# Benefits of Rank in Attention Layers

Noah Amsel, Gilad Yehudai, Joan Bruna


This is joint work with my PhD advisor Joan Bruna, and with Gilad Yehudai, who is a postdoc in the department
I’m going to talk about the rank of an attention layer

# Motivation

- How do we get the most out of transformers?
- How do we set the hyperparameters?

Transformers work amazingly well. Not just in NLP, but standard in vision, audio. used in time series forecasting, tabular data, PDEs / scientific data, biological data… all contexts
The transformer architecture has a bunch of different hyperparameters. I'm interested in how to set these.
If you don't think about this at least a little, there might be some surprises in store for you
There's this great paper with a really clean story about how transformers are
So we run their exact code only we change the number of heads. Weird… why does adding heads make the model worse?

# Motivation

- How do we get the most out of transformers?
- How do we set the hyperparameters?

What Can Transformers Learn In-Context?
A Case Study of Simple Function Classes

Shivam Garg*
Stanford University
shivamg@cs.stanford.edu

Dimitris Tsipras*
Stanford University
tsipras@stanford.edu

Percy Liang
Stanford University
pliang@cs.stanford.edu

Gregory Valiant
Stanford University
valiant@stanford.edu

# Motivation

- How do we get the most out of transformers?
- How do we set the hyperparameters?

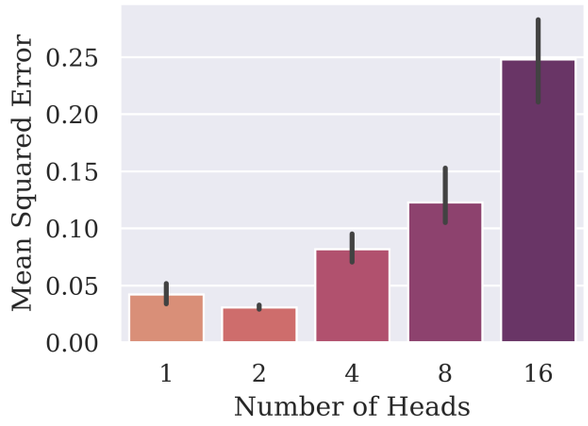
What Can Transformers Learn In-Context?
A Case Study of Simple Function Classes


Shivam Garg*
Stanford University
shivamg@cs.stanford.edu

Dimitris Tsipras*
Stanford University
tsipras@stanford.edu

Percy Liang
Stanford University
pliang@cs.stanford.edu

Gregory Valiant
Stanford University
valiant@stanford.edu

# Ancient History

What is an attention layer?
I know that by now you've seen this a thousand times. I encourage you to not zone out because it's important that we all interpret the formulas the same we and get on the same page with the notation. Give this a high attn score (lol).
To make it simpler let's look at attention with a single query.
These queries, keys and values are living in r dimensions
(actually the values were the same as the keys, but whatever)
[next]
dot product attention
[next]
now we can vectorize all of these expressions (so softmax is applied to this column vector)
[next]
and it all reduces like this

# Ancient History

- “Neural Machine Translation by Jointly Learning to Align and Translate” [Bahdanau, Cho & Bengio, 2016]

# Ancient History

- “Neural Machine Translation by Jointly Learning to Align and Translate” [Bahdanau, Cho & Bengio, 2016]
  - Given query $\mathbf{q}$, keys $\{\mathbf{k}_j\}$, and values $\{\mathbf{v}_j\}$ in $\mathbb{R}^r$

# Ancient History

- “Neural Machine Translation by Jointly Learning to Align and Translate” [Bahdanau, Cho & Bengio, 2016]
  - Given query $\mathbf{q}$, keys $\{\mathbf{k}_j\}$, and values $\{\mathbf{v}_j\}$ in $\mathbb{R}^r$
  - Attention score: $s_j = \mathrm{FNN}(\mathbf{k}_j, \mathbf{q})$

# Ancient History

- “Neural Machine Translation by Jointly Learning to Align and Translate” [Bahdanau, Cho & Bengio, 2016]
  - Given query $\mathbf{q}$, keys $\{\mathbf{k}_j\}$, and values $\{\mathbf{v}_j\}$ in $\mathbb{R}^r$
  - Attention score: $s_j = \mathrm{FNN}(\mathbf{k}_j, \mathbf{q})$
  - Attention coefficients: $\alpha_j = \dfrac{\exp(s_j)}{\sum_k \exp(s_k)}$

# Ancient History

- “Neural Machine Translation by Jointly Learning to Align and Translate” [Bahdanau, Cho & Bengio, 2016]
  - Given query $\mathbf{q}$, keys $\{\mathbf{k}_j\}$, and values $\{\mathbf{v}_j\}$ in $\mathbb{R}^r$
  - Attention score: $s_j = \mathrm{FNN}(\mathbf{k}_j, \mathbf{q})$
  - Attention coefficients: $\alpha_j = \dfrac{\exp(s_j)}{\sum_k \exp(s_k)}$
  - Convex combo of values: $\sum_j \alpha_j \mathbf{v}_j$

# Ancient History

- “Neural Machine Translation by Jointly Learning to Align and Translate” [Bahdanau, Cho & Bengio, 2016]
  - Given query $\mathbf{q}$, keys $\{\mathbf{k}_j\}$, and values $\{\mathbf{v}_j\}$ in $\mathbb{R}^r$
  - Attention score: $s_j = \mathbf{k}_j \cdot \mathbf{q}$
  - Attention coefficients: $\alpha_j = \dfrac{\exp(s_j)}{\sum_k \exp(s_k)}$
  - Convex combo of values: $\sum_j \alpha_j \mathbf{v}_j$

# Ancient History

- "Neural Machine Translation by Jointly Learning to Align and Translate" [Bahdanau, Cho & Bengio, 2016]
  - Given query $\mathbf{q}$, keys $\{\mathbf{k}_j\}$, and values $\{\mathbf{v}_j\}$ in $\mathbb{R}^r$
  - Attention score: $s_j = \mathbf{k}_j \cdot \mathbf{q}$ $\quad\quad$ $\mathbf{s} = \mathbf{K}\mathbf{q}$
  - Attention coefficients: $\alpha_j = \dfrac{\exp(s_j)}{\sum_k \exp(s_k)}$ $\quad\quad$ $\alpha = \mathrm{sm}(\mathbf{s})$
  - Convex combo of values: $\sum_j \alpha_j \mathbf{v}_j$ $\quad\quad$ $\mathbf{V}\alpha$

# Ancient History

- "Neural Machine Translation by Jointly Learning to Align and Translate" [Bahdanau, Cho & Bengio, 2016]
  - Given query $\mathbf{q}$, keys $\{\mathbf{k}_j\}$, and values $\{\mathbf{v}_j\}$ in $\mathbb{R}^r$
  - Attention score: $s_j = \mathbf{k}_j \cdot \mathbf{q}$ $\quad\quad$ $\mathbf{s} = \mathbf{K}\mathbf{q}$
  - Attention coefficients: $\alpha_j = \dfrac{\exp(s_j)}{\sum_k \exp(s_k)}$ $\quad\quad$ $\alpha = \mathrm{sm}(\mathbf{s})$
  - Convex combo of values: $\sum_j \alpha_j \mathbf{v}_j$ $\quad\quad$ $\mathbf{V}\alpha$ $\quad\quad$ $= \mathbf{V}\ \mathrm{sm}(\mathbf{K}^\top \mathbf{q})$

