Compact Proofs

Measuring Quality of Understanding with a Compression-Based Metric

Team

Jason Gross

Rajashree Agrawal

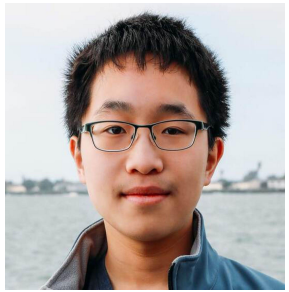
Lawrence Chan

Louis Jaburi

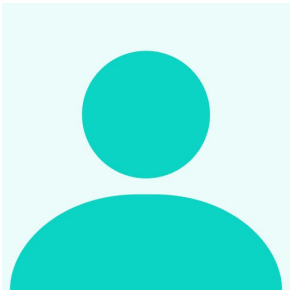
Ronak Mehta

Thomas Kwa

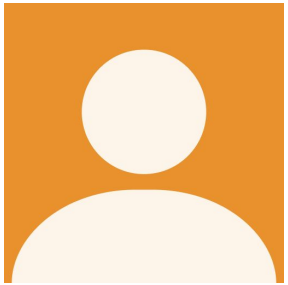
Chun Hei Yip



Euan Ong



Soufiane Nourbir



Alex Gibson

Why care about mech interp?

- **Understanding** models
- Model **evaluation** and **control**
- **Guarantees**
- Model **distillation**

Why care about metrics on mech interp?

- **Optimization**: If we can measure it, we can optimize for it
- **Automation**: Suppose we got AGI tomorrow, can we build a trustworthy, automated pipeline for discovering mechanistic explanation of model behavior?
- **Guarantees**: Maybe mech interp can help us generate explanations with the highest standard of trustworthiness: formal proofs

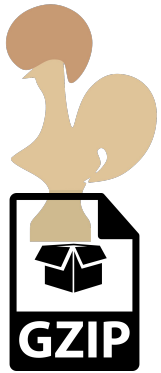
Formalizing proof length to quantify compression

Behavioral claim ≈ True behavior of the model

theorem statement

Proof = sound computation of worst-case error (divergence in behavior)

Length of proof = cost of running computation



https://www.svgrepo.com/svg/29028/gzip-file-format-variant https://calebstanford.com/2019/01/15/coq-vector-image/

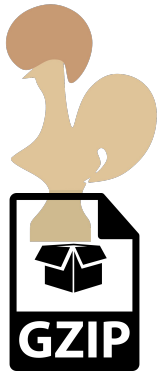
Formalizing proof length to quantify compression

Behavioral claim ≈ True behavior of the model

$$\underbrace{\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[f(y, M(x))\right] \geq b}_{\text{theorem statement}}$$

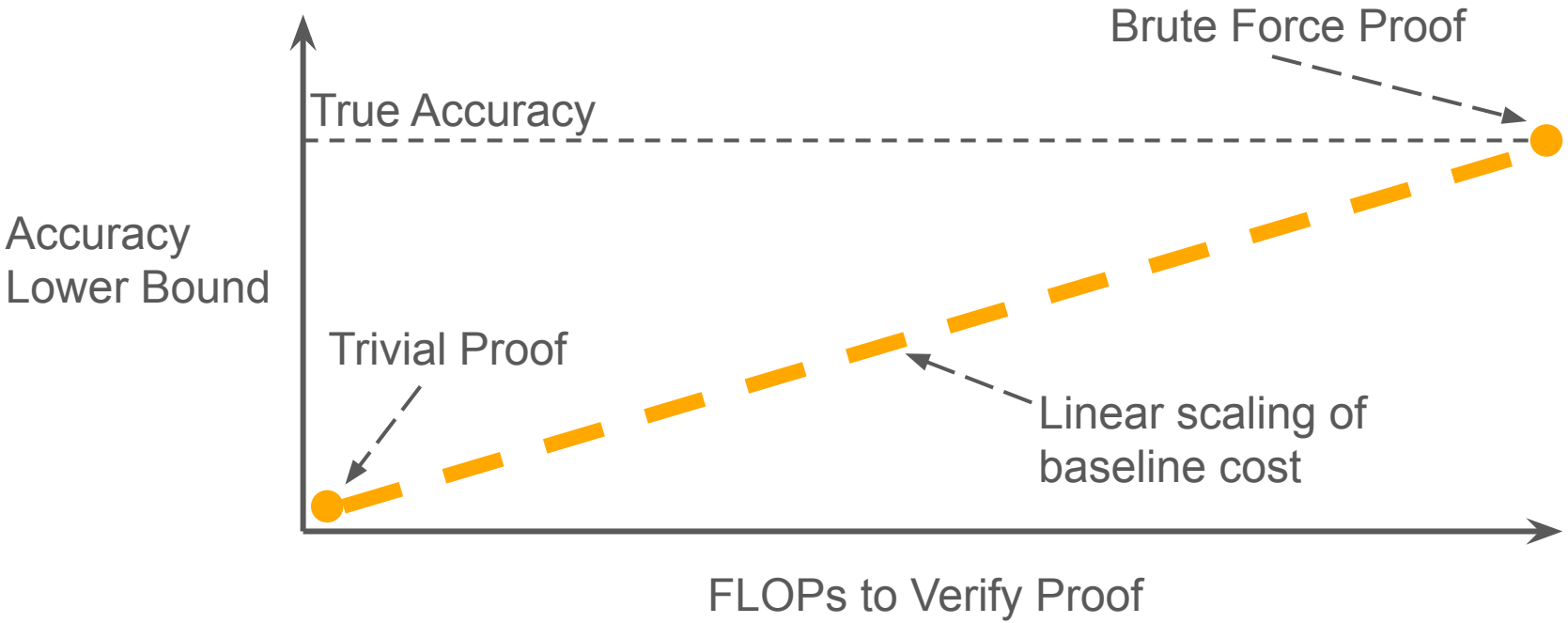
Proof = sound computation of worst-case error (divergence in behavior)

