SPARSE AUTOENCODERS FIND HIGHLY INTERPRETABLE FEATURES IN LANGUAGE MODELS

Hoagy Cunningham[*12], **Aidan Ewart**[*13], **Logan Riggs**[*1], **Robert Huben**, **Lee Sharkey**[4]
[1]EleutherAI, [2]MATS, [3]Bristol AI Safety Centre, [4]Apollo Research
{hoagycunningham, aidanprattewart, logansmith5}@gmail.com

ABSTRACT

One of the roadblocks to a better understanding of neural networks' internals is *polysemanticity*, where neurons appear to activate in multiple, semantically distinct contexts. Polysemanticity prevents us from identifying concise, human-understandable explanations for what neural networks are doing internally. One hypothesised cause of polysemanticity is *superposition*, where neural networks represent more features than they have neurons by assigning features to an overcomplete set of directions in activation space, rather than to individual neurons. Here, we attempt to identify those directions, using sparse autoencoders to reconstruct the internal activations of a language model. These autoencoders learn sets of sparsely activating features that are more interpretable and monosemantic than directions identified by alternative approaches, where interpretability is measured by automated methods. Ablating these features enables precise model editing, for example, by removing capabilities such as pronoun prediction, while disrupting model behaviour less than prior techniques. This work indicates that it is possible to resolve superposition in language models using a scalable, unsupervised method. Our method may serve as a foundation for future mechanistic interpretability work, which we hope will enable greater model transparency and steerability.

1 INTRODUCTION

Advances in artificial intelligence (AI) have resulted in the development of highly capable AI systems that make decisions for reasons we do not understand. This has caused concern that AI systems that we cannot trust are being widely deployed in the economy and in our lives, introducing a number of novel risks (Hendrycks et al., 2023), including potential future risks that AIs might deceive humans in order to accomplish undesirable goals (Ngo et al., 2022). Mechanistic interpretability seeks to mitigate such risks through understanding how neural networks calculate their outputs, allowing us to reverse engineer parts of their internal processes and make targeted changes to them (Cammarata et al., 2021; Wang et al., 2022; Elhage et al., 2021).

To reverse engineer a neural network, it is necessary to break it down into smaller units (features) that can be analysed in isolation. Using individual neurons as these units has had some success (Olah et al., 2020; Bills et al., 2023), but a key challenge has been that neurons are often *polysemantic*, activating for several unrelated types of feature (Olah et al., 2020). Also, for some types of network activations, such as the residual stream of a transformer, there is little reason to expect features to align with the neuron basis (Elhage et al., 2023).

Elhage et al. (2022) investigate why polysemanticity might arise and hypothesise that it may result from models learning more distinct features than there are dimensions in the layer. They call this phenomenon *superposition*. Since a vector space can only have as many orthogonal vectors as it has dimensions, this means the network would form an overcomplete basis of non-orthogonal

[*]Equal contribution
Code to replicate experiments can be found at https://github.com/HoagyC/sparse_coding

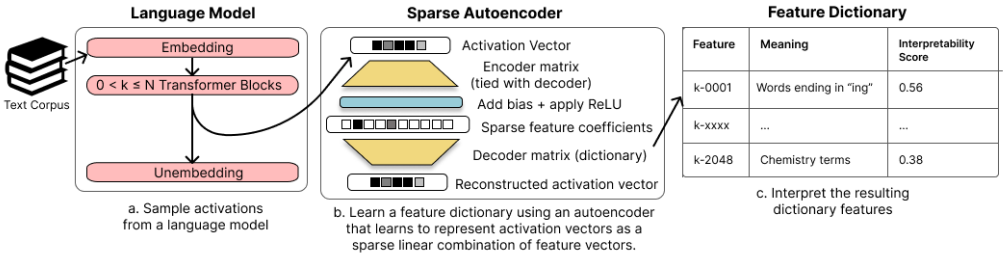

Figure 1: An overview of our method. We a) sample the internal activations of a language model, either the residual stream, MLP sublayer, or attention head sublayer; b) use these activations to train a neural network, a sparse autoencoder whose weights form a feature dictionary; c) interpret the resulting features with techniques such as OpenAI's autointerpretability scores.

features. Features must be sufficiently sparsely activating for superposition to arise because, without high sparsity, interference between non-orthogonal features prevents any performance gain from superposition. This suggests that we may be able to recover the network's features by finding a set of directions in activation space such that each activation vector can be reconstructed from a sparse linear combinations of these directions. This is equivalent to the well-known problem of sparse dictionary learning (Olshausen & Field, 1997).

