Multi-layer SAEs

Authors:
Sid Baskaran, Michael Sklar

———

Title: Brief Investigations of a Multi-layer Sparse Auto-Encoder

Code: https://github.com/sidnb13/tensor-saes

We ran a Multi-Layer Sparse Auto-Encoder on the concatenated activations of some pythia models. Within the same token-position, every feature reads into and out of all layers, simultaneously. Note that this setup is quite different from the 'MLSAE' of Lawson et al. [https://arxiv.org/pdf/2409.04185], which are not multi-layer in our sense. Footnote: their setup could be described as using a single dictionary for several simultaneously-fitted single-layer SAEs.]

We were curious to explore whether our multi-layer SAE would perform a primitive featurization of the model's internal pathways, as well as other brief qualitative investigation. We made relatively few modifications to the backbone of EleutherAI's top-k sparse auto-encoder code. We do not claim that this SAE is high-quality. Our goal was observation for qualitative takeaways, noting that this operation of reading-in from multiple layers and reading-out to multiple layers is a tempting component to include in interpretability tools.

Our main findings:

(1) We had hypothesized in advance that we'd see features corresponding to information dropped-in via attention at one layer, and its flow to subsequent layers. Indeed, we do observe a banding phenomenon where many features are specialized to have a trigger-layer and feed out to that layer and subsequent layers. This specialization is quite visible, but not very clean.

(2) We tried to assess the extent to which these features are capturing internal linear pathways of the model. They do to a small extent, but the capture is not high-quality. There is variation over features in terms of quality. The commonly-activating features' decoder vectors are relatively similar from layer to layer, but are also aligned to capture local causal effects to a small extent. We measured the local gradients to see if adding epsilon times the decoder-direction d_l in layer l will change the network proportionally to d_{l+1} in the subsequent layer. We find that this is often pretty accurate for the layer immediately after the "trigger", but less so for subsequent layers.

(3) We tried adding a bias term to each decoder feature, just for curiosity. This didn’t seem to improve anything at all.

Architecture Description

We started with EleutherAI’s top-k SAE training code and made the following choices/modifications:

-We use hidden state activations of pythia 70M to train the SAE, with every layer normalized
-We concatenate the activations over multiple layers, so that the inputs are of size: d_embed*num_layers
-We use a tied random gaussian initialization for the encoder and decoder vectors of size d_embed * num_layer. We found FVU (“Fraction of Variance Unexplained), which is equivalent to the SAE loss function, was often >> 1 at init, so we scaled encoder weight and bias by c*sqrt(fvu_out/fvu_in) for c = ___ to reduce the hockey-stick learning curve
[define c. also what are fvu_out, fvu_in, exactly?]
-aux-k-loss, we did not use (for some reason this made fvu go up)
-topk:
-dictionary size:

[INSERT Training plot for 70m model]

Our total “Fraction of Variance Unexplained” (fvu) was ~10% [input actual stat]
(note that this is a small model)

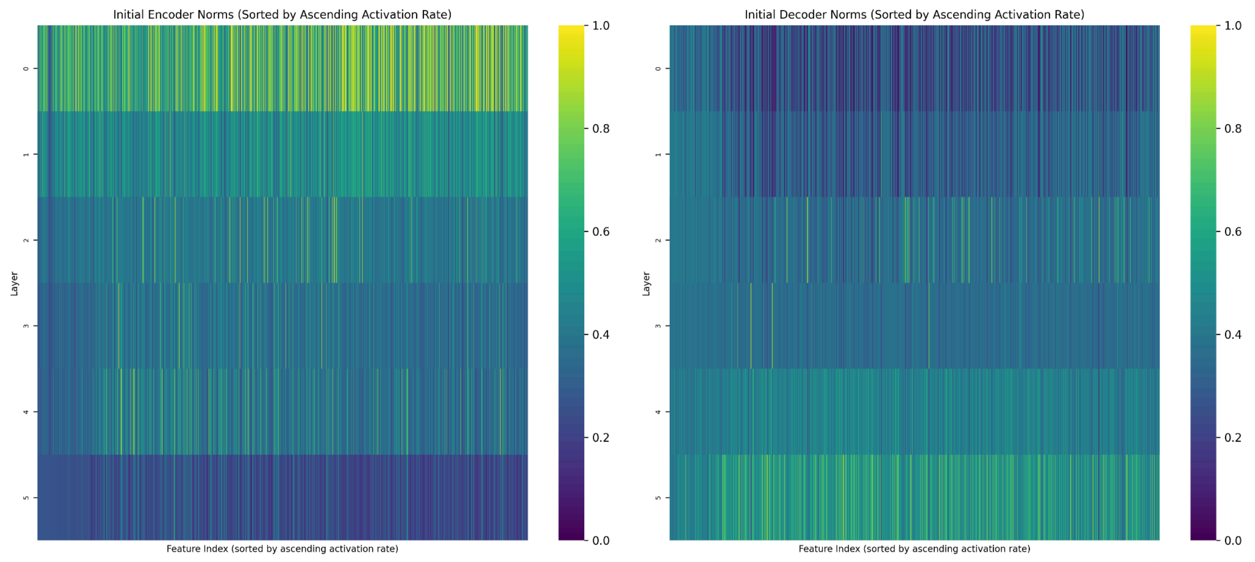
Subsection: Banding

The total norm of the encoder is the sum of norms from its components in each layer,
||enc_f ||^2 = Sum_{l} || enc_{f,l}||^2

When we bucket features according to which of their part-norms enc{f,l}’s is greatest, we observe strong banding.

