Data, AI, and Society Lecture 2

How LLMs Work

Kenneth W. Regan
University at Buffalo (SUNY)

CSE199 Freshman Seminar, Fall 2026

Large Language Models

- First popularly identified as **chatbots**...then as **search extenders**...
- Now can do advanced reasoning tasks: coding—at systems and API levels!—and research mathematics.
- Their **Transformer** architecture with **Attention** mechanism originally came out of language translation.
- Big improvements in Google Translate starting 10 years ago presaged what we're seeing now.
- Show "8 Hours Without Internet" examples from 2024, now.
- LLMs are not the only way to do AI, but they more than suffice.
- We will cover their structure and ingredients—and what implications stem from them.

Can we hope to keep control over AI?

- **No** because AI is modeled on our brain function and, teaming with vastly greater computing power, will soon surpass us.
- **Yes** because AI (still) operates within a simple architecture of **linear functions** alternating with nonlinear **threshold tests** in a repetitive series of **layers**.

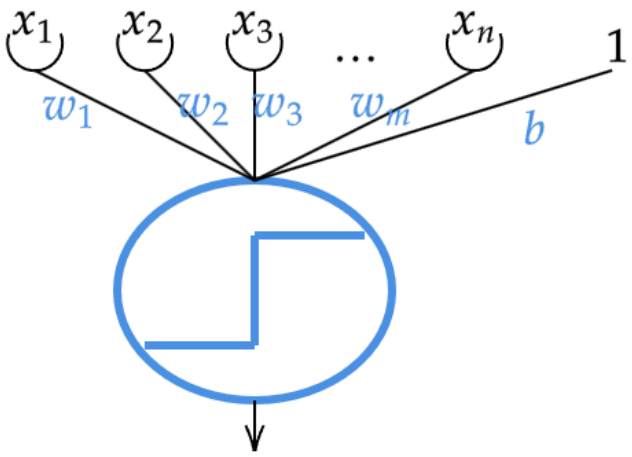
Defining some terms here: A **threshold gate** first computes an affine linear function of some number m of inputs $x_1, \ldots . x_m$:

$$u = w_1 x_1 + w_2 x_2 + \cdots w_m x_m + b,$$

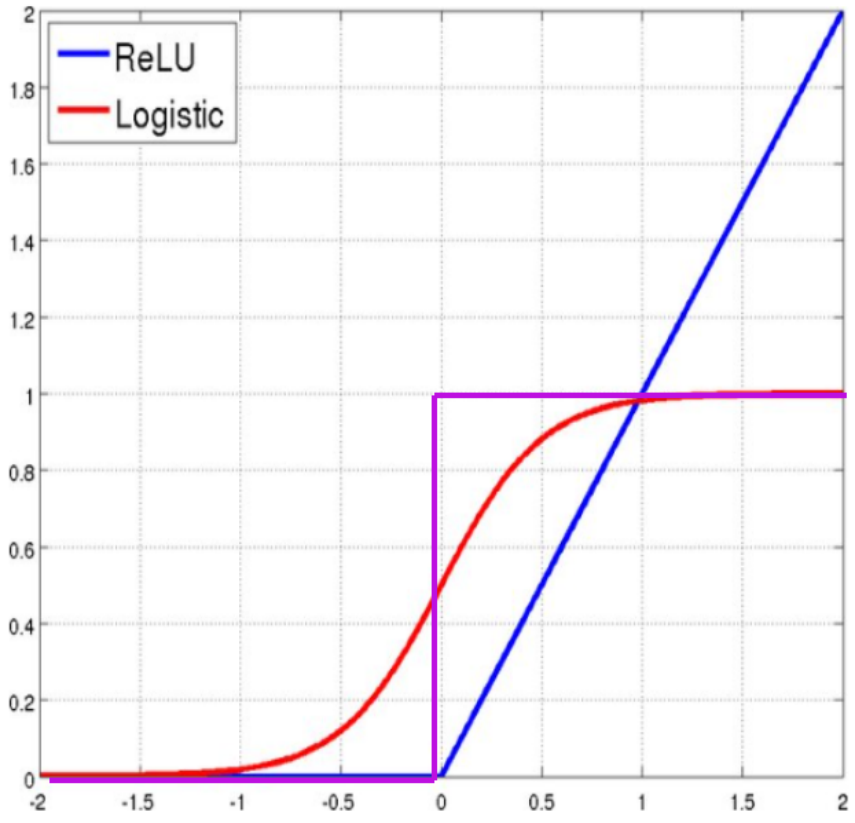
where the **weights** w_i and **bias** b belong to the gate. Then it tests $u > 0$? and outputs an **activation value** $a(u)$ accordingly.

A **perceptron** has just one **layer** of threshold gates that feed into a Boolean AND gate.

