Efficient Matryoshka loss calculation

The vanilla sparse autoencoder latents are calculated by

$$f(\mathbf{x})_i = A\mathbf{x} + \mathbf{b}$$

and the prediction of a vanilla sparse autoencoder with N latents is given by

$$\hat{\mathbf{x}} = \mathbf{c} + \sum_{i=0}^{N-1} f(\mathbf{x})_i \mathbf{d}_i$$

The sparsity loss I use for both 'Vanilla' and 'Matryoshka' is a bit different from L1, but I believe it is comparable.[1] The SAE loss I use is

$$\mathcal{L}(x) = \mathrm{MSE}(\mathbf{x}, \hat{\mathbf{x}}) + \lambda \sum_{i=0}^{N-1} \log\Big(|f(\mathbf{x})_i| \cdot \|\mathbf{d}_i\|_2 + \epsilon\Big)$$

In practice I use $\epsilon = 0.1$.

Now we'll build up to the Matryoshka loss. The idea is to train on a mixture on losses, each of which is the vanilla SAE loss on a prefix of the Matryoshka SAE latents.

Define $\hat{\mathbf{x}}_p$ for $0 < p \leq N$ by

$$\hat{\mathbf{x}}_p = \mathbf{c} + \sum_{i=0}^{p-1} f(\mathbf{x})_i \mathbf{d}_i$$

Then the SAE prefix loss, $\mathcal{L}_p$, is given by

$$\mathcal{L}_p(\mathbf{x}) = \mathrm{MSE}(\mathbf{x}, \hat{\mathbf{x}}_p) + \lambda \sum_{i=0}^{p-1} \log\Big(|f(\mathbf{x})_i| \cdot \|d_i\|_2 + \epsilon\Big)$$

For every batch, we sample P prefixes from a truncated Pareto distribution along with the full-prefix to get the vector of prefixes p_j. See [here] for how prefixes are sampled. With 1000 latents and 3 prefixes, we might sample $p_j = [121, 562, 1000]$ as our prefixes. Assume that the prefix vector p_j is sorted from shortest prefix to longest. Then the Matryoshka loss is defined

$$\mathcal{L}_{\text{🪆}}(x) = \sum_{j=0}^{P-1} \mathcal{L}_{p_j}(x),$$
$$p_j \sim \mathrm{Pareto}[N].$$

A naive calculation of the Matryoshka loss would involve a different SAE forward pass for each prefix. I avoid this with a faster implementation.

In order to efficiently calculate $\hat{\mathbf{x}}_{p_j}$, recall that

$$\hat{\mathbf{x}}_{p_j} = \sum_{k=0}^{p_{j\text{-}1}-1} f(x)_k \mathbf{d}_k$$

Let us label the difference between two adjacent SAE-prefix outputs by δ_j.

$$\delta_j \stackrel{\text{def}}{=} \begin{cases} \hat{\mathbf{x}}_{p_j} - \hat{\mathbf{x}}_{p_{j\text{-}1}}, & j > 0 \\ \hat{\mathbf{x}}_{p_0}, & j = 0 \end{cases}$$

Or equivalently,

[1] Compare to square-root sparsity penalty[3] and tanh[1][2]. I focused on SAEs with log sparsity penalties as I found the features slightly more interpretable and it was a Pareto improvement on L0/FVU vs L1 and possibly square root. L1 penalty SAEs seemed to exhibit similar feature splitting as log penalty. I don't currently believe this affects the generality of my results, but it seems plausible that log-sparsity SAEs would exhibit more extreme feature absorption.

$$
\delta_j = \begin{cases} \displaystyle\sum_{k=p_{j\text{-}1}}^{p_j-1} f(x)_k \mathbf{d}_k, & j > 0 \\ \displaystyle\sum_{k=0}^{p_0-1} f(x)_k \mathbf{d}_k, & j = 0 \end{cases}
$$

Note that δ_j is cheaper to compute than $\hat{\mathbf{x}}_{p_j}$ for $j > 0$ because δ_j only uses $p_j - p_{j\text{-}1}$ latents while $\hat{\mathbf{x}}_{p_j}$ uses p_j latents. The efficiency trick here is to calculate the δ_j and then take a cumulative sum to get the $\hat{\mathbf{x}}_{p_j}$.

A very similar procedure can make the Matryoshka sparsity loss calculation more efficient.

Define

$$
\Delta_j = \begin{cases} \displaystyle\sum_{k=p_{j\text{-}1}}^{p_j-1} \log\Big(|f(x)_k| \cdot \|\mathbf{d}_k\|_2 + \epsilon\Big), & j > 0 \\ \displaystyle\sum_{k=0}^{p_j-1} \log\Big(|f(x)_k| \cdot \|\mathbf{d}_k\|_2 + \epsilon\Big), & j = 0 \end{cases}
$$

Then

$$
\begin{aligned}
\mathcal{L}_{\text{🪆}}(x) &= \sum_{j=0}^{P-1} \mathcal{L}_{p_j}(x), \\
&= \sum_{j=0}^{P-1} \left(\mathrm{MSE}(\mathbf{x}, \hat{\mathbf{x}}_{p_j}) + \lambda \sum_{i=0}^{p_j-1} \log\Big(|f(\mathbf{x})_i| \cdot \|\mathbf{d}_i\|_2 + \epsilon\Big) \right), \\
&= \sum_{j=0}^{P-1} \left(\mathrm{MSE}\left(\mathbf{x}, \sum_{k=0}^{j-1} \delta_k\right) + \lambda \sum_{k=0}^{j-1} \Delta_k \right)
\end{aligned}
$$

The algorithm for computing the Matryoshka loss is

- Calculate $f(x)_i$.
- Calculate δ_i and Δ_i using $f(x)_i$ and $\mathbf{d}_i$.
- Take a cumulative sum[https://pytorch.org/docs/stable/generated/torch.cumsum.html] of the δ_j to get each $\hat{\mathbf{x}}_j$.
- Calculate the MSE using the $\hat{\mathbf{x}}_j$
- Take a cumulative sum of the Δ_j to get the sparsity loss term in each $\mathcal{L}_p$.
- Add all sparsity and MSE losses to get the final Matryoshka loss.

Code for the above along with the truncated Pareto sampling can be found in [github link].

References

[1] Adam Jermyn et al. *Dictionary Learning Update*. 2024. URL: `https://transformer-circuits.pub/2024/feb-update/index.html#dict-learning-tanh` (visited on 11/13/2024).

[2] Jack Lindsey, Hoagy Cunningham, and Tom Conerly. *Interpretability Evals for Dictionary Learning*. Ed. by Adly Templeton. 2024. URL: `https://transformer-circuits.pub/2024/august-update/index.html#interp-evals` (visited on 11/13/2024).

[3] Logan Riggs and Jannik Brinkmann. *Improving SAE's by Sqrt()-ing L1 & Removing Lowest Activating Features*. Mar. 2024. URL: `https://www.lesswrong.com/posts/YiGs8qJ8aNBgwt2YN/improving-sae-s-by-sqrt-ing-l1-and-removing-lowest` (visited on 11/13/2024).