

# Linear Algebra Refresher

The Operations Used in *Semantic Types and Their Vector Counterparts* —  
and the  $\lambda$ -Expression Each One Computes

James Pustejovsky

Brandeis University  
Department of Computer Science

CS 135: Computational Semantics

Fall 2026

## What This Is For

Today's main deck translates Montague's type system into vectors. It uses a small, fixed set of operations, over and over.

| Operation                        | Notation                       | Times used today |
|----------------------------------|--------------------------------|------------------|
| Inner product                    | $\mathbf{u} \cdot \mathbf{v}$  | 37               |
| Hadamard (componentwise) product | $\mathbf{u} \odot \mathbf{v}$  | 12               |
| Transpose                        | $M^T$                          | 9                |
| Contraction (Part IV preview)    | $A \lrcorner B$                | 5                |
| Wedge (Part IV preview)          | $\mathbf{u} \wedge \mathbf{v}$ | 3                |

That is the whole list. No eigenvalues, no determinants, no matrix inverses, no proofs.

Every slide here is a computation you can check by hand, on the same four-element domain we use all day.

And each is paired with the  $\lambda$ -expression it computes. You have been writing those since Session 1; the claim here is that you have also, without knowing it, been doing linear algebra.

# The Reframe That Makes Everything Easier

A vector is usually introduced as an arrow, or as a list of numbers. For our purposes, neither is the useful picture.

A vector is a **function from an index set to numbers**.

$$\mathbf{v} : \{1, 2, \dots, n\} \longrightarrow \mathbb{R}$$

Writing  $\mathbf{v} = (v_1, v_2, \dots, v_n)$  is just tabulating that function.

## Why this matters today

We are going to take the index set to be the **domain of individuals**. A vector then assigns a number to every individual — which is exactly what a predicate does, once you agree that “true” is 1.

## Two Notations, One Set of Objects

Here is the dictionary the rest of the deck fills in. Nothing on the right is new to you.

| Semantic type                             | $\lambda$ -expression                        | Linear-algebra object |
|-------------------------------------------|----------------------------------------------|-----------------------|
| $e$                                       | $a$                                          | one-hot vector $e_i$  |
| $t$                                       | $\text{sleep}(a)$                            | a number, 0 or 1      |
| $\langle e, t \rangle$                    | $\lambda x. \text{sleep}(x)$                 | indicator vector      |
| $\langle e, \langle e, t \rangle \rangle$ | $\lambda y. \lambda x. \text{like}(x, y)$    | matrix                |
| application                               | $(\lambda x. P(x))(a) \rightsquigarrow P(a)$ | contraction           |

Read the last row slowly.  $\beta$ -reduction — the one computational step of the lambda calculus — is going to turn out to be summation over a shared index.

That is not a metaphor or a loose analogy. It is an identity, and by the end of this deck you will have checked it twice by hand.

## Our Domain, Fixed for the Rest of the Day

$$\mathcal{D}_e = \{ \text{Alice, Bob, Carol, Fido} \}$$

The order matters and never changes. Position 1 is Alice, position 2 is Bob, position 3 is Carol, position 4 is Fido.

| Individual | Vector       | Name           |
|------------|--------------|----------------|
| Alice      | (1, 0, 0, 0) | $\mathbf{e}_1$ |
| Bob        | (0, 1, 0, 0) | $\mathbf{e}_2$ |
| Carol      | (0, 0, 1, 0) | $\mathbf{e}_3$ |
| Fido       | (0, 0, 0, 1) | $\mathbf{e}_4$ |

These are the **standard basis vectors**, also called **one-hot** vectors: a single 1, everything else 0.

An individual is not represented by any interesting content. It is represented by *which slot it occupies*. That is the entire encoding, and most of today's results follow from it.

## A Set Is a Vector of Zeros and Ones

Take any subset of the domain and write 1 in the position of each member.

| Set     | Members           | Indicator vector |
|---------|-------------------|------------------|
| student | Alice, Bob, Carol | (1, 1, 1, 0)     |
| dog     | Fido              | (0, 0, 0, 1)     |
| sleep   | Carol, Fido       | (0, 0, 1, 1)     |

This is the **indicator vector** of the set. One-hot vectors are a special case:  $e_3 = (0, 0, 1, 0)$  is the indicator of {Carol}.

### The $\lambda$ reading

$(1, 1, 1, 0)$  is the tabulation of  $\lambda x. \text{student}(x)$ : run  $x$  through the domain in order and write down each truth value. An indicator vector is a predicate of type  $\langle e, t \rangle$ , written out.

We write  $\mathbf{1} = (1, 1, 1, 1)$  for the all-ones vector — that is  $\lambda x. \top$ , the predicate true of everything.

