

Reduction and Prediction

From Fifty Thousand Dimensions to Three Hundred

James Pustejovsky

Brandeis University
Department of Computer Science

CS 135: Computational Semantics

Fall 2026

Where We Left Off

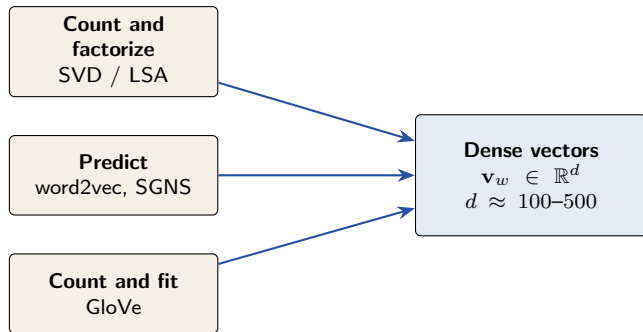
Session 7 built a word vector in three decisions: choose a context, weight the counts, compare with cosine. It worked — the margin between correct and incorrect pairs went from 0.05 to 0.96.

It also left three problems, all consequences of the matrix being $|V| \times |C|$ and mostly zero.

Problem	Symptom
Sparsity as blindness	espresso and cappuccino share no context $\Rightarrow$ cosine exactly 0
No generalization	columns car and automobile are unrelated coordinates
Dimensionality	50,000 features where downstream models want a few hundred

And one promise: that all three are addressed by a single move — find a low-dimensional space in which the observed matrix is approximately recoverable.

Today: Three Routes, One Destination



The three routes were developed a quarter-century apart, in different communities, with different justifications — and the middle one was widely described as having replaced the first.

The result we will prove today: the middle route is the first route, with a particular weighting and a particular objective. Session 7 already put the clue on the board — the *shifted PMI* row in the weighting table.

The Fourth Reason to Reduce

Three of the reasons are practical. The fourth is the interesting one, and it is not about size at all.

Generalization

Two words with **no shared context** have cosine exactly zero, no matter how similar they are. Reduction can give them a nonzero similarity — by forcing both to be expressed in terms of a small set of shared latent directions.

This is a genuinely different kind of claim from “the matrix is too big.” It says the reduced representation can assert something the data did not contain.

Is that inference, or invention?

The answer: reduction adds an **inductive bias** — the assumption that the true matrix is approximately low rank. That is information only if the assumption is right. We will test it on a case where we know the answer.

The Tool: Singular Value Decomposition

Theorem

Every real matrix $M \in \mathbb{R}^{m \times n}$ factors as

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

with $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ orthogonal, and Σ diagonal with entries $\sigma_1 \geq \sigma_2 \geq \dots \geq 0$.

Read the three factors as a change of coordinates, a scaling, and a change of coordinates back:

$V^\top$	rotate the <i>context</i> space to a new orthogonal basis
Σ	scale each new direction by its importance σ_i
U	rotate into the <i>word</i> space

The new basis directions are not contexts. They are linear combinations of contexts, ordered by how much of the matrix they explain.

First: What Does “Rank” Mean?

The theorem we are heading for is about *rank- k* approximation, so we need that word.

The simplest case

A matrix has **rank 1** when every row is a multiple of one single row — one pattern, rescaled.

$$\begin{pmatrix} 3 & 1 \\ 6 & 2 \\ 0 & 0 \end{pmatrix} \quad \text{rows are } 1 \cdot (3, 1), \quad 2 \cdot (3, 1), \quad 0 \cdot (3, 1)$$

There is only one thing going on here. Knowing the pattern $(3, 1)$ and the three scale factors $(1, 2, 0)$ is knowing the whole matrix: **five numbers instead of six**, and the saving grows with the size.

Rank k then means: the matrix is a sum of k such patterns, and no fewer.

Equivalently, the row space has dimension k . We will use the sum-of-patterns reading throughout, because it is the one that makes the theorem obvious.

The Outer Product Builds Those Patterns

Given a column $\mathbf{u} \in \mathbb{R}^m$ and a row $\mathbf{v} \in \mathbb{R}^n$, the **outer product** $\mathbf{u}\mathbf{v}^\top$ is the $m \times n$ matrix

$$(\mathbf{u}\mathbf{v}^\top)_{ij} = u_i v_j$$

$$\mathbf{u} = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}, \quad \mathbf{v}^\top = (3, 1) \quad \implies \quad \mathbf{u}\mathbf{v}^\top = \begin{pmatrix} 3 & 1 \\ 6 & 2 \\ 0 & 0 \end{pmatrix}$$

$\mathbf{v}$ **the pattern** — what the rows all look like

$\mathbf{u}$ **the weights** — how much of the pattern each row gets

Every rank-1 matrix is an outer product, and every outer product is rank 1. The two notions are the same.

Contrast the inner product $\mathbf{u} \cdot \mathbf{v}$, which sums the index away and returns a scalar. The outer product sums nothing and *raises* the order from 1 to 2.

The SVD, Rewritten

Now read $M = U\Sigma V^\top$ one column at a time. It is a **sum of rank-1 pieces**:

$$M = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^\top + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^\top + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^\top$$

$\mathbf{v}_i$ the i th pattern over *contexts*

$\mathbf{u}_i$ how much each *word* uses that pattern

σ_i how important the pattern is, and they are sorted

This is the same factorization as before, written so you can see what it is *for*. The matrix is decomposed into named patterns, ordered by how much of the data each one explains.

