

Binding: From Outer Products to Convolution

Why HRR works, what it costs, and where the roles come from

James Pustejovsky

Brandeis University
Department of Computer Science

CS 135: Computational Semantics

Fall 2026

Where This Picks Up

Last session ended on a theorem and a division of labor. For any bilinear form $\mathbf{x}^\top G \mathbf{y}$, split G into its two halves:

$$G = \underbrace{\frac{1}{2} (G + G^\top)}_{S, \text{ symmetric}} + \underbrace{\frac{1}{2} (G - G^\top)}_{A, \text{ antisymmetric}}$$

S	similarity, relatedness, co-occurrence – everything cosine can see
A	who likes whom, what entails what, which argument came first

Today we take one operation that the field actually uses – **circular convolution**, the binding operation of Holographic Reduced Representations – and ask which half it keeps.

The answer is not a matter of degree. One of the two halves is multiplied by zero, exactly, and the consequences run all the way to how a sentence gets represented.

Roadmap

- ➊ **What binding has to do.** Hold two things together so that which-was-which survives. We show addition cannot, in one line.
- ➋ **The answer, and its price.** The outer product does the job perfectly and costs n^2 numbers per pair.
- ➌ **The fold.** Compress n^2 back to n by summing along wrapped diagonals. That fold is exactly circular convolution, and we prove the two definitions agree.
- ➍ **What the fold destroys.** It annihilates A – not approximately, not mostly. Exactly.
- ➎ **Living with it.** Why HRR must stipulate role vectors, how unbinding works, and which of its two losses more dimension can fix.

Part I

What Binding Has To Do

The Job

We want one vector that represents a *pair*: a role and its filler, an attribute and its value, a slot and what sits in it.

agent $\mapsto$ John	who did it
patient $\mapsto$ Mary	who it was done to
color $\mapsto$ red	an attribute–value pair

The requirement: whatever we build must distinguish *John as agent* from *John as patient*. If it does not, we have a bag of words with extra steps.

Everything in this deck follows from taking that requirement seriously and watching what each candidate operation does to it.

Addition Cannot Do It, and the Failure Is Exact

Suppose we represent *John loves Mary* by adding the word vectors.

$$\mathbf{t}_1 = \mathbf{v}_{\text{John}} + \mathbf{v}_{\text{loves}} + \mathbf{v}_{\text{Mary}} \quad \mathbf{t}_2 = \mathbf{v}_{\text{Mary}} + \mathbf{v}_{\text{loves}} + \mathbf{v}_{\text{John}}$$

$$\mathbf{t}_1 = \mathbf{t}_2, \text{ so } \cos(\mathbf{t}_1, \mathbf{t}_2) = 1.0000 \quad - \quad \text{not approximately, } \textit{identically}.$$

Verified at $d = 2048$ with random atoms: the two sums agree to the last bit, because addition is commutative and the two sentences use the same three words.

This is the same observation as Problem Set 2's order test – boathouse against houseboat – arriving from the other side. There the cosine of the two *predictions* was 1.000 while the observed vectors were nowhere near each other. A commutative operation cannot represent an ordered pair. That is an algebraic fact, not a tuning problem.

Part II

The Answer, and Its Price

The Outer Product Keeps Everything

To bind role $\mathbf{r}$ to filler $\mathbf{f}$, take the outer product:

$$\mathbf{r}\mathbf{f}^\top \in \mathbb{R}^{n \times n}, \quad \text{with} \quad (\mathbf{r}\mathbf{f}^\top)_{ij} = r_i f_j$$

And it is *not* $\mathbf{f}\mathbf{r}^\top$. In fact $\mathbf{f}\mathbf{r}^\top = (\mathbf{r}\mathbf{f}^\top)^\top$ – the two orders are transposes of one another, which is as clean a record of “which came first” as one could want.

$$\begin{aligned} \mathbf{u} &= (1, 2, 0, 1), & \mathbf{v} &= (2, 0, 1, 1) \\ \mathbf{u}\mathbf{v}^\top &= \begin{pmatrix} 2 & 0 & 1 & 1 \\ 4 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 1 \end{pmatrix} & \mathbf{v}\mathbf{u}^\top &= \begin{pmatrix} 2 & 4 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 1 \\ 1 & 2 & 0 & 1 \end{pmatrix} \end{aligned}$$

Different matrices, as required. And nothing has been lost: the matrix has rank 1, and dividing any nonzero row by the appropriate scalar returns $\mathbf{v}^\top$.

The Price

One pair costs n^2 numbers. The vectors going in cost n each. That ratio is the whole problem.

n	a vector	one bound pair	ratio
4	4	16	$4\times$
300	300	90,000	$300\times$
1000	1000	1,000,000	$1000\times$

Worse than the storage: **the result is the wrong kind of object**. A bound pair is a matrix, but its operands were vectors, so you cannot bind a bound pair to anything without going up another order. Nesting the agent of the event that John described would climb through n^2 , n^4 , n^8 .

So we need to come back down to $\mathbb{R}^n$. The question is not whether to compress – it is *what to throw away*, and that question has a precise answer.

Where This Came From

The outer-product account is Smolensky's **tensor product variable binding** (1990), and it was proposed as an answer to exactly the challenge we have been circling: can a distributed representation carry symbolic structure?

bind one pair	$\mathbf{r} \otimes \mathbf{f}$, an order-2 tensor
bind a pair of pairs	an order-4 tensor
a structure of depth d	an order- $2d$ tensor

It works, and it is exact – unbinding a TPR returns the filler with no noise at all. **What it does not do is stay the same size**, and the growth is not gentle: at $n = 300$, a depth-3 structure is $300^6 \approx 7 \times 10^{14}$ numbers.