# Singe-head Attention

Given $\mathbf{y}$ and $\mathbf{x}_1, \ldots \mathbf{x}_N \in \mathbb{R}^d$, let

$$\mathbf{q} = \mathbf{W}^Q \mathbf{y} \qquad \mathbf{K} = \mathbf{W}^K \mathbf{X} \qquad \mathbf{V} = \mathbf{W}^V \mathbf{X}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{r \times d}$.

$$\mathbf{V}\ \mathrm{sm}(\mathbf{K}^\top \mathbf{q})$$

The next step is to think of the queries keys and values as coming from some other vectors via a linear layer.
So the query comes from y; keys/values from the x's. the x's and y come from a higher dimensional space R^d
If we substitute this into the formula from the last slide, we get this
[next]
and if we map this back to the higher dimensional space with another linear layer, we get
[next]

# Singe-head Attention

Given $\mathbf{y}$ and $\mathbf{x}_1, \ldots \mathbf{x}_N \in \mathbb{R}^d$, let

$$\mathbf{q} = \mathbf{W}^Q \mathbf{y} \qquad \mathbf{K} = \mathbf{W}^K \mathbf{X} \qquad \mathbf{V} = \mathbf{W}^V \mathbf{X}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{r \times d}$.

$$\mathbf{W}^V \mathbf{X} \ \mathrm{sm}\left( \left( \mathbf{W}^K \mathbf{X} \right)^\top \mathbf{W}^Q \mathbf{y} \right)$$

# Singe-head Attention

Given $\mathbf{y}$ and $\mathbf{x}_1, \ldots \mathbf{x}_N \in \mathbb{R}^d$, let

$$\mathbf{q} = \mathbf{W}^Q \mathbf{y} \qquad \mathbf{K} = \mathbf{W}^K \mathbf{X} \qquad \mathbf{V} = \mathbf{W}^V \mathbf{X}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V, \mathbf{W}^O \in \mathbb{R}^{r \times d}$.

$$\left(\mathbf{W}^O\right)^\top \mathbf{W}^V \mathbf{X} \ \mathrm{sm}\left(\left(\mathbf{W}^K \mathbf{X}\right)^\top \mathbf{W}^Q \mathbf{y}\right)$$

# Multi-head Attention

Given $\mathbf{y}$ and $\mathbf{x}_1, \ldots \mathbf{x}_N \in \mathbb{R}^d$

$$\sum_{h=1}^{H} \left(\mathbf{W}_h^O\right)^\top \mathbf{W}_h^V \, \mathbf{X} \, \mathrm{sm}\left(\left(\mathbf{W}_h^K \, \mathbf{X}\right)^\top \mathbf{W}_h^Q \, \mathbf{y}\right)$$

where $\mathbf{W}_h^Q, \mathbf{W}_h^K, \mathbf{W}_h^V, \mathbf{W}_h^O \in \mathbb{R}^{r \times d}$.

- Num parameters: $4rHd$

That was one attention head.
Now add up H copies with different weight matrices
The number of parameters is…

# Multi-head Attention

Given $\mathbf{y}$ and $\mathbf{x}_1, \ldots \mathbf{x}_N \in \mathbb{R}^d$

$$\sum_{h=1}^{H} \mathbf{W}_h \, \mathbf{X} \, \mathrm{sm}\left(\mathbf{X}^\top \mathbf{M}_h \mathbf{y}\right)$$

where $\mathbf{M}_h, \mathbf{W}_h \in \mathbb{R}^{d \times d}$ are rank-$r$.

- Num parameters: $4rHd$

We can write this a little differently by merging the matrices.
Now we're rewriting this without the keys queries and values. The only remnant of them is rank-r requirement. And that's why we should think of "r" as the rank.
By the way we could actually make M_h and W_h have different ranks, like r and r2, but no one does this
So given the input dimension d, a multi-head attention layer has these two hyperparameters: H and r, and it has 4rHd parameters.

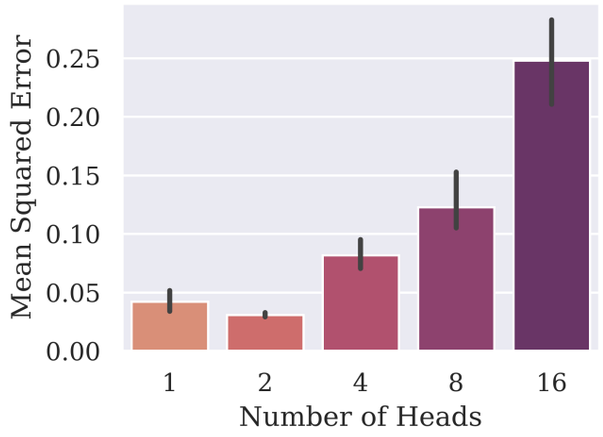
| | | | num layers | MLP depth | | value rank | |
|---|---|---|---|---|---|---|---|
| | | hidden dim | | MLP width | attn rank | | num heads |
| Year | Model | $d$ | $L$ | $w$ | $D$ | $r$ | $r_2$ | $H$ |
| 2017 | Attention is all you need [59] | 512 | 6 | $4d$ | 2 | 64 | $r$ | $d/r$ |

Num params $= 4rHd = 4d^2$
Same as $H = 1,\ r = d$

1. Let's take a look at how the original transformers paper set the hyperparameters
(go thru each). this choice of scaling
2. Subsequent work? d and L are growing like crazy. lots of work on scaling laws for these parameters
The other parameters are barely changing
rank is always on the order of 100, and everyone is adopting this scaling of d/r…
(of course, H is growing too, but always according to this same scaling)
Surprisingly this scaling is actually hard coded into PyTorch and xFormers libraries. You'd have to start editing the C++… no one questions it
how are we supposed to set r? How are we supposed to set H?
"This paper raises doubts about the universal practice of setting r << d and H = d/r

| | | | num layers | MLP depth | value rank | |
|---|---|---|---|---|---|---|
| | | hidden dim | MLP width | attn rank | | num heads |
| Year | Model | $d$ | $L$ | $w$ | $D$ | $r$ | $r_2$ | $H$ |
| 2017 | Attention is all you need [59] | 512 | 6 | $4d$ | 2 | 64 | $r$ | $d/r$ |
| 2018 | GPT, GPT-2 [44, 45] | 768 | 12 | $4d$ | 2 | 64 | $r$ | $d/r$ |
| 2019 | Bert-Large [16] | 1,024 | 24 | $4d$ | 2 | 64 | $r$ | $d/r$ |
| 2021 | ViT-Huge [17] | 1280 | 32 | $4d$ | 2 | 80 | $r$ | $d/r$ |
| | CLIP (text encoder) [43] | 1,024 | 12 | $4d$ | 2 | 64 | $r$ | $d/r$ |
| | Jurassic-1 | 13,824 | 76 | $4d$ | 2 | 144 | $r$ | $d/r$ |
| | Gopher 280B [46] | 16,384 | 80 | $4d$ | 2 | 128 | $r$ | $d/r$ |
| | LaMDA [55] | 8192 | 64 | $8d$ | 2 | 128 | $r$ | $2d/r$ |
| 2022 | Chinchilla 70B [27] | 8,192 | 80 | $4d$ | 2 | 128 | $r$ | $d/r$ |
| | GPT-3 [8] | 12,288 | 96 | $4d$ | 2 | 128 | $r$ | $d/r$ |
| 2023 | PaLM [12] | 18,432 | 118 | $4d$ | 2 | 256 | $r$ | $2d/3r$ |
| | LLaMA, Llama-2 [57, 58] | 8,192 | 80 | $8d/3$ | 2 | 128 | $r$ | $d/r$ |
| 2024 | OLMo [23] | 8,192 | 80 | $8d/3$ | 2 | 128 | $r$ | $d/r$ |

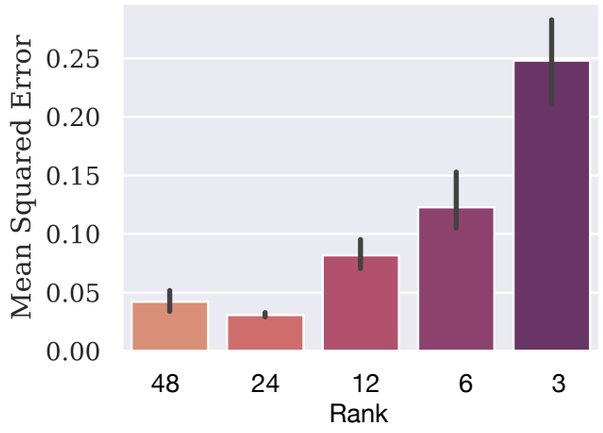
# Back to Garg et al.




