What the Inner Product Left Out

From factorization to binding

James Pustejovsky

Brandeis University
Department of Computer Science

CS 135: Computational Semantics

Fall 2026

Where We Are

Three routes to a word vector, all landing in the same place: $\mathbf{v}_w \cdot \mathbf{v}_c$ approximates some monotone function of $\mathrm{PMI}(w, c)$.

That table settled what the routes have in common. It left two things unexplained, and both turn out to matter for where this course is going.

The two questions

1. The table says SVD is solved by eigendecomposition and the other two by gradient descent. **Why?** Not as an engineering fact – as a mathematical one.
2. Every method so far compares meanings with an inner product. **Why that operation**, and what does it cost?

The first question has a sharp answer involving a theorem you already know. The second one opens onto the second half of the course.

Two Rows That Are Really One Row

From the comparison table:

	SVD on PPMI	**SGNS**	**GloVe**
Cell weighting	uniform	by frequency	$f(x)$, capped
Solved by	eigendecomposition	SGD	SGD

These read as two independent facts about three systems. They are not independent. **The second row is a consequence of the first.**

Gradient descent is not what makes the predict models neural. It is what is left once you give up uniform weighting.

What Eckart–Young Promised, Exactly

The theorem, stated carefully

Among all matrices of rank at most k, the truncated SVD M_k minimizes

$$\|M - X\|_F^2 \;=\; \sum_{ij} \left(M_{ij} - X_{ij}\right)^2$$

Look at what the sum does. **Every cell contributes equally.** A cell resting on 48 observations and a cell resting on two are weighted the same, because the Frobenius norm has no idea how the numbers got there.

That is not a limitation of the proof technique. Uniform weighting is a *premise* of the theorem. The spectral decomposition is the answer to that specific question and to no other.

Session 8b put this geometrically: the SVD finds the axes of the ellipsoid, and truncation flattens the shortest ones. Nothing in that picture asks how much evidence each direction rests on.

Why Anyone Would Weight the Cells

Return to the Session 7 counts behind the PPMI matrix:

cell	count	
(coffee, cup)	48	a solid estimate
(laptop, the)	2	essentially noise

Both become a PPMI value, and the Frobenius norm then treats them as equally worth fitting. Given Zipf, most of the matrix is the second kind.

The obvious fix, and its price

Weight each cell by how much evidence it rests on: $\min_{\mathrm{rank}(X)\le k} \sum_{ij} W_{ij}(M_{ij} - X_{ij})^2$. There is no spectral solution to this problem, and weighted low-rank approximation is NP-hard in general.

Srebro & Jaakkola 2003 for the algorithms; Gillis & Glineur 2011 for the hardness.

The Experiment

Same matrix, same rank. Two factorizations, each optimizing a different loss.

A	rank-2 truncated SVD of the Session 7 PPMI matrix
B	rank-2 factorization found by alternating least squares, with each cell weighted by its raw co-occurrence count

Score both on both objectives:

	unweighted loss	**weighted loss**
A: rank-2 SVD	**0.0923**	2.3264
B: rank-2 weighted ALS	0.8989	**1.2294**

Reading the Result

	unweighted	**weighted**
rank-2 SVD	$\mathbf{0.0923}$	2.3264
rank-2 weighted ALS	0.8989	$\mathbf{1.2294}$

Each wins its own column and loses the other. The SVD is nearly ten times better on the unweighted objective, exactly as Eckart–Young guarantees it must be, and 47% worse on the weighted one.

Neither factorization is better. They are answers to different questions.

And the second question has no closed form, which is why B had to be found by iteration. Nothing about B is neural, probabilistic, or predictive. It is least squares with weights, and that alone was enough to leave linear algebra behind.

What This Settles About "Count versus Predict"

The usual framing	Counting methods use linear algebra; predicting methods use neural networks and gradient descent. A change of paradigm.
What is actually true	All three methods factor a matrix of association scores. They differ in *which cells they trust*. Uniform trust admits a spectral solution; non-uniform trust does not.

Levy and Goldberg made this point algebraically: SGNS is implicitly factoring shifted PMI. The weighting argument makes it geometrically concrete – SGNS is factoring roughly the same matrix the SVD does, in roughly the same way, while caring more about the cells it has seen often.

That is a real difference and it does help. It is not a different theory of what a word is.

Three Routes, and a Fourth

There is a fourth way to get from a large sparse count matrix to a small dense one. It is older than word2vec, and it is the door into the second half of this course.

