Gemma Scope: Open Sparse Autoencoders Everywhere All At Once on Gemma 2

Tom Lieberum[1], **Senthooran Rajamanoharan**[1], **Arthur Conmy**[1], **Lewis Smith**[1], **Nicolas Sonnerat**[1], **Vikrant Varma**[1], **János Kramár**[1], **Anca Dragan**[1], **Rohin Shah**[1] **and Neel Nanda**[1]
[1]Google DeepMind

Sparse autoencoders (SAEs) are an unsupervised method for learning a sparse decomposition of a neural network's latent representations into seemingly interpretable features. Despite recent excitement about their potential, research applications outside of industry are limited by the high cost of training a comprehensive suite of SAEs. In this work, we introduce Gemma Scope, an open suite of JumpReLU SAEs trained on all layers and sub-layers of Gemma 2 2B and 9B and select layers of Gemma 2 27B base models. We primarily train SAEs on the Gemma 2 pre-trained models, but additionally release SAEs trained on instruction-tuned Gemma 2 9B for comparison. We evaluate the quality of each SAE on standard metrics and release these results. We hope that by releasing these SAE weights, we can help make more ambitious safety and interpretability research easier for the community. Weights and a tutorial can be found at https://huggingface.co/google/gemma-scope and an interactive demo can be found at https://neuronpedia.org/gemma-scope.

arXiv:2408.05147v2 [cs.LG] 19 Aug 2024

1. Introduction

There are several lines of evidence that suggest that a significant fraction of the internal activations of language models are sparse, linear combination of vectors, each corresponding to meaningful features (Elhage et al., 2022; Gurnee et al., 2023; Mikolov et al., 2013; Nanda et al., 2023a; Olah et al., 2020; Park et al., 2023). But by default, it is difficult to identify which vectors are meaningful, or which meaningful vectors are present. Sparse autoencoders are a promising unsupervised approach to do this, and have been shown to often find causally relevant, interpretable directions (Bricken et al., 2023; Cunningham et al., 2023; Gao et al., 2024; Marks et al., 2024; Templeton et al., 2024). If this approach succeeds it could help unlock many of the hoped for applications of interpretability (Hubinger, 2022; Nanda, 2022; Olah, 2021), such as detecting and fixing hallucinations, being able to reliably explain and debug unexpected model behaviour and preventing deception or manipulation from autonomous AI agents.

However, sparse autoencoders are still an immature technique, and there are many open problems to be resolved (Templeton et al., 2024) before these downstream uses can be unlocked – especially validating or red-teaming SAEs as an approach, learning how to measure their performance, learning how to train SAEs at scale efficiently and well, and exploring how SAEs can be productively applied to real-world tasks.

As a result, there is an urgent need for further research, both in industry and in the broader community. However, unlike previous interpretability techniques like steering vectors (Li et al., 2023; Turner et al., 2024) or probing (Belinkov, 2022), sparse autoencoders can be highly expensive and difficult to train, limiting the ambition of interpretability research. Though there has been a lot of excellent work with sparse autoencoders on smaller models (Bricken et al., 2023; Cunningham et al., 2023; Dunefsky et al., 2024; Marks et al., 2024), the works that use SAEs on more modern models have normally focused on residual stream SAEs at a single layer (Engels et al., 2024; Gao et al., 2024; Templeton et al., 2024). In addition, many of these (Gao et al., 2024; Templeton et al., 2024) have been trained on proprietary models which makes it more challenging for the community at large to build on this work.

To address this we have trained and released the weights of Gemma Scope: a comprehensive, open suite of JumpReLU SAEs (Rajamanoharan

Corresponding author(s): tlieberum@google.com
© 2024 Google DeepMind. All rights reserved

et al., 2024b) on every layer and sublayer of Gemma 2 9B and 2B (Gemma Team, 2024b),[1] as well select layers of the larger 27B model in this series. We release these weights under a permissive CC-BY-4.0 license[2] on HuggingFace to enable and accelerate research by other members of the research community.