Building on Sharkey et al. (2023), we train sparse autoencoders to learn these sets of directions. Our approach is also similar to Yun et al. (2021), who apply sparse dictionary learning to all residual stream layers in a language model simultaneously. Our method is summarised in Figure 1 and described in Section 2.

We then use several techniques to verify that our learned features represent a semantically meaningful decomposition of the activation space. First, we show that our features are on average more interpretable than neurons and other matrix decomposition techniques, as measured by autointerpretability scores (Section 3) (Bills et al., 2023). Next, we show that our features enable less-disruptive model editing (Section 4). We also show that we are able to pinpoint the features used for a set task more precisely than other methods (Section 5). Finally, we run case studies on a small number of features, showing that they are not only monosemantic but also have predictable effects on the model outputs, and can be used for fine-grained circuit detection. (Section 6).

2 Taking Features out of Superposition with Sparse Dictionary Learning

To take network features out of superposition, we employ techniques from *sparse dictionary learning* (Olshausen & Field, 1997; Lee et al., 2006). Suppose that each of a given set of vectors $\{\mathbf{x}_i\}_{i=1}^{n_{\text{vec}}} \subset \mathbb{R}^d$ is composed of a sparse linear combination of unknown vectors $\{\mathbf{g}_j\}_{j=1}^{n_{\text{gt}}} \subset \mathbb{R}^d$, i.e. $\mathbf{x}_i = \sum_j a_{i,j}\mathbf{g}_j$ where $\mathbf{a_i}$ is a sparse vector. In our case, the data vectors $\{\mathbf{x}_i\}_{i=1}^{n_{\text{vec}}}$ are internal activations of a language model, such as Pythia-70M (Biderman et al., 2023), and $\{\mathbf{g}_j\}_{j=1}^{n_{\text{gt}}}$ are unknown, ground truth network features. We would like learn a dictionary of vectors, called dictionary features, $\{\mathbf{f}_k\}_{k=1}^{n_{\text{feat}}} \subset \mathbb{R}^d$ where for any network feature $\mathbf{g}_j$ there exists a dictionary feature $\mathbf{f}_k$ such that $\mathbf{g}_j \approx \mathbf{f}_k$.

To learn the dictionary, we train an autoencoder with a sparsity penalty term on its hidden activations (Olshausen & Field, 1997). The autoencoder is a neural network with a single hidden layer of size $d_{\text{hid}} = Rd_{\text{in}}$, where d_{in} is the dimension of the language model internal activation vectors[1], and R is a hyperparameter that controls the ratio of the feature dictionary size to the model dimension. We use the ReLU activation function in the hidden layer (Fukushima, 1975). We also use tied weights for our neural network, meaning the weight matrices of the encoder and decoder are transposes of

[1]For the residual stream in Pythia-70M, $d_{\text{in}} = 512$.

each other.[2] Thus, on input vector $\mathbf{x} \in \{\mathbf{x}_i\}$, our network produces the output $\hat{\mathbf{x}}$, given by

$$\begin{aligned} \mathbf{c} &= \mathrm{ReLU}(M\mathbf{x} + \mathbf{b}) && (1) \\ \hat{\mathbf{x}} &= M^T\mathbf{c} && (2) \\ &= \sum_{i=0}^{d_{\text{hid}}-1} c_i \mathbf{f}_i && (3) \end{aligned}$$

where $M \in \mathbb{R}^{d_{\text{hid}} \times d_{\text{in}}}$ and $\mathbf{b} \in \mathbb{R}^{d_{\text{hid}}}$ are our learned parameters, and M is normalised row-wise[3]. Our parameter matrix M is our feature dictionary, consisting of d_{hid} rows of dictionary features $\mathbf{f}_i$. The output $\hat{\mathbf{x}}$ is meant to be a reconstruction of the original vector $\mathbf{x}$, and the hidden layer $\mathbf{c}$ consists of the coefficients we use in our reconstruction of $\mathbf{x}$.