	How the dimensions are chosen	**Cost**
SVD	computed from the data	factor the whole matrix
SGNS, GloVe	learned from the data	many passes of SGD
Random indexing	chosen in advance, at random	one pass, no factorization

Kanerva and Sahlgren, from the late 1990s. Incremental, streaming, never materializes the co-occurrence matrix – every property word2vec was praised for in 2013, fifteen years earlier.

It is also, though nobody called it that at the time, a vector symbolic architecture.

Random Indexing

1. Give every *context* a random sparse vector of dimension d: mostly zeros, a handful of ± 1 entries in random positions. Call it the context's **index vector**.
2. Give every *word* a zero vector of dimension d.
3. Walk the corpus. Each time word w occurs with context c, add c's index vector to w's.

That is the entire algorithm. The result is PR where P is the count matrix and R is the random $n_C \times d$ index matrix, computed without ever building P.

Why it could possibly work

Johnson–Lindenstrauss (1984). Any n points in any dimension can be projected onto $O(\log n/\epsilon^2)$ random dimensions with all pairwise distances preserved to within a factor $1 \pm \epsilon$. The bound depends on the *number of points*, not on the dimension they started in.

Does It Work?

300 words, 3000 contexts, PPMI, 3.8% dense. Compare each reduction's cosines against the full-dimensional cosines.

	SVD		**Gaussian proj.**		**Random indexing**	
d	err	corr	err	corr	err	corr
50	0.2444	0.744	0.1103	0.310	0.1086	0.270
100	0.1280	0.836	0.0782	0.401	0.0770	0.395
200	0.0346	0.934	0.0573	0.531	0.0573	0.530
400	0.0000	1.000	0.0407	0.661	0.0374	0.666

err = mean absolute cosine error on genuinely similar pairs; corr = correlation with the full-dimensional cosines over all pairs.

The three random projections are **indistinguishable from each other**: the design of the random matrix barely matters, which is what Johnson–Lindenstrauss predicts.

And the two metrics disagree about who wins. Random projection has the *lower error* on similar pairs; the SVD has the far *higher correlation*. Something more interesting than "one is better" is going on.

Bias and Variance

Split the error into its systematic and random parts, at $d = 100$:

	mean signed error	**sd of error**	**mean |error|**
SVD	$+0.0498$	0.0615	0.0594
Random projection	-0.0065	0.0988	0.0792

And split by whether the pair was similar to begin with:

	dissimilar pairs	**similar pairs**
SVD	$+0.0225$	$+0.1280$
Random projection	$+0.0094$	$+0.0156$

The SVD is **biased and low variance**: it systematically inflates similarity, and inflates the already-similar pairs eight times harder. Random projection is **nearly unbiased and high variance**: it preserves what was there and adds noise.

Two Kinds of Reduction

That $+0.128$ is not an error. It is the espresso and cappuccino effect, measured across 1464 word pairs.

	SVD	**Random projection**
Basis	derived from the data, exactly orthogonal	assigned in advance, approximately orthogonal
What it does	discovers structure and imposes it	preserves distances
Similarity it reports	partly invented	only what was observed
Good for	generalizing past sparsity	representing without distorting

These are not competing implementations of one idea. The SVD reduces by *finding* structure; random projection reduces by *refusing to look for any*.

Which raises the question the rest of this deck answers: if your reduction invents nothing, and you still want structure in the space, **how do you put it there yourself?**

Back to the Outer Product

From Session 8, on two column vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$:

	result	**what happens to the index**
inner $\mathbf{u} \cdot \mathbf{v}$	a scalar	summed away
outer $\mathbf{u}\mathbf{v}^\top$	an $m \times n$ matrix	kept; order rises from 1 to 2

Everything this course has done with word vectors uses the first row. Similarity is $\cos$, which is an inner product. Composition, when we tried it in Session 10, was an inner product. Attention is an inner product.

The inner product is not a neutral choice of similarity measure. It is the operation that *throws away which dimension agreed with which*. That is precisely the information composition needs, which is why vectors compose badly.

Bind With the Whole Outer Product

Smolensky's tensor product representations (1990) take the other row seriously. To bind a role to a filler, take their outer product:

$$\mathbf{r} \otimes \mathbf{f} \;=\; \mathbf{r}\mathbf{f}^{\top}$$

A structure is then a sum of bindings:

$$\mathbf{S} \;=\; \mathbf{r}_{\text{agent}} \otimes \mathbf{f}_{\text{cat}} \;+\; \mathbf{r}_{\text{verb}} \otimes \mathbf{f}_{\text{chased}} \;+\; \mathbf{r}_{\text{theme}} \otimes \mathbf{f}_{\text{dog}}$$

Recover a filler	$\mathbf{r}_{\text{agent}}^{\top}\mathbf{S}$, exact when the roles are orthonormal
Structure preserved	yes, completely
Invertible	yes, exactly

This is everything the inner product could not give us. It is also the first genuinely compositional vector representation, and it predates all of distributional semantics as practiced here.