Length of proof = cost of running computation

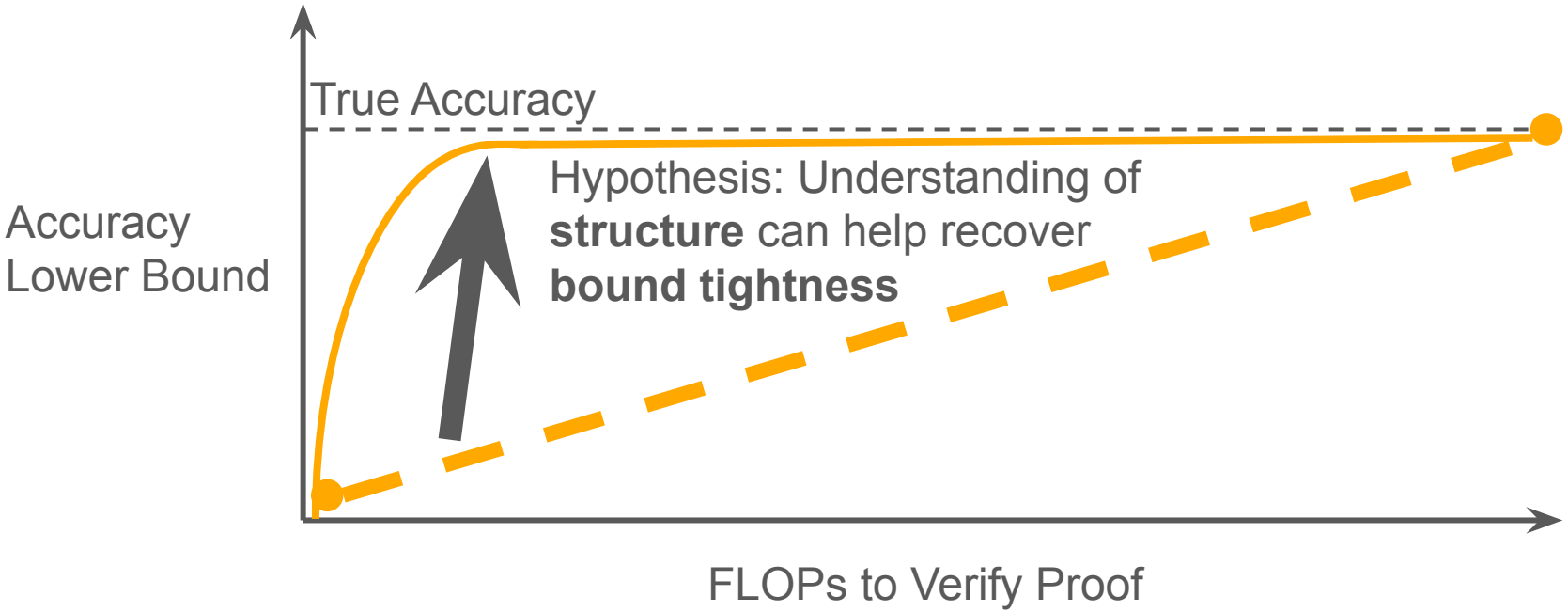


https://www.svgrepo.com/svg/29028/gzip-file-format-variant https://calebstanford.com/2019/01/15/coq-vector-image/

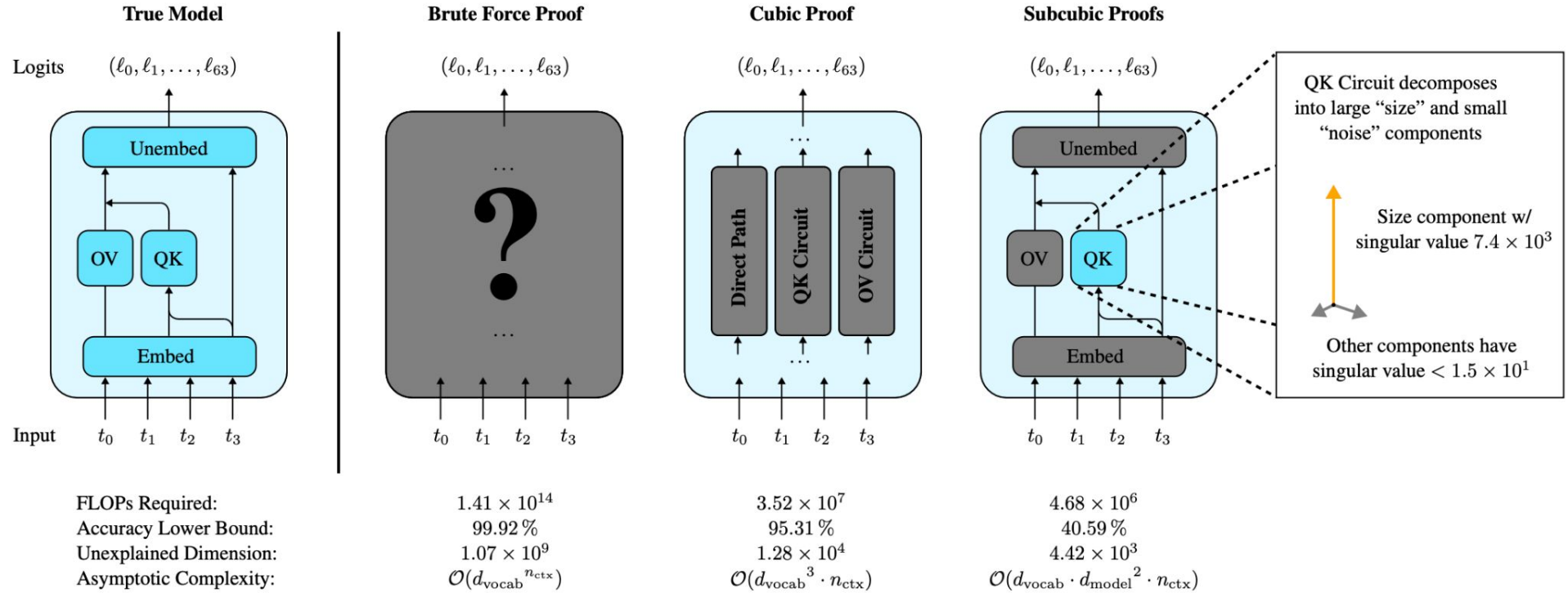
Quantifying the compute-cost of explanations



Does understanding improve upon the linear baseline?



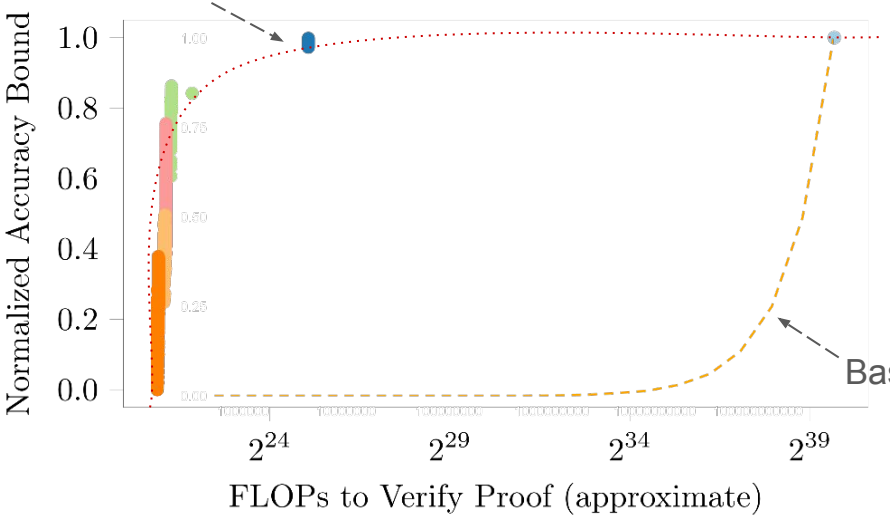
Proofs with varying mechanistic understanding



	True Model	Brute Force Proof	Cubic Proof	Subcubic Proofs
FLOPs Required:		1.41×10^{14}	3.52×10^7	4.68×10^6
Accuracy Lower Bound:		99.92 %	95.31 %	40.59 %
Unexplained Dimension:		1.07×10^9	1.28×10^4	4.42×10^3
Asymptotic Complexity:		$\mathcal{O}({d_{\text{vocab}}}^{n_{\text{ctx}}})$	$\mathcal{O}({d_{\text{vocab}}}^3 \cdot n_{\text{ctx}})$	$\mathcal{O}(d_{\text{vocab}} \cdot {d_{\text{model}}}^2 \cdot n_{\text{ctx}})$

We found an empirical “pareto frontier”