And now truncation has an obvious meaning: **keep the big terms, drop the small ones.**

The Three Factors, For Our Matrix

$$\underbrace{\begin{pmatrix} 0 & 0.76 & 0.86 & 0 & 0 & 0.84 \\ 0 & 0.96 & 0.98 & 0 & 0 & 0.50 \\ 0.01 & 0 & 0 & 1.05 & 1.00 & 0 \\ 0 & 0 & 0 & 1.04 & 1.16 & 0 \end{pmatrix}}_{M \ (4 \times 6)} = \underbrace{\begin{pmatrix} 0.00 & 0.70 & -0.72 & 0.00 \\ 0.00 & 0.72 & 0.70 & 0.00 \\ 0.68 & 0.00 & 0.00 & -0.73 \\ 0.73 & 0.00 & 0.00 & 0.68 \end{pmatrix}}_{U \ (4 \times 4)} \underbrace{\begin{pmatrix} 2.13 & 0 & 0 & 0 \\ 0 & 2.02 & 0 & 0 \\ 0 & 0 & 0.29 & 0 \\ 0 & 0 & 0 & 0.08 \end{pmatrix}}_{\Sigma \ (4 \times 4)} \underbrace{\begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.70 & 0.72 & 0.00 \\ 0.00 & 0.60 & 0.65 & 0.00 & 0.00 & 0.47 \\ 0.00 & 0.41 & 0.25 & 0.00 & 0.00 & -0.88 \\ -0.07 & 0.00 & 0.00 & -0.72 & 0.69 & 0.00 \end{pmatrix}}_{V^T \ (4 \times 6)}$$

How to read it: column i of U , the i th entry of Σ , and row i of V^T belong together — they are the i th term. The **first** and **second** are picked out above.

Three Things to Notice

Σ is diagonal	every off-diagonal entry is 0. All Σ does is <i>scale</i> — it mixes nothing. The content is in U and $V^\top$.
and sorted	$2.13 > 2.02 > 0.29 > 0.08$, always descending. That ordering is what makes “keep the top k ” well defined.
U and V are orthonormal	columns of U are unit length and mutually perpendicular, likewise rows of $V^\top$. This is the property that makes truncation optimal.

A note on shapes. $V^\top$ is 4×6 , not 6×6 . This is the **economy** (or thin) SVD: the last two rows of the full $V^\top$ would multiply zero singular values and contribute nothing, so they are dropped.

In general, an $m \times n$ matrix of rank r has at most $\min(m, n)$ nonzero singular values. Here $\min(4, 6) = 4$, which is why there are four terms and not six.

Our Matrix: the First Two Patterns

Run it on the Session 7 matrix. $\sigma_1 = 2.127$:

$$\mathbf{u}_1 = (0, 0, 0.681, 0.733), \quad \mathbf{v}_1 = (0, 0, 0, 0.696, 0.719, 0)$$

Weights on laptop and tablet; pattern on screen and battery. **This term is “device-ness”.**

$\sigma_2 = 2.016$:

$$\mathbf{u}_2 = (0.697, 0.717, 0, 0), \quad \mathbf{v}_2 = (0, 0.604, 0.645, 0, 0, 0.467)$$

Weights on coffee and tea; pattern on cup, brew, morning. **This term is “beverage-ness”.**

Nobody supplied those two categories. They are what the factorization found, and they are the two largest terms because they explain the most.

All four entries of $\mathbf{u}_1$ are non-negative, and likewise $\mathbf{u}_2$ — these terms *add* structure. The next two will not.

Our Matrix: the Last Two Patterns

$\sigma_3 = 0.292$, an order of magnitude smaller:

$$\mathbf{u}_3 = (-0.717, +0.697, 0, 0), \quad \mathbf{v}_3 = (0, 0.414, 0.247, 0, 0, -0.876)$$

Look at the signs. coffee gets $-$ and tea gets $+$ — this term does not describe a topic, it describes a **difference within one**: mostly that coffee leans on morning more than tea does.

$\sigma_4 = 0.084$:

$$\mathbf{u}_4 = (0, 0, -0.733, +0.681)$$

Same shape, separating laptop from tablet.

The structure of the spectrum

The **large** terms are the topics. The **small** terms are the distinctions *within* a topic, and they are the ones carrying opposite signs.

Keep that in view. It tells you in advance what truncation will cost.

Truncation Is Just Dropping Terms

$$M = \underbrace{2.127 \mathbf{u}_1 \mathbf{v}_1^\top + 2.016 \mathbf{u}_2 \mathbf{v}_2^\top}_{\text{keep: the two topics}} + \underbrace{0.292 \mathbf{u}_3 \mathbf{v}_3^\top + 0.084 \mathbf{u}_4 \mathbf{v}_4^\top}_{\text{discard: the within-topic detail}}$$

Setting $k = 2$ does not “compress” in any mysterious sense. It deletes the last two summands.

What you keep	the structure shared across a group of words
What you lose	what distinguishes members of that group

So the smoothing we are about to observe is not a side effect. It is the definition of what we did.

And you can already predict the antonym problem from here: hot and cold differ in a low- σ direction, and low- σ directions are what truncation removes.

How Wrong Is the Result? The Frobenius Norm

To say “closest” we need to measure the distance between two matrices. The simplest way: **flatten each one into a long vector and take the ordinary length of the difference.**

$$\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2}$$

No new ideas — the Euclidean norm, applied to a matrix that has been unrolled.

Worked.

$$A = \begin{pmatrix} 2 & 1 \\ 2 & 0 \end{pmatrix} \longrightarrow (2, 1, 2, 0) \longrightarrow \|A\|_F = \sqrt{4 + 1 + 4 + 0} = \sqrt{9} = 3$$

For a *distance*, subtract first and then measure: $\|A - B\|_F$ is the length of the vector of cell-by-cell differences. That is the quantity Eckart–Young minimizes.

Each Singular Value Is Itself a Size

Before the identity, one step that is easy to miss.

