

Conceptors: An Introduction from First Principles

A walk from "what is a word to a computer?" to "what is a conceptor, and why would anyone want one?"

Tyler M. Moore^{1,2} · Ilias Triantafyllopoulos³ · Ren Tao⁴ · Neville G. Ryant⁵ · Sunghye Cho⁶ · Lyle H. Ungar⁴ · João Sedoc³

¹ Department of Psychiatry, Perelman School of Medicine, University of Pennsylvania · ² Lifespan Brain Institute, Penn Medicine and Children's Hospital of Philadelphia · ³ Department of Technology, Operations, and Statistics, Stern School of Business, New York University · ⁴ Department of Computer and Information Science, University of Pennsylvania · ⁵ Linguistic Data Consortium, University of Pennsylvania · ⁶ Department of Communication Sciences and Disorders, University of Cincinnati

Manuscript · October 2026 Correspondence: tymoore@pennmedicine.upenn.edu ORCID 0000-0002-1384-0151
DOI 10.5281/zenodo.23734290

Recommended citation: Moore, T. M., Triantafyllopoulos, I., Tao, R., Ryant, N. G., Cho, S., Ungar, L. H., & Sedoc, J. (2026). *Conceptors: An introduction from first principles*. Manuscript retrieved from <https://mooremetrics.com/conceptors>. <https://doi.org/10.5281/zenodo.23734290>

Before we start

Who this is for, and a promise

This article is written for a curious, educated reader who has not done anything technical in a long time. You may remember, at least vaguely, what a t-test is and what a line of best fit is; that is all the background it assumes. You have probably never heard the words "encoding," "layer," "steering," or "conceptor" used the way this article uses them.

Here is the promise. If you read this in order and do not skip the early sections (even where they feel slow), then by the end you will be able to explain, in your own words and with a picture in your head:

1. what it means to **encode** a word or a sentence as numbers;
2. how a **large language model** is trained, and why that is really just the line of best fit with more knobs;
3. what the **layers** of such a model are, and why the middle ones matter most for our purposes;
4. what **steering** a model means;
5. what a **conceptor** is;
6. what steering can do that no amount of careful prompt-writing can do, and what a conceptor adds to steering.

Each step leans on the one before it. Losing the thread early makes everything later feel like magic, so the early sections are deliberately thorough, and important ideas are restated more than once, from different angles. In the later, harder sections you will see boxes labeled *Another way to think about it*; those are second doors into the same room, in case the first one was locked for you.

What you need to know already

Everything in this article reduces to a handful of things most people met in school or college and still half-remember:

- **Averages, and differences between averages.** The single most important object in this article, the "steering arrow," is literally the difference between two group averages: the top half of a t-test.
- **Spread.** The idea that a set of numbers has a typical distance from its average (a standard deviation). A conceptor is built from how much a set of examples spreads out, direction by direction.
- **The line of best fit.** Training a language model is the same idea as fitting a line: adjust the knobs until the error is as small as you can get it.

- **Tables of numbers.** Rows and columns, like a spreadsheet. Whenever this article says "row of numbers," picture one row of a spreadsheet; whenever it says "table," picture the whole sheet. Nothing more exotic is needed. The article shows the one formula of the research literature exactly twice, in Section 5.2 and in the glossary, for readers who want to connect what they read here to the papers; it is never needed to follow the argument.

How the illustrations were made

Most explanations of this material use cartoon diagrams: a box labeled "Encoder" with an arrow to a box labeled "Decoder." Those pictures do not help anyone who does not already know what is in the boxes. This article avoids them. Instead, nearly every figure was produced by running a real language model and plotting what came out.

The model is **GPT-2** (the 124-million-parameter version released by OpenAI in 2019). It is small and old by today's standards, which is a feature here: it is small enough to run on an ordinary computer, its internals are fully public, and yet it displays every phenomenon this article needs to show. Where this article quotes results on bigger, newer models, it says so and cites the source. A short appendix describes exactly how each experiment was done.

THE RESEARCH BEHIND THIS ARTICLE

The technical content is drawn from research papers listed in full at the end. The central one is a study of the geometry of conceptors in language models by Triantafyllopoulos and colleagues (in preparation), several of whom are authors of this article; its published precursor is Triantafyllopoulos and colleagues (2026). The others are: a paper that steers a model to improve mental-health questionnaire scoring (Ravenda and colleagues, 2025); a paper that reads depression-symptom signals directly out of a model's internal numbers (Zhu, Subramanian and colleagues, 2026); a study of "human value" steering directions (Abootorabi and colleagues, 2026); a study of what each layer of a model represents (Skean and colleagues, 2025); a study of how models represent truth versus falsehood (Marks and Tegmark, 2024); the paper that first applied conceptors to language models (Postmus and Abreu, 2024); and a long study from Anthropic of the concepts a model "has in mind" as it works (Gurnee, Sofroniew, Lindsey and colleagues, 2026). Where a specific number comes from one of them, the text says which.

Contents

- 1 Words as numbers (what "encoding" means)
- 2 How a language model is trained
- 3 Layers
- 4 Reading and steering

[5](#) [Conceptors](#)

[6](#) [What steering can do that prompts cannot, and what conceptors add](#)

[7](#) [The whole story on one page · Glossary · Sources · Appendix: how the experiments were run](#)

1 Words as numbers (what "encoding" means)

A computer cannot do arithmetic on the word *coffee*. It can only do arithmetic on numbers. So the first thing a language model must do is turn every word into numbers, in a way that keeps the word's meaning. That is what encoding is. This section builds the idea slowly, because everything later stands on it.

1.1 You already encode things

Think of a credit score. A person's entire borrowing history is squeezed into one number, 300 to 850, arranged so that *nearby numbers mean similar risk*. Two applicants with scores of 740 and 750 will be treated much the same; 740 and 550 will not. The number is not the person, but it carries the part of the person that a lender cares about. That is an encoding: a fixed rule for writing something as numbers so that the numbers preserve what matters.

One number is rarely enough. A house listing uses several: square footage, number of bedrooms, year built, distance to the nearest school. Together those four numbers are a small **row** in a spreadsheet, and two houses with similar rows are similar houses. Wine reviewers do the same with sweetness, acidity, tannin and body. A personality test turns a person into five numbers. A doctor's chart turns a patient into a row of vital signs. In every case, the trick is the same:

THE CORE IDEA

An encoding writes a thing as a row of numbers, chosen so that similar things get similar rows. Once things are rows of numbers, "how similar are these two things?" becomes "how close are these two rows?", which is a question arithmetic can answer.

Figure 1 shows the smallest possible version with foods. Each food gets two numbers, a sweetness score and a saltiness score (rough, made-up values, just to make the point). With two numbers per food, each food becomes a dot on a page. Notice three things about the picture, because all three will come back for language models:

1. **Similar foods land near each other.** Pretzels sit next to potato chips; honey sits next to cake.
2. **Distance is a fair measure of similarity.** Bread is closer to plain rice than to honey, and that matches intuition.
3. **Directions on the page mean something.** Moving to the right means "sweeter." Moving up means "saltier." The orange arrow is not a food; it is a *direction*, and it has a meaning of its own.

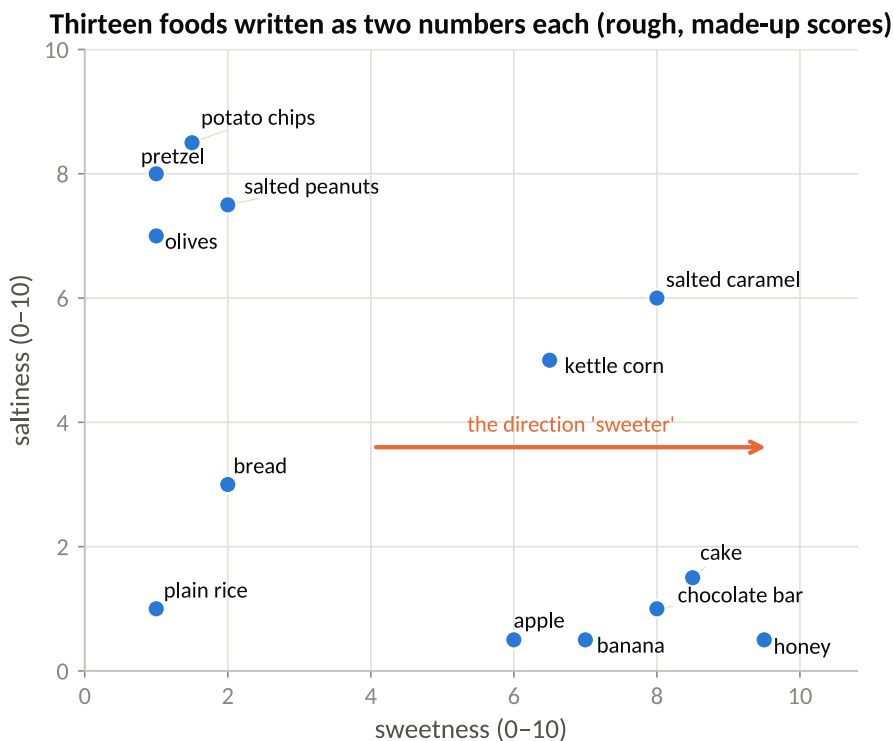


Figure 1. Thirteen foods, each encoded as two numbers (sweetness and saltiness, on invented 0–10 scales). Similar foods become nearby dots, and the direction "sweeter" is a meaningful arrow on the page. Everything a language model does with words starts from this picture, only with hundreds of numbers instead of two.

1.2 Nobody chooses the columns

For foods, a person chose the two columns, sweetness and saltiness. For a language model there is no such person. The model uses hundreds of columns per word (768 in the model used for this article, and several thousand in today's large models), and no human decided what any column means. The columns are filled in automatically during training, by a process described in Section 2. For now, take it on faith that there is a way to fill them in so that words used in similar ways get similar rows.

The principle behind it is old and has a slogan from the 1950s: "*You shall know a word by the company it keeps.*" The words *coffee* and *tea* appear in the same kinds of places: after "a cup of," before "with milk," near "hot," "drink," "morning." The words *coffee* and *truck* do not. A procedure that gives similar rows to words that keep similar company will, without anyone telling it so, end up putting coffee next to tea and far from truck.

1.3 A word really is a row of 768 numbers

Figure 2 shows real rows from the real model. GPT-2 keeps a **lookup table** with one row per word-piece it knows (50,257 rows) and 768 columns. The figure shows the first 16 of the 768 numbers for seven words. This is exactly what the model "sees" when it reads the word *cat*: not the letters c-a-t, but the row of 768 numbers that begins 0.01, 0.04, 0.16, -0.22, 0.03, -0.10, -0.34, -0.08, ...

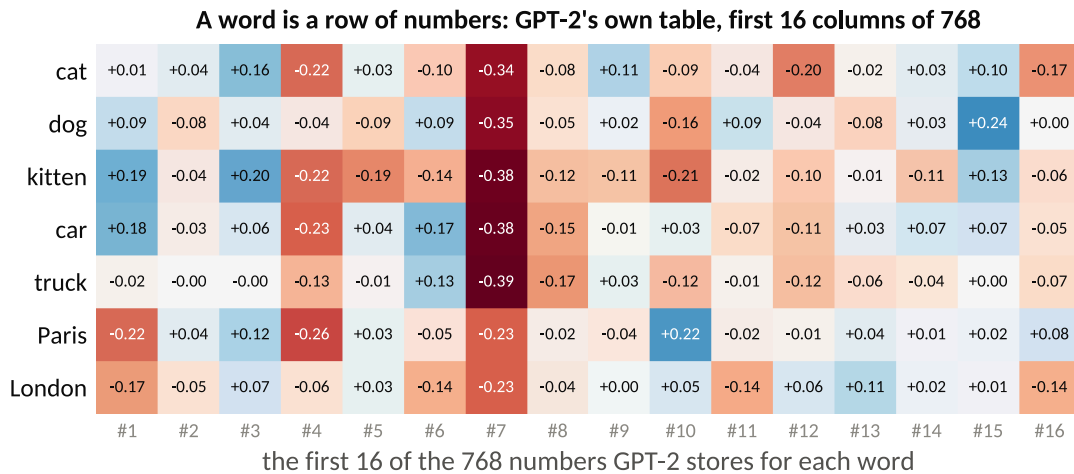


Figure 2. The first 16 of the 768 numbers GPT-2 stores for each of seven words, exactly as they sit in the model's lookup table (blue positive, red negative). No single column "means" anything on its own, and you cannot spot the family resemblances by eye from 16 columns; they only become visible when all 768 numbers are compared at once, which is what the similarity scores in Section 1.4 do.

Two things to notice. First, no individual column has an obvious meaning. Column #7 is not "furriness." The information is spread across the whole row, the way a person's creditworthiness is spread across many line items rather than sitting in one. Second, the rows look like noise to a human eye, and that is fine: the model never looks at one column in isolation either. It only ever combines all 768 at once, and at that level the resemblance between *cat*, *dog* and *kitten* is strong and measurable, as the next subsection shows.

1.4 Similarity is closeness, and we can measure it

With two numbers per item you can see closeness on a page. With 768 you cannot draw the page, but you can still compute closeness. The standard score is computed almost exactly like a correlation coefficient between two rows: it comes out near 1 when the two rows rise and fall together, and near 0 when they are unrelated. (Technically it is the correlation without first subtracting each row's mean, and it is called the *cosine similarity*; you can treat it as "a correlation between two rows of numbers.") Here are real values from the model's lookup table:

Pair of words	Similarity of their rows	Pair of words	Similarity
guitar – piano	0.67	guitar – banana	0.35
teacher – student	0.63	teacher – truck	0.31
coffee – tea	0.62	coffee – truck	0.31
happy – glad	0.62	happy – truck	0.23
Paris – London	0.58	Paris – banana	0.21
doctor – nurse	0.58	doctor – banana	0.23

Pair of words	Similarity of their rows	Pair of words	Similarity
cat – dog	0.55	cat – truck	0.27

The absolute values are not what matters (unrelated words sit around 0.2–0.3 in this model rather than at 0, because all words share some common "this is an English word" background). What matters is the ordering, which is consistently sensible. Figure 3 shows the same idea as a grid, and Figure 4 shows a map.

Similarity between GPT-2's rows of numbers (1 = identical direction, 0 = unrelated)

cat	1.00	0.55	0.50	0.38	0.27	0.23	0.27	0.31	0.27	0.25
dog	0.55	1.00	0.48	0.42	0.33	0.28	0.21	0.22	0.28	0.28
kitten	0.50	0.48	1.00	0.29	0.30	0.26	0.24	0.29	0.30	0.29
car	0.38	0.42	0.29	1.00	0.54	0.45	0.27	0.27	0.26	0.22
truck	0.27	0.33	0.30	0.54	1.00	0.48	0.25	0.23	0.23	0.26
bus	0.23	0.28	0.26	0.45	0.48	1.00	0.23	0.23	0.25	0.28
Paris	0.27	0.21	0.24	0.27	0.25	0.23	1.00	0.58	0.28	0.24
London	0.31	0.22	0.29	0.27	0.23	0.23	0.58	1.00	0.26	0.22
happy	0.27	0.28	0.30	0.26	0.23	0.25	0.28	0.26	1.00	0.55
sad	0.25	0.28	0.29	0.22	0.26	0.28	0.24	0.22	0.55	1.00
	cat	dog	kitten	car	truck	bus	Paris	London	happy	sad

Figure 3. Similarity between rows for ten words. The three blocks along the diagonal (animals, vehicles, cities) show that words within a family have similar rows. Note the curious fact that *happy* and *sad* are quite similar (0.55): they keep the same company ("I feel ...", "she looked ..."). Being opposites is a kind of closeness. We will return to this.

48 words placed on a map so that similar rows of numbers land near each other

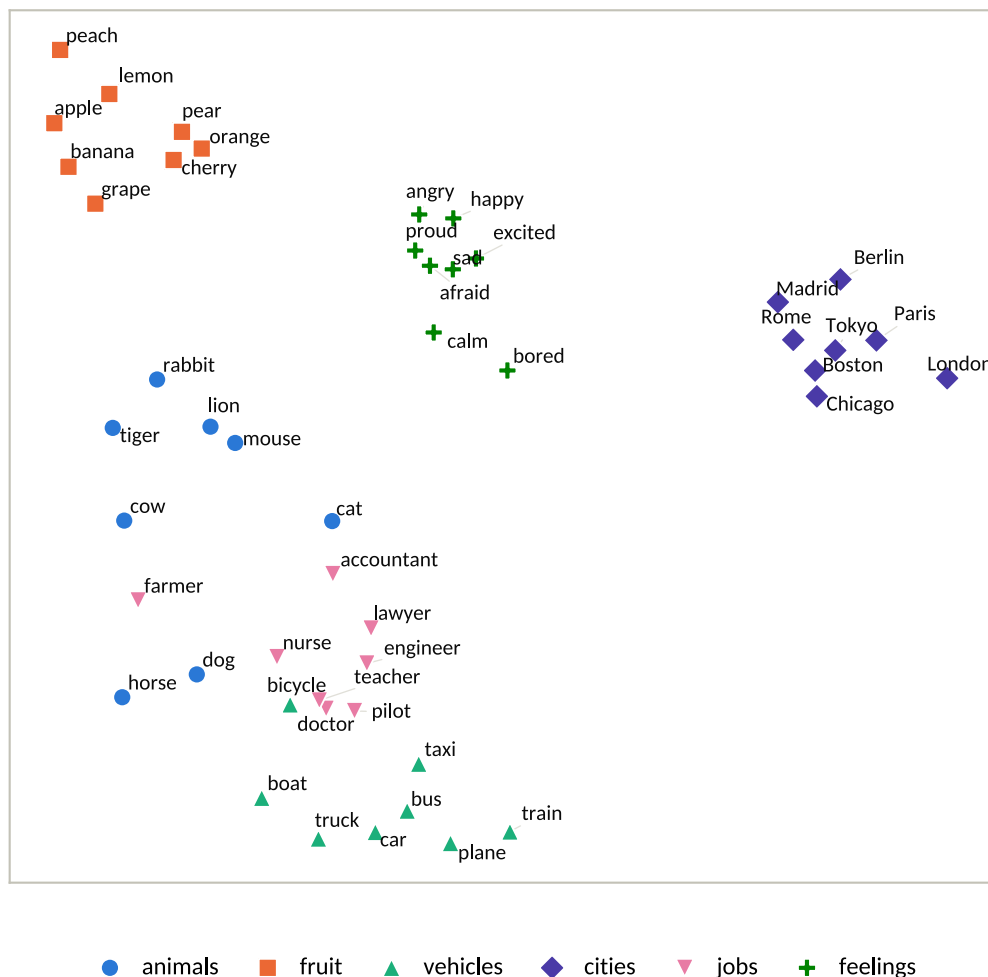


Figure 4. Forty-eight words from GPT-2's lookup table, placed on a flat map so that words with similar rows land near each other (a standard "squash 768 dimensions down to 2 while preserving distances as well as possible" procedure). Nobody told the model that Berlin and Madrid belong together, or that lemon and peach do. The map has no named axes; only the distances mean anything.

Pause on Figure 4 for a moment. The model was never given a dictionary, a list of cities, or a list of fruits. It saw only text. The clustering is entirely a by-product of "words that keep similar company get similar rows." That is what it means to say the numbers carry meaning.

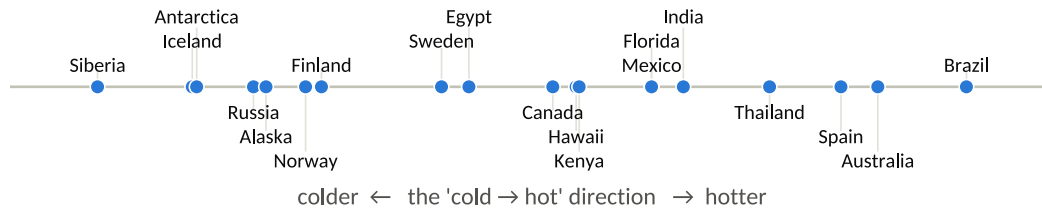
1.5 Directions have meaning too

In the food picture, "to the right" meant "sweeter." The same thing is true in the 768-number space, and it is the single most important fact in this article, because steering and conceptors are both built on it.

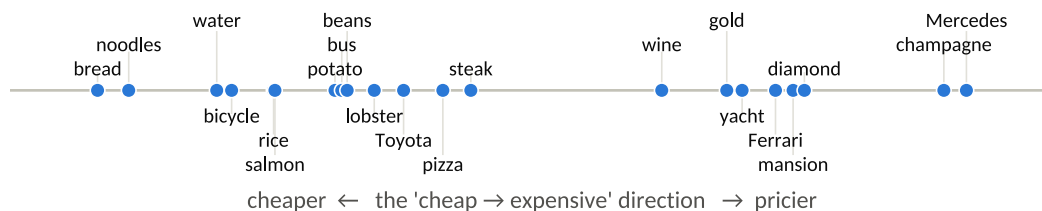
Here is how to find a meaningful direction. Take a few words that mean "cold" (cold, freezing, icy, chilly, frozen) and average their rows. Take a few that mean "hot" (hot, warm, tropical, sunny) and average those. Subtract: *hot-average minus cold-average*. The result is a row of 768 numbers that is

not a word at all; it is an arrow pointing from "the cold region" to "the hot region." Now take any other word, say *Iceland*, and ask how far along that arrow it sits. Do that for twenty places and you get a ruler (Figure 5, top).