HRR is the response. Plate's question was whether one could keep the structure and the fixed size at once. The rest of this deck is the answer, and the answer has a precise price.

Part III

The Fold

A Minute on Modulo Arithmetic

Everything that follows is indexed by $(i + j) \bmod n$, so it is worth a slide on what that means.

$a \bmod n$ is **the remainder when a is divided by n** . It is always one of $0, 1, \dots, n - 1$, however large a gets.

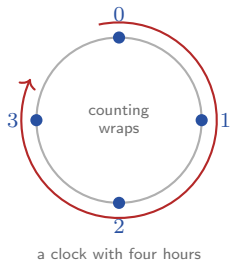
a	0	1	2	3	4	5	6	7	8	9
$a \bmod 4$	0	1	2	3	0	1	2	3	0	1

Count 0, 1, 2, 3 and then **start over**. 4 and 8 are back at 0; 9 is back at 1.

Negatives work the same way, counting backwards around the same circle – this is where people slip:

$$-1 \bmod 4 = 3, \quad -2 \bmod 4 = 2, \quad -4 \bmod 4 = 0$$

The familiar case is a twelve-hour clock: four hours after ten o'clock is two o'clock, because $14 \bmod 12 = 2$.



The One Table We Actually Need

We will be adding a row index i to a column index j , both in $\{0, 1, 2, 3\}$, and reducing mod 4. Here is every case:

$(i + j) \bmod 4$	j			
	0	1	2	3
$i = 0$	0	1	2	3
$i = 1$	1	2	3	0
$i = 2$	2	3	0	1
$i = 3$	3	0	1	2

Read a few off:

$i = 1, j = 2$: $1 + 2 = 3$, no wrap.

$i = 2, j = 3$: $2 + 3 = 5$, and $5 \bmod 4 = 1$.

$i = 3, j = 1$: $3 + 1 = 4$, and $4 \bmod 4 = 0$.

$i = 3, j = 3$: $3 + 3 = 6$, and $6 \bmod 4 = 2$.

Notice the shape. Each row is the previous one shifted left by one, with the value that fell off the end reappearing on the right. Every value 0–3 appears exactly once in each row and once in each column.

Keep this table in mind. Later it reappears as the little corner labels on a 4×4 grid of products, and the four cells carrying the same label are the four that get added together.

Compressing the Outer Product

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

We have n^2 numbers and need n . HRR's answer is a loop over two rows of numbers – **no matrix required**, and we will not draw one until the answer is computed.

The recipe

- 1 **Reverse** **v**. Once, at the start. Plain reversal: last entry first.
- 2 **Rotate** it one place right. Whatever falls off the right comes back on the left.
- 3 Line it up under **u**, multiply each column, add. That is the next output.
- 4 **Go back to step 2**, until you have n outputs.

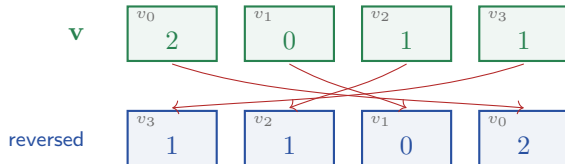
One reversal, then the same rotate-multiply-add step n times. **There is no special case** – even the first output comes after a rotation.

The rotation is where *circular* comes from. After n of them the row is back where it began, which is why there are n outputs and no more.

Step One: Reverse, Once

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Plain reversal – read $\mathbf{v}$ from the back. No index is special; the whole row turns around.



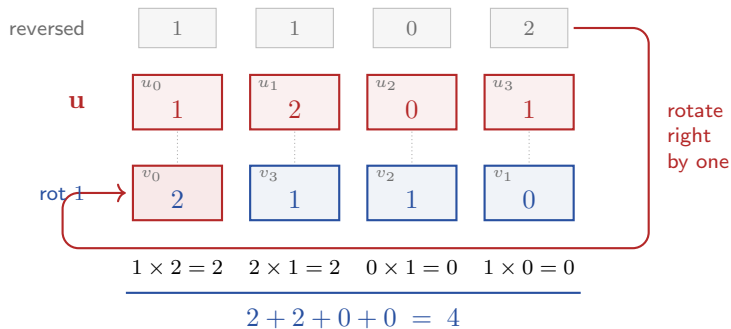
$\text{reverse}(\mathbf{v}) = (1, 1, 0, 2)$, carrying indices v_3, v_2, v_1, v_0 . Computed once; the loop never reverses again.

For later: reversing and then rotating once gives the *involution* $\mathbf{v}^*$ – the vector you unbind with in Part VI, and our very first row.

Output $k = 0$: After the First Rotation

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Rotate the reversed row one place right. The 2 at the far right comes round to the front.