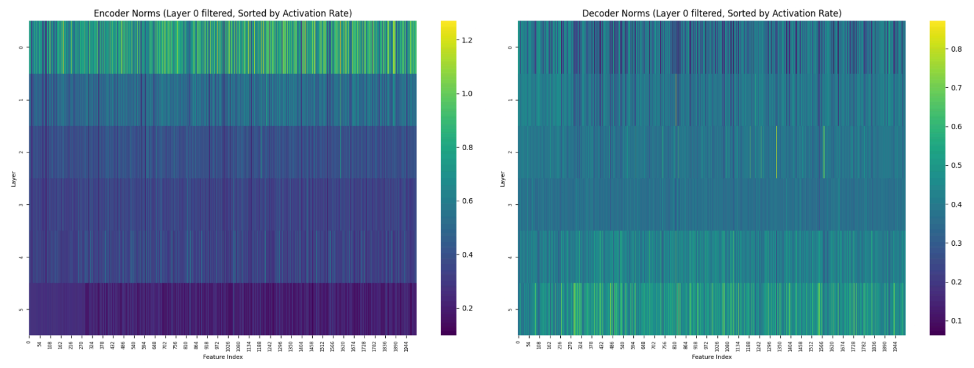
The encoder norms are focused on a particular layer, and in observing the corresponding decoder norms they tend to read out strongly to that same layer and subsequent layers with diminishing effect. The decoder norms tend to write out from the trigger layer onwards.

Here are what the encoder and decoder norms look like, for features with at least 1 activation over 64,000 tokens. [<- SID, IS THAT TRUE?]
This graphic is sorted, with least active features on the left, and most active on the right.



We can now split these according the layer of the encoder which is firing most strongly

Note that layer zero (the embedding layer) has the most such features, hence the bands are thinner:



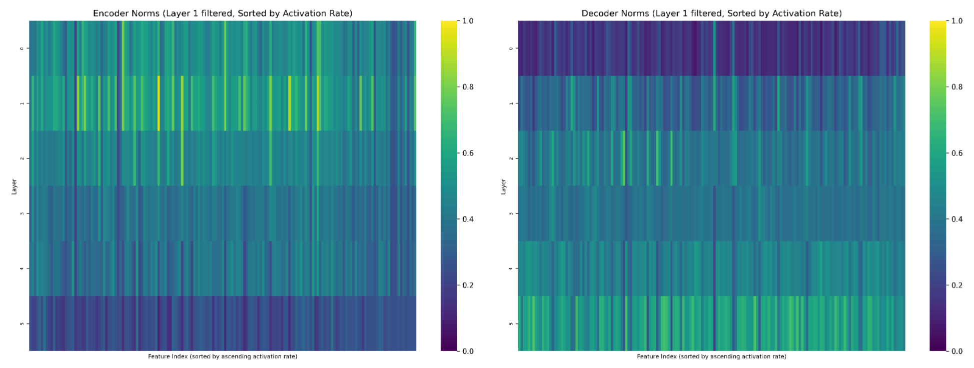
Layer 0 Statistics:

Average Active Features
Per Layer: 1997.00

Percentage Active: 71.68%

Mean Encoder Norm: 0.4254

Mean Decoder Norm: 0.4001



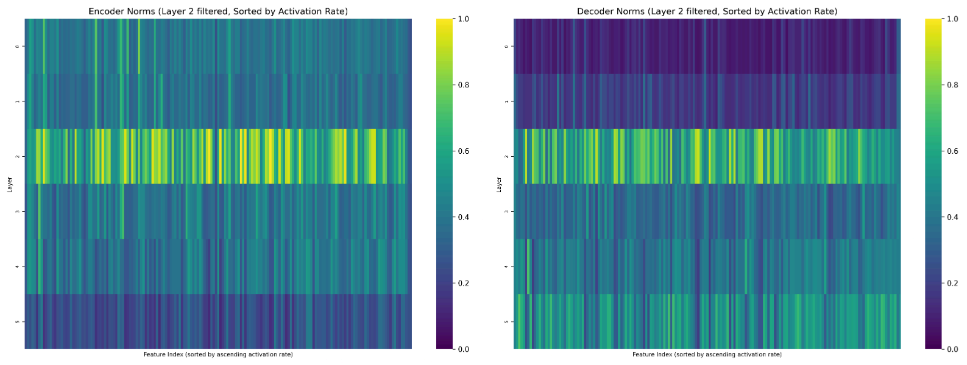
Layer 1 Statistics:

Average Active Features
Per Layer: 163.00

Percentage Active: 5.86%

Mean Encoder Norm: 0.4144

Mean Decoder Norm: 0.3848



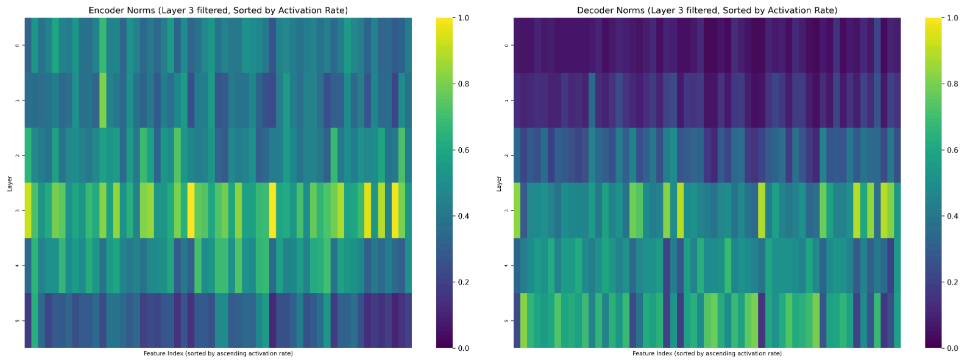
Layer 2 Statistics:

Average Active Features Per Layer: 171.00

Percentage Active: 6.15%

Mean Encoder Norm: 0.4193

Mean Decoder Norm: 0.3592



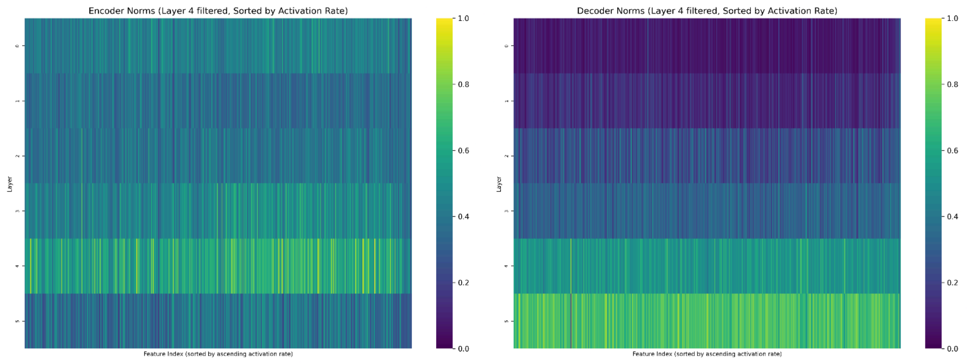
Layer 3 Statistics:

Average Active Features Per Layer: 57.00

Percentage Active: 2.05%

Mean Encoder Norm: 0.4629

Mean Decoder Norm: 0.3438



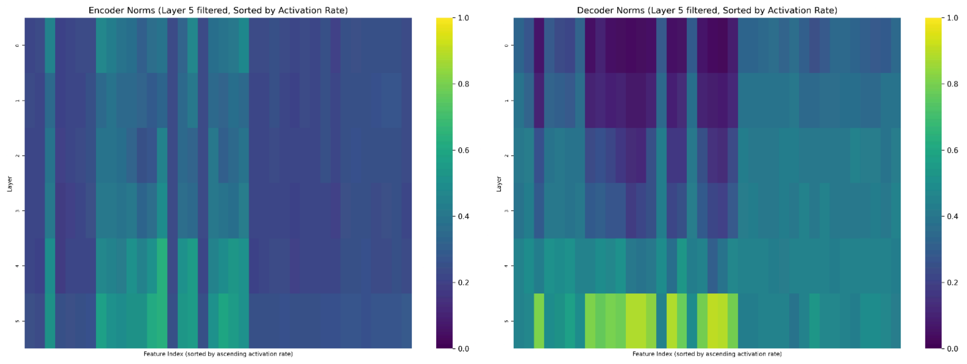
Layer 4 Statistics:

Average Active Features Per Layer: 423.00

Percentage Active: 15.20%

Mean Encoder Norm: 0.4397

Mean Decoder Norm: 0.3386



Layer 5 Statistics:

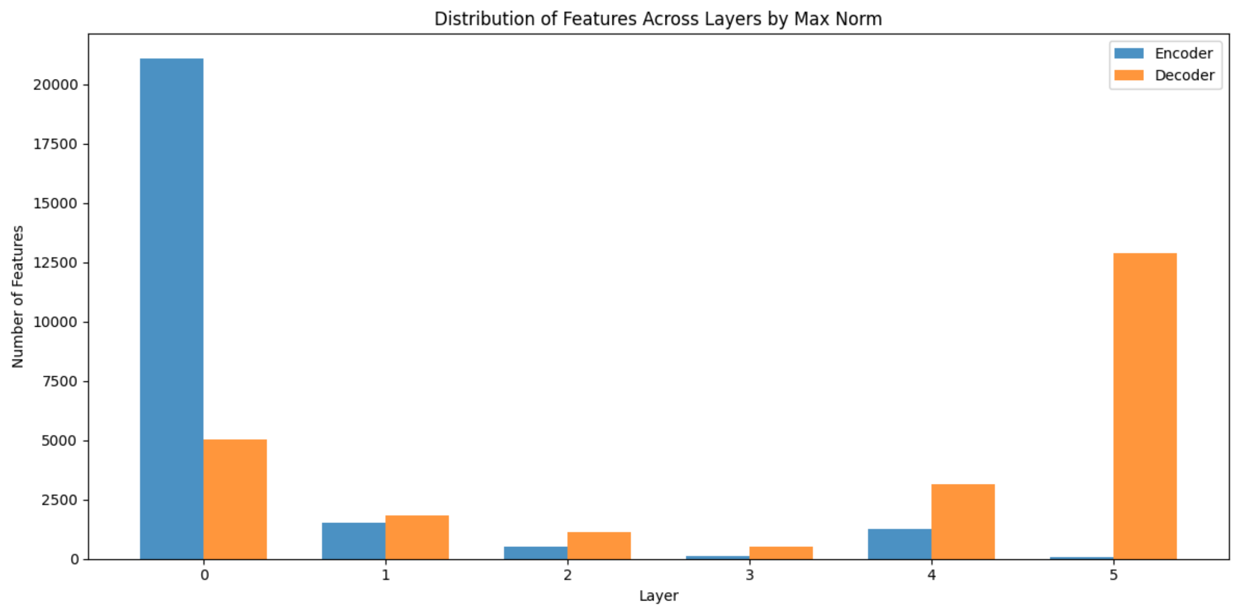
Average Active Features Per Layer: 38.00

