Sparse but Wrong: Incorrect L0 Leads to Incorrect Features in Sparse Autoencoders

Anonymous Author(s)
Affiliation
Address
email

Abstract

Sparse Autoencoders (SAEs) extract features from LLM internal activations, meant to correspond to single concepts. A core SAE training hyperparameter is L0: how many features should fire per token on average. Existing work compares SAE algorithms using sparsity–reconstruction tradeoff plots, implying L0 is a free parameter with no single correct value. In this work we study the effect of L0 on BatchTopK SAEs, and show that if L0 is not set precisely, the SAE fails to learn the underlying features of the LLM. If L0 is too low, the SAE will mix correlated features to improve reconstruction. If L0 is too high, the SAE finds degenerate solutions that also mix features. Further, we demonstrate a method to determine the correct L0 value for an SAE on a given training distribution, which finds the true L0 in toy models and coincides with peak sparse probing performance in LLMs. We find that most commonly used SAEs have an L0 that is too low. Our work shows that, to train SAEs with correct features, practitioners must set L0 correctly.

1 Introduction

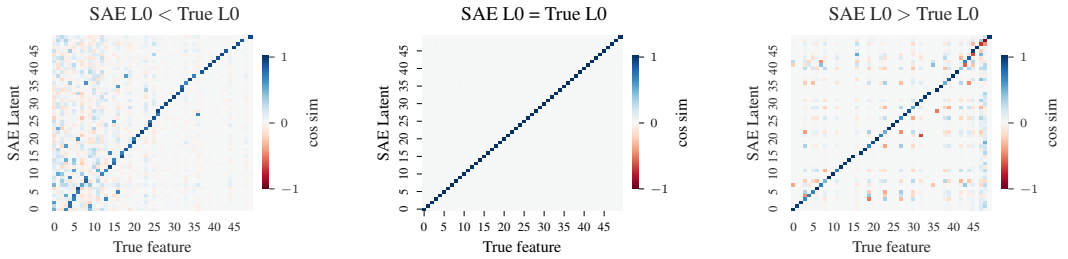
Sparse autoencoders (SAEs) decompose the dense, polysemantic activations of LLMs into interpretable latent features [7, 2] using sparse dictionary learning [14]. SAEs have the advantage of being unsupervised, and can be scaled to millions of neurons in its hidden layer (hereafter called "latents") [17, 10]. When training an SAE, practitioners must decide on the L0 of the SAE: that is, the sparsity; how many latents activate on average for a given input.[1] L0 is typically considered a neutral design choice: most of the literature evaluates SAEs at a range of L0 values, referring to this as a "sparsity–reconstruction tradeoff" [10, 15], implying any L0 is equally valid.

However, recent work shows the same trend: too low an L0 leads to worse SAE performance on downstream tasks [11, 4]. In this work, we explore the effect of L0 on SAEs. Starting by exploring a toy model with correlated features, we demonstrate that if the L0 is too low, the SAE can "cheat" by mixing together components of correlated features, achieving better reconstruction compared to an SAE with correct, disentangled features. We consider this to be a manifestation of feature hedging [6], where the SAE abuses feature correlations to compensate for insufficient resources to correctly model the underlying features. Furthermore, we show that if the L0 is too high, the SAE can learn degenerate mixtures of features. We show that it is possible to determine the correct L0 of an SAE by observing projection magnitude between the SAE decoder and training activations.

We validate these findings on Gemma-2-2b [16], demonstrating that the same decoder patterns we observe in our toy model experiments also manifest in LLM SAEs. We further validate that the optimal L0 we find with our method in Gemma-2-2b matches with peak performance on sparse

[1] TopK and BatchTopK SAEs [10, 3] set the L0 (K) directly, whereas L1 and JumpReLU [7, 2, 15] adjust it via a coefficient in the loss. In any case, all SAE trainers must decide on the target L0.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.



(a) SAE L0 = 5 (b) SAE L0 = 11 (c) SAE L0 = 18

Figure 1: SAE decoder cosine similarity with true features. True features L0 = 11.

probing tasks [11]. Our findings show that L0 must be set correctly for SAEs to learn correct features, and implies that most SAEs used by researchers today have too low an L0.