Pareto frontier from incorporating mechanistic understanding

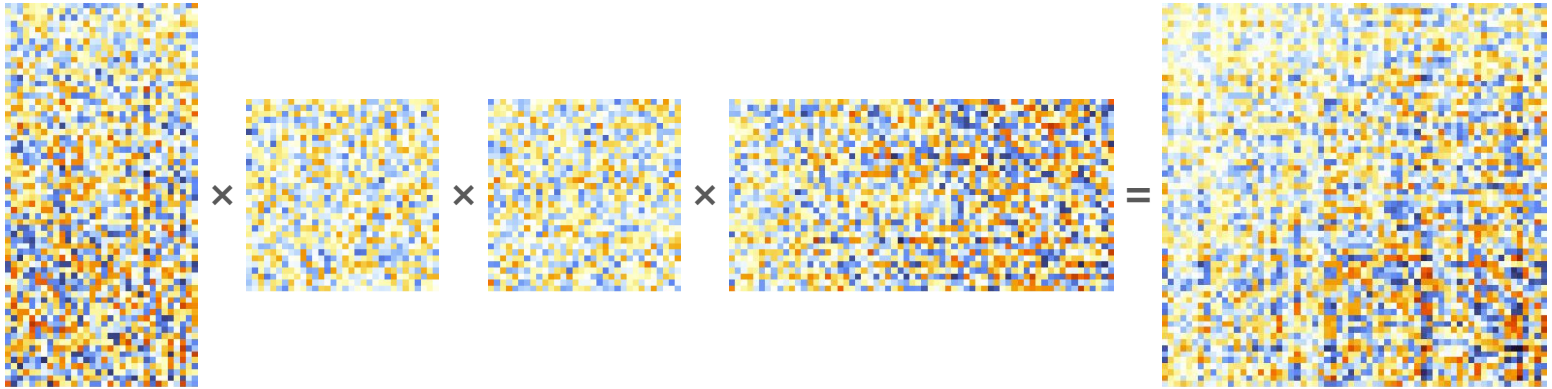


- brute force (acc: 0.9992 ± 0.0015)
- cubic (rel acc: 0.9853 ± 0.0038)
- subcubic (rel acc: 0.833 ± 0.011)
- attention-$d_{\text{vocab}}{d_{\text{model}}}^2$ (rel acc: 0.807 ± 0.013)
- direct-quadratic (rel acc: 0.663 ± 0.060)
- attention-$d_{\text{vocab}}{d_{\text{model}}}^2$, direct-quadratic (rel acc: 0.637 ± 0.060)

Baseline approach in log scale

Puzzle: Why does more structure not always mean better bound?

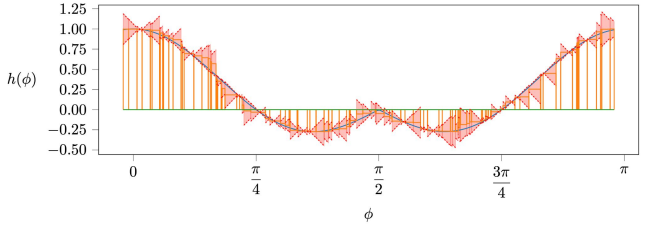
Compounding errors from lack of structure



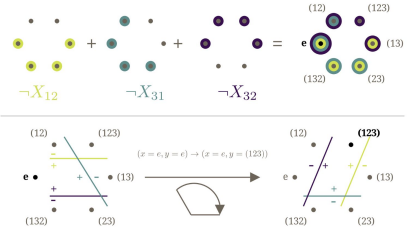
Approximation Strategy	Result	Complexity
(exact) max row diff	≈ 1.8	$(\mathcal{O}({d_{\text{vocab}}}^2 d_{\text{model}}))$
2 · (max abs value)	≈ 2.0	$(\mathcal{O}({d_{\text{vocab}}}^2 d_{\text{model}}))$
max row diff on subproduct	≈ 5.7	$(\mathcal{O}(d_{\text{vocab}} {d_{\text{model}}}^2))$
recursive max row diff	≈ 97	$(\mathcal{O}(d_{\text{vocab}} d_{\text{model}}))$

Applying Compact Proofs

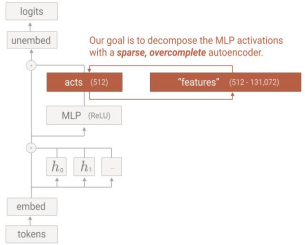
- Compressing MLPs (integration)



- Ground truth for comparing mech interp approaches (groups)



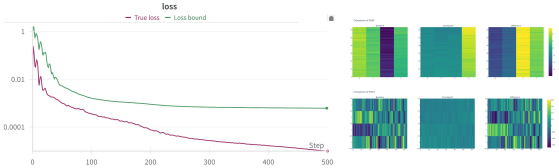
- Optimization targets for representation search (SAEs)

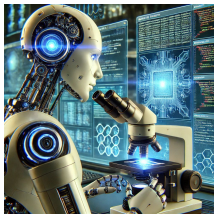


Some images from https://transformer-circuits.pub/2023/monosemantic-features

Open Problems for Scaling Compact Proofs

- Fix noise
  - Fine-tuning; or heuristic arguments; or sampling
- Suppress exponential in # layers
  - Toy model: induction heads
- Autoformalize proofs
  - AlphaProof
- Autointerp
  - Step 2: ???
- Step 3: Profit





Images by GPT-4o

https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/

Questions?