Places, ordered by their position along one direction in GPT-2's number-space



Things, ordered by their position along one direction in GPT-2's number-space



Events and things, ordered by their position along one direction in GPT-2's number-space

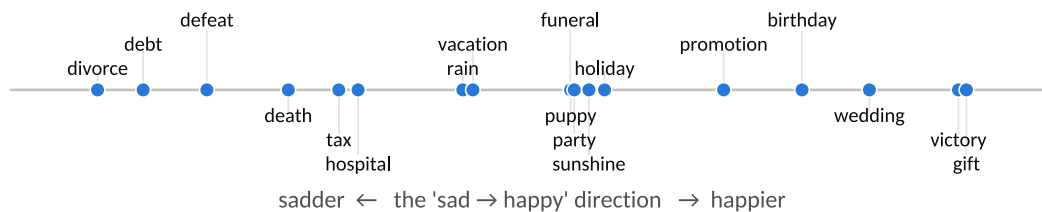


Figure 5. Three rulers, each made from GPT-2's lookup table. Top: places, ordered by their position along the direction from "cold words" to "hot words." Middle: things, ordered along the direction from "cheap words" to "expensive words." Bottom: events, ordered along "sad" to "happy." Nobody labeled any of these; the orderings fall out of the geometry. They are imperfect (Egypt lands on the cool side of the temperature ruler; a funeral lands, absurdly, just on the happy side of center), which is an honest picture of a small 2019 model's raw lookup table. The middle layers of the model, which we meet in Section 3, do considerably better.

Three rulers, three directions, none of them designed by a person. Siberia, Antarctica and Iceland at one end; Brazil, Australia and Spain at the other. Bread, noodles and water at one end; Mercedes, champagne, diamond and mansion at the other. Divorce, debt and defeat at one end; gift, victory, wedding and birthday at the other.

No single one of the 768 columns is "temperature," just as no single measurement on a doctor's chart is "fitness." But a particular *combination* of the columns (the hot-minus-cold arrow) behaves like a temperature gauge, and once you have the combination you can apply it to any row and read off a number.

The arrow trick even supports a kind of arithmetic that surprised the researchers who first noticed it. Take the row for *king*, subtract the row for *man*, add the row for *woman*, and look for the word whose row is closest to the result. In GPT-2's lookup table the answer is *queen* (similarity 0.71, with *princess* in second place). Do the same with *bigger* – *big* + *small* and you get *smaller*. The "male-to-female" direction and the "plain-to-comparative" direction are both real, reusable arrows in the space.

1.6 From a word to a whole sentence

So far, every word has a fixed row. But a word's meaning depends on its neighbors: *bank* in "river bank" is not *bank* in "bank account," and the sentence "The soup was cold" means something different from the sentence "The beer was cold." A fixed lookup table cannot capture that. This is where the rest of the model comes in.

The rows in the lookup table are only the model's *starting point*. As the model reads a sentence, it repeatedly rewrites the row of numbers sitting at each word position, mixing in information from the surrounding words, until the row at a given position no longer represents "the word *was*" but something more like "the state of this sentence so far, up to and including *was*." How that rewriting happens is the subject of Section 3 (layers). What it produces is a **contextual encoding**: a row of 768 numbers that encodes the sentence-so-far rather than a single word.

Two small housekeeping facts will help you read the later figures:

- **The model does not quite work in words; it works in "tokens."** Common words are one token each. Rare or long words are split into pieces. GPT-2 reads "Dear Sir or Madam" as five tokens: Dear Sir or Mad am , and reads the word "unbelievable" as un bel iev able . (The marks a token that begins with a space.) Punctuation marks are tokens too. When this article says "word," read "token."
- **Every token position carries its own row of numbers.** A ten-token sentence is, inside the model, a table of ten rows and 768 columns. Keep that table in mind: rows are tokens, and there are exactly as many rows as tokens. Section 3 shows how the layers rewrite all ten rows.

2 How a language model is trained

Section 1 asked you to take on faith that the 768 columns get filled in "automatically during training." This section pays that debt. The punchline: training a language model is the line of best fit, with one change of task and a great many more knobs.

2.1 The line of best fit, revisited

You have seen a line fitted to a scatter of points. Recall what that actually involves. A line has two **knobs**: a slope and an intercept. For any setting of the two knobs, there is an **error score**: add up the squared vertical misses between the line and the points. Fitting the line means finding the knob settings that make the error score as small as possible. Figure 6 shows this in three panels.

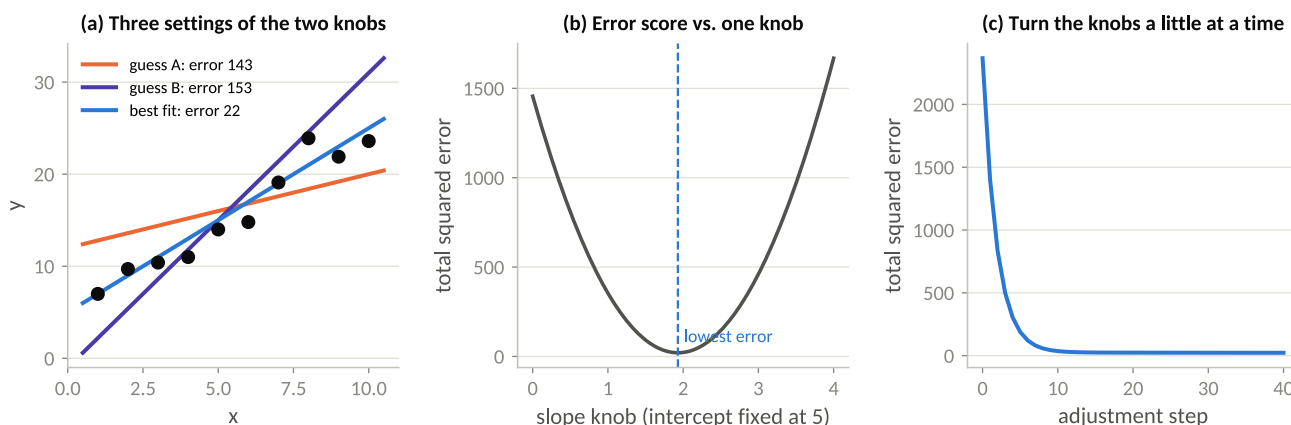


Figure 6. (a) Ten points and three candidate lines; each line's two knobs produce an error score. (b) Holding the intercept fixed and turning only the slope knob traces out a valley; the best slope is at the bottom. (c) A different way to find the bottom, and the one that scales to millions of knobs: start anywhere, nudge every knob a little in whichever direction reduces the error, and repeat. The error falls step by step.

Panel (c) deserves a second look, because it is the actual procedure used to train language models. For a straight line you could solve for the best knobs with a formula. With millions of knobs you cannot. Instead, you compute, for each knob, whether turning it up or down would reduce the error, turn each knob a tiny amount the helpful way, and repeat, thousands or millions of times. Each repetition is one **adjustment step**. The error creeps down. That is all "training" means: *repeatedly nudging the knobs to shrink an error score*. It is like tuning an old radio by ear: you cannot see the station, but you can hear the static, and you turn the dial a little at a time in whichever direction makes the static fade.

2.2 The task: guess the next word

A line of best fit predicts y from x . What does a language model predict? The answer is almost embarrassingly simple: **given the text so far, it predicts the next word**.

- "Peanut butter and ____"
- "Dear Sir or ____"
- "To be or not to ____"
- "The nurse checked the ____"

Rather than committing to one answer, the model produces a **probability for every token it knows** (50,257 of them for GPT-2). Figure 7 shows the real output for four prompts. After "To be or not to" the model puts 96% of its probability on *be*. After "Dear Sir or" it puts 68% on the token *Mad* (the first piece of "Madam"). After "The nurse checked the" it is spread out: *patient*, *blood*, *baby*, *room*, all plausible, none dominant. That spread is honest: many continuations are reasonable, and the model says so.

GPT-2's top guesses for the next token, with the probability it gives each (▬ = a leading space)

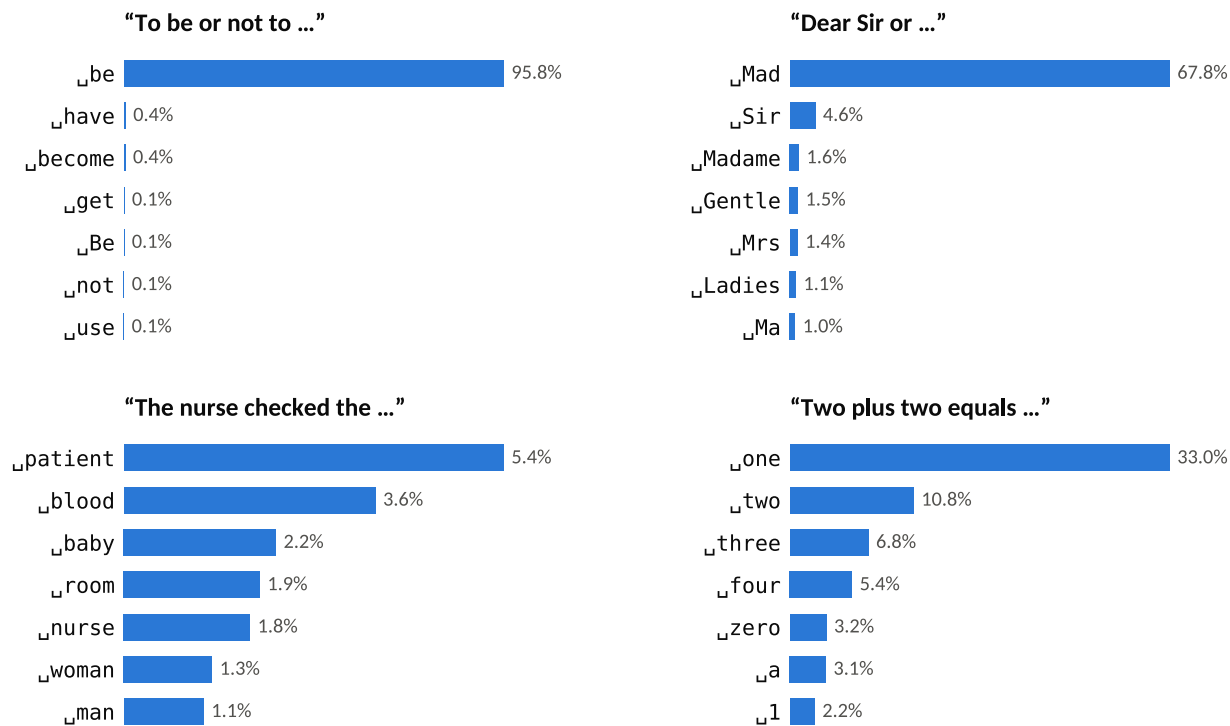


Figure 7. GPT-2's actual next-token probabilities for four prompts. The fourth panel is a reminder of what this model is and is not: asked to complete "Two plus two equals," it gives 33% to *one* and only 5% to *four*. It is not calculating; it is predicting which token tends to follow such text, and in its training data phrases like "two plus two equals one" (as a joke, a lyric, a title) were common enough. Larger, newer models do far better at arithmetic, but through the same mechanism: predicting text.

To produce a paragraph, the model simply repeats: predict a distribution over the next token, pick one (the most likely, or a random draw weighted by the probabilities), append it to the text, and predict again. Every fluent paragraph a chatbot has ever written was produced one token at a time in this way. Here, for instance, is GPT-2 continuing the prompt "The librarian opened the book and", picking randomly from its distribution at each step:

The librarian opened the book and found his hand pressed against the parchment. "I see," he said, "I know you're the one who writes and

It is not deep, and it stops mid-sentence because we stopped it after 25 tokens, but it is grammatical English on topic, produced by nothing more than 25 rounds of "guess the next token."

2.3 What plays the role of the error score

For the line, the error was the squared miss. For next-word prediction, the error is **how little probability the model gave to the word that actually came next**. If the true next word was *jelly* and the model had given *jelly* 90%, that is a small error; if it had given *jelly* 0.1%, that is a large one. (The precise formula is the negative logarithm of that probability, which behaves the way you would want: it is near zero when the model was confident and right, and it grows without bound as the model's probability for the truth shrinks toward zero.) Add this up over billions of words of text and you have the error score that training tries to shrink.

Now the whole recipe can be stated in one breath:

TRAINING A LANGUAGE MODEL, IN ONE SENTENCE

Take an enormous pile of ordinary text; at every position, have the model guess the next word; score the guess by how much probability it gave to the word that was really there; nudge all of the model's knobs a hair in the direction that would have made that probability higher; repeat for a very long time.

GPT-2 was trained this way on about 40 gigabytes of web pages (eight million documents). Modern models are trained on tens of trillions of words. Nobody labels anything. Nobody writes a rule. The text itself supplies the right answer at every position, because the right answer is simply the word that came next.

2.4 A tiny model, trained in front of you

It is hard to believe that "guess the next word" alone could produce the sensible word map of Figure 4. So here is a demonstration small enough to see all of, run for this article.

A made-up world has 36 words and 180 sentences, all of the form "<person> drank <drink> from a cup," "<person> ate <food> on a plate," "<person> drove <vehicle> to work," with four people, five drinks, five foods, five vehicles and a few variations of ending. A tiny next-word predictor was built with just **two** numbers per word instead of 768 (so the encoding can be drawn directly, with no squashing), plus one weighted-sum step from those two numbers to a probability for each possible next word. In all it has 180 knobs. The two numbers for every word started out random. Then the recipe above was run for 3,000 small adjustment steps. Figure 8 shows what happened.

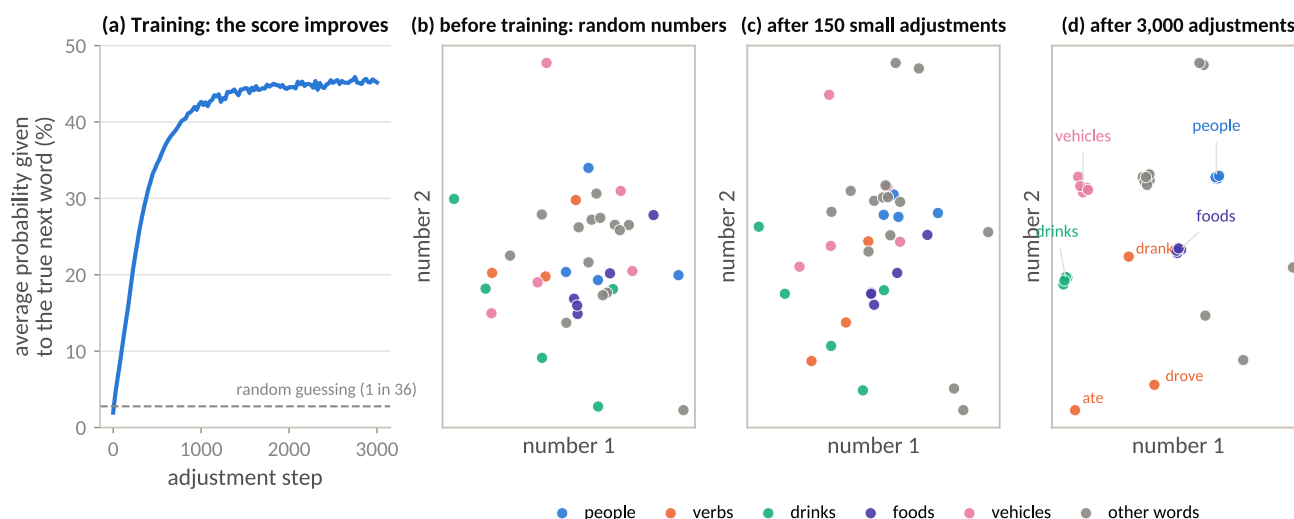


Figure 8. A 180-knob next-word predictor trained from scratch on a 36-word toy language. (a) The average probability the model gives to the true next word rises from about 2% (a random guess among 36 words gives 2.8%) to about 45%. (b–d) The two numbers per word, plotted as dots, before, during and after training. Nobody told the model which words were drinks; drinks ended up together because they are all followed by "from," all foods by "on," all vehicles by "to," and all people by a verb. The encoding is a by-product of prediction.

Look at panel (d). All five drinks sit on top of one another. All five foods do. All four people do. The three verbs sit apart from everything, and apart from each other, because each verb predicts a different set of next words. The model was never told any of this. It was only told, 3,000 times, "your guess for the next word was too unsure; adjust." Words that play the same role in predicting what comes next get pushed to the same place, because putting them in the same place is the cheapest way to make the same predictions for them.

That is the entire secret of Figure 4. GPT-2's 768 numbers per word came from the same process, on a vastly larger scale: a much bigger world (the web), a much bigger lookup table, but the same rule. Words that are followed and preceded by similar things end up with similar rows.

2.5 What the knobs are

You do not need to know the model's inner wiring to follow this article, but one fact makes the next section much easier: **almost every knob is a weight in a weighted sum.** Figure 9 shows the whole idea with a miniature: rows of 5 numbers instead of 768, and a sentence of 3 tokens.

(a) One unit: 5 numbers in, 1 number out

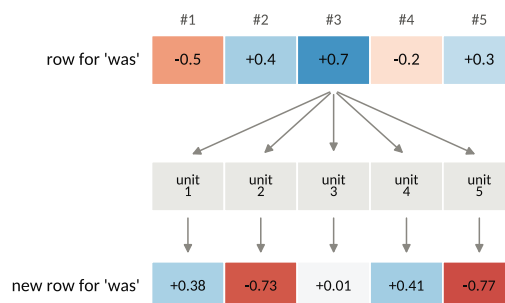
	#1	#2	#3	#4	#5
row for 'was'	-0.5	+0.4	+0.7	-0.2	+0.3
weights (knobs)	+0.0	-0.3	+0.9	+0.3	-0.5
products	-0.00	-0.12	+0.63	-0.06	-0.15

add them up, plus a bias: $+0.30 + +0.1 = +0.40$

squash to a sensible range: $\tanh(+0.40) = +0.38$

+0.38 ← the unit's single output number

(b) Five units side by side, each with its own weights, all reading the same row



Each unit contributes one number, so the row is rewritten at the same length; nothing collapses to a single total. GPT-2's bank has 768 units per layer.

(c) The same bank of units applied to every row of the table

	before						after				
	#1	#2	#3	#4	#5		#1	#2	#3	#4	#5
The	+0.8	-0.3	+0.1	+0.5	-0.6	bank of units	+0.62	+0.81	+0.41	+0.12	+0.75
food	+0.2	+0.9	-0.4	+0.3	+0.1		-0.45	+0.32	-0.03	+0.52	-0.78
was	-0.5	+0.4	+0.7	-0.2	+0.3		+0.38	-0.73	+0.01	+0.41	-0.77

3 rows in, 3 rows out: a table the same shape, with every number rewritten. In GPT-2 the rows are 768 long, so a 10-token sentence is a 10×768 table that becomes a new 10×768 table. One layer does this once.

Figure 9. A miniature of the model's basic operation (rows of 5 numbers instead of 768). (a) One *unit*: multiply each input number by its own weight, add up the products plus a bias, squash the total into a sensible range. The weights and bias are its knobs. Out comes one number. (b) A *bank* of five units, side by side: all five read the same input row, each with its own weights, so five numbers come out, a new row as long as the old one. (c) The same bank applied to every row of the table: three rows in, three rows out, every number rewritten. GPT-2 does this with 768-number rows and banks of 768 (and 3,072) units.

Read panel (a) first. A unit takes a row of numbers, multiplies each by its own weight, adds the products (plus a constant called a bias), and passes the total through a squashing function that keeps it in a sensible range. Out comes *one* number. If you have ever written a spreadsheet formula like $= 0.3 \times A2 - 1.1 \times B2 + 0.05 \times C2 + \dots$, you have built a unit; its coefficients are its knobs.

Now panel (b), which is where most people's mental picture goes wrong. One unit produces one number, so you might expect a row of 768 numbers to collapse into a single total. It does not, because the model does not use one unit; it uses a whole *bank* of them, side by side, each with its own set of weights, all reading the same input row at once. Five units produce five numbers; 768

units produce 768 numbers. The row is **rewritten, not summed up**: the output has exactly as many numbers as the input, each of them a different weighted combination of all the inputs. "Side by side" means nothing more than "many units, same input, one output number each."

Panel (c) puts the bank to work on a whole sentence. The table has one row per token. The same bank of units, with the same weights, is applied to every row separately: the row for *The* goes in and a new row for *The* comes out; the row for *food* goes in and a new row for *food* comes out; and so on. A sentence of ten tokens is a 10×768 table going in and a new 10×768 table coming out. Nothing gets shorter or narrower. At no point does the sentence become a single column of totals; the only place a row is ever boiled down to a handful of numbers is at the very end, when the final row is compared against every word in the vocabulary to produce the next-word probabilities of Figure 7.

Stack a few dozen of these operations one after another, each feeding the next, and you have a language model. This kind of arrangement, weighted sums stacked in stages, is what the term **neural network** refers to; the name is historical (the design was loosely inspired by brain cells) and carries no other meaning here. GPT-2's 124 million knobs are almost entirely the weights of its units, plus the $50,257 \times 768$ lookup table from Section 1.

There is one more ingredient, which lets a token's row look at the other rows in the sentence (it is called "attention," and it is why *bank* can be rewritten differently after *river* than after *account*). It, too, is built from weighted sums; the weights just get to depend on the words present. Section 3 shows it with real numbers.