2 Background

Sparse autoencoders (SAEs). An SAE decomposes an input activation $\mathbf{x} \in \mathbb{R}^d$ into a hidden state $\mathbf{a}$ consisting of h hidden neurons, called "latents". An SAE is composed of an encoder $\mathbf{W}_{\text{enc}} \in \mathbb{R}^{h \times d}$, a decoder $\mathbf{W}_{\text{dec}} \in \mathbb{R}^{d \times h}$, a decoder bias $\mathbf{b}_{\text{dec}} \in \mathbb{R}^d$, and encoder bias $\mathbf{b}_{\text{enc}} \in \mathbb{R}^h$, and a nonlinearity σ, typically ReLU or a variant like JumpReLU [15], TopK [10] or BatchTopK [3].

$$\mathbf{a} = \sigma(\mathbf{W}_{\text{enc}}(\mathbf{x} - \mathbf{b}_{\text{dec}}) + \mathbf{b}_{\text{enc}}) \tag{1}$$

$$\hat{\mathbf{x}} = \mathbf{W}_{\text{dec}}\mathbf{a} + \mathbf{b}_{\text{dec}} \tag{2}$$

3 Toy model experiments

We set up a toy model with 50 mutually orthogonal true features $F = \{f_0, \ldots, f_{49}\} \in \mathbb{R}^{100}$. Each feature f_i has firing probability P_i. We set $P(f_0) = 0.345$, and we linearly decrease P_i to $P_{49} = 0.05$, so feature firing probability decreases with feature number. We randomly generate a correlation matrix, so the firings of each feature are correlated with other features. Features fire with magnitude $\sim \mathcal{N}(1.0, 0.15)$. We sampling true feature firings, and summing all firing features to create a training input for the SAE. The goal of the SAE, then, is to learn these true features despite only being trained on their summed activations. Since we have ground-truth knowledge, we know the true number of features firing on average. We call this the *true L0*, which is 11 for this toy model.

Throughout this work, we use BatchTopK SAEs [3], as this architecture allows directly controlling L0 rather than controlling it indirectly via a L1 coefficient [7, 2] or L0 coefficient [15] in the loss. For these toy model experiments, we train SAEs on 15M synthetic samples using SAELens [1].

We train SAEs with L0 that is too low (L0=5), exactly correct (L0=11), and too high (L0=18). Results are shown in Figure 1. When the SAE L0 matches the true L0 (Figure 1b), the SAE exactly learns the true features. When the SAE L0 is too low (Figure 1a), the SAE mixes components of correlated features together, especially breaking latents tracking high-frequency features. When SAE L0 is too high (Figure 1c), the SAE learns degenerate solutions that mix features together. The further the L0 is from the true L0, the more broken the SAE becomes. Interestingly, when L0 is too high the SAE still learns many correct latents, but *when L0 is too low, every latent in the SAE is affected*.

3.1 MSE loss incentivizes low-L0 SAEs to learn incorrect features

Why do SAEs with too low L0 not learn the true features? We take the correct SAE from Figure 1b and set L0=5, to match the SAE from Figure 1a. We then generate 100k training samples and calculate the Mean Square Error (MSE) of both these SAEs. The SAE with incorrect latents from Figure 1a achieves a MSE of 2.73, while the SAE with ground-truth correct latents achieves a much worse MSE of 4.88. Thus, *MSE loss actively incentivizes low L0 SAEs to learn incorrect latents*. Furthermore, this shows that using reconstruction error as a target, and thus the sparsity–reconstruction tradeoff

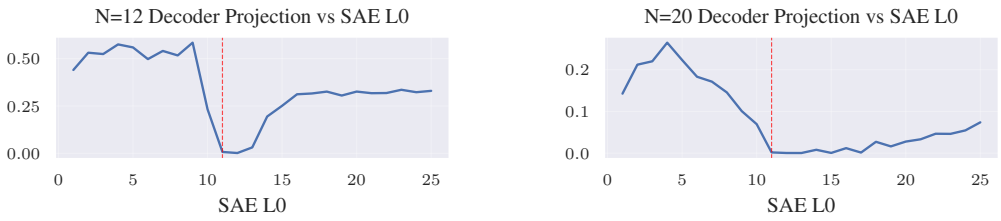


Figure 2: N[th] decoder projection vs SAE L0 for N=12 (left) and N=20 (right) on our toy model SAEs. The true L0, 11, is marked by a dotted line on the plots. Both settings of N are minimized at the true L0, although the slopes of the metric are slightly different depending on N.

[10, 15, 7] is harmful: when the L0 is too low, the incorrect SAE achieves better reconstruction than the ground-truth correct SAE. We explore this further in Appendix A.2.

3.2 Detecting the true L0 using the SAE decoder

Figure 1 reveals that the SAE decoder latents contain mixes of underlying features, both when the L0 is too high and also when it is too low. As the SAE approaches the correct L0, each SAE latent has fewer components of multiple true features mixed in, becoming more monosemantic. Thus, we expect that when the SAE has the correct L0, most latents should have near zero projection on arbitrary training inputs, because they usually do not contain the feature being tracked by that latent. If we are far from the correct L0, then SAE latents contain components of many underlying features, and we expect latents to project unexpectedly strongly on arbitrary training inputs.

