

INTRODUCTION TO LINEAR ALGEBRA FOR COMPUTER SCIENCE

Vectors, Projections, and Dimensionality Reduction

Lecture 1 Notes · Mert Inan · University of Denver

The book of nature is written in the language of mathematics.

Galileo Galilei, *Il Saggiatore* (1623)

What we will learn today. This lecture builds three ideas, each one depending on the last.

- §1. **Vectors in $\mathbb{R}^n$.** We will learn how to represent real-world objects (coffee orders, students, words) as lists of numbers that live in a geometric space. We will define the dot product and use it to measure similarity.
- §2. **Projection and dimensionality reduction.** We will learn how to compress a vector onto a single direction using the dot product, and ask: which direction preserves the most information? The answer will involve eigenvectors of the covariance matrix.
- §3. **Connections to computer science.** We will see the same mathematics at work in neural networks and large language models, and understand why linear algebra is the mathematical language of modern CS.

§1 VECTORS IN $\mathbb{R}^n$

1.1 Data as vectors

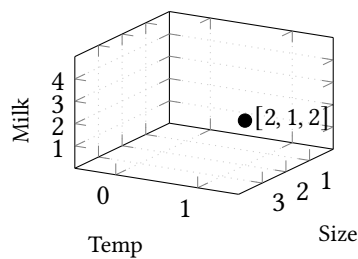
Let us start with a simple scenario. Suppose you are ordering coffee and you need to communicate your order to a robot that only understands numbers. You might agree on the following coding scheme: size (S = 1, M = 2, L = 3), temperature (iced = 0, hot = 1), milk (none = 1, oat = 2, whole = 3, almond = 4).

[**Pause.** What would your coffee order look like as a list of three numbers?]

A medium hot oat latte becomes the list $[2, 1, 2]$. In mathematics, we have a name for this.

DEFINITION 1.1. A **vector** in $\mathbb{R}^n$ is an ordered list of n real numbers. We write $\mathbf{x} = [x_1, x_2, \dots, x_n]^\top$.

So our coffee order is $\mathbf{x} = [2, 1, 2]^\top \in \mathbb{R}^3$. We can think of it as a point in a three-dimensional space, where the axes are size, temperature, and milk:



Every possible coffee order is a point somewhere in this space. Two orders that differ in only one coordinate (say, iced vs. hot) sit close together. Orders that differ across all three are far apart. Already we have a useful principle: *similarity between things becomes distance between points*.

But coffee is simple. It only takes three numbers. What about something more complex?

[**Pause.** If you had to describe a student at this university using only numbers, what would you measure? Take a moment and come up with a few properties.]

You might list: GPA, age, credit hours completed, commute distance, hours of sleep, number of current courses, cups of coffee per day. That is seven properties, so each student is a vector in $\mathbb{R}^7$. We cannot visualize seven dimensions, but the mathematics works exactly the same way as the coffee example: each student is a point, and students with similar numbers are nearby points.

The idea of describing the world through coordinates is not unique to computer science. Einstein's special relativity (1905) proposed that an event in the universe requires four coordinates: $(x, y, z, t) \in \mathbb{R}^4$. Adding time as a dimension was radical at the time. Computer science takes this much further: we build spaces with hundreds, thousands, or millions of dimensions, one for every property we choose to measure.

1.2 The dot product and cosine similarity

Once data lives in $\mathbb{R}^n$, we need a tool to measure how much two vectors “agree.” This is the dot product.

DEFINITION 1.2. The **dot product** of $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ is

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^n x_i y_i = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n.$$

The **norm** (length) of a vector is $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$. The **cosine similarity** is

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}.$$

Let us compute one together.

EXAMPLE 1.3. Let $\mathbf{a} = [2, 1, 2]^\top$ (medium hot oat) and $\mathbf{b} = [2, 0, 2]^\top$ (medium iced oat). Then

$$\mathbf{a} \cdot \mathbf{b} = (2)(2) + (1)(0) + (2)(2) = 8, \quad \|\mathbf{a}\| = \sqrt{4 + 1 + 4} = 3, \quad \|\mathbf{b}\| = \sqrt{4 + 0 + 4} = 2\sqrt{2}.$$

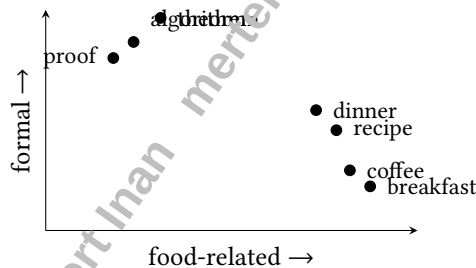
[**Pause.** Try computing $\cos \theta$ for these two orders. What does the value tell you about how similar they are?]