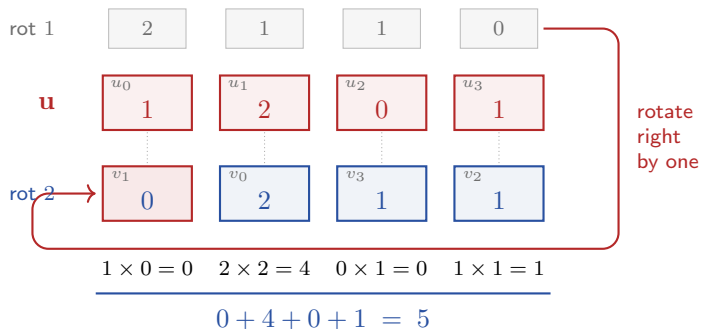


First output: 4. This row, $(2, 1, 1, 0)$, is the involution $\mathbf{v}^*$.

Output $k = 1$: Rotate Again

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Same step, applied to the row we just used. The 0 at the far right comes round to the front.

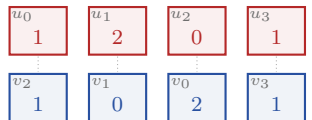


Second output: 5. Nothing was reversed again – only rotated.

Outputs $k = 2$ and $k = 3$: Twice More

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

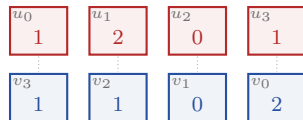
after rotation 3



$$1 \cdot 1 = 1 \quad 2 \cdot 0 = 0 \quad 0 \cdot 2 = 0 \quad 1 \cdot 1 = 1$$

$$\underline{1+0+0+1 = 2}$$

after rotation 4



$$1 \cdot 1 = 1 \quad 2 \cdot 1 = 2 \quad 0 \cdot 0 = 0 \quad 1 \cdot 2 = 2$$

$$\underline{1+2+0+2 = 5}$$

Look at the right-hand row: it is $\text{reverse}(\mathbf{v}) = (1, 1, 0, 2)$ again. Four rotations have returned the row to where it started, so a fifth would merely repeat output 0. That is why there are exactly n outputs.

The Result, and the Formula

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Four passes, one per output slot. k is the pass number: pass k is $\text{reverse}(\mathbf{v})$ rotated k further places, and it produces coordinate k . Its lower row as indices:

k	lower row	products	output
0	$v_0 \ v_3 \ v_2 \ v_1$	$2 + 2 + 0 + 0$	4
1	$v_1 \ v_0 \ v_3 \ v_2$	$0 + 4 + 0 + 1$	5
2	$v_2 \ v_1 \ v_0 \ v_3$	$1 + 0 + 0 + 1$	2
3	$v_3 \ v_2 \ v_1 \ v_0$	$1 + 2 + 0 + 2$	5

Each row is the one above it slid one place right, and the first is $\text{reverse}(\mathbf{v})$ rotated once. So under u_i sits v_{k-i} . Writing that as a sum:

$$(\mathbf{u} \circledast \mathbf{v})_k = \sum_i u_i v_{(k-i) \bmod n} \implies \mathbf{u} \circledast \mathbf{v} = (4, 5, 2, 5)$$

That is the textbook definition of circular convolution, derived from four passes you did by hand.

The Same Products, Seen in the Matrix

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Now bring back the outer product. Its cell (i, j) holds $u_i v_j$ – so all sixteen products we might ever need are sitting in it, and each pass picked four of them.

	v_0 = 2	v_1 = 0	v_2 = 1	v_3 = 1
$u_0 = 1$	2	0	1	1
$u_1 = 2$	4	0	2	2
$u_2 = 0$	0	0	0	0
$u_3 = 1$	2	0	1	1

the shaded four are **exactly**
the four products from the
 $k = 1$ pass:

$$u_0 v_1, u_1 v_0, u_2 v_3, u_3 v_2 \\ = 0 + 4 + 0 + 1 = 5$$

Nothing new has been computed – the same four numbers, found a different way. But look at *which* cells they are: $(0, 1), (1, 0), (2, 3), (3, 2)$. In every one, $i + j \equiv 1 \pmod{4}$ – even $(2, 3)$, whose indices add to 5. The next slide checks all four, one at a time.

Checking the Four Cells, One at a Time

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

A cell's address is (i, j) : row i belongs to **u**, column j belongs to **v**, and the cell holds $u_i v_j$. The four shaded cells, with their index sums:

cell (i, j)	entry	value	$i + j$	$(i + j) \bmod 4$	
$(0, 1)$	$u_0 v_1 = 1 \cdot 0$	0	1	1	
$(1, 0)$	$u_1 v_0 = 2 \cdot 2$	4	1	1	
$(2, 3)$	$u_2 v_3 = 0 \cdot 1$	0	5	1	$5 = 4 + 1$
$(3, 2)$	$u_3 v_2 = 1 \cdot 1$	1	5	1	$5 = 4 + 1$

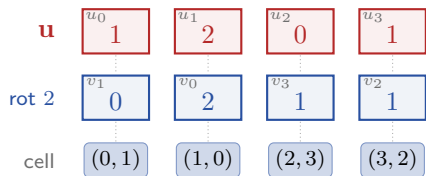
The first two add to 1 outright. The last two add to 5 – but indices only run from 0 to 3, so counting past 3 starts over at 0, like a clock with four hours. 5 is one full turn plus 1, so $5 \equiv 1 \pmod{4}$.

The shading marks positions, not values. Two shaded cells hold 0 only because row 2 is all zeros ($u_2 = 0$) and column 1 is all zeros ($v_1 = 0$). A cell is collected for *where* it is, not for what it holds.

Why These Four? Read the Strip Downward

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

The $k = 1$ pass again: each column pairs a u_i with a v_j , and that pair is a cell address.