$\mathbf{u}_i$ and $\mathbf{v}_i$ are **unit** vectors, and for unit vectors the outer product has Frobenius norm exactly 1:

$$\|\mathbf{u}_i \mathbf{v}_i^\top\|_F^2 = \sum_{j,k} u_j^2 v_k^2 = \left(\sum_j u_j^2 \right) \left(\sum_k v_k^2 \right) = 1 \cdot 1 = 1$$

$$\|\sigma_i \mathbf{u}_i \mathbf{v}_i^\top\|_F = \sigma_i$$

σ_i is the **Frobenius size of the i th term**. Not a score attached to it — its actual measured size.

Note the direction. The norm does not *rank* the singular values — the SVD produces them already sorted, and each one is a measurement of its own piece. Changing the norm would not reshuffle them; it would change which truncation is optimal.

On our matrix the four terms measure 2.127, 2.016, 0.292, 0.084 — the singular values, exactly.

The Singular Values *Are* the Size of the Matrix

A useful identity

$$\|M\|_F = \sqrt{\sum_i \sigma_i^2}$$

Check on our matrix: $\sqrt{2.127^2 + 2.016^2 + 0.292^2 + 0.084^2} = \sqrt{8.681} = 2.946$ — and squaring all twenty-four entries of M and adding gives 8.681 too.

So the singular values are not just importance scores — squared and summed, they *are* the size of the matrix.

Why it is true. The SVD writes M as a sum of rank-1 pieces that are *mutually orthogonal* — the property we noted about U and V . For orthogonal pieces, squared lengths add:

$$\|M\|_F^2 = \sum_i \|\sigma_i \mathbf{u}_i \mathbf{v}_i^\top\|_F^2 = \sum_i \sigma_i^2$$

That is Pythagoras, in a space whose points are matrices — each term contributes its own σ_i^2 and nothing else, with no cross terms, because orthogonal pieces do not interfere.

Which is also why dropping a term costs you exactly σ_i^2 and not more. That is the next slide.

Why This Is the Right Tool: Eckart–Young

Keep only the top k directions:

$$M_k = U_k \Sigma_k V_k^\top, \quad \text{rank}(M_k) = k.$$

Eckart–Young–Mirsky

M_k is the **best** rank- k approximation to M in Frobenius norm:

$$\|M - M_k\|_F = \min_{\text{rank}(X) \leq k} \|M - X\|_F = \sqrt{\sum_{i>k} \sigma_i^2}.$$

Read the right-hand side against the previous slides: **the error is exactly the terms you discarded**, squared and summed. On our matrix, $\sqrt{0.292^2 + 0.084^2} = 0.304$, and computing $\|M - M_2\|_F$ directly gives 0.304.

So the singular values are the **error budget**: before truncating you know what it will cost. This is where the “retain 90% of the variance” heuristic comes from.

“In Frobenius norm” is a choice, and not obviously the right loss for a co-occurrence matrix. We return to this at GloVe’s weighting function.

Why That Theorem Is Not Obvious

It is easy to hear Eckart–Young as a restatement of the definition. It is not.

What we did	sorted the terms by σ and kept the first k
What was claimed	no <i>other</i> rank- k matrix, however constructed, is closer

Truncation is a **greedy** procedure: take the biggest piece, then the next, never reconsidering. Greedy algorithms are usually not optimal — the field of dynamic programming exists because planning ahead normally beats them.

Here greedy *is* optimal, provably. The rank-1 pieces are mutually orthogonal, so each contribution is independent of the others — nothing is gained by trading a large term for two small ones.

That orthogonality is what U and V are for, and it is why this factorization gets a theorem when others do not.

The result holds for any unitarily invariant norm, not just Frobenius — Mirsky's extension of Eckart and Young's 1936 theorem.

Quick Check

Before we run the recipe — three questions on what we just built.

- 1 The matrix $\begin{pmatrix} 2 & 4 \\ 3 & 6 \end{pmatrix}$ — what is its rank? Write it as an outer product $\mathbf{u}\mathbf{v}^\top$.
- 2 Our matrix had $\sigma_3 = 0.292$ and $\sigma_4 = 0.084$. What is the relative error $\|M - M_3\|_F / \|M\|_F$ if we truncate at $k = 3$ instead of 2?
- 3 $\mathbf{u}_3$ had entries of opposite sign on coffee and tea. What would it have meant if $\mathbf{u}_3$ had been all non-negative?

Quick Check

Before we run the recipe — three questions on what we just built.

- 1 The matrix $\begin{pmatrix} 2 & 4 \\ 3 & 6 \end{pmatrix}$ — what is its rank? Write it as an outer product $\mathbf{u}\mathbf{v}^\top$.
- 2 Our matrix had $\sigma_3 = 0.292$ and $\sigma_4 = 0.084$. What is the relative error $\|M - M_3\|_F / \|M\|_F$ if we truncate at $k = 3$ instead of 2?
- 3 $\mathbf{u}_3$ had entries of opposite sign on coffee and tea. What would it have meant if $\mathbf{u}_3$ had been all non-negative?

Answers.

- 1 Rank 1 — the second column is twice the first, and the second row is $1.5\times$ the first. $\mathbf{u} = (2, 3)^\top$, $\mathbf{v} = (1, 2)$.
- 2 $0.084/2.946 = 0.029$, under 3%. At $k = 2$ it was $0.304/2.946 = 10.3\%$.
- 3 It would have been a third *topic* — another group of words sharing a pattern — rather than a distinction drawn inside an existing group. **Sign structure in $\mathbf{u}$ is how you tell the two apart.**

The Recipe

- 1 Build the PPMI matrix M , size $|V| \times |C|$.
- 2 Compute $M = U\Sigma V^\top$.
- 3 Choose k .
- 4 Take the word vectors to be the rows of

$$W = U_k \Sigma_k^\alpha, \quad \alpha \in [0, 1].$$

$\alpha = 1$	$W = U_k \Sigma_k$	the “natural” choice; preserves distances in M
$\alpha = 0.5$	$W = U_k \Sigma_k^{1/2}$	symmetric between words and contexts
$\alpha = 0$	$W = U_k$	discards the singular values entirely

Empirically, $\alpha = 0$ often wins — throwing away the very quantities that told you which directions mattered.

