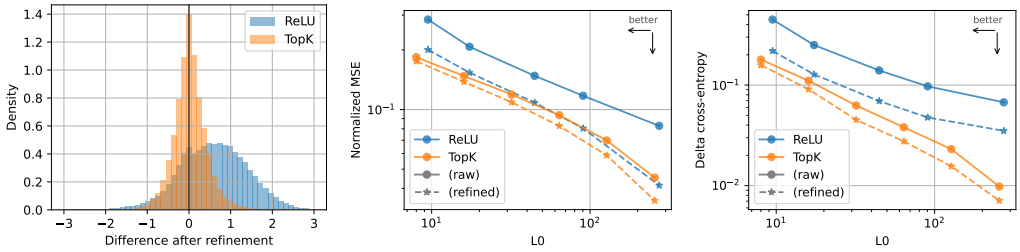

Figure 9: Latent activations can be refined to improve reconstruction from a frozen set of latents. For ReLU autoencoders, the refinement is biased toward positive values, consistent with compensating for the shrinkage caused by the L_1 penalty. For TopK autoencoders, the refinement is not biased, and also smaller in magnitude. The refinement only closes part of the gap between ReLU and TopK.

5.2 Comparison with other activation functions

Other recent works on sparse autoencoders have proposed different ways to address the L_1 activation shrinkage, and Pareto improve the L_0-MSE frontier [Wright and Sharkey, 2024, Taggart, 2024, Rajamanoharan et al., 2024]. Wright and Sharkey [2024] propose to fine-tune a scaling parameter per latent, to correct for the L_1 activation shrinkage. In Gated sparse autoencoders [Rajamanoharan et al., 2024], the selection of which latents are active is separate from the estimation of the activation magnitudes. This separation allows autoencoders to better estimate the activation magnitude, and avoid the L_1 activation shrinkage. Another approach is to replace the ReLU activation function with a ProLU [Taggart, 2024] (also known as TRec [Konda et al., 2014], or JumpReLU [Erichson et al., 2019]), which sets all values below a positive threshold to zero $J_\theta(x) = x \cdot \mathbf{1}_{(x>\theta)}$. Because the parameter θ is non-differentiable, it requires a approximate gradient such as a ReLU equivalent (ProLU-ReLU) or a straight-through estimator (ProLU-STE) [Taggart, 2024].

We compared these different approaches in terms of reconstruction MSE, number of active latents L_0, and downstream cross-entropy loss (Figure 2 and 5). We find that they significantly improve the reconstruction-sparsity Pareto frontier, with TopK having the best performance overall.

5.3 Progressive recovery

In a progressive code, a partial transmission still allows reconstructing the signal with reasonable fidelity [Skodras et al., 2001]. For autoencoders, learning a progressive code means that ordering latents by activation magnitude gives a way to progressively recover the original vector. To study this property, we replace the autoencoder activation function (after training) by a TopK(k') activation function where k' is different than during training. We then evaluate each value of k' by placing it in the L_0-MSE plane (Figure 10).

We find that training with TopK only gives a progressive code up to the value of k used during training. MSE keeps improving for values slightly over k (a result also described in [Makhzani and Frey, 2013]), then gets substantially worse as k' increases (note that the effect on downstream loss is more muted). This can be interpreted as some sort of overfitting to the value k.

5.3.1 Multi-TopK

To mitigate this issue, we sum multiple TopK losses with different values of k (Multi-TopK). For example, using $\mathcal{L}(k) + \mathcal{L}(4k)/8$ is enough to obtain a progressive code over all k' (note however that training with Multi-TopK does slightly worse than TopK at k). Training with the baseline ReLU only gives a progressive code up to a value that corresponds to using all positive latents.

5.3.2 Fixed sparsity versus fixed threshold

At test time, the activation function can also be replaced by a JumpReLU activation, which activates above a fixed threshold θ, $J_\theta(x) = x \cdot \mathbf{1}_{(x>\theta)}$. In contrast to TopK, JumpReLU leads to a selection of active latents where the number of active latents can vary across tokens. Results for replacing the activation function at test-time with a JumpReLU are shown in dashed lines in Figure 10.

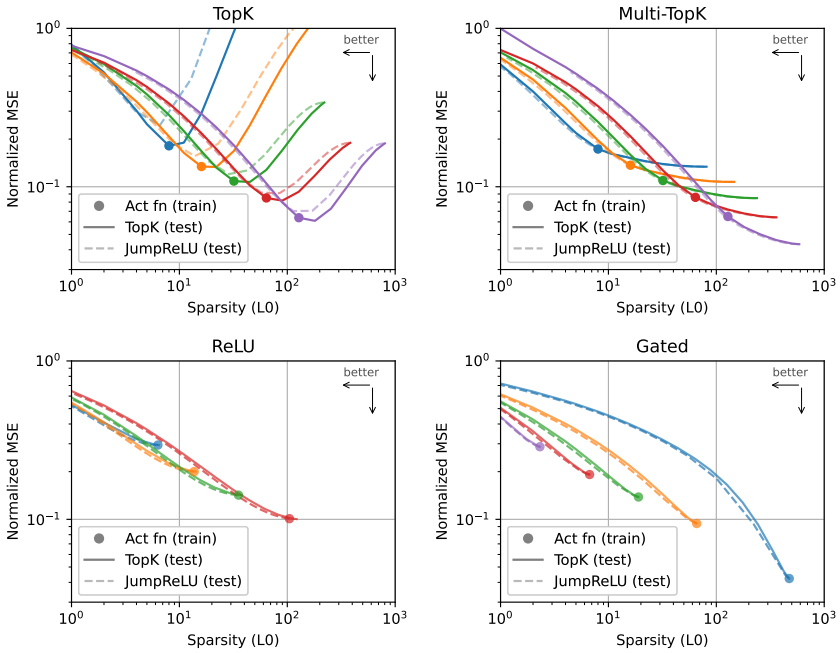

Figure 10: Sparsity levels can be changed at test time by replacing the activation function with either TopK(k) or JumpReLU(θ), for a given value k or θ. TopK tends to overfit to the value of k used during training, but using Multi-TopK improves generalization to larger k.