**r = d / H**

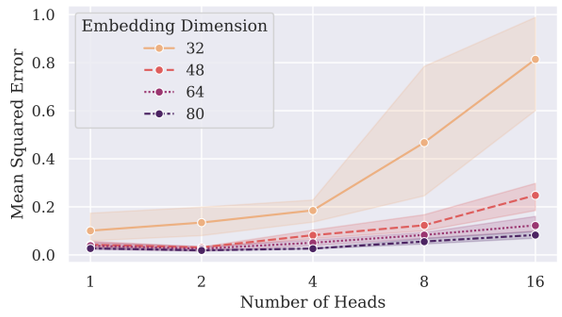
The code using PyTorch, PyTorch always uses the standard scaling
So adding more heads doesn’t actually add more parameters
All these models have the same number of parameters
Ok so probably people know that, but because the number of parameters is the same they might not think that this hyperparameter is so important. which is why they don’t mess with it much
For this task, high rank is important. NOT number of parameters

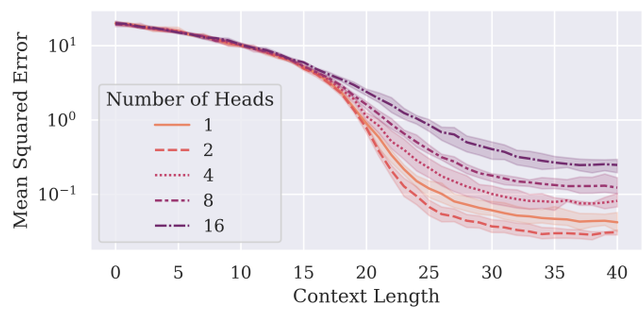
# Back to Garg et al.




**r = d / H**

# More Garg et al.




We do a few other varieties on this experiment. High rank (low #heads) is better

# Theory papers assume full-rank

- Practitioners use low $r$ and $H = d/r$
- Theoreticians assume $r = d$ and $H \gg 1$
  - Is this a good proxy for real-world transformers?

Now I’m not going to prove that low rank is a bad idea in practice for most problems. But even if you think that this actually is the best way to set the parameters, it’s still important to understand for it theory
This is the transformer theory reading group after all

# Related Work

# Depth Separation for MLPs

So we’re interested in the importance of rank and in the tradeoff between rank and number of heads.
I want you to think of this as analogous to an older question about feed forward networks

Matus extends this separation to deeper networks

# Depth Separation for MLPs

- **Q:** 2-layer NNs are universal approximators. So why are deep NNs better?

# Depth Separation for MLPs

- **Q:** 2-layer NNs are universal approximators. So why are deep NNs better?
- **A:** Deep NNs are more parameter-efficient
  - Pay for some depth, save a *lot* of width

# Depth Separation for MLPs

- **Q:** 2-layer NNs are universal approximators. So why are deep NNs better?
- **A:** Deep NNs are more parameter-efficient
  - Pay for some depth, save a *lot* of width
- **Thm:** There exists a function which
  - depth 3 MLP can represent with width $d^5$
  - depth 2 MLP needs width $e^{cd}$ to approximate

# Depth Separation for MLPs

- **Q:** 2-layer NNs are universal approximators. So why are deep NNs better?
- **A:** Deep NNs are more parameter-efficient
  - Pay for some depth, save a *lot* of width
- **Thm:** There exists a function which
  - depth 3 MLP can represent with width $d^5$
  - depth 2 MLP needs width $e^{cd}$ to approximate
- [Eldan & Shamir ‘16] [Telgarsky ‘16] [Daniely ‘17] [Chatziafratis+ ’19] …

# Separations in Expressive Power

Construct a function showing that you can…

**MLP:**

…pay a bit for depth, save a *lot* of width

**Multi-head attention layer:**

…pay a bit for rank, save a *lot* of heads

to reiterate, here is the goal of these separation results in the theory of representational capacity
make the rank bigger, save parameters by using way fewer heads

# Another rank separation result

Our main question is whether using low-rank heads limits the representational power of attention
There's another paper that addresses this question, among others.
And we actually heard a talk on it in this very reading group from Clayton
THM:
Their paper also has a bunch of other results, including comparing attention to RNNs, to 3-way tensor attention. And their target function is closely akin to the kinds of structured reasoning tasks that people want to apply transformers to
Great! so we have a hardness result relating to the rank of the transformer. The rank has to be big enough…right?
but really this is only a barrier if rH is small. So even if the rank is really small, it's no problem. Just set H inversely proportionally large, like in the standard scaling.

But we want to know, is low rank attention still weak even if H is huge?
prohibitively large… large enough that you basically need big rank.

# Another rank separation result

- “Representational Strengths and Limitations of Transformers” [Sanford, Hsu, Telgarsky, 2023]

# Another rank separation result

- “Representational Strengths and Limitations of Transformers” [Sanford, Hsu, Telgarsky, 2023]
- **Thm:** A single layer of multi-head self-attention cannot compute Match3 unless $rHp > \tilde{\Omega}(N)$
  - Proved using communication complexity ($p$ is precision)

# Another rank separation result

- "Representational Strengths and Limitations of Transformers" [Sanford, Hsu, Telgarsky, 2023]
- **Thm:** A single layer of multi-head self-attention cannot compute Match3 unless $rHp > \tilde{\Omega}(N)$
  - Proved using communication complexity ($p$ is precision)
- **Q:** Is low $r$ a limitation, or just low $rH$?

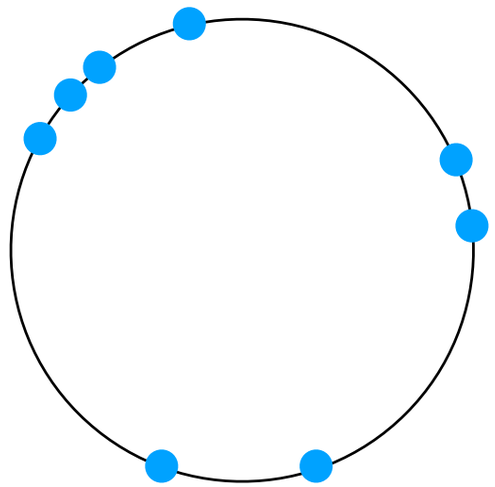
# Another rank separation result

- "Representational Strengths and Limitations of Transformers" [Sanford, Hsu, Telgarsky, 2023]
- **Thm:** A single layer of multi-head self-attention cannot compute Match3 unless $rHp > \tilde{\Omega}(N)$
  - Proved using communication complexity ($p$ is precision)
- **Q:** Is low $r$ a limitation, or just low $rH$?
- We extend hardness guarantee to…
  - Prohibitively large $H$
  - $p = \infty$
  - $\epsilon$-approximation