Our autoencoder is trained to minimise the loss function

$$\mathcal{L}(\mathbf{x}) = \underbrace{||\mathbf{x} - \hat{\mathbf{x}}||_2^2}_{\text{Reconstruction loss}} + \underbrace{\alpha||\mathbf{c}||_1}_{\text{Sparsity loss}}$$

where α is a hyperparameter controlling the sparsity of the reconstruction. The ℓ^1 loss term on $\mathbf{c}$ encourages our reconstruction to be a sparse linear combination of the dictionary features. It can be shown empirically (Sharkey et al., 2023) and theoretically (Wright & Ma, 2022) that reconstruction with an ℓ^1 penalty can recover the ground-truth features that generated the data. For the further details of our training process, see Appendix B.

3 Interpreting Dictionary Features

3.1 Interpretability at Scale

Having learned a set of dictionary features, we want to understand whether our learned features display reduced polysemanticity, and are therefore more interpretable. To do this in a scalable manner, we require a metric to measure how interpretable a dictionary feature is. We use the automated approach introduced in Bills et al. (2023) because it scales well to measuring interpretability on the thousands of dictionary features our autoencoders learn. In summary, the autointerpretability procedure takes samples of text where the dictionary feature activates, asks a language model to write a human-readable interpretation of the dictionary feature, and then prompts the language model to use this description to predict the dictionary feature's activation on other samples of text. The correlation between the model's predicted activations and the actual activations is that feature's interpretability score. See Appendix A and Bills et al. (2023) for further details.

We show descriptions and top-and-random scores for five dictionary features from the layer 1 residual stream in Table 1. The features shown are the first five under the (arbitrary) ordering in the dictionary.

3.2 Sparse dictionary features are more interpretable than baselines

We assess our interpretability scores against a variety of alternative methods for finding dictionaries of features in language models. In particular, we compare interpretability scores on our dictionary features to those produced by a) the default basis, b) random directions, c) Principal Component Analysis (PCA), and d) Independent Component Analysis (ICA). For the random directions and for

[2] We use tied weights because (a) they encode our expectation that the directions which detect and define the feature should be the same or highly similar, (b) they halve the memory cost of the model, and (c) they remove ambiguity about whether the learned direction should be interpreted as the encoder or decoder direction. They do not reduce performance when training on residual stream data but we have observed some reductions in performance when using MLP data.

[3] Normalisation of the rows (dictionary features) prevents the model from reducing the sparsity loss term $||\mathbf{c}||_1$ by increasing the size of the feature vectors in M.

Feature	Description (Generated by GPT-4)	Interpretability Score
1-0000	parts of individual names, especially last names.	0.33
1-0001	actions performed by a subject or object.	-0.11
1-0002	instances of the letter 'W' and words beginning with 'w'.	0.55
1-0003	the number '5' and also records moderate to low activation for personal names and some nouns.	0.57
1-0004	legal terms and court case references.	0.19

Table 1: Results of autointerpretation on the first five features found in the layer 1 residual stream. Autointerpretation produces a description of what the feature means and a score for how well that description predicts other activations.

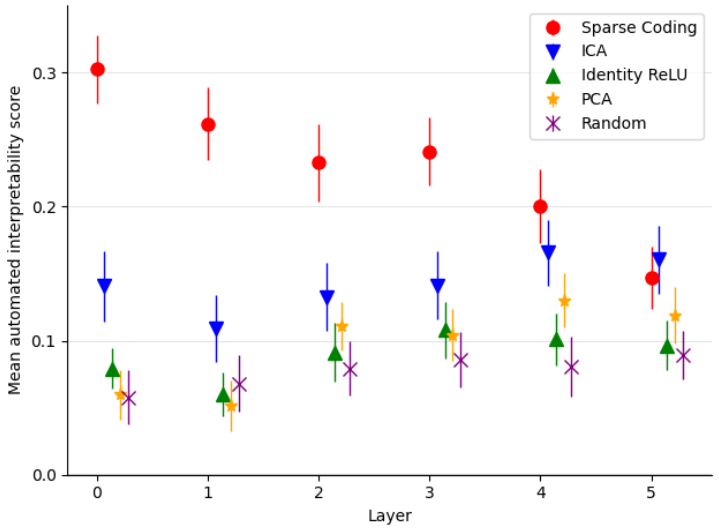

Figure 2: Average top-and-random autointerpretability score of our learned directions in the residual stream, compared to a number of baselines, using 150 features each. Error bars show 95% confidence intervals around means. The feature dictionaries used here were trained for 10 epochs using $\alpha = .00086$ and $R = 2$.

the default basis in the residual stream, we replace negative activations with zeros so that all feature activations are nonnegative [4].

