Analyzing (In)Abilities of SAEs via Formal Languages

Abhinav S Menon
IIIT Hyderabad
abhinav.m@research.iiit.ac.in

Manish Shrivastava
IIIT Hyderabad
m.shrivastava@iiit.ac.in

David Krueger
MILA
david.scott.krueger@gmail.com

Ekdeep Singh Lubana
CBS, Harvard University
ekdeeplubana@fas.harvard.edu

Abstract

Sparse autoencoders (SAEs) have been central to the effort of finding interpretable and disentangled directions of representation spaces in neural networks, in both image and text domains. While the efficacy and pitfalls of autoencoders in the vision domain are well-studied, there is a lack of corresponding results, both qualitative and quantitative, for the text domain. We address these gaps by testing the approach on a synthetic testbed of formal languages, with easily controllable and definable attributes in the input space. We define and train language models on a set of formal grammars, and train SAEs on the latent representations of these models under a wide variety of hyperparameter settings. We identify several interpretable latents in the SAEs, and formulate a scaling law defining the relationship between the reconstruction loss of SAEs and their hidden size. We show empirically that the presence of latents correlating to certain features of the input does not imply a causal function in the computation and that the performance of SAEs is highly sensitive to inductive biases.

1 Introduction

In recent years, mechanistic interpretability (MI) has gained currency as an approach towards understanding the functioning of deep neural models. One MI paradigm that has seen remarkable progress across domains is sparse autoencoders (SAEs), which aims to disentangle data (usually either real-world data or internal representations of models) into independent, interpretable components [1–9] (see App. A for related work). In the vision domain, autoencoder-based interpretability methods have been extensively studied [10–13], from both empirical and theoretical points of view. Results on the (non-)identifiability of disentangled representations, the feasibility of the autoencoder paradigm, and the necessity of inductive biases have been established, taking the perspective that autoencoders are a method of *dictionary learning*. The objectives of training autoencoders have also been quantified through a number of metrics. Many of these results have been arrived at through a comprehensive evaluation of the method on a *synthetic testbed* [10]. We propose an analogous study for the text domain, aiming to stress-test the SAE approach to interpretability of LM representations. Specifically, we conduct a wide-ranging study of SAEs on a set of formal languages, which we propose as a synthetic testbed for the text domain. Our contributions are as follows.

- We define formal languages of different complexities, train Transformers on these languages, and then analyze SAEs trained on their representations under several different hyperparameter settings. We **identify several features** occurring in these SAEs that correspond to variables central to the data generating process (*e.g.* depth, part of speech). We also observe a scaling law in the reconstruction losses of a particular class of SAEs with respect to hidden dimension size.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

- We demonstrate, in line with results from the vision domain, that the identifiability of disentangled features is not robust. Results are **highly sensitive** to changes in normalization methods, hyperparameters, and other such settings. For example, models trained under L_1 regularization consistently fail to find interpretable features; additionally, normalization has widely varying effects on the reconstruction capabilities of the SAEs.
- We demonstrate that mere identification of **disentangled features does not imply said features will be causally relevant** to the model's computation. We believe that future approaches should work towards integrating causality as a necessary property into the training objective of SAEs, similar to prior work in vision [14].

2 Experiments

Our experiments consist of training SAEs (of various paradigms) on the intermediate representation of Transformer models trained on our formal languages. We explain the data generating process, model architectures, and autoencoder paradigms next.

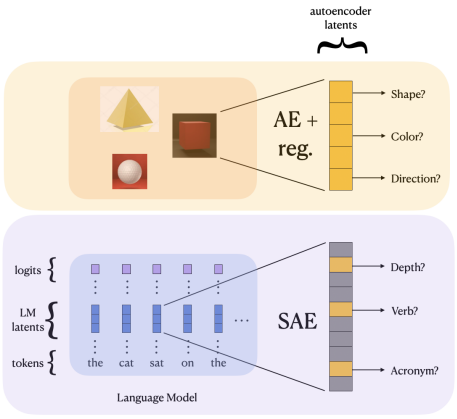


Figure 1: **The autoencoder paradigm for interpretability.** Autoencoders have formed the basis of approaches to disentanglement via dictionary learning in the vision domain [10–12, 15]. They are incentivized towards disentanglement via some form of regularization. While SAEs have similarly been used to disentangle latent representations of language models for interpretability, few of the guarantees and assumptions of the paradigm have been shown to transfer across the domains.

2.1 Data and Models

The formal languages we work with are probabilistic context-free grammars (PCFGs), which are generated by starting with a fixed 'start' symbol, and probabilistically replacing nonterminals according to production rules. We work with three PCFGs, intended to represent levels of complexity (in parsing and generation). In order of increasing complexity, the languages we consider are Dyck-2 (the language consisting of all matched bracket sequences with two types of brackets), Expr (a language of prefix arithmetic expressions), and English (a simple fragment of English syntax with only subject-verb-object constructions). See App. B for precise details.

We train Transformer models on an autoregressive language modelling task on each of the above languages (separately). This is implemented as next token prediction and cross-entropy loss. The models have 128-dimensional embeddings, with 4 attention heads and MLPs, and 2 blocks. In all cases, the models achieve more than 99% validity (*i.e.*, under stochastic decoding, the strings generated belong to the language more than 99% of the time).

2.2 SAEs

Broadly, we use the conventional SAE architecture, which consists of two linear layers (an encoder and a decoder transform) with an activation function in between. More explicitly, if the sizes of the input and hidden state are d and h respectively, then our SAEs implement functions of the type

$$\mathrm{SAE}(x) = W_{\text{dec}}(\sigma(W_{\text{enc}}(x) + b_{\text{enc}})) + b_{\text{dec}},$$

where $W_{\text{enc}} \in \mathbb{R}^{d\times h}$, $W_{\text{dec}} \in \mathbb{R}^{h\times d}$, and $b_{\text{enc}} \in R^d, b_{\text{dec}} \in R^h$. We pick h from $\{d, 2d, 4d, 8d\}$ and set $\sigma = \mathrm{ReLU}$. We optimize on the MSE between x and $\mathrm{SAE}(x)$. The input x to the SAEs is taken from end of the first block of the Transformer model. We allow for two main modifications to this architecture: the normalization method and `pre_bias`. There are three parts of the operation of the SAE that can be normalized – the input, the reconstruction, and the decoder directions. We allow for four settings: *no normalization (I), input and decoder with* `pre_bias` *(II), input alone (III), and input and reconstruction (IV).* By `pre_bias`, we refer to the addition of a learnable vector $b_{\text{pre}} \in \mathbb{R}^d$ to the input before applying the encoder transform, which is then subtracted after the decoder transform. Sparsity is enforced via two main methods. The first (and most common in

existing work) is an L_1-regularization term added to the loss, which encourages latent representations to have low L_1-norm. The hyperparameter for this method is the weight of the regularization term. We also use, following Gao et al. [7], top-k regularization – at the SAE hidden layer, after the activation function, we select the highest k latents, and zero out the rest. Effectively, we force the L_0 norm of the latents to be at most k, *i.e.*, the hyperparameter is k itself.