## Practice: Read Both Directions

**Write the indicator vector.**

- 1 {Alice, Fido}
- 2 {Bob}
- 3 the set of everything except Carol

**Name the set.**

- 4  $(0, 1, 1, 0)$
- 5  $(1, 1, 1, 1)$

## Practice: Read Both Directions

**Write the indicator vector.**

- ① {Alice, Fido}
- ② {Bob}
- ③ the set of everything except Carol

**Name the set.**

- ④  $(0, 1, 1, 0)$
- ⑤  $(1, 1, 1, 1)$

**Answers.**

- ①  $(1, 0, 0, 1)$
- ②  $(0, 1, 0, 0)$ , which is  $\mathbf{e}_2$
- ③  $(1, 1, 0, 1)$
- ④ {Bob, Carol}
- ⑤ the whole domain,  $\mathbf{1}$

If this felt mechanical, good — it is. The content is in what the operations do to these vectors, not in the vectors themselves.

## Operation 1: Componentwise (Hadamard) Product

Multiply position by position. The result is a vector, the same length as the inputs.

$$(u_1, \dots, u_n) \odot (v_1, \dots, v_n) = (u_1 v_1, \dots, u_n v_n)$$

On indicator vectors,  $1 \times 1 = 1$  and everything else is 0, so:

$\mathbf{a} \odot \mathbf{b}$  is the indicator vector of  $A \cap B$ .

**Worked.** *sleeping student:*

$$(1, 1, 1, 0) \odot (0, 0, 1, 1) = (1 \cdot 0, 1 \cdot 0, 1 \cdot 1, 0 \cdot 1) = (0, 0, 1, 0)$$

Just Carol — which is right, since Carol is the only student who sleeps.

$\lambda x. [\text{student}(x) \wedge \text{sleep}(x)]$  — Predicate Modification, from Session 2.

The symbol  $\odot$  is sometimes written  $*$  or called the Schur product. In NumPy it is simply  $\mathbf{a} * \mathbf{b}$ , as opposed to  $\mathbf{a} @ \mathbf{b}$ .

## Complement, Union, and Difference

| $\lambda$ -expression               | Vector operation                                        | Example                                          |
|-------------------------------------|---------------------------------------------------------|--------------------------------------------------|
| $\lambda x.[A(x) \wedge B(x)]$      | $\mathbf{a} \odot \mathbf{b}$                           | $(1, 1, 1, 0) \odot (0, 0, 1, 1) = (0, 0, 1, 0)$ |
| $\lambda x.\neg A(x)$               | $\mathbf{1} - \mathbf{a}$                               | $\mathbf{1} - (0, 0, 1, 1) = (1, 1, 0, 0)$       |
| $\lambda x.[A(x) \vee B(x)]$        | $\mathbf{a} + \mathbf{b} - \mathbf{a} \odot \mathbf{b}$ | inclusion–exclusion, componentwise               |
| $\lambda x.[A(x) \wedge \neg B(x)]$ | $\mathbf{a} \odot (\mathbf{1} - \mathbf{b})$            | $(1, 1, 1, 0) \odot (1, 1, 0, 0) = (1, 1, 0, 0)$ |

**Note the third row.** Plain addition is *not* union:  $(1, 1, 1, 0) + (0, 0, 1, 1) = (1, 1, 2, 1)$ , and that 2 is not a truth value. You have to subtract the overlap to stay inside  $\{0, 1\}$ .

This is the first sign of something that recurs all term: the vector space happily produces objects that are not in the image of our encoding.

Staying inside  $\{0, 1\}^n$  takes care — and later on we stop trying. That sentence is worth unpacking.

# Why We Eventually Stop Trying

$\{0, 1\}^n$  is **not a subspace of  $\mathbb{R}^n$** . It is a finite set of corners, and it is not closed under the operations the space actually has.

| Operation                                           | Result                                         | Still a set? |
|-----------------------------------------------------|------------------------------------------------|--------------|
| $\mathbf{a} \odot \mathbf{b}$                       | $0 \cdot 0, 0 \cdot 1, 1 \cdot 1 \in \{0, 1\}$ | yes          |
| $1 - \mathbf{a}$                                    | flips each bit                                 | yes          |
| $\mathbf{a} + \mathbf{b}$                           | $(1, 1, \textcolor{red}{2}, 1)$                | no           |
| $2\mathbf{a}, \frac{1}{2}(\mathbf{a} + \mathbf{b})$ | scaled, averaged                               | no           |

So every composition rule has to be *chosen* so its output lands back on a corner. The inclusion–exclusion patch in the union row is not set theory — it is damage control.

## And then we give up on it

In Part II the entities are learned and dense, so there are no corners to return to: truth values become degrees in  $[0, 1]$ , and a predicate becomes a direction with a score for every point. Nobody checks closure, because the interior *is* the space. **That is not sloppiness.** The moment you want two entities at any angle other than  $90^\circ$ , you have left the corners — the cube and similarity cannot both be had.

## Operation 2: The Inner Product

Multiply position by position, then **add everything up**. The result is a single number.

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

---

|                                             |                                                                |
|---------------------------------------------|----------------------------------------------------------------|
| Hadamard $\mathbf{u} \odot \mathbf{v}$      | multiply, <b>keep</b> the positions $\Rightarrow$ a vector     |
| Inner product $\mathbf{u} \cdot \mathbf{v}$ | multiply, <b>sum over</b> the positions $\Rightarrow$ a number |

---

$$\mathbf{u} \cdot \mathbf{v} = \sum_i (\mathbf{u} \odot \mathbf{v})_i$$

The two operations differ in exactly one respect: whether the index is summed away. That distinction is the whole of what “contraction” will mean later, and it is worth fixing now.