We now define a metric we call *N[th] decoder projection score*, or s_n^{dec}, that we can minimize to find the optimal L0 of the SAE. Given SAE inputs $\mathbf{x} \in \mathbb{R}^{b \times d}$ where b is the batch size and d is the input dimension, we first compute the decoder projections for all latents:

$$\mathbf{Z} = (\mathbf{x} - \mathbf{b}_{\text{dec}})\mathbf{W}_{\text{dec}}^{\top} \in \mathbb{R}^{b \times h} \tag{3}$$

where $\mathbf{b}_{\text{dec}} \in \mathbb{R}^d$ is the decoder bias and $\mathbf{W}_{\text{dec}} \in \mathbb{R}^{d \times h}$ is the decoder weight matrix with h latent dimensions. To aggregate across the batch, we flatten $\mathbf{Z}$ to obtain $\mathbf{z} \in \mathbb{R}^{bh}$ and sort these values in descending order to get $\mathbf{z}_{\downarrow}$. The N[th] decoder projection is then defined as:

$$s_n^{\text{dec}} = \mathbf{z}_{\downarrow}[n \cdot b] \tag{4}$$

where $n \in \{1, 2, \ldots, h\}$ indexes the latent by its ranking. The multiplication by b accounts for the batch dimension, effectively selecting the n^{th} highest projection value when considering all samples in the batch. For this to work n should be sufficiently larger than a reasonable guess at the correct L0, as in a perfect SAE, the decoder for these latents should be uncorrelated with input activations. Empirically, picking n up to about $h/2$ seems to work well.

We calculate s_n^{dec} for $n = 18$ and $n = 22$, with SAE L0 ranging from 2 to 25 in Figure 2. We see that the metric is minimized at the true L0, 11, in both cases, although the slope of the metric changes depending on n. In both cases, the slope of s_n^{dec} is flat when L0 is slightly higher than the true L0.

4 LLM experiments

Next, we train a series of BatchTopK SAEs [3] on Gemma-2-2b [16] layer 12 with different L0 values and calculate s_n^{dec} for these SAEs. Each SAE is trained on 500M tokens from the Pile [9] using SAELens [1] with a learning rate of $3e^{-4}$. During training, we ensure that the decoder remains normalized with $||W_{dec}||_2 = 1$ so s_n^{dec} calculations use the same scale for every latent. We train an SAE at each L0 from $10, 20, 30, \ldots, 250$. After L0=250, due to compute costs, we increase the spacing of the L0 of the SAEs up to L0=2500.

We plot s_n^{dec} vs L0 for Gemma-2-2b SAEs in Figure 3 for N=700 and N=10k (more plots in Appendix A.7). We see the same pattern for these SAEs as we saw in our toy model SAEs, namely that the metric is minimized at roughly the same L0, 200-250, despite the choice of N. As in the toy model SAEs case, using a higher N value results in a shallower slope leading up the the minimum s_n^{dec} L0.

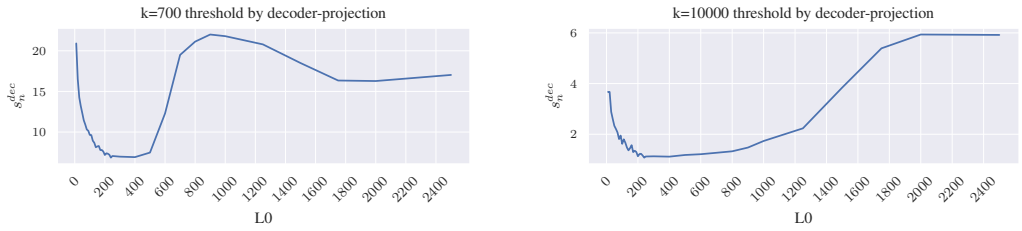


Figure 3: N[th] decoder projection vs SAE L0 for N=700 (left) and N=10k (right) on Gemma-2-2b layer 12 SAEs with 32k width. Here as well, we see that s_n^{dec} is minimized at the same point regardless of the choice of N, but the slope and shape of the s_n^{dec} curve varies depending on the choice of N. These plots imply that the correct L0 for these SAEs is around 200-250, much higher than typically used.