That is the third instance of the pattern Session 7 named: a principled quantity, modified by an unprincipled exponent, because the modification works. Keep counting them.

Worked: the Session 7 Matrix

Take the PPMI matrix you computed by hand last session. Four words, six contexts.

	the	cup	brew	screen	battery	morning
coffee	0	0.76	0.85	0	0	0.84
tea	0	0.96	0.98	0	0	0.50
laptop	0.01	0	0	1.05	1.00	0
tablet	0	0	0	1.04	1.16	0

Its singular values:

$$\sigma_1 = 2.127, \quad \sigma_2 = 2.016, \quad \sigma_3 = 0.292, \quad \sigma_4 = 0.084.$$

Fraction of $\sum \sigma_i^2$ retained	$k = 1$	$k = 2$	$k = 3$
	52.1%	98.9%	99.9%

There is a visible gap between σ_2 and σ_3 — a factor of seven. The matrix is telling us it is essentially two-dimensional.

The Two-Dimensional Word Vectors

Taking $k = 2$ and $\alpha = 1$:

	dim 1	dim 2
coffee	0.000	1.405
tea	0.000	1.447
laptop	1.448	0.000
tablet	1.558	0.000

Six numbers per word became two, and the two clusters landed on separate axes.

Pair	Cosine, 6 dims	Cosine, 2 dims
coffee/tea	0.959	1.000
laptop/tablet	0.997	1.000
coffee/laptop	0.000	0.000

Two-thirds of the coordinates discarded, and every similarity judgment preserved or sharpened.

What Do the Latent Dimensions Mean?

The rows of V^T say which contexts each latent direction is built from.

Direction	Loads on	Weight
dim 1	battery	0.719
	screen	0.696
	everything else	≈ 0
dim 2	brew	0.645
	cup	0.604
	morning	0.467

You would be forgiven for calling these “device-ness” and “beverage-ness.” Nobody supplied those categories; they fell out of the factorization.

But do not generalize from this. On a real matrix the latent directions are *not* interpretable: they are arbitrary orthogonal combinations of tens of thousands of contexts. This example is interpretable because it was built to be.

The Reconstruction, and What It Changed

Multiply the pieces back: $M_2 = U_2 \Sigma_2 V_2^\top$.

	the	cup	brew	screen	battery	morning
coffee	0.00	0.85	0.91	0.00	0.00	0.66
tea	0.00	0.87	0.93	0.00	0.00	0.68
laptop	0.00	0.00	0.00	1.01	1.04	0.00
tablet	0.00	0.00	0.00	1.08	1.12	0.00

Relative error $\|M - M_2\|_F / \|M\|_F = 10.3\%$. But look at *which* entries moved:

Cell	Observed	Reconstructed
(coffee, cup)	0.76	0.85 pulled <i>up</i> toward tea
(coffee, morning)	0.84	0.66 pulled <i>down</i> toward tea
(tea, morning)	0.50	0.68 pulled <i>up</i> toward coffee

Reduction did not merely compress — it **smoothed**. Each profile moved toward the profiles of the words it resembles. That is the inductive bias, doing its work in plain sight.

The Demonstration That Matters

Smoothing is nice. The real claim was **generalization**: similarity between words that share no context at all. Here is a corpus built to test it.

	cup	brew	morn.	shot	foam	roast	screen	batt.	charg.	app	boot
coffee	32	31	27	0	0	21	0	0	0	0	0
tea	27	0	23	16	0	0	0	0	0	0	0
espresso	0	20	0	16	0	22	0	0	0	0	0
cappuccino	25	0	20	0	11	0	0	0	0	0	0
laptop	0	0	0	0	0	0	32	31	0	25	20
tablet	0	0	0	0	0	0	36	0	20	25	0
phone	0	0	0	0	0	0	0	32	20	0	27

Check the two red rows. espresso was seen with brew, shot, roast. cappuccino was seen with cup, morning, foam. **They share no context whatsoever.**

The counts were generated from two latent topics and then sparsified, which is a reasonable caricature of what a real corpus does to a real lexicon.

Before: Full PPMI, Eleven Dimensions

Pair		Cosine
espresso / cappuccino	no shared context	0.000
espresso / coffee	shares brew, roast	0.656
coffee / tea	shares cup, morning	0.383
laptop / phone	shares battery, boot	0.565
tablet / phone	shares charger	0.315
espresso / laptop	different topics	0.000

Two words from the same small semantic field, both plainly beverages, score **exactly zero** — the same score as a beverage against a computer.

This is the sparsity problem in its purest form. The cosine is not low because the evidence says they differ. It is zero because the evidence is silent.

After: Two Dimensions

Pair	11 dims	$k = 2$
espresso / cappuccino	0.000	1.000
espresso / coffee	0.656	1.000
coffee / tea	0.383	1.000
laptop / phone	0.565	1.000
tablet / phone	0.315	1.000
espresso / laptop	0.000	0.000
cappuccino / phone	0.000	0.000

Every within-topic pair goes to 1. Every cross-topic pair stays at 0.

The model has asserted that espresso and cappuccino are similar on the basis of evidence that never mentioned them together. It reached that through coffee, which overlaps with both — **transitivity through a shared latent direction.**

And Now the Uncomfortable Part

Run the same computation at $k = 3$.

Pair	11 dims	$k = 2$	$k = 3$
espresso / cappuccino	0.000	1.000	-0.036
espresso / coffee	0.656	1.000	0.914
coffee / tea	0.383	1.000	0.940

One extra dimension destroys the result.

The reason is exactly the mechanism that produced it. At $k = 2$ the model has no room to represent anything but the two topics, so it must explain espresso and cappuccino with the same direction. At $k = 3$ it has room to re-fit the accident of *which* contexts each word happened to be observed with — and that accident is precisely what made them disjoint.