# Our Rank Separation

# Nearest Neighbor Function

target points $\mathbf{x}_i$

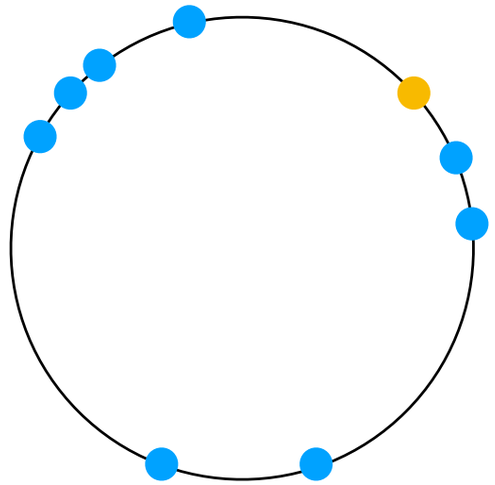


I told you we were looking for a function that's easy for full rank attention and hard for low-rank attention. and here it is…
the input is target points x_1 thru x_N on the sphere
and the other input is a source point y on the sphere
[next][next]
except we're doing this on a high dimensional sphere
think of this like semantic search. we have some database or context window, and we want to find the entry or the word that most closely matches the query
Highly symmetric: rotationally invariant, invariant to permutations of these points

# Nearest Neighbor Function

target points $\mathbf{x}_i$

source point $\mathbf{y}$

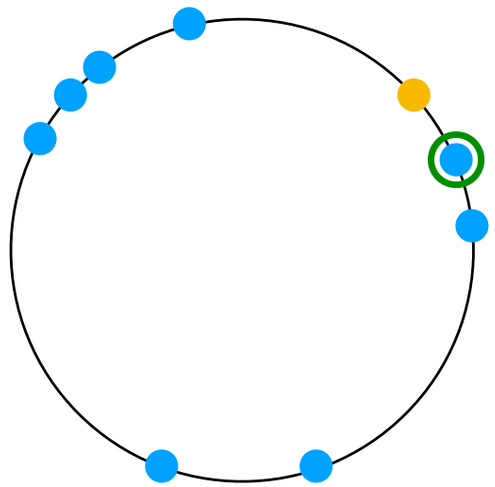


# Nearest Neighbor Function

target points $\mathbf{x}_i$

source point $\mathbf{y}$

output $f(\mathbf{x}_1, \ldots, \mathbf{x}_N; \mathbf{y})$



# Target Function: Nearest Neighbor

For $\mathbf{y}, \mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{S}^{d-1}$

$$
\begin{aligned}
f(\mathbf{x}_1, \ldots, \mathbf{x}_N; \mathbf{y}) &:= \operatorname*{argmin}_{\mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}} \|\mathbf{x} - \mathbf{y}\|_2 \\
&= \operatorname*{argmax}_{\mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}} \mathbf{x}^\top \mathbf{y} \\
&= \mathbf{X}\ \mathrm{hm}(\mathbf{X}^\top \mathbf{y}) \\
&= \mathbf{I}_d\ \mathbf{X}\ \mathrm{sm}\left(\mathbf{X}^\top \left(10^{10} \cdot \mathbf{I}_d\right) \mathbf{y}\right)
\end{aligned}
$$

full rank

we are doing this minimization
since these are unit vectors, it's the same as this maximization problem
and we can rewrite this argmax as follows.
and here you can see that this has the same form as a full-rank attention head
we could use hardmax heads, or we could just scale up the weights to make the softmax act like a hardmax

# Upper Bound

For $\mathbf{y}, \mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{S}^{d-1}$, the nearest neighbor function can be exactly represented by a single full-rank hardmax attention head.

"$r = d$, $H = 1$ suffices"

And this is for any inputs, the distribution doesn't matter
Our main theorem is a lower bound for this approximating this function for low rank r

# Lower Bound (Informal)

If $H \lesssim (d/r)^{1/\epsilon}$, then

$$\mathop{\mathbb{E}}_{\mathbf{x}_1 \perp \mathbf{x}_2,\ \mathbf{y} \sim \mathcal{U}(\mathbb{S}^{d-1})} \left\| f(\mathbf{x}_1, \mathbf{x}_2; \mathbf{y}) - \sum_{h=1}^{H} \mathrm{attn}_h(\mathbf{x}_1, \mathbf{x}_2; \mathbf{y}) \right\|_2^2 \geq \epsilon$$

Unless the number of heads is polynomially large in d / rank, you can’t approximate the target accurately.
This is a multi-head attention layer with H heads and rank-r attention.
The degree of the polynomial dependence on d/r is related to the error of approximation. As epsilon gets smaller, the necessary number of heads grows exponentially
This is informal, I’m simplifying this RHS a lot
For the upper bound, any number of target points
Here we stick to two target points for simplicity, but in high dimensions, more target points makes the problem harder
The upper bound holds for any reasonable distribution on the sphere
For the lower bound, we’re drawing uniformly at random from the sphere, conditional on x1 and x2 being orthogonal (which happens anyway in high dimension)

# Generalizing Attention

Standard: $$\sum_{h=1}^{H} \left(\mathbf{W}_h^O\right)^\top \mathbf{W}_h^V \, \mathbf{X} \, \mathrm{sm}\left(\left(\mathbf{W}_h^K \, \mathbf{X}\right)^\top \mathbf{W}_h^Q \, \mathbf{y}\right)$$

where $\mathbf{W}_h^Q, \mathbf{W}_h^K, \mathbf{W}_h^V, \mathbf{W}_h^O \in \mathbb{R}^{r \times d}$.

Generalized: $$\sum_{h=1}^{H} \mathbf{V}_h \, \mathbf{X} \, \phi_h\left(\mathbf{K}_h^\top \mathbf{X}, \, \mathbf{y}\right)$$

where $\mathbf{V}_h \in \mathbb{R}^{d \times d}, \quad \mathbf{K}_h \in \mathbb{R}^{d \times r}, \quad \phi_h : \mathbb{R}^{r \times N} \times \mathbb{R}^d \to \Delta^N$.

Our lower bound holds for a generalization of attention
Instead of creating keys and queries in r dimensions and than taking their dot products, we can use any function we want to compute the attention scores them. For example, we could add positional encodings, or RoPE! can be a different function for each head.
We're still projecting the x's down to dimension r (I'm renaming the weight matrix), but the y doesn't have to get projected.
So this is a throwback to generalized attention, but still with this rank-r bottleneck.
And V can be full rank.
Just to be clear, we're overloading notation. So V and K are parameters, they aren't the values and keys. VX is like the values and K^T X is the keys.