For autoencoders trained with TopK, the test-time TopK and JumpReLU curves are superimposed only for values corresponding to an L_0 below the training L_0, otherwise the JumpReLU activation is worse than the TopK activation. This discrepancy disappears with Multi-TopK, where both curves are nearly superimposed, which means that the model can be used with either a fixed or a dynamic number of latents per token without loss in reconstruction. The two curves are also superimposed for autoencoders trained with ReLU. Interestingly, it is sometimes more efficient to train a ReLU model with a low L_1 penalty and to use a TopK or JumpReLU at test time, than to use a higher L_1 penalty that would give a similar sparsity level (a result independently described in Nanda et al. [2024]).

6 Limitations and Future Directions

We believe many improvements can be made to our autoencoders.

- TopK forces every token to use exactly k latents, which is likely suboptimal. Ideally we would constrain $\mathbb{E}[L_0]$ rather than L_0.
- The optimization can likely be greatly improved, for example with learning rate scheduling,[16] better optimizers, and better aux losses for preventing dead latents.
- Much more could be done to understand what metrics best track relevance to downstream applications, and to study those applications themselves. Applications include: finding vectors for steering behavior, doing anomaly detection, identifying circuits, and more.
- We're excited about work in the direction of combining MoE [Shazeer et al., 2017] and autoencoders, which would substantially improve the asymptotic cost of autoencoder training, and enable much larger autoencoders.
- A large fraction of the random activations of features we find, especially in GPT-4, are not yet adequately monosemantic. We believe that with improved techniques and greater scale[17] this is potentially surmountable.

[16] Anecdotally, we also found that lowering learning rates helped with decreasing dead latents.

[17] both in number of latents and in training tokens

- Our probe based metric is quite noisy, which could be improved by having a greater breadth of tasks and higher quality tasks.
- While we use n2g for its computational efficiency, it is only able to capture very simple patterns. We believe there is a lot of room for improvement in terms of more expressive explanation methods that are also cheap enough to simulate to estimate explanation precision.
- A context length of 64 tokens is potentially too few tokens to exhibit the most interesting behaviors of GPT-4.

7 Related work

Sparse coding on an over-complete dictionary was introduced by Mallat and Zhang [1993]. Olshausen and Field [1996] refined the idea by proposing to learn the dictionary from the data, without supervision. This approach has been particularly influential in image processing, as seen for example in [Mairal et al., 2014]. Later, Hinton and Salakhutdinov [2006] proposed the autoencoder architecture to perform dimensionality reduction. Combining these concepts, sparse autoencoders were developed [Lee et al., 2007, Le et al., 2013, Konda et al., 2014] to train autoencoders with sparsity priors, such as the L_1 penalty, to extract sparse features. Makhzani and Frey [2013] refined this concept by introducing k-sparse autoencoders, which use a TopK activation function instead of the L_1 penalty. Makelov et al. [2024] evaluates autoencoders using a metric that measures recovery of features from previously discovered circuits.

More recently, sparse autoencoders were applied to language models [Yun et al., 2021, Lee Sharkey, 2022, Bricken et al., 2023, Cunningham et al., 2023], and multiple sparse autoencoders were trained on small open-source language models [Marks, 2023, Bloom, 2024, Mossing et al., 2024]. Marks et al. [2024] showed that the resulting features from sparse autoencoders can find sparse circuits in language models. Wright and Sharkey [2024] pointed out that sparse autoencoders are subject to activation shrinking from L_1 penalties, a property of L_1 penalties first described in Tibshirani [1996]. Taggart [2024] and Rajamanoharan et al. [2024] proposed to use different activation functions to address activation shrinkage in sparse autoencoders. Braun et al. [2024] proposed to train sparse autoencoders on downstream KL instead of reconstruction MSE.

Kaplan et al. [2020] studied scaling laws for language models which examine how loss varies with various hyperparameters. Clark et al. [2022] explore scaling laws related to sparsity using a bilinear fit. Lindsey et al. [2024] studied scaling laws specifically for autoencoders, defining the loss as a specific balance of reconstruction and sparsity (rather than simply reconstruction, while holding sparsity fixed).

Acknowledgments

We are deeply grateful to Jan Leike and Ilya Sutskever for leading the Superalignment team and creating the research environment which made this work possible. We thank David Farhi for supporting our work after their departure.

We thank Cathy Yeh for explorations in finding features. We thank Carroll Wainwright for initial insights into clustering latents.

We thank Steven Bills, Dan Mossing, Cathy Yeh, William Saunders, Boaz Barak, Jakub Pachocki, and Jan Leike for many discussions about autoencoders and their applications. We thank Manas Joglekar for some improvements on an earlier version of our activation loading. We thank William Saunders for an early autoencoder visualization implementation.

We thank Trenton Bricken, Dan Mossing, and Neil Chowdhury for feedback on an earlier version of this manuscript.

We thank Trenton Bricken, Lawrence Chan, Hoagy Cunningham, Adam Goucher, Ryan Greenblatt, Tristan Hume, Jan Hendrik Kirchner, Jake Mendel, Neel Nanda, Noa Nabeshima, Chris Olah, Logan Riggs, Fabien Roger, Buck Shlegeris, John Schulman, Lee Sharkey, Glen Taggart, Adly Templeton, Vikrant Varma for valuable discussions.