: Final Similarity Score

$$\text{cosine}(\text{ice}, \text{steam}) = \frac{3}{2.236 \times 2.646} = \frac{3}{5.917} = \mathbf{0.507}$$

Higher cosine similarity means more semantically related words

The Formula

$$\cos(\theta) = 0.507$$

$$\theta = \arccos(0.507)$$

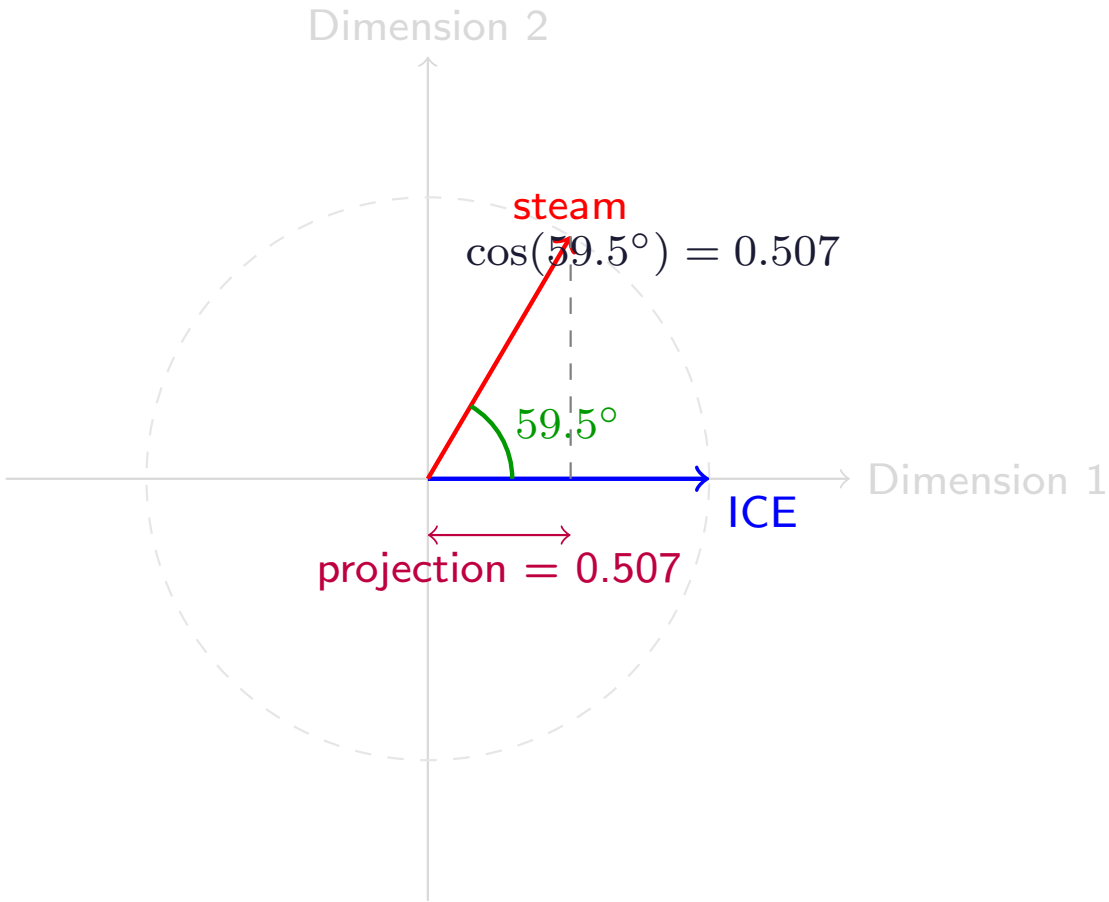
Calculation

$$\theta = \arccos(0.507) = 1.039 \text{ radians}$$

$$\theta = 1.039 \times \frac{180^\circ}{\pi} = \mathbf{59.5}^\circ$$

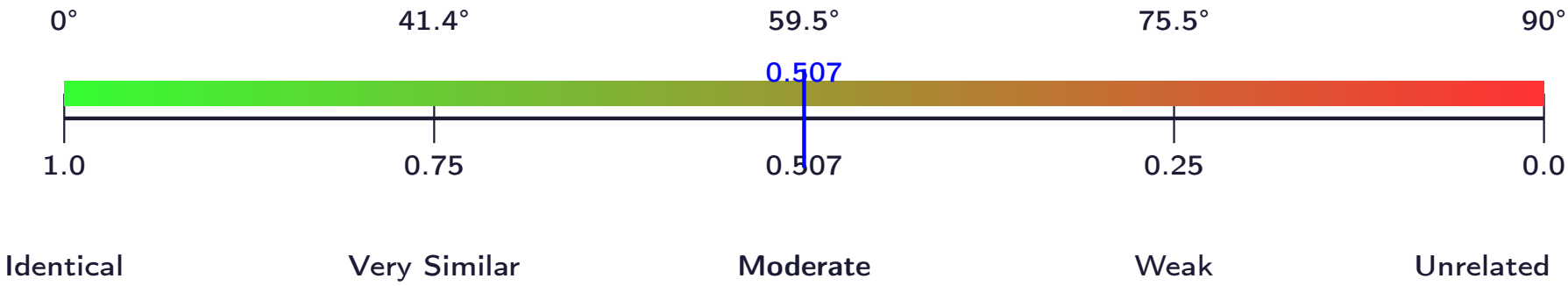
0.507 is the cosine of a **59.5°** angle