Extra Content

Proof length

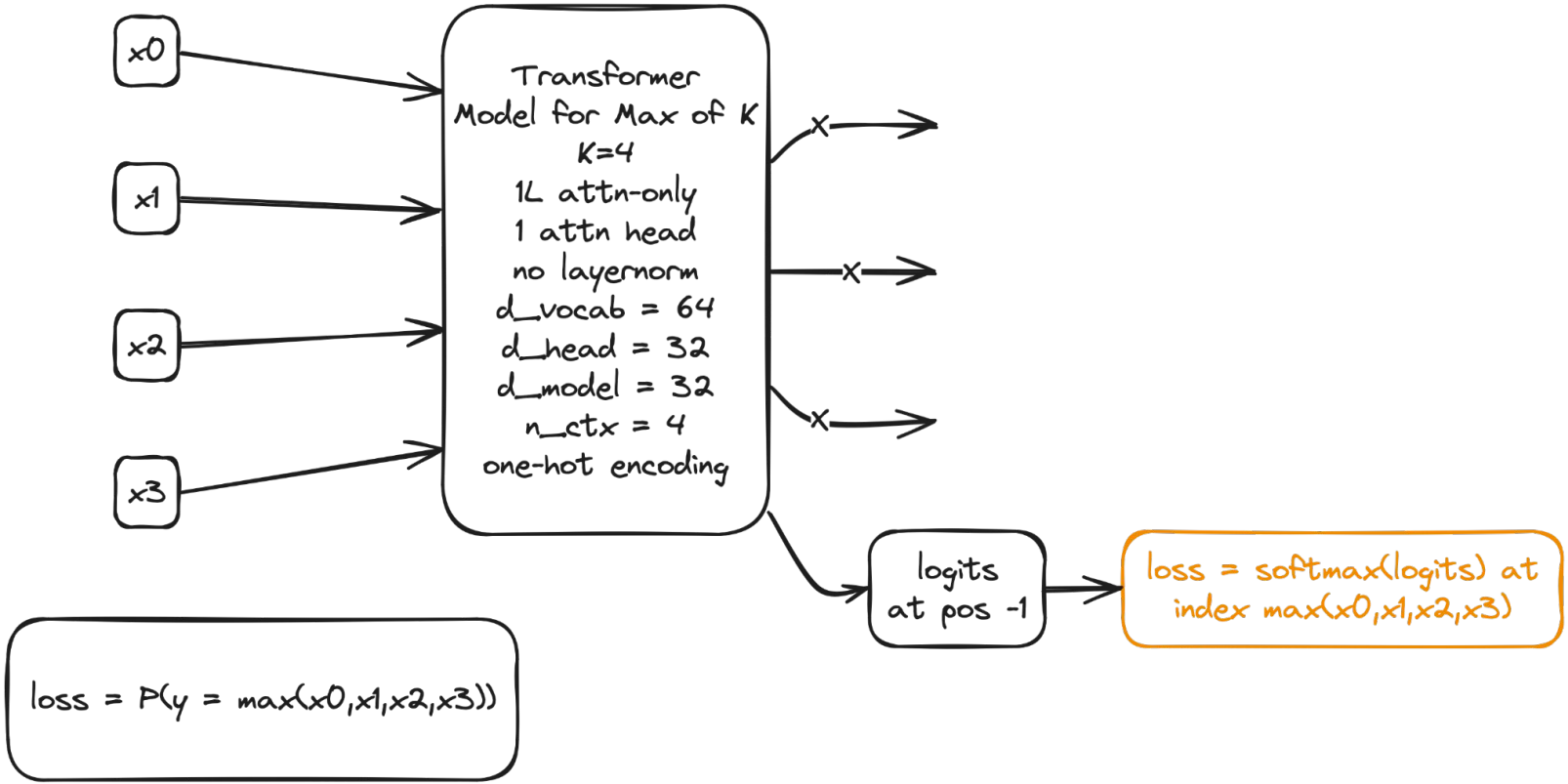
Theorem: $\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[f(y, M(x))\right] \geq b$

Our proofs consist of two components:

1. Proof that a particular computation C, when run with any model's weights, produces a valid bound on that model's performance
2. A trace of running C proving that $\mathrm{C}(M) = b$

In practice, proof length will be dominated by (2)

Case study: Max of K



Brute Force Proof Sketch

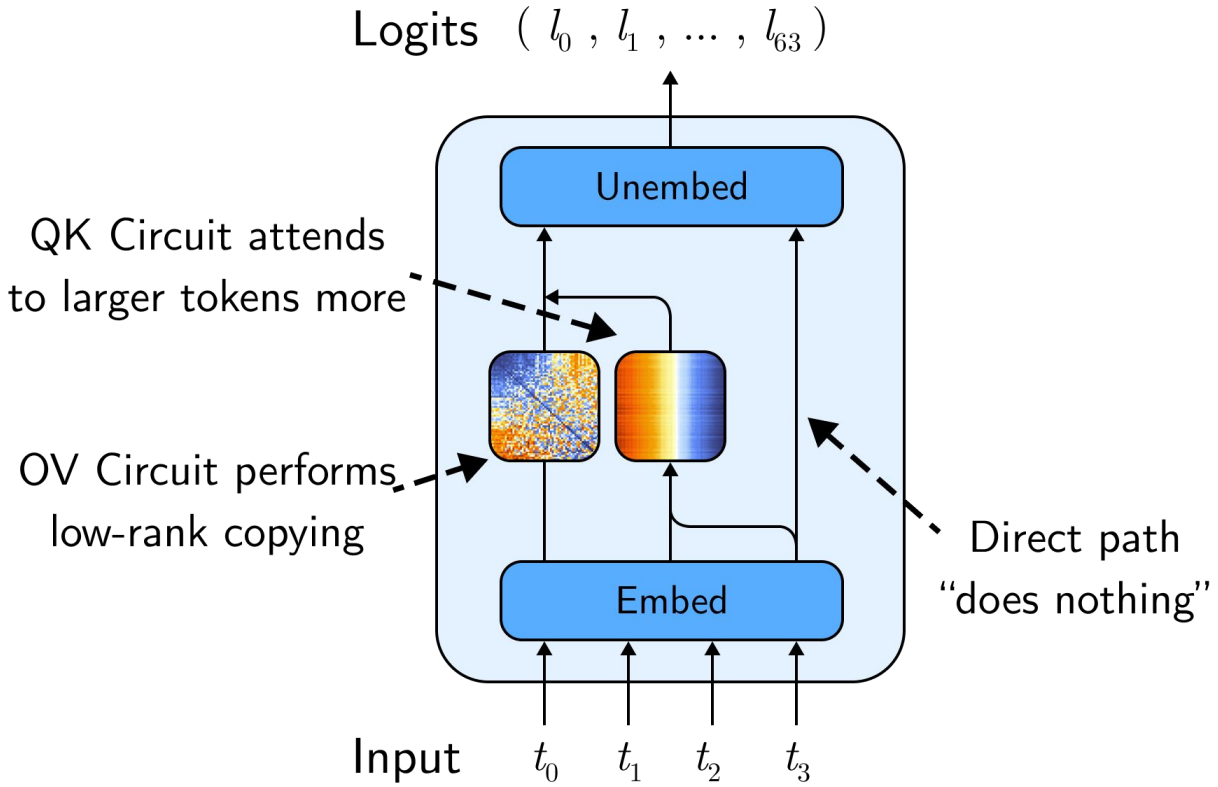
Theorem: $\mathbb{E}\left[\arg\max(M(x)[-1]) = \max_i x_i\right] > .9973$

Computation C:

```
def C(M):
  count = 0
  for x in possible_sequences:
    count+=(M(x)[..., -1, :].argmax(dim=-1) == x.max(dim=-1).values).sum().item()
  return count / len(possible_sequences)
```

1. Lemma: C produces a valid bound.
   Proof. Exercise
2. Lemma: $C(M) = .9973$
   Proof: By computation.

Attend More to Bigger Tokens & Copy



Results: Proof Size vs. Tightness of Bound

Description of Proof	**Complexity Cost**	**Bound**
Brute force	$\mathcal{O}(v^{k+1}kd)$	0.9992 ± 0.0015
Cubic	$\mathcal{O}(v^3k^2)$	0.9531 ± 0.0087
Sub-cubic	$\mathcal{O}(v^2 \cdot k^2 + v^2 \cdot d)$	0.820 ± 0.013
w/o mean+diff		0.488 ± 0.079
Low-rank QK	$\mathcal{O}(v^2k^2 + \underbrace{vd^2}_{\text{QK}} + \underbrace{v^2d}_{\text{EU\&OV}})$	0.795 ± 0.014
SVD only		0.406 ± 0.077
Low-rank EU	$\mathcal{O}(v^2k^2 + \underbrace{vd}_{\text{EU}} + \underbrace{v^2d}_{\text{QK\&OV}})$	0.653 ± 0.060
SVD only		$(3.38 \pm 0.06) \times 10^{-6}$
Low-rank QK&EU	$\mathcal{O}(v^2k^2 + \underbrace{vd^2}_{\text{QK}} + \underbrace{vd}_{\text{EU}} + \underbrace{v^2d}_{\text{OV}})$	0.627 ± 0.060
SVD only		$(3.38 \pm 0.06) \times 10^{-6}$
Quadratic QK	$\mathcal{O}(v^2k^2 + \underbrace{vd}_{\text{QK}} + \underbrace{v^2d}_{\text{EU\&OV}})$	0.390 ± 0.032
Quadratic QK&EU	$\mathcal{O}(v^2k^2 + \underbrace{vd}_{\text{QK\&EU}} + \underbrace{v^2d}_{\text{OV}})$	0.285 ± 0.036