Threshold Gate and Alternative Activation Functions



$$\begin{cases} 1 \text{ if } \sum_{i=1}^{m} w_i x_i + b > 0 \\ 0 \text{ otherwise} \end{cases}$$

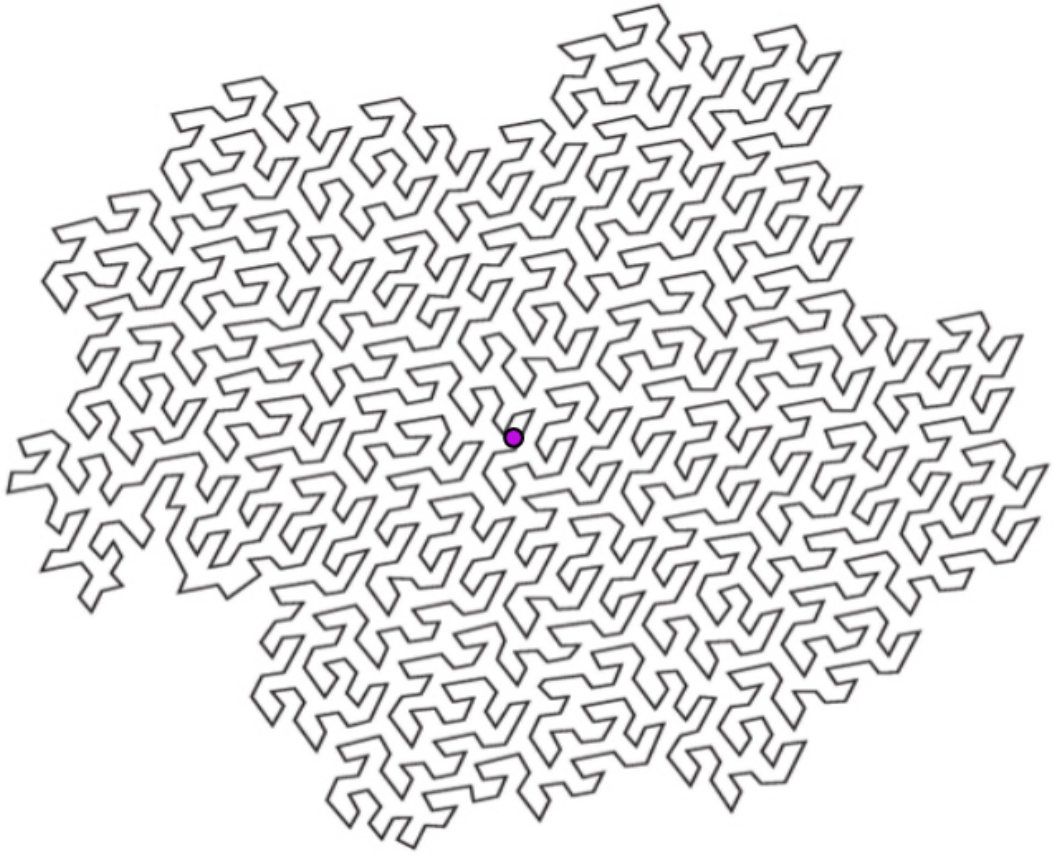


Modified from 2016 Robot World Cup paper Albani et al.,
https://link.springer.com/chapter/10.1007/978-3-319-68792-6_33

Can we keep advantage over AI?—cont'd

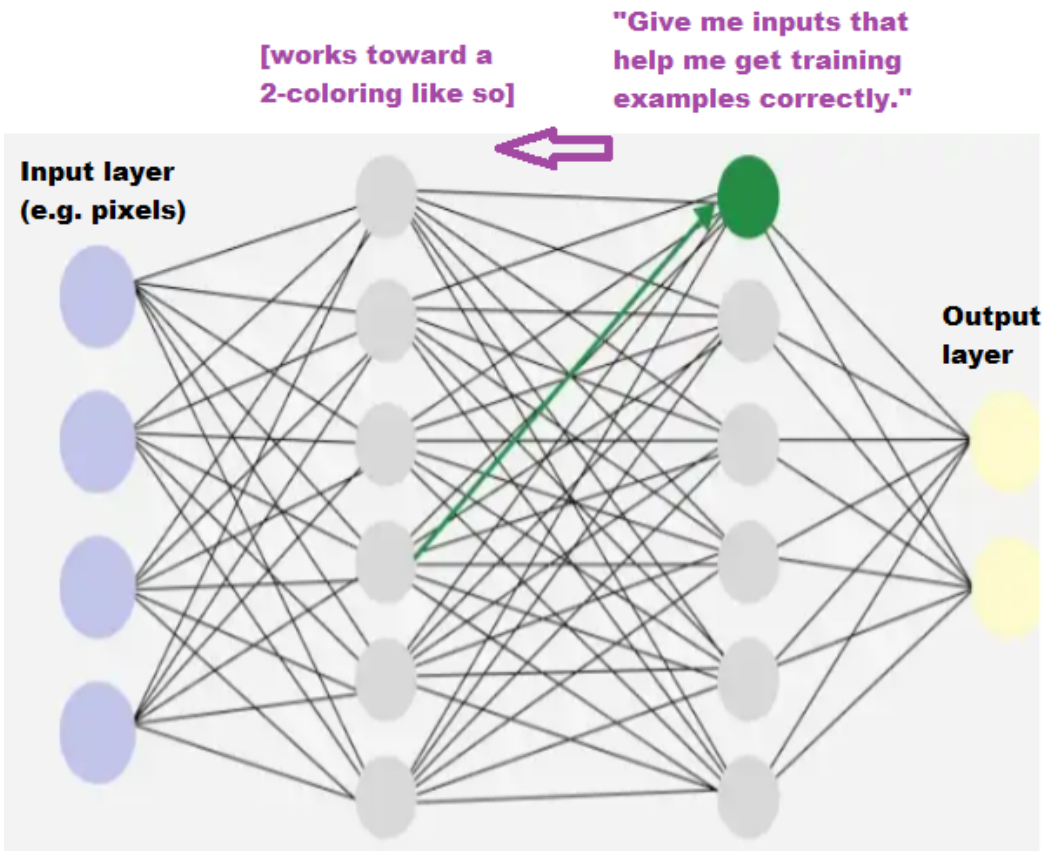
No because our brains cannot readily tell whether the point at right is inside or outside the closed curve---a **parity** task. Nor can a perceptron. This lack was a major cause of the "AI Winter" in the 1970s and 1980s.

