BatchTopK Sparse Autoencoders

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Sparse autoencoders (SAEs) have emerged as a powerful tool for interpreting language model activations by decomposing them into sparse, interpretable features. A popular approach is the TopK SAE, that uses a fixed number of the most active latents per sample to reconstruct the model activations. We introduce BatchTopK SAEs, a training method that improves upon TopK SAEs by relaxing the top-k constraint to the batch-level, allowing for a variable number of latents to be active per sample. BatchTopK SAEs consistently outperform TopK SAEs at reconstructing activations from GPT-2 Small and Gemma 2 2B. BatchTopK SAEs achieve comparable reconstruction performance to the state-of-the-art JumpReLU SAE, but have the advantage that the average number of latents can be directly specified, rather than approximately tuned through a costly hyperparameter sweep. We provide code for training and evaluating these BatchTopK SAEs at [redacted].

1 Introduction

Sparse autoencoders (SAEs) have been proven effective for finding interpretable directions in the activation space of language models [1, 2, 9, 6]. SAEs find approximate, sparse, linear decompositions of language model activations by learning a dictionary of interpretable latents from which the activations are reconstructed.

The objective used in training SAEs [1] has both a sparsity and a reconstruction term. These are naturally in tension as, for an optimal dictionary of a given size, improving the reconstruction performance requires decreasing sparsity and vice versa. Recently, new architectures have been proposed to address this issue, and achieve better reconstruction performance at a given sparsity level, such as Gated SAEs [6], JumpReLU SAEs [7], and TopK SAEs [3].

In this paper, we introduce BatchTopK SAEs, a novel variant that extends TopK SAEs by relaxing the top-k constraint to a batch-level constraint. This modification allows the SAE to represent each sample with a variable number of latents, rather than assuming that all model activations consist of the same number of units of analysis. By selecting the top activations across the entire batch, BatchTopK SAEs enable more flexible and efficient use of the latent dictionary, leading to improved reconstruction without sacrificing average sparsity. During inference we remove the batch dependency by estimating a single global threshold parameter.

Through experiments on the residual streams of GPT-2 Small [5] and Gemma 2 2B [8], we show that BatchTopK SAEs consistently outperform both TopK and JumpReLU SAEs in terms of reconstruction performance across various dictionary sizes and sparsity levels, although JumpReLU SAEs have less downstream CE degradation in large models with a high number of active latents. Moreover, unlike JumpReLU SAEs, BatchTopK SAEs allow direct specification of the sparsity level without the need for tuning additional hyperparameters.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