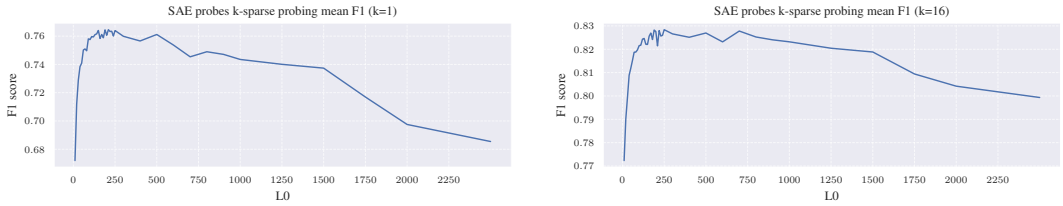


Figure 4: Sparse probing F1 score vs L0 for Gemma-2-2b layer 12 SAEs. Results are shown for k=1 (left) and k=16 (right). Both results are maximized at L0 around 200-250, matching our s_n^{dec} findings.

We test these SAEs using the sparse probing benchmark from Kantamneni et al. [11], consisting of over 100 sparse probing tasks. We use the normal setting from this benchmark with k=1 and k=16 sparse probes. Results are shown in Figure 4. The results closely match what the s_n^{dec} metric predicts, showing that L0 around 200-250 yields the highest results on the sparse probing benchmark.

5 Related work

Chanin et al. [6] explores feature hedging, showing SAEs mix correlated features into latents if the SAE is too narrow. We consider our work a version of feature hedging due to low L0. Till [18] shows SAEs may increase sparsity by inventing features. Chanin et al. [5] discuss the problem of feature absorption, where SAEs can improve their sparsity score by mixing hierarchical features together. Engels et al. [8] investigates SAE errors and finds that SAE error may be pathological and non-linear.

6 Discussion

While most practitioners of SAEs understand that having too high of a L0 is problematic (at some point the sparse autoencoder is not sparse), our work shows that having too low of L0 is perhaps even worse. Our work has several important implications for the field. First, the L0 used by most SAEs is lower than it ideally should be, as a cursory search of open source SAEs on Neuronpedia [13] shows L0 less than 100 and even less than 50 is very common even for SAEs trained on large models (see Appendix A.5). Our work further shows that the framing of a sparsity–reconstruction tradeoff, as commonly discussed by most SAE papers [7, 10, 15], is a misleading metric: when L0 is too low, an SAE with a correct dictionary achieves worse reconstruction than an incorrect SAE that mixes components of correlated features (see Appendix A.2).

While we expect that the correct L0 for a given SAE depends on the dataset used to train the SAE, we are also excited that our analysis may give us hints as to the true number of firing features in superposition in any given LLM activation.

The next step in our work is use our s_n^{dec} metric to located the correct L0 for an SAE during training. We discuss one such technique in Appendix A.4. We also hope to estimate how the correct L0 varies with SAE and model size and layer, and investigate more SAE architectures.

References

[1] Joseph Bloom, Curt Tigges, Anthony Duong, and David Chanin. Saelens. `https://github.com/jbloomAus/SAELens`, 2024.

[2] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, et al. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2, 2023.

[3] Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk sparse autoencoders. *arXiv preprint arXiv:2412.06410*, 2024.

[4] Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. Learning multi-level features with matryoshka sparse autoencoders. *arXiv preprint arXiv:2503.17547*, 2025.

[5] David Chanin, James Wilken-Smith, Tomáš Dulka, Hardik Bhatnagar, and Joseph Bloom. A is for absorption: Studying feature splitting and absorption in sparse autoencoders. *arXiv preprint arXiv:2409.14507*, 2024.

[6] David Chanin, Tomáš Dulka, and Adrià Garriga-Alonso. Feature hedging: Correlated features break narrow sparse autoencoders. *arXiv preprint arXiv:2505.11756*, 2025.

[7] Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=F76bwRSLeK`.

[8] Joshua Engels, Logan Riggs, and Max Tegmark. Decomposing the dark matter of sparse autoencoders. *arXiv preprint arXiv:2410.14670*, 2024.

[9] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[10] Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

[11] Subhash Kantamneni, Joshua Engels, Senthooran Rajamanoharan, Max Tegmark, and Neel Nanda. Are sparse autoencoders useful? a case study in sparse probing. *arXiv preprint arXiv:2502.16681*, 2025.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Johnny Lin. Neuronpedia: Interactive reference and tooling for analyzing neural networks, 2023. URL `https://www.neuronpedia.org`. Software available from neuronpedia.org.

[14] Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

[15] Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders. *arXiv preprint arXiv:2407.14435*, 2024.

[16] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu

Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Cogan, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. Gemma 2: Improving open language models at a practical size, 2024. URL `https://arxiv.org/abs/2408.00118`.

[17] Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, Alex Tamkin, Esin Durmus, Tristan Hume, Francesco Mosconi, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. `https://transformer-circuits.pub/2024/scaling-monosemanticity/`, May 2024. Accessed on May 21, 2024.