The generalization was not free. It was bought by denying the model the capacity to fit the sparsity pattern.

How Should You Choose k ?

The standard heuristic is to retain some fixed fraction of $\sum \sigma_i^2$. Apply it here:

	Variance retained	espresso/cappuccino
$k = 2$	55.4%	1.000
$k = 3$	76.7%	-0.036

A “retain 75% of the variance” rule picks $k = 3$ and throws away the effect we wanted.

Why the heuristic fails

Frobenius norm measures reconstruction of the *observed* matrix. But the observed matrix is not the target — its zeros are missing data, not evidence of dissimilarity. We are asking the model to be **wrong** about the observations in a particular useful way, and a criterion that rewards fidelity to observations cannot express that.

In practice k is chosen by downstream evaluation — not a failure of rigor, but the admission that we have no intrinsic criterion for what the latent space should contain.

A Word on Cost

Method	Complexity	$ V = 50,000$
Full SVD	$O(mn \min(m, n))$	infeasible
Truncated (Lanczos)	$O(\text{nnz} \cdot k)$	hours
Randomized SVD	$O(\text{nnz} \cdot k + (m+n)k^2)$	minutes

Two practical facts worth carrying:

- The methods that make this tractable exploit **sparsity** — which is why PPMI's clip is load-bearing twice over, once for quality and once for feasibility.
- SVD is a *batch* method. Adding a new document means recomputing. This turns out to be one of the real practical advantages of the methods we meet next, and it is rarely the one cited.

Halko, Martinsson & Tropp (2011) for randomized methods; the relevant point for us is that “count-based methods do not scale” was an overstatement even in 2013.

LSA: the First Version of This Argument

Deerwester, Dumais, Furnas, Landauer & Harshman (1990)

Apply truncated SVD to a **term–document** matrix. Retrieve documents by cosine in the reduced space rather than by term overlap.

The motivating problem was stated in retrieval terms but is exactly ours:

Synonymy	query <i>car</i> , document says <i>automobile</i> , overlap is zero
Polysemy	query <i>bank</i> , document is about rivers, overlap is spurious

Reduction addresses the first, for the reason we just demonstrated.

It does not address the second, and LSA's authors said so. One vector per term means one vector for both senses of bank — which is Session 9's entire subject.

The Result That Made People Pay Attention

Landauer & Dumais (1997) trained LSA on roughly 4.6 million words of encyclopedia text and gave it the synonym section of the TOEFL exam.

	Correct
LSA, 300 dimensions	64.4%
Average non-native applicant to a US university	64.5%
Chance (four alternatives)	25%

No dictionary, no thesaurus, no supervision, no linguistic knowledge of any kind.
Co-occurrence counts, a truncated SVD, and a cosine.

Whatever one concludes about the strength of the claim, the empirical fact is a real one and it was obtained in 1997, sixteen years before word2vec.

Landauer and Dumais's Strong Claim

They went further than reporting a result. Their paper is titled *A Solution to Plato's Problem*, and the argument runs:

- ① Children acquire vocabulary far faster than direct exposure to each word can explain.
- ② Reduction lets evidence about word A inform the representation of word B that never co-occurred with it — exactly what we just watched happen.
- ③ Therefore dimensionality reduction may be a mechanism of **induction**, not merely compression, and may model how sparse exposure yields dense lexical knowledge.

Step 2 is a theorem about linear algebra. Step 3 is a claim about cognition, and it rests on the inductive bias being the right one.

Our $k = 3$ experiment is the smallest possible counterweight: the same mechanism, with one parameter changed, produces the opposite verdict on the same data. A mechanism whose output flips with a hyperparameter is a candidate explanation, not an established one.

The claim has been productively argued about for thirty years, which is more than most claims in this field manage.

The Second Route: Stop Counting, Start Predicting

Mikolov, Chen, Corrado & Dean (2013)

Do not build a matrix. Set up a **prediction task**, train a model with a small hidden layer, and take the learned parameters as the embeddings.

	Count-based	Prediction-based
Build	explicit $ V \times C $ matrix	nothing explicit
Then	factorize	optimize by SGD
Passes over corpus	one	several
New data	recompute	continue training
Vectors are	rows of $U_k \Sigma_k^\alpha$	learned parameters

The compression is now a *byproduct*: the hidden layer is narrow, so the model must find a d -dimensional code that supports the prediction. Nobody computes a rank- k approximation of anything — or so it appears.

Skip-Gram

Given a centre word w , predict each context word c in a window around it.



Maximize, over the corpus:

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

with the model

$$P(c \mid w) = \frac{\exp(\mathbf{v}_c \cdot \mathbf{v}_w)}{\sum_{c' \in V} \exp(\mathbf{v}_{c'} \cdot \mathbf{v}_w)}.$$

Note the two vectors per word: $\mathbf{v}_w$ when it is the centre, $\mathbf{v}_c$ when it is the context. This detail looks like an implementation convenience. It is not — it is what makes the factorization interpretation work.

The Denominator Is the Problem

$$P(c \mid w) = \frac{\exp(\mathbf{v}_c \cdot \mathbf{v}_w)}{\sum_{c' \in V} \exp(\mathbf{v}_{c'} \cdot \mathbf{v}_w)}$$

That sum runs over the entire vocabulary, for **every training example**. With $|V| = 10^5$ and 10^9 tokens in the corpus, each with a handful of context words, this is arithmetic on the order of 10^{14} dot products per epoch.

Two standard fixes:

- **Hierarchical softmax** — arrange the vocabulary as a binary tree, reducing the cost to $O(\log |V|)$ per example.
- **Negative sampling** — replace the whole probabilistic model with a different objective. This is the one that matters for us.

Notice the shape of the move: the softmax was the principled formulation, and it is abandoned for tractability. What replaces it is not an approximation to it.