The Dimension Problem

Bind two vectors of dimension n and you get n^2 numbers. Bind three and you get n^3.

structure	**order**	**dimension at** $n = 1000$
a word	1	10^3
role $\otimes$ filler	2	10^6
nested one deep	3	10^9
nested two deep	4	10^{12}

Recursive structure is the whole point of a theory of syntax, and tensor products make recursion exponentially expensive in space.

So the design problem is stated: keep the invertibility of the outer product, without keeping its dimension. Everything called a vector symbolic architecture is an answer to that one question.

Fold It Back Down

Holographic Reduced Representations (Plate, 1995) bind with **circular convolution**:

$$(\mathbf{a} \circledast \mathbf{b})_k \;=\; \sum_{i=0}^{n-1} a_i \, b_{(k-i) \bmod n}$$

The output has dimension n, not n^2. And the formula is not a new operation – look at which products it sums.

What circular convolution is

Form the full outer product $\mathbf{a}\mathbf{b}^\top$. Then sum it along its wrapped anti-diagonals: entry k of the result is the sum of every $a_i b_j$ with $i + j \equiv k \pmod{n}$.

Binding is the outer product, folded back down to n numbers.

The Identity, Verified

Two random vectors in $\mathbb{R}^6$:

circular convolution $\mathbf{a} \circledast \mathbf{b}$	$(0.3460,\ 0.3313,\ 0.5021,\ 0.4297,\ -0.9461,\ -0.8325)$
$\mathbf{a}\mathbf{b}^\top$ summed along wrapped anti-diagonals	$(0.3460,\ 0.3313,\ 0.5021,\ 0.4297,\ -0.9461,\ -0.8325)$

$\mathbf{b}^\top$

	0	1	2	3	4	5
	1	2	3	4	5	0
$\mathbf{a}$	2	3	4	5	0	1
	3	4	5	0	1	2
	4	5	0	1	2	3
	5	0	1	2	3	4

every cell labeled $k = (i + j) \bmod 6$; the shaded cells are the six products that sum to give entry 2

The 36 products of the outer product are distributed among 6 output slots, six products each.

What the Folding Costs

Thirty-six numbers went in and six came out, so something was lost. Unbinding is by correlation with the involution $\mathbf{a}^*$, and it recovers $\mathbf{b}$ only approximately:

	$\cos(\text{recovered, correct})$, items superposed			
d	1	3	5	10
64	0.732	0.512	0.396	0.309
1024	0.709	0.500	0.416	0.302
4096	0.708	0.497	0.404	0.300

A single bound pair recovers at 0.708, and more dimensions do not help. That number is $1/\sqrt{2}$, exactly: unbinding multiplies by $|\mathcal{F}(\mathbf{a})|^2$ in the Fourier domain, which for Gaussian $\mathbf{a}$ is exponential with standard deviation equal to its mean, giving $\cos \to 1/\sqrt{2}$ at every d.

The fix is to choose $\mathbf{a}$ **unitary** – unit magnitude in every frequency bin. Then $|\mathcal{F}(\mathbf{a})|^2 = 1$ and binding is *exactly* invertible.

With Unitary Vectors

	$\cos(\text{recovered},\ \text{correct})$			
d	1 item	3	5	10
64	1.000	0.566	0.416	0.317
256	1.000	0.575	0.447	0.309
1024	1.000	0.576	0.451	0.322
4096	1.000	0.577	0.446	0.320

One pair: exact, at every dimension. Beyond one, the columns are $1/\sqrt{3}$, $1/\sqrt{5}$, $1/\sqrt{10}$.

The superposition law

Store k bound pairs in one vector by adding them. The retrieved item has $\cos = 1/\sqrt{k}$ with the correct answer, **independent of dimension**.

So dimension buys nothing here either. To see what it does buy, we have to ask what $1/\sqrt{k}$ is being compared against.