Figure 2 shows that our dictionary features are far more interpretable by this measure than dictionary features found by comparable techniques. We find that the strength of this effect declines as we move through the model, being comparable to ICA in layer 4 and showing minimal improvement in the final layer.

This could indicate that sparse autoencoders work less well in later layers but also may be connected to the difficulties of automatic interpretation, both because by building on earlier layers, later features may be more complex, and because they are often best explained by their effect on the output. Bills et al. (2023) showed that GPT-4 is able to generate explanations that are very close to the average quality of the human-generated explanations given similar data. But they also showed that current LLMs are limited in the kinds of patterns that they can find, sometimes struggling to find patterns that center around next or previous tokens rather than the current token, and in the current protocol are unable to verify outputs by looking at changes in output or other data.

We do show, in Section 6, a method to see a feature's causal effect on the output logits by hand, but we currently do not send this information to the language model for hypothesis generation. The case

[4]For PCA we use an online estimation approach and run the decomposition on the same quantity of data we used for training the autoencoders. For ICA, due to the slower convergence times, we run on only 2GB of data, approximately 4 million activations for the residual stream and 1m activations for the MLPs.

studies section also demonstrates a closing parenthesis dictionary feature, showing that these final layer features can give insight into the model's workings.

See Appendix C for a fuller exploration of different learned dictionaries through the lens of automatic interpretability, looking at both the MLPs and the residual stream.

4 Dictionary Features Enable Surgical Concept Erasure

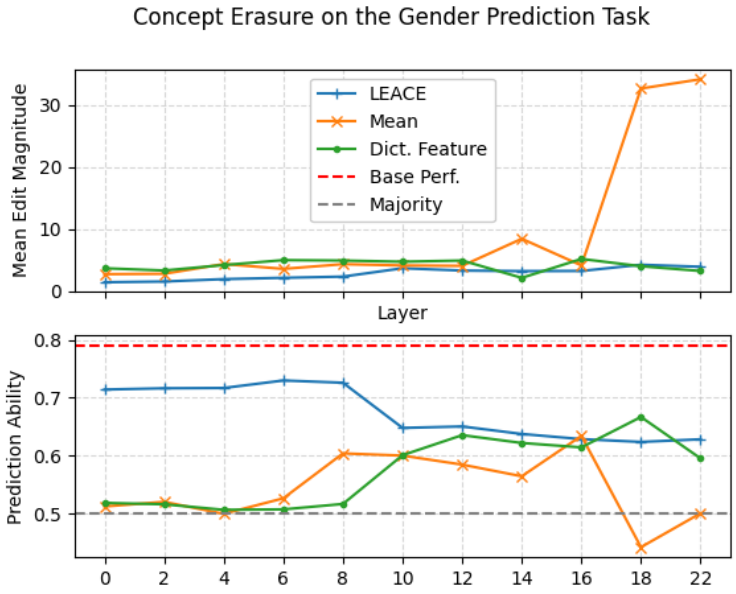

Figure 3: Comparison of concept erasure methods applied to Pythia-410M at various layers. We show the mean edit magnitude (top) and the ability of the altered model to complete the pronoun-prediction task (bottom). Lower is better in both plots. The horizontal 'Base Perf.' line shows the performance of the unedited model, and the horizontal 'Majority' line shows the accuracy of random guessing.

To demonstrate that our dictionary features correspond to functional units in the network, we turn to *concept erasure*. Concept erasure is the task of removing a specified feature from a learned representation (Belrose et al., 2023). Here, we assess each variant of concept erasure by how much our edits hinder the ability of our models to predict a certain concept while aiming to otherwise minimally disrupt model behaviour. On this task, we are able to outperform methods such as LEACE (Belrose et al., 2023) that hide a concept from arbitrary linear probes, while still using single-rank ablations, which we claim is evidence that our linear features better represent the underlying computational units of the model.

