C.4 Decoder normalization

After each step we renormalize columns of the decoder to be unit-norm, following Bricken et al. [2023]. This normalization (or a modified L1 term, as in Conerly et al. [2024]) is necessary for L1 autoencoders, because otherwise the L1 loss can be gamed by making the latents arbitrarily small. For TopK autoencoders, the normalization is optional. However, we find that it still improves MSE, so we still use it in all of our experiments.

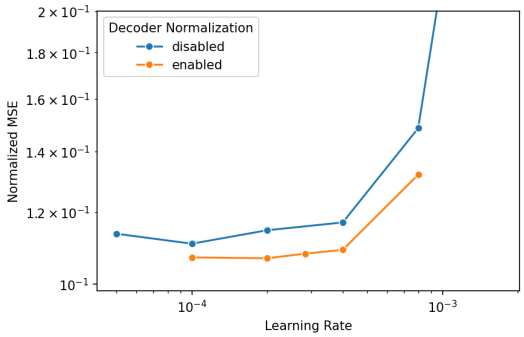

Figure 18: The decoder normalization slightly improves loss (gpt2sm 128k, k=32).

D Systems

Scaling autoencoders to the largest scales in this paper would not be feasible without our systems improvements. Model parallelism is necessary once parameters cannot fit on one GPU. A naive implementation can be an order of magnitude slower than our optimized implementation at the very largest scales.

D.1 Parallelism

We use standard data parallel and tensor sharding [Shoeybi et al., 2019], with an additional allgather for the TopK forward pass to determine which k latents should are in the global top k. To minimize the cost of this allgather, we truncate to a capacity factor of 2 per shard—further improvements are possible but would require modifications to NCCL. For the largest (16 million) latent autoencoder, we use 512-way sharding. Large batch sizes (Section A.4) are very important for reducing the parallelization overhead.

The very small number of layers creates a challenge for parallelism - it makes pipeline parallelism [Huang et al., 2019] and FSDP [Zhao et al., 2023] inapplicable. Additionally, opportunities for communications overlap are limited because of the small number of layers, though we do overlap host to device transfers and encoder data parallel comms for a small improvement.

D.2 Kernels

We can take advantage of the extreme sparsity of latents to perform most operations using substantially less compute and memory than naively doing dense matrix multiplication. This was important when scaling to large numbers of latents, both via directly increasing throughput and reducing memory usage.

We use two main kernels:

- `DenseSparseMatmul`: a multiplication between a dense and sparse matrix
- `MatmulAtSparseIndices`: a multiplication of two dense matrices evaluated at a set of sparse indices

Then, we have the following optimizations:

1. The decoder forward pass uses `DenseSparseMatmul`

2. The decoder gradient uses `DenseSparseMatmul`
3. The latent gradient uses `MatmulAtSparseIndices`
4. The encoder gradient uses `DenseSparseMatmul`
5. The pre-bias gradient uses a trick of summing pre-activation gradient across the batch dimension before multiplying with the encoder weights.

Theoretically, this gives a compute efficiency improvement of up to 6x in the limit of sparsity, since the encoder forward pass is the only remaining dense operation. In practice, we indeed find the encoder forward pass is much of the compute, and the pre-activations are much of the memory.

To ensure that reads are coalesced, the decoder weight matrix must also be stored transposed from the typical layout. We also use many other kernels for fusing various operations for reducing memory and memory bandwidth usage.

E Qualitative results

E.1 Subjective latent quality

Throughout the project, we stumbled upon many subjectively interesting latents. The majority of latents in our GPT-2 small autoencoders seemed interpretable, even on random positive activations. Furthermore, the ablations typically had predictable effects based on the activation conditions. For example, some features that are potentially part of interesting circuits in GPT-2:

- An unexpected token breaking a repetition pattern (A B C D ... A B X!). This upvotes future pattern breaks (A B Y!), but also upvotes continuation of the pattern right after the break token (A B X $\rightarrow$ D).
- Text within quotes, especially activating when within two nested sets of quotes. Upvotes tokens that close the quotes like " or ', as well as tokens which close multiple sets of quotes at once, such as "' and '".
- Copying/induction of capitalized phrases (A B ... A $\rightarrow$ B)

In GPT-4, we tended to find more complex features, including ones that activate on the same concept in multiple languages, or on complex technical concepts such as algebraic rings.

You can explore for yourself at our viewer.

E.2 Finding features

Typical features are not of particular interest, but having an autoencoder also lets one easily find relevant features. Specifically, one can use gradient-based attribution to quickly compute how relevant latents are to behaviors of interest [Baehrens et al., 2010, Simonyan et al., 2013].