Why It Has to Be Hyperdimensional

Two unrelated random vectors are not orthogonal, but they are nearly so, and the margin improves with dimension:

d	mean $\lvert\cos\rvert$	$\approx 0.8/\sqrt{d}$	$P(\lvert\cos\rvert > 0.1)$
64	0.1068	0.0997	0.425
256	0.0524	0.0499	0.115
1024	0.0236	0.0249	0.000
4096	0.0132	0.0125	0.000
16384	0.0067	0.0062	0.000

At $d = 64$, two unrelated vectors land above $\cos = 0.1$ nearly half the time. At $d = 1024$ it never happened in four hundred trials. Random high-dimensional vectors behave like distinct symbols, and they do so without anyone assigning them.

Capacity Is Bought With Dimension

Retrieval works while the signal $1/\sqrt{k}$ stands clear of the noise floor $0.8/\sqrt{d}$:

d	noise floor	signal-to-noise at $k = 10$	items at $\mathrm{SNR} > 5$
64	0.1068	3.0	3
256	0.0524	6.0	14
1024	0.0236	13.4	71
4096	0.0132	23.9	228
16384	0.0067	46.9	880

Capacity grows linearly in d. This is the whole reason the field is called *hyperdimensional* computing: the representational power comes from dimension, not from any cleverness in the encoding.

Contrast the distributional tradition, where d is around 300 and chosen to make downstream classifiers work. VSA dimensions are $10{,}000$ and chosen to make symbols not collide.

The Silent Choice

Put the two halves of this deck together.

Distributional semantics	reduces by finding structure (SVD, SGNS, GloVe), then compares with the **inner** product
VSA	reduces by random projection, which finds nothing, then builds structure with a folded **outer** product

Each tradition made one choice and lived with its consequences. Distributional semantics gets similarity for free and cannot compose. VSA composes exactly and has no notion of graded similarity beyond what you put in.

Neither chose both, because in ordinary linear algebra the inner product and the outer product are different operations returning objects of different shapes. You have to pick one.

Four Positions

	operation	**keeps**	**costs**
Distributional	$\mathbf{a} \cdot \mathbf{b}$	similarity	all structure
Tensor products	$\mathbf{a}\mathbf{b}^\top$	structure, exactly	dimension, exponentially
VSA	$\mathbf{a} \circledast \mathbf{b}$	structure, fixed dimension	exactness
Geometric algebra	$\mathbf{a}\mathbf{b} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathbf{b}$	both, as separate grades	you must track grade

The geometric product does not choose. It returns a scalar part and a bivector part, in one object, and keeps them distinguishable. The similarity is in grade 0; the structure is in grade 2.

Whether that is the right move for semantics is the question the second half of this course exists to ask. What today established is that it is a *live* question, because the two alternatives each gave something up, and we can now say exactly what.

A Loop Worth Noticing

One empirical result closes the circle between the two halves.

Ethayarajh, 2019	Learned embeddings are strongly **anisotropic**: random word pairs have high average cosine, because the vectors share a dominant direction
Mu & Viswanath, 2018	Remove the top few principal components from trained embeddings and they get *better* on almost every benchmark

Read that second row through today's material. The postprocessing is an SVD. What it does is make the embedding space **more isotropic** – that is, more like a space of quasi-orthogonal random vectors.

The standard fix for a learned distributional space is to make it resemble the space VSA starts from. That is not an argument that VSA is right. It is a reason to take the comparison seriously.

What Comes Next

Settled today	
Why SGD	weighting the cells forfeits Eckart–Young, and with it the spectral solution
Count versus predict	a disagreement about which cells to trust, not about what a word is
A fourth reduction	random projection preserves rather than discovers
Binding	the outer product, folded to keep dimension fixed
Open	
Grade	can a representation carry similarity and structure at once, and should it

Reading: Plate 1995 on HRR; Sahlgren 2005 on random indexing; Gayler 2003 for the VSA framing; Smolensky 1990 for tensor products; Levy, Goldberg & Dagan 2015 for the count/predict comparison.

Exercises

1. The rank-2 SVD scored 0.0923 unweighted and 2.3264 weighted; the ALS factorization scored 0.8989 and 1.2294. Explain why no factorization could beat 0.0923 in the first column, and why several might beat 1.2294 in the second.
2. Circular convolution sums the outer product along wrapped anti-diagonals. Suppose instead you summed along wrapped *diagonals* ($i - j \bmod n$). What operation would you have, and would it still be usable for binding?
3. At $d = 1024$ the noise floor is 0.024 and a 71-item bundle sits at $\mathrm{SNR} = 5$. How large must d be to hold 1000 items at the same reliability? Confirm the scaling is linear.
4. Random projection reported $+0.0156$ on similar pairs where the SVD reported $+0.1280$. Argue which of the two you would prefer for a retrieval system, and which for a model of lexical semantics.