2.6 After training, nothing moves

Training happens once, at enormous expense, and then stops. When you use the model, the knobs are frozen. Typing a prompt does not change any knob. What a prompt does is change the *numbers flowing through* the frozen machinery, the way a different set of inputs flows through the same spreadsheet formulas. Keep these two things separate in your mind:

- **The knobs (weights).** Fixed after training. This is the model's long-term knowledge.
- **The activations.** The rows of numbers sitting at each token position while the model reads a particular text. These change with every prompt. They are the model's short-term "state of mind" about the text in front of it.

How many rows are there, and which of them change? Suppose the model is given the four-token prompt "The soup was salty". It does not read the words one by one; it takes in all four tokens at once and creates four rows, one per token. Those four rows then go through the first stage of processing together, then the second, and so on (Section 3). When the model produces a fifth token and reads the extended text, a fifth row is created, and it turns out the first four rows come out exactly as before: the model is deliberately built so that a token's row can draw on the tokens *before* it but never on the tokens after it (we will see this in the mixing weights of Section 3), so adding

words at the end never changes the rows of earlier words. In an experiment for this article, appending two tokens to "The food was" changed the existing three rows by less than 0.000002. So the picture is: one row per token, all present at once, earlier rows untouched by later ones.

Steering and conceptors, the subjects of Sections 4 and 5, never touch the knobs. They reach in and edit the activations, while the model is reading.

3 Layers

Section 2 said the model rewrites the table of rows "a few dozen times, one stage after another." Each of those stages is a layer. This section shows what the layers do to the numbers, clears up the two most common confusions about them, and finds where in the sequence the meaning we care about is easiest to read.

3.1 Why stack simple steps

A single weighted sum can only do one simple thing: it draws a straight line (or its many-dimensional equivalent). That is why a line of best fit cannot follow a curve. The way to get complicated behavior out of simple parts is to *stack* them, so that the output of one stage becomes the input of the next. Each stage is simple; the sequence is not.

You have seen this structure many times. Grading an exam is a stack of stages: raw answers are turned into points per question; points are totaled into section scores; section scores are combined into a final grade. Each stage does a modest calculation, and the intermediate results (the section scores) are meaningful quantities in their own right, often more informative to a teacher than either the raw answers at the start or the single letter grade at the end. Hold on to that last observation; it will turn out to be true of language models too.

3.2 Rows are tokens; layers are versions

Here is the picture to hold in mind, and Figure 10 draws it with real numbers. The sentence is a **table**: one row per token, 768 columns. A **layer** takes that whole table in and puts a whole new table out, of exactly the same shape, with every number rewritten. GPT-2 has 12 layers, so there are 13 versions of the table: the starting version (rows copied straight from the lookup table of Section 1) and one version after each layer. Recent large models have around 60 to 120 layers, and correspondingly more versions.

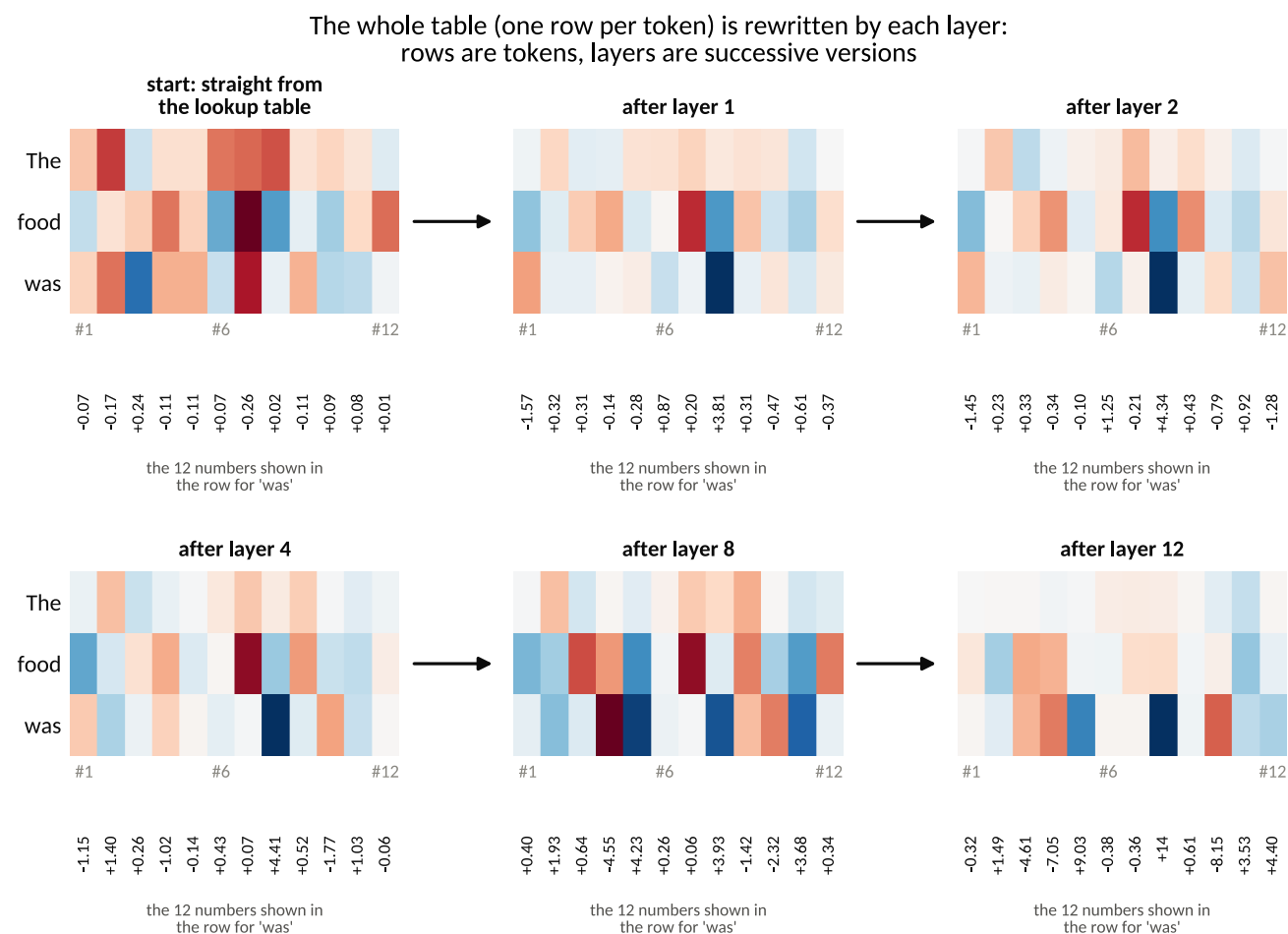


Figure 10. The prompt "The food was" as GPT-2 actually processes it: a three-row table (one row per token; only the first 12 of the 768 columns are shown, each panel colored on its own scale), in six of its thirteen versions. Every layer rewrites all three rows. Rows never turn into layers or vice versa: the rows are the tokens, and the layers are the successive versions of the same table. Below each panel are the actual numbers in the row for *was*.

Two confusions are worth heading off, because almost everyone falls into them.

Confusion 1: "the rows are the layers." They are not. Rows are tokens. A five-token prompt has five rows at every layer. The layers are not stacked *inside* the table like the layers of a cake; they are stacked *in time*, like successive edits of the same document. If it helps, think of thirteen photocopies of a spreadsheet, each a reworked version of the one before, all with the same rows and columns.

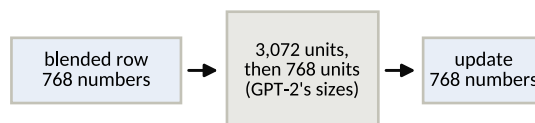
Confusion 2: "each row goes through all the layers on its own, then the next row does." Also not. All the rows go through layer 1 together, then all go through layer 2 together, and so on. This matters because, inside a layer, a row is allowed to look at the other rows (that is how context gets in). At the end, the model reads its next-word prediction off the *last* row of the final table, the row for the most recent token; that row has had twelve chances to absorb everything before it.

Figure 11 opens up one layer to show the three steps by which it turns one version of the table into the next.

(a) Step 1, mixing: how much each row borrows from the rows before it (layer 1)

	from 'The'	from 'food'	from 'was'
row 'The'	100%	–	–
row 'food'	81%	19%	–
row 'was'	69%	10%	20%

A row never looks ahead (the dashes), so adding later tokens never changes earlier rows. These blend weights are computed from the words present; they are not fixed knobs.

(b) Step 2, the bank of weighted sums: each blended row, separately

Exactly the operation of the earlier miniature figure, at full size. The same knobs are used for every row and every sentence; only the numbers flowing through differ.

(c) Step 3, write back: the update is added to the row (real numbers, row for 'was', first 8 of 768, layer 1)

row before	-0.07	-0.17	+0.24	-0.11	-0.11	+0.07	-0.26	+0.02
	+							
layer's update	-1.50	+0.48	+0.08	-0.03	-0.17	+0.80	+0.46	+3.79
row after	-1.57	+0.32	+0.31	-0.14	-0.28	+0.87	+0.20	+3.81

'Writes back' means: the new row is the old row plus the update; nothing is thrown away, so information laid down by early layers survives to the end.

Figure 11. What one layer does, with real numbers from GPT-2's layer 1 on the prompt "The food was." (a) Mixing: each row is blended with the rows before it. The percentages are the actual blend weights the model computed (averaged over its twelve "attention heads"); the row for *was* is made of 69% *The*, 10% *food* and 20% itself. The dashes show that a row never draws on tokens that come after it. (b) The bank of weighted sums from Figure 9, at full size, applied to each blended row on its own. (c) Write-back: the result is *added* to the old row, so nothing laid down earlier is lost. The numbers shown are the first 8 of the 768 in the row for *was*, before and after layer 1.

So, in plain words, a layer does three things to the table:

1. **Mixing.** Each row is replaced by a blend of itself and the rows before it (the row for *was* becomes mostly *The* plus a little *food* plus some of itself, in the example). The blend is still a row of 768 numbers. The blend weights are computed on the spot from the words present; they are not fixed knobs, which is what lets the same layer treat *bank* differently after *river* than after *account*. (The model actually runs twelve such blends in parallel, called "attention heads," each with its own weights; the figure shows their average.)
2. **The bank of weighted sums.** Each blended row goes through the bank of units from Figure 9, separately, producing an *update*: a row of 768 numbers.
3. **Write-back.** The update is added to the row. "Writes back" simply means that the layer's output is old row + update, for every row of the table. Because it is an addition rather than a replacement, information laid down by early layers survives to the end. (The research literature calls each row, as it is carried through the layers and added to at every one, the **residual**

stream; whenever you meet that phrase in a paper, or in this article's appendix and glossary, it means exactly this: the row of numbers for one token, as it passes through the layers.)

One consequence of the write-back is visible in the numbers of Figure 10: because every layer adds to the row, the numbers in it get larger with depth. Figure 12 shows the typical size of one number in the row for *was*: about 0.2 in the lookup-table version, about 2 after layer 1, and about 18 after layer 12, a roughly hundredfold growth. This is a mundane fact, but it matters in Section 4, because "how big a nudge is big" depends on which version of the table you nudge.

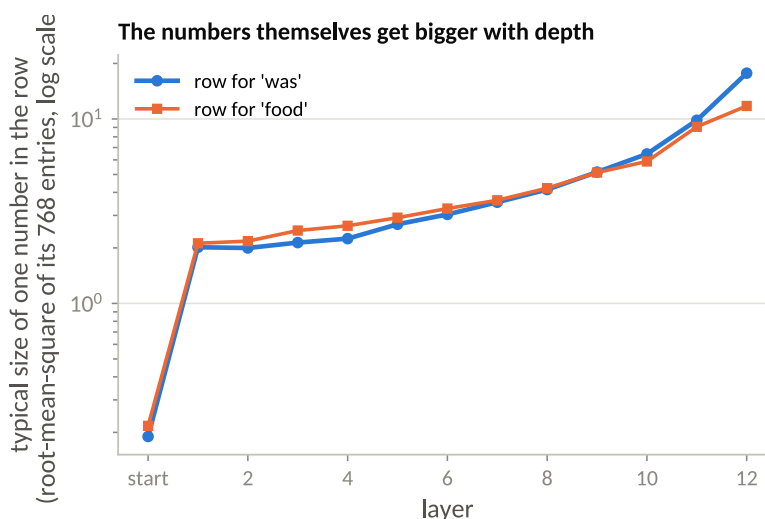


Figure 12. The typical size of a single number in a row (the root-mean-square of its 768 entries) at each depth, for two of the three rows in "The food was." The row always has 768 numbers; what grows is how large those numbers are.

What is the model doing as it rewrites the table? One crude but revealing trick is to take the row for *was* at an intermediate layer, pretend it is the final version, and ask what next-word prediction it would produce. (Researchers call this a "logit lens.") Doing that for the same prompt gives the table below.

Row for was taken from ...	Top guesses for the token after "The food was", if that row were the final one
the start (lookup table)	livest (68%), mathemat (14%), theless (10%) ... nonsense: the row still only says "the word was"
after layer 1	not (27%), also (15%), a (7%), definitely (3%)
after layer 4	not (30%), also (20%), definitely (14%), supposed (5%)
after layer 7	also (41%), not (11%), definitely (7%), supposed (4%), mostly (3%)
after layer 8	also (28%), mostly (7%), definitely (7%), not (6%), delicious (6%)
after layer 9	delicious (47%), cooked (13%), definitely (3%), served (2%)
after layer 11	delicious (75%), cooked (5%), tasty (5%), very (1%)
after layer 12 (the real output)	delicious (6%), good (5%), so (3%), very (3%), a (2%)

Read it top to bottom as a sequence of drafts. At the start the row knows only that the word is *was*, and the guesses are word-fragments. After one layer it has absorbed enough context to propose grammatical continuations (*not*, *also*). Through the middle layers it develops a view about *what kind of thing* comes next (an evaluation of food: *delicious* first appears at layer 8 and dominates by layer 9). The final layer spreads the probability over the specific plausible words (*delicious*, *good*, *so*, *very*). The topic ("food") and the tone ("positive") are present in the middle versions before the exact word is chosen at the end.

ANOTHER WAY TO THINK ABOUT IT

Successive drafts of a letter. The first draft has the facts in the wrong order; the middle drafts have the argument right but the wording rough; the final draft has the exact sentences. If you wanted to know what the letter is *about*, a middle draft would serve you as well as the final one, and it would not yet be cluttered with decisions about exact phrasing. The middle layers of a language model are its middle drafts.

3.3 Where does "positive versus negative" live?

The most important experiment in this article is a simple one, and it uses only the two-group comparison behind a t-test. Forty-eight pairs of short sentences were written for this article, each pair identical in topic and structure but opposite in sentiment:

Positive	Negative
The movie was wonderful and I loved every minute.	The movie was awful and I hated every minute.
I feel great today.	I feel miserable today.

Positive	Negative
The customer service was helpful and quick.	The customer service was useless and slow.
This laptop is fast and reliable.	This laptop is slow and unreliable.
... and 44 more pairs on hotels, weather, concerts, teachers, coffee, gardens, and so on.	

Each sentence was fed to GPT-2. A sentence of ten tokens gives a table of ten rows at every layer, and for this experiment we keep just one of them: the row at the sentence's **final period**. From here on that row is called the sentence's **summary row**, and there is one summary row per sentence per layer. Why the period? Because at the start, the period's row is identical for every sentence (it is just the token "." from the lookup table). Whatever the period's row knows about the sentence at later layers, it must have *picked up from the layers*, by mixing with the rows before it. That makes it a clean probe of what the layers write.

Now the comparison. At each layer we have 48 summary rows from positive sentences and 48 from negative ones, each row 768 numbers long. Ask the question a t-test asks: *how far apart are the two groups, relative to their spread*? Figure 13 shows the arithmetic with the literal numbers, and it is worth walking through, because the same three steps recur throughout the rest of the article.

(a) Summary rows of six sentences (layer 8, first 8 of 768 numbers), the two group averages, and their difference: the arrow

	#1	#2	#3	#4	#5	#6	#7	#8	
The movie was wonderful and I loved every minute.	+1.52	-2.66	+0.75	-0.69	+0.25	-0.11	+9.39	-0.55	... (760 more)
I feel great today.	+2.53	-3.92	-0.97	+1.03	+2.34	+0.80	+8.48	-2.07	... (760 more)
I love my new job.	+1.90	-1.99	+0.03	-0.46	+2.69	-0.01	+8.08	-1.50	... (760 more)
average of all 48 positive rows	+1.89	-3.10	-0.76	-0.81	+0.69	+0.46	+8.96	+0.32	... (760 more)
The movie was awful and I hated every minute.	+1.44	-1.45	+0.67	+0.46	-0.41	-2.67	+9.00	-1.01	... (760 more)
I feel miserable today.	+3.41	-2.41	-1.58	+1.96	+0.83	-0.63	+7.97	-0.02	... (760 more)
I hate my new job.	+2.73	-1.02	-0.54	+0.88	+1.84	-2.39	+7.88	-2.77	... (760 more)
average of all 48 negative rows	+2.04	-2.62	-0.73	+0.47	-0.11	-0.92	+8.59	+0.07	... (760 more)
the arrow = positive average – negative average	-0.14	-0.47	-0.03	-1.29	+0.80	+1.39	+0.37	+0.25	... (760 more)

(b) Scoring one sentence: multiply its row by the arrow, number by number, and add up

"The movie was wonderful and I loved every minute."

row × arrow, first 8 of 768 products:

-0.01 +0.06 -0.00 +0.04 +0.01 -0.01 +0.17 -0.01 ... + 760 more

total = score on the arrow = +14.2

(its negative twin "The movie was awful and I hated every minute." scores -8.1)

The arrow is scaled to length 1 first, and rows are measured from the midpoint between the two averages, so a score of 0 means 'halfway between the groups'.

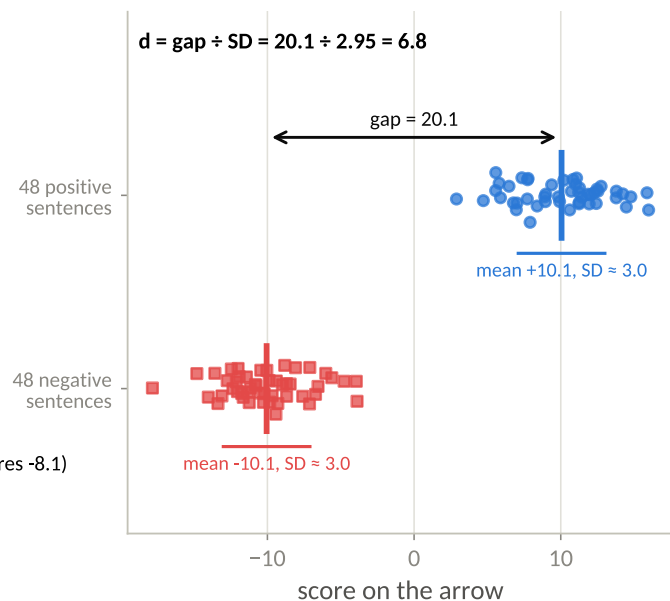
(c) All 96 scores: the gap between the groups, in units of their spread

Figure 13. The arithmetic behind Figure 14, with real layer-8 numbers. (a) Six summary rows (first 8 of 768 columns), the average row of each group, and the arrow: positive average minus negative average, column by column. (b) To score one sentence, multiply its summary row by the arrow, column by column, and add the 768 products; the result is one number, how far along the arrow the sentence sits. (c) Do that for all 96 sentences: the two groups' scores, their means, their spread, and the ratio $d = \text{gap} \div \text{spread}$.

- Average.** Add up the 48 positive summary rows, column by column, and divide by 48: one average row. Do the same for the 48 negative rows. (Panel (a), the two bold rows.)
- Subtract.** Positive average minus negative average, column by column. The result is a row of 768 numbers, the **arrow**. It is exactly what the numerator of a two-sample t-test would be, computed for each of the 768 columns and collected into a row. (Panel (a), bottom row.)
- Project.** To give one sentence a single score, multiply its summary row by the arrow, column by column, and add up all 768 products. (Panel (b).) Geometrically this measures how far along the

arrow the sentence sits; a sentence whose row points the same way as the arrow gets a large positive score, one pointing the opposite way a large negative score.

With one score per sentence, the rest is the familiar two-group comparison: take the gap between the two groups' mean scores and divide by their pooled standard deviation. That ratio is Cohen's d , the standardized mean difference. At layer 8 the positive sentences average +10.1, the negatives -10.1, the pooled spread is about 3.0, and $d = 20.1 \div 2.95 = 6.8$ (panel (c)). Figure 14(a) repeats this calculation at every layer. One caution: the arrow was computed from the same 96 sentences it is then used to score, which flatters it a little. The dashed line in the figure redoes it properly, computing the arrow from 36 of the pairs and scoring only the 12 held-out pairs (averaged over twenty random splits); it runs just below the solid line (6.7 against 6.9 at layer 8), so the flattery is small, and panel (b) is held out throughout.