Also written  $\langle \mathbf{u}, \mathbf{v} \rangle$  or, in matrix notation,  $\mathbf{u}^\top \mathbf{v}$ .

# The Inner Product Computes Three Different Things

Depending on what the two arguments are, one operation answers three questions.

| Form                            | $\lambda$ -expression                           | Question answered         |
|---------------------------------|-------------------------------------------------|---------------------------|
| $\mathbf{p} \cdot \mathbf{e}_i$ | $(\lambda x.P(x))(d_i) \rightsquigarrow P(d_i)$ | is $d_i$ in the set $P$ ? |
| $\mathbf{a} \cdot \mathbf{b}$   | $ \lambda x.[A(x) \wedge B(x)] $                | how big is $A \cap B$ ?   |
| $\mathbf{a} \cdot \mathbf{1}$   | $ \lambda x.A(x) $                              | how big is $A$ ?          |

The first row is **predication** — and note what it says:  $\beta$ -reducing  $(\lambda x.P(x))(a)$  and computing  $\mathbf{p} \cdot \mathbf{e}_a$  are the same act.

The second is **counting**, and it is what makes quantifiers work.

Why the second row is true:  $a_i b_i = 1$  exactly when  $i$  is in both sets, and 0 otherwise. Summing a vector of ones and zeros counts the ones. So  $\mathbf{a} \cdot \mathbf{b} = |A \cap B|$ , with no approximation.

## Worked: Three Inner Products

Recall  $\mathcal{V}(\text{student}) = (1, 1, 1, 0)$ ,  $\mathcal{V}(\text{sleep}) = (0, 0, 1, 1)$ ,  $\mathcal{V}(\text{dog}) = (0, 0, 0, 1)$ .

### 1. Does Carol sleep?

$$(0, 0, 1, 1) \cdot (0, 0, 1, 0) = 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 = 1 \quad \text{true}$$

### 2. How many students sleep?

$$(1, 1, 1, 0) \cdot (0, 0, 1, 1) = 0 + 0 + 1 + 0 = 1$$

### 3. How many students are there?

$$(1, 1, 1, 0) \cdot (1, 1, 1, 1) = 1 + 1 + 1 + 0 = 3$$

With items 2 and 3 in hand you can already evaluate *every student sleeps*: it is true exactly when  $|S \cap P| = |S|$ , that is, when  $1 = 3$ . **False**, correctly.

# Orthogonality

Two vectors are **orthogonal** when their inner product is zero.

$$\mathbf{u} \cdot \mathbf{v} = 0$$

For indicator vectors this has a plain reading: the two sets are disjoint.

$$\mathcal{V}(\text{student}) \cdot \mathcal{V}(\text{dog}) = (1, 1, 1, 0) \cdot (0, 0, 0, 1) = 0$$

No student is a dog. The geometry and the semantics agree.

**Now apply it to individuals.** For any two distinct  $i \neq j$ :

$$\mathbf{e}_i \cdot \mathbf{e}_j = 0$$

because the single 1 in each sits in a different position, so every product in the sum has a zero in it.

# The Consequence Everything Turns On

Every pair of distinct individuals is orthogonal. So compute the distance between any two:

$$\|\mathbf{e}_i - \mathbf{e}_j\| = \sqrt{1^2 + (-1)^2} = \sqrt{2}$$

| Pair                                | Distance   |
|-------------------------------------|------------|
| Alice, Bob                          | $\sqrt{2}$ |
| Alice, Fido                         | $\sqrt{2}$ |
| <i>any two distinct individuals</i> | $\sqrt{2}$ |

**Every individual is exactly as far from every other** — Alice is no closer to Bob than to Fido.

So this encoding has *no notion of similarity at all* — not a weak one, none. It is forced by the one-hot choice, and it is the most important fact in today's lecture.

## Where this returns

Nothing depends on the *number*  $\sqrt{2}$ ; it is  $\sqrt{1+1}$ . What matters are the two facts behind it —  $\mathbf{e}_i \cdot \mathbf{e}_i = 1$  and  $\mathbf{e}_i \cdot \mathbf{e}_j = 0$ . In Session 12 those reappear as the **defining relations** of  $\text{Cl}(n, 0)$ : what is a fact about distance here is the multiplication table there.

## Length, Distance, and Angle

Three quantities you will meet in the second half of the main deck, and repeatedly after today.

| Quantity | Formula                                                             | Reads as                           |
|----------|---------------------------------------------------------------------|------------------------------------|
| Norm     | $\ \mathbf{u}\  = \sqrt{\mathbf{u} \cdot \mathbf{u}}$               | how long                           |
| Distance | $\ \mathbf{u} - \mathbf{v}\ $                                       | how far apart                      |
| Cosine   | $\frac{\mathbf{u} \cdot \mathbf{v}}{\ \mathbf{u}\  \ \mathbf{v}\ }$ | the angle, with length divided out |

**Worked.**  $\|\mathcal{V}(\text{student})\| = \sqrt{(1, 1, 1, 0) \cdot (1, 1, 1, 0)} = \sqrt{3}$ .

Note a pleasant coincidence of the  $\{0, 1\}$  encoding:  $\mathbf{a} \cdot \mathbf{a} = |A|$ , so the squared norm of a set's indicator vector is its cardinality.