Under u_i the lower row holds v_{k-i} . Call that subscript j :

$$j = k - i \iff i + j = k$$

So a pass collects exactly the cells whose indices add to k – it is just the strip, read downward.

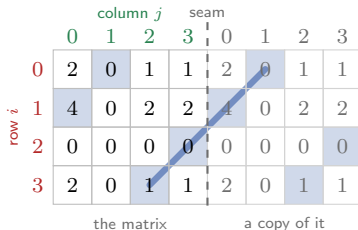
	on the strip	on the grid (mod 4)
$i = 2$	$v_{1-2} = v_{-1}$, which is v_3	$2 + 3 = 5 \equiv 1$
$i = 3$	$v_{1-3} = v_{-2}$, which is v_2	$3 + 2 = 5 \equiv 1$

One wrap, two views: the subscript goes below 0 on the strip, the sum goes past 3 on the grid.

Why the Line Breaks, and Why “Circular”

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

On the grid, pass 1's cells look like two pieces. Set a copy of the matrix beside it, and they line up:



Down and to the left, the line runs through (0,1) and (1,0) *in the copy*, crosses the seam, and continues through (2,3) and (3,2): **one straight anti-diagonal that runs off the edge.**

Glue column 3 to column 0 – roll the matrix into a cylinder – and there is no break at all. **That gluing is the circular in circular convolution:** on the strips, whatever fell off the right came back on the left.

All Four Passes at Once

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Do the same for every k : color each cell by its label $(i + j) \bmod 4$.

	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2
label $(i + j) \bmod 4$				

	0	1	2	3
0	2	0	1	1
1	4	0	2	2
2	0	0	0	0
3	2	0	1	1
entry $u_i v_j$				

Every cell carries exactly one label, so the four passes split the sixteen cells into four groups of four – nothing counted twice, nothing left out.

Add up each color: 4, 5, 2, 5 – the same four outputs the rotations gave.

Only $k = 3$ is unbroken: it is the main anti-diagonal. The other three are that stripe slid one, two or three columns right, and whatever slides past column 3 reappears at column 0.

Pass k collects exactly the cells labeled k . The next slide turns that into a definition.

Naming the Operation: the Fold

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Each pass picked the cells whose indices sum to k , modulo 4 – the label in each cell's corner. That grouping deserves a name.

⁰ 2	¹ 0	² 1	³ 1
¹ 4	² 0	³ 2	⁰ 2
² 0	³ 0	⁰ 0	¹ 0
³ 2	⁰ 0	¹ 1	² 1

Definition

For **any** $n \times n$ matrix M , the *fold* of M is the vector $\text{fold}(M) \in \mathbb{R}^n$ with

$$\text{fold}(M)_k = \sum_{i+j \equiv k \pmod{n}} M_{ij}$$

– add up every cell whose two indices sum to k .

Read “for any M ” carefully. The fold does not care where the matrix came from – and we are about to apply it to matrices that are not outer products at all.

On the grid: cells labeled 0 hold 2, 2, 0, 0, summing to 4; cells labeled 1 hold 0, 4, 0, 1, summing to 5. So $\text{fold}(\mathbf{uv}^\top) = (4, 5, 2, 5)$.

The Fold, Worked on Our Matrix

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Take $M = \mathbf{uv}^\top$ and apply the definition literally, writing entries as M_{ij} rather than $u_i v_j$ – the fold never looks at where they came from.

$$M = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{pmatrix} 2 & 0 & 1 & 1 \\ 4 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 1 \end{pmatrix} \end{matrix}$$

$$\text{fold}(M)_0 = M_{00} + M_{13} + M_{22} + M_{31} = 2+2+0+0 = 4$$

$$\text{fold}(M)_1 = M_{01} + M_{10} + M_{23} + M_{32} = 0+4+0+1 = 5$$

$$\text{fold}(M)_2 = M_{02} + M_{11} + M_{20} + M_{33} = 1+0+0+1 = 2$$

$$\text{fold}(M)_3 = M_{03} + M_{12} + M_{21} + M_{30} = 1+2+0+2 = 5$$

Check the subscripts in any line: the two digits sum to k , or to $k+4$. In the $k=1$ line: $0+1$, $1+0$, $2+3 = 5$, $3+2 = 5$.

$\text{fold}(M) = (4, 5, 2, 5)$ – the same four numbers the rotations gave.

Two Facts About the Fold

index	0	1	2	3
u	1	2	0	1
v	2	0	1	1

Both are needed in what follows, and both are one line.

1. The fold of an outer product is the convolution

$$\text{fold}(\mathbf{u}\mathbf{v}^\top) = \mathbf{u} \circledast \mathbf{v}$$

Cell (i, j) holds $u_i v_j$, and the cells with $i + j \equiv k$ are exactly the pairs the k th rotation lined up. Both give $(4, 5, 2, 5)$.

2. The fold is linear

$$\text{fold}(M + N) = \text{fold}(M) + \text{fold}(N), \quad \text{fold}(cM) = c \text{fold}(M)$$

Immediate, since each output is a plain sum of entries. This lets us split a matrix in two, fold the halves separately, and add the results.

Sixteen in, four out. Twelve degrees of freedom spent – and fact 2 is how we will say which twelve.

In-Class Exercise 1

Let $\mathbf{a} = (1, 0, 2, 0)$ and $\mathbf{b} = (0, 3, 0, 1)$ in $\mathbb{R}^4$.