Visual Representation



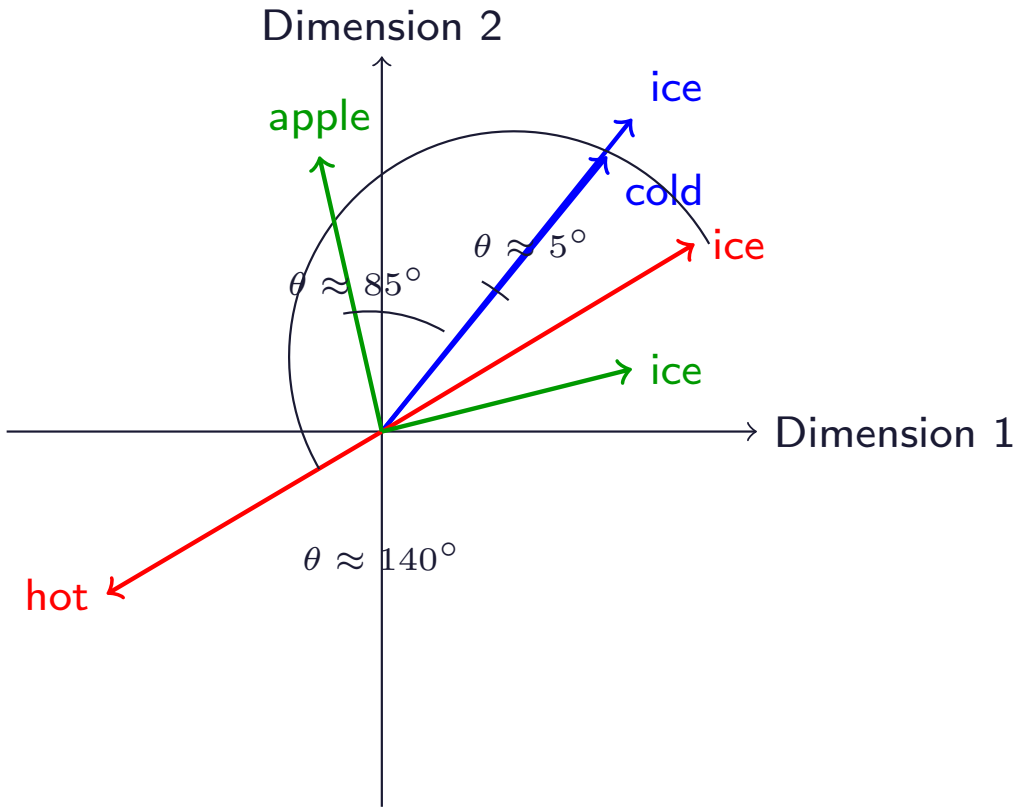
- **Blue vector:** Reference direction
- **Red vector:** At $59.5°$ angle
- **Purple projection:** Length $= 0.507$

Where 0.507 Falls on the Spectrum



Range	Interpretation
0.80 - 1.00	Very similar / synonyms
0.60 - 0.80	Related concepts
yellow!20 0.50 - 0.60	**Moderate similarity (0.507 here)**
0.30 - 0.50	Weakly related
0.00 - 0.30	Unrelated

The Core Intuition



- Small angle → high similarity (ice-cold)
- Large angle → low similarity (ice-hot)
- Perpendicular → unrelated (ice-apple)

Cosine Similarity Formula

Definition

$$\text{cosine}(a, b) = \frac{a \cdot b}{||a|| \times ||b||} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}}$$

Range

$$-1 \leq \text{cosine}(a, b) \leq 1$$

- **+1**: Vectors point in exactly the same direction
- **0**: Vectors are orthogonal (perpendicular) - no relationship
- **-1**: Vectors point in opposite directions

Step 7: Dimensionality Reduction

Before (9D sparse vectors):

$$\text{ice} = [0, 1, 1, 0, 1, 1, 0, 0, 1] \quad \text{(mostly zeros)}$$

After (3D dense vectors) - GloVe's job:

$$\begin{aligned} \text{ice} &= [0.8, 0.3, -0.2] \\ \text{steam} &= [0.7, -0.1, 0.4] \\ \text{cold} &= [0.9, 0.4, -0.3] \\ \text{hot} &= [0.6, -0.3, 0.5] \\ \text{water} &= [0.75, 0.1, 0.1] \end{aligned}$$

All numbers are non-zero and meaningful

Step 9: Word Analogies

Analogy: "ice is to cold as steam is to ?"