Cosine will do nothing useful today — with one-hot entities it is 0 for every distinct pair. It becomes the central measurement next session, once the vectors stop being one-hot.

## Matrices: Tables Indexed by Two Sets

If a vector assigns a number to each individual, a **matrix** assigns a number to each *ordered pair* of individuals.

Take like =  $\{(A, B), (B, C), (C, F), (F, F)\}$ , read as “row likes column.”

| $M_{\text{like}}$ | A | B | C | F |
|-------------------|---|---|---|---|
| Alice             | 0 | 1 | 0 | 0 |
| Bob               | 0 | 0 | 1 | 0 |
| Carol             | 0 | 0 | 0 | 1 |
| Fido              | 0 | 0 | 0 | 1 |

$M_{ij} = 1$  means individual  $i$  likes individual  $j$ .

A vector is indexed by one thing and a matrix by two — which is why a one-place predicate becomes a vector and a two-place relation becomes a matrix. The pattern continues: three places would need a three-index table.

The number of indices is called the **order** (or rank) of the tensor. A number is order 0, a vector order 1, a matrix order 2.

## Matrix Times Vector: Picking Out a Slice

$$(M\mathbf{v})_i = \sum_j M_{ij} v_j$$

Row  $i$  of  $M$ , inner-producted with  $\mathbf{v}$ ; do that for every row and collect the results.

**Worked.** Apply  $M_{\text{like}}$  to  $\mathbf{e}_{\text{Carol}} = (0, 0, 1, 0)$ :

$$M\mathbf{e}_{\text{Carol}} = \text{column C of } M = (0, 1, 0, 0)$$

The set of individuals who like Carol — just Bob.

### The shortcut worth memorizing

Multiplying a matrix by a one-hot vector **selects a column**. Multiplying a one-hot row vector by a matrix **selects a row**. Nothing else happens.

So  $M\mathbf{e}_j$  answers “who likes  $j$ ?”, and what it hands back is not a number — it is a whole predicate.

## What $M\mathbf{e}_j$ Actually Returns

A matrix times a one-hot vector is a  $\beta$ -reduction.

$$\llbracket \text{like} \rrbracket(c) = (\lambda y. \lambda x. \text{like}(x, y))(c) \rightsquigarrow \lambda x. \text{like}(x, c)$$

$$M \mathbf{e}_{\text{Carol}} = (0, 1, 0, 0)$$

Those are the same object.  $(0, 1, 0, 0)$  is  $\lambda x. \text{like}(x, c)$  tabulated over the domain — true of Bob, false of everyone else.

|                    | Type                                      | Order |
|--------------------|-------------------------------------------|-------|
| Before             | $\langle e, \langle e, t \rangle \rangle$ | 2     |
| After one argument | $\langle e, t \rangle$                    | 1     |

One  $\lambda$  discharged on the left, one index summed on the right — not analogous, the same bookkeeping written twice.

## Transpose: Swap the Two Indices

$$(M^{\top})_{ij} = M_{ji}$$

Reflect the table across its diagonal: rows become columns.

| $M$ | A | B | C | F | $M^{\top}$ | A | B | C | F |
|-----|---|---|---|---|------------|---|---|---|---|
| A   | 0 | 1 | 0 | 0 | A          | 0 | 0 | 0 | 0 |
| B   | 0 | 0 | 1 | 0 | B          | 1 | 0 | 0 | 0 |
| C   | 0 | 0 | 0 | 1 | C          | 0 | 1 | 0 | 0 |
| F   | 0 | 0 | 0 | 1 | F          | 0 | 0 | 1 | 1 |

$M$  encodes *likes*;  $M^{\top}$  encodes *is liked by*. They are different relations — Alice likes Bob, Bob does not like Alice — so  $M \neq M^{\top}$ .

### The $\lambda$ reading

$$M = \lambda y. \lambda x. \text{like}(x, y), \quad M^{\top} = \lambda x. \lambda y. \text{like}(x, y)$$

Same predicate, the two  $\lambda$ s swapped. Transposition *is* re-currying.

## The Bilinear Form: Filling Both Arguments

$$\mathbf{a}^\top M \mathbf{b} = \sum_i \sum_j a_i M_{ij} b_j$$

Both indices are summed away, so the result is a **number**.

**Worked.** *Alice likes Bob*, with  $\mathbf{a} = \mathbf{e}_1$  and  $\mathbf{b} = \mathbf{e}_2$ :

$$\mathbf{e}_1^\top M \mathbf{e}_2 = M_{12} = 1 \quad \text{true}$$

**And Bob likes Alice:**

$$\mathbf{e}_2^\top M \mathbf{e}_1 = M_{21} = 0 \quad \text{false}$$

With one-hot arguments the double sum collapses to reading a single cell: sandwiching a matrix between two basis vectors is table lookup, written in linear algebra.