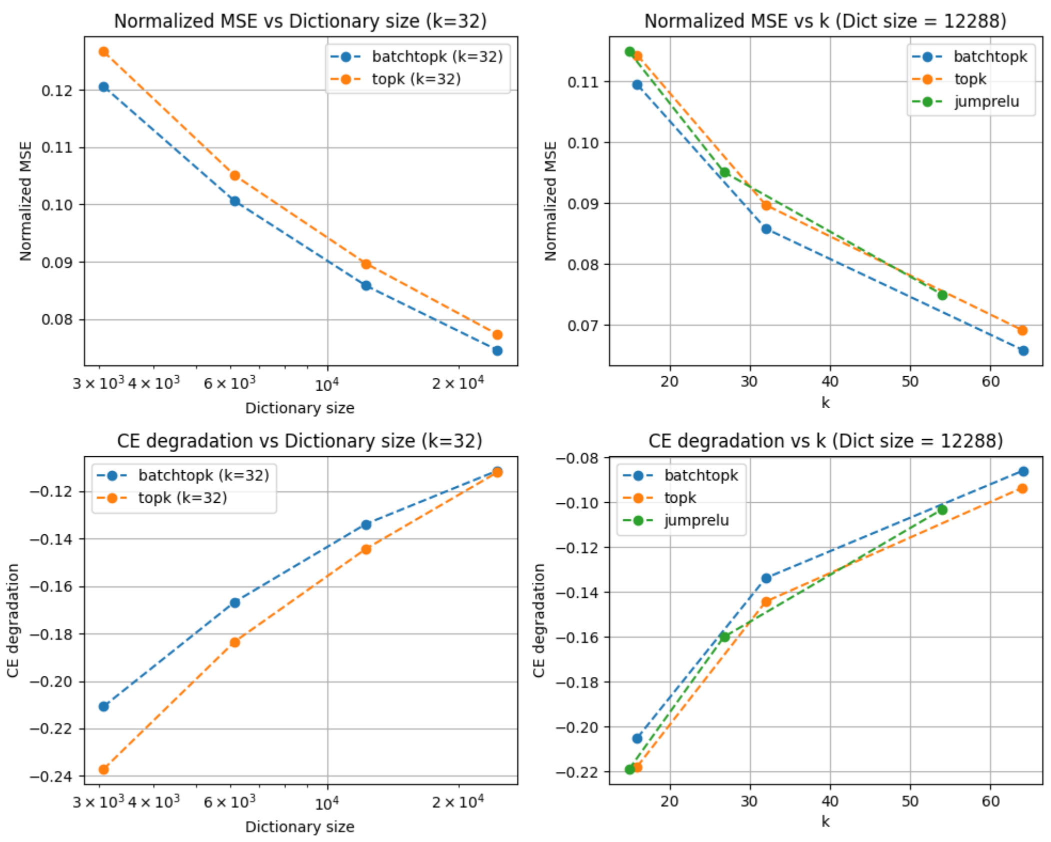

Figure 1: On GPT-2 Small activations, BatchTopK largely achieves a better NMSE and CE compared to standard TopK across different dictionary sizes, for a fixed number of active latents of 32 (Left). JumpReLU SAEs are omitted from this comparison as their L0 cannot be fixed to a value. For fixed dictionary size (12288) and different levels of k, BatchTopK outperforms TopK and JumpReLU SAES, both in terms of NMSE and CE (Right).

2 Background: Sparse Autoencoder Architectures

Sparse autoencoders aim to learn efficient representation of data by reconstruction inputs while enforcing sparsity in the latent space. In the context of large language models, SAEs decompose model activations $\mathbf{x} \in \mathbb{R}^n$ into sparse linear combinations of learned directions, which are often interpretable and monosemantic.

An SAE consists of an encoder and a decoder:

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

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

where $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^m$ is the sparse latent representation, and $\hat{\mathbf{x}}(\mathbf{f}) \in \mathbb{R}^n$ is the reconstructed input. The activation function σ enforces non-negativity and sparsity in $\mathbf{f}(\mathbf{x})$.

SAEs are trained on the activations of a language model at a particular site, such as the residual stream, on a large text corpus, using a loss function of the form

$$\mathcal{L}(\mathbf{x}) := \underbrace{\|\mathbf{x} - \hat{\mathbf{x}}(\mathbf{f}(\mathbf{x}))\|_2^2}_{\mathcal{L}_{\text{reconstruct}}} + \underbrace{\lambda \mathcal{S}(\mathbf{f}(\mathbf{x}))}_{\mathcal{L}_{\text{sparsity}}} + \alpha \mathcal{L}_{\text{aux}} \tag{3}$$

where $\mathcal{S}$ is a function of the latent coefficients that penalizes non-sparse decompositions, and λ is a sparsity coefficient, where higher values of λ encourage sparsity at the cost of higher reconstruction error. Some architectures also require the use of an auxiliary loss $\mathcal{L}_{\text{aux}}$, for example to recycle inactive latents in TopK SAEs.

ReLU SAEs [1] use the L1-norm $S(\boldsymbol{f}) := ||\boldsymbol{f}||_1$ as an approximation to the L0-norm for the sparsity penalty. This provides a gradient for training unlike the L0-norm, but suppresses latent activations harming reconstruction performance [6]. Furthermore, the L1 penalty can be arbitrarily reduced through reparameterization by scaling the decoder parameters, which is resolved in [1] by constraining the decoder directions to the unit norm. Resolving this tension between activation sparsity and value is the motivation behind the newer architecture variants.

TopK SAEs [3, 4] enforce sparsity by retaining only the top k activations per sample. The encoder is defined as:

$$\boldsymbol{f}(\boldsymbol{x}) := \text{TopK}(\mathbf{W}_{\text{enc}}\mathbf{x} + \mathbf{b}_{\text{enc}}) \tag{4}$$

where TopK zeroes out all but the k largest activations in each sample. This approach eliminates the need for an explicit sparsity penalty but imposes a rigid constraint on the number of active latents per sample. An auxiliary loss $\mathcal{L}_{aux} = ||e - \hat{e}||^2$ is used to avoid dead latents, where $\hat{e} = W_{dec}z$ is the reconstruction using only the top-k_{aux} dead latents (usually 512), this loss is scaled by a small coefficient α (usually 1/32).

JumpReLU SAEs [7] replace the standard ReLU activation function with the JumpReLU activation, defined as

$$\text{JumpReLU}_\theta(z) := zH(z - \theta) \tag{5}$$

where H is the Heaviside step function, and θ is a learned parameter for each SAE latent, below which the activation is set to zero. JumpReLU SAEs are trained using a loss function that combines L2 reconstruction error with an L0 sparsity penalty, using straight-through estimators to train despite the discontinuous activation function. A major drawback of the sparsity penalty used in JumpReLU SAEs compared to (Batch)TopK SAEs is that it is not possible to set an explicit sparsity and targeting a specific sparsity involves costly hyperparameter tuning. While evaluating JumpReLU SAEs, [7] chose the SAEs from their sweep that were closest to the desired sparsity level, however this resulted in SAEs with significantly different sparsity levels being directly compared. JumpReLU SAEs use no auxiliary loss function.