- 1 Compute $\mathbf{a} \circledast \mathbf{b}$ with the two-row recipe: for each k , write $\mathbf{a}$ forwards, write $\mathbf{b}$ backwards from b_k with wrapping, multiply columns, add.
- 2 Now write down the outer product $\mathbf{a}\mathbf{b}^\top$ and check one of your four answers against it, by adding the cells whose corner label is that k .
- 3 Compute $\mathbf{b} \circledast \mathbf{a}$. Before you do: predict the answer.
 - ★ The vector $\mathbf{a}$ has two nonzero entries and so does $\mathbf{b}$, giving four nonzero products. Yet $\mathbf{a} \circledast \mathbf{b}$ has fewer than four nonzero entries. Why?

Five minutes. Item 4 is the one worth arguing about.

In-Class Exercise 1: Answers

1. Four nonzero entries, at $(0, 1)$, $(0, 3)$, $(2, 1)$, $(2, 3)$:

$$\mathbf{a}\mathbf{b}^\top = \begin{pmatrix} 0 & 3 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2. Groups: $k = 1$ gets $a_0b_1 = 3$; $k = 3$ gets $a_0b_3 = 1$ and $a_2b_1 = 6$, total 7; $k = 1$ also gets $a_2b_3 = 2$. Collecting,

$$\mathbf{a} \circledast \mathbf{b} = (0, 5, 0, 7)$$

3. $\mathbf{b} \circledast \mathbf{a} = (0, 5, 0, 7)$ – identical, although $\mathbf{a}\mathbf{b}^\top \neq \mathbf{b}\mathbf{a}^\top$.

4. ★ Two of the four products landed in the same group: a_0b_1 (group 1) and a_2b_3 (group 1) were added together, as were a_0b_3 and a_2b_1 (both group 3). Collisions are not a defect of this example – they are what compression is. Two independent facts became one number, and no later operation can separate them again.

Part IV

What the Fold Destroys

The Surprise

We adopted the outer product because it distinguishes $\mathbf{u}\mathbf{v}^\top$ from $\mathbf{v}\mathbf{u}^\top$. Then we folded it to get the size down. Fold both:

$$\mathbf{u} \circledast \mathbf{v} = (4, 5, 2, 5) \qquad \mathbf{v} \circledast \mathbf{u} = (4, 5, 2, 5)$$

Circular convolution is commutative. The operation we adopted in order to keep order does not keep order.

This is not a property of our particular $\mathbf{u}$ and $\mathbf{v}$. It holds for every pair of vectors, and you can see it from the formula: the group containing cell (i, j) depends on $i + j$, and $i + j = j + i$.

So the sixteen numbers held the distinction and the four do not. Something specific was destroyed, and we have the machinery to say exactly what.

Split the Matrix Before Folding

Every square matrix is its symmetric part plus its antisymmetric part. For $M = \mathbf{u}\mathbf{v}^\top$:

$$S = \frac{1}{2}(M + M^\top) = \begin{pmatrix} 2 & 2 & \frac{1}{2} & \frac{3}{2} \\ 2 & 0 & 1 & 1 \\ \frac{1}{2} & 1 & 0 & \frac{1}{2} \\ \frac{3}{2} & 1 & \frac{1}{2} & 1 \end{pmatrix} \quad A = \frac{1}{2}(M - M^\top) = \begin{pmatrix} 0 & -2 & \frac{1}{2} & -\frac{1}{2} \\ 2 & 0 & 1 & 1 \\ -\frac{1}{2} & -1 & 0 & -\frac{1}{2} \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \end{pmatrix}$$

S is unchanged by transposing; A changes sign. Between them they account for M exactly, and A is where “which came first” lives – swapping $\mathbf{u}$ and $\mathbf{v}$ flips its sign and leaves S alone.

Folding is linear, so $\text{fold}(M) = \text{fold}(S) + \text{fold}(A)$. We can fold the halves separately and see which one survives.

Fold Each Half

what is folded	result	
$M = \mathbf{u}\mathbf{v}^\top$	(4, 5, 2, 5)	the binding
symmetric half S	(4, 5, 2, 5)	all of it
antisymmetric half A	(0, 0, 0, 0)	none of it

The fold annihilates A exactly. Not mostly, not to within rounding. Every one of the four group sums of A is the integer zero.

And now the commutativity is explained rather than merely observed: $\mathbf{u} \circledast \mathbf{v}$ and $\mathbf{v} \circledast \mathbf{u}$ differ only in the sign of A , and A contributes nothing either way.

The next slide says why this had to happen – it is a fact about the grouping, not about these numbers.

Why: Transpose Fixes Every Group

Transposing sends cell (i, j) to cell (j, i) . Which group does the image land in?

$$(i, j) \text{ is in group } i + j \quad \implies \quad (j, i) \text{ is in group } j + i = i + j$$

The same group. So transposing shuffles the cells *within* each group and never moves a cell from one group to another. Checked for all four groups of our 4×4 .

The one-line proof

Let a_k be the group- k sum of A . Transposing permutes group k onto itself, so the sum is unchanged: $a_k(A^\top) = a_k(A)$. But $A^\top = -A$, so $a_k(A^\top) = -a_k(A)$. Hence $a_k = -a_k$, and $a_k = 0$.

A quantity equal to its own negative is zero. No property of $\mathbf{u}$ or $\mathbf{v}$ entered the argument, so this holds for every pair of vectors in every dimension.