[18] Demian Till. Do sparse autoencoders find true features? *LessWrong*, 2024. URL `https://www.lesswrong.com/posts/QoR8noAB3Mp2KBA4B/do-sparse-autoencoders-find-true-features`.

A Technical Appendices and Supplementary Material

A.1 Toy model details

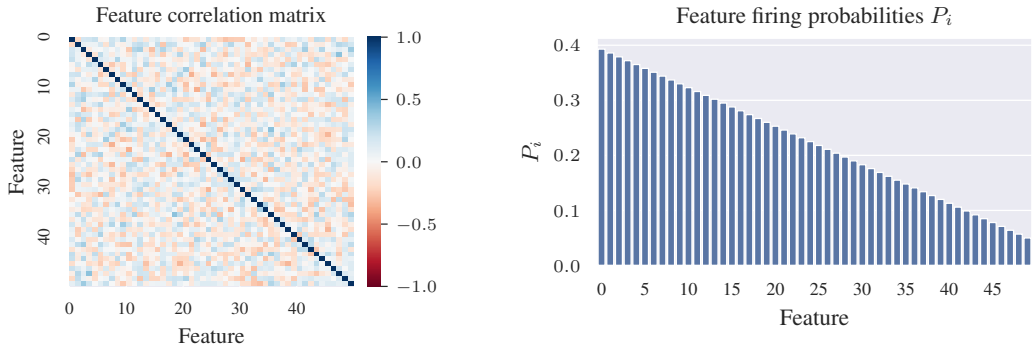


Figure 5: (left) random correlation matrix and (right) base feature firing probabilities for toy model.

A.2 Sparsity–reconstruction tradeoff in toy models

It is common in previous SAE work to discuss a sparsity–reconstruction tradeoff[7, 10, 15], where the assumption is that having better reconstruction at a given sparsity is inherently better, and is a sign the SAE is correct. As we discussed in Section 3.1, when the L0 of the SAE is lower than optimal, the SAE can find ways to "cheat" and get a better MSE score by mixing components of correlated features together. This results in an SAE where the latents are no longer monosemantic, and do not track ground-truth features.

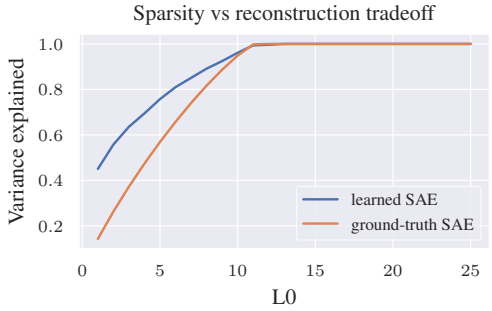
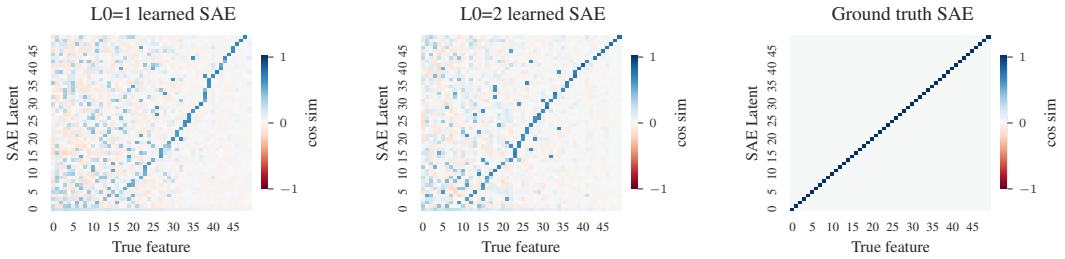


Figure 6: Sparsity (x) vs reconstruction (y) for learned SAEs and a ground-truth correct SAE at a given L0 in our toy model. When L0 is lower than the true L0 of the toy model, the learned SAE gets a higher reconstruction score than the ground-truth correct SAE, highlighting the problems with optimizing for reconstruction as a proxy for the quality of the SAE.

We explore the sparsity–reconstruction tradeoff in SAEs trained on our toy model at various L0s. Since we have the ground-truth correct features in our toy model, we also construct a ground-truth SAE that perfectly represents these true features. We manually set the L0 of the ground truth SAE while leaving the encoder and decoder fixed at the correct features. We plot the variance explained vs L0 in Figure 6 for the learned and ground-truth SAEs. We see that when the L0 of SAE is lower than the true L0 of the toy model, the ground-truth correct SAE actually scores worse on reconstruction than the incorrect learned SAE! This highlights the problems with optimizing reconstruction as a proxy for SAE quality.