# Proof Sketch

I’m going to try to keep this a bit high level but if I’m going too quickly tell me

# Reduction to Scalar Function on $\mathbb{S}^{d-1} \times \mathbb{S}^{d-1}$

$$\mathop{\mathbb{E}}_{\mathbf{x}_1 \perp \mathbf{x}_2,\ \mathbf{y} \sim \mathcal{U}(\mathbb{S}^{d-1})} \left\| f(\mathbf{x}_1, \mathbf{x}_2; \mathbf{y}) - \sum_{h=1}^{H} \mathrm{attn}_h(\mathbf{x}_1, \mathbf{x}_2; \mathbf{y}) \right\|_2^2$$

$$\geq \frac{1}{2} \mathop{\mathbb{E}}_{\mathbf{x},\ \mathbf{y} \sim \mathcal{U}(\mathbb{S}^{d-1})} \left( \mathrm{sgn}\left(\mathbf{x}^\top \mathbf{y}\right) - \sum_{h=1}^{H} g_h(\mathbf{x}, \mathbf{y}) \right)^2$$

**(follows by projecting onto $\mathbf{x}_1 - \mathbf{x}_2$)**

where $g_h(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{V}_h \mathbf{x} \cdot \tilde{\phi}_h\left(\mathbf{K}_h^\top \mathbf{x}, \mathbf{y}\right)$

I'm not going to this carefully, but it's just a change of basis.
So now we're drawing two vectors iid, and we have this very simple scalar target function that's basically like a single random neuron: dot product, then threshold
It's basically the same as our nearest neighbor target function, which outputs either x_1 or x_2 depending on the inner products with y. This outputs +1 or -1 depending on the inner product with y
Another simplification for ease of presentation, let's assume V_h is the identity
[next]
Think of it like saying, "there's no privileged direction"
Since x is a unit vector, these part goes away.
The main idea of the proof is to find a suitable orthogonal basis in which to expand the these functions and show that a bunch of basis elements that appear in the expansion of the target function are missing from the expansion of the attn layer.
So all leftover energy from the target function is error

# Reduction to Scalar Function on $\mathbb{S}^{d-1} \times \mathbb{S}^{d-1}$

$$\mathop{\mathbb{E}}_{\mathbf{x}_1 \perp \mathbf{x}_2,\ \mathbf{y} \sim \mathcal{U}(\mathbb{S}^{d-1})} \left\| f(\mathbf{x}_1, \mathbf{x}_2; \mathbf{y}) - \sum_{h=1}^{H} \mathrm{attn}_h(\mathbf{x}_1, \mathbf{x}_2; \mathbf{y}) \right\|_2^2$$

$$\geq \frac{1}{2} \mathop{\mathbb{E}}_{\mathbf{x},\ \mathbf{y} \sim \mathcal{U}(\mathbb{S}^{d-1})} \left( \mathrm{sgn}\left(\mathbf{x}^\top \mathbf{y}\right) - \sum_{h=1}^{H} g_h(\mathbf{x}, \mathbf{y}) \right)^2$$

(follows by projecting onto $\mathbf{x}_1 - \mathbf{x}_2$)

where $g_h(\mathbf{x}, \mathbf{y}) = \tilde{\phi}_h\left(\mathbf{K}_h^\top \mathbf{x}, \mathbf{y}\right)$

## Representing $\operatorname{sgn}\left(\mathbf{x}^\top \mathbf{y}\right)$ using rank-1 features

But first I'm going to show you a funny way to represent this target function…
[next]
let's define…
[next] we can take a linear combination
u_i here is a weight that you assign to the head k_i, q_i.
[next] instead of a finite linear combination
this is a letter T by the way, I'm not sure why it looks this way
[next]
in fact our target function can be represented this way.
what is u?

# Representing $\operatorname{sgn}\left(\mathbf{x}^\top \mathbf{y}\right)$ using rank-1 features

- A rank-1 “feature function”: $(\mathbf{x}, \mathbf{y}) \mapsto \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}\ \mathbf{q}^\top \mathbf{y}\right)$
  - Essentially a rank-1 hardmax attention head parameterized by $\mathbf{q}, \mathbf{k} \in \mathbb{R}^d$

# Representing $\operatorname{sgn}\left(\mathbf{x}^\top \mathbf{y}\right)$ using rank-1 features

- A rank-1 “feature function”: $(\mathbf{x}, \mathbf{y}) \mapsto \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}\ \mathbf{q}^\top \mathbf{y}\right)$
  - Essentially a rank-1 hardmax attention head parameterized by $\mathbf{q}, \mathbf{k} \in \mathbb{R}^d$
- Linear combo of rank-1 features: $(\mathbf{x}, \mathbf{y}) \mapsto \sum_i u_i \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}_i\ \mathbf{q}_i^\top \mathbf{y}\right)$

# Representing $\operatorname{sgn}\left(\mathbf{x}^\top \mathbf{y}\right)$ using rank-1 features

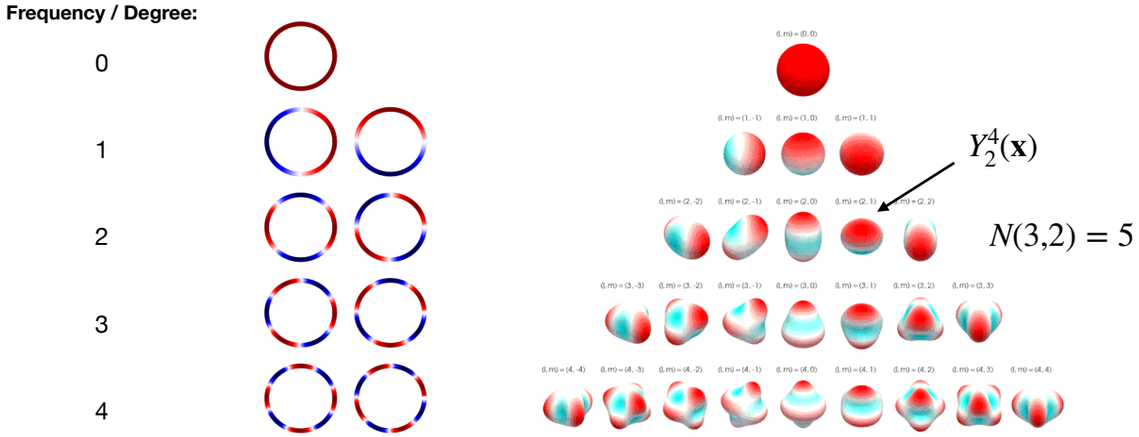
- A rank-1 "feature function": $(\mathbf{x}, \mathbf{y}) \mapsto \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}\ \mathbf{q}^\top \mathbf{y}\right)$
  - Essentially a rank-1 hardmax attention head parameterized by $\mathbf{q}, \mathbf{k} \in \mathbb{R}^d$
- Linear combo of rank-1 features: $(\mathbf{x}, \mathbf{y}) \mapsto \sum_i u_i \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}_i\ \mathbf{q}_i^\top \mathbf{y}\right)$
- $[\mathscr{T} u](\mathbf{x}, \mathbf{y}) := \int_{\mathbb{S}^{d-1} \times \mathbb{S}^{d-1}} u(\mathbf{q}, \mathbf{k}) \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}\ \mathbf{q}^\top \mathbf{y}\right)$ for any $u : \mathbb{S}^{d-1} \times \mathbb{S}^{d-1} \to \mathbb{R}$

# Representing $\operatorname{sgn}\left(\mathbf{x}^\top \mathbf{y}\right)$ using rank-1 features

- A rank-1 "feature function": $(\mathbf{x}, \mathbf{y}) \mapsto \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}\ \mathbf{q}^\top \mathbf{y}\right)$
  - Essentially a rank-1 hardmax attention head parameterized by $\mathbf{q}, \mathbf{k} \in \mathbb{R}^d$
- Linear combo of rank-1 features: $(\mathbf{x}, \mathbf{y}) \mapsto \sum_i u_i \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}_i\ \mathbf{q}_i^\top \mathbf{y}\right)$
- $[\mathcal{T} u](\mathbf{x}, \mathbf{y}) := \int_{\mathbb{S}^{d-1} \times \mathbb{S}^{d-1}} u(\mathbf{q}, \mathbf{k}) \operatorname{sgn}\left(\mathbf{x}^\top \mathbf{k}\ \mathbf{q}^\top \mathbf{y}\right)$ for any $u : \mathbb{S}^{d-1} \times \mathbb{S}^{d-1} \to \mathbb{R}$
- **Lemma:** $\exists u$ s.t. $\operatorname{sgn}\left(\mathbf{x}^\top \mathbf{y}\right) = \left[\mathcal{T} u\right](\mathbf{x}, \mathbf{y})$
  - What's $u$?