# The Same Derivation, Twice

**In the  $\lambda$ -calculus:**

$$\begin{aligned}(\lambda y. \lambda x. \text{like}(x, y))(b)(a) &\rightsquigarrow (\lambda x. \text{like}(x, b))(a) \\ &\rightsquigarrow \text{like}(a, b) : t\end{aligned}$$

**In linear algebra:**

$$\mathbf{e}_1^\top M \mathbf{e}_2 = \mathbf{e}_1 \cdot \underbrace{(M \mathbf{e}_2)}_{(0,1,0,0)} = 1$$

**Two  $\beta$ -reductions on the left, two contractions on the right, in the same order.**

And the intermediate object is the same in both:  $\lambda x. \text{like}(x, b)$  and  $(0, 1, 0, 0)$  are two spellings of one predicate.

The main deck's "Worked: *Every student sleeps*" is this slide with a quantifier in place of the subject.

## Two Ways to Get There, and Why It Matters

The same number, computed in two orders:

|               | Step 1                                             | As a $\lambda$ -term                    |
|---------------|----------------------------------------------------|-----------------------------------------|
| Object first  | $M\mathbf{b} = \text{who likes } b$                | $\lambda x. \text{like}(x, b)$          |
| Subject first | $\mathbf{a}^\top M = \text{whom } a \text{ likes}$ | $\lambda y. \text{like}(\mathbf{a}, y)$ |

**Worked, object first.**  $M\mathbf{e}_2 = (1, 0, 0, 0)$ , then  $(1, 0, 0, 0) \cdot (1, 0, 0, 0) = 1$ .

**Worked, subject first.**  $\mathbf{e}_1^\top M = (0, 1, 0, 0)$ , then  $(0, 1, 0, 0) \cdot (0, 1, 0, 0) = 1$ .

Both give 1, and the intermediate objects are different: one is a set of likers, the other a set of likees. Each is a partially applied predicate.

In the main deck this becomes the claim that *currying determines contraction order*: the type  $\langle e, \langle e, t \rangle \rangle$  says which argument is consumed first, and that choice picks which of these two predicates you build on the way.

Montague's order is the first row — object first — which is why  $M\mathbf{e}_j$ , and not  $\mathbf{e}_i^\top M$ , is the one that matches  $\llbracket \text{like} \rrbracket(j)$ .

## In-Class Exercise: Get Your Hands Dirty

Using  $\mathcal{D}_e = \{\text{Alice, Bob, Carol, Fido}\}$  and  $\text{like} = \{(A, B), (B, C), (C, F), (F, F)\}$ :

| $M$ | A | B | C | F |
|-----|---|---|---|---|
| A   | 0 | 1 | 0 | 0 |
| B   | 0 | 0 | 1 | 0 |
| C   | 0 | 0 | 0 | 1 |
| F   | 0 | 0 | 0 | 1 |

$$\mathcal{V}(\text{sleep}) = (0, 0, 1, 1)$$

- 1 Is *Bob likes Carol* true? Write it as a bilinear form, then evaluate.
  - 2 Compute  $M \mathbf{e}_{\text{Fido}}$ . Name the set in English, **and** write the  $\lambda$ -term it is.
  - 3 Compute  $M^\top \mathbf{e}_{\text{Carol}}$ . Why is this a different set from the answer to 2, and which  $\lambda$ -term is it?
- ★ Compute  $M^\top \mathcal{V}(\text{sleep})$ . **Something goes wrong.** What is the result, why is it not a set, and what would you have to do to turn it into one?

Items 1–3 are drill. Item 4 is the one worth arguing about — try it before looking.

## In-Class Exercise: Answers

1.  $\mathbf{e}_2^\top M \mathbf{e}_3 = M_{23} = 1$ , true. Bob does like Carol.
2.  $M \mathbf{e}_F = \text{column } F = (0, 0, 1, 1) = \{\text{Carol, Fido}\}$  — those who like Fido, i.e.  $\lambda x. \text{like}(x, f)$ .
3.  $M^\top \mathbf{e}_C = \text{row } C \text{ of } M = (0, 0, 0, 1) = \{\text{Fido}\}$  — whom Carol likes, i.e.  $\lambda y. \text{like}(c, y)$ . Different because the transpose fixes the *other* argument: 2 fixes the object, 3 fixes the subject.
4.  $M^\top(0, 0, 1, 1) = (0, 0, 0, 2)$ .

There is a 2 in it, so it is not an indicator vector and therefore not a set. Fido is liked by Carol *and* by Fido, and both of them sleep — so the entry counts, it does not assert.

Applying a relation to a **set** rather than to an individual gives  $(\lambda x. |\{y : \text{sleep}(y) \wedge \text{like}(y, x)\}|)$  — a function from individuals to *counts*, not to truth values. To recover a predicate, threshold:  $v \geq 1$  gives *is liked by some sleeper*;  $v = |\text{sleep}|$  gives *is liked by every sleeper*. Those two thresholds are the *some* and *every* tests you will meet in the main deck — quantification is what you do to a count vector.