Percentage Active: 1.37%

Mean Encoder Norm: 0.3036

Mean Decoder Norm: 0.3692

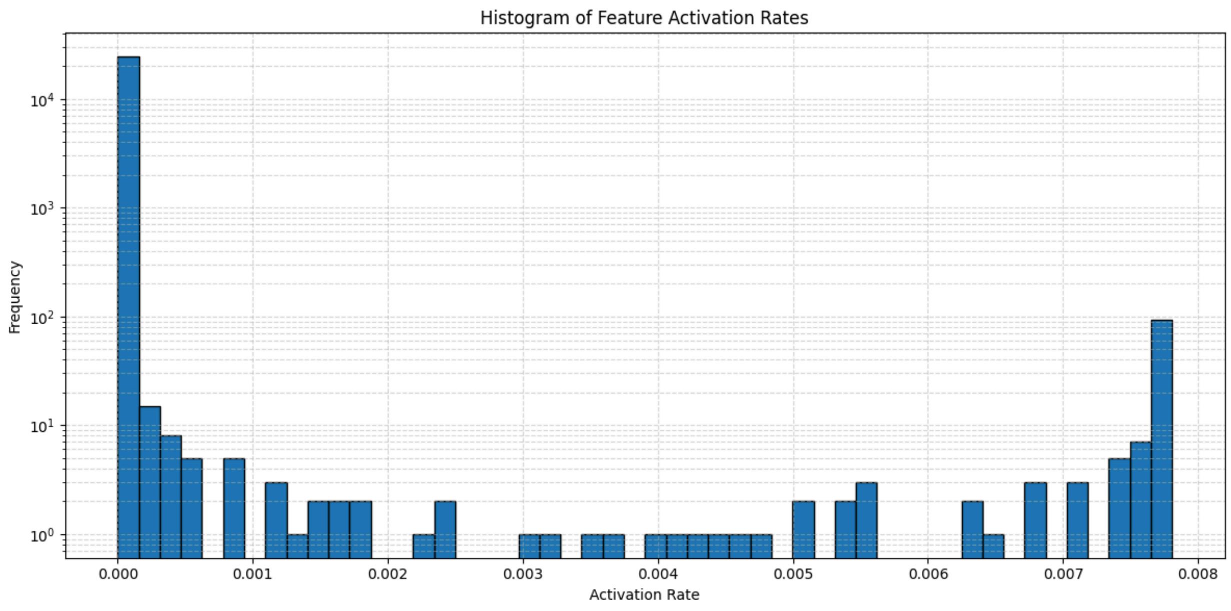
It is a common theme to see the decoders read out most strongly to the `trigger-layer' and all subsequent layers. Why do layers 0, 1, and 4 form a larger number of features, compared to 2, 3, and 5?



We might hypothesize that the layer-0 features are picking up on embeddings. Perhaps layer 4 includes some strong, predictable triggers for the network `outputting its token decisions' ?

Now, the activation distribution. We observe that the majority of features consistently have very low activation rates across a random sample of the dataset.

[Note: Mike is still a bit skeptical about the banding at the upper end.
Would not trust this result without a bit more on-the-ground investigation to check why we're getting such consistent once-per-sequence;
Can you color the plot by trigger-layer-bucket? Are these features at the top-end all triggering in layer 0, and corresponding to the positional embeddings? Then this would make more sense why there is so much reliability! Worth checking that to confirm the pattern]