When $\cos \theta = 1$, the vectors point in the same direction (identical). When $\cos \theta = 0$, they are orthogonal (completely unrelated). This interpretation holds in any $\mathbb{R}^n$, whether $n = 3$ or $n = 300$.

1.3 Words as vectors

Here is where things get interesting. The linguist J.R. Firth observed in 1957 that “you shall know a word by the company it keeps.” The word *coffee* tends to appear near *morning*, *cup*, *energy*. The word *algorithm* appears in entirely different contexts.

Modern algorithms like Word2Vec and GloVe turn this observation into mathematics: they encode each word as a vector in $\mathbb{R}^{300}$ such that words appearing in similar contexts receive nearby vectors.



In this simplified two-dimensional picture, “coffee” and “breakfast” cluster together (both food-related, informal), while “algorithm” and “theorem” cluster elsewhere (formal, non-food). Real embeddings use 300 dimensions, but the geometric principle is identical: meaning is encoded as position, and similarity is cosine similarity.

Notice the contrast with a naive representation. If our vocabulary has 100,000 words, we could assign each word a *one-hot* vector in $\mathbb{R}^{100,000}$, with a single 1 in the position corresponding to that word and 0s everywhere else. In that representation, every pair of words is equally distant: “coffee” and “tea” are just as far apart as “coffee” and “algorithm.” Embedding systems compress $\mathbb{R}^{100,000}$ down to $\mathbb{R}^{300}$, and the compression is structured so that semantic relationships are preserved as geometric relationships.

[**Pause.** Can you think of two words that would be very close in this space? Two that would be very far apart?]

The takeaway from this section: any collection of objects with measurable properties can be represented as points in $\mathbb{R}^n$. The dot product and cosine similarity give us a precise, computable notion of “how similar.” The number of dimensions n can be as large as we need it to be.

§2 PROJECTION AND DIMENSIONALITY REDUCTION

2.1 The problem of redundant dimensions

We just saw that words can be represented as vectors in very high-dimensional spaces. Let us look at another text representation to see why this creates a problem.

In the *bag-of-words* model, a text document is represented as a vector of word counts over the entire vocabulary. If the vocabulary has 100,000 words, each document is a vector in $\mathbb{R}^{100,000}$: the i -th entry counts how many times the i -th word appears. A collection of m documents becomes a matrix in $\mathbb{R}^{100,000 \times m}$.

This space is enormous. But language is highly structured.

[**Pause.** If a document has a non-zero count for the word “car” (say, dimension 5,021), what do you expect for the word “auto” (dimension 5,022)? What about the word “pancake”?]

If “car” appears, “auto” is likely to appear too, because they describe the same concept. “Pancake” is probably absent. Most of the 100,000 dimensions are either zero or correlated with other dimensions. The effective information content is much lower than the raw dimensionality suggests. Most of those dimensions are telling us roughly the same thing in slightly different ways.

This motivates the central question of this section: *can we find a smaller set of numbers that faithfully captures the structure of the data?*

2.2 Projection onto a direction

The mathematical operation that makes this possible is *projection*. The idea is straightforward: pick a direction in the space, and for each data point, record how far it extends along that direction.

DEFINITION 2.1. Let $\mathbf{u} \in \mathbb{R}^n$ be a unit vector ($\|\mathbf{u}\| = 1$). The **scalar projection** of $\mathbf{x}$ onto $\mathbf{u}$ is

$$z = \mathbf{x} \cdot \mathbf{u} = \sum_{i=1}^n x_i u_i.$$

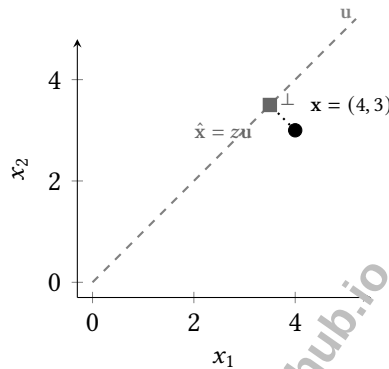
The number z is the one-dimensional representation of $\mathbf{x}$ along the direction $\mathbf{u}$. We can reconstruct an approximation $\hat{\mathbf{x}} = z \mathbf{u}$, and the residual $\mathbf{x} - \hat{\mathbf{x}}$ is always perpendicular to $\mathbf{u}$.