# Summation Indices, and What Contraction Means

Every operation today has the same shape. Write them side by side with the index bookkeeping visible.

| Operation                      | Formula                                     | $\lambda$ -counterpart  |
|--------------------------------|---------------------------------------------|-------------------------|
| $\mathbf{u} \odot \mathbf{v}$  | $(\mathbf{u} \odot \mathbf{v})_i = u_i v_i$ | no $\lambda$ discharged |
| $\mathbf{u} \cdot \mathbf{v}$  | $\sum_i u_i v_i$                            | one $\beta$ -reduction  |
| $M\mathbf{v}$                  | $\sum_j M_{ij} v_j$                         | one $\beta$ -reduction  |
| $\mathbf{a}^\top M \mathbf{b}$ | $\sum_i \sum_j a_i M_{ij} b_j$              | two $\beta$ -reductions |

## The rule

**Contraction** is summing over a shared index; each lowers the total order by two.  $\beta$ -reduction is contraction. Predicate modification is the one row discharging no  $\lambda$  — and the one row that sums nothing.

Look hard at rows 2 and 3 before the next slide: they carry the same entry in the right-hand column.

## Two Operations That Are One Operation

$\mathbf{p} \cdot \mathbf{e}_a$  and  $M\mathbf{e}_j$  have different names — “inner product” and “matrix–vector multiplication” — and they are **one  $\beta$ -reduction each**.

$$\underbrace{1 + 1}_{(\lambda x.P(x))(a)} \rightarrow 0 \rightsquigarrow P(a)$$

$$\underbrace{2 + 1}_{(\lambda y.\lambda x.R(x,y))(b)} \rightarrow 1 \rightsquigarrow \lambda x.R(x,b)$$

The only difference is the **order of the function** being applied. A unary predicate contracted with an entity returns a truth value; a binary one returns a predicate.

Inner product and matrix–vector multiplication are not two things. They are **contraction**, applied to tensors of different order — and each is one step of  $\beta$ -reduction.

Same for the bilinear form:  $\mathbf{a}^\top M \mathbf{b}$  is  $2 + 1 + 1 \rightarrow 0$ , two contractions and two  $\beta$ -steps. The pattern does not stop at matrices.

## A $\lambda$ -Term We Have Not Tried

Every  $\lambda$ -term so far had each bound variable occurring **once**. Now take one that does not.

*Alice likes herself.      Every dog chases itself.*

The predicate in both is the same shape:

$$\lambda x. \text{like}(x, x) : \langle e, t \rangle$$

Note the type. This is an ordinary one-place predicate — a *set* — so it must have an indicator vector like any other. Nothing exotic is being asked for.

So let us just build it, the way we built every other set today.

## Building It By Hand

Which individuals like themselves? Ask the question once per individual, and read the answer off  $M$ .

| Question               | Cell to read | Value |     |
|------------------------|--------------|-------|-----|
| Does Alice like Alice? | $M_{AA}$     | 0     | no  |
| Does Bob like Bob?     | $M_{BB}$     | 0     | no  |
| Does Carol like Carol? | $M_{CC}$     | 0     | no  |
| Does Fido like Fido?   | $M_{FF}$     | 1     | yes |

$$\mathcal{V}(\lambda x. \text{like}(x, x)) = (0, 0, 0, 1)$$

Only Fido likes himself. Correct, and no new machinery was needed.

The cells we read were  $M_{11}, M_{22}, M_{33}, M_{44}$  — the ones where the row index equals the column index. That list has a name: the **diagonal** of  $M$ .

So “diagonal” is not a new tool introduced to solve a problem. It is the name for what we just did by hand.

## But Notice What Kind of Step That Was

Compare the diagonal with the operation that sums it — the **trace**, written  $\text{tr}(M)$ : add up the entries down the diagonal,  $\text{tr}(M) = \sum_i M_{ii} = 0 + 0 + 0 + 1 = 1$ .

|                  | Formula                         | Answers                     | Result                  |
|------------------|---------------------------------|-----------------------------|-------------------------|
| $\text{diag}(M)$ | $M_{ii}$ , <b>kept</b>          | <i>who</i> likes themselves | $(0, 0, 0, 1)$ a vector |
| $\text{tr}(M)$   | $\sum_i M_{ii}$ , <b>summed</b> | <i>how many</i> do          | 1 a number              |

You have seen that contrast today. It is Hadamard versus inner product: keep the positions, or sum them away. The same distinction, one level up.

### The part that is genuinely new

Every operation so far took **two** arguments and *eliminated* a shared index. Both of these take **one** argument and *force two of its indices to be equal*. That is not contraction — contraction consumes an index from each of two arguments, and here there is only one argument.

And it is not something you can build out of  $M\mathbf{v}$  and  $\cdot$  either: no product of  $M$  with a vector can make the two slots be the same slot.

# Why Those Slides Matter More Than They Look

|                      | $\lambda$ -calculus | Linear algebra           |
|----------------------|---------------------|--------------------------|
| Apply an argument    | $\beta$ -reduce     | contract                 |
| Use a variable twice | write $x$ twice     | no contraction does this |

In the  $\lambda$ -calculus, reusing a variable is free — you write the letter again. Variables have **names**, and a name can appear anywhere in the body. *That is the whole reason  $\lambda x.\text{like}(x, x)$  was easy to write and awkward to compute.*

In linear algebra there are no names, only positions, and contraction always consumes one index from each of two arguments. Copying an index is a different operation, and it must be added.