3 Results

Features. Qualitatively, we observe that top-k-regularized SAEs tend to have more interpretable features than L_1-regularized ones. The former results in interpretable features across all languages – we find features representing fundamental aspects of the corresponding grammars, as discussed next briefly. See App. C for further results, including other features we are able to identify and how strongly features correlate with their claimed explanations.

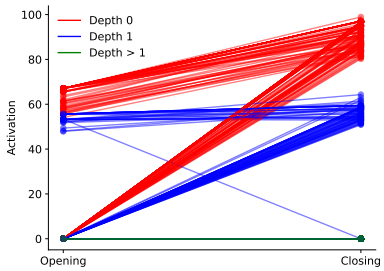


Figure 2: A feature matching corresponding opening and closing brackets. Each line represents a pair of brackets, and joins the opening bracket's activation (left) to the closing bracket's (right). We note that the depth and opening activation are sufficient to determine the closing activation, and that the opening and closing activations are sufficient to determine the depth.

In the case of Dyck-2, we expect the depth (the number of brackets yet to be closed) to be represented. We find features that *threshold* the depth of tokens, *i.e.*, they activate on tokens with depth above a certain threshold depth D. Usually, D is greater than the mean; for instance, when mean depth is 8.3, we find $D = 11$. We also find features that match corresponding pairs of opening and closing brackets, usually at lower depths (Fig. 2). These features take values from only a few small ranges, and their values at an opening bracket determine those they take at the matching closing bracket. These features, even though they are binary, indicate that the token representations do maintain some form of stack counting (which aligns with results from prior work [16]).

In Expr, an analogue of depth thresholding is maintaining a counter that indicates how many more expressions are needed to complete the sequence (see App. B). We find a feature that activates exactly when this counter's value is 1, *i.e.*, when exactly one expression is required (Fig. 3). This provides strong evidence of a counter process being implemented—in fact, generation without this process would be rather convoluted (see Alg. 1).

An inference we can make from the Expr features is that there is an implicit "type" feature in the representations, distinguishing operators of different valences. A natural place to look for this is the parts of speech in our English grammar, and we find that k-regularized SAEs do contain features corresponding to each part of speech. For example, we illustrate the 'adjective' feature in Fig. 4.

Further, we note a scaling law in the accuracy of the top-k-regularized autoencoders. The reconstruction loss they achieve (after a fixed number of iterations) decreases according to a power law with the size of the autoencoder's hidden layer; similar results were seen by Sharkey et al. [2], relating the reconstruction loss to the L1 penalty coefficient.

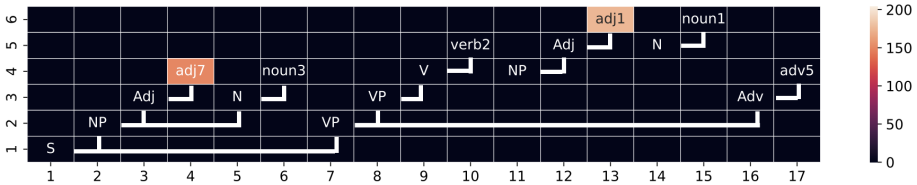


Figure 4: A feature that activates only on adjectives, at any position. Here, depth is represented by the y-axis and position by the x-axis; the lines connect nonterminals to their productions (see App. B for the production rules). The cell color represents the activation magnitude.

Table 1: Reconstruction Accuracy (%) averaged over regularization values and hidden size. We present the accuracies for SAEs with no normalization (I); with inputs and decoder normalized, and `pre_bias` (II); with inputs normalized (III); and with inputs and reconstructions normalized (IV).

Language	L_1				top-k			
	I	II	III	IV	I	II	III	IV
Dyck-2	0.01	0.0	0.01	0.0	49.27	3.48	50.02	0.06
Expr	33.31	6.50	0.06	0.49	99.88	69.14	99.76	0.0
English	0.29	1.13	0.01	0.01	92.46	50.12	80.79	0.37

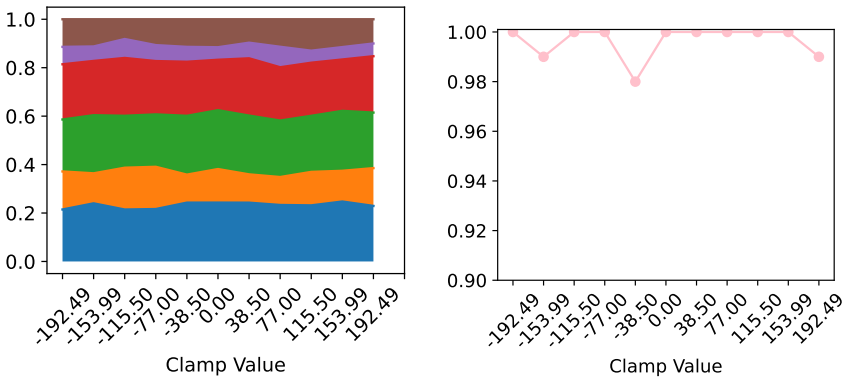


Figure 5: We intervene on the model, replacing its hidden representations with the SAE's reconstructions, clamping a single latent (in this case, the one corresponding to adjectives) to fixed value. These fixed values are selected at uniform intervals from $[-v_{\max}, v_{\max}]$, where $v_{\max}$ is the maximum value taken by that latent. For each value of the clamp (x-axis), we plot the fraction of each part of speech (nouns, pronouns, adjectives, verbs, adverbs, and conjunctions) in the generated text (left) and the fraction of generations that are grammatical (right). We see that the clamp has no effect.