Gemma Scope was a significant engineering challenge to train. It contains more than 400 sparse autoencoders in the main release[3], with more than 30 million learned features in total (though many features likely overlap), trained on 4-16B tokens of text each. We used over 20% of the training compute of GPT-3 (Brown et al., 2020), saved about 20 Pebibytes (PiB) of activations to disk, and produced hundreds of billions of sparse autoencoder parameters in total. This was made more challenging by our decision to make a *comprehensive* suite of SAEs, on every layer and sublayer. We believe that a comprehensive suite is essential for enabling more ambitious applications of interpretability, such as circuit analysis (Conmy et al., 2023; Hanna et al., 2023; Wang et al., 2022), essentially scaling up Marks et al. (2024) to larger models, which may be necessary to answer mysteries about larger models like what happens during chain of thought or in-context learning.

In Section 2 we provide background on SAEs in general and JumpReLU SAEs in particular. Section 3 contains details of our training procedure, hyperparameters and computational infrastructure. We run extensive evaluations on the trained SAEs in Section 4 and a list of open problems that Gemma Scope could help tackle in Section 5.

[1]We also release one suite of transcoders (Dunefsky et al. (2024); Appendix B), a 'feature-splitting' suite of SAEs with multiple widths trained on the same site (Section 4.3), and some SAEs trained on the Gemma 2 9B IT model (Kissane et al. (2024b); Section 4.5).

[2]Note that the Gemma 2 models are released under a different, custom license.

[3]For each model, layer and site we in fact release multiple SAEs with differing levels of sparsity; taking this into account, we release the weights of over 2,000 SAEs in total.

2. Preliminaries

2.1. Sparse autoencoders

Given activations $\mathbf{x} \in \mathbb{R}^n$ from a language model, a sparse autoencoder (SAE) decomposes and reconstructs the activations using a pair of encoder and decoder functions $(\mathbf{f}, \hat{\mathbf{x}})$ defined by:

$$\mathbf{f}(\mathbf{x}) := \sigma\left(\mathbf{W}_{\text{enc}}\mathbf{x} + \mathbf{b}_{\text{enc}}\right), \tag{1}$$

$$\hat{\mathbf{x}}(\mathbf{f}) := \mathbf{W}_{\text{dec}}\mathbf{f} + \mathbf{b}_{\text{dec}}. \tag{2}$$

These functions are trained to map $\hat{\mathbf{x}}(\mathbf{f}(\mathbf{x}))$ back to $\mathbf{x}$, making them an autoencoder. Thus, $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^M$ is a set of linear weights that specify how to combine the $M \gg n$ columns of $\mathbf{W}_{\text{dec}}$ to reproduce $\mathbf{x}$. The columns of $\mathbf{W}_{\text{dec}}$, which we denote by $\mathbf{d}_i$ for $i = 1 \ldots M$, represent the dictionary of directions into which the SAE decomposes $\mathbf{x}$. We will refer to to these learned directions as *latents* to disambiguate between learnt 'features' and the conceptual features which are hypothesized to comprise the language model's representation vectors.[4]

The decomposition $\mathbf{f}(\mathbf{x})$ is made *non-negative* and *sparse* through the choice of activation function σ and appropriate regularization, such that $\mathbf{f}(\mathbf{x})$ typically has much fewer than n non-zero entries. Initial work (Bricken et al., 2023; Cunningham et al., 2023) used a ReLU activation function to enforce non-negativity, and an L1 penalty on the decomposition $\mathbf{f}(\mathbf{x})$ to encourage sparsity. TopK SAEs (Gao et al., 2024) enforce sparsity by zeroing all but the top K entries of $\mathbf{f}(\mathbf{x})$, whereas the JumpReLU SAEs (Rajamanoharan et al., 2024b) enforce sparsity by zeroing out all entries of $\mathbf{f}(\mathbf{x})$ below a positive threshold. Both TopK and JumpReLU SAEs allow for greater separation between the tasks of determining which latents are active, and estimating their magnitudes.

2.2. JumpReLU SAEs

In this work we focus on JumpReLU SAEs as they have been shown to be a slight Pareto improve-

[4]This is different terminology from earlier work (Bricken et al., 2023; Rajamanoharan et al., 2024a,b), where feature is normally used interchangeably for both SAE latents and the language models features

ment over other approaches, and allow for a variable number of active latents at different tokens (unlike TopK SAEs).