## Carry this forward

This is the first appearance of a problem that recurs all term: **argument-filling is not variable binding**. Reflexives, relative pronouns, donkey anaphora and scope all turn on it, and the extra structure needed (Frobenius or bialgebraic operations) is what Parts III and IV have to supply.

## Two Ways to Change the Basis

Our coordinates came from an arbitrary decision: we listed the domain as Alice, Bob, Carol, Fido. Two things could be meant by “changing” that, and they are not the same.

|                |                                                                                                                                                        |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Relabel</b> | list the domain in another order — Bob, Alice, Carol, Fido. A <i>permutation</i> of the axes. Semantically empty: the order was ours, not the model's. |
| <b>Rotate</b>  | <i>mix</i> the axes, so coordinate 1 becomes something like $0.7 \text{ Alice} + 0.7 \text{ Bob}$ .                                                    |

We cannot actually rotate — and that is the point

There is no individual “ $0.7 \text{ Alice} + 0.7 \text{ Bob}$ .” A rotation **destroys** today's encoding, because the axes have to *be* the individuals.

So rotation is a **diagnostic**, not a proposal: it asks whether an operation depends on the geometry — lengths and angles — or on the accident of which grid we drew.

## What Survives Each

| Operation                                     | Relabel? | Rotate? |                                  |
|-----------------------------------------------|----------|---------|----------------------------------|
| $\mathbf{u} \cdot \mathbf{v}$                 | yes      | yes     | preserved by any orthogonal map  |
| $\ \mathbf{u}\ $ , distance, cosine           | yes      | yes     | all built from the inner product |
| $M\mathbf{v}$ , $\mathbf{a}^\top M\mathbf{b}$ | yes      | yes     | with $M$ transformed to match    |
| $\mathbf{u} \odot \mathbf{v}$                 | yes      | no      | survives relabeling only         |

Read the third column first: Hadamard is *fine* under relabeling. Reordering the domain reorders both vectors the same way, so  $(P\mathbf{u}) \odot (P\mathbf{v}) = P(\mathbf{u} \odot \mathbf{v})$ . Our encoding does not secretly depend on alphabetical order.

But it is the only row that fails under mixing — and permutations are exactly the basis changes that keep “one axis = one individual.” That is why  $\odot$  is legitimate today and not in general.

Here is the counterexample, in two dimensions, so you can see what goes wrong.

## Hadamard Depends on the Coordinates

Take  $\mathbf{u} = (1, 0)$  and  $\mathbf{v} = (0, 1)$ , and rotate by  $45^\circ$ .

**Before rotating:**

$$\mathbf{u} \odot \mathbf{v} = (1 \cdot 0, 0 \cdot 1) = (0, 0)$$

**Rotate first:**  $R\mathbf{u} = \frac{1}{\sqrt{2}}(1, 1), \quad R\mathbf{v} = \frac{1}{\sqrt{2}}(-1, 1).$

$$(R\mathbf{u}) \odot (R\mathbf{v}) = \frac{1}{2}(-1, 1) \neq (0, 0) = R(\mathbf{u} \odot \mathbf{v})$$

Rotating and then multiplying gives a different answer from multiplying and then rotating.

By contrast the inner product is untouched:  $(R\mathbf{u}) \cdot (R\mathbf{v}) = \frac{1}{2}(-1 + 1) = 0 = \mathbf{u} \cdot \mathbf{v}.$

So  $\odot$  is not really an operation on *vectors*. It is an operation on *lists of coordinates*, and it means something today only because we fixed a basis in which each axis is an individual.

## Why This Matters Later

Today the grid is meaningful because **we put it there**. In Part II the axes are **learned**: dimension 137 of an embedding is not a context and not an entity — it is whatever the optimizer produced.

---

|                                        |                                |
|----------------------------------------|--------------------------------|
| Two training runs, different seeds     | different axes, same structure |
| SVD with $\alpha = 0$ vs. $\alpha = 1$ | different axes                 |
| word2vec vs. GloVe                     | different axes                 |

---

### The test, restated

**If your answer changes when the axes change, you measured your coordinate system, not the meaning.**

Cosine survives, so cosine similarity between learned embeddings is a claim about meaning.  $\odot$  does not — and  $\odot$  is what the multiplicative models of phrase composition reach for.

Part IV's wedge and contraction are defined *without reference to coordinates* — not the only argument for geometric algebra, but the one this slide sets up.

## Two Operations You Will Only See Named Today

The main deck previews two operations from geometric algebra. You are not expected to compute with them yet.

| Symbol                         | Name             | One-line idea                                                                                          |
|--------------------------------|------------------|--------------------------------------------------------------------------------------------------------|
| $\mathbf{u} \wedge \mathbf{v}$ | wedge            | builds structure: combines two objects into a higher-order one, and $\mathbf{u} \wedge \mathbf{u} = 0$ |
| $A \lrcorner B$                | left contraction | consumes structure: the basis-free version of the summing we did today                                 |

The reason they appear at all is the previous slide. Both are defined *without* reference to a basis, so they do not have Hadamard's problem — and Part III builds the algebra in which they live.