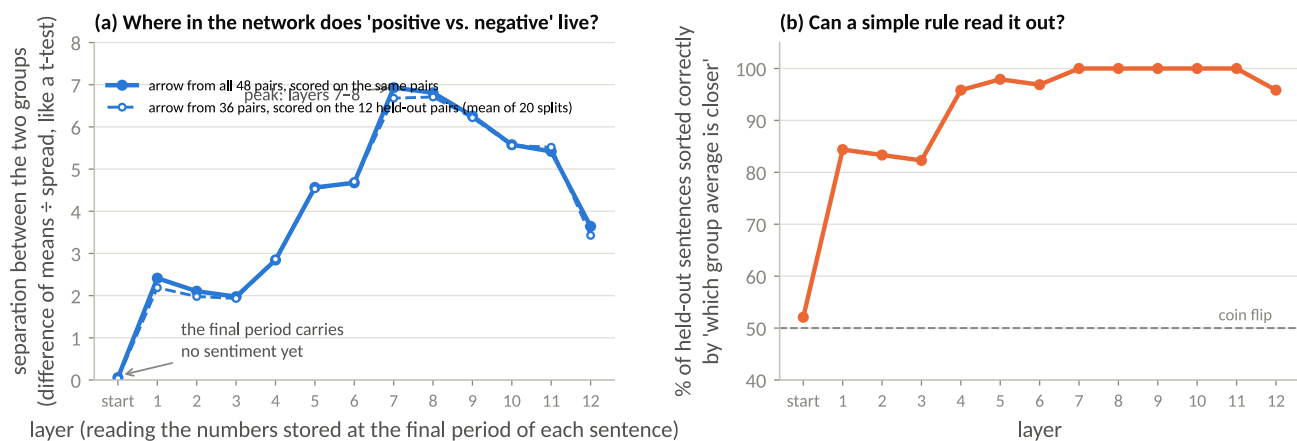


Figure 14. (a) Standardized separation (Cohen's d) between 48 positive and 48 negative sentences, using each sentence's summary row (the row at its final period), at each depth of GPT-2. It is essentially zero at the start (the period knows nothing yet), climbs through the layers, peaks at layers 7-8 at nearly 7 standard deviations, and falls off in the last layers. Solid line: arrow computed from all 48 pairs and applied to the same sentences. Dashed line: arrow computed from 36 pairs and applied to the 12 held-out pairs, averaged over 20 random splits. (b) The same thing as a classification test: for each held-out sentence, is its summary row closer to the positive average or the negative average? By layer 7 every one of the 96 is sorted correctly.

Three features of the curve deserve attention.

1. **It starts at zero.** At the start, straight from the lookup table, the period's row carries no sentiment, as promised; the two groups are indistinguishable ($d = 0.06$, and the sort-by-nearest-average test is at 52%, a coin flip). The tokens have no "meaning" in the model until they have run through the layers.
2. **It peaks in the middle.** By layers 7 and 8 the two groups are almost seven standard deviations apart. For comparison, a "large" effect in a psychology experiment is $d = 0.8$. The model has written "this sentence was positive/negative" into the summary row so firmly that a rule as crude as "which average is closer?" sorts 100% of held-out sentences correctly.
3. **It drops at the end.** After layer 12 the separation has fallen to $d = 3.6$. The last layers are busy converting meaning into a specific next-token guess, and in doing so they partly overwrite the

tidy summary that the middle layers built.

Figure 15 shows the same story as pictures. At each of four depths, the 96 sentences are plotted with the horizontal axis being the arrow (computed from 36 of the pairs) and the vertical axis being the largest other direction of variation; the 12 held-out pairs are drawn hollow. After layer 1 the groups overlap; after layer 8 they are cleanly split, held-out sentences included; after layer 12 they have begun to merge again.

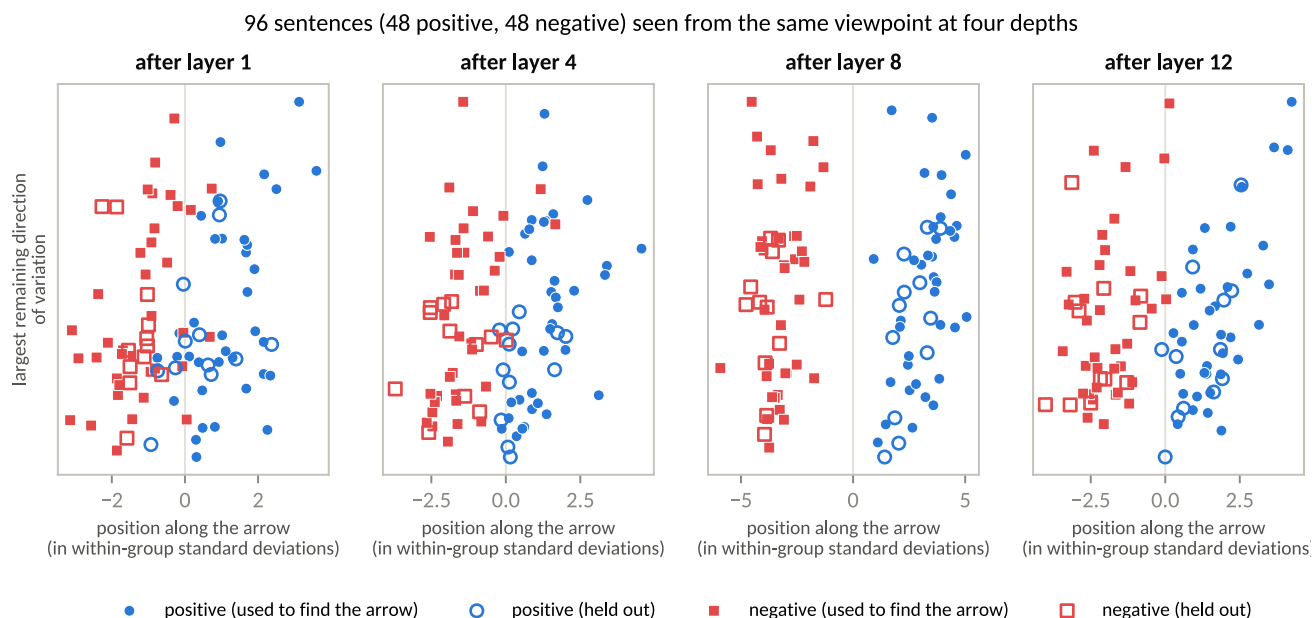


Figure 15. The 96 sentences at four depths, each sentence one dot placed by its summary row, viewed along the arrow from the negative average to the positive average (horizontal) and the largest remaining direction of variation (vertical). The arrow was computed from 36 pairs; the 12 hollow-marked pairs were held out and are placed using the same arrow. The middle layer separates them best.

3.4 This is a general finding, not a quirk of one experiment

The "middle layers hold the most readable meaning" pattern shows up throughout the research literature, on models far larger than GPT-2:

- Skean and colleagues (2025) tested every layer of several models as a source of sentence encodings on 32 standard tasks and found that an intermediate layer beat the final layer in nearly every case, by up to 16 percentage points, in models of several different designs. Their explanation is that the middle layers keep the essential information while discarding low-level clutter, and the last layer is "overly specialized" to the next-word guess.
- Zhu, Subramanian and colleagues (2026) did a version of our experiment with three groups instead of two. They fed descriptions of depression symptoms to a 62-layer model, grouped into mood symptoms, bodily symptoms and thoughts of death, and asked at every layer how far apart the three groups' summary rows sat relative to their spread (a three-group version of Cohen's d ; the groups need not be opposites, any more than three treatment arms in a trial

need to be). Three different distance measures agreed on the same answer: the groups were most separable at layer 21, about a third of the way in.

- Ravenda and colleagues (2025), steering a 32-layer model, simply chose the middle layer, citing exactly this body of evidence.
- Triantafyllopoulos and colleagues (in preparation), the central source on conceptors, read their numbers from the midpoint layer of each of five models (layer 13 of 26, 21 of 42, 18 of 36, 16 of 32, 16 of 32).
- Gurnee, Sofroniew, Lindsey and colleagues (2026) report that the concepts a model is "poised to talk about" carry coherent content only in an intermediate band of layers, with the last layers tied to the imminent output.

WHAT TO TAKE FROM THIS SECTION

Rows are tokens; layers are successive versions of the whole table. Early versions: a row mostly still says which word this is. Middle versions: the row summarizes what the sentence so far means, including things like topic, tone, and (as Marks and Tegmark showed for factual statements) whether it is true. Late versions: the row is reshaped into a specific next-word guess. **If you want to read a model's "opinion" about a text, or change it, the middle layers are where to look.** Everything in Sections 4 and 5 happens at layer 8 of GPT-2's 12, using each sentence's summary row.

4 Reading and steering

Section 3 found that, at layer 8, the summary rows of positive and negative sentences sit far apart along one arrow. This section uses that arrow twice: first to *read* the model's assessment of new text, then to *change* it. The second use is what "steering" means.

4.1 The arrow, stated plainly

You have now met the arrow several times, so here is its definition in one place. Take a set of examples of one kind (positive sentences) and a set of the opposite kind (negative sentences). Run them through the model. At a chosen layer, take each sentence's summary row (here, the row at its final period). Average the rows within each group. Subtract:

$$\text{arrow} = (\text{average summary row of the positive sentences}) - (\text{average summary row of the negative sentences})$$

It is a row of 768 numbers (Figure 13(a), bottom), and it is exactly the numerator of a two-sample t-test, computed separately for each of the 768 columns and collected into a row. In the research literature it is called a "difference-of-means direction," a "mass-mean probe," or, when used for

steering, a "steering vector" or "contrastive activation addition." All of these are the same object. Note that it always takes *two* groups to make one: the arrow points from the second group's average toward the first group's average, so it has a direction; averaging the two groups together would give you a center point, not an arrow.

4.2 Reading: a ruler for sentiment

Given the arrow, any new text can be scored: run it through the model, take its layer-8 summary row, and ask how far along the arrow it sits (measured from the midpoint between the two group averages; this is step 3 of Figure 13). Figure 16 does this for sentences the arrow never saw: twelve held-out positive sentences, their twelve negative twins, sixteen neutral sentences with no opinion in them, and ten brand-new sentences written afterwards specifically to try to trip it up.

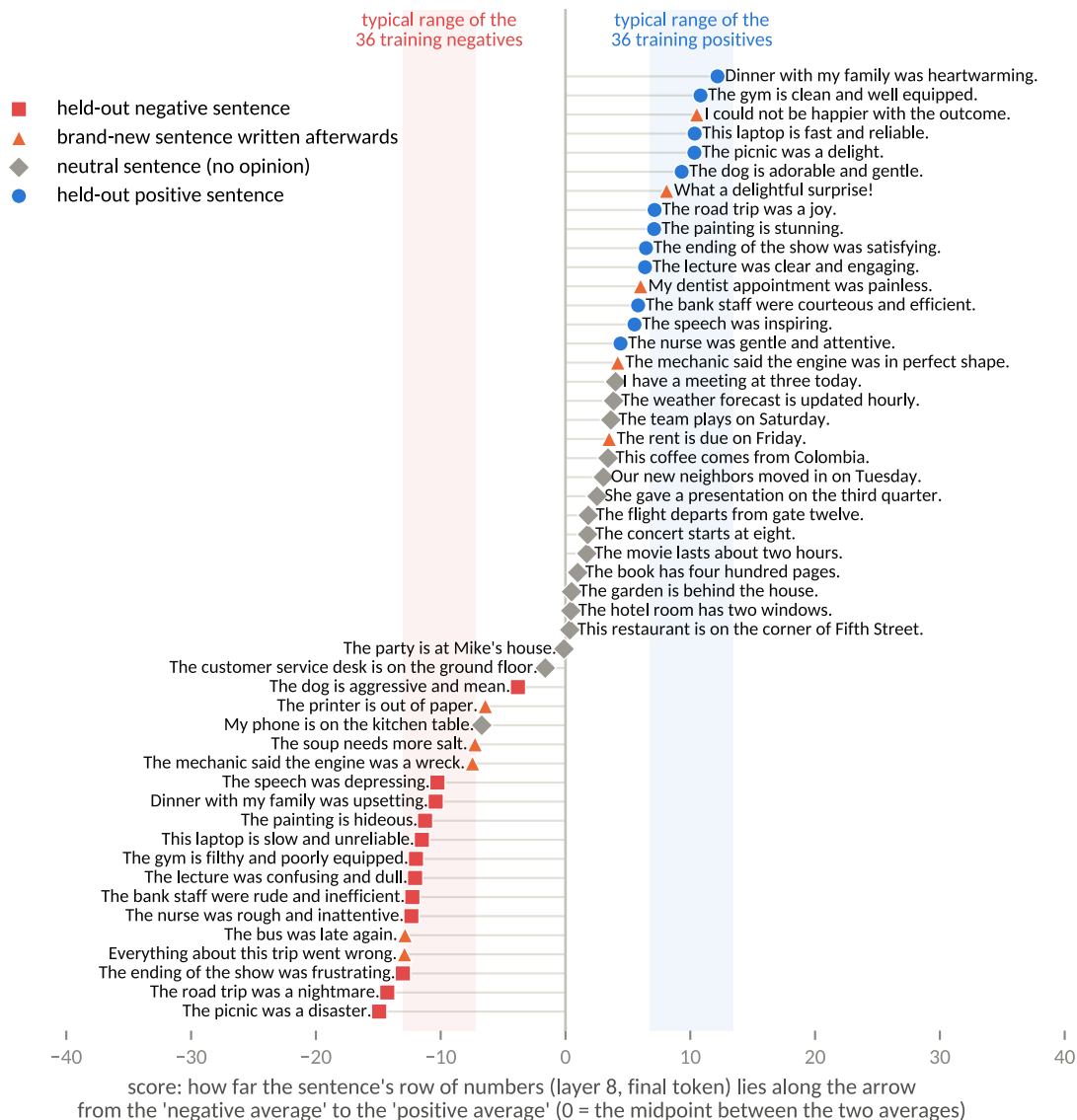


Figure 16. A sentiment ruler. The arrow was computed from 36 sentence pairs; every sentence shown here was held out from that computation. Positive sentences land on the right, negative on the left, neutral factual sentences cluster near zero, and brand-new sentences land roughly where a human would put them ("The bus was late again" and "Everything about this trip went wrong" are far to the left; "I could not be happier with the outcome" near the top). It is not perfect: "The rent is due on Friday" reads as mildly positive, and "The mechanic said the engine was in perfect shape" as only mildly positive.

Nothing in this ruler was trained. No classifier was fitted. Two averages were subtracted, and the resulting arrow generalizes to sentences about mechanics, dentists and rent that have nothing to do with the movies, hotels and coffee it was built from. That generalization is the evidence that the arrow captures something like "positive versus negative" itself, rather than "these particular 72 sentences."

The same recipe reads other things. Marks and Tegmark (2024) built datasets of simple true and false statements ("The city of Paris is in France," "The city of Hanoi is in Poland," "Sixty-one is larger than seventy-four") and computed true-minus-false arrows inside the Llama-2 models (a family of

openly released models from Meta, in sizes from 7 to 70 billion knobs, some sixty to six hundred times larger than GPT-2). In the two larger models, an arrow built only from number-comparison statements sorted true from false statements about Spanish vocabulary with better than 95% accuracy: same arrow, unrelated topic.

A concept does not have to come in opposite pairs to be read this way. Zhu, Subramanian and colleagues (2026) worked with three groups of depression-symptom descriptions taken from the wording of clinical questionnaires: mood symptoms ("feels sad most of the day"), bodily symptoms ("wakes in the early hours and cannot fall asleep again") and thoughts of death. None of the three is the opposite of another. So instead of one arrow between two groups, they used the three group averages as three *landmarks* in the number-space (a row in their model has 5,376 numbers rather than 768; nothing else changes), and scored a new passage by how far it lies toward each landmark, adjusting for the fact that the three landmarks are not at right angles to one another. When they scored passages from a memoir and a clinical handbook, passages that clinicians had labeled "bodily" scored highest on the bodily landmark, and likewise for the other two. They did also build one genuine arrow, from all three symptom groups together toward a group of cheerful passages, and it separated depressive from happy text on held-out data. Same arithmetic, arrows or landmarks as the concept requires.

ANOTHER WAY TO THINK ABOUT IT

The arrow is like a formula you discovered for a score, the way body-mass index is a formula computed from height and weight. Nobody designed people to have a BMI; it is a combination of measurements that happens to track something useful. Likewise nobody designed the model to have a "sentiment column," but a particular combination of its 768 columns tracks sentiment, and once you know the combination you can compute it for any text.

4.3 Steering: pushing the numbers

Now the leap. If the model's later layers read sentiment off the layer-8 row, what happens if we *write* to that row? Concretely: let the model process a prompt as usual up through layer 8; then, before layer 9 runs, add the arrow (multiplied by a chosen strength) to every row of the table; then let layers 9 through 12 continue as if nothing had happened. None of the model's knobs change. Only the activations, the numbers passing through, are edited. This is **steering**.

$$\text{edited row at layer 8} = \text{original row} + \text{strength} \times \text{arrow} \quad (\text{applied to every row of the table})$$

Figure 17 shows the result for the prompt "We tried the new restaurant on Main Street. The food was really ____". With no steering, the model gives 74% to *good* and a further 15% to other positive words; it gives essentially nothing to negative words. Add the arrow with strength -2 and the

picture flips: 65% *bad*, then *awful*, *terrible*, *poor*; positive words fall to about 1%. Add it with strength +1 and the positive words become more varied and enthusiastic (*great* rises from 5% to 11%). The strength is a dial, and the model's mood turns with it.

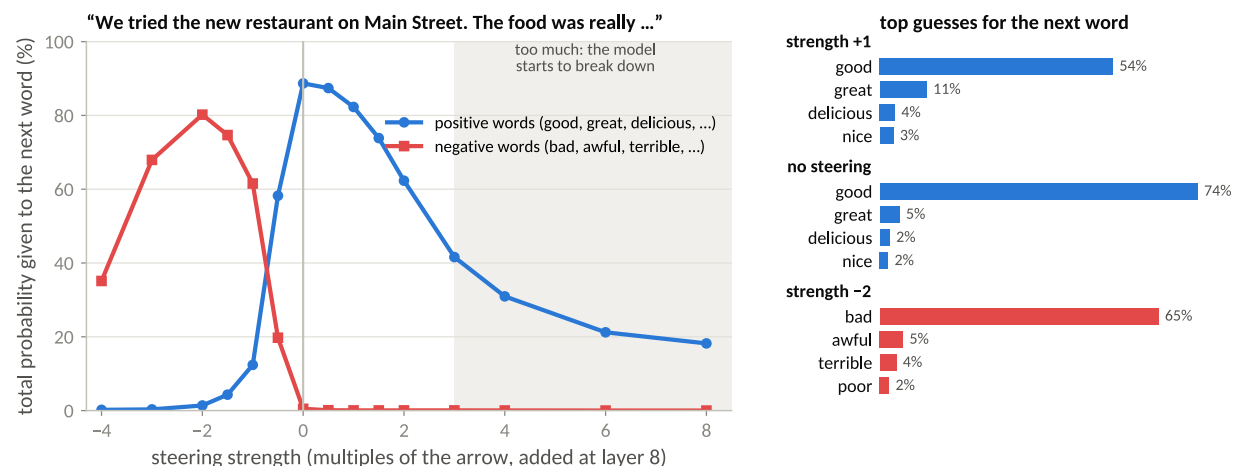


Figure 17. Steering GPT-2's next-word prediction by adding the sentiment arrow at layer 8. Left: the total probability given to positive words (blue) and negative words (red) as the steering strength is dialed from -4 to +8. Right: the top four guesses at three settings. Beyond about +3 the model starts to break down: the probability of every sensible word falls, and the output becomes incoherent.

The edit changes not just the next word but everything the model goes on to write, because each generated token is fed back in and processed through the same edited layer. The table below shows full continuations (20 tokens, always taking the most likely token) at several strengths, for three prompts.

Strength	"We tried the new restaurant on Main Street. The food was ..."
-2	terrible. I tried to get a sandwich. I tried to get a sandwich. I tried to get
-1	terrible. I was so angry. I was like, 'I'm going to go to the restaurant
0	good, but the service was not great. I would recommend this place to anyone who is looking for
+1	great and the service was great. The staff was friendly and the food was great. The staff is
+2	great and the service was great. The staff is always friendly and the prices are always great.
+5	very much a complete and very very special.<end>The 2017-2018 and 2016-2017 and 2016
+10	great and and and and and and and and and and and and and and and and and

Strength	"My new phone arrived yesterday. So far it ..."
-2	's been a mess. I'm going to try to get it back. I'm going to try
-1	's been a mess. I'm not sure what to do. I'm going to try to get
0	's been working fine. I'm not sure if it's a new phone or a new model.
+1	's been great. I'm very happy with the phone. I'm very happy with the price.
+2	is a great experience. I am very happy with the price and the quality of the product.
+5	is a very happy and very very very very very very very very very very very very very very very very

Strength	"I watched the movie last night. Overall, the movie was ..."
-2	awful. I was trying to figure out what to do. I was trying to figure out what to
-1	a lot of fun. I was really disappointed. I was really disappointed. I was really disappointed.
0	a great experience. I'm not sure if I would have liked it if I had been a little
+1	a great experience. I'm very happy with the way the characters are portrayed. I'm very happy
+2	a great experience. The movie is a great experience. The movie is a great

Notice that the arrow was built from sentences about movies, hotels, coffee and gardens, yet it steers a paragraph about a phone. Notice also the repetitiveness: GPT-2 small is prone to loops even unsteered ("I was really disappointed. I was really disappointed."), and steering does not cure that. And notice what happens at strength +5 and +10: the text degenerates into "very very very" and "and and and." Pushing a row of numbers far outside the region the model has ever seen does not produce a very, very positive review; it produces nonsense. Steering is a dial with a useful range, not a slider to infinity.