# Spherical Harmonics

An orthonormal basis for $L^2(\mathbb{S}^{d-1} \to \mathbb{R})$




# basis elements of degree $\ell$ in on $\mathbb{S}^{d-1}$ is $N(d, \ell)$

Spherical harmonics are just an orthonormal basis for functions that take a point on the sphere and output a real number
Think of these as a generalization of the Fourier basis.
Fourier analysis shows that sins and cosine are an orthonormal basis for periodic functions, that is functions that are defined on a circle.
Spherical harmonics are an orthonormal basis for functions on the sphere.
The Fourier basis is organized by frequency. 0, 1, 2, … infinity
Spherical harmonics are the same.
The Fourier basis has two elements of each frequency. sine and cosine.
The spherical harmonics, since we're in higher dimensions, have more than two. We have a notation for this N(d, ell). So for example, in ambient dimension 3, and degree 2, there are 5 basis elements.
If we consider the span of harmonics of one degree, that's called the ell'th harmonic
Any function can be decomposed into its degree 0 part (coming from a 1 dimensional subspace), its degree 1 part (an element in a 3 dimensional subspace), degree 2 part (an element in a 5 dimensional subspace)

# Expansion of $\operatorname{sgn}\left(\mathbf{x}^\top \mathbf{y}\right)$

- Recall: $\operatorname{sgn}\left(\mathbf{x}^\top \mathbf{y}\right) = \left[\mathcal{T} u\right](\mathbf{x}, \mathbf{y})$
- **Lemma:** $\left\{\mathcal{T}\left(Y_\ell^i \otimes Y_\ell^i\right)\right\}_{\ell,i}$ is orthogonal
- $\mathcal{T} u = \sum_{\ell=0}^{\infty} \sum_{i=1}^{N(d,\ell)} \beta_\ell \cdot \mathcal{T}\left(Y_\ell^i \otimes Y_\ell^i\right)$
  - $\beta_\ell$ is known, independent of $i$, decays slowly: $N(d,\ell) \cdot \beta_\ell \sim \frac{d}{\ell^2}$

We're going to represent our function with this integral operator. And now u is a function that takes two unit vectors. Represent it by products of two spherical harmonics
It turns out that if we represent functions this way, we get an orthonormal basis. So our basis elements look like this, and anything in the range of T can be represented using this basis, including sign.
Target function is isotropic, symmetric. all basis elements from the same harmonic get the same weight.
In other words, the "energy" of this function is spread evenly within each harmonic
and the total energy of the harmonics decays slowly

# Expansion of the head functions

- $\phi_h\left(\mathbf{K}_h^\top \mathbf{x}, \mathbf{y}\right)$ only cares about an $r$-dimensional projection of $\mathbf{x}$
- ⇒ ortho to many spherical harmonics, e.g. $\phi_h\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \mathbf{y}\right) \perp x_3^5$
- **Lemma:** $\phi_h\left(\mathbf{K}_h^\top \mathbf{x}, \mathbf{y}\right)$ is orthogonal to $\mathcal{T}(Y_\ell^i \otimes Y_\ell^i)$ for all $Y_\ell^i$ in $\ell$th harmonic except for a subspace of dimension $M(d, \ell) \leq \binom{r+\ell}{\ell}$
  - out of dimension $N(d, \ell)$

Remember, the head functions are low rank
They cannot be isotropic / symmetric like the target function. they care about r directions, and not the other d - r.

# Combining

$$\mathop{\mathbb{E}}_{\mathbf{x},\,\mathbf{y}\sim\mathcal{U}(\mathbb{S}^{d-1})}\left(\operatorname{sgn}\left(\mathbf{x}^{\top}\mathbf{y}\right)-\sum_{h=1}^{H}g_h(\mathbf{x},\mathbf{y})\right)^2$$

$$=\left\|\sum_{\ell=0}^{\infty}\left[\sum_{i=1}^{N(d,\ell)}\beta_\ell\mathcal{T}\left(Y_\ell^i\otimes Y_\ell^i\right)-\sum_{h=1}^{H}\sum_{i=1}^{M(d,\ell)}?_{h,\ell,i}\mathcal{T}\left(Y_\ell^i\otimes Y_\ell^i\right)\right]\right\|^2$$

$$\geq\left\|\sum_{\ell=0}^{\infty}\beta_\ell\left(N(d,\ell)-HM(d,\ell)\cdot T\left(Y_\ell^i\otimes Y_\ell^i\right)\right\|^2=\sum_{\ell=0}^{\infty}\beta_\ell^2\left(N(d,\ell)-HM(d,\ell)\right)\right.$$

If you do the asymptotic, N(d, ell) grows very quickly with both d and l, much faster than M.
Beta decays with ell, but not very fast. So unless H is quite large, this final expression is a serious lower bound on the error of approximation

# [End Proof Sketch]

For this part. you have my permission to zone out if you wish

# Lower Bound

**Theorem 2** (Low-Rank Approximation Lower Bounds, Equivariant Case)**.** *There exist universal constants $c, c', C$ and $C'$ such that if either of the following sets of assumptions hold:*

1. High-accuracy regime*:* $r \le d-3$, $\epsilon \le \frac{c}{d+1}$, *and*
$$H \le C \cdot 2^{d-(r+1)\log_2(2d/r)} .$$
2. High-dimensional regime: $d \ge 5$, $\epsilon \ge \frac{c'}{d-2e^2 \cdot r}$ *and*
$$H \le \frac{1}{2}\left(\frac{1}{2e} \cdot \frac{d}{r + C'/\epsilon}\right)^{C'/\epsilon} .$$

think of $H \lesssim (d/r)^{1/\epsilon}$

*Then, for any choice of $H$ rank-$r$ generalized attention heads $\phi_h : \mathbb{R}^{r\times 2} \to \Delta^1$, $\boldsymbol{V}_h \in \mathbb{R}^{d\times d}$, $\boldsymbol{K}_h \in \mathbb{R}^{d\times r}$ the error of approximating the nearest neighbor function is bounded as follows*

$$\mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_1,\boldsymbol{x}_2 \sim \mathcal{D}_2(\mathbb{S}^{d-1}) \\ \boldsymbol{y} \sim \mathrm{Unif}(\mathbb{S}^{d-1})}} \left\| f(\boldsymbol{X};\boldsymbol{y}) - \sum_{h=1}^{H} \boldsymbol{V}_h \boldsymbol{X} \phi_h\left(\boldsymbol{K}_h^\top \boldsymbol{X}, \boldsymbol{y}\right) \right\|_2^2 \ge \epsilon ,$$

Intuitively, the problem gets harder as eps shrinks to zero and as d grows to infinity. If you fix d and take epsilon to small, like smaller than smaller than O(1/d), you get this first regime, (where the number of heads has to grow exponentially in d - r.)
If you fix epsilon and take d to be big, like bigger than 1/eps, then you're in this regime.
but they're both basically (d/r)^1/eps.
So this means that if you use r < d

# Alternative Lower Bound

There exists a target function $f^*$ such that unless $H \cdot \max_h \|\mathbf{V}_h\| \lesssim c^{d-r}$, the error of approximation is bounded as follows:

$$\mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_1,\boldsymbol{x}_2\sim\mathcal{D}_2\left(\mathbb{S}^{d-1}\right)\\ \boldsymbol{y}\sim\mathcal{N}\left(0,\frac{1}{\sqrt{d}}I\right)}}\left[\left\|T(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{y})-f^*(\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{y})\right\|^2\right] > \frac{1}{40}$$