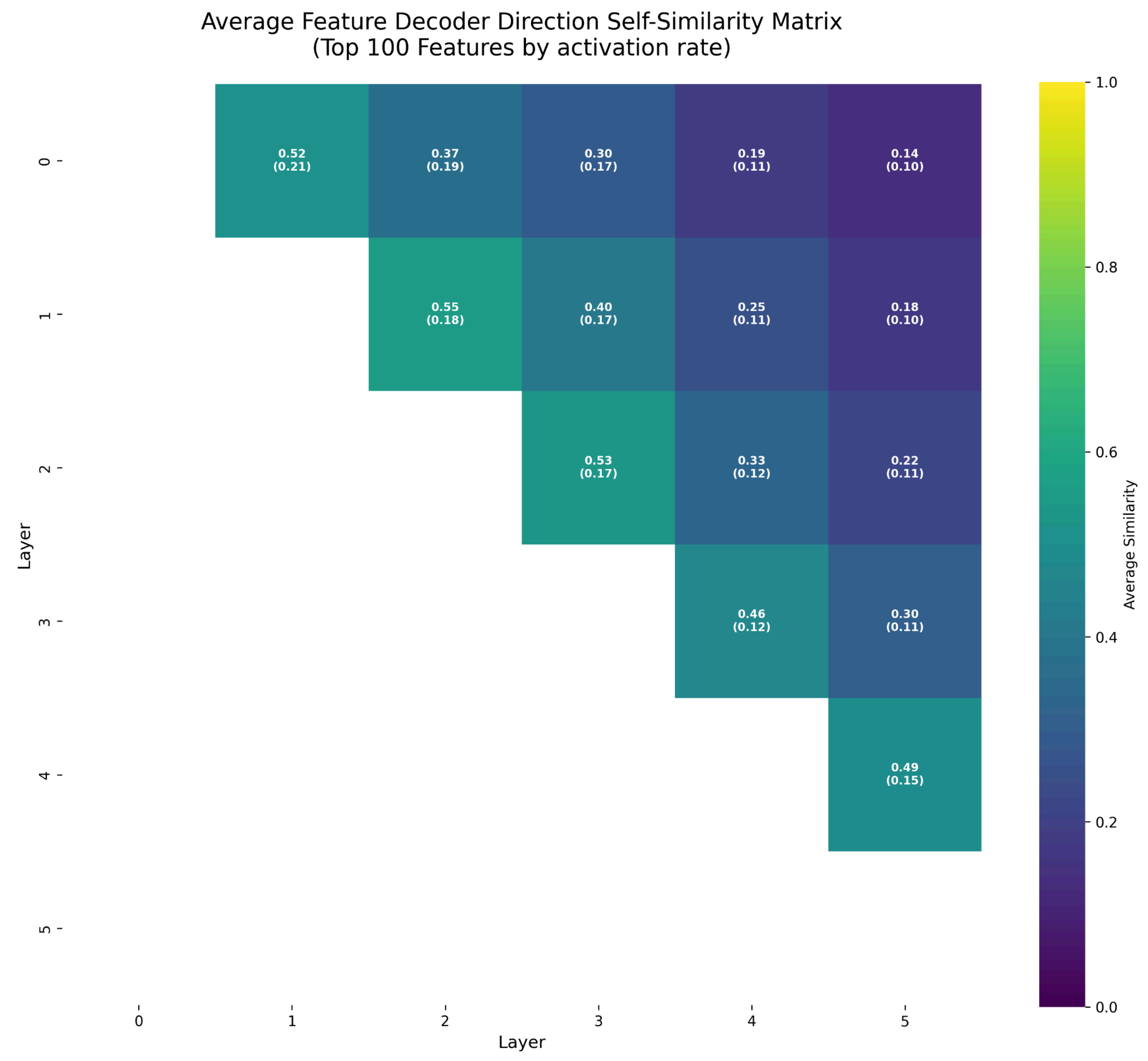


Feature activation rate is computed on a per-token basis for 64K tokens on the test set of RedPajama1T-Sample. We used this to define active vs inactive splits in the graphs below, defining inactive as features not observed over these 64K tokens.

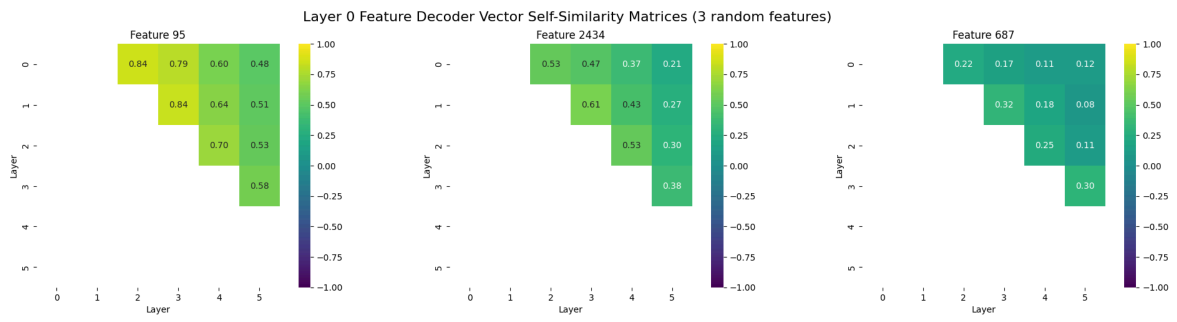
Subsection: Decoder Directional Cosines

The decoder directions are, for most features, pointing in roughly similar directions at different layers. This is not surprising, as the residual stream copies info from each layer to the next, the features ought to capture this identity component.

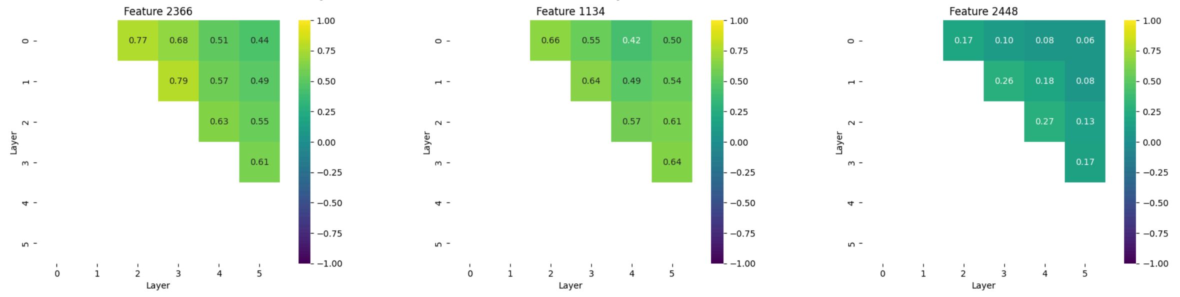
The plot below computes average cosines between the decoders at layer l and l+k, averaged over the 100 most active features. The distribution's SD is in parentheses.



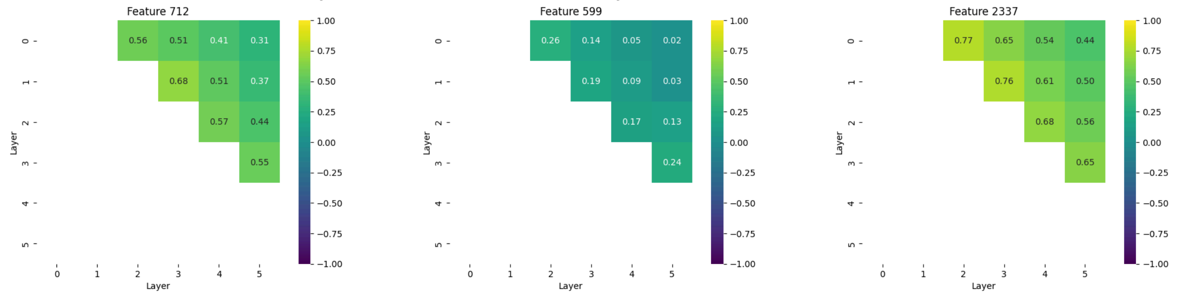
We now present this split-out across our buckets of features by trigger-layer.