We show the cosine similarity of the SAE decoder latents with the ground truth features for the SAEs learned with L0=1 and L0=2 compared with the ground-truth SAE in Figure 7c. Both these SAEs outperform the ground-truth SAE on variance explained by over 2x despite learning horribly polysemantic latents bearing little resemblance to the underlying true features of the model.



(a) Learned SAE L0 = 1 (b) Learned SAE L0 = 2 (c) Ground truth SAE

Figure 7: SAE decoder cosine similarity with true features for the learned SAEs with L0=1 and L0=2, compared with the ground-truth correct SAE. The learned SAEs score more than twice as a high as the ground truth SAE on variance explained when L0=1 and L0=2, despite having extremely corrupted, polysemantic latents.

A.3 Transitioning L0 during training

We explore the effect of transitioning the L0 of the SAE during training using the toy model from Section 3. This toy model has a true L0 of 11. We train BatchTopK SAEs with a final L0 of 11, but starting with L0 either too high or too low, and linearly transitioning to the correct L0 over the first 25k steps of training, leaving the SAE at the correct L0 for the final 5k steps of training. We use a starting L0 of 20 for the case where we start too high, and use a starting L0 of 2 for the case where we start too low. Results are shown in Figure 8.

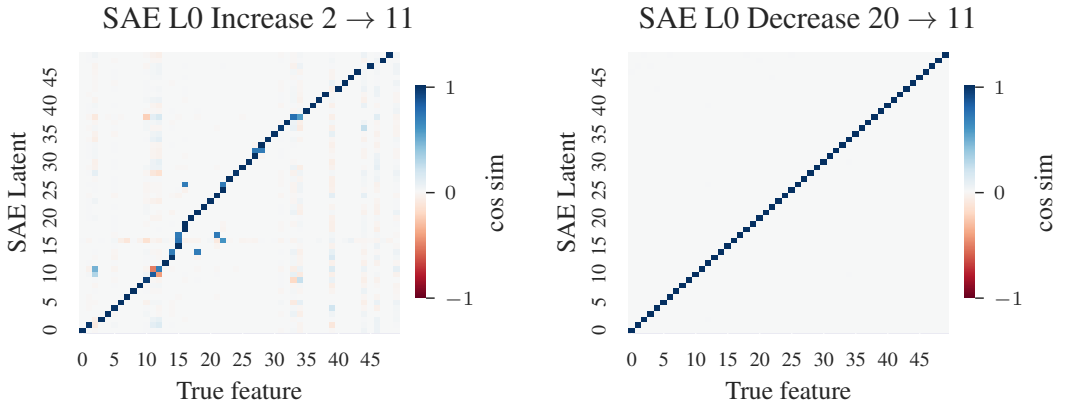


Figure 8: Transitioning L0 from too low (left) and too high (right) to the correct L0 during training. When the starting L0 is too high, the SAE still learns the correct features at the end of training. However, when L0 is too low, the SAE cannot recover fully and still learns many incorrect features at the end of training.

We see that decreasing the L0 of the SAE from a too high value to the correct value still results in the SAE learning correct features. However, when the SAE starts from a too low L0, the SAE cannot fully recover when the L0 is adjusted to the correct value later. It seems that the latents the SAE learns when L0 is too low is a local minimum that is difficult from the SAE to escape from even when the L0 is later corrected. This is likely because the latents learned when L0 is too low are optimized by gradient descent to achieve a higher MSE loss than is achievable by the correct latents under the same L0 constraint. However, when L0 is too high, there is no equivalent optimization pressure, and is thus less likely to be a local minimum.

A.4 Automatically finding the correct L0 during training

A natural next step of our finding that the correct L0 occurs when Nth decoder projection, s_n^{dec}, metric is minimized is to use this to find the correct L0 automatically during training. This is a meta-learning

task, as the L0 is a hyperparameter of the training process. We find there are several challenges to directly using s_n^{dec} as an optimization target:

- **Small gradients directly above correct L0** In our plots of s_n^{dec} from both toy models and Gemma-2-2b, we find that the metric is relatively flat in a region start at the correct L0 and extending to higher L0 values. We thus need a way to traverse this flat region and stop once the metric starts to increase again.
- **The impact of changing L0 is delayed** We find that it takes many steps after changing L0 for s_n^{dec} to also change, meaning it is easy to overshoot the target L0 or oscillate back and forth.
- **Dropping L0 too low can harm the SAE** As we saw in Appendix A.3, if the L0 is too low the SAE can permanently end up in poor local minima. We thus want to avoid dropping below the correct L0, even temporarily, to avoid permanently breaking the SAE. We therefore need to start with L0 too high and slowly decrease it until we find the correct L0.
- **Noise during training** We find that while s_n^{dec} shows clear trends after training for many steps, it can be noisy on each training sample. So our optimization needs to be robust to this noise.