- Differences: bias in upper bound, dependence on $\|\mathbf{V}_h\|$, proof techniques

We also prove a different lower bound using a different target function.
Previous bound: (d/r)^{1/eps} is only polynomial in d. So for a constant epsilon, like .05, it's d^20.
Now we get exp in d, even for constant eps.
The target function is a combination of biased nearest neighbor functions with different biases
Dependence on V
notice y is from a gaussian here, but basically the same

# What about more layers?

- Low-rank attention layers are weaker, even for large $H$
- **Q:** Is a low-rank *transformer* weaker? … idk
- Construction: *modified* rank-1 transformer works for $N = 2$
- Conjecture: low-rank transformer fails for large $N$, unlike full-rank one

works means “approximates this target function in a parameter efficient way”
recall N is the number of target points
But this depends on knowing that there are only two targets.
If you add more targets, you have to use a bigger transformer. And this is too bad, because the whole point of using a transformer is that you want it to work seamlessly for inputs with many many tokens
And remember our full rank construction from the beginning works on any N

## Modified Attention Construction: $L = 2,\ N = 2$

Let’s say we are using hardmax heads

Can consider this modifying the transformer, or tagging the input and using a higher dimensional transformer (dim = d + 2)

No MLP needed

# Modified Attention Construction: $L = 2,\ N = 2$

- Random rank-1 head $\mathrm{hm}\left(\mathbf{X}^\top \mathbf{q}\mathbf{q}^\top \mathbf{y}\right)$ guesses correctly w.p. $\sim \frac{1}{2} + \frac{1}{\sqrt{d}}$

# Modified Attention Construction: $L = 2,\ N = 2$

- Random rank-1 head $\mathrm{hm}\left(\mathbf{X}^\top \mathbf{q}\mathbf{q}^\top \mathbf{y}\right)$ guesses correctly w.p. $\sim \frac{1}{2} + \frac{1}{\sqrt{d}}$
- Majority vote of many such heads is correct with high probability

# Modified Attention Construction: $L = 2,\ N = 2$

- Random rank-1 head $\mathrm{hm}\left(\mathbf{X}^\top \mathbf{q}\mathbf{q}^\top \mathbf{y}\right)$ guesses correctly w.p. $\sim \frac{1}{2} + \frac{1}{\sqrt{d}}$
- Majority vote of many such heads is correct with high probability
- To tally the votes, need extra “index” and “scratchpad” dimensions
  - Input: $\begin{bmatrix} \mathbf{x}_1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{x}_2 \\ -1 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{y} \\ 0 \\ 0 \end{bmatrix}$

# Modified Attention Construction: $L = 2,\ N = 2$

- Random rank-1 head $\mathrm{hm}\left(\mathbf{X}^\top \mathbf{q}\mathbf{q}^\top \mathbf{y}\right)$ guesses correctly w.p. $\sim \frac{1}{2} + \frac{1}{\sqrt{d}}$
- Majority vote of many such heads is correct with high probability
- To tally the votes, need extra “index” and “scratchpad” dimensions
  - Input: $\begin{bmatrix} \mathbf{x}_1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{x}_2 \\ -1 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{y} \\ 0 \\ 0 \end{bmatrix}$
- Attn layer 1: each head votes. Sum the votes in index dimension. Save in $\mathbf{y}$’s scratchpad dimension

# Modified Attention Construction: $L = 2,\ N = 2$

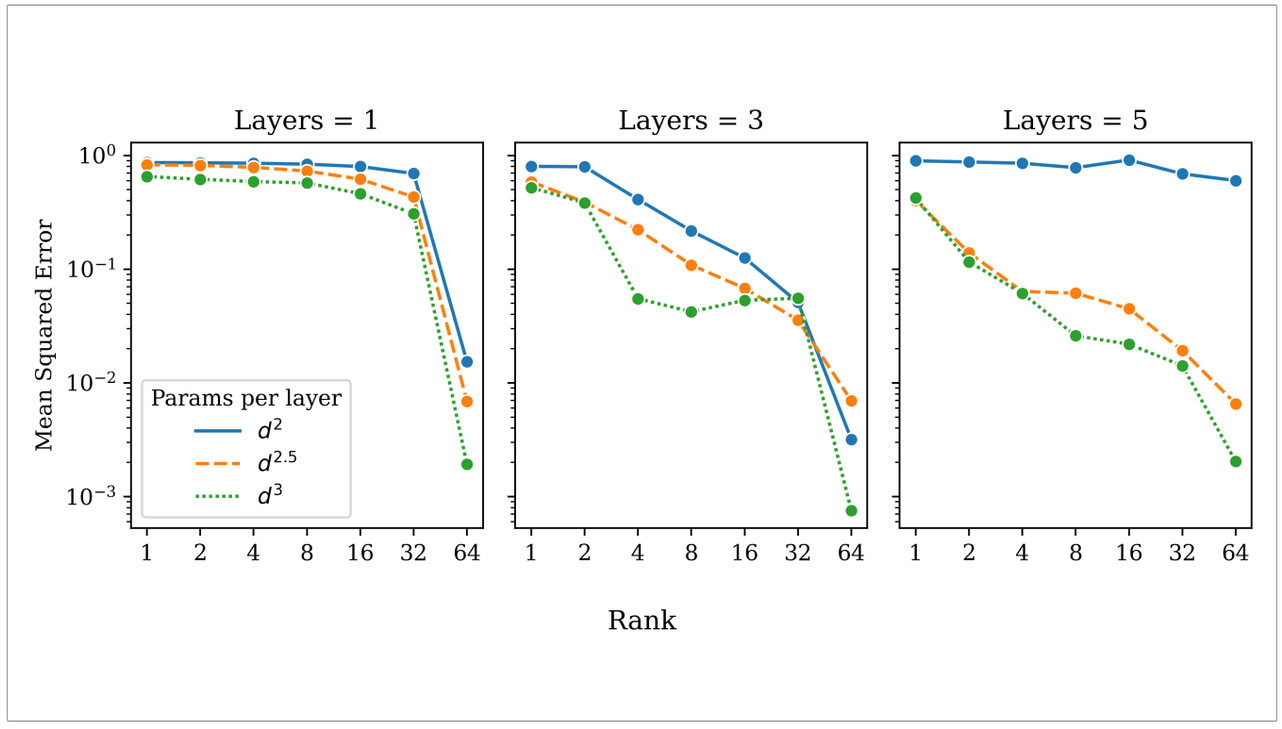
- Random rank-1 head $\mathrm{hm}\left(\mathbf{X}^\top \mathbf{q}\mathbf{q}^\top \mathbf{y}\right)$ guesses correctly w.p. $\sim \frac{1}{2} + \frac{1}{\sqrt{d}}$
- Majority vote of many such heads is correct with high probability
- To tally the votes, need extra “index” and “scratchpad” dimensions
  - Input: $\begin{bmatrix} \mathbf{x}_1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{x}_2 \\ -1 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{y} \\ 0 \\ 0 \end{bmatrix}$
- Attn layer 1: each head votes. Sum the votes in index dimension. Save in $\mathbf{y}$’s scratchpad dimension
- Attn layer 2: look up the target $\mathbf{x}_1$ or $\mathbf{x}_2$ whose sign matches the tally

# Experiments

# Experimental Setup

- Off-the-shelf multilayer transformers from PyTorch (but $H = d^c/r$, RMSNorm)
- "Farthest neighbor" with self-attention $\mathbf{x}_1, \ldots \mathbf{x}_N \sim (\mathbb{S}^{d-1})$
- No positional encodings (yes biases)
- Fix $d = 64, N = 16$
- Best of 5 runs