Red-Teaming Cubic Proof

Insight: There is a single token we want to be paying attention to (the max)

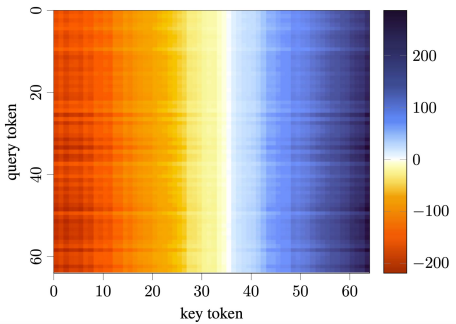
Convexity of Softmax:

(max token, non-max token)
⇒ score of (how bad it is to pay attention to,
how much attention is paid to it for each fixed query token)

(max token, query token)
⇒ sort non-max tokens by a combined score

If model succeeds when all non-max non-query tokens are A, it also succeeds when tokens are better than A.

Modulo handling positional encodings, this allows us to run the model on only d_vocab^3 sequences instead of d_vocab^4.



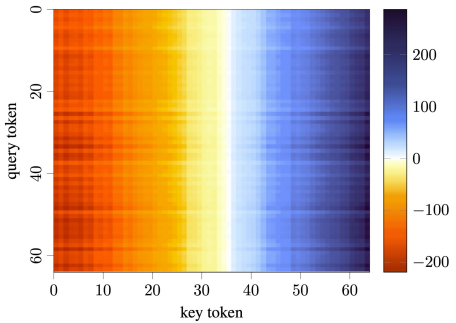
Convexity in PyTorch Code

Overview
Proofs Approach
Case Study: Max of 4
Applications: Modular Arithmetic and SAEs

Red-Teaming Sub-cubic Proof

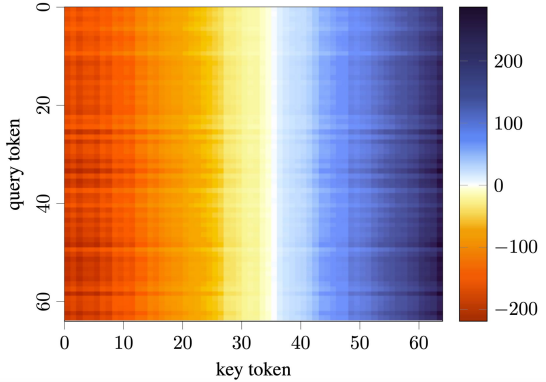
Insight: QK attention dominates OV

- “badness” ordering of tokens is largely independent of max token
- we can do a (slow) proof search procedure to determine “gap size” (max - largest non-max)
- for each query & max, we can use gap to bound how much attention we pay to non-max tokens
- we then must independently pick which token is worst to pa attention to (combined score is too expensive to compute exactly)

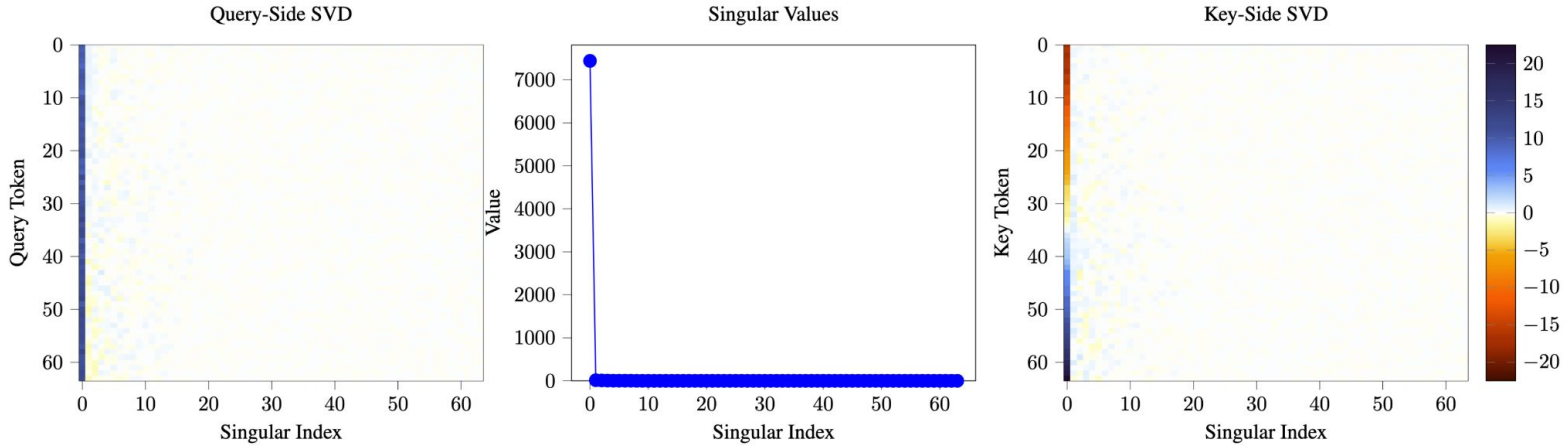


Red-Teaming $d\_vocab^2 d\_model \Rightarrow d\_vocab\ d\_model^2$

Attention Score (EQKE)

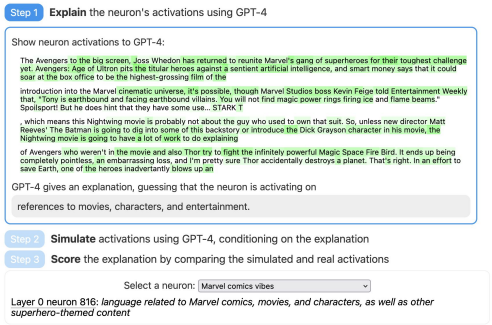


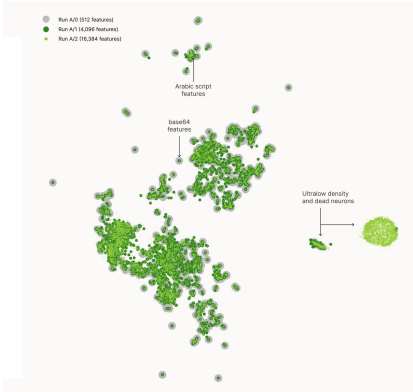
SVD

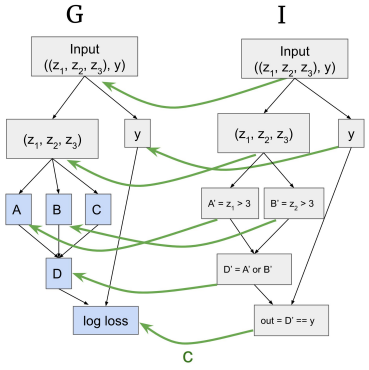


What we can contribute to SOTA

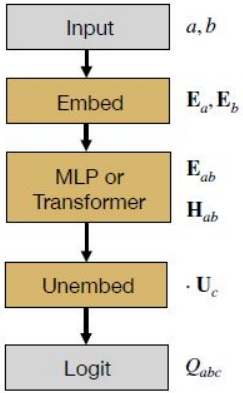
- Manual mech interp: where should we spend our research budget?
- SAEs: how do we reduce the dependence on human judgment?
- GPT explains GPT: we're hoping to use LLMs to formalize proofs
- Causal Scrubbing: we add rigor and a compression metric