Taking these requirements into account, we present an optimization procedure to find the L0 that minimizes s_n^{dec} automatically during training. We first estimate the gradient of s_n^{dec}, hereafter referred to as to as the metric, m, with respect to L0, $dm/dL0$. We first define an evaluation step t as a set number of training steps (we evaluate every 100 training steps). At t we change L0 by δ_{L_0}. At the next evaluation step, $t+1$, we evaluate m. We use a sliding average of s_n^{dec} over the past 10 training steps to calculate m to help account for noise. We the estimate $dm/dL0$ as:

$$\frac{dm}{dL0} = \frac{m_{t+1} - m_t}{\delta_{L0}}$$

Next, we add a small negative bias to this gradient estimate to encourage our estimate to push L0 lower even if the loss landscape is relatively flat. We use a bias magnitude $0 < b < 1$ that is multiplied by the magnitude of our gradient estimate, so that our biased estimate can never change the sign of the gradient estimate, but can gently nudge it to be more negative in flat, noisy regions of the loss landscape. We find $b = 0.1$ works well. Thus, our biased gradient estimate $dm_b/dL0$ is calculated as below:

$$\frac{dm_b}{dL0} = \frac{dm}{dL0} - b\left|\frac{dm}{dL0}\right|$$

We then provide this gradient to the Adam optimizer [12] with default settings, and allow it to change the L0 parameter.

We add the following optional modifications to this algorithm. First, we clip the gradient estimates $dm/dL0$ to be between -1 and 1. We also set a minimum and maximum δ_{L_0}. The minimum is added to avoid the denominator of our gradient estimate being near 0, and the maximum is chosen to keep the L0 from changing too quickly. In practice, we find a minimum δ_{L_0} between 0 and 1 seems to work well, and a maximum δ_{L_0} between 1 and 5 seems to work well.

We find that this optimization strategy works very well in toy models, but requires a lot of hyperparameter tuning to work in real LLMs. The starting L0, n for s_n^{dec}, b, learning rate for the Adam optimizer, and min and max δ_{L_0} values all have a big impact on how fast and how aggressively the optimization works. The slop of m around the correct L0 is shallow, so it is easy to overshoot. We also find that different values of n take more or less time to converge during training. We expect it is possible to further simplify and improve this process in future work.

A.5 L0 of open-source SAEs on Neuronpedia

We analyze common open-source SAEs as provided by Neuronpedia [13] and SAELens [1]. We include all SAEs cross-listed in both SAELens and Neuronpedia with an L0 reported in SAELens. We show the results as a histogram in Figure 9. Our analysis shows that for layer 12 of Gemma-2-2b,

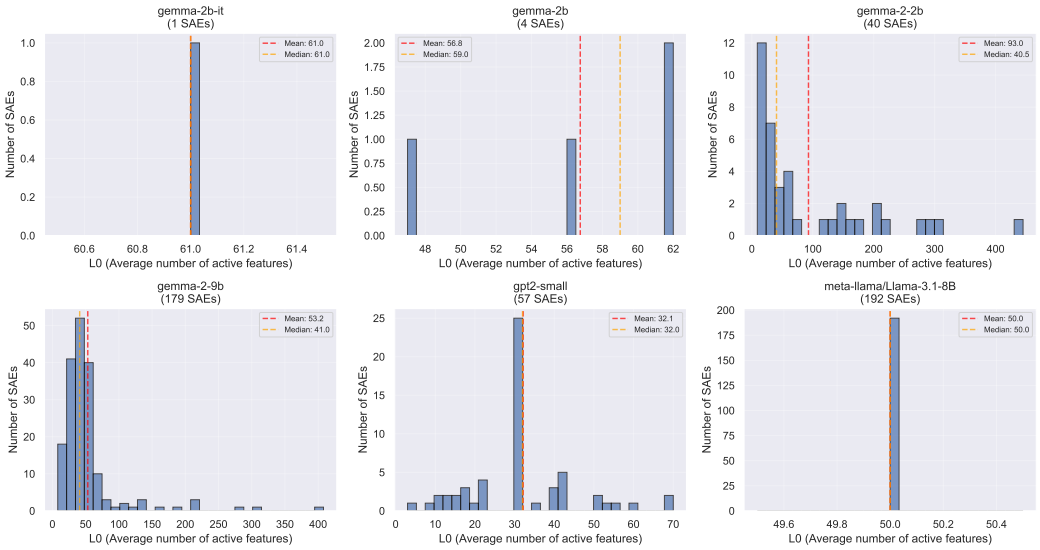


Figure 9: L0 of SAEs on Neuronpedia with known L0 listed in SAELens.

the correct L0 should be around 200-250. However, we find that most open-source SAEs have L0 below 100, much lower than our analysis expects to be ideal.