Sensitivity to Hyperparameters. As already remarked, top-k-regularized SAEs consistently reveal more interpretable features in our languages than L_1-regularized SAEs. We also measure the *reconstruction accuracy*, or the percentage of valid generations of the model after the latents are substituted with the SAE reconstructions. Table 1 shows the average reconstruction accuracy for each regularization-`pre_bias`-normalization combination (averaged across regularization values and hidden sizes). We see that top-k-regularized SAEs usually (but not always) outperform L_1-regularized ones if all other settings are kept the same. We also note from this table the high sensitivity to hyperparameter settings that SAEs exhibit – no clear trend is visible across languages, regularization methods, or normalization settings.

Causality. An interesting aspect of the features we identify is that despite strongly correlating with the discussed explanation (see Table 2), they do *not* have the expected causal effects. For example, we intervene on the part-of-speech features in English and the counter features in Dyck-2 models, and they do not change the output distributions in an expected manner. We present an example in Fig. 5; more examples are shown in App. D.

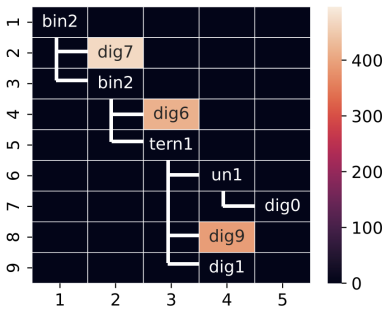


Figure 3: A feature that activates when exactly one more expression is required. Here, the x-axis is token depth, and the y-axis is token index. The lines connect the operators to their operands.

4 Conclusion

Inspired by studies in vision [10, 13], we propose a minimalistic setup to assess challenges in use of SAEs for model interpretability. We validate our setup by identifying semantically meaningful features and demonstrating the sensitivity of said features' extraction to inductive

biases. We further demonstrate a lack of causality, *i.e.*, interventions on these features do not yield intended effects. We expect these results to bear out at scale as well, *e.g.*, we find it likely that features identified by SAEs will not always be causally relevant to model computation. Explicit tools for identifying causal graphs based on SAE features will thus be needed [6]. Looking forward, we argue causality should be modeled as a constraint within the SAE training pipeline itself, similar to prior work in vision [14].

References

[1] Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, János Kramár, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *arXiv preprint arXiv:2408.05147*, 2024.

[2] Lee Sharkey, Dan Braun, and Beren Millidge. [interim research report] taking features out of superposition with sparse autoencoders. `https://www.lesswrong.com/posts/z6QQJbtpkEAX3Aojj`, 2024.

[3] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nicholas L. Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E. Burke, Tristan Hume, Shan Carter, Tom Henighan, and Chris Olah. Towards monosemanticity: Decomposing language models with dictionary learning. `https://transformer-circuits.pub/2023/monosemantic-features`, 2023.

[4] 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.

[5] Aleksandar Makelov, George Lange, and Neel Nanda. Towards principled evaluations of sparse autoencoders for interpretability and control. *arXiv preprint arXiv:2405.08366*, 2024.

[6] Samuel Marks, Can Rager, Eric J Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. *arXiv preprint arXiv:2403.19647*, 2024.

[7] 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.

[8] Samyak Jain, Ekdeep Singh Lubana, Kemal Oksuz, Tom Joy, Philip HS Torr, Amartya Sanyal, and Puneet K Dokania. What makes and breaks safety fine-tuning? mechanistic study. *arXiv preprint arXiv:2407.10264*, 2024.

[9] Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, János Kramár, Rohin Shah, and Neel Nanda. Improving dictionary learning with gated sparse autoencoders. *arXiv preprint arXiv:2404.16014*, 2024.

[10] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pages 4114–4124. PMLR, 2019.

[11] Frederik Träuble, Elliot Creager, Niki Kilbertus, Francesco Locatello, Andrea Dittadi, Anirudh Goyal, Bernhard Schölkopf, and Stefan Bauer. On disentangled representations learned from correlated data. In *International conference on machine learning*, pages 10401–10412. PMLR, 2021.

[12] Sébastien Lachapelle, Tristan Deleu, Divyat Mahajan, Ioannis Mitliagkas, Yoshua Bengio, Simon Lacoste-Julien, and Quentin Bertrand. Synergies between disentanglement and sparsity: Generalization and identifiability in multi-task learning. In *International Conference on Machine Learning*, pages 18171–18206. PMLR, 2023.

[13] Francesco Locatello, Ben Poole, Gunnar Rätsch, Bernhard Schölkopf, Olivier Bachem, and Michael Tschannen. Weakly-supervised disentanglement without compromises. In *International Conference on Machine Learning*, pages 6348–6359. PMLR, 2020.

[14] Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Towards causal representation learning. *arXiv preprint arXiv:2102.11107*, 2021.

[15] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *International Conference on Learning Representations*, 2017.

[16] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. *arXiv preprint arXiv:2009.11264*, 2020.

[17] Aapo Hyvärinen and Petteri Pajunen. Nonlinear independent component analysis: Existence and uniqueness results. *Neural networks*, 12(3):429–439, 1999.

[18] Chandler Squires, Anna Seigal, Salil S Bhate, and Caroline Uhler. Linear causal disentanglement via interventions. In *International Conference on Machine Learning*, pages 32540–32560. PMLR, 2023.

[19] Anna Sepliarskaia, Julia Kiseleva, and Maarten de Rijke. How to not measure disentanglement. *arXiv preprint arXiv:1910.05587*, 2019.

[20] Rui Shu, Yining Chen, Abhishek Kumar, Stefano Ermon, and Ben Poole. Weakly supervised disentanglement with guarantees. *arXiv preprint arXiv:1910.09772*, 2019.

[21] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

Appendix

A Related Work

Prior work that has influenced the SAE approach to interpretability can be organized into four main directions: disentanglement, inductive biases, evaluation and metrics, and the nature of SAE features.