4.1 Erasure on the 'Gender Prediction by Name' Task

We evaluate our approach on the pronoun prediction task, using the 'Gender by Name' dataset (mis, 2020). In this task, we few-shot prompt Pythia-410M to complete sentences of the form "My name is $\{name\}$ and my gender is $\{gender\}$." with the stereotypical gender for a given name. We restrict ourselves to male and female genders for simplicity and measure language model predictive ability via the true positive rate of a classifier with log probabilities equal to the model output logits of the corresponding tokens.

The feature dictionaries we used were trained with ℓ^1 penalty coefficient $\alpha = 8e-4$ and dictionary ratio $R = 4$. Note that in contrast to Pythia-70M, Pythia-410M has 24 layers and residual stream dimension $d_{\text{in}} = 1024$. In this section, we perform *full-rank ablations* of dictionary features by

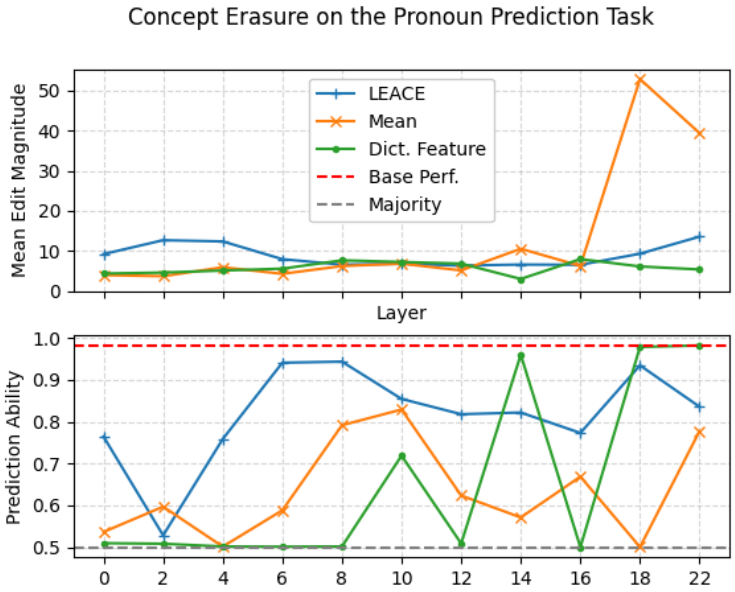

Figure 4: We find that the erasure ability of dictionary feature projections outperform LEACE and difference-in-means projections when transferred to a related task, both in terms of erasure precision as measured by mean edit magnitude (top) and their ability to harm model performance on the task (bottom). Lower is better in both plots.

projecting model activations to the orthogonal complement of that feature. In other words, we prevent the model activations from activating in that direction.

To select dictionary features for ablation, we rank them by their ability to hide gender information from a linear probe, as in Ravfogel et al. (2022), and then evaluate the top four dictionary features on their performance at the 'Gender by Name' task on a 32-element 'train' dataset distinct from the test dataset.

We compare our approach to two benchmarks: LEACE (Belrose et al., 2023) and difference-in-means projections. The difference-in-means projection $P_{\mathbf{u}}$ is defined as

$$\begin{aligned} \mathbf{u} &= \frac{1}{|\mathcal{M}|}\sum_{\mathbf{v}\in\mathcal{M}}\mathbf{v} - \frac{1}{|\mathcal{F}|}\sum_{\mathbf{w}\in\mathcal{F}}\mathbf{w} \\ \hat{\mathbf{u}} &= \frac{\mathbf{u}}{\|\mathbf{u}\|} \\ P_{\mathbf{u}}(\mathbf{x}) &= \mathbf{x} - \langle\hat{\mathbf{u}}, \mathbf{x}\rangle\hat{\mathbf{u}} \end{aligned}$$

where $\mathcal{M}$ and $\mathcal{F}$ are the residual stream activations from the layer being studied corresponding to male and female prompts, respectively. In other words, difference-in-means projection finds the difference $\mathbf{u}$ between the centroids of each gender class, and projects each activation $\mathbf{x}$ onto the subspace orthogonal to that direction.

We apply erasure interventions to each layer individually, and we assume activations at different token positions are i.i.d. for the purposes of LEACE and difference-in-means projections.