A.6 Limitations

We limit our investigated to Gemma-2-2b layer 12 due to the cost associated with training so many SAEs at different L0s. We investigate only BatchTopK SAEs, due to these SAEs allowing direct control over the L0 of the SAE without being moderated through a sparsity coefficient as in L1 SAEs [7, 2] or JumpReLU SAEs [15], but do not expect our findings to be meaningfully different for these other architectures.

A.7 Extended Gemma-2-2b Nth decoder projection plots

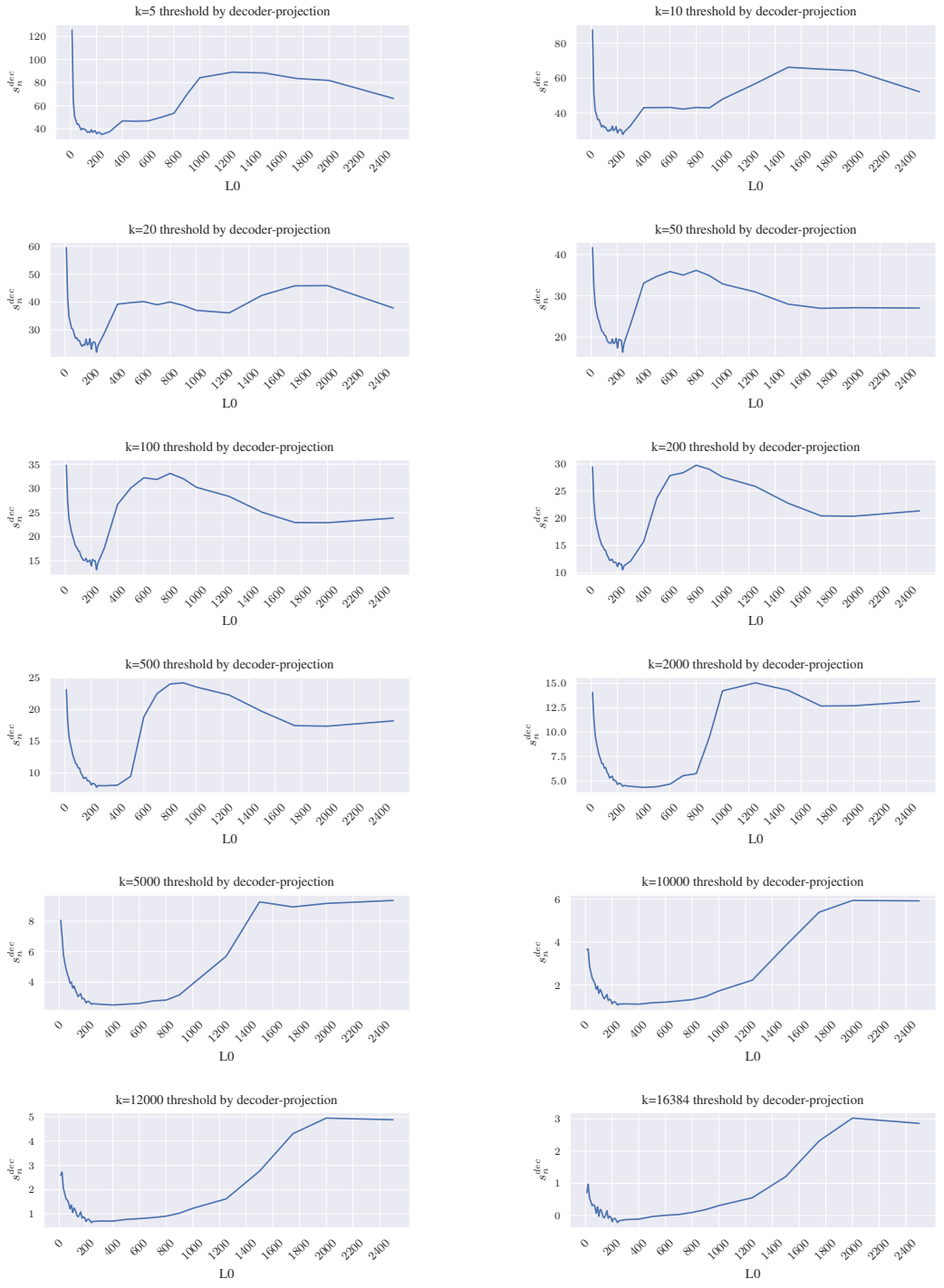


Figure 10: Extended Nth decoder projection plots. Gemma-2-2b, layer 12, 32k latents. Regardless of the choice of N, all plots are minimized around the same L0 range, 200-250.