ANOTHER WAY TO THINK ABOUT IT

Prompting is like leaving a note for a colleague asking for an upbeat report. Steering is like adjusting the colleague's mood directly, before they sit down to write. The note may be misread, ignored, or overruled by the next note; the mood shapes every sentence from the inside. And, as with people, there is such a thing as too much: a colleague who is *only* mood and no longer paying attention to the facts produces gibberish.

4.4 Steering is not prompting

It is worth being precise about the difference, because Section 6 turns on it.

- **A prompt is more text.** "Please write a positive review." The model reads it the way it reads everything, through the lookup table and all twelve layers. Whatever effect it has, it has by way of the model's ordinary reading process, and the model's ordinary reading process can misunderstand it, weigh it lightly, forget it after a few hundred words, or be talked out of it by later text.
- **Steering is an edit to the activations.** It does not pass through the reading process. It is applied at a chosen layer, at every step of generation, with a chosen strength, regardless of what the text says. It cannot be "misread," because it was never read.

Neither one changes the model's knobs. Both leave the model exactly as it was for the next user. But they act at different places: one at the input, one in the middle of the computation.

4.5 A real example: steering a model to correct a cautious bias

Ravenda and colleagues (2025) faced a practical problem. They wanted a small model (Llama-3.1-8B, an 8-billion-knob open model) to judge whether Reddit posts were relevant to particular items of a standard depression questionnaire (the BDI-II). Small models are attractive here for privacy and cost. But the small model had a systematic **cautious bias**: it flagged far too many posts as relevant. The authors describe this as a preference for avoiding false negatives in a sensitive clinical setting; where that preference came from, they do not say, and neither will this article.

So they built an arrow. For each questionnaire item, they collected the model's middle-layer rows when it answered correctly and when it answered incorrectly, and took the difference of the averages. Then they steered with it, choosing the strength by a calibration rule on validation data. Figure 18 shows their published results on the test set.

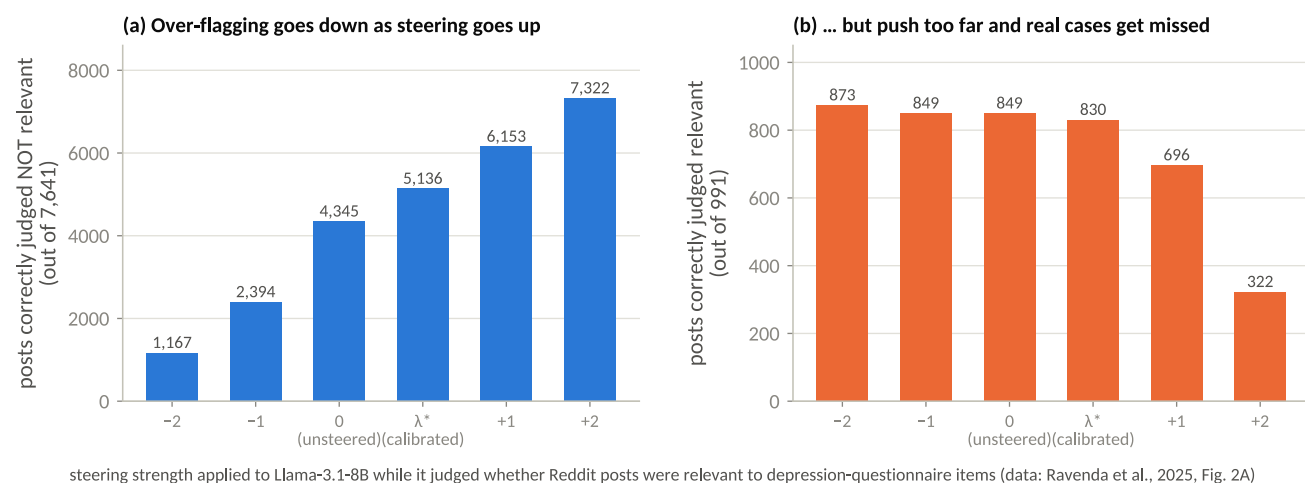


Figure 18. Redrawn from the counts reported by Ravenda et al. (2025, their Figure 2A). An 8-billion-knob model judged 8,632 Reddit posts for relevance to depression-questionnaire items, with a steering arrow added at its middle layer at various strengths. (a) The number of irrelevant posts correctly left unflagged rises steadily with steering strength: from 4,345 with no steering to 5,136 at the calibrated strength. (b) The number of genuinely relevant posts correctly flagged is nearly unchanged at the calibrated strength (849 to 830), but collapses if the steering is pushed further (696, then 322). The calibrated strength buys an 18% improvement on the large "irrelevant" class at a 2% cost on the small, clinically important "relevant" class.

Two lessons. First, the correction was applied inside the computation rather than by rewording the question: the authors did not report trying to fix the bias with a different prompt, so the paper does not show that a prompt would have failed; what it shows is that an edit to the middle-layer numbers, built from two averages, was enough. Second, the strength matters, and there is a real trade-off; the same figure shows that over-steering destroys the sensitivity that made the model useful. The authors went on to use the steered model to fill in whole questionnaires from users' posting histories, and reported that the steered 8-billion-knob model was competitive with much larger commercial models on that task, ahead of some and behind others.

A second published example points the other way, toward reading rather than fixing. Abootorabi and colleagues (2026) built steering arrows for twenty human values (benevolence, achievement, tradition, self-direction, ...) in several models, and asked whether the *angles between the arrows* matched the well-known psychological "circle of values," in which compatible values sit near each other and conflicting ones sit opposite. For arrows built the simple difference-of-means way, they did: the correlation between the model's arrow geometry and the psychological theory reached 0.5, and steering a model toward one value nudged it toward compatible values and away from opposing ones, as the theory predicts. The arrows are not arbitrary; they reflect a structure the model has learned about people.

4.6 The limits of a single arrow

The arrow is a wonderful tool, and it will not go away in Section 5; conceptors are built on top of the same material. But it has three limitations, and naming them now will make conceptors feel inevitable rather than arbitrary.

1. **Everyone gets the same push.** Adding the arrow moves every row by the same amount in the same direction, whether the sentence was already strongly positive, weakly positive, or negative. A review that was already glowing gets pushed toward "very very very"; a review that was scathing gets pushed only part-way toward neutral. A push that does not depend on where you started is a blunt instrument.
2. **It is one direction.** A concept like "negative sentiment" is not a single line. There is anger, disappointment, boredom, disgust; there is the difference between a bad meal and a bad movie. In the 768-number space, "the set of negative sentences" is a *cloud with a shape*, and the arrow only records where its center is relative to the positive cloud's center. It records nothing about the cloud's spread.
3. **It is estimated from averages, so it is noisy when examples are few.** With two or three examples per side, the arrow is mostly the idiosyncrasies of those examples. Nothing in the arrow itself knows which of its 768 components are "sentiment-like" and which are noise.

A conceptor addresses all three. It keeps the arrow's information about where the concept sits, adds information about the concept's shape, uses that shape to act differently on different inputs, and uses it to clean up noisy estimates. That is the next section.

5 Conceptors

A steering arrow records one direction: from the average of one group of examples to the average of its opposite. A conceptor records, in addition, the *shape* of the region the examples occupy, and uses that shape as a filter. This section builds a conceptor from the same 96 sentences as before, one step at a time, keeps returning to actual sentences to show what each step means, and shows what a conceptor does that the arrow cannot.

5.1 A concept has a shape

Go back to the 96 summary rows at layer 8 (one row per sentence: the row at its final period), and look at them from a new angle. Figure 19 plots them with the arrow as the horizontal axis (as in Figure 15) and, vertically, the direction along which the sentences vary most *apart from* the arrow. Each dot is one sentence's summary row, placed by two of its coordinates. Ellipses are drawn around each group and around all 96 together.

A concept has a shape, not just a direction (real layer-8 numbers, 2 of 768 dimensions shown)

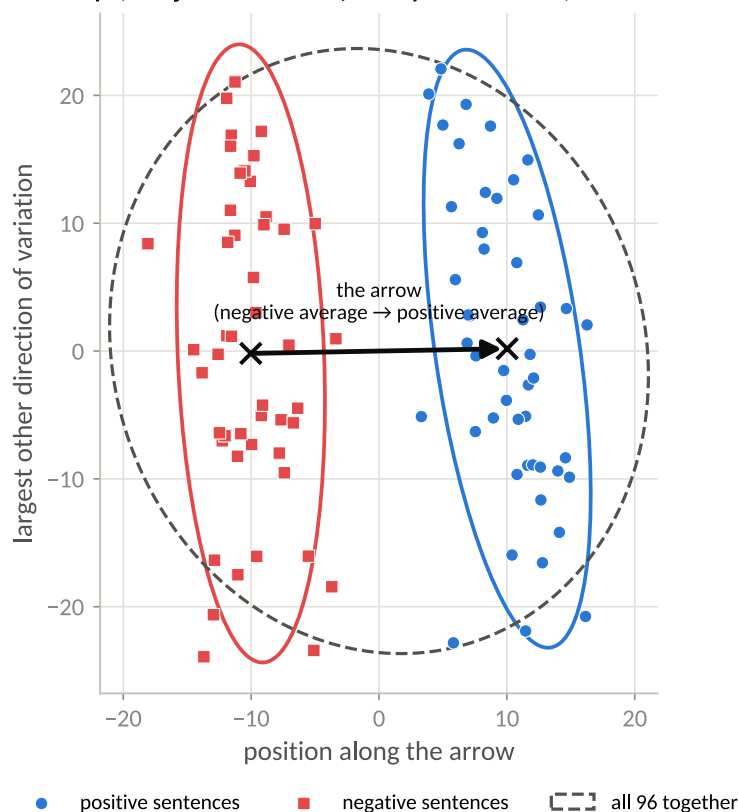


Figure 19. The 96 summary rows at layer 8 (one dot per sentence), viewed along the arrow (horizontal) and the largest other direction of variation (vertical). Each group is a cloud with a shape, not a point; and the two clouds are *taller* than they are wide. The arrow describes only where the two centers (x) are. The dashed ellipse is the shape of all 96 pooled together.

Two observations, both of which will matter.

First, **each group is a cloud with a shape**. The positive sentences do not all sit at one point; they spread out, and the spread has structure: it is much larger in some directions than in others. Some of that spread is meaningful variation within the concept (a glowing review of a concert is not the same thing as a glowing review of a laptop); some is topic and wording that has nothing to do with sentiment.

Second, and this surprises most people, **the biggest direction of variation among all 96 sentences is not the arrow**. The standard deviation of the pooled cloud along the arrow is 10.5; along the vertical direction in the figure it is 11.8. To say this precisely we need a ranking that the next subsection introduces properly: take the pooled cloud and list its directions of spread from the one it stretches along most to the one it stretches along least (a list we will call direction 1, direction 2, and so on). The arrow lines up best with *direction 3* on that list, not direction 1. Directions 1 and 2 are mixtures of sentence shape and sentiment that both groups share. If you tried to discover "sentiment" by looking for the biggest direction of variation in the data (the way principal component analysis would), you would find something else first. This is why the arrow is computed from the group averages, and why a conceptor, which captures the whole shape, needs the labels too.

ANOTHER WAY TO THINK ABOUT IT

Picture dog breeds plotted by height and weight. "Greyhound" is not a point on that plot; it is an elongated cloud (greyhounds vary a lot in height and rather little in weight for their height). To describe what a greyhound is, you need the cloud's *shape*: which direction it stretches along and which it does not. A single arrow from "the average non-greyhound" to "the average greyhound" would miss that entirely.

5.2 From spread-per-direction to gain-per-direction

Here is the recipe for building a conceptor from a set of example summary rows. One word first, because it is used throughout: a **gain** is a multiplier applied to a signal, the way the volume knob on an amplifier multiplies a sound. Gain 1 means "pass it through unchanged," gain 0 means "silence it," gain 0.3 means "turn it down to 30%." It is a neutral word; a high gain is not "good," it just means "kept." A conceptor is a set of gains, one per direction, and for that reason this article will often call it a **filter**: something you pass a row through, that keeps some of it and turns down the rest.

1. **Center the examples.** Subtract the average row from every example, so the cloud sits around zero.
2. **Find the directions of spread and how much spread there is along each.** Any cloud of points can be described by a set of perpendicular directions, ordered from "the direction the cloud stretches along most" to "the direction it stretches along least," with a variance attached to each. (For a two-dimensional cloud, these are the long and short axes of the ellipse that best

fits it; in 768 dimensions there are 768 such directions. This is the same idea as principal component analysis.) Figure 20(a) shows the variances for our 96 sentences: the biggest direction has variance 142, the tenth about 27, the fortieth about 4, and the ninety-fifth 0.26. With only 96 examples, a cloud can only spread in at most 95 directions, so the remaining 673 of the 768 directions have variance exactly zero.

3. **Turn each variance into a gain between 0 and 1.** This is the one formula in the article, and it is a very tame one:

$$\text{gain along a direction} = \text{spread} \div (\text{spread} + \lambda)$$

where λ ("lambda") is a single positive number you choose (more on it below). If the spread is much bigger than λ , the gain is close to 1. If the spread is much smaller than λ , the gain is close to 0. If the spread is exactly zero, the gain is exactly zero. Figure 20(b) shows the curve and the actual gains for our 95 directions at $\lambda = 1.4$: the top direction gets 0.99, the fortieth gets 0.73, the sixtieth 0.50, the ninety-fifth 0.16, and the 673 directions with no spread get 0.

4. **The conceptor is the list of directions together with their gains.** That is all it is. Written as a formula in the research literature, the conceptor is $C = R (R + \lambda I)^{-1}$, where R is the covariance matrix of the centered examples; but that formula does nothing more than steps 2 and 3 at once, and you can read it as "the covariance, with each direction's variance replaced by its gain."

What do these directions actually correspond to, in sentences? This is the natural question, and the honest answer is: each direction is an *axis* with two ends, not a category. "Positive" and "negative" are not two different directions; they are the two ends of one direction. To see which is which, take each direction in turn and look at which sentences sit at its two extremes. For our conceptor ("arrow overlap") is how much the direction lines up with the sentiment arrow: 1 would mean identical, 0 unrelated):

Direction	Sentences at one end	Sentences at the other end
3 variance 106 arrow overlap 0.80	My phone is completely broken. · This laptop is slow and unreliable. · The car runs terribly. · I hate my new job.	The picnic was a delight. · The concert last night was amazing. · The road trip was a joy. · The meeting was productive and pleasant.
4 variance 71 arrow overlap 0.01	I hate my new job. · I feel great today. · I love my new job. · I feel miserable today.	The book is brilliant from start to finish. · The book is boring from start to finish. · The painting is stunning. · His new song is dull and annoying.
5 variance 67 arrow overlap 0.08	The teacher was patient and encouraging. · The doctor was careless and dismissive. · She gave an excellent presentation. · The doctor was thorough and reassuring.	The garden looks beautiful this spring. · The garden looks dreadful this spring. · The weather this weekend was lovely. · The apartment is dark and cramped.

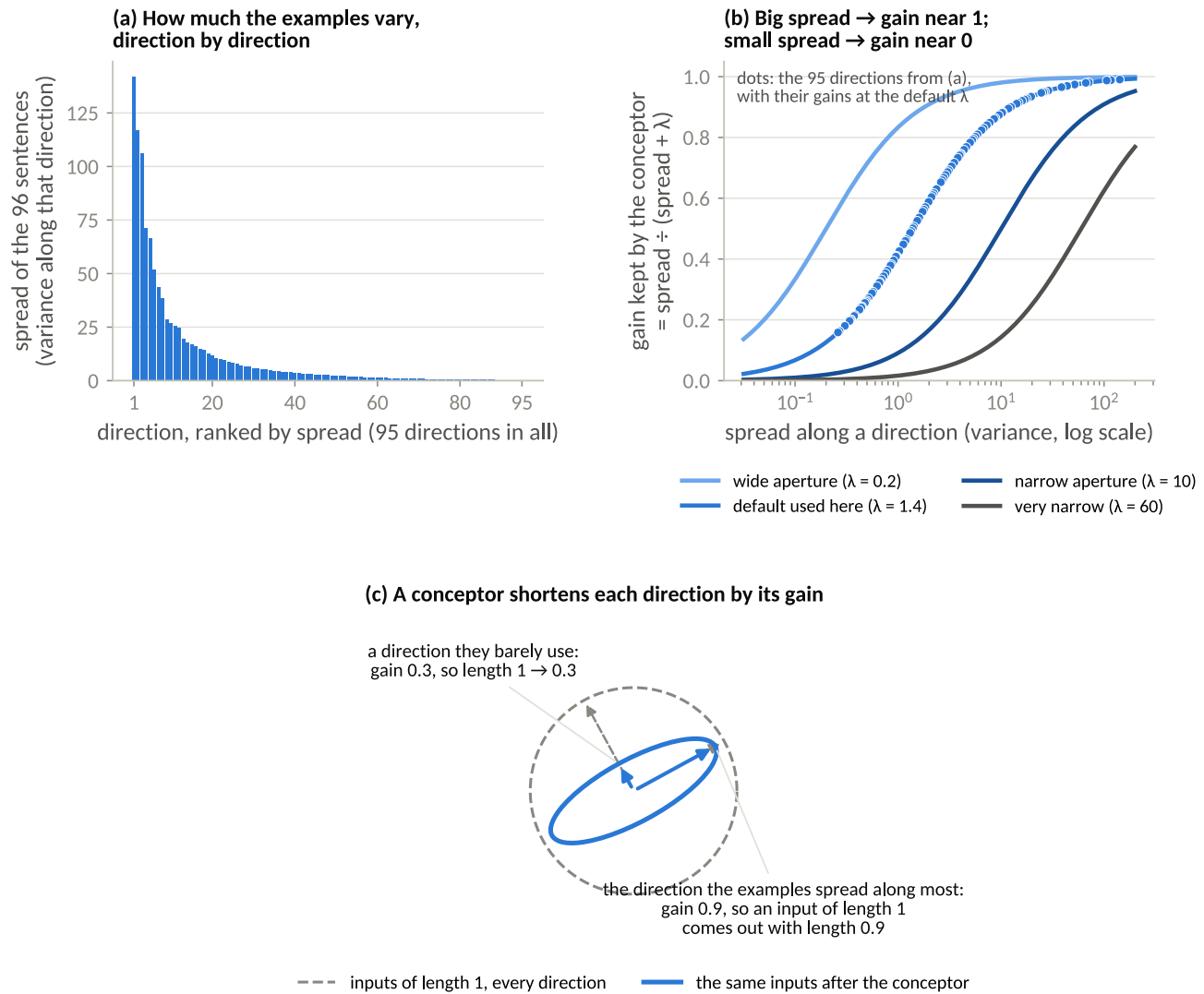


Figure 20. Building a conceptor from the 96 layer-8 summary rows. (a) The variance of the examples along each of their 95 directions of spread, ranked. (b) The rule that turns a variance into a gain, for four settings of λ ; dots are the actual 95 directions at $\lambda = 1.4$. (c) What the gains do to an input: an input of length 1 pointing along a direction the examples use heavily comes out with length 0.9 (gain 0.9); one pointing along a direction they barely use comes out with length 0.3 (gain 0.3). Applied to inputs pointing every which way (the dashed circle), the filter turns the circle into an ellipse.

Direction 3 is the sentiment axis: negative sentences at one end, positive at the other. Direction 4 runs from first-person feelings ("I feel ...", "I love/hate ...") to remarks about books, paintings and songs, with both positive and negative sentences at each end: it is a topic axis, not a sentiment axis. Direction 5 runs from sentences about people (teachers, doctors, a presenter) to sentences about places and weather, again with both sentiments at each end. Directions 1 and 2 (not shown) mix sentence shape with sentiment. So "the 20 strongest directions" of this conceptor means: the sentiment axis, plus a couple of shape-and-sentiment axes, plus a collection of topic axes along which the example sentences happen to vary. A conceptor that keeps all of them keeps "the kinds of variation that occur among opinionated sentences like these," which is both more than the arrow and, as Section 5.7 shows, sometimes more than you wanted. One point of bookkeeping, since the

phrase recurs: the experiments below use a conceptor cut down to those 20 leading directions, with their gains, and the other 75 directions dropped. That is a *truncated* conceptor, not the full one defined in step 4, which keeps all 95 directions; the cut is made to match the protocol of Triantafyllopoulos and colleagues, who use 20 throughout, and the appendix reports that the full conceptor gives the same results in every experiment where both were run.

To apply a conceptor to any row of numbers (the summary row of a new sentence, say): express the row as a combination of the conceptor's directions, multiply the amount along each direction by that direction's gain, and put the pieces back together. Directions the examples spread along a lot pass through almost unchanged; directions the examples never used are removed. Figure 20(c) shows the effect on inputs of every direction: a circle becomes an ellipse whose long axis is the direction with the highest gain.

ANOTHER WAY TO THINK ABOUT IT

A graphic equalizer on a stereo has a slider for each frequency band, from 0 (silence that band) to 1 (pass it through). A conceptor is an equalizer for meaning: one slider for each direction in the number-space, with the slider set high for directions the concept's examples actually use and low for directions they do not. Fitting a conceptor is letting the examples set the sliders.