We find single features in early layers of the model that, when ablated, outperform LEACE and difference-in-means projections in model prediction accuracy removal (Figures 3 and 4) while having a lower disruption to model behaviour overall, as measured by KL-divergence on the training distribution (Appendix D).

4.2 Transfer to Pronoun Prediction

In order to test for some limited generality in using our learned features for erasure, we test the ability for the features in the previous section to generalise to erasure on a different task. We instead few-shot prompt the model to perform completions of the form "{*name*} went to the store, where {*pronoun*} bought an {*object*}.", choosing the stereotypical pronoun for a given name. We show in Figure 4 that the highest-ranked features for gender prediction often permit good erasure on the pronoun prediction task, while methods such as difference-in-means projection and LEACE fail.

5 Identifying Causally-Relevant Dictionary Features for Indirect Object Identification

We use automated circuit discovery methods (Conmy et al., 2023) to identify dictionary features that are causally relevant for the Indirect Object Identification (IOI) task (Wang et al., 2022). Because our dictionary features are more monosemantic than features identified by other techniques like PCA, we are able to better localize the dictionary features that are causally responsible for model behaviour on the IOI task.

5.1 Adapting activation patching to dictionary features

We apply *activation patching* (Wang et al., 2022) to identify the dictionary features in a given layer responsible for some specified behaviour. To do this, we start with some counterfactual *corrupted* data $\bar{X}_i$ as in (Wang et al., 2022) and identify the minimum number of interventions needed to reproduce the behaviour of the model when run on the uncorrupted data. In practice, we use a variant of automated circuit discovery (ACDC) (Conmy et al., 2023) in which we ablate dictionary features one at a time until the marginal ablation decreases performance beyond the threshold τ. Following Conmy et al. (2023), we measure behaviour reconstruction with KL-divergence from the model's uncorrupted behavior, and we vary the threshold τ to estimate the tradeoff between

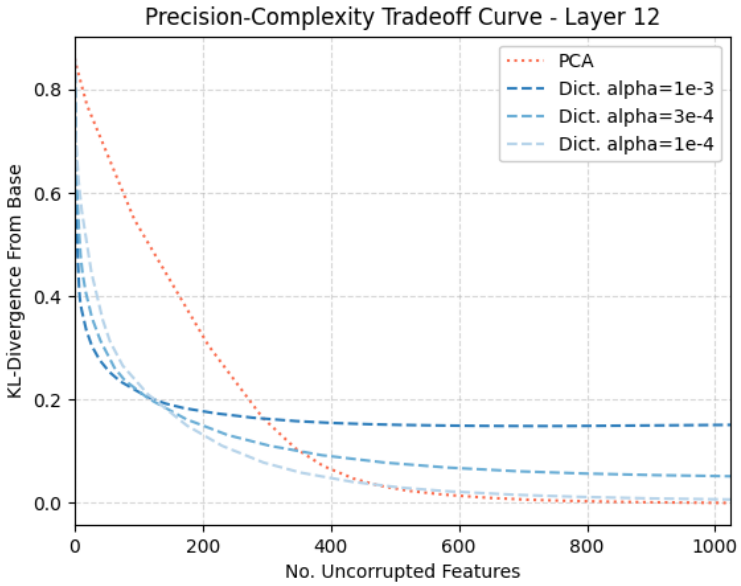

Figure 5: We find a very small number of dictionary features that are causally responsible for the uncorrupted behaviour on the IOI task. Increasing the sparsity of the learned dictionaries improves our ability to precisely locate the responsible dictionary features at the cost of lower overall reconstruction accuracy.

intervention precision and reconstruction of the uncorrupted behaviour. We intervene on activations with the formula

$$\mathbf{x}'_i = \bar{\mathbf{x}}_i + \sum_{j \in F} (c_{i,j} - \bar{c}_{i,j}) \mathbf{f}_j$$

where $\mathbf{x}'_i$ is the edited activation, F is the set of uncorrupted dictionary features, $\bar{\mathbf{x}}_i$ is the transformer's embedding of the corrupted datapoint $\bar{X}_i$, and $\bar{c}_{i,j}$ and $c_{i,j}$ are the activations of our autoencoders on the corrupted and uncorrupted datapoints respectively. We perform interventions on the residual stream at layer 12 of Pythia-410M and compare with an analogous intervention using PCA.