Let us work through a concrete case.

EXAMPLE 2.2. Let $\mathbf{x} = [4, 3]^\top$ and $\mathbf{u} = \frac{1}{\sqrt{2}}[1, 1]^\top$ (the 45° diagonal). Then

$$z = [4, 3] \cdot \frac{1}{\sqrt{2}}[1, 1]^\top = \frac{4+3}{\sqrt{2}} \approx 4.95.$$

Two numbers have been compressed to one. The reconstruction is $\hat{\mathbf{x}} = z\mathbf{u} \approx [3.5, 3.5]^\top$, and the residual is $\mathbf{x} - \hat{\mathbf{x}} \approx [0.5, -0.5]^\top$, which is perpendicular to $\mathbf{u}$.

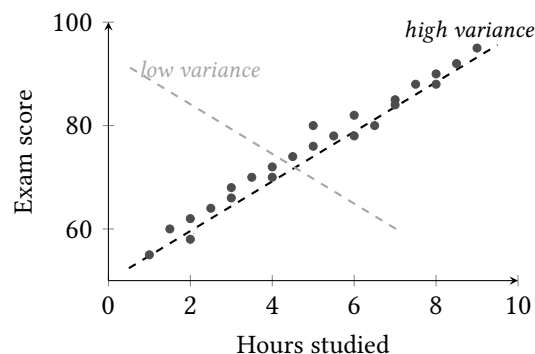


Geometrically, projection keeps the component of $\mathbf{x}$ that lies along $\mathbf{u}$ and discards the perpendicular component. This is the geometry of compression: we sacrifice the perpendicular part to reduce from two numbers to one.

[**Pause.** Look at the figure. If $\mathbf{u}$ pointed along the x_1 -axis instead of the diagonal, what would the projection of $(4, 3)$ be? What information would be lost?]

2.3 Choosing the best direction

The projection operation works for any direction $\mathbf{u}$. But the choice of direction matters enormously. Consider a cloud of data points in $\mathbb{R}^2$ that forms an elongated shape:



[**Pause.** Imagine projecting all the points onto the low-variance direction (the short axis). What happens to students who had very different exam scores?]

They collapse onto nearly the same value. The compression destroys exactly the differences we care about. Projecting onto the high-variance direction (the long axis) preserves the spread: students who were far apart remain far apart.

The quantity that formalizes this is the **variance of the projected data**. Given m data points $\mathbf{x}_1, \dots, \mathbf{x}_m$ with mean $\bar{\mathbf{x}} = \frac{1}{m} \sum_{j=1}^m \mathbf{x}_j$, the variance along a unit vector $\mathbf{u}$ is

$$\text{Var}_{\mathbf{u}} = \frac{1}{m} \sum_{j=1}^m ((\mathbf{x}_j - \bar{\mathbf{x}}) \cdot \mathbf{u})^2.$$

The best single direction for compression is the $\mathbf{u}$ that maximizes this.

2.4 The answer: eigenvectors of the covariance matrix

We can write $\text{Var}_{\mathbf{u}}$ in a cleaner form. Define the **covariance matrix**:

$$C = \frac{1}{m} \sum_{j=1}^m (\mathbf{x}_j - \bar{\mathbf{x}})(\mathbf{x}_j - \bar{\mathbf{x}})^{\top} \in \mathbb{R}^{n \times n}.$$

Then the variance becomes $\text{Var}_{\mathbf{u}} = \mathbf{u}^{\top} C \mathbf{u}$, and the optimization problem has a remarkably clean solution:

PROPOSITION 2.3. *The unit vector maximizing $\mathbf{u}^{\top} C \mathbf{u}$ is the eigenvector of C corresponding to its largest eigenvalue: $C \mathbf{u}^* = \lambda_1 \mathbf{u}^*$, where $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$.*

We will prove this later in the course using Lagrange multipliers. For now, the important thing is what it tells us: eigenvectors are the directions along which data varies most. They are not an abstract algebraic curiosity that exists only in textbooks. They answer a concrete, practical question about how to compress data optimally.