A.1 Disentanglement

Locatello et al. [10], working in the image domain, carry out a wide-ranging empirical and theoretical study of VAEs on synthetic image data, and prove a non-identifiability result on the learnt latent representations. In other words, infinitely many possibilities for disentangled latent representations exist if only the data is given, and so the disentanglement of the actual representations learnt is extremely sensitive to the inductive biases of the autoencoder being used. Thus, no guarantees about the interpretability or task-specific usefulness of the learnt representations can be assumed. Locatello et al. [10] note also that identifiability results are in general not obtainable for the nonlinear case [17]. Furthermore, Träuble et al. [11] observe that real-world data can only be generated from highly correlated latents, possibly with a complex causal relationship. They prove also that disentangled representations do not represent an optimum in this case, and so entangled representations are learnt. However, they also note that supervision can be leveraged to achieve true disentanglement – supplementary data linking priors to observations can be used for weak supervision during training to resolve latent correlations. Later works outlined settings in which identifiability results can in fact be proven. For example, Lachapelle et al. [12] propose a bi-level optimization problem, where the representations are optimized for reconstruction as well as performance on downstream tasks via sparse classifiers, and Squires et al. [18] demonstrate how to achieve disentanglement with interventional data (data from observations with individual latents ablated).

A.2 Scaling Laws and Inductive Biases in SAEs

Locatello et al. [10] highlight the importance of drawing attention to the inductive biases of autoencoders while using them to achieve disentanglement. They show that the objective function and random seed together are responsible for roughly 80% of the performance of VAE encoders, demonstrating the lack of robustness in the method. Many works have also obtained empirical results on the relationship between the dictionary size (*i.e.*, the latent size of the autoencoder) and the features learnt. For example, Sharkey et al. [2] (working with numerical data) note that recovering the ground-truth features requires a dictionary size of 1-8x the number of latents, and that if sparsity is enforced by L_1-regularization, then larger dictionary sizes need larger penalties. Other studies have found that "dead" features (*i.e.*, features that don't activate on any sample) begin to occur from a dictionary size of about 4x [4], that a single feature in small SAEs "splits" into several features (whose union represents the former feature) in larger SAEs [3, 5], and that several features are simply the same token in various contexts, like a physics "the" and a mathematics "the" [3].

A.3 Metrics

Many aspects of SAEs have been identified as important for evaluation, and many metrics exist for each of these. Mainly, they can be classified along two axes: interpretability vs. disentanglement, and supervised vs. unsupervised. Supervised metrics require some ground-truth dictionary of features to evaluate against, which is generally assumed to be human-interpretable. Therefore, interpretability and disentanglement are tied together in these metrics. For example, BetaVAE uses the accuracy of a classifier trained to predict ground-truth features from learned ones [10, 19]; consistency and restrictiveness measure the sensitivity of ground-truth features to learned ones [20]; and maximum mean cosine similarity (MMCS) maps the two sets of features using the cosine similarity [2]. However, when no ground-truth is available, a feature set may be disentangled but not interpretable, or vice versa. Thus, unsupervised metrics evaluate interpretability and disentanglement separately. Examples of metrics for interpretability are controllability, which evaluates how 'easy' it is to control the model output by intervening on a feature set [5], and next logit attribution, where the causal role of the feature in the model's final logit output is examined [3]. Notably, Makelov et al. [5] find that SAEs

trained on task-specific data (IOI in their case study) learn meaningful directions, while those trained on full data perform on par with those that have random directions kept frozen through training.

A.4 Correlational and Causal Features

A number of studies demonstrate the causal effects of SAE features. For example, Bricken et al. [3] show that the features representing Arabic-language text can be clamped to a high value, increasing the probability of generating Arabic text. Most notably, Marks et al. [6] use SAE directions to identify circuits in models (across layers) responsible for specific tasks, like subject-verb agreement across relative clauses. However, as shown in our results, a bulk of features have no causal effects on the model computation. We claim similar results can be easily demonstrated in natural settings as well.

B Formal Grammars

We use, as mentioned in Sec. 2, three formal grammars on which we train transformer-based language models. The exact specification of these grammars, in terms of a context-free grammar (CFG) is presented below. Note that the actual data generation process is defined by a probabilistic CFG, which requires probabilities to be assigned to each of the productions of a nonterminal. We omit these probabilities for legibility here, but readers can refer to our codebase (`https://github.com/Abhinav271828/pcfg-sae-mint`) for details.

B.1 Dyck-2

Given n types of brackets (that is, $2n$ symbols consisting of n opening and the matching n closing brackets), the Dyck-n consists of all the valid sequences of brackets. Algorithmically, a string belonging to Dyck-n can be parsed by maintaining a stack of opening brackets, and popping the topmost one when the corresponding closing bracket is encountered. If a closing bracket that does not match the topmost opening bracket is encountered, the string is rejected. The production rules that express the generation of strings from Dyck-2, then, are as follows.

$$\begin{aligned} S &\to S\,S \mid B_1 \mid B_2 \\ B_1 &\to (\,S\,) \mid (\ ) \\ B_2 &\to [\,S\,] \mid [\ ] \end{aligned}$$

B.2 Expr

The Expr language is the set of prefix arithmetic expressions, in which operands are single digits from 0 to 9, and operators may be unary, binary or ternary. There are three operators of each type. Note that it is possible to view the vocabulary as being organized according to *arity*, or number of arguments needed. Thus digits are symbols of arity 0; unary operators of arity 1; binary operators of arity 2; and ternary operators of arity 3.

Since the syntax is prefix, there is no need to define precedence for unambiguous parsing. For parsing, it is sufficient to maintain a counter that keeps track of how many more expressions are needed to complete the sequence – this counter starts at 1 (as the whole sequence, which is pending, represents one expression), and is incremented by $n-1$ when we encounter a token of arity n. Thus, for instance, if 3 more expressions are needed to complete the sequence and a binary operator is encountered, we now need 4 more expressions – two to complete the binary operator, and two more to satisfy the original three. Parsing succeeds if this value reaches 0 at the end of the string, and fails if it becomes negative at any point during the parse.

The production rules for this language are as follows.

$$\begin{aligned} S &\to O \mid D \\ D &\to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ O &\to U\,S \mid B\,S\,S \mid T\,S\,S\,S \\ U &\to \text{un}_1 \mid \text{un}_2 \mid \text{un}_3 \\ B &\to \text{bin}_1 \mid \text{bin}_2 \mid \text{bin}_3 \\ T &\to \text{tern}_1 \mid \text{tern}_2 \mid \text{tern}_3 \end{aligned}$$

Algorithm 1 Identify points in the sequence where exactly one more expression is expected, without maintaining an explicit counter.