Skip-Gram with Negative Sampling

Reframe. Instead of asking “which of $|V|$ words comes next,” ask a binary question: **was this (w, c) pair actually observed, or did I make it up?**

For each observed pair (w, c) , draw k fake context words $c_1, \dots, c_k$ from a noise distribution and maximize

$$\log \sigma(\mathbf{v}_c \cdot \mathbf{v}_w) + \sum_{i=1}^k \mathbb{E}_{c_i \sim P_n} [\log \sigma(-\mathbf{v}_{c_i} \cdot \mathbf{v}_w)]$$

where $\sigma(x) = 1/(1 + e^{-x})$.

First term	push $\mathbf{v}_w$ and $\mathbf{v}_c$ together for pairs that occurred
Second term	push $\mathbf{v}_w$ and $\mathbf{v}_{c_i}$ apart for k pairs that did not

One $|V|$ -way classification has become $k + 1$ binary ones. Typical k : 5 to 20 for small corpora, 2 to 5 for large ones.

The Noise Distribution: a Number You Have Seen

What is P_n , the distribution the fake context words are drawn from?

$$P_n(c) = \frac{\#(c)^{0.75}}{\sum_{c'} \#(c')^{0.75}}$$

That is the same 0.75 from Session 7.

Last session it appeared as *context distribution smoothing*, a fix for PMI overweighting rare contexts. Here it is the negative-sampling distribution, and Mikolov et al. report only that it “outperformed significantly” the alternatives tried.

Same exponent, same job

Both flatten the context distribution: rare contexts get a higher effective probability, frequent ones lower. In Session 7 that stopped barista beating cup on three observations; here it stops rare words being useless as negatives.

This is the first hint that the two methods are doing the same thing. It will not be the last.

Subsampling: the Third Solution to the

word2vec discards each token w during training with probability

$$P_{\text{drop}}(w) = 1 - \sqrt{\frac{t}{f(w)}}, \quad t \approx 10^{-5}$$

where $f(w)$ is the word's corpus frequency. Frequent words are dropped aggressively; rare ones are kept.

Session 7, PMI	the gets weight ≈ 0 , by the formula
Session 7, $\alpha = 0.75$	rare contexts are damped, by an exponent
Today, subsampling	the is deleted from the corpus, probabilistically

Three lectures, three unrelated-looking mechanisms, one underlying problem: high-frequency words dominate the counts and carry no information.

A second effect is easy to miss: deleting tokens *widens* the effective window, since the window is counted after deletion. Part of word2vec's reported gain comes from this, not the objective.

CBOW, Briefly

The other architecture in the same paper: predict the centre word *from* the sum of its context vectors.

	Skip-gram	CBOW
Predicts	context from centre	centre from context
Training examples	one per (centre, context)	one per position
Speed	slower	faster
Rare words	better	worse
Small corpora	better	worse

Skip-gram with negative sampling became the default, and it is the variant with the clean theoretical analysis.

Note in passing what CBOW does to compose a context: it **sums** the vectors. Session 10 will have something to say about that choice.

What Is SGNS Actually Computing?

By 2014 the picture in the literature was roughly: counting is the old way, prediction is the new way, and prediction wins.

Levy & Goldberg (2014), *Neural Word Embedding as Implicit Matrix Factorization*

SGNS with k negative samples, at its optimum, satisfies

$$\mathbf{v}_w \cdot \mathbf{v}_c = \text{PMI}(w, c) - \log k$$

for every pair (w, c) . That is: **SGNS is factorizing the shifted PMI matrix.**

The same PMI you computed by hand last session, shifted by a constant that depends only on the number of negative samples.

The derivation is short. It is worth doing, because it is one of very few places in Part II where a claim about a neural method can be settled by algebra rather than by benchmark.

The Derivation, Part 1: Set Up

Write $x = \mathbf{v}_w \cdot \mathbf{v}_c$ and let $\#(w, c)$, $\#(w)$, $\#(c)$, N be the counts from Session 7.

If d is large enough, the vectors are expressive enough that each x can be chosen **independently** of the others. So the global objective decomposes, and we can optimize one cell at a time.

The objective contributed by the pair (w, c) :

$$\ell(x) = \#(w, c) \log \sigma(x) + k \cdot \frac{\#(w) \#(c)}{N} \log \sigma(-x).$$

$\#(w, c)$	how often the pair was observed — the positive term
$k \#(w) \#(c) / N$	how often it is expected as a negative sample

The second expression is the expected number of times (w, c) is drawn as a fake pair: w appears $\#(w)$ times, each draws k negatives, and c is drawn with probability $\#(c)/N$.

The Derivation, Part 2: Differentiate

Using $\frac{d}{dx} \log \sigma(x) = \sigma(-x)$ and $\frac{d}{dx} \log \sigma(-x) = -\sigma(x)$:

$$\frac{d\ell}{dx} = \#(w, c) \sigma(-x) - k \frac{\#(w) \#(c)}{N} \sigma(x) = 0.$$

Divide by $\sigma(x)$ and use $\sigma(-x)/\sigma(x) = e^{-x}$:

$$\#(w, c) e^{-x} = k \frac{\#(w) \#(c)}{N} \implies e^x = \frac{\#(w, c) N}{\#(w) \#(c)} \cdot \frac{1}{k}.$$

Take logs:

$$x = \log\left(\frac{\#(w, c) \cdot N}{\#(w) \cdot \#(c)}\right) - \log k = \boxed{\text{PMI}(w, c) - \log k}$$

Four lines. The quantity SGNS drives each dot product toward is the PMI of the pair, offset by $\log k$.

Logarithms here are natural, inherited from the sigmoid. Session 7 used $\log_2$, so this PMI is the earlier one times $\ln 2$ — a global constant that changes no cell's sign and no similarity ranking.