Following the methodologies of Templeton et al. [2024] and Mossing et al. [2024], we found features by using a hand-written prompt, with a set of "positive" and "negative" token predictions, and back-propagating from logit differences to latent values. We then consider latents sorted by activation times gradient value, and inspect the latents.

We tried this methodology with a $n = 2^{17} = 131072$, $k = 32$ GPT-2 autoencoder and were able to quickly find a number of safety relevant latents, such as ones corresponding to profanity or child sexual content. Clamping these latents appeared to have causal effect on the samples. For example, clamping the profanity latent to negative values results in significantly less profanity (some profanity can still be observed in situations where it copies from the prompt).

E.3 Latent activation distributions

We anecdotally found that latent activation distributions often have multiple modes, especially in early layers.

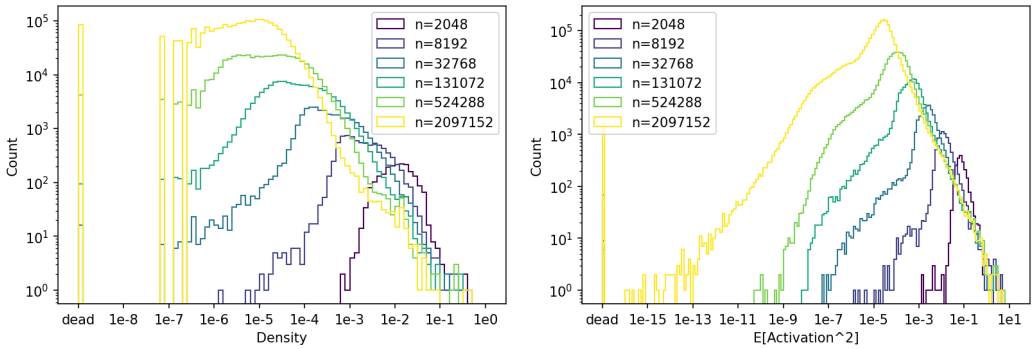

Figure 19: Distributions of latent densities, and average squared activation. Note that we do not observe multiple density modes, as observed in [Bricken et al., 2023]. Counts are sampled over 1.5e7 total tokens. Note that because latents that activate every 1e7 tokens are considered dead during training (and thus receive AuxK gradient updates), 1e-7 is in some sense the minimum density, though the AuxK loss term coefficient may allow it to be lower.

E.4 Latent density and importance curves

We find that log of latent density is approximately Gaussian (Figure 19). If we define the importance of a feature to be the expected squared activation (an approximation of its marginal impact on MSE), then log importance looks a bit more like a Laplace distribution. Modal density and modal feature importance both decrease with number of total latents (for a fixed k), as expected.

E.5 Solutions with dense latents

One option for a sparse autoencoder is to simply learn the k directions that explain the most variance. As k approaches d_{model}, this "principal component"-like solution may become competitive with solutions where each latent is used sparsely. In order to check for such solutions, we can simply measure the average density of the d_{model} densest latents. Using this metric, we find that for some hyperparameter settings, GPT-2 small autoencoders find solutions with many dense latents (Figure 20), beginning around $k = 256$ but especially for $k = 512$.

This coincides with when the scaling laws from Section 3 begin to break - many of our trends between k and reconstruction loss bend significantly. Also, MSE becomes significantly less sensitive to n, at $k = 512$.

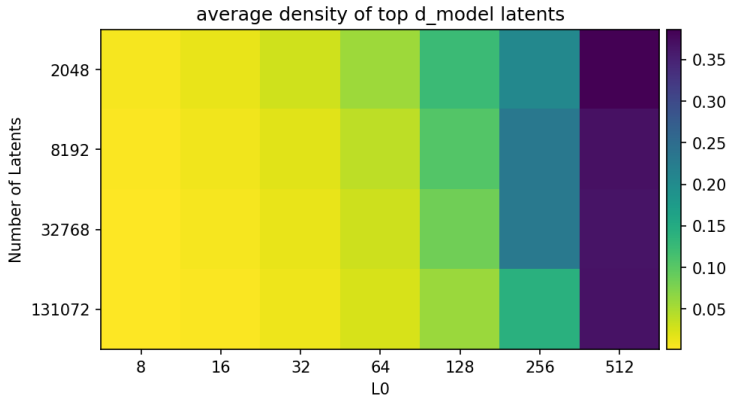

Figure 20: Average density of the d_{model} most-dense features, divided by L_0, for different autoencoders. When $k = 512$, the learned autoencoders have many dense features. This corresponds to when ablations stop having sparse effects Section 4.5, and anecdotally corresponds to noticeably less interpretable features. For $k = 256$, $n = 2048$, there is perhaps an intermediate regime.