Require: Tokens $t_1, \ldots, t_n$
  $i = 1$
  while $i \leq n$ **do**
    if t_i is a unary operator **then**
      mark t_i
      $i \leftarrow i + 1$
    else if t_i is a binary operator **then**
      $i \leftarrow$ the position of the last token of the first operand of t_i
      mark t_i
    else if $_i$ is a ternary operator **then**
      $i \leftarrow$ the position of the last token of the second operand of t_i
      mark t_i
    else
      $i \leftarrow i + 1$
    end if
  end while

As we note in Sec. 2, SAEs trained on Expr models find a feature that activates on tokens where exactly one more expression is needed to complete the sequence, *i.e.*, where the counter above has a value of 1. We consider it reasonable to assume the implicit computation of such a counter based on the existence of this feature, since the algorithm required to identify these tokens (Algorithm 1) is much more convoluted if we forbid explicitly computing this counter.

B.3 English

We define a simple fragment of English, intended to capture major parts of speech and bridge the gap between parsing languages like Dyck-2 and Expr above, and natural language parsing. We retain the two most common sentence constructions, but ignore more complicated syntactic features like agreement and relative clauses, and morphological features, like conjugations and declensions. This grammar can be parsed using any standard CFG parsing algorithm, like Earley or CKY parsing [21].

The rules for the grammar are as follows.

$$
\begin{aligned}
S &\to \text{NP VP} \\
\text{NP} &\to \text{Pro} \mid \text{N} \mid \text{NP Conj NP} \mid \text{Adj N} \\
\text{VP} &\to \text{V} \mid \text{V NP} \mid \text{VP Conj VP} \mid \text{VP Adv}
\end{aligned}
$$

Each part of speech – nouns (N), verbs (V), pronouns (Pro), conjunctions (Conj), adjectives (Adj) and adverbs (Adv) – have ten tokens each. We omit the production rules listing these for brevity, but refer readers to the codebase (`https://github.com/Abhinav271828/pcfg-sae-mint`) for details.

C Features

We describe here several other features we were able to identify in our SAEs. Results are shown in Table 2. Note that all the SAEs listed here are trained without `pre_bias` and without normalization. Thus, the hyperparameters specified in column 3 are the expansion factor (ratio between hidden size and input size) and, according to the regularization method, either the L_1 coefficient or k. For each feature, we note the hyperparameter settings of the SAE, our hypothesized explanation, and the correlation (Pearson coefficient) between the feature's activation and the explanation.

In the Expr model, the SAEs identify features relating to a counter variable (based on the parsing process described in Section D.3) and position. Of the former kind, we find a feature that activates exactly when there is one more expression left to complete the sequence; as an example of the latter, there are features that activate only on the last token of a sequence.

In the model trained on a fragment of English, we find features representing several parts of speech – adjectives, verbs, adverbs, and conjunctions. Each of these activate to varying degrees on tokens belonging to the respective part of speech.

In the Dyck-2 model, as in the case of Expr, we see features that correspond to an intuitive stack-based parsing process. In Section 3, we present a feature that 'matches' corresponding opening and closing trends. There are also features that threshold the depth of (number of unclosed brackets that appear before) a token, *i.e.*, they activate when the depth is above a certain value (11 in this case). We also find features that identify a combination of depth and whether a bracket is opening or closing.

Table 2: Features Identified by SAEs. We give a description of each feature, with the language it is found in, and the correlation (Pearson coefficient) between the activations and the explanation. All the SAEs are trained without `pre_bias` or normalization; we therefore identify them by their expansion factor and regularization factor (α in the case of L_1- and k in the case of top-k regularization).

Explanation	Language	Settings	Correlation
One more expression required to complete the sequence.	Expr	(2, $k = 16$)	0.972
Last token.	Expr	(8, $\alpha = 10^{-3}$)	0.984
Verbs.	English	(8, $k = 128$)	0.964
Adjectives.	English	(8, $k = 128$)	0.985
Adverbs.	English	(8, $k = 128$)	0.999
Conjunctions.	English	(8, $k = 128$)	0.932
Stack depth 11 or more.	Dyck	(8, $k = 128$)	0.924
All brackets at depth 0, and the first opening and all closing brackets at depth 1.	Dyck	(8, $k = 128$)	0.912

D Causality of Features

We present here a detailed outline of our experimental protocols in the causality experiments. We also examine other features for causal effects, particularly part-of-speech features (for English) and the counter feature (for Expr, described in Sec. 3).

D.1 Experimental Protocols

In order to examine the causal effects of a feature identified by an SAE, we rely on interventions that replace this feature by some fixed value, and continue the computation of the larger model. In other words, consider a computational graph of the language model. The node corresponding to the layer that is being examined, *i.e.*, the first hidden layer, is replaced by three nodes:

i. the actual representations generated by the previous layer, which form the input to the SAE;

ii. the hidden layer *of the SAE*, which consists of the features identified by the SAE;

iii. the output, or the reconstruction, of the SAE.

The last node feeds back into the larger model, where the original input should have been used. Thus, we effectively replace the model activations with SAE reconstructions of those activations. We then intervene on node (ii) above – we select a single element in the SAE representation, set it to our value, and recompute the modified reconstruction, which is then used in the rest of the LM's forward pass.

The exact intervention that we carry out is defined in terms of the maximum value v_{max} that the feature attains across a sample of 1280 sequences. We then select the values for the intervention by spacing 10 intervals across the range $[-v_{\text{max}}, v_{\text{max}}]$, *i.e.*, the intervention values are

$$v_i = -v_{\text{max}} + \frac{i}{10} \cdot 2v_{\text{max}}, \forall i \in \{0, 1, \ldots, 10\}.$$

Note that the above leads to 11 possible values for the interventions. As recorded in Fig. 5, we then measure the distribution of parts of speech in the generations of the model under this intervention, and the percentage of the generations that remain grammatical.

D.2 English: Parts of Speech

The results of interventions on various features are shown in Fig. 6. For each part of speech that we show, we follow the same process as in the case of adjectives (Fig. 5):

- determine the maximum value v_{max} taken by a certain latent;
- pick clamp values at uniform intervals in the range $[-v_{\text{max}}, v_{\text{max}}]$;
- examine (i) the distribution of parts of speech and (ii) the grammaticality of the generations after clamping the latent to each value.

We observe that, similar to the adjectives feature in the main paper (see Fig. 5), the features for other parts of speech do not show causal effects either.