A second analogy: it is shrinkage. A cautious statistician who distrusts estimates based on little evidence shrinks them toward zero, and shrinks the well-supported ones less. The gain formula is precisely such a rule: a direction with a lot of spread (strong evidence that the concept uses it) is kept nearly whole, and a direction with little spread is shrunk hard.

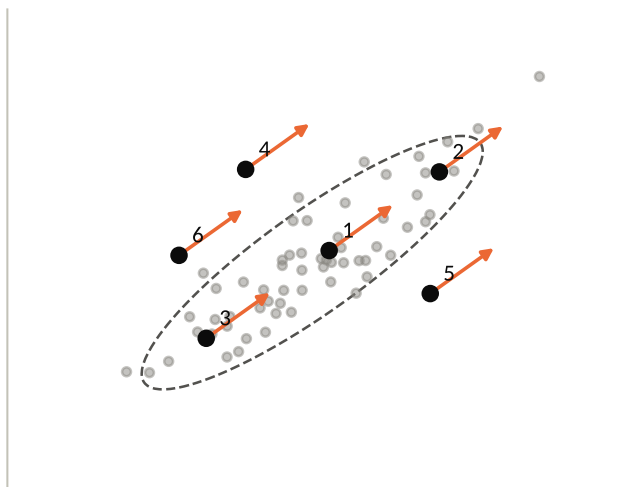
5.3 The aperture: the "how picky" dial

λ controls how sharply the conceptor discriminates between well-used and rarely-used directions. The research literature usually talks instead about the **aperture**, a number α with $\lambda = 1/\alpha^2$, so that a wide aperture (large α , small λ) lets nearly every direction through, and a narrow aperture (small α , large λ) keeps only the dominant few. Figure 20(b) shows the curves for four settings. Wide aperture: almost all 95 directions get gains near 1 and the conceptor is close to a plain "keep everything the examples touched." Narrow aperture: only the top handful survive. The experiments in this article use λ equal to the average variance per direction (about 1.4 here), the default used by Triantafyllopoulos and colleagues. The average gain across all 768 directions, called the **quota**, is 0.076 for our conceptor: it keeps a small slice of the space and mutes the rest, which is what a filter for one concept should do.

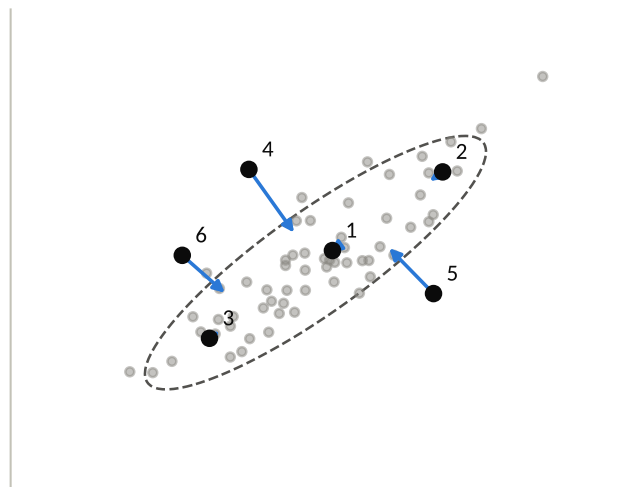
5.4 Three things you can do with a concepton

Before the figure, the correspondence to keep in mind, because it is easy to lose: a *row* (768 numbers) is drawn as a *dot*; a set of example sentences is a *cloud* of dots; the cloud's *shape* is the ellipse the concepton records; the *arrow* is the line from one group's center to the other's. Figure 21 draws all of this in two dimensions instead of 768 (so that the dots can be seen), on a toy cloud, and shows the three operations that come up in the research literature, next to the arrow for comparison. Six test points are marked; think of each as a new sentence's summary row.

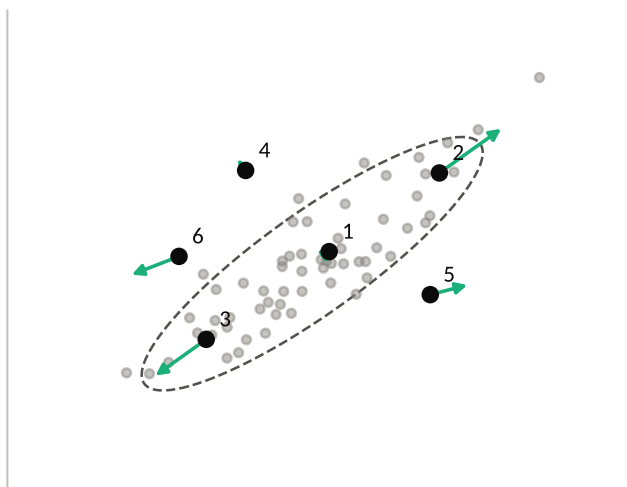
(a) Adding an arrow: every point moves by the same amount, wherever it started



(b) Applying the concepton: off-pattern points are pulled in; on-pattern points barely move



(c) Amplifying with the concepton: each point is pushed further along the pattern, its own way



How to read these panels

- gray dots: the example rows the concept was fitted on
- dashed ellipse: the shape of those examples
- numbered dots: six new rows (new sentences)
- (a) where the arrow moves each row
- (b) where the concepton, used as a filter, moves it
- (c) where the concepton, used as an amplifier, moves it

Row 1 sits at the center; rows 2 and 3 lie along the pattern; rows 4, 5 and 6 lie off it.

Figure 21. A toy cloud of examples (gray dots, each one a row), its shape (dashed ellipse), and six test rows (numbered dots). (a) Adding the arrow: every row moves by the same amount. (b) Filtering through the concepton (new row = center + the concepton applied to the row's offset from the center): off-pattern rows 4, 5 and 6 are pulled toward the pattern; on-pattern rows 1, 2 and 3 barely move. (c) Amplifying with the concepton (new row = row + strength × filtered offset from the center): every row is pushed further out along the pattern, in *its own* direction, by an amount that depends on where it already is. Row 1, at the center, is left almost alone.

- **(a) Adding the arrow** is Section 4's steering. Same displacement for everyone. In sentence terms: every sentence, whatever it says, is nudged the same distance toward "positive."
- **(b) Filtering.** Replace a row by the center plus the conceptor applied to the row's offset from the center. Whatever part of the row lies along directions the concept uses is kept; whatever lies elsewhere is shrunk toward the center. In sentence terms: a sentence about something the examples never touched (say, the rules of chess) has its chess-related components turned down and is pulled toward "a typical opinionated sentence," while a sentence that already reads like the examples is left nearly as it was. This is how conceptors were used in their original setting, and it is how they were first applied to language models (Postmus and Abreu, 2024, who multiplied activations by a conceptor to make a model perform a task).
- **(c) Amplifying.** Keep the row and *add* a multiple of the filtered offset. The effect is like the arrow, except that the push is computed from the row itself: a row that already leans one way along the pattern gets pushed further that way; a row that leans the other way gets pushed the other way; a row at the center is hardly touched. In sentence terms: a negative sentence is made more negative and a positive one more positive, by the same filter. This is the operation Triantafyllopoulos and colleagues study, and it is the one that matters most in the comparison with the arrow.

Look again at panel (c) and compare with (a). In (a), point 3 (already far along the pattern in the lower-left direction) is pushed toward the upper right, against its own tendency. In (c), it is pushed further lower-left. The arrow has one opinion about where everything should go; the conceptor's push depends on the input. In the language of the field, addition is a *fixed displacement*, while conceptor amplification is *input-dependent*.

5.5 The same thing, done for real inside GPT-2

Here is the experiment that makes the difference concrete, run on the real sentences. It mirrors, at small scale, the central experiment of Triantafyllopoulos and colleagues.

1. Split the 48 pairs into 36 "source" pairs and 12 held-out pairs.
2. From the 72 source summary rows, compute the arrow (Section 4) and fit a conceptor (Section 5.2) on all 72 rows pooled together, centered on their overall average.
3. Keep a fixed **readout**: the sentiment ruler of Figure 16, defined by the source arrow. It will be used to score rows before and after editing, and it is never changed.
4. Take the 24 held-out summary rows and edit each in several ways: (a) add the arrow; (b) amplify with the conceptor, keeping its 20 strongest directions (the sentiment axis plus the shape and topic axes described above); and (c), as a control, the same amplification with no conceptor at all, simply scaling up each row's offset from the center. The strength of every edit is set so that the average size of the displacement is the same (10% of the typical distance of a source row from the center).
5. Score every held-out row on the ruler before and after each edit.

24 held-out sentences, edited inside GPT-2 (layer 8) with the same total edit size, then re-read with the fixed sentiment ruler

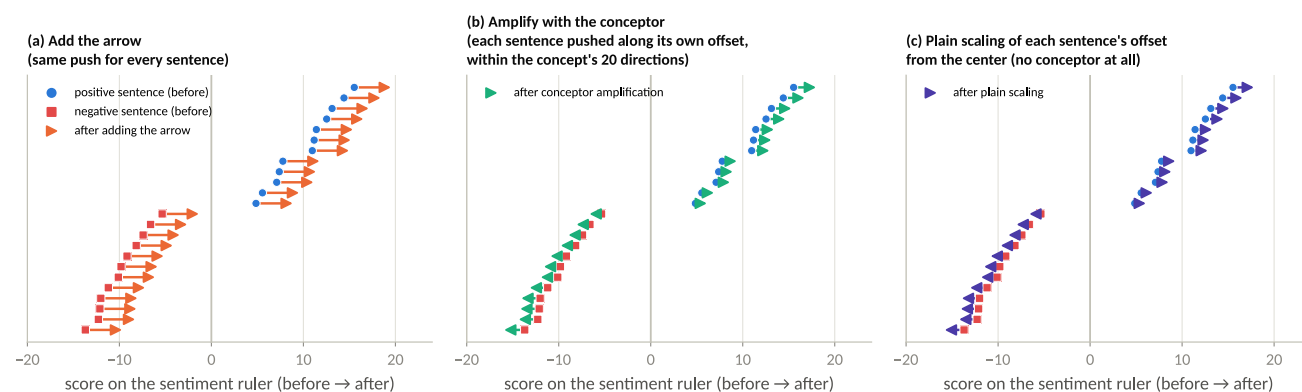


Figure 22. Twenty-four held-out sentences (12 positive, 12 negative), scored on the fixed sentiment ruler before (dot) and after (triangle) an edit of the same total size. (a) Adding the arrow moves every sentence to the right by the same amount: the positives become more positive, but the negatives become *less* negative. (b) Amplifying with the conceceptor moves positives right and negatives left: both poles are strengthened, each sentence pushed in its own direction. (c) So does plain scaling of each sentence's offset from the center, with no conceceptor involved; on the ruler the two are nearly indistinguishable. The two-sidedness comes from scaling an offset rather than adding a fixed vector; what the conceceptor adds is *where* the push is allowed to go.

Two of the held-out pairs make the point in numbers. "The picnic was a delight." starts at +11.2 on the ruler; adding the arrow takes it to +14.4, and the conceceptor takes it to +12.4. Its twin, "The picnic was a disaster.", starts at -13.7; adding the arrow drags it to -10.4 (less negative: the ruler now misjudges it), while the conceceptor pushes it to -15.2 (more negative). Likewise "The nurse was gentle and attentive." goes from +4.9 to +8.1 (arrow) or +5.4 (conceceptor), and "The nurse was rough and inattentive." goes from -12.0 to -8.8 (arrow) or -13.3 (conceceptor). The arrow does not know or care which sentence it is pushing; the conceceptor's push comes from the sentence itself.

Averaged over 20 random source/held-out splits: adding the arrow raised the positive sentences' scores by 3.25 and raised the negative sentences' scores by the same 3.25, which for a negative sentence is a loss of 3.25 in the strength of its negativity; the edit strengthened both poles in 0 of 20 splits. Conceceptor amplification with 20 directions raised positives by 1.11 and deepened negatives by 1.04, strengthening both poles in 20 of 20 splits.

Now the control, panel (c), which is the honest part of the story. Plain scaling of each row's offset from the center, with no conceceptor, raised positives by 1.01 and deepened negatives by 0.94, also in 20 of 20 splits. The reason is not mysterious once stated: the ruler measures each sentence from the midpoint between the two group averages, which is also the conceceptor's center, so *any* edit that multiplies a row's offset from that center by a positive number moves a positive sentence further positive and a negative one further negative, automatically. Two-sidedness is a property of scaling an offset rather than adding a fixed vector. It is not a property of the conceceptor's shape. Three further comparisons at the same edit size make the point from the other side. Scaling only the part of the offset that lies along the arrow (a hard projection onto the arrow's line, the crudest one-direction filter there is) strengthened both poles in 20 of 20 splits and by more than anything else, 3.12 and 2.92, because every bit of the edit budget is spent on the one direction the ruler

reads. A conceptor cut down to the single direction most aligned with the arrow (direction 3 of Section 5.2) did likewise, 2.07 and 2.12 in 20 of 20; cut down instead to its single strongest direction, direction 1, which is not the sentiment axis, it managed only 0.16 and 0.12 and succeeded in 12 of 20, which says that the direction matters, not that one direction cannot do the job. Even a random set of 20 directions strengthened both poles in 20 of 20 splits, by a tiny 0.17 and 0.14.

So what does the conceptor add, if not two-sidedness? Panel (b) against panel (c) shows it, though not on the ruler, which only reads one direction. Plain scaling pushes a sentence further along *everything* that distinguishes it from the center: its sentiment, its topic, its wording, its length, and the hundreds of directions in which it differs from the examples for reasons that have nothing to do with opinion. The conceptor pushes it only along the directions the opinionated example sentences actually vary in, and in proportion to how much they vary there (Figure 20). The arrow-line projection is the opposite extreme: it pushes along one direction and nothing else, which is why it scores highest on a ruler that reads that one direction. The conceptor sits between the two: input-dependent like plain scaling, confined to the concept's own region like the arrow, and, unlike the arrow's line, keeping the room in which a bad meal and a bad movie remain different sentences (Section 6.2). That is the right description of the operation, and the two-sided result is a consequence of it rather than evidence for it.

Triantafyllopoulos and colleagues report the two-sided pattern across five models (from 2 to 9 billion knobs) and three concepts (sentiment, political stance, expressions of depression): adding the mean-difference arrow strengthened the positive pole and weakened the opposite pole in every one of 30 conditions; a 20-direction conceptor fitted on both poles strengthened both in all 30; a 1-direction conceptor managed it in 10. Figure 25(b) in the next section shows their averages, and the control of panel (c) applies to those numbers in the same way.

WHY "BOTH POLES" MATTERS, AND WHAT IT DOES AND DOES NOT SHOW

If all you ever want is to make a model more positive, an arrow is fine. But often what you want is to make a model *more decisive about whatever the input already is*: sharper at registering negativity when it is there and positivity when it is there, using one operation, without deciding in advance which way to push. That is a two-sided goal, and *adding* an arrow cannot serve it, because it pushes everything one way. Any operation that scales a row's offset from the center can serve it, the conceptor among them. What the conceptor contributes is the choice of *which* part of the offset gets scaled: the part that lies in the region the examples of the concept occupy, and nothing else.

5.6 Cleaning up a noisy estimate

The third limitation of the arrow in Section 4.6 was noise when examples are few. A conceptor fitted on plenty of examples of a concept can be used to clean up an arrow estimated from very few, because it knows which directions are "sentiment-like." This is the one place in the article

where a conceptor is applied not to a sentence's row but to an arrow, so it deserves a careful example.

Suppose all you have is two sentences: "I ate a healthy dinner." (mildly positive) and "These shoes don't fit." (mildly negative). You can make an arrow from them exactly as in Section 4, with groups of size one: the first sentence's summary row minus the second's. That arrow is a poor estimate of "the sentiment direction," for two reasons at once: the sentences are weak examples, and there is only one pair, so the difference between them contains everything that distinguishes those two particular sentences (dinner versus shoes, eating versus fitting, the wording) mixed in with the little sentiment they carry. To judge it we need a reference arrow that the cleanup step has never seen, so the 48 pairs are split: pairs 1–36 are used to fit the conceptor, and the arrow from pairs 37–48 is the reference. Measured against that reference, the two-sentence arrow agrees with it at a similarity of only 0.32.

Now pass the two-sentence arrow through the 20-direction conceptor fitted on pairs 1–36. The conceptor knows nothing about dinner or shoes, and nothing about the reference pairs; it keeps the components of the arrow that lie along directions opinionated sentences actually vary in, and turns down the rest. Its agreement with the reference arrow rises to 0.48. (A hard projection onto the same 20 directions, with no gains, gives 0.47; here the soft and hard versions do the same work.) To check that this matters for steering, the same restaurant prompt was steered with each arrow (scaled to the same length, pushed toward "negative"): with the raw two-sentence arrow the model's top guesses were an indecisive *bad* 26%, *good* 25%; with the cleaned arrow, *bad* 38% and *good* down to 9%; with the arrow from the 36 fitting pairs, *bad* 62%. The cleaned arrow is still weak, because two weak sentences only contain so much, but it points in a better direction.

Two things to be clear about. First, this is not one steering instrument steering another; the conceptor here is not steering anything. It is acting as a *template* of what sentiment-like variation looks like, a quality-control step applied to the arrow before the arrow is used. Second, the reader's intuition that it would "remove concepts related to dinner and shoes" is the right one, with one refinement: the conceptor removes whatever lies *outside* the directions that the 96 opinionated sentences vary along; dinner-versus-shoes is removed only insofar as that kind of variation was absent from the examples, which is why the fitting examples matter so much.

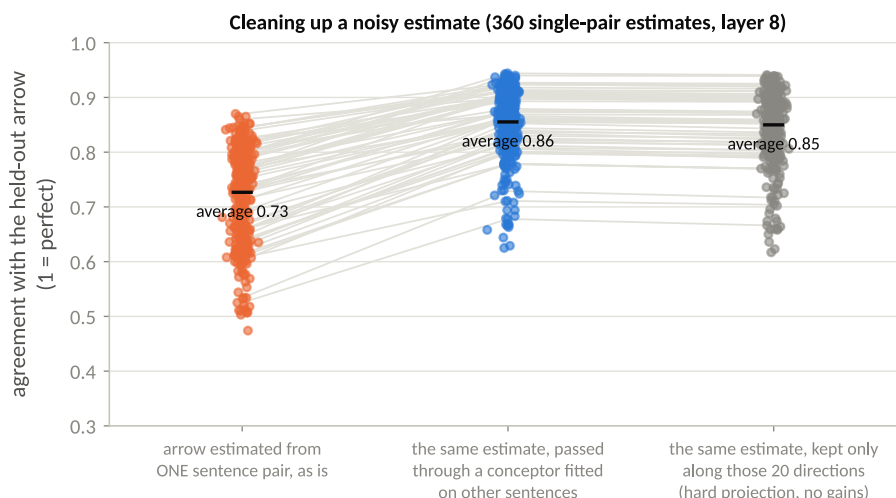


Figure 23. The same idea tested systematically: each dot is an arrow estimated from a single held-out sentence pair, scored by its agreement with the arrow from the other held-out pairs. Passing the estimate through a conceptor fitted on the 36 source pairs raises average agreement from 0.73 to 0.86 (middle); keeping only the conceptor's 20 directions with no gains at all (a hard projection, right) does the same, 0.85.

Figure 23 repeats the test 360 times with single pairs drawn from the held-out sentences (which are proper, strongly worded examples, hence the higher starting agreement): on average 0.73 before, 0.86 after. The figure also shows that, on these sentences, the gains are doing almost nothing: a hard projection onto the same 20 directions scores 0.85. The cleanup comes from knowing *which* directions sentiment-like variation lives in; the soft gains are a refinement that matters more when the directions are less cleanly separated. Triantafyllopoulos and colleagues report the same effect across all five of their models, with the largest gains when examples are scarcest, with the soft conceptor ahead of the hard projection in their data, and they note honestly that with plenty of examples the cleanup is no longer needed and can even hurt slightly.

5.7 A cautionary tale: a conceptor captures everything the examples share

One experiment for this article did not go the way a naive reading would predict, and it teaches something important. A conceptor was fitted on the layer-8 summary rows of the 48 *positive* sentences alone, and then applied as a filter (operation (b) above) to the row at the end of the prompt "We tried the new restaurant on Main Street. The food was", in the hope of producing an even more positive next word. Instead, the model's top guesses for the next token became *The*, *It*, *I*, *We*: words that begin a new sentence.

The reason is instructive. The rows the conceptor was fitted on were all summary rows, taken at the *final period* of a sentence. The strongest thing those rows have in common is not "positive"; it is "we are at the end of a sentence." The conceptor faithfully kept that, and the model, handed a row that screamed "end of sentence," did the sensible thing and started a new one. A conceptor captures the whole shape of its examples, including whatever they share that you did not intend.

Triantafyllopoulos and colleagues say the same thing in their own words: the directions a conceptor retains "may also reflect topic or wording." Choosing the fitting examples is where the thinking happens; the arithmetic afterward is automatic.

A WARNING LABEL FOR EVERYTHING IN THIS SECTION

Fitting a conceptor is easy. Fitting the conceptor you *meant* requires examples that vary in the ways you want to keep and are matched in the ways you want to ignore. The sentence pairs used here were written to differ only in sentiment; that matching is doing a lot of quiet work in every figure.