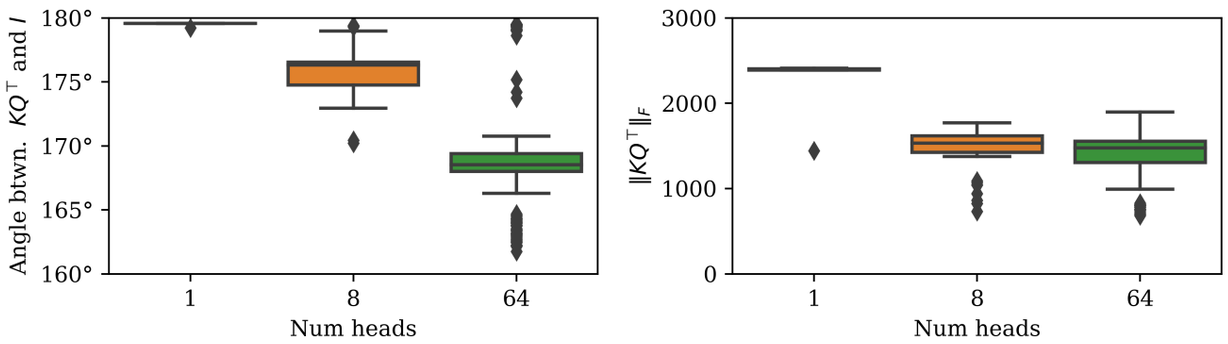
We want to see whether the predictions of our theory bear out for reasonable dimension and error.
We especially want to see what happens when we add more layers. Attention but also MLP, layer norm, etc.
(d^c / r) … SO number of parameters per layer is d^c
Before, we had a single source point y and separate target points x. To mimic the way transformers are usually used, we just have target points. Each one attends to everything and we average the error.
Now the nearest neighbor function is boring since each point would just return itself. So use "farthest neighbor" instead




For each line, we use the same number of parameters per layer.
So the blue line, d^2 is the standard scaling. The x-axis is rank. So on the right (r = 64) we have a single full rank head for blue. or for green, d full rank heads.
On the left, d rank-1 heads for blue or d^2 rank-1 heads for green
Note that this a log-log plot
For one layer, the results are crystal clear: full rank is necessary and sufficient to learn it accurately
For more layers, the low-rank transformers do ok but it’s still clear that they are still much weaker than large rank transformers.
Notice this line (L=2, param = d^2) is junk since we know L=5 shouldn’t really be worse than L = 3. just harder to train

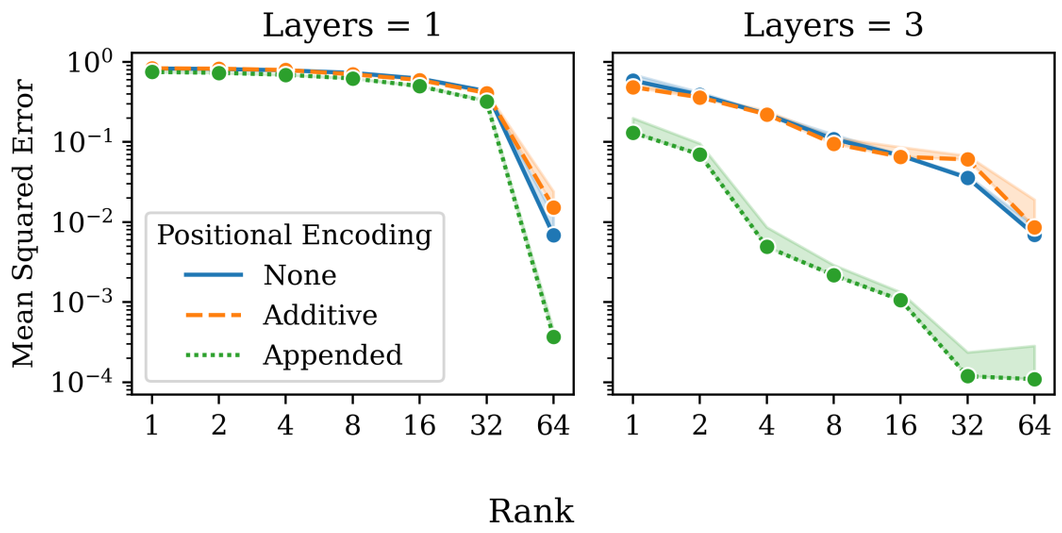
# What does full-rank attention learn?

- Expected: $\mathbf{I}_d\ \mathbf{X}\ \mathrm{sm}\left(\mathbf{X}^\top\left(-10^{10}\cdot\mathbf{I}_d\right)\mathbf{y}\right)$



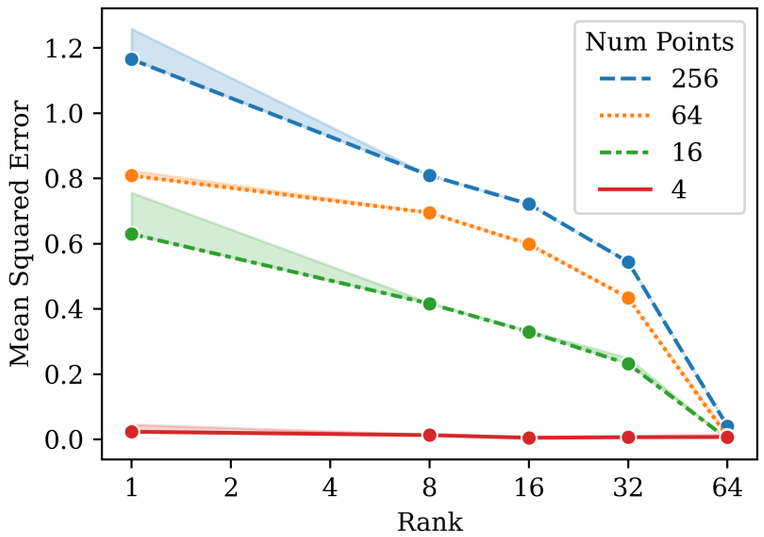

Expected: add a minus sign b/c FARTHEST neighbor
Data comes from the full-rank, single layer models.
Aggregated over 5 runs, each with between 1 and 64 heads
Let’s see how close this learned matrix is to a scaled identity matrix.
On the left, we show Frobenius angle. basically showing that, this matrix is almost exactly parallel with -I
magnitude is big. each entry is like > 30. Big enough to make softmax act like hardmax

# Modifying Attention




Blue is just the same as before
Orange we have learned positional encodings that get added on. This is the standard way to use transformers, and recall our theory covers this, and it doesn't help at all
Green: we append extra dimensions to each input like in the low rank majority vote construction. You can think of these like a position vector that is appended instead of added.
People don't really do that in practice, but maybe they should?
For one layer this really changes nothing. But like in our construction for L > 1, the low-rank head can use these appended positional embeddings to help overcome the low-rank bottleneck. it takes several steps / layers

# Role of $N$




Our lower bound is for N = 2, but this is ok because the problem gets harder as N gets bigger. More distractor points, harder to guess correctly

here, we use two layer network with d^2 / r heads

# Conclusion

Low-rank attention is fundamentally weaker than full-rank attention,

even for $H \gg d/r$

# Open / in progress

- Is the lower bound tight? (pretty much: just use random rank-1 heads)
- Can we prove hardness for $L > 1$? Is our conjecture true?
- Other tradeoffs in transformer hyperparameters besides $r$ and $H$
- Are there hard problems that look more like text (not isotropic?)