Moreover, AI can employ more than one layer of threshold gates. By the 1990s we had algorithms and computing power to train **deep neural networks**.

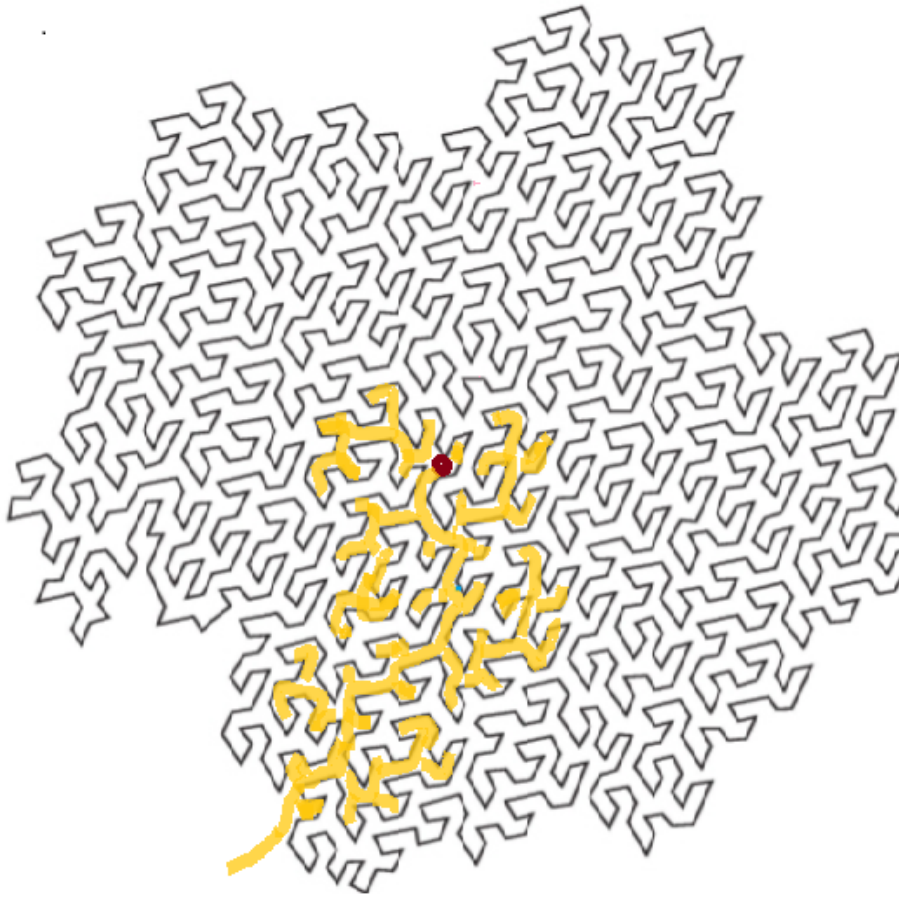


mathoverflow.net/questions/8521/nice-proof-of-the-jordan-curve-theore

Training Multiple Layers: Backpropagation



Modified from https://www.geeksforgeeks.org/deep-learning/layers-in-artificial-neural-networks-ann/
Getting 100% correctness may need a third threshold layer plus a Boolean AND layer.



Gradient descent algorithm minimizes error over any number of layers.

Utility, Loss, and Training

- Suppose we are training a neural net to distinguish cats and dogs.
- A training run might give it 500 pictures of each.
- The **utility** is how many it gets right.
- The **loss** is how many it gets wrong.
- After/during each run, weights are adjusted (helped by backprop), and changes that reduce the loss are kept.
- Squaring the loss is often a more effective measure, hence emphasis on loss.
- This initially *aligns* model performance with *prediction success*.
- Factoring in other positive objective criteria is more complicated—though the mechanism of **attention** (to come later) helps.

Representing Attributes By Vectors

Attributes a that are **independent** are represented by basis vectors $|a\rangle$ in an N-dimensional vector space that are **strictly orthogonal**.