Using vector arithmetic:

$$\text{cold} - \text{ice} + \text{steam} = ?$$

$$[0.9, 0.4, -0.3] - [0.8, 0.3, -0.2] + [0.7, -0.1, 0.4]$$
$$= [0.8, 0.0, 0.3]$$

This resulting vector is closest to:

$$\text{hot} = [0.6, -0.3, 0.5]$$

Cosine similarity: 0.87

Step 10: Real-time Usage Example

Input: "The ice feels ___"

1. Convert "ice" to vector: $[0.8, 0.3, -0.2]$
2. Find words with highest similarity:

Word	Similarity to "ice"
cold	?
water	?
solid	?
steam	?
hot	?

Model predicts "cold" as most likely completion

Our GloVe Vectors (3D)

Dense word vectors from training

$$\text{ice} = [0.8, 0.3, -0.2]$$
$$\text{cold} = [0.9, 0.4, -0.3]$$
$$\text{steam} = [0.7, -0.1, 0.4]$$
$$\text{hot} = [0.6, -0.3, 0.5]$$
$$\text{apple} = [0.1, 0.8, 0.7]$$
$$\text{car} = [0.2, 0.7, 0.6]$$

ice-cold: Related Words

$$\begin{aligned}\text{dot product} &= 0.8 \times 0.9 + 0.3 \times 0.4 + (-0.2) \times (-0.3) \\ &= 0.72 + 0.12 + 0.06 = \mathbf{0.90}\end{aligned}$$

$$\begin{aligned}||ice|| &= \sqrt{0.8^2 + 0.3^2 + (-0.2)^2} \\ &= \sqrt{0.64 + 0.09 + 0.04} = \sqrt{0.77} = \mathbf{0.877}\end{aligned}$$

$$\begin{aligned}||cold|| &= \sqrt{0.9^2 + 0.4^2 + (-0.3)^2} \\ &= \sqrt{0.81 + 0.16 + 0.09} = \sqrt{1.06} = \mathbf{1.030}\end{aligned}$$

$$\text{cosine} = \frac{0.90}{0.877 \times 1.030} = \frac{0.90}{0.903} = \mathbf{0.996}$$

ice-steam: Moderately Related

$$\begin{aligned}\text{dot product} &= 0.8 \times 0.7 + 0.3 \times (-0.1) + (-0.2) \times 0.4 \\ &= 0.56 - 0.03 - 0.08 = \mathbf{0.45}\end{aligned}$$

$$\begin{aligned}||steam|| &= \sqrt{0.7^2 + (-0.1)^2 + 0.4^2} \\ &= \sqrt{0.49 + 0.01 + 0.16} = \sqrt{0.66} = \mathbf{0.812}\end{aligned}$$

$$\text{cosine} = \frac{0.45}{0.877 \times 0.812} = \frac{0.45}{0.712} = \mathbf{0.632}$$

ice-hot: Opposite Concepts

$$\begin{aligned}\text{dot product} &= 0.8 \times 0.6 + 0.3 \times (-0.3) + (-0.2) \times 0.5 \\ &= 0.48 - 0.09 - 0.10 = \mathbf{0.29}\end{aligned}$$

$$\begin{aligned}||hot|| &= \sqrt{0.6^2 + (-0.3)^2 + 0.5^2} \\ &= \sqrt{0.36 + 0.09 + 0.25} = \sqrt{0.70} = \mathbf{0.837}\end{aligned}$$

$$\text{cosine} = \frac{0.29}{0.877 \times 0.837} = \frac{0.29}{0.734} = \mathbf{0.395}$$

ice-apple: Unrelated Concepts

$$\begin{aligned}\text{dot product} &= 0.8 \times 0.1 + 0.3 \times 0.8 + (-0.2) \times 0.7 \\ &= 0.08 + 0.24 - 0.14 = \mathbf{0.18}\end{aligned}$$

$$\begin{aligned}||apple|| &= \sqrt{0.1^2 + 0.8^2 + 0.7^2} \\ &= \sqrt{0.01 + 0.64 + 0.49} = \sqrt{1.14} = \mathbf{1.068}\end{aligned}$$

$$\text{cosine} = \frac{0.18}{0.877 \times 1.068} = \frac{0.18}{0.937} = \mathbf{0.192}$$

Summary of Results

Word Pair	Cosine Similarity
ice-cold	0.996
ice-steam	0.632
ice-hot	0.395
ice-apple	0.192

Higher similarity $\rightarrow$ More semantically related

The Distributional Hypothesis

Core Principle

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

— Zellig Harris (1954)

Context Patterns:

- **ice** appears with: cold, freeze, solid, winter, water
- **steam** appears with: hot, vapor, boil, pressure, water
- **apple** appears with: fruit, eat, red, pie, tree

How Vectors Capture Meaning

Latent Dimensions (Interpretation)

	Dim 1 (temperature)	Dim 2 (physical state)	Dim 3 (water-related)
ice	$+0.9$	$+0.8$	$+0.7$
cold	$+0.8$	$+0.7$	$+0.3$
steam	$+0.2$	-0.7	$+0.8$
hot	-0.8	-0.3	$+0.2$
apple	$+0.1$	$+0.2$	-0.8

Similar patterns $\rightarrow$ Similar directions $\rightarrow$ High cosine similarity

Visual Intuition: 2D Projection