The second eigenvector (perpendicular to the first, with the second-largest eigenvalue) gives the second-best direction. The third is perpendicular to both, and so on. Keeping the top k eigenvectors and projecting onto them compresses $\mathbb{R}^n$ to $\mathbb{R}^k$ while retaining exactly $\frac{\lambda_1 + \dots + \lambda_k}{\lambda_1 + \dots + \lambda_n}$ of the total variance. This procedure is called **Principal Component Analysis** (PCA), and it is the optimal linear compression to k dimensions.

Returning to our bag-of-words example: if documents in $\mathbb{R}^{100,000}$ have most of their meaningful variation concentrated in 200 directions, PCA can compress each document from 100,000 numbers to 200 while retaining the structure that distinguishes, say, a sports article from a cooking recipe.

§3 CONNECTIONS TO COMPUTER SCIENCE

Everything we developed today (vectors, dot products, projections, eigenvectors) is the same mathematics that powers real systems you interact with every day.

3.1 Neural networks

A neural network transforms data through successive layers:

$$\mathbb{R}^{784} \xrightarrow{W_1, \sigma} \mathbb{R}^{256} \xrightarrow{W_2, \sigma} \mathbb{R}^{64} \xrightarrow{W_3} \mathbb{R}^{10}$$

where each W_i is a weight matrix and σ is a nonlinear activation function applied element-wise. Each layer takes its input, multiplies by a matrix (a linear operation, the kind we study in this course), and then applies σ .

What happens if we remove σ ? The composition $W_3 W_2 W_1$ is a single linear map from $\mathbb{R}^{784}$ to $\mathbb{R}^{10}$: a learned linear projection, closely related to PCA. The nonlinearity σ is what allows neural networks to capture curved structure that linear projections cannot represent, like data arranged along a spiral rather than an ellipse.

PCA is the linear, closed-form case. Neural networks generalize it to the nonlinear, learned, task-specific case. Understanding PCA gives you the mathematical foundation on which deep learning is built.

3.2 Large language models and the dot product

Large language models (LLMs) like GPT and Claude operate in extremely high-dimensional vector spaces. When you type a sentence, each word (or subword token) is mapped to a vector in $\mathbb{R}^d$ for some large d (GPT-3 uses $d = 12,288$). The model transforms these vectors through dozens of layers, each one a matrix multiplication followed by a nonlinearity, exactly the architecture described above.

The *attention mechanism* at the heart of these models computes similarity between token vectors using a scaled version of the dot product, the same operation we defined in §1. When the model decides that the word “bank” in a sentence is related to “river” rather than “money,” it is comparing dot products between vectors. The linear algebra from today’s lecture is, quite literally, the mathematics running inside every LLM interaction.

This is one of the reasons this linear algebra course is a prerequisite for advanced CS coursework: the operations you learn here (dot products, projections, eigenvectors, matrix factorizations) are the operations that power these systems.

§4 SUMMARY

Let us collect what we learned today.

The big idea. Data can be represented as vectors in $\mathbb{R}^n$. Most real-world data lives in spaces that are much larger than necessary. The fundamental question of dimensionality reduction is: which directions in the space carry the most information?

The tools we built.

1. **Vectors** ($\mathbf{x} \in \mathbb{R}^n$): ordered lists of numbers, representing objects as points in a space.
2. **Dot product** ($\mathbf{x} \cdot \mathbf{y} = \sum x_i y_i$): measures agreement between vectors. Normalized, it gives cosine similarity.
3. **Projection** ($\mathbf{z} = \mathbf{x} \cdot \mathbf{u}$): compresses a vector onto a direction. The residual is perpendicular to $\mathbf{u}$.
4. **Variance maximization**: the best direction to project onto is the one that maximizes the spread of the projected data.
5. **Eigenvectors of C** : these are exactly the directions that maximize variance. PCA keeps the top k of them.

The thread connecting them. Vectors let us represent data. The dot product lets us measure similarity and compute projections. Projections let us compress. Eigenvectors tell us which compressions are optimal. This same thread runs through word embeddings, text compression, neural networks, and large language models.