- E.g.: $\{|0\rangle, |1\rangle\}$; $\{|\text{alive}\rangle, |\text{dead}\rangle\}$ have $n = 1$, $N = 2$.
- $\{|\clubsuit\rangle, |\diamondsuit\rangle, |\heartsuit\rangle, |\spadesuit\rangle\}$ have $N = 4$.

Schrödinger's cat is in a state that becomes either "alive" or "dead" when you observe it. It can't stay both.

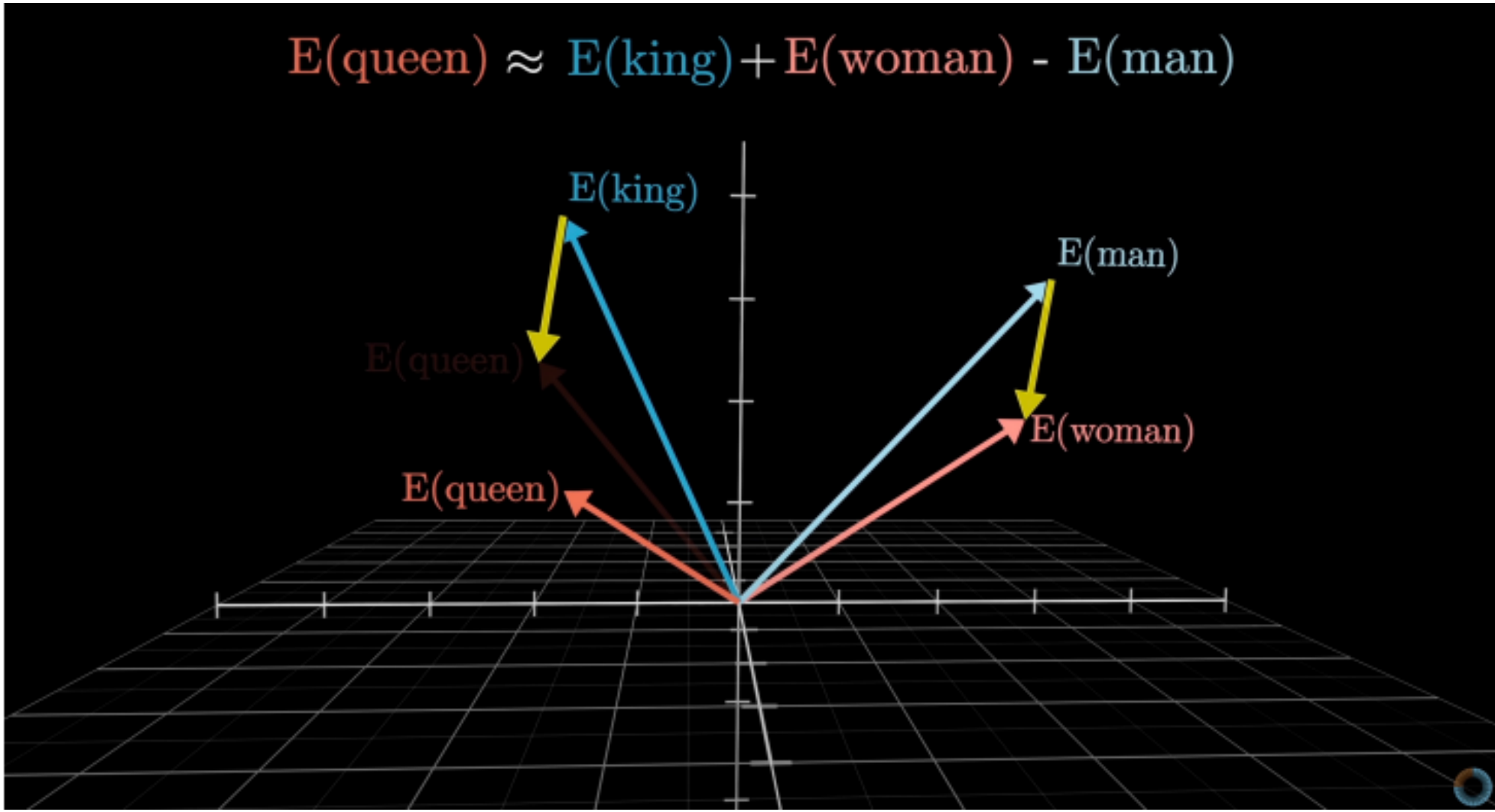
But with LLMs, you really can have "zombie cats." And you can make live cats be close-to-dead or make them point away from being dead with 9 times the magnitude.

How LLMs Represent Attributes By Vectors

- With LLMs, attributes w not strictly independent.
- A famous math theorem says that an n-dimensional space can accommodate a much bigger number N of vectors that are "almost orthogonal."
- E.g. the Qwen 3 1.7B model maps $N = 151,936$ tokens t into $n = 2,048$ space vectors v_t.
- Weighted sums of vectors blend their semantics.
- We may also move atomic attribute vectors around to express similarity—depending on context. E.g.:

In color, ♣ and ♠ are black; ♢ and ♡ are red.
In bridge, ♣ and ♢ are "minor"; ♡ and ♠ are "major."