D.3 Expr: Counter

We examine the feature, described in Sec. 3, that activates when one more expression is required to complete the sequence. We pass an incomplete sequence to the model, and intervene on this feature by clamping it to one of $\{-v_{\text{max}}, 0, v_{\text{max}}\}$. We then examine the generations that result in each case. We expect that high clamp values will cause the model to generate only one more expression (even if more may be required); and low values will cause it to generate more than one (even if exactly one is needed). Table 3 presents the results of this experiment; for each input sample, we mention the number of expressions it requires to complete it, and the number of expressions actually generated by the model under each intervention. As is the case with the English model, we observe that there is no causal effect of this feature.

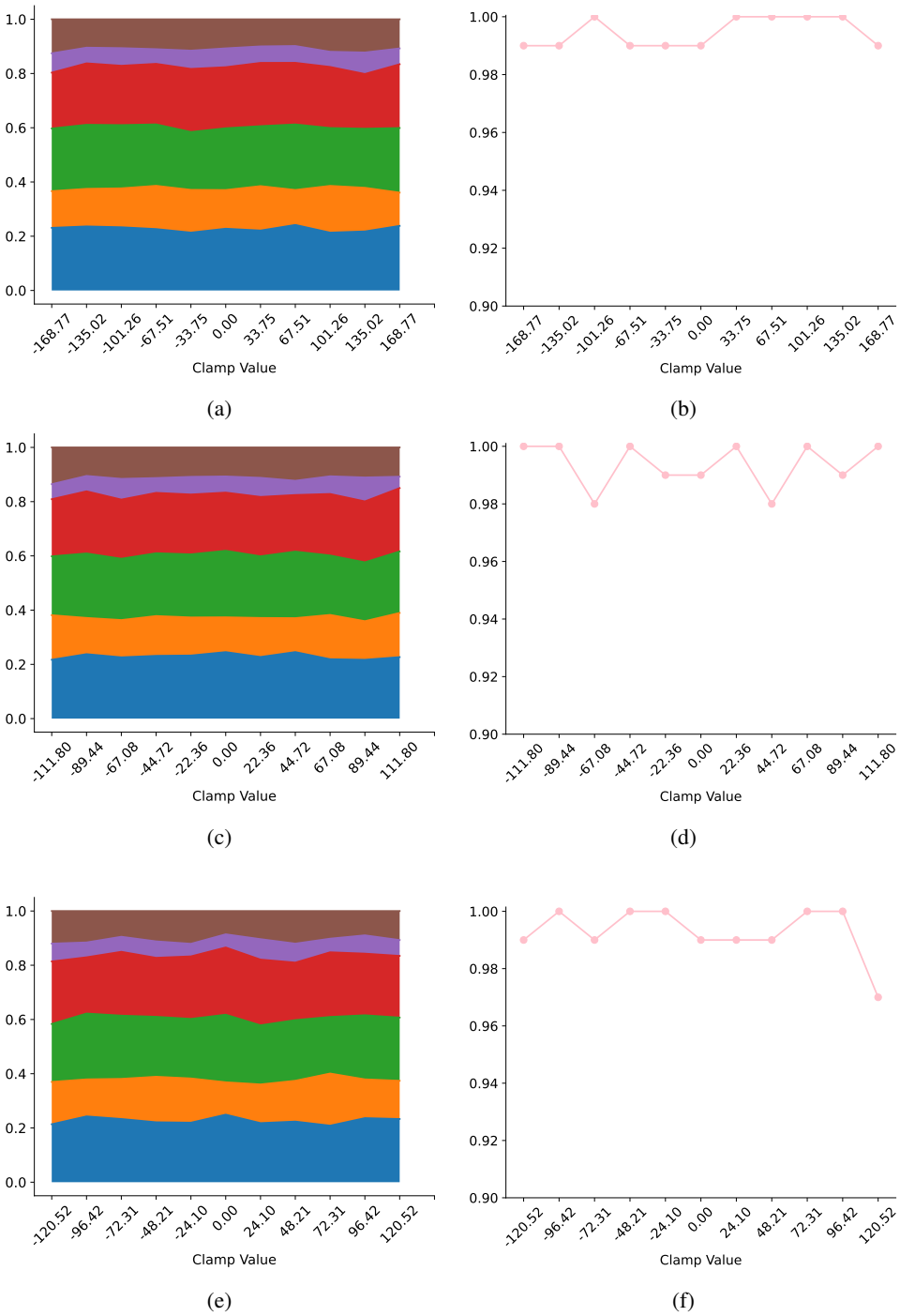


Figure 6: We examine the causality of features corresxponding to other parts of speech: verbs ((a) and (b)), conjunctions ((c) and (d)), and adverbs ((e) and (f)).

Table 3: Behavior of the Expr model under interventions.

Input Sequence	#Required	Clamp $-v_{\max}$	Clamp 0	Clamp $v_{\max}$
`un2 tern1 dig8 bin0 tern2 bin2 bin2 dig6 dig7 dig5`	4	4	4	4
`bin1 un1 tern1 bin2 dig0 dig7 dig0 bin1 dig3 dig7`	1	1	1	1
`un0 tern1 dig9 dig7 un0 tern1 un2 bin2 dig6 dig6`	2	2	2	2
`tern1 dig1 dig7 tern1 tern1 dig7 tern1 tern1 un0 un1`	8	8	8	8

E Power Law in Reconstruction Loss

We noted in Sec. 3 that the top-k regularized SAEs reveal a power law in their reconstruction losses. The trends for various runs are shown in Fig. 7.