JumpReLU activation The JumpReLU activation is a shifted Heaviside step function as a gating mechanism together with a conventional ReLU:

$$\sigma(\mathbf{z}) = \text{JumpReLU}_{\boldsymbol{\theta}}(\mathbf{z}) := \mathbf{z} \odot H(\mathbf{z} - \boldsymbol{\theta}). \quad (3)$$

Here, $\boldsymbol{\theta} > 0$ is the JumpReLU's vector-valued learnable threshold parameter, $\odot$ denotes elementwise multiplication, and H is the Heaviside step function, which is 1 if its input is positive and 0 otherwise. Intuitively, the JumpReLU leaves the pre-activations unchanged above the threshold, but sets them to zero below the threshold, with a different learned threshold per latent.

Loss function As loss function we use a squared error reconstruction loss, and directly regularize the number of active (non-zero) latents using the L0 penalty:

$$\mathcal{L} := \|\mathbf{x} - \hat{\mathbf{x}}(\mathbf{f}(\mathbf{x}))\|_2^2 + \lambda \|\mathbf{f}(\mathbf{x})\|_0, \quad (4)$$

where λ is the sparsity penalty coefficient. Since the L0 penalty and JumpReLU activation function are piecewise constant with respect to threshold parameters $\boldsymbol{\theta}$, we use straight-through estimators (STEs) to train $\boldsymbol{\theta}$, using the approach described in Rajamanoharan et al. (2024b). This introduces an additional hyperparameter, the kernel density estimator bandwidth ε, which controls the quality of the gradient estimates used to train the threshold parameters $\boldsymbol{\theta}$.[5]

3. Training details

3.1. Data

We train SAEs on the activations of Gemma 2 models generated using text data from the same distribution as the pretraining text data for Gemma 1 (Gemma Team, 2024a), except for the one suite of SAEs trained on the instruction-tuned (IT) model (Section 4.5).

For a given sequence we only collect activations from tokens which are neither `BOS`, `EOS`, nor padding. After activations have been generated, they are shuffled in buckets of about 10^6 activations. We speculate that a perfect shuffle would not significantly improve results, but this was not systematically checked. We would welcome further investigation into this topic in future work.

During training, activation vectors are normalized by a fixed scalar to have unit mean squared norm.[6] This allows more reliable transfer of hyperparameters (in particular the sparsity coefficient λ and bandwidth ε) between layers and sites, as the raw activation norms can vary over multiple orders of magnitude, changing the scale of the reconstruction loss in Eq. (4). Once training is complete, we rescale the trained SAE parameters so that no input normalization is required for inference (see Appendix A for details).

As shown in Table 1, SAEs with 16.4K latents are trained for 4B tokens, while 1M-width SAEs are trained for 16B tokens. All other SAEs are trained for 8B tokens.

Location We train SAEs on three locations per layer, as indicated by Fig. 1. We train on the attention head outputs before the final linear transformation W_O and RMSNorm has been applied (Kissane et al., 2024a), on the MLP outputs after the RMSNorm has been applied and on the post MLP residual stream. For the attention output SAEs, we concatenate the outputs of the individual attention heads and learn a joint SAE for the full set of heads. We zero-index the layers, so layer 0 refers to the first transformer block after the embedding layer. In Appendix B we define *transcoders* (Dunefsky et al., 2024) and train one suite of these.

[5] A large value of ε results in biased but low variance estimates, leading to SAEs with good sparsity but sub-optimal fidelity, whereas a low value of ε results in high variance estimates that cause the threshold to fail to train at all, resulting in SAEs that fail to be sparse. We find through hyperparameter sweeps across multiple layers and sites that $\varepsilon = 0.001$ provides a good trade-off (when SAE inputs are normalized to have a unit mean squared norm) and use this to train the SAEs released as part of Gemma Scope.

[6] This is similar in spirit to Conerly et al. (2024), who normalize the dataset to have mean norm of $\sqrt{d_{\text{model}}}$.