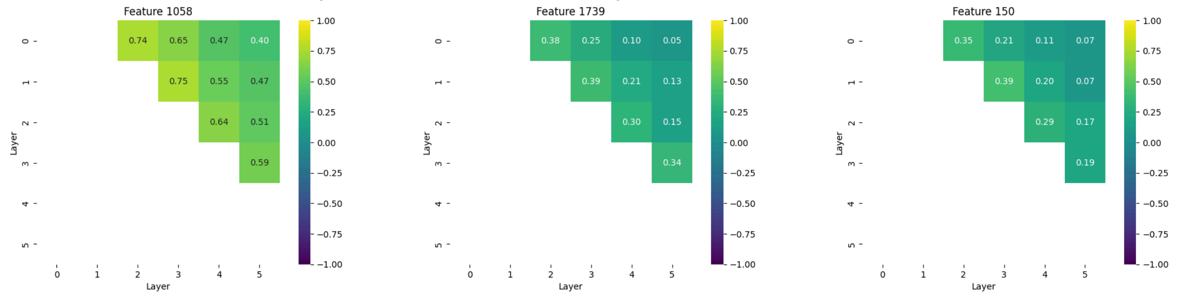
Layer 1 Feature Decoder Vector Self-Similarity Matrices (3 random features)



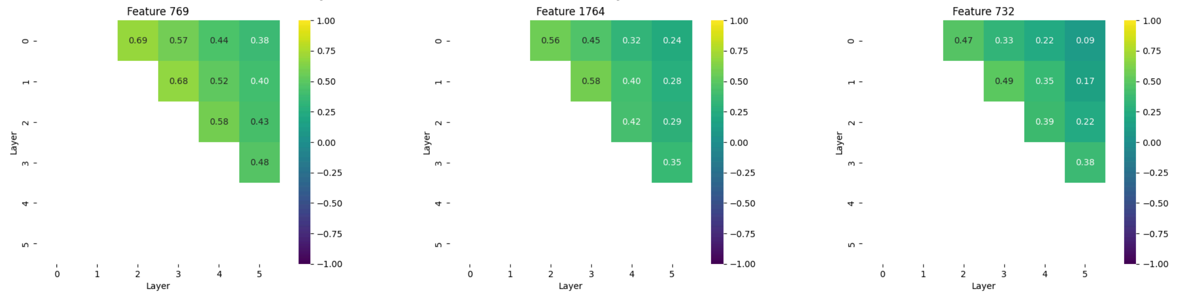
Layer 2 Feature Decoder Vector Self-Similarity Matrices (3 random features)



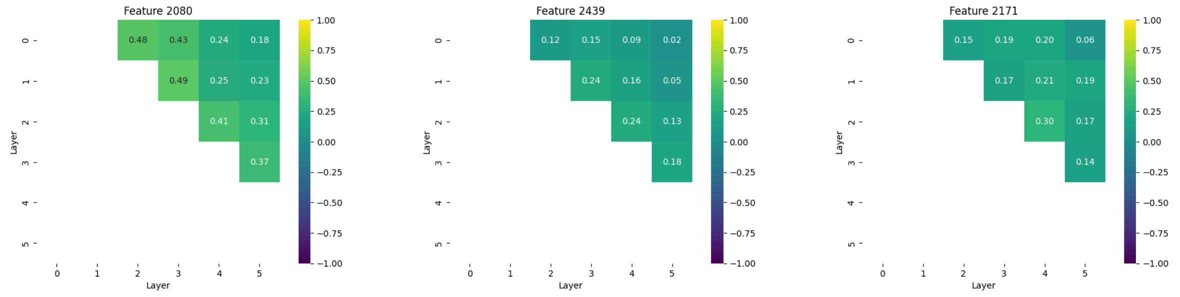
Layer 3 Feature Decoder Vector Self-Similarity Matrices (3 random features)



Layer 4 Feature Decoder Vector Self-Similarity Matrices (3 random features)



Layer 5 Feature Decoder Vector Self-Similarity Matrices (3 random features)



Note: we filtered out zero-activating features. However, there are still some features with relatively low self-cosine.

Subsection: Information Flow

Next, we look at the linear pathways followed by signals.

Specifically, we consider the hypothesis that MLSAE features would capture internal linear structure by encouraging the decoder vectors to align with the “causal gradients”, by which we mean that increasing the activation stream in the direction of *dec_{f,l}* by *epsilon* and tracking its effect to layer *l+k* would roughly have the same effect as moving *epsilon* in direction *dec_{f,l+k}.*

We measure the following quantities:

1. The cosine of *dec_{f,l+k}* with c_{l, l+k, f), the direction in layer l+k “caused” by *dec_{f,l}.*

This causal direction is given by:

c_{l, l+k, f) := VJP(f_{l -> l+k}(h_l) , dec_{f, l})

Where f_{l -> l+k}( ) is the function which maps layer l activations to layer l+k activations.

If the SAE were featurizing local linear pathways in completely perfect fashion, we might hope for a cosine of 1. We don’t see this:

[Insert plot]

2. The *strength* of that causation-direction when projected onto the decoder direction:

   c_{l, l+k, f)^T dec_{f, l_k} / ||dec_{f, l_k}||_2

Even if this cosine direction were typically less than one, we might hope that we are able to perfectly capture the local effect in the direction of dec_{f, l_k} in some average causal sense, yielding a *strength* of 1. We also don’t see this, it’s not capturing the full local-linear effect