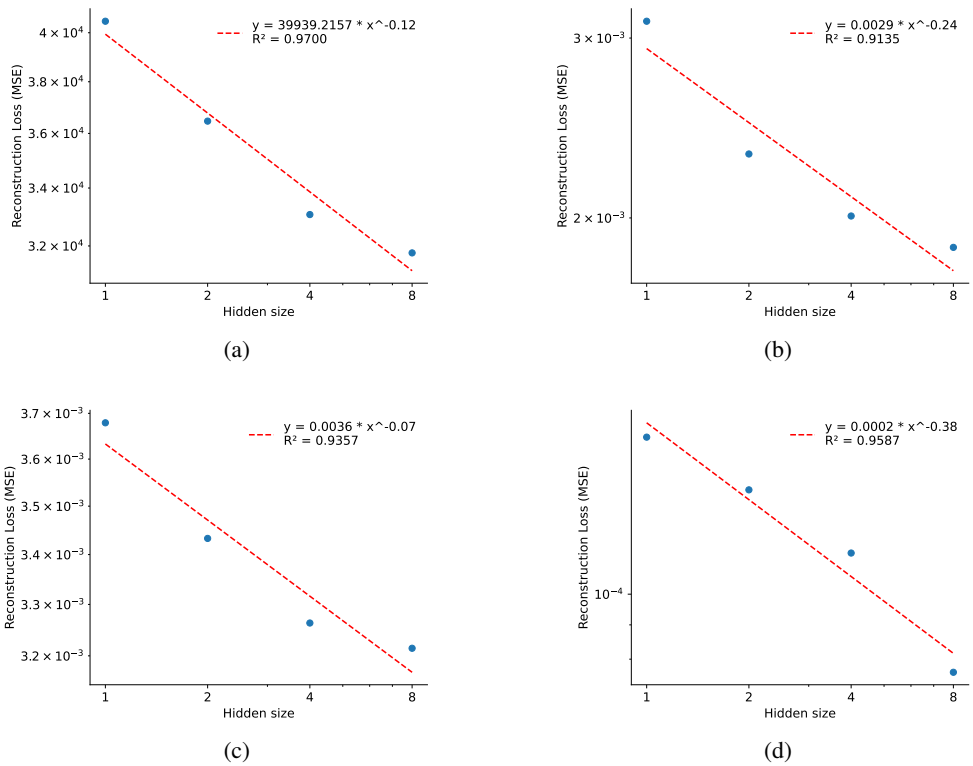


Figure 7: The power law we identify for each top-k regularized set of autoencoders. The four run sets are **(a)** without `pre_bias` or normalization; **(b)** with `pre_bias` and normalized inputs and decoder directions; **(c)** without `pre_bias` and normalized inputs; and **(d)** without `pre_bias` and normalized inputs and reconstructions.

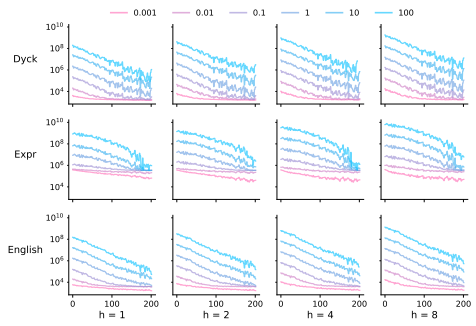
F Training Details

Figs. 8 to 15 show the details of the various hyperparameter settings we have considered, along with plots of the losses during training. For convenience, we record these details in Table 4 as well. Note that, in the case of L_1-regularized SAEs, the training loss is the sum of the reconstruction loss (the MSE loss between the SAE reconstructions and the inputs) and the regularization loss (the norm of the hidden representation, multiplied by the L_1 coefficient).

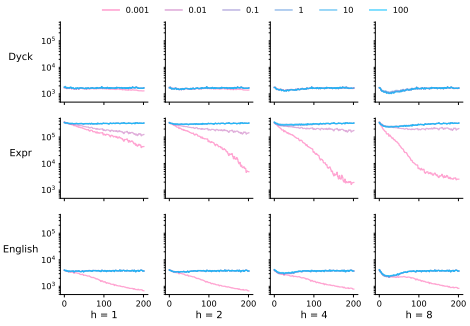
All the autoencoders were trained for 5000 steps, where each step consisted of the embeddings of 128 sequences of that language. The sequences were generated in an online fashion, by sampling from a probabilistic CFG corresponding to the language that the transformer was trained on. The mean sequence length for the languages are 14.03 (for Dyck), 3.90 (for Expr), and 8.26 (for English). This corresponds to about 9 million, 2.5 million, and 5 million tokens respectively. We refer readers to our codebase (`https://github.com/Abhinav271828/pcfg-sae-mint`) for details of the data generation process, hyperparameter sweeps, etc.

Table 4: Hyperparameter Settings and Corresponding Figures.

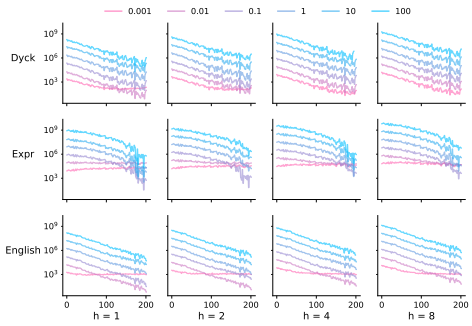
Figure	Regularization	`pre_bias`	Normalization
Fig. 8	L_1	False	none
Fig. 9	top-k	False	none
Fig. 10	L_1	True	input, decoder
Fig. 11	top-k	True	input, decoder
Fig. 12	L_1	False	input
Fig. 13	top-k	False	input
Fig. 14	L_1	False	input, reconstruction
Fig. 15	top-k	False	input, reconstruction



(a) Train Loss (sum of reconstruction and regularization losses)

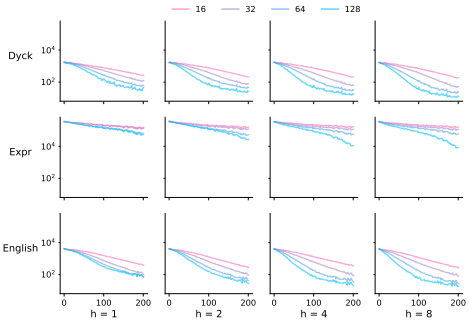


(b) Reconstruction Loss



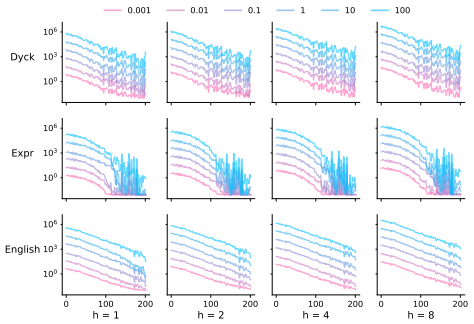
(c) Regularization Loss

Figure 8: L_1-regularized SAEs, without `pre_bias` and without normalization.