Modular Arithmetic Interpretability



Step 1: Embed token a and b to a circle where $w_k = 2\pi k/p$ for some $k \in [1, 2 \cdots, p-1]$

$$a \to \mathbf{E}_a \equiv (\mathbf{E}_{a,x}, \mathbf{E}_{a,y}) = (\cos(w_k a), \sin(w_k a)), b \to \mathbf{E}_b \equiv (\mathbf{E}_{b,x}, \mathbf{E}_{b,y}) = (\cos(w_k b), \sin(w_k b))$$

Clock Algorithm

Step 2: compute the angle sum using multiplication.

$$\mathbf{E}_{ab} \equiv \begin{pmatrix} \mathbf{E}_{ab,x} \\ \mathbf{E}_{ab,y} \end{pmatrix} = \begin{pmatrix} \mathbf{E}_{a,x}\mathbf{E}_{b,x} - \mathbf{E}_{a,y}\mathbf{E}_{b,y} \\ \mathbf{E}_{a,x}\mathbf{E}_{b,y} + \mathbf{E}_{a,y}\mathbf{E}_{b,x} \end{pmatrix} = \begin{pmatrix} \cos(w_k(a+b)) \\ \sin(w_k(a+b)) \end{pmatrix}$$

$$\mathbf{H}_{ab} = \mathbf{E}_{ab}$$

Pizza Algorithm

Step 2.1: compute the vector mean.

$$\mathbf{E}_{ab} = (\mathbf{E}_a + \mathbf{E}_b)/2 = (\cos(w_k a) + \cos(w_k b), \sin(w_k a) + \sin(w_k b))/2$$

Step 2.2: using $\mathbf{E}_{ab}$ and nonlinearities to compute $\mathbf{H}_{ab}$

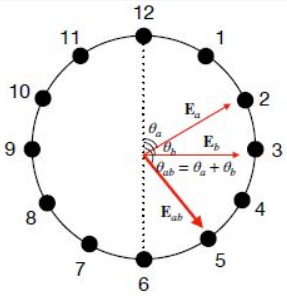
$$\mathbf{H}_{ab} = |\cos(w_k(a-b)/2)|(\cos(w_k(a+b)), \sin(w_k(a+b)))$$

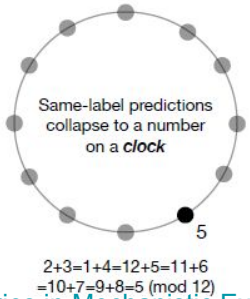
Step 3: score possible outputs c using a dot product.

$$Q_{abc} = \mathbf{U}_c \cdot \mathbf{H}_{ab}\ ,\ \ \mathbf{U}_c \equiv (\mathbf{E}_{c,x}, \mathbf{E}_{c,y}) = (\cos(w_k c), \sin(w_k c))$$

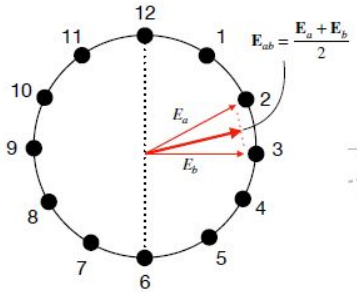
$$Q_{abc}(\text{Clock}) = \cos(w_k(a+b-c))$$

$$Q_{abc}(\text{Pizza}) = \left|\cos(w_k(a-b)/2)\right| \cos(w_k(a+b-c))$$





2+3=1+4=12+5=11+6
=10+7=9+8=5 (mod 12)



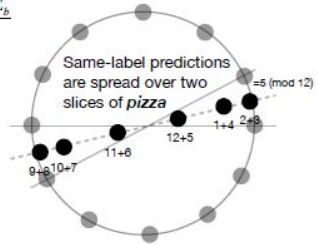


Image from [2306.17844] The Clock and the Pizza: Two Stories in Mechanistic Explanation of Neural Networks

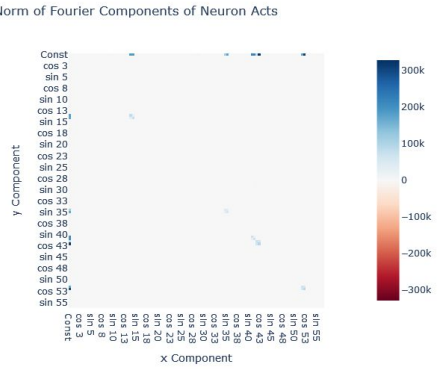
Budget $\mathcal{O}(p^2 \text{d\_mlp} + p^3)$

We need $\mathcal{O}(p^2\ \text{d\_mlp})$ just to multiply all the matrices

With this budget, we can compute everything except the MLP noise by brute force

The only interp is “unembed is low-rank”

“2D fourier basis” ⇒ requires $\mathcal{O}(p^2\ \text{d\_computation})$ to validate



https://colab.research.google.com/drive/1F6_1_cWXE5M7WocUcpQWp3v8z4b1jL20#scrollTo=UuI6zMAxjU1I&line=1&uniqifier=1

Explaining Pizza MLP in $\mathcal{O}(p \text{ d\_mlp})$

Enough to brute force the MLP behavior (w/o noise)

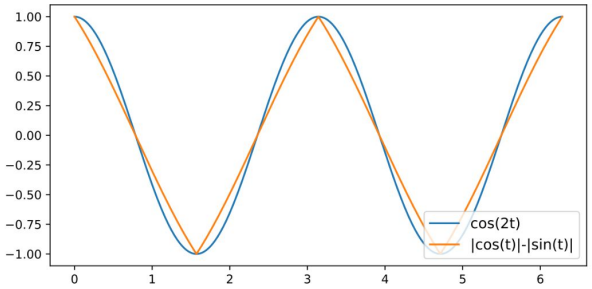


Figure 8: $|\cos(t)| - |\sin(t)|$ is approximately $\cos(2t)$ for any $t \in \mathbb{R}$

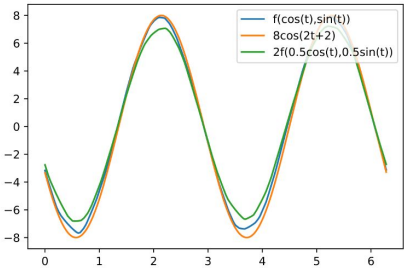


Figure 26: $f(\cos(t), \sin(t))$ well-approximates $8\cos(2t+2)$.

L.4 Approximation

Everything becomes much clearer after realigning the matrices. For a pizza and its two corresponding principal embedding / unembedding dimensions, $W'_E[a] + W'_E[b] \approx \cos(w_k(a-b)/2) \cdot (\cos(w_k(a+b)/2), \sin(w_k(a+b)/2))$ will be mapped by realigned W_1 into its corresponding columns (which are different for every pizza), added with b_1 and apply ReLU. The result will then be mapped by the realigned W_2, added with *realigned* b_2, and finally multipled by $(\cos(w_k c), \sin(w_k c))$.