What Circular Convolution Actually Is

Circular convolution is the symmetric half of the outer product, folded.

Written out: for any $\mathbf{u}, \mathbf{v}$,

$$\mathbf{u} \circledast \mathbf{v} = \text{fold}(\mathbf{u}\mathbf{v}^\top) = \text{fold}(S), \quad \text{fold}(A) = \mathbf{0}.$$

So the commutativity of $\circledast$ was never a quirk of the definition to be worked around. The operation is blind to order because the order-carrying half of its input is precisely what the fold removes. An operation built out of S alone could not have been anything but symmetric.

Worth pausing on how specific this is. We did not find that HRR loses “some” information, or that it is “lossy”. We found that it applies a projection whose kernel contains all of A , and A was the entire reason we reached for the outer product in the first place.

The Fast Version, and a Second Proof

Nobody computes the fold by looping over n^2 cells. The convolution theorem turns it into three transforms and one elementwise product:

$$\mathbf{u} \circledast \mathbf{v} = \mathcal{F}^{-1}(\mathcal{F}\mathbf{u} \odot \mathcal{F}\mathbf{v})$$

– cost $O(n \log n)$ rather than $O(n^2)$, which is what makes HRR practical at $n = 10,000$.

On our example, $\mathcal{F}\mathbf{u} = (4, 1-i, -2, 1+i)$ and $\mathcal{F}\mathbf{v} = (4, 1+i, 2, 1-i)$, whose elementwise product is $(16, 2, -4, 2)$. Transforming back gives $(4, 5, 2, 5)$, as before.

And there is the commutativity again, in one glance. The middle step is $\mathcal{F}\mathbf{u} \odot \mathcal{F}\mathbf{v}$, an *elementwise product of two spectra*. Elementwise multiplication of numbers is commutative, so $\circledast$ must be. The Fourier picture makes the symmetry obvious and hides where it came from; the matrix picture shows you the half that was deleted. Keep both.

Why Not Just Keep Both Halves?

An obvious thought: if the fold destroys A , transmit A separately. Count what that costs.

n	S holds	A holds	the fold keeps
4	10	6	4
300	45,150	44,850	300
1000	500,500	499,500	1000

A is just as quadratic as S . Carrying it alongside would put us back at n^2 – which is the outer product we were trying to escape. There is no cheap place to put the antisymmetric half, because it is half of a quadratic object.

This is the genuine impasse, and it is worth naming as one rather than papering over. Either you pay n^2 and keep order, or you pay n and supply order by convention. The geometric algebra unit later in the course is an attempt at a third answer: keep both halves in one object whose products stay inside the algebra.

Looking Ahead: A Third Answer

The geometric algebra unit picks up from this impasse, and Lecture 14 on Hopf algebras, *The Coproduct Behind the Product*, takes it furthest. A preview, with this session's $\mathbf{u} = (1, 2, 0, 1)$ and $\mathbf{v} = (2, 0, 1, 1)$:

$$\mathbf{uv} = \underbrace{3}_{\mathbf{u} \cdot \mathbf{v} = \text{tr } S} + \underbrace{\mathbf{u} \wedge \mathbf{v}}_{\text{a plane}}, \quad \mathbf{vu} = 3 - \mathbf{u} \wedge \mathbf{v}$$

- The fold kept four numbers and lost the order. This product keeps one number from S and all of A , as a plane: on the six coordinate planes $(0, 1), (0, 2), \dots, (2, 3)$, $\mathbf{u} \wedge \mathbf{v}$ has coordinates $-4, 1, -1, 2, 2, -1$, twice the entries of A above its diagonal. Swapping $\mathbf{u}$ and $\mathbf{v}$ flips only the plane.
- A still costs $\binom{n}{2}$ numbers, as the last table said it must. What changes is the shape: the result lies in an algebra whose products stay inside it, so it can be bound again.
- The lecture's tool is the **coproduct**, which lists the ways to cut an object in two. For $\mathbf{u} \wedge \mathbf{v}$ its middle term is $\mathbf{u} \otimes \mathbf{v} - \mathbf{v} \otimes \mathbf{u}$, the matrix $2A$ – exactly the half the fold destroyed. Pairing the cut pieces with a bilinear form builds the product of any two multivectors.

Part V

Why the Collisions Are Survivable

the question Exercise 1 raised

The Collision Problem

Exercise 1 ended on something that should have bothered you. Two of the four nonzero products landed in the same group and were added together, and **no later operation can separate them again**.

That was not bad luck with a small example. It is what the fold does, by construction:

n	cells	groups	cells summed per output
4	16	4	4
64	4,096	64	64
1024	1,048,576	1024	1024

So every output number is a sum of a thousand unrelated products. Each one is noise with respect to the others. Why does anything at all survive this?

The answer is the single fact the whole field rests on, and it is a fact about *dimension*, not about convolution.

Random Vectors Are Nearly Orthogonal

Draw two vectors at random in $\mathbb{R}^d$ and measure the angle between them. In low dimension the answer is anybody's guess; in high dimension it is almost always ninety degrees.

d	mean cos	sd of cos	$1/\sqrt{d}$
16	-0.0024	0.2506	0.2500
64	-0.0004	0.1247	0.1250
256	+0.0005	0.0624	0.0625
1024	-0.0001	0.0313	0.0312
4096	+0.0002	0.0157	0.0156
10,000	-0.0000	0.0100	0.0100

20,000 random pairs at each dimension.