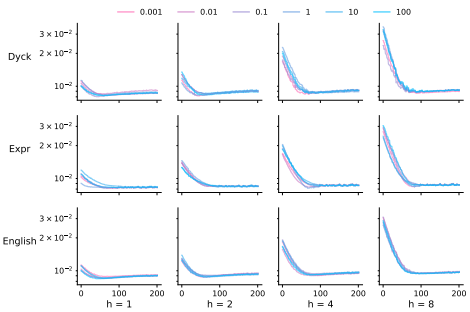


(a) Train Loss (only reconstruction loss)

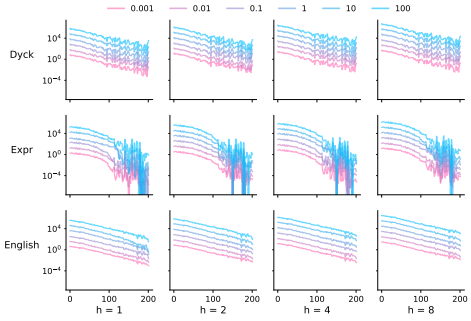
Figure 9: top-k-regularized SAEs, without `pre_bias` and without normalization.



(a) Train Loss (sum of reconstruction and regularization losses)

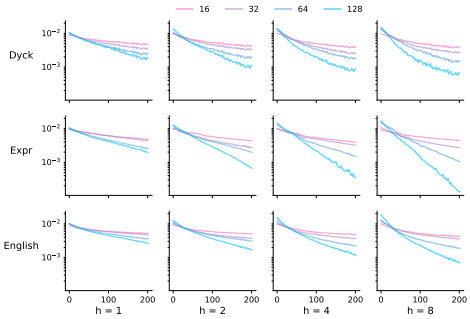


(b) Reconstruction Loss



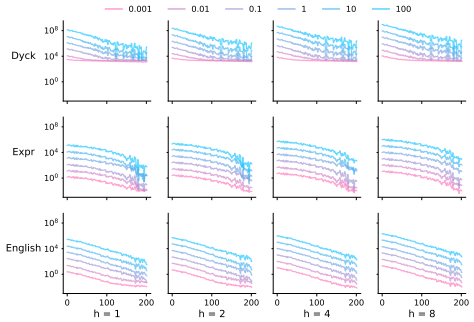
(c) Regularization Loss

Figure 10: L_1-regularized SAEs, with `pre_bias` and normalized inputs and decoder directions.

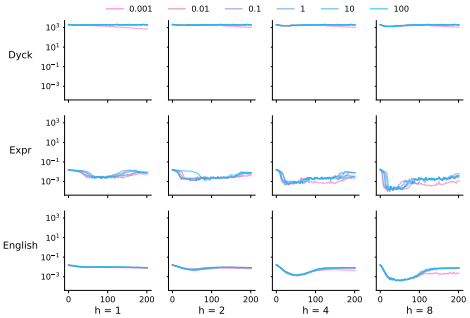


(a) Train Loss (only reconstruction loss)

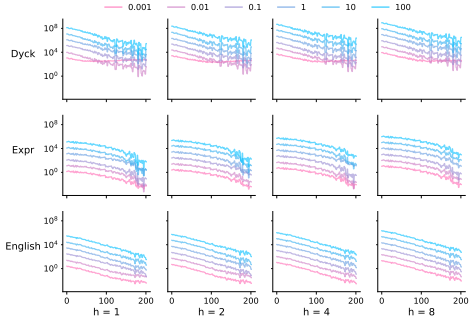
Figure 11: top-k-regularized SAEs, with `pre_bias` and normalized inputs and decoder directions.



(a) Train Loss (sum of reconstruction and regularization losses)

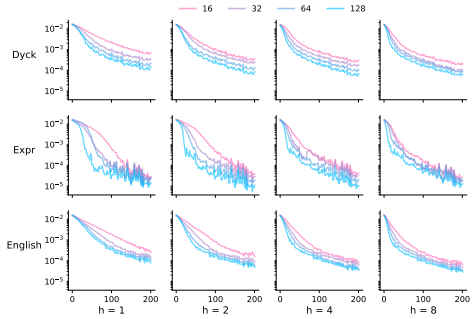


(b) Reconstruction Loss



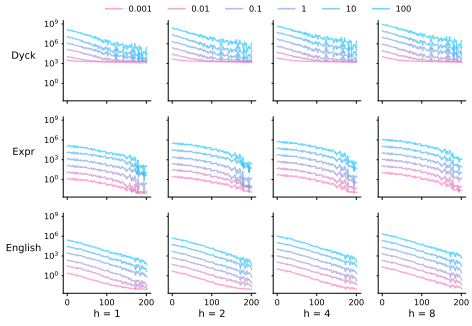
(c) Regularization Loss

Figure 12: L_1-regularized SAEs, without `pre_bias` and normalized inputs.

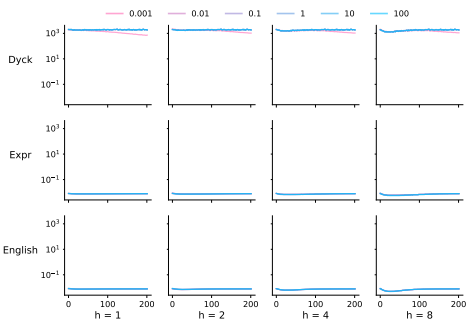


(a) Train Loss (only reconstruction loss)

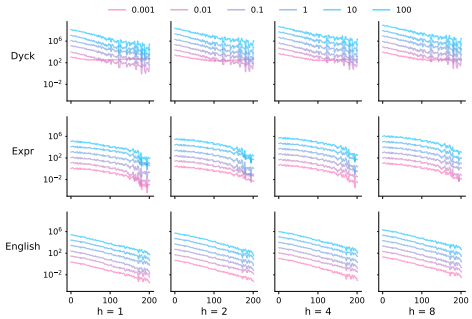
Figure 13: top-k-regularized SAEs, without `pre_bias` and normalized inputs.



(a) Train Loss (sum of reconstruction and regularization losses)

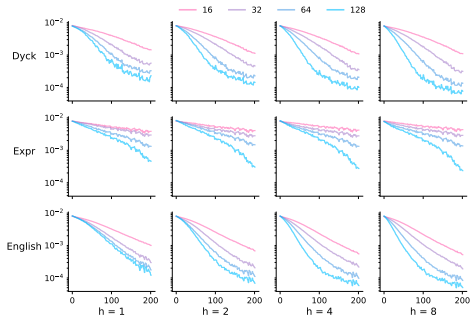


(b) Reconstruction Loss



(c) Regularization Loss

Figure 14: L_1-regularized SAEs, without `pre_bias` and normalized inputs and reconstructions.



(a) Train Loss (only reconstruction loss)

Figure 15: top-k-regularized SAEs, without `pre_bias` and normalized inputs and reconstructions.