5.8 AND, OR, NOT: conceptors can be combined like filters

A steering arrow for "positive" and a steering arrow for "formal" can be added together, but adding two arrows just makes a third arrow, and there is no natural way to say "positive but not formal." Conceptors, because they are filters with a gain per direction, support genuine logical combination, and the combinations have simple meanings:

- **NOT A** keeps exactly the directions A mutes, and mutes the ones A keeps (its gain along each direction is 1 minus A's gain).
- **A AND B** keeps only directions that both A and B keep.
- **A OR B** keeps directions that either keeps.

Figure 24 shows what these do to a circle of inputs in two dimensions. If you have ever combined criteria in a spreadsheet or a search ("Region = West AND Amount > 10,000"), the logic is the same, except that the criteria are soft (gains between 0 and 1 rather than pass/fail). In sentence terms: a conceptor for "opinionated" AND a conceptor for "about food" keeps the directions that opinionated sentences about food use, and NOT "about food" removes food-related variation from whatever it is applied to.

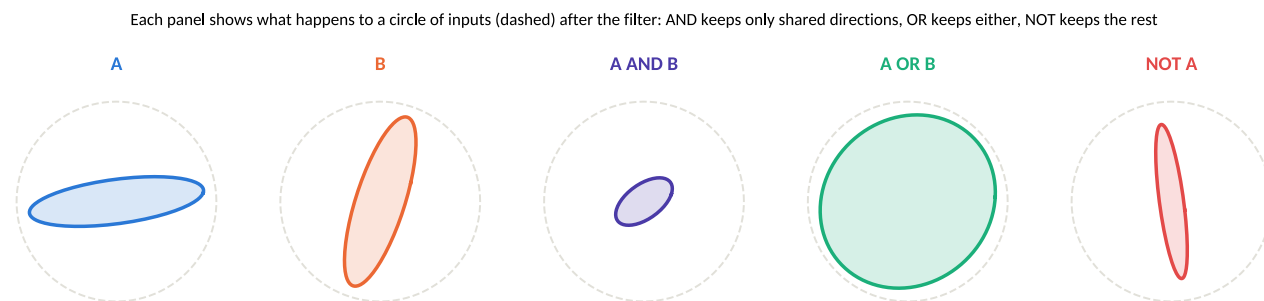


Figure 24. Two toy conceptors, A and B, fitted on clouds that stretch in different directions, and the three logical combinations. AND keeps only the direction both clouds share (a small ellipse along the diagonal); OR keeps nearly everything either uses; NOT A keeps what A discards. The exact formulas are in the original conceptor papers; the picture is what they mean.

This is not merely elegant. Postmus and Abreu (2024) tested it on composite tasks: making a model both translate a word into French *and* give its antonym, say. A conceptor formed by AND-combining the two tasks' conceptors outperformed the natural alternative of averaging the two tasks' steering arrows, on all three composite tasks they tried. And Triantafyllopoulos and colleagues measured how much different concepts overlap in a model, which tells you how much collateral damage NOT will do: in all five of their models, "sentiment" and "expressions of depression" shared many directions (overlap around 0.3), while neither shared much with "political stance" (overlaps of roughly 0.04 to 0.18). Removing the political directions from a model's activations would leave 82–97% of the sentiment or depression pattern intact; removing sentiment would damage the depression-expression pattern considerably more (67–78% left). That matches intuition: someone describing feeling hopeless is also expressing negative sentiment, so the two patterns share directions, while an opinion about tax policy can be delivered in any mood. Conceptors make such interference measurable in advance.

5.9 Fitting from opposites versus fitting from presence and absence

The central question of Triantafyllopoulos and colleagues is one that will feel familiar from any experiment with a control group: *what should the comparison group be?* Suppose you want a conceptor for "positive sentiment." Two reasonable choices:

- **Bipolar fit:** pool positive sentences with their negative twins. "I found the film wonderfully engaging." together with "I found the film painfully dull."
- **Unipolar (presence-versus-absence) fit:** pool positive sentences with neutral ones on the same topics. "I found the film wonderfully engaging." together with "I watched the film in the evening."

The two choices give different shapes. The bipolar cloud stretches strongly along the sentiment axis, because it contains both extremes; the unipolar cloud stretches from "opinion" to "no opinion," which is a different direction (recall from Figure 3 that *happy* and *sad* are closer to each other than either is to an unrelated word like *truck*). The experiments across five models say which to use for what. If the goal is two-sided amplification (Section 5.5), the bipolar fit succeeded in 30 of 30 conditions and the unipolar fit in only 7 of 30, because only the bipolar conceptor knows what the negative end looks like. If the goal is simply to detect whether an opinion is present at all, presence-versus-absence is the natural comparison, just as you would compare a treatment group with an untreated one rather than with a group given the opposite treatment. Neither is universally better; the fit should match the use.

The same authors also prove a tidy fact about the bipolar fit that is worth knowing in words, with our sentences as the example. A conceptor fitted on both poles pooled together is exactly the sum of three parts:

1. **The average of the two "own-pole" shapes.** Fit one conceptor on the 48 positive sentences around their own center and one on the 48 negatives around theirs, and average them. This

part carries the shapes: the fact that positive sentences vary from "delicious food" to "friendly staff" to "beautiful garden," and negative ones likewise.

2. **A one-direction piece along the arrow.** This carries the separation between the two averages: the sentiment axis itself.
3. **A shape-difference piece,** present only if the two poles have *different* shapes: if, say, negative sentences varied more in wording than positive ones did. It is zero when the two clouds have identical shapes.

Removing either of the last two pieces made the conceptor worse at reconstructing held-out contrasts in every one of 270 comparisons they ran. In plain terms: the arrow is inside the conceptor (piece 2), and the conceptor also knows things the arrow does not (pieces 1 and 3).

5.10 Where conceptors came from

Conceptors were invented by Herbert Jaeger in 2014 for a different problem. He was working with a kind of network (weighted sums stacked in stages, as in Section 2.5, but with the stages looped back on themselves so that the network's row of numbers evolves step by step over time, like the state of a system rather than the reading of a sentence). He wanted one such network to memorize many different patterns at once, rhythms and signals, and the trouble was that patterns stored in the same network interfered with one another. His solution was exactly the recipe of Section 5.2: while the network was producing one pattern, collect the rows of numbers it passed through, find the directions they spread along, turn the spreads into gains, and keep the resulting filter. To recall that pattern later, switch its filter on, which confines the network's row to the pattern's own directions and silences the others; to release it, switch the filter off. The AND/OR/NOT algebra was part of the original design, so that patterns could be combined and subtracted. Postmus and Abreu (2024) were the first to apply the idea to steering large language models, and Triantafyllopoulos and colleagues (in preparation) give the first careful account of what a conceptor fitted on language-model activations actually contains, and how the choice of fitting examples shapes it. The field is young: that account is explicit that its results concern edited activations and fixed readouts, and that steering of generated text with conceptors is left as future work beyond Postmus and Abreu's initial demonstrations.

6 What steering can do that prompts cannot, and what conceptors add

Anyone who has used a chatbot knows you can change its behavior by asking nicely: "be concise," "write in a friendly tone," "do not speculate." So why reach inside the model at all? This section answers that in two steps: first, what a prompt fundamentally cannot reach, which is an argument about steering of any kind, arrow or conceptor; then, what a

conceptor adds on top of the arrow, each point backed by an experiment you have already seen or a published result, with the controls that say how much of it is the conceptor's doing.

6.1 What a prompt can and cannot reach

A prompt is text. Everything it does, it does by being read: turned into rows of numbers at the lookup table and pushed through all the layers, alongside everything else in the conversation. That has consequences.

A prompt has no grip unless the model's reading process gives it one. The most direct demonstration is the small model used throughout this article. GPT-2 was trained only to continue text; nobody ever trained it to follow instructions. Here is what happens when you try to instruct it to be negative, compared with steering:

What was tried	Probability on positive words	on negative words	Top guess
Plain prompt: "We tried the new restaurant on Main Street. The food was really ..."	89%	0.5%	good (74%)
Prefixed with "Write a negative review."	86%	2%	good (73%)
Prefixed with "This is a one-star review."	88%	0.6%	good (73%)
Prefixed with "Please be very critical and negative in what follows."	86%	0.8%	good (69%)
Prefixed with "Warning: the following review is extremely negative."	78%	6%	good (66%)
Plain prompt, steered with the arrow at strength -2 (Section 4)	1%	80%	bad (65%)

Five different instructions, none of which moves the model more than a few percentage points; one edit to the layer-8 numbers, which flips it completely. This table is not a fair contest, and it is not meant as one: GPT-2 was never trained to follow instructions, and a modern chat model would comply with "write a negative review" at once. What the table shows is the *mechanism*, stripped of instruction-following: an instruction is text, and text acts only through the reading process. But the mechanism is the same in both cases. An instruction works only to the extent that the model's layers convert those words into the right internal state. Where that conversion is weak, missing, or actively opposed by the model's training, the instruction is words on a page. Steering writes the internal state directly.

A bias can be corrected where it is computed. Ravenda and colleagues' clinical model (Section 4.5) was a modern instruction-following model, prompted in plain language to answer only "relevant" or "not relevant," and it over-flagged. The authors did not report trying to reword their way out of it, so nothing here says that a prompt could not have helped; what their result shows is that an edit at the middle layer, built from two averages, corrected the bias with a measurable and tunable trade-off, which is a kind of control a prompt does not offer.

A prompt is not a dial. There is no reliable way to ask for "37% more positive," or to sweep smoothly from very negative to very positive and watch the output change continuously, as Figure 17 did. Steering strength is a real number.

A prompt can be forgotten or overruled by later text; an edit is not part of the argument. A prompt sits at the start; everything after it competes for the model's attention. An edit applied at every generation step, inside the computation, is not in that competition, because the text never sees it. Here is the demonstration on GPT-2: the prompt was given a strongly positive lead-in, "This is a glowing five-star review. We tried the new restaurant on Main Street. The food was really ...", and the arrow was added at strength -2 anyway. Unsteered, the model says *good* (67%). Steered, it says *bad* (53%), then *awful*, *terrible*, *poor*, with 75% of its probability on negative words. The text said five stars; the edit said otherwise; the edit won. Whether that is good or bad depends entirely on who is doing the editing and why; Section 6.3 returns to this.

A prompt cannot read. Prompts change output. They give you no number that says how positive, how true, or how symptom-laden the model judged a passage to be. The arrow and the conceptor give you that number (Figure 16), and it is the same object you steer with.

ANOTHER WAY TO THINK ABOUT IT

A sign at the entrance of a library asking visitors to keep their voices down is a prompt: it is read, interpreted, sometimes ignored, and forgotten by the second floor. Soundproofing the walls is steering; it applies to everyone, automatically, whether or not they read the sign. A conceptor is soundproofing tuned to the building's actual acoustics, room by room, rather than one blanket layer of foam everywhere.

6.2 What conceptors add on top of the arrow

Steering with an arrow already reaches what a prompt cannot, for the reasons above. Everything in this subsection is about what a conceptor adds *beyond the arrow*, and in each case the question to ask is the one Section 5.5 asked: how much of the effect is the conceptor, and how much is something simpler?

1. The push depends on the input and stays inside the concept's region

You saw this in Figure 22 and the sentences around it: with one conceptor, "The picnic was a delight" moved from $+11.2$ to $+12.4$ and "The picnic was a disaster" from -13.7 to -15.2 , while the arrow moved both to the right ($+14.4$ and -10.4). Triantafyllopoulos and colleagues' central result,

averaged over five models and three concepts, is reproduced in Figure 25(b): adding the arrow changed the readout by $+0.29$ for the positive pole and -0.29 for the opposite pole, while the 20-direction conceptor changed both by about $+0.03$ in the strengthening direction, in 30 of 30 conditions. Read that figure with Section 5.5 in mind. The contrast between the two bars is the contrast between adding a fixed vector and scaling an offset, and any scaling of the offset would produce the two-sided pattern; a hard projection onto the arrow's line would produce it with a far larger number, because it spends the whole edit on the one direction the readout measures. The conceptor's $+0.03$ is small for the same reason it is interesting: the edit is spread over the twenty directions the concept actually occupies, so only a fraction of it lands on the readout. The property the conceptor offers is therefore not "two-sided" but "input-dependent and confined to the concept's subspace": sharper at registering whatever the input already is, along the directions in which such inputs genuinely differ, and nowhere else.

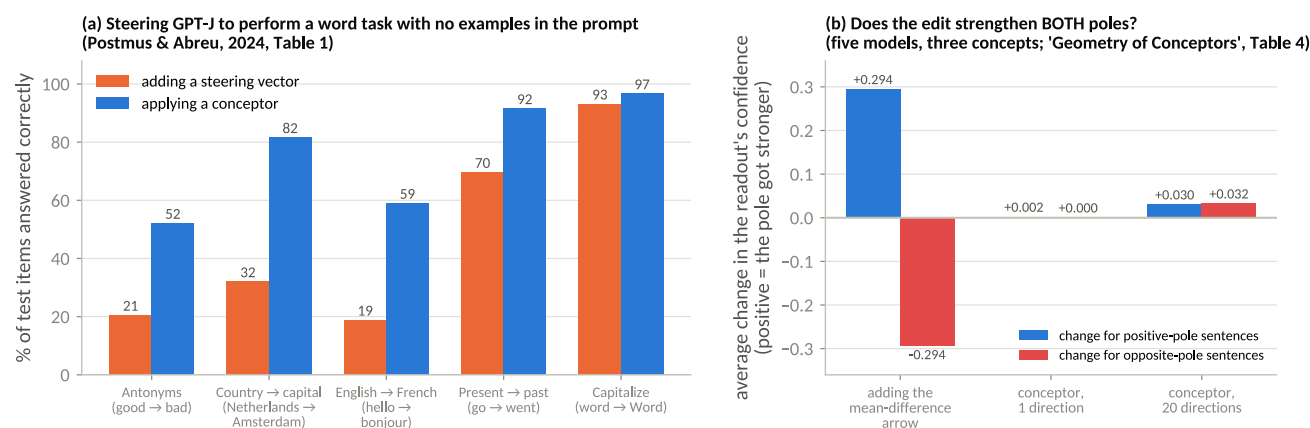


Figure 25. Two published comparisons, redrawn from the numbers reported in the papers. (a) Postmus and Abreu (2024, Table 1) steered GPT-J (a 6-billion-knob model) to perform word tasks with no examples in the prompt, using either an added vector or a conceptor built from the same activations. Conceptors more than doubled accuracy on antonyms, country-to-capital and English-to-French. (b) Triantafyllopoulos and colleagues (in preparation, Table 4): five models, three concepts, edits of matched size; adding the mean-difference arrow strengthens one pole and weakens the other, while a 20-direction conceptor fitted on both poles strengthens both. As Section 5.5 shows, the two-sided pattern follows from scaling an offset rather than adding a vector; the figure should be read as "input-dependent" rather than "stronger."

2. A conceptor keeps more of the concept, and the right twenty directions

An arrow is one direction, so passing a sentence through "the arrow only" reduces it to a single number: its position on the ruler. That throws away every other distinction. Here is what that costs, with two real held-out negatives. "The picnic was a disaster." and "The lecture was confusing and dull." are different complaints about different things; their summary rows have a similarity of only 0.10 (they point in quite different directions). Keep only what lies along the arrow and their similarity becomes exactly 1.00: they have been collapsed onto the same line, differing only by how far along it they sit (-14.9 versus -12.0). Pass them instead through the 20-direction conceptor and their similarity is 0.18: still the same sentiment, but recognizably different sentences, because the topic axes (the picnic-versus-lecture kind of variation, Section 5.2) were kept. Twenty directions will always keep more than one, so the fair comparison is with other twenty-direction filters. On

our held-out sentences, the arrow's line keeps 10% of a summary row (measured from the center, squared), the 20-direction conceptual 53%, a hard projection onto the same 20 directions 56%, and a projection onto 20 directions chosen at random 3%. The conceptual's advantage over the arrow is mostly that it has twenty directions rather than one; its advantage over a random twenty is that they are the *right* twenty, the ones opinionated sentences vary along, which a random choice misses almost entirely. (The random projection also leaves the two complaints looking more alike, similarity 0.48, than the conceptual does.)

Triantafyllopoulos and colleagues measured this at scale: how much of a held-out sentence's summary row survives when it is passed through the arrow's line alone versus a 20-direction conceptual, averaged across their five models:

Concept	Retained by the arrow's line alone	Retained by a 20-direction conceptual
Sentiment	36%	68%
Political stance	24%	52%
Expressions of depression	39%	60%

Percent of a held-out sentence's row (measured from the center, squared) retained after filtering; averaged over the two poles and five models. Source: Triantafyllopoulos et al., Table 2.

The same caution applies to this table: a 20-direction filter of any kind retains more than a line does, and the table does not include a hard-projection or random-subspace column, so it shows that the conceptual's directions capture most of what a held-out sentence carries, not that the gains are doing the work. The gap is largest for the concepts that are inherently multi-faceted (political stance, depression), which is exactly where a single arrow is most obviously too crude. When the authors held out entire *groups* (all the Yelp reviews, or all the tweets about one topic) and asked how well a conceptual fitted on the other groups captured the held-out group's own arrow, the 20-direction conceptual fitted on both poles pooled together won in 14 or 15 of 15 model-and-concept conditions over a conceptual fitted on the positive examples alone or on the negative examples alone. (This is a different comparison from the bipolar-versus-unipolar one of Section 5.9: there, the alternative was a presence-versus-absence fit, which has its own uses; here, the alternatives were one-sided fits with no comparison group at all, and those simply capture less.) More of the concept, and more of it generalizes.

3. Conceptors combine logically and clean up noisy estimates

Both were shown in Section 5 and are only recalled here. AND, OR and NOT (Section 5.8) give a composable algebra of behaviors that arrows lack ("polite AND concise"; "opinionated AND NOT about food"), with the overlap between two filters computable before anything is deployed; Postmus and Abreu found AND-combined conceptors beat averaged steering vectors on every composite task they tested. And a conceptual fitted once on good examples cleans up an arrow

estimated from a handful (Section 5.6): 0.32 to 0.48 for the dinner-and-shoes pair, 0.73 to 0.86 over 360 single-pair estimates, with the caveat recorded there that a hard projection onto the same directions did as well on our sentences; Triantafyllopoulos and colleagues report the soft version ahead of the hard one across their five models (normalized errors of 0.51 to 0.65 against 1.0 for the raw estimate, better than the hard projection in all 60 conditions).

4. Conceptors are also a diagnostic: they tell you where to look

The quota (the conceptor's average gain, Section 5.3) can be computed at every layer with no labels and no classifier. Triantafyllopoulos and colleagues computed it, layer by layer, for five models, and compared the resulting curve with a second curve: how accurately a classifier trained on the model's rows at each layer could recognize the concept on six public *benchmarks* (a benchmark is a standard, publicly shared set of labeled examples, such as a few thousand movie reviews marked positive or negative, that researchers use so that different methods can be compared on the same test). The two curves had the same hills and valleys: the layers where the quota was high were the layers where the concept was most readable, in all 30 model-and-benchmark combinations (average correlation across layers 0.79). In other words, a conceptor fitted on a handful of example sentences gives a cheap map of which layers hold the concept in readable form, which is the question Section 3 answered laboriously with a t-test at every layer. The authors are careful to say this is a descriptive guide, not a rule; it is nonetheless a guide that prompts cannot provide, since prompts have no access to layers at all.

And there is the accuracy result in Figure 25(a), from the first application of conceptors to language models: on five word tasks in GPT-J, steering by conceptor beat steering by an added vector on every task, roughly doubling or tripling accuracy on the three hardest (antonyms 21% to 52%; country-to-capital 32% to 82%; English-to-French 19% to 59%). Those tasks are trivial for a modern chat model with a prompt, of course. The point of the comparison is not "conceptors beat prompts on translation"; it is that *given the same internal information*, a filter that knows the concept's shape extracts far more of it than an arrow does.

6.3 Four scenarios

Concrete situations make the abstractions bite. Each of these is either drawn from the papers or is a straightforward extrapolation from them; the last paragraph of the section says which is which.

Scenario A: a screening tool that must run locally

A clinic wants to flag which patient messages are relevant to specific depression symptoms so a clinician can review them. Privacy rules require the model to run on the clinic's own hardware, which means a small model; and the small model, as Ravenda and colleagues found, over-flags. A steering arrow corrected that, with a trade-off the authors could measure and tune, and the steered model's questionnaire scoring was competitive with much larger cloud models. Now add the conceptor's contribution: the paper on symptom landmarks (Zhu, Subramanian and colleagues) found that mood, bodily and death-related symptoms occupy distinct but heavily overlapping

regions of the activation space (their three landmarks were so nearly aligned that a special correction was needed to read them separately). Overlapping regions are exactly what conceptors and their AND/NOT algebra are built to handle, and a conceptor fitted on both symptom-present and symptom-absent examples would sharpen the model's own reading in both directions (present becomes clearer, absent becomes clearer), which is what a screening tool needs, rather than a one-way push toward "flag it." Whether that yields clinical-grade performance is unproven; Triantafyllopoulos and colleagues say plainly that their depression results "do not establish clinical validity." But the machinery fits the problem in a way a prompt does not.

Scenario B: "polite and concise," dialed, not begged for

A customer-service deployment wants responses that are courteous and short. With prompts, you write "be polite and concise," discover that politeness makes the model verbose, add "but brief," discover that brevity makes it curt, and iterate through wordings with no way to measure what you have. With conceptors, as in point 3 above, you fit one filter on polite-versus-curt examples and one on concise-versus-verbose examples, AND them, dial the strength, and read the overlap to know in advance how much the two will interfere. The composite-task results of Postmus and Abreu are the existing evidence that AND-combined conceptors behave as intended; applying this to tone rather than word tasks is the extrapolation.