Embedding Example (from 3 Blue 1 Brown)

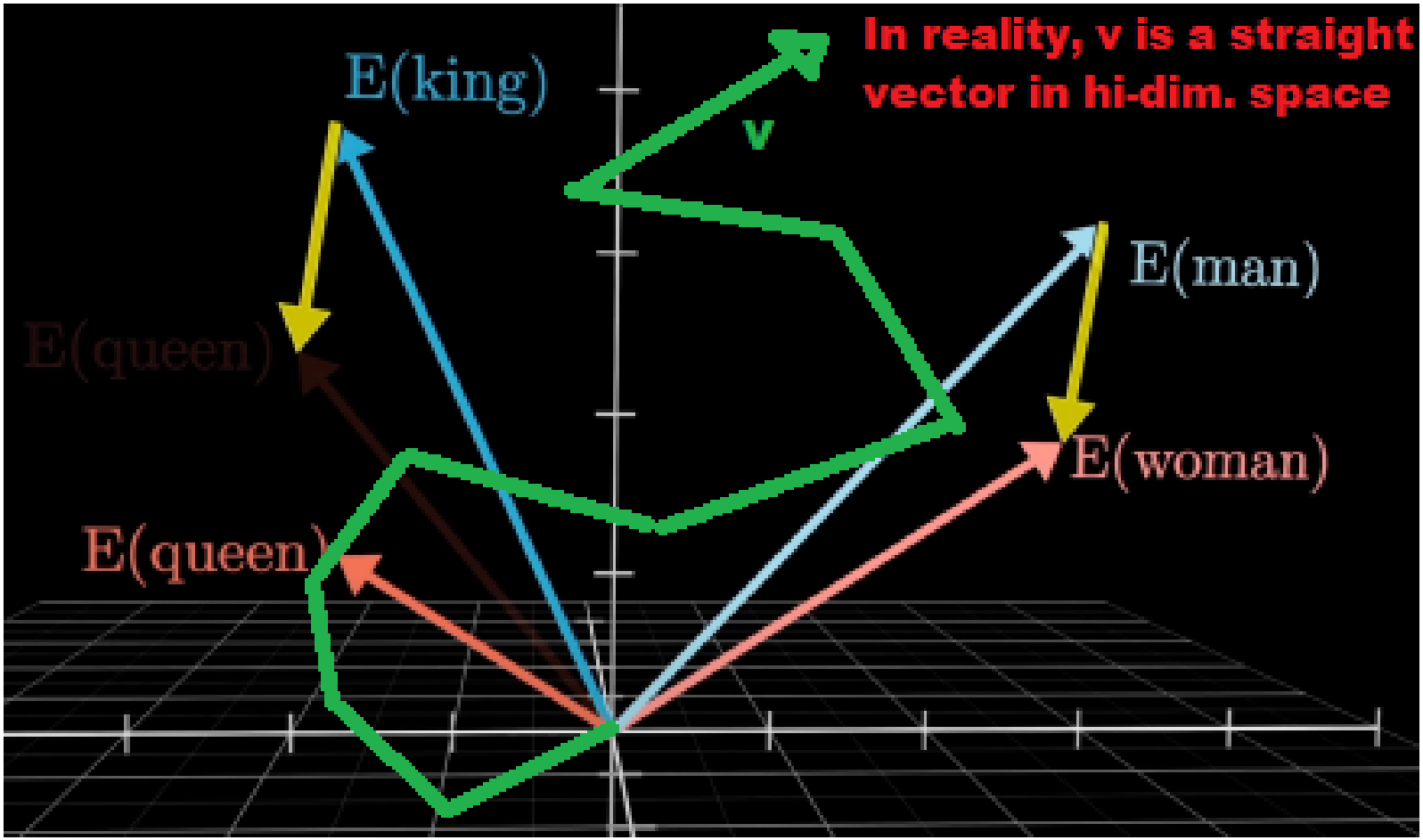


Some Numbers

- The **Qwen 3** model family, up to **3.7**, have $N = 151,936$ tokens.
- Split between English and Chinese, plus padding/control tokens.
- Embedding ("latent") spaces: $n = 1{,}024$ to $5{,}120$ dimensions.
- Long words can be split **id.io.sync.ra.tic.ally**. (Semantically should be **idio.syn.crat.ic.ally** but '**sync**' has other uses.)
- Basic English started with **only 850 words**.
- Then **200 international** and **1,000 technical** words were added.
- Wikipedia is approaching **3 million biographies**...but, people can be represented via kennings (as in Norse sagas).
- **Efficient to double-up many tokens, with-and-without leading spaces.**
- **Lots of wiggle room**—not only for resolving multiple word meanings by context but also for nuanced shading of meanings.

Training and Inferencing (simplified, OK for Qwen)

- **Training** moves N-many n-dim. vectors e_t into "nimble" positions.
- **Inferencing** moves a single **hidden state vector** v.
- The **logit score** for each t is the **dot-product** of e_t with the final v.