For the first two principal dimensions, realigned W_1 has 44 corresponding columns (with coefficients of absolute value > 0.1). Let the embedded input be $(x, y) = W'_E[a] + W'_E[b] \approx \cos(w_k(a-b)/2) \cdot (\cos(w_k(a+b)/2), \sin(w_k(a+b)/2))$, the intermediate columns are

$\text{ReLU}([0.530x - 1.135y + 0.253, -0.164x - 1.100y + 0.205, 1.210x - 0.370y + 0.198, -0.478x - 1.072y + 0.215, -1.017x + 0.799y + 0.249, 0.342x - 0.048y + 0.085, 1.149x - 0.598y + 0.212, -0.443x + 1.336y + 0.159, -1.580x - 0.000y + 0.131, -1.463x + 0.410y + 0.178, 1.038x + 0.905y + 0.190, 0.568x + 1.188y + 0.128, 0.235x - 1.337y + 0.164, -1.180x + 1.052y + 0.139, -0.173x + 0.918y + 0.148, -0.200x + 1.060y + 0.173, -1.342x + 0.390y + 0.256, 0.105x - 1.246y + 0.209, 0.115x + 1.293y + 0.197, 0.252x + 1.247y + 0.140, -0.493x + 1.252y + 0.213, 1.120x + 0.262y + 0.239, 0.668x + 1.096y + 0.205, -0.487x - 1.302y + 0.145, 1.134x - 0.862y + 0.273, 1.143x + 0.435y + 0.171, -1.285x - 0.644y + 0.142, -1.454x - 0.285y + 0.218, -0.924x + 1.068y + 0.145, -0.401x + 0.167y + 0.106, -0.411x - 1.389y + 0.249, 1.422x - 0.117y + 0.227, -0.859x - 0.778y + 0.121, -0.528x - 0.216y + 0.097, -0.884x - 0.724y + 0.171, 1.193x + 0.724y + 0.131, 1.086x + 0.667y + 0.218, 0.402x + 1.240y + 0.213, 1.069x - 0.903y + 0.120, 0.506x - 1.042y + 0.153, 1.404x - 0.064y + 0.152, 0.696x - 1.249y + 0.199, -0.752x - 0.880y + 0.106, -0.956x - 0.581y + 0.223])$.

For the first principal unembedding dimension, it will be taken dot product with

$[1.326, 0.179, 0.142, -0.458, 1.101, -0.083, 0.621, 1.255, -0.709, 0.123, -1.346, -0.571, 1.016, 1.337, 0.732, 0.839, 0.129, 0.804, 0.377, 0.078, 1.322, -1.021, -0.799, -0.339, 1.117, -1.162, -1.423, -1.157, 1.363, 0.156, -0.165, -0.451, -1.101, -0.572, -1.180, -1.386, -1.346, -0.226, 1.091, 1.159, -0.524, 1.441, -0.949, -1.248]$.

Call this function $f(x, y)$. When we plug in $x = \cos(t), y = \sin(t)$, we get a function that well-approximated $8\cos(2t+2)$ (Figure 26). Therefore, let $t = w_k(a+b)/2$, the dot product will be approximately $8|\cos(w_k(a-b)/2)|\cos(w_k(a+b)+2)$, or $|\cos(w_k(a-b)/2)|\cos(w_k(a+b))$ if we ignore the phase and scaling. This completes the picture we described above.

[2306.17844] The Clock and the Pizza: Two Stories in Mechanistic Explanation of Neural Networks

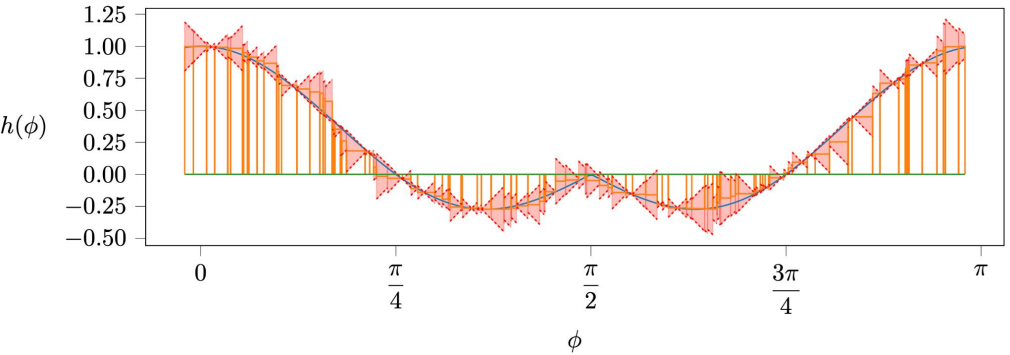
Budget $\mathcal{O}(p$ + d_mlp): Numerical Integration

sublinear in total parameter count ⇒ explanation w/ understanding

$\mathcal{O}$(p + d_mlp) bounds ⇒ evidence that numerical integration is "what's really going on"

Absolute error: 0.6 (vs 0.04 empirical error; vs 0.85 baseline)

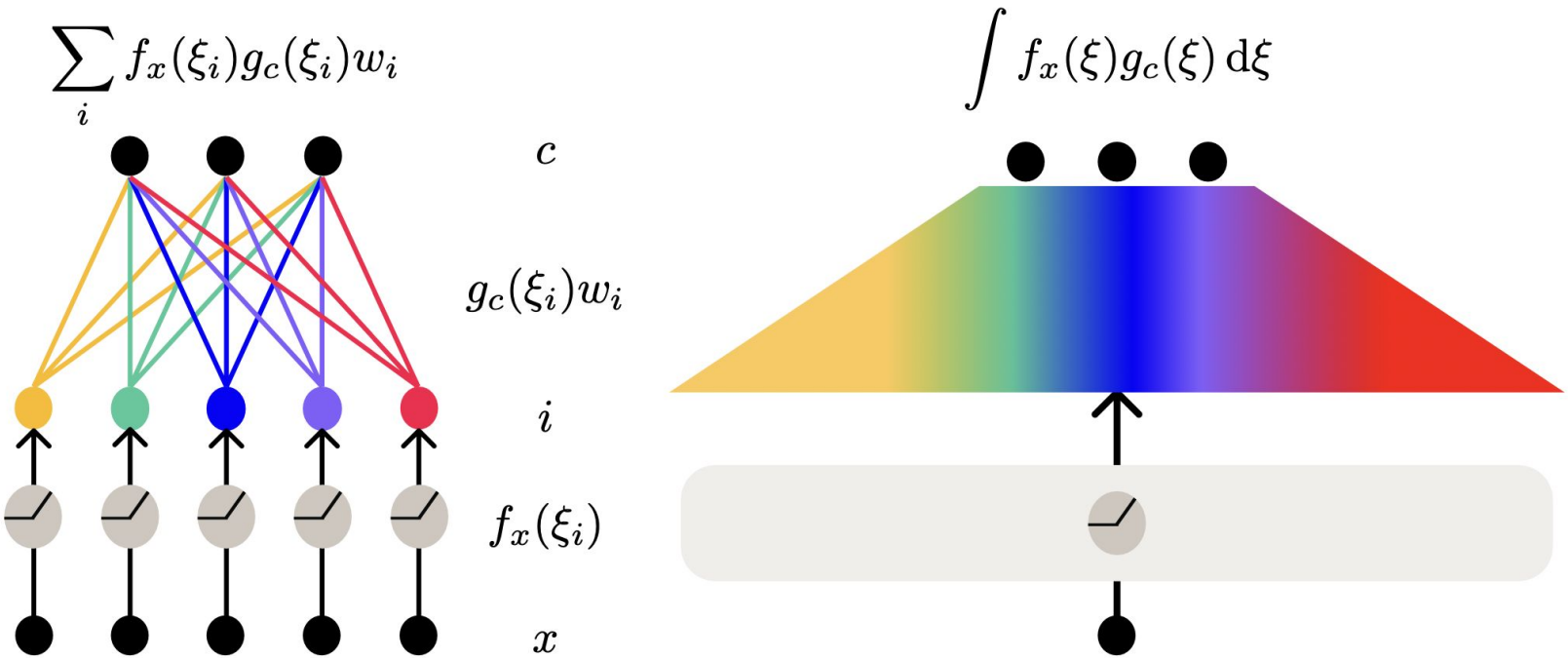
Inability to reduce error ⇒ lacking understanding (how to align & scale intervals)