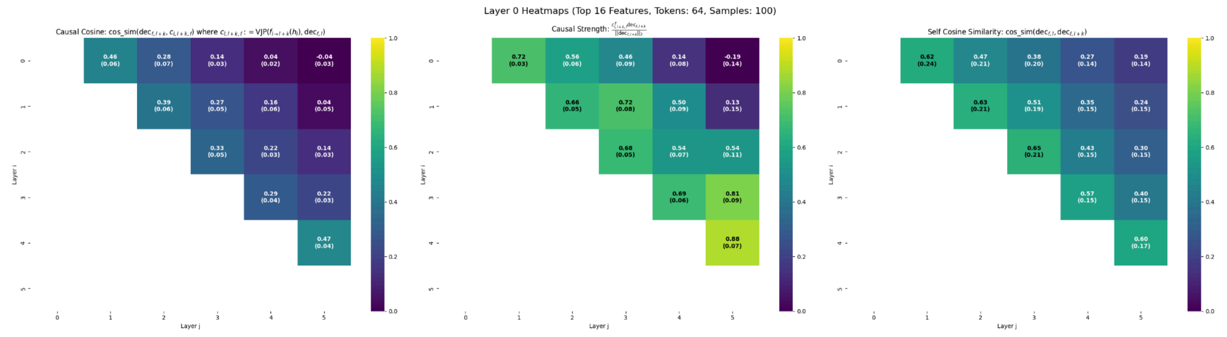
Note here, we consider 64 tokens per input example and average scores over tokens in which the features are active.
Statistics are then computed over 100 examples from the test set. [SID: <- Is this still true or old news?]

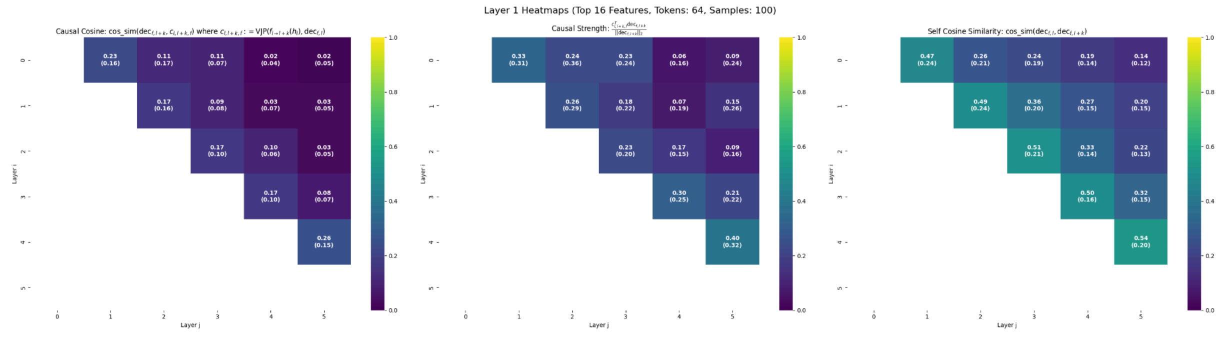
Let’s look at some individual features, and how these cosine and strength metrics vary across individual tokens where the feature fires:

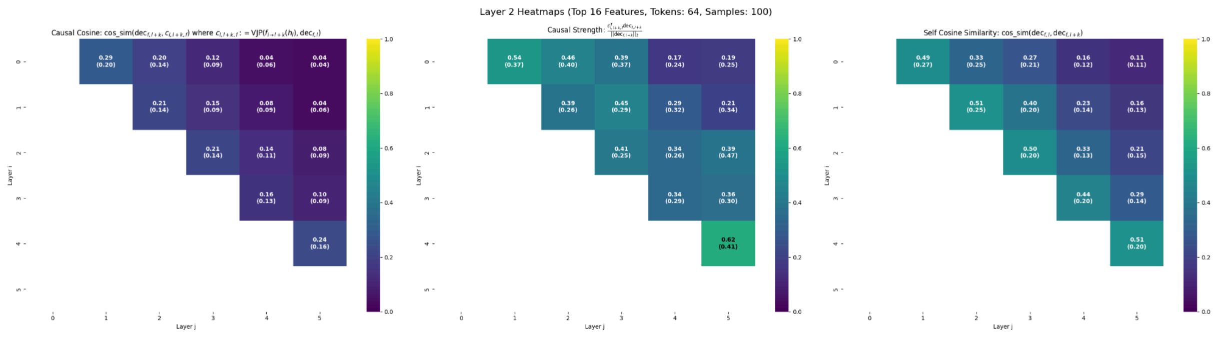
[sid: can you insert this, or put this in the appendix?]

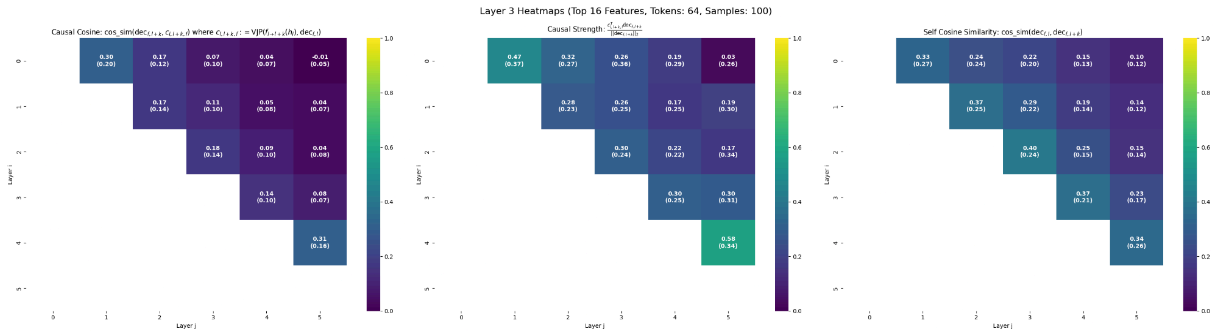
E.g. for the sequence
[' think', ' a', ' clone', ' of', ' hers', ' is', ' still', ' out', ' there', '.', ' She', ' also', ' knows', ' that', ' one', ' of']
we plot the metrics for each trigger layer 1 through 5.