Training and Inferencing (cont'd)

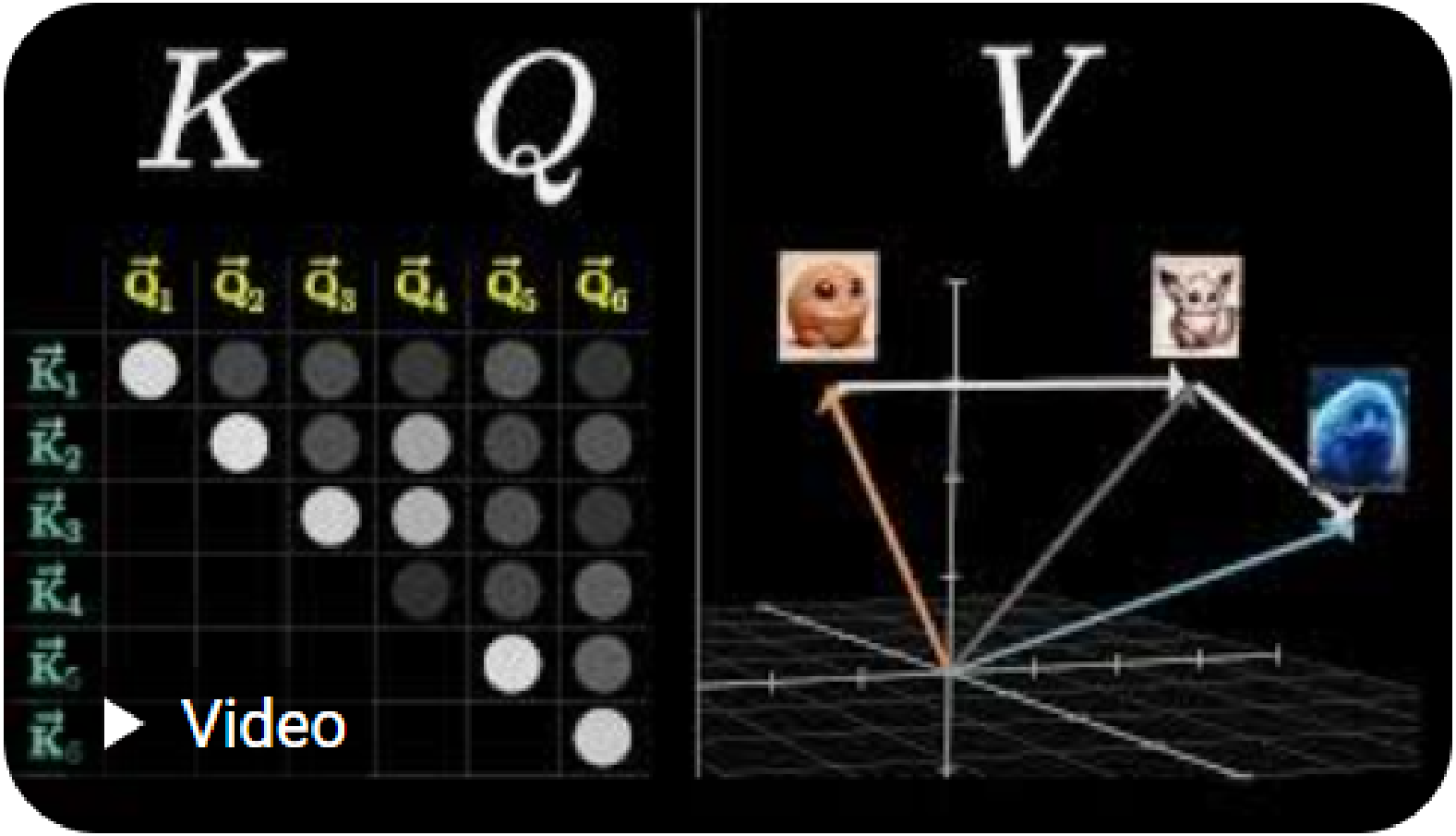
- Scores are converted to **probabilities** p_t of t being the next word/token (by an operation called **softmax**).
- In **training**, the e_t are moved so would increase p_T for the correct next-token T, over all "junctures" of a gazillion training texts.
- In **generation**, the next t is chosen—either by highest p_t or by *sampling* (i.e., by "rolling p_t-weighted dice").
  - Then t is added to prior text and inferencing starts afresh at the new juncture.
  - Looking multiple words ahead can influence v, still generating one word at a time.
  - The v is moved as each layer is applied, until the final layer settles it.
  - The embeddings e_t do not move.
- At each new juncture, the influence of prior text is updated.
- Then the new v is updated layer-by-layer.

Transformers and Attention

The main idea—which has existed for less than 10-15 years—is to picture words/tokens as psychologically **needing** and **giving validation**.

- Candidate next-words can **query** *earlier* words in a given text (which are known by that point)—up to a **context window** size T.
- T is generous now: 32K tokens midrange, 1M for Qwen-3.7 Max.
- Earlier words can provide **keys** for any later word.
- All-to-all connections otherwise—need lots of tarined weights and compute.
- Queries and keys project into a vector space of dimension $d \ll n$, in which dot-products again quantify the strength of association.
- So we multiply the $T \times d$ **key matrix** K and the $d \times T$ **query matrix** Q.

Transformers and Attention—cont'd



Multiplying KQ creates "hot spots" of gratified attention. Composing that on value vectors V updates the LLM's implicit understanding of words in the *latent* space.