5.2 Precise Localisation of IOI Dictionary Features

As shown in Figure 5, we find that dictionaries with a larger ℓ^1 sparsity coefficient α are able to represent the relevant information for the IOI task with far fewer dictionary features at the cost of having lower overall reconstruction accuracy. This lower reconstruction accuracy entails a larger corrupted residual, which appears in Figure 5 as a larger minimum KL-divergence. We are able to restrict uncorrupted information to a very small number of directions without disrupting model behaviour.

6 Case Studies

In this section, we investigate individual dictionary features, highlighting several that appear to correspond to a single human-understandable explanation (i.e., that are monosemantic). We perform three analyses of our dictionary features to determine their semantic meanings: (1) *Input*: We identify which tokens activate the dictionary feature and in which contexts, (2) *Output*: We determine how ablating the feature changes the output logits of the model, and (3) *Intermediate features*: We identify the dictionary features in previous layers that cause the analysed feature to activate.

6.1 Input: Dictionary Features are Highly Monosemantic

We first analyse our dictionary directions by checking what text causes them to activate. An idealised monosemantic dictionary feature will only activate on text corresponding to a single real-world feature, whereas a polysemantic dictionary feature might activate in unrelated contexts.

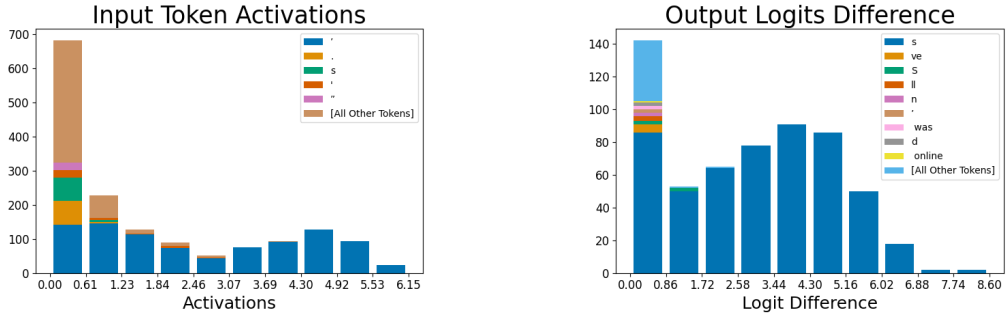

Figure 6: Histogram of token counts for dictionary feature 556. (Left) For all datapoints that activate dictionary feature 556, we show the count of each token in each activation range. The majority of activations are apostrophes, particularly for higher activations. Notably the lower activating tokens are conceptually similar to apostrophes, such as other punctuation. (Right) We show which token predictions are suppressed by ablating the feature, as measured by the difference in logits between the ablated and unablated model. We find that the token whose prediction decreases the most is the "s" token. Note that there are 12k logits negatively effected, but we set a threshold of 0.1 for visual clarity.

To better illustrate the monosemanticity of certain dictionary features, we plot the histogram of activations across token activations. This technique only works for dictionary features that activate for a small set of tokens. We find dictionary features that only activate on apostrophes (Figure 6); periods; the token " the"; and newline characters. The apostrophe feature in Figure 6 stands in contrast to the default basis for the residual stream, where the dimension that most represents an apostrophe is displayed in Figure 14 in Appendix E.1; this dimension is polysemantic since it represents different information at different activation ranges.

Although the dictionary feature discussed in the previous section activates only for apostrophes, it does not activate on *all* apostrophes. This can be seen in Figures 17 and 18 in Appendix E.2, showing two other apostrophe-activating dictionary features, but for different contexts (such as "[I/We/They]'ll" and "[don/won/wouldn]'t"). Details for how we searched and selected for dictionary features can be found in Appendix E.3.

6.2 Output: Dictionary Features have Intuitive Effects on the Logits

In addition to looking at which tokens activate the dictionary feature, we investigate how dictionary features affect the model's output predictions for the next token by ablating the feature from the residual stream[5]. If our dictionary feature is interpretable, subtracting its value from the residual stream should have a logical effect on the predictions of the next token. We see in Figure 6 (Right) that the effect of removing the apostrophe feature mainly reduces the logit for the following "s". This matches what one would expect from a dictionary feature that detects apostrophes and is used by the model to predict the "s" token that would appear immediately after the apostrophe in possessives and contractions like "let's".