3 BatchTopK Sparse Autoencoders

We introduce **BatchTopK SAEs** as an improvement over standard TopK SAEs. In BatchTopK, we replace the sample-level TopK operation with a batch-level BatchTopK function. Instead of selecting the top k activations for each individual sample, we select the top $n \times k$ activations across the entire batch of n samples, setting all other activations to zero. This allows for a more flexible allocation of active latents, where some samples may use more than k latents while others use fewer, potentially leading to better reconstructions of the activations that are more faithful to the model.

The training objective for BatchTopK SAEs is defined as:

$$\mathcal{L}(\mathbf{X}) = ||\mathbf{X} - \text{BatchTopK}(\mathbf{W}_{enc}\mathbf{X} + \mathbf{b}_{enc})\mathbf{W}_{dec} + \mathbf{b}_{dec})||_2^2 + \alpha\mathcal{L}_{\text{aux}} \tag{6}$$

Here, $\mathbf{X}$ is the input data batch; $\mathbf{W}_{\text{enc}}$ and $\mathbf{b}_{\text{enc}}$ are the encoder weights and biases, respectively; $\mathbf{W}_{\text{dec}}$ and $\mathbf{b}_{\text{dec}}$ are the decoder weights and biases. The function BatchTopK selects the top $n \times k$ activations across the batch to enforce sparsity. The term $\mathcal{L}_{\text{aux}}$ is an auxiliary loss scaled by the coefficient α, used to prevent dead latents, and is the same as in TopK SAEs.

BatchTopK introduces a dependency between the activations for the samples in a batch. We alleviate this during inference by using a threshold θ that is estimated as the average of the minimum positive activation values across a number of batches:

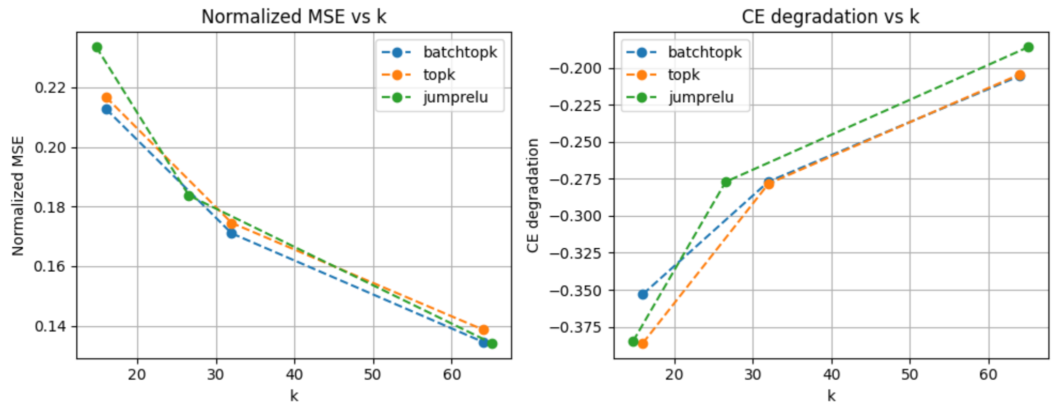

Figure 2: On Gemma 2 2B activations, BatchTopK outperforms TopK SAEs across different values of k. Although BatchTopK has a better reconstruction performance (left), it only outperforms JumpReLU in terms of downstream CE degradation in the setting where k=16 (right).

$$\theta = \mathbb{E}_{\mathbf{X}}[\min\{z_{i,j}(\mathbf{X}) | z_{i,j}(\mathbf{X}) > 0\}] \tag{7}$$

where $z_{i,j}(\mathbf{X})$ is the jth latent activation of the ith sample in a batch $\mathbf{X}$. With this threshold, we use the JumpReLU activation function during inference instead of the BatchTopK activation function, zeroing out all activations under the threshold θ.

4 Experiments

We evaluate the performance of BatchTopK on the activations of two LLMs: GPT-2 Small (residual stream layer 8) and Gemma 2 2B (residual stream layer 12). We use a range of dictionary sizes and values for k, and compare our results to TopK and JumpReLU SAEs in terms of normalized mean squared error (NMSE) and cross-entropy degradation. For the experimental details, see Appendix A.2.

We find that for a fixed number of active latents (L0=32) the BatchTopK SAE has a lower normalized MSE and less cross-entropy degradation than TopK SAEs on both GPT-2 activations (Figure 1) and Gemma 2 2B (Figure 2). Furthermore, we find that for a fixed dictionary size (12288) BatchTopK outperforms TopK for different values of k on both models.

