Exploring Tuned Lens

Motivation: I learned about Tuned Lenses in a survey of mechanistic interpretability, and though they were conceptually neat so wanted to explore a little more. In theory, they map residual activations to a parallel space which is always the same embedding space, unlike the standard residual stream, and I just wanted to explore the idea.

Background on Tuned Lens: They train an affine probe per block to predict the final distribution, improving robustness over logit lens and aligning intermediate states to a shared predictive space called the “lens-aligned predictive space”.

Note on terminology:
“Tuned Residual Stream” → lens-aligned predictive space: This term is really inaccurate because it’s not actually a stream at all, but I’ve left it for historical purposes.

Hypothesis 0:

Idea
Use singular value decomposition (SVD) on the unembedding matrix W_U to label dimensions of the tuned residual stream by their contribution to the logits.

- Large singular-value directions = **“output subspace”** (direct effect on token distribution).
- Small singular-value directions = **“internal subspace”** (little or no direct effect).

(Note: “internal” was for “internal workspace”, which was an incorrect notion since the space is not a stream, but I keep it for historical reasons)

Limitations

- Depends heavily on lens quality and training data. *(Foreshadowing)*
- Only valid in-distribution (not checked here)

Testing

- Ablate activations in large vs. small singular value dimensions.

- Expect output space ablations to strongly affect logits; internal space ablations to have little effect.

Results

- With my own lenses trained on shit poor data (subset of Wikitext) → failed (Figure H0.1).
- With pretrained TunedLens lenses → worked (Figure H0.2).
- Internal ablations = low KL error across layers (expected).
- Output ablations = effect grows with depth (later layers contribute more directly to logits).

Figure H0.1 (Failure)
Lens invariance (KL base || ablated), per layer:

layer	KL(remove_internal)	KL(remove_output)
0	1.628242	1.112164
1	1.521681	1.093168
2	1.620879	1.144677
3	1.949149	1.288377
4	1.743988	1.392923
5	1.760827	1.430902
6	1.646383	1.430052
7	1.547635	1.555865
8	1.590856	1.633952
9	1.661054	1.816359
10	1.964285	1.667613
11	2.996644	1.504377

- No difference in (my own trained) lens predictive accuracy after ablation of internal vs output activations
- Likely caused by poorly generalized lenses due to data.

Figure H0.2 (Ablations with pretrained lens)
Lens performance (KL, bigger means worse) when you:

Layer	Ablate Internals	Ablate Output
0	KL=2.885	KL=5.726
1	KL=1.258	KL=2.374
2	KL=1.361	KL=3.334
3	KL=1.301	KL=3.806
4	KL=1.307	KL=3.981
5	KL=1.319	KL=4.180

6	KL=1.350	KL=6.137
7	KL=1.493	KL=8.436
8	KL=1.994	KL=7.110
9	KL=1.811	KL=10.030
10	KL=1.398	KL=14.389
11	KL=2.226	KL=15.235

Hypothesis 1: Circuit Ends in Output Subspace

Idea
If an activation projects into the **output subspace**, it behaves like the endpoint of a circuit (directly influencing tokens). If it projects only into the internal subspace, it's the interior/beginning of a circuit.

Limitations

- Need a way to propagate output importance back through the lenses to the residual stream, which is definitely possible but I ran out of time doing math because my linear algebra is rusty.

Testing

- Ablate internal vs. output dims in the *model's own residual stream* (sanity check).
- Planned to identify simple circuits (ie induction heads) and see if the outputs lit up, ran out o time.

Results (Figure H1.1)

- Output ablations consistently have slightly more effect.
- Large gap in final layer, as expected.
- Suggests internal subspace features are useful but less critical.

Figure H1.1 (Sanity Check)
Model performance KL(base || ablated) when we:

Layer	Ablate Internal	Ablate Output
0	10.836	14.311
1	8.766	11.111
2	8.336	10.600

3	7.098	9.395
4	7.097	12.343
5	6.463	11.454
6	5.211	9.556
7	2.784	7.242
8	2.895	8.195
9	2.534	7.584
10	4.865	11.037
11	3.824	30.351

- This is not too surprising assuming the hypothesis, the differences are not that extreme except for the last one

The rest is a mess, read on at your own risk.

Hypothesis 2: if we tune the lenses on the embedding dimension, we can do a similar process and find the beginnings of circuits that take tokens as inputs.

- To tune on the embedding dimension, the lens L_i for layer i will map from W_E to residual activation i, and then we train to minimize KL div from model predictions as before
- I'd need to train this myself but the shitty wikidata I can use in the time limit is clearly not good enough
- I'd be surprised to see very many circuit beginnings which are further back or even in the middle
    - Would be disappointing if they're all far in the front
- Can look at a kinda distribution of circuit inputs and outputs
- Testing this later is the reason why I tried training the lens myself, bc needed to change architecture for this modification, but man that was a poor prioritization.

Vague idea 1: if the tuned residual space is the same embedding space, and we convert model layers to modules which input and output in the tuned residual space, then we can arbitrarily reorder layers which aren't reliant on one another

- Need to figure out exactly how to modularize layer, maybe use lens transpose
- Could use shuffle invariance as a metric for how much one component impacts the other
- **This was based on incorrect idea of the space being a stream**

Vague idea 2: could generalize from using the model's embedding and unembedding matrices to more general embedding and unembedding components. Like using an unembedding matrix and masking tokens which aren't relevant to some topic, or like using the back half of the model as the embedding component and tuning lenses to the first half.

- Maybe can use smartly selected embedding and unembedding components to identify the beginnings and ends of circuits that input or output specific features

        - How make unembedding to recognize a feature? Use a linear classifier, this is just linear probing
        - Maybe can use
    - Assuming they're in distribution with the training data used for the lens ofc

NOTE i've been saying component but tuned lens is built to analyze layers, need to test if it actually generalizes to components or if im batshit

- By component I mean some node of some defined computational graph on the model

Application:

- Honestly I don't think this is very useful as is
- maybe could use end and beginning identification to cut down on the search space for all circuits, like in automated circuit finding

Because we're looking at the input space of the unembedding matrix, we can rank the tuned residual stream dims by how much impact they have on logits, we can potentially propagate this info backwards from the lens maps and label neurons based on their direct influence on the final logit

- And if we break the model into a component graph, we can use this to find node contributions to logits, and potentially find the ends of circuits

Challenges:

- Using a compute server with outdated cuda drivers and no root access meant I had to learn and use a lot of outdated pytorch.