The mean is zero at every dimension – random vectors have no systematic alignment. What changes is the spread, and it shrinks as $1/\sqrt{d}$, matching the last column to three decimals.

That is why the thousand-fold collision is survivable. The thousand things summed into each output are mutually near-orthogonal, so they add like a random walk rather than like a thousand copies of the same error.

It Is the Worst Case That Has to Be Small

An average of zero is not enough. Retrieval compares one correct answer against many distractors and takes the best match, so **a single unlucky pair with high cosine is enough to return the wrong symbol.**

d	sd of cos	largest cos seen in 20,000 pairs
16	0.2506	0.871
64	0.1247	0.469
256	0.0624	0.292
1024	0.0313	0.122
4096	0.0157	0.064
10,000	0.0100	0.039

At $d = 16$ some pair of “unrelated” atoms sits at 0.871 – as similar as genuine synonyms. **At $d = 10,000$ the worst offender in twenty thousand tries reaches 0.039**, and a correct match anywhere above 0.2 wins comfortably.

The maximum falls more slowly than the standard deviation, which is why practical systems use dimensions in the thousands rather than the hundreds.

Why “Hyperdimensional”

Now the name means something specific rather than sounding impressive.

High dimension is not a performance tweak. It is the major assumption. Binding collides n products into every output; superposition piles several bindings on top of one another; unbinding returns the answer plus a sum of cross terms. Every one of those is tolerable only because unrelated vectors are nearly orthogonal, and that is only true when d is large.

Run the same architecture at $d = 16$ and it does not degrade gracefully – it fails, because the distractors are as close as the answers. The usual working range in the VSA literature is d from about 1,000 to 10,000.

And note which of our two losses this helps. Raising d shrinks every number in these tables. It does nothing for the antisymmetric half, which the fold multiplied by zero in any dimension whatsoever. We now have both halves of that story, and can go back to using the operation.

Part VI

Living With It

Consequence: the Roles Must Be Stipulated

Represent *John loves Mary* the way HRR does, by binding roles to fillers and adding:

$$\mathbf{t} = \mathbf{r}_{\text{agent}} \circledast \mathbf{v}_{\text{John}} + \mathbf{r}_{\text{patient}} \circledast \mathbf{v}_{\text{Mary}}$$

This works. Unbinding with $\mathbf{r}_{\text{agent}}$ returns something close to $\mathbf{v}_{\text{John}}$, and the sentence is distinguishable from *Mary loves John*.

But notice what is doing the work. Since $\circledast$ is commutative,

$$\mathbf{r}_{\text{agent}} \circledast \mathbf{v}_{\text{John}} = \mathbf{v}_{\text{John}} \circledast \mathbf{r}_{\text{agent}}$$

– the trace does not record which operand was the role and which was the filler. **That distinction lives entirely in the designer's decision** that $\mathbf{r}_{\text{agent}}$ is a role vector and $\mathbf{v}_{\text{John}}$ is not. The algebra will not tell you, and nothing in $\mathbf{t}$ recovers it.

Why That Is a Real Cost

It is tempting to call this bookkeeping. Three reasons it is not.

- ❶ **The role inventory has to be fixed in advance.** You cannot discover a new role from data the way you can discover a new word – you must allocate a vector for it and declare it a role.
- ❷ **Roles and fillers cannot be the same objects.** In language they often are: in *the agent of the event*, agent is being referred to, not used as a slot. A representation that cannot let a role appear as a filler will not handle that.
- ❸ **The symbolic content re-enters from outside.** We set out to see how much symbolic structure vector operations could supply on their own. The answer here is: the structure is carried by a convention the designer imposed, not by the geometry.

Hold onto point 3. It is the question the whole second half of this course is about, and we will meet it again when the roles come from an algebra rather than a stipulation.

Unbinding: an Approximate Inverse

To get the filler back out, convolve with the **involution** of the role – the vector with its coordinates reversed, fixing the first:

$$\mathbf{r}^* = (r_0, r_{n-1}, r_{n-2}, \dots, r_1), \quad \mathbf{r}^* \circledast (\mathbf{r} \circledast \mathbf{f}) \approx \mathbf{f}$$

The approximation is doing real work. Expanding the left side gives $\mathbf{f}$ plus a sum of cross terms, and those terms do not vanish – they are small because random high-dimensional vectors are nearly orthogonal, which is the subject of the next session.

$d = 2048$, random atoms, trace = $\mathbf{r}_{\text{ag}} \circledast \mathbf{v}_{\text{John}} + \mathbf{r}_{\text{pt}} \circledast \mathbf{v}_{\text{Mary}}$	
$\cos(\mathbf{r}_{\text{ag}}^* \circledast \mathbf{t}, \mathbf{v}_{\text{John}})$	0.599
$\cos(\mathbf{r}_{\text{ag}}^* \circledast \mathbf{t}, \mathbf{v}_{\text{Mary}})$	0.052
$\cos(\mathbf{r}_{\text{ag}}^* \circledast \mathbf{t}, \mathbf{v}_{\text{Bill}})$	-0.009

0.599 is not 1. The retrieved vector is not John; it is a noisy thing nearest to John. Turning it into the symbol requires a lookup – *cleanup* – which is where the discreteness comes back.