In addition, BatchTopK outperforms JumpReLU SAEs on both measures on GPT-2 Small model activations at all levels of sparsity. On Gemma 2 2B model activations the results are more mixed: although BatchTopK achieves better reconstruction than JumpReLU for all values of k, BatchTopK only outperforms JumpReLU in terms of CE degradation in the lowest sparsity setting (k=16).

Figure 3 shows a histogram of the number of active BatchTopK SAE latents per sample, corroborating our hypothesis that the fixed TopK in [3] is too restrictive and that samples may require a variable number of active dictionary latents for reconstruction.

5 Conclusion

In this work, we have introduced BatchTopK Sparse Autoencoders, a novel extension of TopK SAEs that relaxes the fixed per-sample sparsity constraint to a batch-level constraint. By selecting the top activations across the entire batch rather than enforcing a strict limit per sample, BatchTopK allows for a variable number of active latents per sample. This flexibility enables the model to allocate more latents to complex samples and fewer to simpler ones, improving overall reconstruction performance without sacrificing average sparsity. We evaluated BatchTopK SAEs using the standard metrics of reconstruction loss and sparsity, and due to the architectural similarity to TopK SAEs we anticipate the interpretability of latents to be similar but did not evaluate human interpretability.

References

[1] 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.

[2] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

[3] 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, 2024.

[4] Alireza Makhzani and Brendan Frey. k-sparse autoencoders, 2014.

[5] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[6] Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, János Kramár, Rohin Shah, and Neel Nanda. Improving dictionary learning with gated sparse autoencoders, 2024.

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

[8] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[9] Adly Templeton, Tom Conerly, and et al. *Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet*. Anthropic, 2024.

A Supplemental Material

A.1 Experimental Details

In this appendix, we provide details about the datasets used, model configurations, and hyperparameters for our experiments.

We trained our sparse autoencoders (SAEs) on the OpenWebText dataset[1], which was processed into sequences of a maximum of 128 tokens for input into the language models.

All models were trained using the Adam optimizer with a learning rate of 3×10^{-4}, $\beta_1 = 0.9$, and $\beta_2 = 0.99$. The batch size was 4096, and training continued until a total of 1×10^9 tokens were processed.

We experimented with dictionary sizes of 3072, 6144, 12288, and 24576 for the GPT-2 Small model, and used a dictionary size of 16384 for the experiment on Gemma 2 2B. In both experiments, we varied the number of active latents k among 16, 32, and 64.

For the JumpReLU SAEs, we varied the sparsity coefficient such that the resulting sparsity would match the active latents k of the BatchTopK and TopK models. The sparsity penalties in the experiments on GPT-2 Small were 0.004, 0.0018, and 0.0008. For the Gemma 2 2B model we used sparsity penalties of 0.02, 0.005, and 0.001. In both experiments, we set the bandwidth parameter to 0.001.

[1] `https://huggingface.co/datasets/openwebtext`

A.2 Active latents per sample

To confirm that BatchTopK SAEs make use of the enabled flexibility to activate a variable number of latents per sample, we plot the distribution of the number of active latents per sample in Figure 3. We observe that BatchTopK indeed uses a wide range of active latents, activating only a single latent on some samples and activating more than 80 on others. The peak on the left of the distribution are model activations on the <BOS>-token. This serves as an example of the advantage of BatchTopK: when the model activations do not contain much information, BatchTopK does not activate many latents, whereas TopK would use the same number of latents regardless of the input.

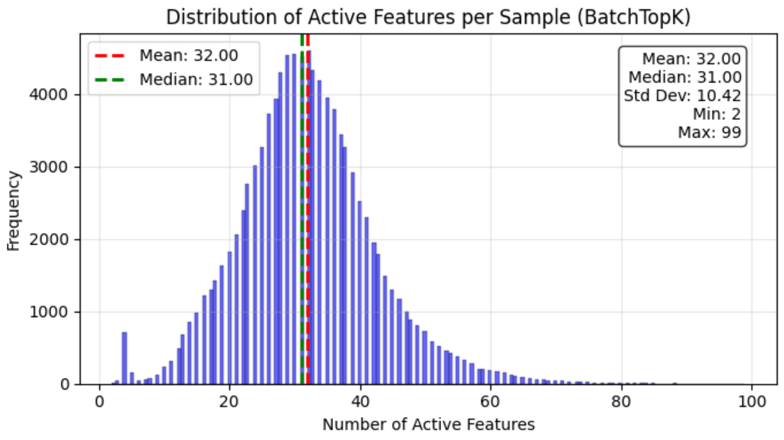

Figure 3: Distribution of the number of active latents per sample for a BatchTopK model. The peak on the left likely corresponds to BOS tokens, demonstrating BatchTopK's adaptive sparsity.