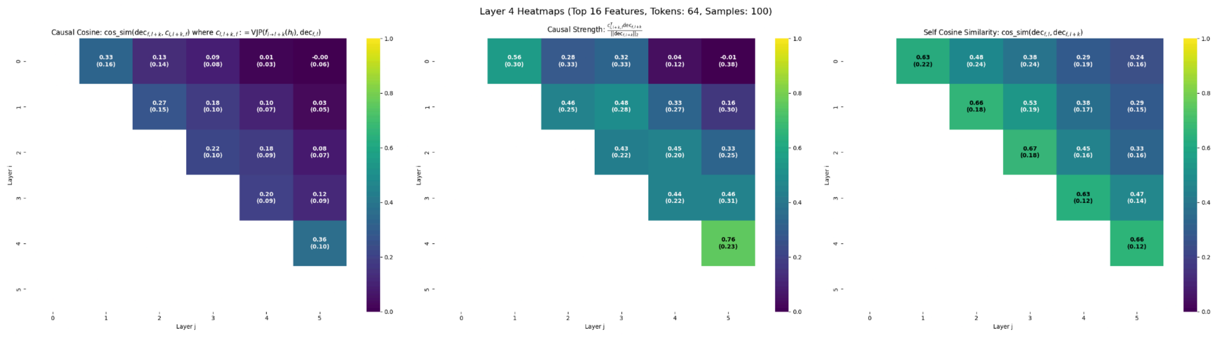
Next, let’s re-aggregate by trigger-bucket and look at averages over the top-16 most active features within the bucket

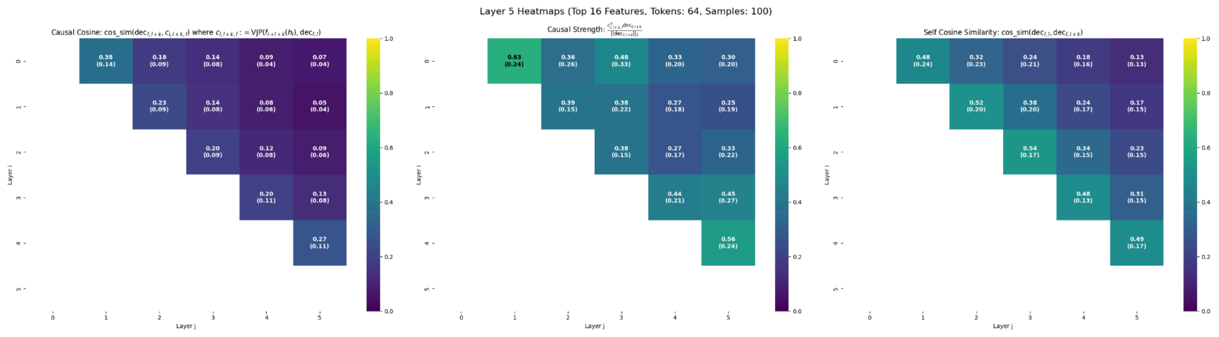








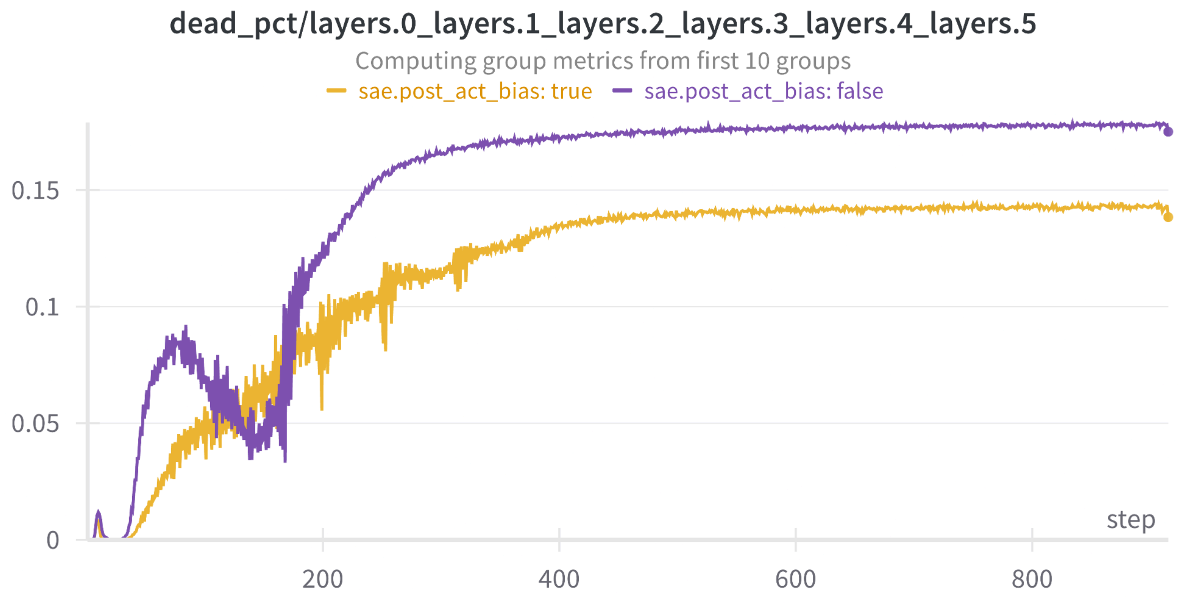




The highly-active layer 0-triggered features seemed to be capturing strong linear effects toward all layers. The highly-active layer 4-triggered features capture strong linear effects toward layer 5 only. In most other cases, the decoder vectors are only weakly aligning the gradient flow from one layer to the next.

Appendix: Experiments with post-activation bias

We tried adding an extra bias to each feature post-activation.
It hurt dead-pct performance and didn't look better on plots.



Insert the token-by-token plot breakdowns here, sid!