Transformers and Attention—cont'd

- Zero out the lower triangle, so no illegal later-to-earlier associations.
- Apply softmax $\exp(\cdot/\sqrt{d})$ *to each column.*
- The resulting matrix U then updates the current **value matrix** V to nudge embedded token vectors. **Attention**(K, Q, V).

This also involves composing linear operators. The "needs validation" mechanism also fits well with backpropagation. Originally used for language translation.

Attention is also freely parallelizable. Multiple modes of attention can be tried—the ones that aren't useful "don't give a rise."

Is it still "All You Need"? Doing insanely well with mathematical theorem-proving right now...

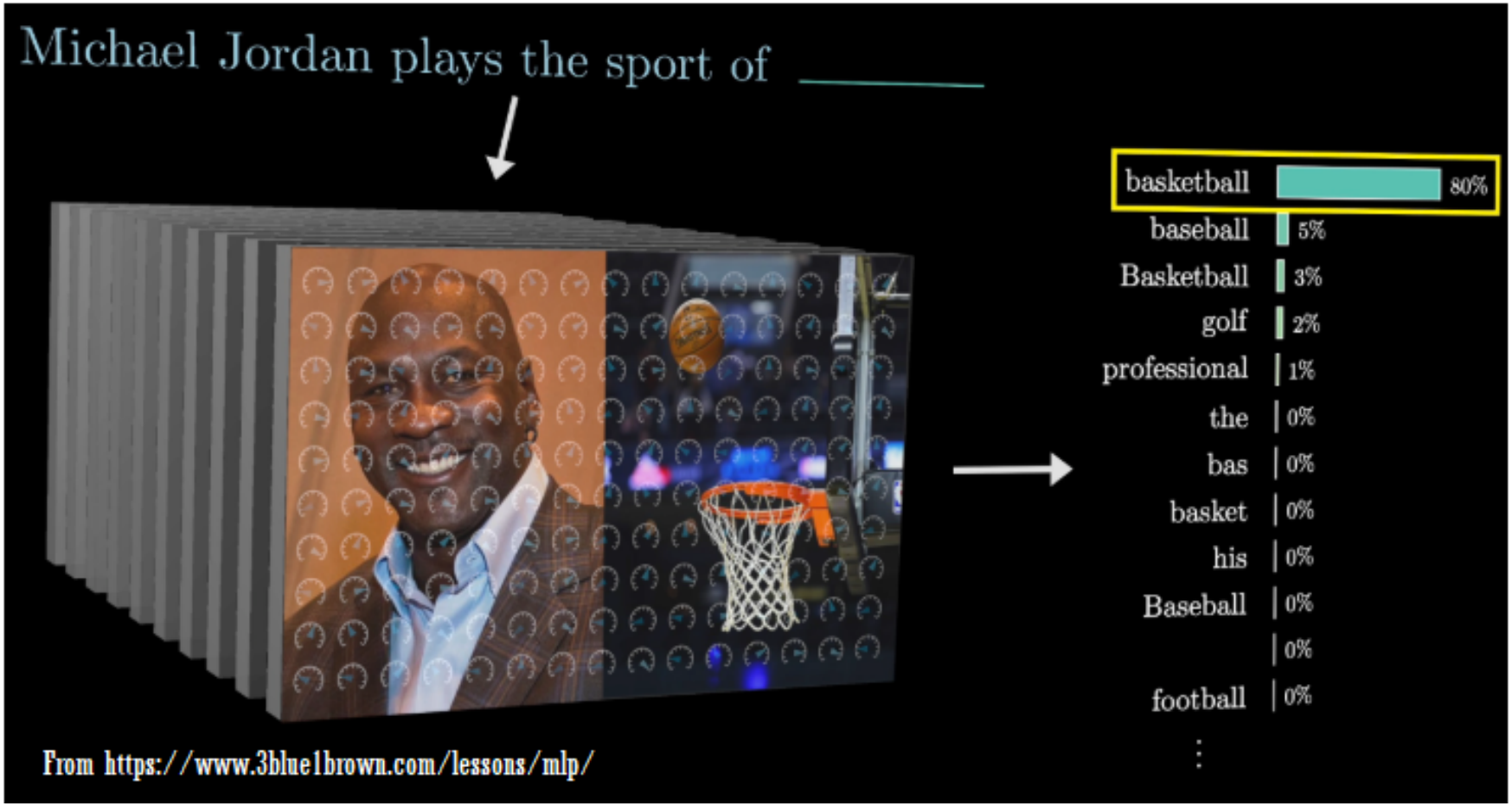
One Further Prediction Benefit

The transformer architecture simultaneously gives predictions for each word in the presented text based on all words before it.



Layers And Layers And Layers And Yet...

...only the outputs of the last (top) layer are used to give probabilities.



Has the network already extracted all useful info from its lower layers?