Reading the Result

Observation	Consequence
The target is $\text{PMI} - \log k$	SGNS is a count-based method
k shifts every cell by a constant	more negatives $\Rightarrow$ more cells pushed below zero
Cells with $\#(w, c) = 0$	never updated; effectively omitted, not set to $-\infty$
$d < V $	exact solution unreachable $\Rightarrow$ implicit low-rank factorization

Look at the second and third rows together. Shifting by $-\log k$ and then never visiting the zero cells has almost exactly the effect of **clipping negatives** — which is PPMI, arrived at by an entirely different route.

The two routes, aligned

Route 1: build PPMI, truncate the SVD. **Route 2:** optimize SGNS.

Route 2 solves a low-rank factorization of a shifted PMI matrix, by stochastic gradient descent instead of by SVD, with a different implicit loss.

So What Actually Changed in 2013?

Levy, Goldberg & Dagan (2015) tuned the count-based pipeline with word2vec's hyperparameters — context distribution smoothing, shifted PPML, subsampling, the α exponent — and compared fairly.

Finding	Detail
No consistent winner	tuned PPML+SVD matches SGNS on most similarity and analogy benchmarks
Hyperparameters dominate	the gap between settings exceeds the gap between methods
The transferable part	word2vec's <i>hyperparameters</i> , not its architecture, carried much of the gain

What did change: speed, memory, streaming rather than materializing a matrix, and an implementation anyone could run. Those are substantial, and they are engineering.

What did not change is the theory of meaning. Both routes represent a word by its weighted co-occurrence profile and compare by inner product.

GloVe: the Third Route Says So Out Loud

Pennington, Socher & Manning (2014)

Fit the log co-occurrence count directly:

$$\min \sum_{w,c} f(\#(w,c)) \left(\mathbf{v}_w \cdot \tilde{\mathbf{v}}_c + b_w + \tilde{b}_c - \log \#(w,c) \right)^2$$

Explicitly count-based, explicitly a factorization, motivated by the observation that *ratios* of co-occurrence probabilities carry the structure.

The weighting function

$$f(x) = \min \left((x/x_{\max})^{0.75}, 1 \right)$$

downweights rare pairs and caps frequent ones.

The exponent is 0.75 again — a third appearance, in a third role.

And note what f is for: it repairs the Frobenius-norm loss, which weights every cell equally. GloVe replaces “best in Frobenius norm” with “best in a loss that knows some cells are better estimated than others” — the qualification flagged on the Eckart–Young slide.

Three Routes, One Table

	SVD on PPMI	SGNS	GloVe
Matrix	explicit	implicit	explicit
Target quantity	$\text{PPMI}(w, c)$	$\text{PMI}(w, c) - \log k$	$\log \#(w, c)$
Loss	Frobenius	logistic	weighted least squares
Cell weighting	uniform	by frequency	$f(x)$, capped
Zero cells	clipped to 0	omitted	omitted
Solved by	eigendecomposition	SGD	SGD
Streaming	no	yes	no (needs counts first)

Read across any row: these are variations on *how to weight and how to optimize*, not variations on what a word is.

All three produce $\mathbf{v}_w \in \mathbb{R}^d$ such that $\mathbf{v}_w \cdot \mathbf{v}_c$ approximates a monotone function of the PMI of the pair. That is the destination the three routes share.

The Gains, Stated Precisely

Gain	Mechanism
Generalization	espresso/cappuccino: 0.000 $\rightarrow$ 1.000 via a shared latent direction
Compactness	$ C \approx 10^5$ becomes $d \approx 300$
Usability	a fixed-width real feature vector, which every downstream model wants
Differentiability	embeddings can be a <i>layer</i> , trained end to end with the task
Streaming	SGNS updates incrementally; no matrix to re-build

The fourth row has the largest downstream consequences, and is not about semantics at all: once a word representation is a differentiable parameter, it stops being preprocessing and becomes part of the model.

That is the road to Session 9: if the embedding is a layer, nothing requires it to be a lookup table.

The Analogy Result

The finding that did more than any other to persuade people that embeddings encode structure:

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

and similarly for *Paris: France:: Rome: Italy, walking: walked:: swimming: swam*.

The apparent implication: relations are **directions**, and the space has linear structure that nobody put there.

If true in the strong form, this would be significant for us — it would suggest that some relational structure survives in the geometry, which is exactly the kind of thing Part IV wants.

So it is worth examining what the result actually shows.

The Analogy Result, Examined

The standard evaluation computes

$$\arg \max_{b^* \in V \setminus \{a, a^*, b\}} \cos(\mathbf{v}_{b^*}, \mathbf{v}_{a^*} - \mathbf{v}_a + \mathbf{v}_b)$$

Finding	Source
The three query words are <i>excluded</i> from the answer set	standard in every implementation
Without that exclusion, the top answer is usually b itself	Linzen (2016)
A baseline that just returns the nearest neighbor of b scores well	Linzen (2016)
$\cos$ multiplication (3CosMul) beats addition	Levy & Goldberg (2014)
Performance varies sharply by relation type	several

Much of the effect is neighborhood structure plus a rule forbidding the obvious wrong answers. The arithmetic contributes, but less than the headline suggests and unevenly across relations. There is real linear structure for some relations — morphological and geographic especially — and the demonstration overstates how much of the work the arithmetic does.

What Was Lost

Loss	Detail
Interpretable coordinates	a PPMI column was cup; dimension 137 is nothing
Sparsity	every word now has a nonzero value in every dimension
Auditability	you can check a PPMI cell against the corpus; you cannot check a trained weight
The exact regime	no coordinate per entity, so 6B's encoding is now unreachable

The last row closes a loop opened in Session 6B, whose “Changing the Base Case” slide replaced the one-hot $\mathbf{e}_i$ with a dense learned $\mathbf{v}_i$. Session 7 gave us non-orthogonality but kept a coordinate per context.