Scenario C: a standing disposition that is not part of the conversation

Consider a model deployed by a company, with two kinds of text arriving in its prompt: the company's own standing instructions and whatever a member of the public types. Both are text, so the model weighs them against each other, and a determined user can write text that outweighs the instructions ("ignore your previous instructions and..."). Section 6.1 showed, in miniature, a different way of installing a disposition: the prompt text said "glowing five-star review," the edit said negative, and the edit won, because the edit is not text and the text cannot see it. An edit applied at every generation step is not a party to the argument going on in the prompt. That is an attractive property for a standing disposition such as a house style or a level of caution, and it is the extrapolation in this scenario, not a demonstrated one: no published result shows an arrow or a conceptor capturing a rule like "never discuss competitors' prices," a tone is not a rule, and whether an edit can be worked around by a determined user is an open question that Section 6.4 lists among the things not yet known. Within that limit, the arrow and the conceptor both have the property; the conceptor additionally lets the disposition be a shaped region ("things that read like our approved style") rather than a single push.

Scenario D: measurement, not just control

Everything above is about changing behavior. Half the value of this machinery is in *reading*. The sentiment ruler of Figure 16 gave a number per sentence with no classifier training. Zhu and colleagues' symptom landmarks turned a memoir and a clinical handbook into per-passage symptom scores that preserved clinicians' rankings. A conceptor extends this from one number per concept to a whole profile: for each concept you have a conceptor for, pass the new sentence's

summary row through it and record what fraction of the row survives. Figure 26 does this with two conceptors built for this article, one for "opinionated sentences" (fitted on 72 of our positive and negative sentences) and one for "plain factual sentences" (fitted on 16 neutral sentences like "The hotel room has two windows."). A held-out opinion, "The picnic was a delight.", keeps 66% of itself under the opinion filter and 11% under the fact filter; a brand-new factual sentence, "The store closes at nine.", keeps 28% and 48%. The profile is a bar per conceptor, not a bar per direction, and it reads as "how much of this sentence lives in each region."

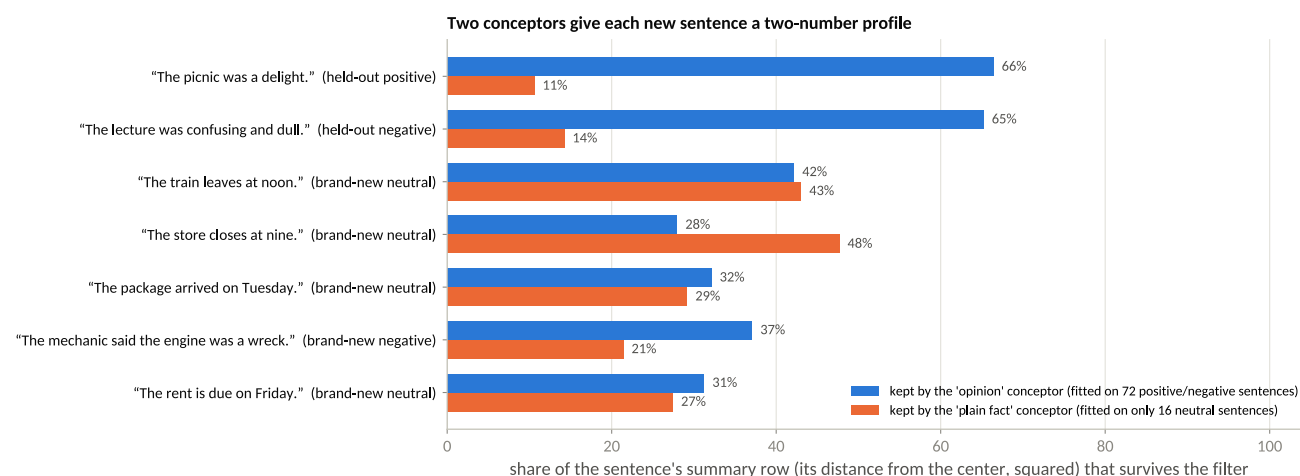


Figure 26. A two-conceptor profile for seven sentences (none of them used to fit either conceptor). Each bar is the share of the sentence's summary row that survives one filter. Opinions survive the opinion filter and not the fact filter; plain factual sentences lean the other way. The fact filter is weaker (its bars are lower and less consistent) because it was fitted on only 16 examples, which is too few to define a region well in 768 dimensions; the two brand-new sentences at the bottom, on topics neither set covered, keep little under either.

A prompt can be asked "is this passage sad?" and will answer in words; it cannot tell you where, in its own representation, the sadness lives, or how it relates to the other things it noticed. Note also the lesson in the weak fact filter: a profile is only as good as the example sets behind it, and sixteen sentences is not enough.

6.4 What is established, what is promising, what is not yet known

A fair summary of the evidence, so that you know how much weight each claim can bear:

- **Established (replicated in this article and in the papers):** concepts like sentiment are linearly readable from middle layers; the difference-of-means arrow reads and steers them; a conceptor built from the same example sentences that produced the arrow contains the arrow and more; scaling a row's offset from the center, whether by a conceptor or by nothing at all, strengthens both poles where adding the arrow cannot, and the conceptor's version of that scaling is the one confined to the concept's region; projecting a few-example estimate onto the concept's leading directions denoises it, softly by a conceptor or, on our sentences, equally well by a hard projection; the choice of comparison group (opposites versus presence/absence) changes what a conceptor does.

- **Established on one model family by one group:** conceptor steering beats added-vector steering on word tasks in GPT-J, and AND-combination works for composite tasks (Postmus and Abreu, 2024).
- **Promising but not yet demonstrated:** conceptor steering of free-form generated text in modern instruction-following models; clinical-grade screening; standing constraints of the kind Scenario C imagines, and robustness of any edit against a determined user; a direct comparison, on the same models, of a conceptor with a hard projection onto the same directions and with plain scaling, which this article could run only on GPT-2. Triantafyllopoulos and colleagues explicitly leave "downstream steering or generated text" to future work and call for "direct intervention studies."
- **Known limitations:** a conceptor captures everything its examples share (Section 5.7); a conceptor fitted on too few examples defines its region poorly (Figure 26); steering of any kind degrades into nonsense when pushed too hard (Figure 17, the "very very very" rows); the arrow and the conceptor both depend on choosing the right layer; and, as Triantafyllopoulos and colleagues' own control shows, for the narrow job of *classifying* which pole a sentence belongs to, the plain arrow was actually slightly better than the 20-direction conceptor in 25 of 30 conditions. The conceptor's advantage is in preserving and shaping the concept, not in drawing the single best dividing line.

THE ONE-PARAGRAPH ANSWER TO "WHY CONCEPTORS?"

A prompt changes what the model reads. A steering arrow changes what the model thinks, in one fixed direction, by one fixed amount. A conceptor changes what the model thinks *in proportion to what it already thinks*, along the directions the concept actually uses and no others, and can be combined with other conceptors by AND, OR and NOT. It is built from the same two averages as the arrow plus one more familiar thing, the spread of the examples, and it costs nothing at run time beyond a few multiplications. Some of what it does, a simpler operation can do too, and this article has tried to say where; what is left is a filter that knows the shape of a concept. That is why researchers working on the inside of language models are paying attention to a 2014 idea from a different corner of the field.

7 The whole story on one page

1. **Encoding.** A word (and later a sentence) is written as a row of hundreds of numbers, arranged so that similar meanings get similar rows. Similarity is closeness; directions in the space (hot-cold, cheap-expensive, sad-happy) have meanings of their own. (Figures 1-5.)
2. **Training.** The rows, and everything else in the model, are set by one procedure: guess the next word, score the guess, nudge all the knobs to make the true word more likely, repeat billions of times. It is the line of best fit with more knobs and a different task. Sensible encodings fall out

as a by-product. Almost every knob is a weight in a weighted sum; a bank of such sums rewrites a row at the same length. (Figures 6–9.)

3. **Layers.** A sentence is a table with one row per token. Each layer rewrites the whole table (mix rows, weighted sums, add the update back), producing a new version of the same table. Rows are tokens; layers are versions. Early versions know the word; middle versions know what the text means; late versions turn meaning into a next-word guess. The middle is where meaning is easiest to read. (Figures 10–15.)
4. **The arrow.** Average the middle-layer summary rows of examples of a concept, average those of its opposite, subtract. The result reads the concept off any new text (a ruler) and, added back in during processing, changes the model's behavior (steering). (Figures 16–18.)
5. **Conceptor.** Take the same examples and record not only where they sit but how they spread, direction by direction; turn each spread into a gain between 0 and 1. The result is a soft filter that keeps what the concept uses and mutes what it does not. Applied as an amplifier, it pushes each input further along its own offset from the center, but only within the concept's own directions; the two-sidedness of that push is shared with plain scaling, the confinement is not. Conceptors combine by AND, OR and NOT. (Figures 19–24.)
6. **Why it matters.** Prompts act on the input and only work insofar as the model reads them the way you hoped. Arrows act inside but push everything the same way. Conceptors act inside, in proportion to the input, along the concept's whole shape and nothing outside it, and are composable and measurable; where a simpler operation does part of the same job, the article says so. (Figures 25–26.)

Glossary

Activation

The row of numbers sitting at one token position at one layer while the model processes a text. Changes with every input. Contrast with *weight*.

Aperture (α) and λ

The single dial on a conceptor that sets how picky it is. $\lambda = 1/\alpha^2$. Large λ (narrow aperture) keeps only directions with large spread; small λ keeps almost everything the examples touched.

Arrow (difference-of-means direction, steering vector, mass-mean probe, CAA vector)

The average row of one group of examples minus the average row of the opposite group. Used to read a concept (project onto it) and to steer (add it). Always made from two groups.

Benchmark

A standard, publicly shared set of labeled examples that researchers use to compare methods on the same test.

Bipolar vs. unipolar fitting

Building an arrow or conceptor from a concept and its opposite ("wonderful" vs. "awful") versus from a concept and its absence ("I found the film wonderfully engaging" vs. "I watched the film in

the evening").

Conceptor

A soft filter built from a set of example activations: a list of directions with a gain between 0 and 1 for each, $\text{gain} = \text{spread} \div (\text{spread} + \lambda)$. Written as a matrix, $C = R(R + \lambda I)^{-1}$, where R is the examples' covariance.

Direction

An axis in the number-space, with two ends. The examples' directions of spread are ranked from most to least spread; "positive" and "negative" are the two ends of one such direction, not two directions.

Encoding

A fixed way of writing a thing as a row of numbers such that similar things get similar rows.

Gain

The multiplier a conceptor applies along one direction: 1 keeps it, 0 silences it, values between turn it down. A neutral word: "high gain" means "kept," not "good."

Hard projection

The all-or-nothing cousin of a conceptor: keep a chosen set of directions with gain exactly 1, discard the rest with gain 0.

Layer

One stage of the model: it takes the current table (one row per token), mixes each row with the rows before it, passes each row through a bank of weighted sums, and adds the result back. GPT-2 has 12; large modern models have 60 to 120. Layers are successive versions of the table, not rows of it.

Logit lens

Reading an intermediate layer's row as if it were the final one, to see what the model would predict "so far."

Lookup table

The model's stored row of numbers for each token, used as the starting version of every row before any layer runs.

Neural network

The general name for an arrangement of weighted sums stacked in stages, of which a language model is one kind. The name is historical and carries no other meaning here.

Quota

A conceptor's average gain across all directions; a summary of how much of the space it keeps.

Residual stream

The row of numbers per token position that flows through all the layers and is updated by each; the "shared memory" of the model.

Steering

Editing activations at a chosen layer while the model runs (typically by adding an arrow times a strength, or by applying a concepton) to change its behavior, without changing any weights.

Summary row

In this article, the single row used to represent a whole sentence: its row at the sentence's final period, at the chosen layer.

Token

The unit the model actually reads: usually a word, sometimes a piece of a word or a punctuation mark. GPT-2 knows 50,257 of them.

Unit, bank of units

A unit is one weighted sum with a squash: many inputs, one output number. A bank is many units side by side reading the same input, so that a row of 768 numbers comes out as a new row of 768.

Weight (knob, parameter)

One of the fixed numbers set during training, almost all of them coefficients in weighted sums. GPT-2 has 124 million; modern models hundreds of billions. Frozen after training.

Sources

Triantafyllopoulos, I., et al. (in preparation). *The Geometry of Conceptors in Neural-Activation Spaces*. The central source: the pooling decomposition, the bipolar/unipolar comparison, the two-sided amplification results (Table 4), the retention results (Table 2), the denoising and quota experiments, and the overlap measurements.

Triantafyllopoulos, I., Cho, Y.-M., Tao, R., Miao, M. M., Rai, S., Ungar, L., Guntuku, S. C., Ryant, N., & Sedoc, J. (2026). *Conceptors for Semantic Steering*. arXiv:2605.04980. The published precursor of the study above: conceptors as soft projections pooled over both poles of a concept, the quota as a layer diagnostic, and Boolean composition.

Abootorabi, M. M., Saghaian, A., Bazshoushtari, A., Rezaei, H., Hwang, E., Shwartz, V., Mousavi, P., & Abolmaesumi, P. (2026). *Steering Geometry: Validating Human Value Geometry in LLM Steering Space*. arXiv:2609.06289.

Gurnee, W., Sofroniew, N., Pearce, A., et al., & Lindsey, J. (2026). *Verbalizable Representations Form a Global Workspace in Language Models*. Anthropic; arXiv:2607.15495.

Jaeger, H. (2014). *Controlling Recurrent Neural Networks by Conceptors*. arXiv:1403.3369. The original concepton paper.

Marks, S., & Tegmark, M. (2024). *The Geometry of Truth: Emergent Linear Structure in LLM Representations of True/False Datasets*. COLM 2024; arXiv:2310.06824.

Postmus, J., & Abreu, S. (2024). *Steering Large Language Models using Conceptors: Improving Addition-Based Activation Engineering*. arXiv:2410.16314. First application of conceptors to language-model steering; source of the numbers in Figure 25(a).

Ravenda, F., Bahrainian, S. A., Raballo, A., & Mira, A. (2025). *Navigating through the hidden embedding space: steering LLMs to improve mental health assessment*. arXiv:2510.16373. Source of the numbers in Figure 18.

Skean, O., Arefin, M. R., Zhao, D., Patel, N., Naghiyev, J., LeCun, Y., & Shwartz-Ziv, R. (2025). *Layer by Layer: Uncovering Hidden Representations in Language Models*. ICML 2025; arXiv:2502.02013.

Zhu, F., Subramanian, A., Constant, A., Wang, C., Gupta, R., & Keller, C. J. (2026). *Interpretable Symptom Vectors for Depression in a Large Language Model*. arXiv:2609.01832.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners*. OpenAI. The GPT-2 model used for every experiment in this article.

Appendix: how the experiments in this article were run

This appendix is for readers who want to know exactly what was done, or to reproduce it. Nothing in it is needed to follow the article.

Model. GPT-2 small (124 million parameters; 12 layers; 768-number residual stream; 50,257-token vocabulary), using the publicly released weights, re-implemented in plain NumPy so that the row of numbers at every layer and position could be recorded and edited, and so that the attention (mixing) weights could be read out. All text was processed with OpenAI's original byte-pair tokenizer. Generation used greedy decoding (always the most likely next token) unless noted; the one randomly sampled continuation in Section 2 used temperature 0.8.

Word encodings (Figures 2–5). Rows were taken directly from the model's token-embedding lookup table for tokens that begin with a space (" cat", not "cat"). Similarity is cosine similarity. The map in Figure 4 is classical multidimensional scaling on cosine distances among 48 hand-chosen words. Each ruler direction is the average embedding of four or five words at one pole minus the average at the other; items are placed by the cosine between their unit-length embedding and the unit-length direction. Analogies use the standard "b – a + c, nearest neighbor excluding a, b, c" procedure.

Tiny model (Figure 8). Vocabulary of 36 tokens; 180 template sentences; a next-token predictor consisting of a 36×2 embedding table and a 2×36 weighted-sum layer with biases (180 parameters), trained by stochastic gradient descent (learning rate 0.5, batches of 64 word pairs, 3,000 steps) on the cross-entropy of the next word. The plotted score is $\exp(-\text{average cross-entropy})$, the geometric-mean probability assigned to the true next word.

Units, layers and mixing (Figures 9–12). Figure 9 is a miniature with invented 5-number rows and random weights, computed exactly as shown (tanh squash). Figures 10–12 use the real residual-stream tables for the prompt "The food was" at all 13 depths; the mixing weights are the layer-1 attention weights averaged over the 12 heads; "row before/after" are the actual layer-0 and layer-1 rows for *was*. The claim that appending tokens leaves earlier rows unchanged was checked by comparing the tables for "The food was" and "The food was really good" (maximum change 1.4×10^{-6} , floating-point noise). Typical entry size is the root-mean-square of the 768 entries.

Sentences. 48 positive/negative pairs and 16 neutral sentences written for this article, listed in the accompanying code. For every sentence, the residual-stream row at the position of the final token (the period), the "summary row," was recorded at the embedding output and after each of the 12 layers.

Layer analysis (Figures 13–15). At each layer, the arrow is the positive mean minus the negative mean; Cohen's d is the difference in group means of the projections onto the unit arrow, divided by the pooled standard deviation. Accuracy is leave-one-pair-out: each pair is held out in turn, both group means are recomputed from the rest, and each held-out sentence is assigned to the nearer mean. Figure 15 uses the arrow from pairs 1–36; the vertical axis is the leading direction of the residual variation after removing the arrow component; both axes are scaled by their pooled within-group standard deviation.

Ruler and steering (Figures 16–17, generation tables, Section 6.1 tables). The ruler uses layer 8 and the arrow from pairs 1–36, with scores measured from the midpoint of the two group means. Steering adds $\text{strength} \times \text{arrow}$ (the full-data arrow, length 20.1) to the residual stream at every position after layer 8, at every generation step. "Positive words" and "negative words" are fixed lists of 21 and 20 single-token adjectives. Entropy of the next-token distribution rises from 1.7 nats unsteered to 5.3 nats at strength +8, the quantitative face of the "breakdown." The prompt-instruction comparison and the "glowing five-star review" test use the same lists and the same arrow.

Conceptor (Figures 19–23). Fitted on the 96 layer-8 summary rows (72 source rows in the split experiments), centered on their mean; $\lambda = \text{trace}(R)/768$ (about 1.4). The 20-direction concepton keeps the 20 leading eigendirections with their gains. The direction table in Section 5.2 lists, for each of the leading eigendirections of the pooled centered covariance, the four sentences with the most negative and most positive coordinates along it. Amplification is $\text{row} \rightarrow \text{row} + t \cdot C_{20}(\text{row} - \text{center})$, with t chosen so that the root-mean-square displacement over source rows equals the arrow's displacement $D = 0.1 \times$ the root-mean-square distance of source rows from the center ($D \approx 3.3$), following the matched-budget protocol of Triantafyllopoulos and colleagues. Results are averaged over 20 random 36/12 splits; Figure 22 shows the first. The controls of Section 5.5 use the same splits and the same budget D : plain scaling is $\text{row} \rightarrow \text{row} + t(\text{row} - \text{center})$ with $t = 0.1$; the arrow-line projection, the hard 20-direction projection, the random 20-direction projection (a random orthonormal basis, fresh for each split) and the one-direction conceptors replace C_{20} in the amplification formula, each with its own t set by the same rule. The full (untruncated) concepton gives +1.05 and +0.99 in 20 of 20 splits, against +1.11 and +1.04 for C_{20} . Retention (Section 6.2) is the squared length of the filtered centered row over the squared length of the unfiltered one, averaged over the 24 held-out rows of each split and over the 20 splits. Held-out Cohen's d by layer (Figure 14) uses the same 20 splits. Denoising uses 30 random splits, 12 single-pair estimates per split, and cosine similarity to the held-out mean difference, for the concepton and for the hard projection onto its 20 directions; the two-sentence example of Section 5.6 fits the concepton on pairs 1–36 and uses the arrow from pairs 37–48 as the reference, with steering arrows rescaled to twice the full-data arrow's length for the comparison. The Boolean figure uses Jaeger's formulas: $\text{NOT } C = I - C$; $A \text{ AND } B = (A^{-1} + B^{-1} - I)^{-1}$; $A \text{ OR } B = \text{NOT}(\text{NOT } A \text{ AND } \text{NOT } B)$. The "distinctness" numbers in Section 6.2 are cosine similarities between two centered summary rows before filtering, after projection onto the arrow's line, after the 20-direction concepton, and after a random 20-direction projection.

Profile (Figure 26). The opinion conceptor is the 20-direction conceptor fitted on the 72 source rows; the fact conceptor keeps the 12 leading directions of a conceptor fitted on the 16 neutral sentences. For each test sentence, the reported share is the squared length of the filtered centered row divided by the squared length of the unfiltered centered row, each conceptor using its own center.

Published numbers. Figure 18 redraws the counts in Figure 2A of Ravenda et al. (2025). Figure 25(a) redraws Table 1 of Postmus and Abreu (2024) for GPT-J; Figure 25(b) and the retention table redraw Tables 4 and 2 of Triantafyllopoulos et al. No figure from any paper is reproduced; only reported numbers are re-plotted, with attribution.