Extra: Why Token Values Are Called "Logits"

x_1	y_1
1	15.454
17	12.507
33	11.988
49	11.571
65	11.173
81	10.962
97	10.8
9,488 more rows Show all	
151921	−10.923
151934	−13.514

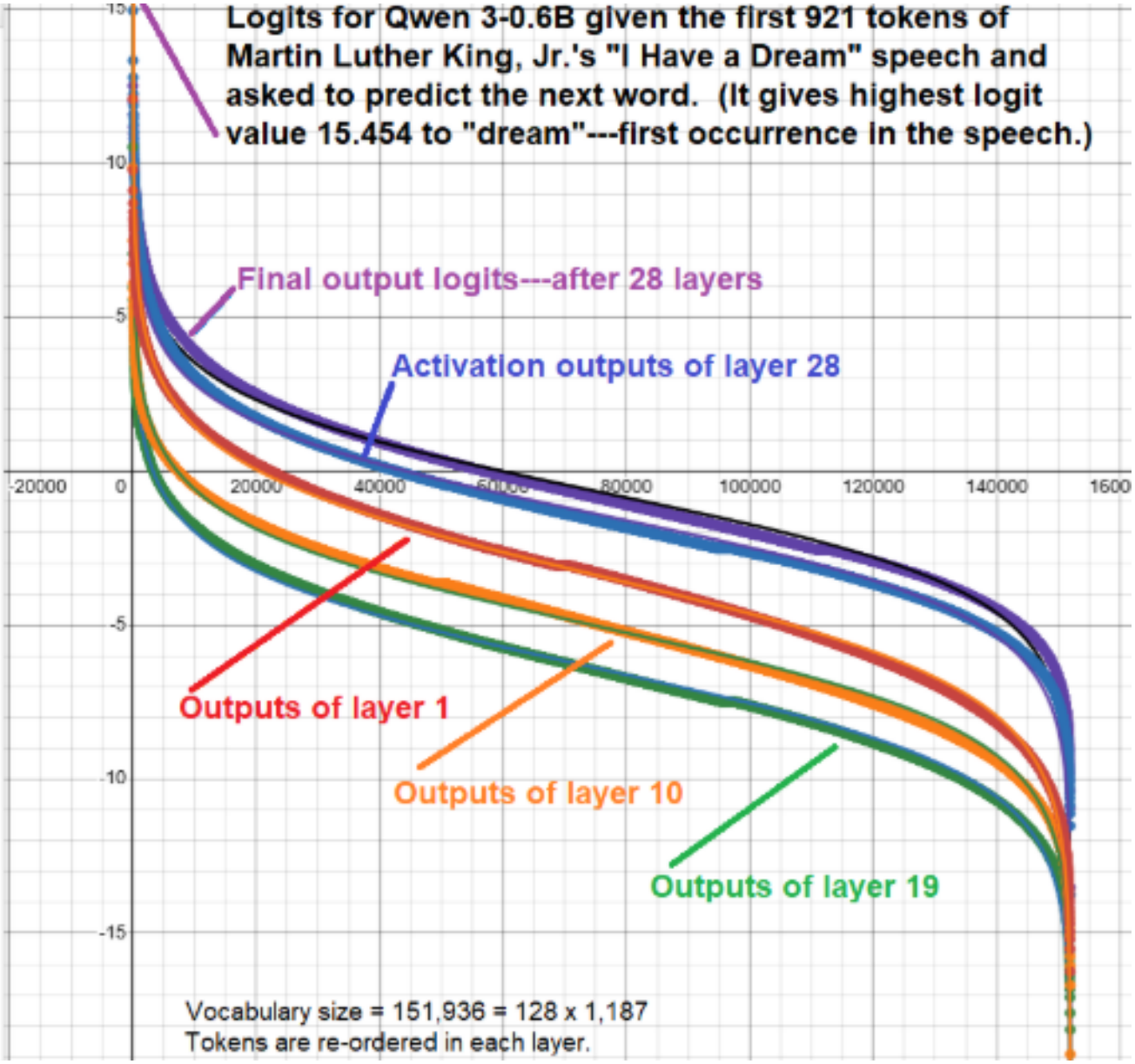
$$y_1 \sim a + b \ln\left(\frac{(151936 - x_1)}{(x_1)}\right)$$

REGRESSION PARAMETERS

$a = -0.68774$ $b = 1.58918$

STATISTICS $R^2 = 0.9884$ RESIDUALS e_1 plot

x_2	y_2



Steep At Top Makes Sense, But Why At Bottom?

1. `" dream"` = 15.454
2. `"strong"` = 14.774
3. `"heart"` = 14.454
4. `" heart"` = 14.409 (this duplication/spacing issue needs to be resolved too)
5. `"faith"` = 14.280
6. `" hope"` = 14.155 ...

Then things settle down so that by 200 or so tokens in, consecutive logit values differ only in the 0.001 place. But at the end:

151,931. `"Shu "` = -12.717
151,932. `"Die Dai "` = -12.871
151,933. `"Neng Hao "` = -13.330
151,934. `".EventType"` = -13.514
151,935. `"<|endoftext|>"` = -inf
151,936. `"<|im_end|>"` = -inf.

Granted, all the probabilities for these tokens obtained after softmax are microscopic. But why should the probability for "Die Dai " be $e^{0.459} = 1.58\ldots$ times higher than that for "Neng Hao "?