Superposing More Than One Pair

A sentence needs several bindings at once, and we add them into one trace. Each extra pair contributes crosstalk to every retrieval.

pairs superposed	1	2	4	8	16	32
cos(retrieved, correct)	0.704	0.576	0.451	0.325	0.242	0.175

$d = 1024$, random atoms, mean of 20 trials.

The signal degrades smoothly rather than falling off a cliff, and it never reaches zero. Retrieval stays usable as long as the correct answer outscores every distractor – which is a question about the gap, not about the absolute number.

Crucially, this loss is of a different kind from the other one. Doubling d improves every number in that row. Doubling d does nothing whatever for A , which was multiplied by zero and is not coming back.

Two Different Losses

“Unbinding is approximate” has been doing double duty. Separate the two, because only one of them is fixable.

Loss	Character	Does more dimension help?
A annihilated	exact, total, structural	no – it was multiplied by zero
S compressed to n	approximate, noisy	yes – crosstalk falls as $1/\sqrt{n}$

At $n = 1000$ the fold first discards the 499,500 numbers of A , which encode nothing but argument order, and then compresses the remaining 500,500 into 1000.

The VSA literature’s capacity analysis – how many pairs can you superpose before retrieval degrades – is entirely about the second row. The first row is not a capacity question at all. It is why HRR has roles in its specification rather than in its arithmetic.

Part VII

Other Ways to Bind

The Options, and What Each Keeps

Operation	Output	Commutes	Keeps A	Cost
outer product $\mathbf{u}\mathbf{v}^\top$	matrix	no	yes	n^2
circular convolution $\mathbf{u} \circledast \mathbf{v}$	vector	yes	no	n
Hadamard $\mathbf{u} \odot \mathbf{v}$	vector	yes	no	n
permutation then multiply $\rho(\mathbf{u}) \odot \mathbf{v}$	vector	no	partly	n

Read the third and fourth columns together. Every operation that returns a vector and commutes has thrown A away – which is close to inevitable, since a commutative operation is by definition insensitive to order.

The last row is how practical systems patch it: apply a fixed permutation to one operand first, which breaks the symmetry by hand. Note that this is the same move as stipulating role vectors – the asymmetry is supplied by the designer, not by the algebra.

Row 1 is the only one that keeps everything, and the only one whose output is the wrong shape to bind again. That tension is the subject of the geometric algebra unit.

What Is Still Missing

We now have binding and unbinding, and we know exactly what they cost. Three things are still unexplained, and all three are the next session.

- ❶ **Why is the noise small?** Unbinding returned 0.599 against distractors near zero. That depends on random high-dimensional vectors being nearly orthogonal, which we have asserted and not shown.
- ❷ **How do we get from a noisy vector to a symbol?** The retrieved vector is not John. Something must map it to John, and that something – *cleanup memory* – is where discreteness re-enters the system.
- ❸ **How much fits?** Superposing more pairs degrades retrieval. There is a capacity law, and it is the second row of our loss table.

Those three give you a working architecture. With them we can build a record, a sequence, a lexical entry, and – without any training at all – an analogy.

Summary

- ① Binding must keep an ordered pair ordered. *Addition cannot: John loves Mary and Mary loves John come out at cosine exactly 1.0000.*
- ② The outer product does it perfectly, and costs n^2 numbers whose shape will not let you bind again.
- ③ Circular convolution is the **fold** of that matrix along wrapped anti-diagonals, $n^2 \rightarrow n$. *The textbook formula $\sum_i u_i v_{(k-i)}$ and the anti-diagonal sum are the same sum.*
- ④ *The fold annihilates A exactly*, because transposing permutes each anti-diagonal onto itself, so A 's group sums equal their own negatives.
- ⑤ Hence circular convolution *is* the symmetric half, folded, and its commutativity was never a quirk.
- ⑥ *HRR therefore stipulates role vectors.* The asymmetry comes from the designer, not the arithmetic – and two of its losses are different in kind, with only one of them answerable by more dimensions.

Reading and Exercises

Reading

- Plate, *Holographic Reduced Representations*, IEEE Trans. Neural Networks, 1995.
- Plate, *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*, CSLI, 2003 – chapters 3 and 4.
- Smolensky, *Tensor product variable binding and the representation of symbolic structures*, Artificial Intelligence, 1990 – the outer-product version, before the fold.

Exercises

- 1 Fold the outer product of $\mathbf{u} = (1, 1, 0, 0)$ and $\mathbf{v} = (0, 1, 1, 0)$ by hand, both orders, and confirm they agree.
- 2 Prove that $\text{fold}(M^\top) = \text{fold}(M)$ for every $n \times n$ matrix M , then deduce the annihilation result in two lines.
- 3 Show that $\mathbf{u} \circledast \mathbf{v}$ depends on $\mathbf{u}$ and $\mathbf{v}$ only through $S = \frac{1}{2}(\mathbf{u}\mathbf{v}^\top + \mathbf{v}\mathbf{u}^\top)$. How many independent numbers does S contain, and how many survive the fold?
- 4 Implement $\circledast$ by fast Fourier transform: $\mathbf{u} \circledast \mathbf{v} = \mathcal{F}^{-1}(\mathcal{F}\mathbf{u} \cdot \mathcal{F}\mathbf{v})$. Verify against your hand computation, and explain why the FFT form makes commutativity obvious in one glance.
- 5 ★ The permutation trick $\rho(\mathbf{u}) \odot \mathbf{v}$ breaks commutativity. Does it recover A ? Construct a small example and say precisely how much of the antisymmetric information survives.