Today the coordinates are gone: no basis vector corresponds to anything, not an entity and not a context. 6B's approximate regime has arrived in full — $d \ll n$, dense, learned, and no way back.

The Ledger, Unchanged

	Part I	Session 7	Today
Similarity	—	✓	✓✓
Generalizes to unseen pairs	—	—	✓
Compact	—	—	✓
Differentiable	—	—	✓
Antonyms distinguished	✓	—	—
Negation	✓	—	—
Quantifier scope	✓	—	—
Monotonicity	✓	—	—
Composition	✓	—	— Session 10

The top block improved. The bottom block is identical to Session 7's.

Reduction is a better *lexical* representation. It is not a step toward compositional semantics, and three hundred dimensions of it are no closer than fifty thousand were.

One Loss Worth Dwelling On: Antonyms Got Worse

Session 7 established that hot and cold are distributionally near-identical, and argued this was a principled consequence rather than a bug.

Reduction **amplifies** it. The mechanism is the one we demonstrated:

- Low-rank approximation pulls each word toward words with similar profiles.
- hot and cold have similar profiles by construction.
- So they are pulled toward each other, and the small idiosyncratic differences that distinguished them are exactly the low-variance directions that truncation discards.

The distinguishing information lives in the tail of the spectrum, and the tail is what we threw away.

This is the same structure as the $k = 3$ result earlier, running the other direction: truncation is a choice about which distinctions to preserve, and it has no way of knowing that antonymy matters and topic does not.

Embeddings Encode the Corpus's Associations

Study	Finding
Bolukbasi et al. (2016)	<i>man</i> : <i>computer</i> <i>program-</i> <i>mer</i> :: <i>woman</i> : <i>homemaker</i> recovered by the analogy method
Caliskan, Bryson & Narayanan (2017)	embeddings reproduce human implicit- association (IAT) effects, including ones tracking documented social bias

The structure of the argument is identical to the antonym argument: (1) the method represents a word by the company it keeps; (2) in the corpus, these words keep that company; (3) therefore the representation encodes the association.

This is the distributional hypothesis working as specified, on a corpus that reflects the world that produced it — not an algorithmic defect, and not fixed by a better optimizer.

Gonen & Goldberg (2019) showed that standard debiasing projections remove the *direction* while leaving the clustering intact.

Practice

1. The Session 7 matrix had $\sigma_3 = 0.292$ and $\sigma_4 = 0.084$. What is the relative Frobenius error at $k = 3$? At $k = 1$?
2. In the SGNS derivation we assumed each $\mathbf{v}_w \cdot \mathbf{v}_c$ could be set independently. For what d is that exactly true, and what goes wrong below it?
3. SGNS drives $\mathbf{v}_w \cdot \mathbf{v}_c \rightarrow \text{PMI}(w, c) - \log k$. What value of k makes the target equal to PMI itself? What happens to the sparsity of the implied matrix as k grows?
4. Take the generalization corpus. Add one occurrence of espresso with foam. Predict — before computing — whether the $k = 3$ cosine for espresso/cappuccino rises above zero, and say why.
5. **Harder.** CBOW composes a context by summing vectors. Give two English sentence pairs that CBOW's context representation cannot distinguish, and say which property of addition is responsible.

Exercise 5 is a rehearsal for Session 10. Keep your answer.

Summary

Point	Detail
Why reduce SVD	sparsity, generalization, dimensionality — the second is the interesting one $M = U\Sigma V^T$; truncation is optimal in Frobenius norm (Eckart–Young)
Worked example	98.9% of the Session 7 matrix in two dimensions
Latent dimensions	interpretable here <i>only</i> because the example was built that way
Reduction smooths	each profile is moved toward the profiles it resembles
Generalization	espresso/cappuccino: 0.000 at 11 dims, 1.000 at $k = 2$
The catch	−0.036 at $k = 3$; capacity to fit sparsity undoes the effect
Choosing k	the variance heuristic picks wrong; only downstream evaluation decides
LSA	1990/1997; TOEFL at 64%; “Plato’s problem” is a cognitive claim, not a theorem
SGNS	binary classification against k negatives, sampled $\propto \#(c)^{0.75}$
The theorem	$\mathbf{v}_w \cdot \mathbf{v}_c = \text{PMI}(w, c) - \log k$: SGNS factorizes shifted PMI
Consequence	count vs. predict is not a theoretical distinction
GloVe	explicit counts, weighted least squares; the 0.75 exponent a third time
Gained	generalization, compactness, differentiability, streaming
Lost	interpretable coordinates, sparsity, auditability, the exact regime
Antonyms	worse: the distinguishing signal is in the discarded tail
Bias	the hypothesis working as specified, on a corpus that reflects its world

Reading and Exercises

Reading: Lappin & Shanahan §2.4–2.7 and Ch. 4.

Primary sources:

- Deerwester, Dumais, Furnas, Landauer & Harshman (1990), *Indexing by Latent Semantic Analysis*
- Landauer & Dumais (1997), *A Solution to Plato's Problem* — read §1 and the discussion
- Mikolov, Sutskever, Chen, Corrado & Dean (2013), *Distributed Representations of Words and Phrases*
- Levy & Goldberg (2014), *Neural Word Embedding as Implicit Matrix Factorization* — the derivation is §2–3 and is four pages
- Levy, Goldberg & Dagan (2015), *Improving Distributional Similarity with Lessons Learned from Word Embeddings*
- Pennington, Socher & Manning (2014), *GloVe*

Optional: Linzen (2016) on analogy baselines; Caliskan et al. (2017) on embedding associations.

Problem Set 2 is out today. It asks you to build the pipeline end to end on a real corpus and reproduce, at small scale, both the generalization effect and the k sensitivity.

Questions?

Next: one vector per word is a commitment —
and *bank* has two meanings