When you see  $\lrcorner$  today, read it as “contract” and keep going. Sessions 12 through 15 do it properly.

# Everything, on One Slide

| Linear algebra                  | $\lambda$ -calculus               | Type                                      | Semantic role                        |
|---------------------------------|-----------------------------------|-------------------------------------------|--------------------------------------|
| $\mathbf{e}_i$                  | $\mathbf{a}$                      | $e$                                       | an individual                        |
| $\mathbf{a} \in \{0, 1\}^n$     | $\lambda x. A(x)$                 | $\langle e, t \rangle$                    | a predicate                          |
| $\mathbf{1} - \mathbf{a}$       | $\lambda x. \neg A(x)$            | $\langle e, t \rangle$                    | negation                             |
| $\mathbf{a} \odot \mathbf{b}$   | $\lambda x. [A(x) \wedge B(x)]$   | $\langle e, t \rangle$                    | predicate modification               |
| $\mathbf{p} \cdot \mathbf{e}_i$ | $(\lambda x. P(x))(d_i)$          | $t$                                       | <b>function application</b>          |
| $\mathbf{a} \cdot \mathbf{b}$   | $ \lambda x. [A(x) \wedge B(x)] $ | num                                       | quantifier arithmetic                |
| $\mathbf{a} \cdot \mathbf{1}$   | $ \lambda x. A(x) $               | num                                       | cardinality                          |
| $M$                             | $\lambda y. \lambda x. R(x, y)$   | $\langle e, \langle e, t \rangle \rangle$ | a binary relation                    |
| $M \mathbf{e}_j$                | $\lambda x. R(x, d_j)$            | $\langle e, t \rangle$                    | partial application                  |
| $M^\top$                        | $\lambda x. \lambda y. R(x, y)$   | $\langle e, \langle e, t \rangle \rangle$ | the other currying                   |
| $\mathbf{a}^\top M \mathbf{b}$  | $R(d_i, d_j)$                     | $t$                                       | transitive sentence                  |
| $\text{diag}(M)$                | $\lambda x. R(x, x)$              | $\langle e, t \rangle$                    | reflexive — <i>not</i> a contraction |

The two left columns are the same objects in two notations. The last row is the one that does not fit, and today's lecture is about how far the correspondence goes before rows like that start appearing.

## Check Yourself Before We Start

Using  $\mathcal{D}_e = \{A, B, C, F\}$ ,  $\mathcal{V}(\text{student}) = (1, 1, 1, 0)$ ,  $\mathcal{V}(\text{sleep}) = (0, 0, 1, 1)$ , and  $M_{\text{like}}$  from earlier:

- 1 Compute  $\mathcal{V}(\text{student}) \odot (\mathbf{1} - \mathcal{V}(\text{sleep}))$  and name the set.
- 2 Compute  $|\text{sleep}|$  using only an inner product.
- 3 Is *Carol likes Fido* true? Write it as a bilinear form and evaluate.
- 4 What is  $M^\top \mathbf{e}_F$ , and what does it denote in English?
- 5 Give two vectors in  $\mathbb{R}^4$  that are orthogonal but whose Hadamard product you cannot predict without knowing their coordinates.

## Check Yourself Before We Start

Using  $\mathcal{D}_e = \{A, B, C, F\}$ ,  $\mathcal{V}(\text{student}) = (1, 1, 1, 0)$ ,  $\mathcal{V}(\text{sleep}) = (0, 0, 1, 1)$ , and  $M_{\text{like}}$  from earlier:

- 1 Compute  $\mathcal{V}(\text{student}) \odot (1 - \mathcal{V}(\text{sleep}))$  and name the set.
- 2 Compute  $|\text{sleep}|$  using only an inner product.
- 3 Is *Carol likes Fido* true? Write it as a bilinear form and evaluate.
- 4 What is  $M^\top \mathbf{e}_F$ , and what does it denote in English?
- 5 Give two vectors in  $\mathbb{R}^4$  that are orthogonal but whose Hadamard product you cannot predict without knowing their coordinates.

**Answers.** (1)  $(1, 1, 1, 0) \odot (1, 1, 0, 0) = (1, 1, 0, 0)$ , the non-sleeping students, Alice and Bob. (2)  $(0, 0, 1, 1) \cdot (1, 1, 1, 1) = 2$ . (3)  $\mathbf{e}_3^\top M \mathbf{e}_4 = M_{34} = 1$ , true. (4)  $M^\top \mathbf{e}_F$  is column F of  $M^\top$ , i.e. row F of  $M$ :  $(0, 0, 0, 1)$  — whom Fido likes, namely Fido. Contrast  $M \mathbf{e}_F = (0, 0, 1, 1)$ , those who like Fido. (5) Any pair with disjoint support, e.g.  $(1, 1, 0, 0)$  and  $(0, 0, 1, 1)$ : orthogonal, product  $(0, 0, 0, 0)$  — but rotate the basis and the product changes while the orthogonality does not.

# Ready?

Now: *Semantic Types and Their Vector Counterparts*

Keep the two-column habit. For every operation today, ask what  $\lambda$ -term it computes — and when the answer is “none,” that is the interesting case.