§5 WHAT COMES NEXT

This lecture was the opening of the course. In the next lectures, we will build the full linear algebra toolkit: matrix multiplication and linear transformations, systems of equations and Gaussian elimination, determinants and invertibility, eigenvalue decomposition (including a proof that eigenvectors solve the variance-maximization problem), the Singular Value Decomposition and its relationship to PCA, and low-rank matrix approximation. Each topic connects back to the geometric intuition from today: data lives in vector spaces, and the operations of linear algebra let us reason about, transform, and compress that data.

End of Lecture 1. See you next time.

§6 PROBLEMS TO THINK ABOUT

These are not graded. They are meant to help you sit with the ideas from today. Try them on your own or with a classmate.

1. **Vectors and similarity.** Represent three movies you have seen recently as vectors in $\mathbb{R}^4$ using the dimensions: (1) how action-heavy (0–10), (2) how funny (0–10), (3) length in minutes, (4) year of release. Compute the dot product and cosine similarity between each pair. Does cosine similarity agree with your intuitive sense of which two movies are most alike?
 2. **Projection by hand.** Let $\mathbf{x} = [3, 4]^\top$ and $\mathbf{u} = [1, 0]^\top$ (the x_1 -axis).
 - (a) Compute the scalar projection $z = \mathbf{x} \cdot \mathbf{u}$.
 - (b) Compute the reconstruction $\hat{\mathbf{x}} = z \mathbf{u}$ and the residual $\mathbf{x} - \hat{\mathbf{x}}$.
 - (c) Verify that the residual is perpendicular to $\mathbf{u}$ by checking their dot product.
 - (d) Now repeat with $\mathbf{u} = \frac{1}{\sqrt{2}}[1, 1]^\top$. Which projection loses less information, and why?
 3. **Variance and direction.** Suppose you have four data points in $\mathbb{R}^2$: $(1, 1)$, $(2, 2)$, $(3, 3)$, $(4, 4)$. These points lie perfectly on the line $x_2 = x_1$.
 - (a) What is the direction of maximum variance?
 - (b) If you project onto this direction, how much variance is retained?
 - (c) What happens if you project onto the perpendicular direction?
 4. **Thinking about text.** Consider a bag-of-words representation over a vocabulary of 5 words: [“cat,” “dog,” “fish,” “computer,” “algorithm”]. Document A is “the cat and the dog” = $[1, 1, 0, 0, 0]^\top$. Document B is “the fish and the cat” = $[1, 0, 1, 0, 0]^\top$. Document C is “the computer algorithm” = $[0, 0, 0, 1, 1]^\top$.
 - (a) Compute cosine similarity between each pair.
 - (b) Which two documents are most similar? Does this match your intuition?
 - (c) These vectors live in $\mathbb{R}^5$. If you could compress to $\mathbb{R}^2$, what two “topic directions” might PCA find?
 5. **A question to sit with.** Pick any object you interact with daily (a playlist, a meal, a route to campus) and think about how you would represent it as a vector. What would the dimensions be? What would “nearby” mean? What information would be lost if you compressed to fewer dimensions?
-

§7 FURTHER READING

The following references are optional. They are here for students who want to explore further, or who want to see where the mathematics from today leads. You are not expected to read these now.

Linear algebra and dimensionality reduction:

- [1] G. Strang, *Introduction to Linear Algebra*, 6th ed., Wellesley-Cambridge Press, 2023.
- [2] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning*, Cambridge, 2014. Ch. 23 covers PCA.
- [3] S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” *Chemometrics and Intelligent Laboratory Systems*, 2(1–3):37–52, 1987.

Word embeddings:

- [4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv:1301.3781*, 2013.
- [5] J. Pennington, R. Socher, and C.D. Manning, “GloVe: Global vectors for word representation,” *EMNLP*, 2014.

Attention, transformers, and LLMs:

- [6] A. Vaswani et al., “Attention is all you need,” *NeurIPS*, 2017.
- [7] J. Elhage et al., “A mathematical framework for transformer circuits,” *Anthropic Research*, 2021.

Where to start: Mikolov et al. [4] is short and readable. The figures in Elhage et al. [7] show how linear algebra concepts appear inside real neural networks. As we build up tools like eigenvalue decomposition and SVD later in the course, these papers will become increasingly accessible.