$$\int_{-\pi}^{\pi} \underbrace{\mathrm{ReLU}\big(\cos(k(a+b)/2+\phi)\big)\cos(2\phi+kc)}_{h(\phi)}\,\mathrm{d}\phi$$

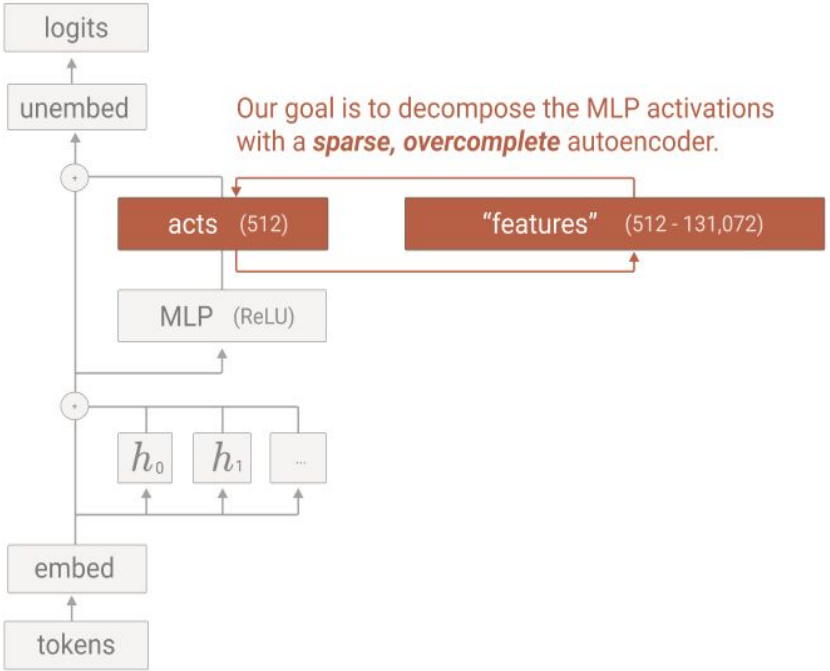
$$= \frac{2}{3}\cos(k(a+b-c))$$

Numerical Integration: Infinite-Width Limit



Current research question: When is this non-vacuous / interesting?

Sparsity Penalty in SAEs



Images from https://transformer-circuits.pub/2023/monosemantic-features

Too much human labor

HOW CAN WE TELL IF THE AUTOENCODER IS WORKING?

Usually in machine learning we can quite easily tell if a method is working by looking at an easily-measured quantity like the test loss. We spent quite some time searching for an equivalent metric to guide our efforts here, and unfortunately have yet to find anything satisfactory.

. . .

Thus we ended up using a combination of several additional metrics to guide our investigations:

1. **Manual inspection:** Do the features seem interpretable?

. . .

We think it would be very helpful if we could identify better metrics for dictionary learning solutions from sparse autoencoders trained on transformers.

https://transformer-circuits.pub/2023/monosemantic-features/index.html#setup-autoencoder

Tanh Penalty in Dictionary Learning

Adam Jermyn, Adly Templeton, Joshua Batson, Trenton Bricken

. . .

We found that autoencoders trained with a $\tanh$ penalty were a Pareto improvement in the space of L0 and loss recovered, often by a wide margin. Unfortunately, we found that the features in these autoencoders were much harder to interpret.

https://transformer-circuits.pub/2024/feb-update/index.html#dict-learning-tanh

Sparsity Penalty in SAEs

Current sparsity penalty: `activations.abs().sum()`

What do SAEs get you?

Case analysis structure in proofs

L1 norm: almost fine if downstream network decomposes additively

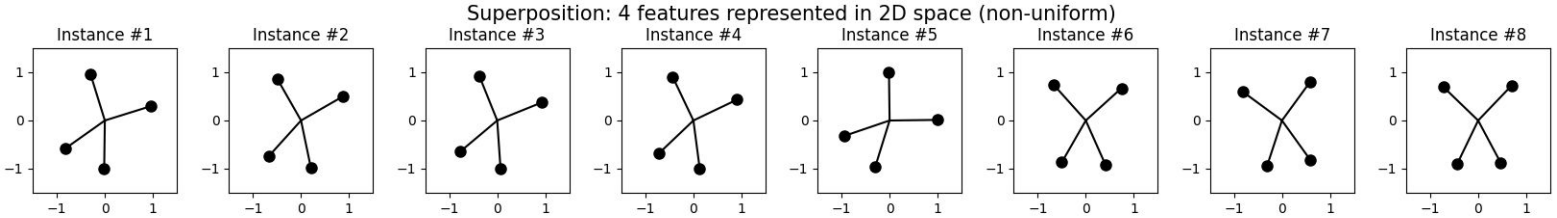
not remotely useful if downstream task is non-linear

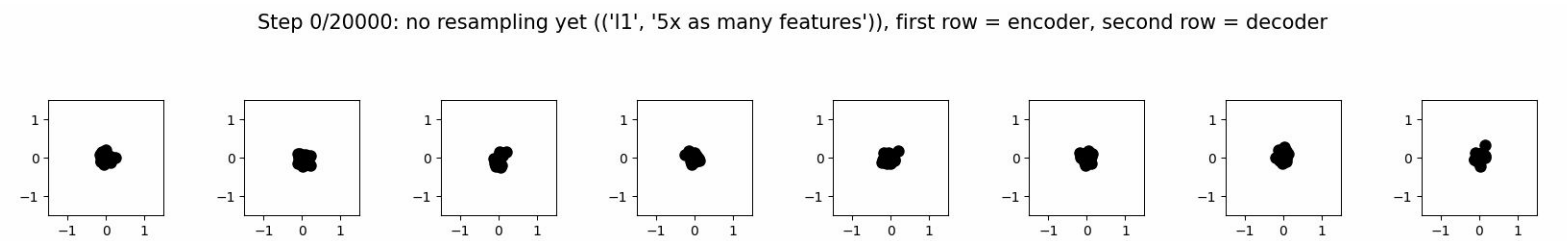
Proofs-frame suggested sparsity penalty:

`(1 + activations.abs()).prod()`

or `activations.abs().log1p().sum()`

Sparsity Penalties in SAEs





https://tinyurl.com/sae-prod1p-colab