6.3 Intermediate Features: Dictionary Features Allow Automatic Circuit Detection

We can also understand dictionary features in relation to the upstream and downstream dictionary features: given a dictionary feature, which dictionary features in previous layers cause it to activate, and which dictionary features in later layers does it cause to activate?

To automatically detect the relevant dictionary features, we choose a target dictionary feature such as layer 5's feature for tokens in parentheses which predicts a closing parentheses (Figure 7). For this target dictionary feature, we find its maximum activation M across our dataset, then sample 20 contexts that cause the target feature to activate in the range $[M/2, M]$. For each dictionary feature in the previous layer, we rerun the model while ablating this feature and sort the previous-layer features by how much their ablation decreased the target feature. If desired, we can then recursively apply this technique to the dictionary features in the previous layer with a large impact. The results of this process form a causal tree, such as Figure 7.

Being the last layer, layer 5's role is to output directions that directly correspond to tokens in the unembedding matrix. In fact, when we unembed feature 5_{2027}, the top-tokens are all closing parentheses variations. Intuitively, previous layers will detect all situations that precede closing parentheses, such as dates, acronyms, and phrases.

[5]Specifically we use less-than-rank-one ablation.

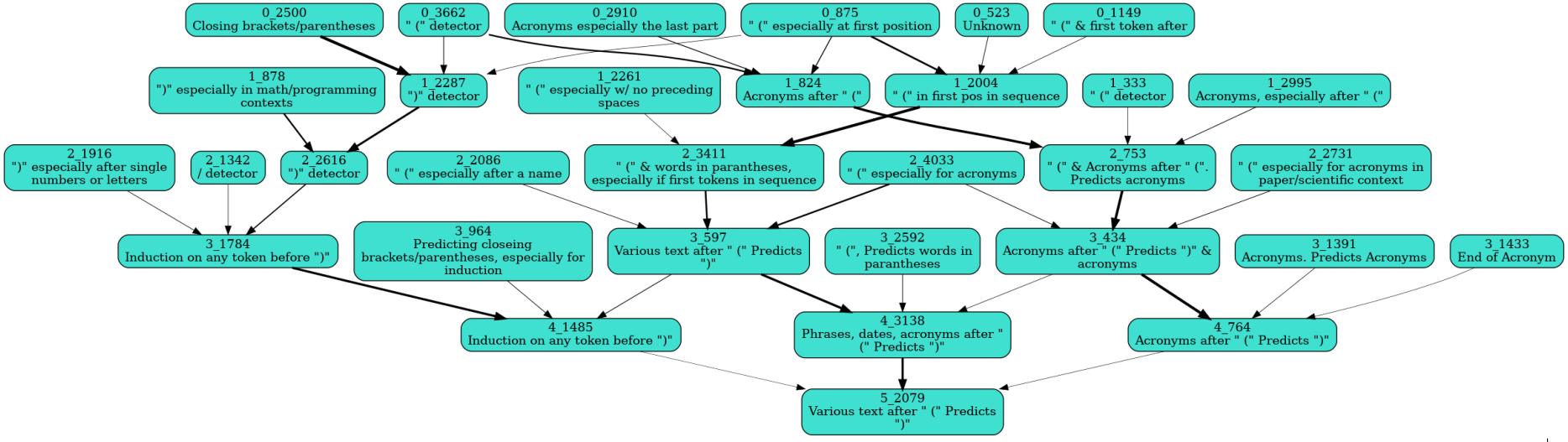

Figure 7: Circuit for the closing parenthesis dictionary feature, with human interpretations of each feature shown. Edge thickness indicates the strength of the causal effect between dictionary features, as measured by ablations. Many dictionary features across layers correspond to similar real-world features and often point in similar directions in activation space, as measured by cosine similarity. For instance, dictionary features 0_3662 and 1_333 both detect "(" and have a cosine similarity of 0.8, even though the diagram does not show them having a connection. This is an artifact of our automatic circuit-detection algorithm only back-chaining from the most relevant dictionary features found.

7 DISCUSSION

7.1 LIMITATIONS AND FUTURE WORK

Ultimately, we would like to find all of the network features used by the language model. At the moment, our dictionary features appear interpretable and causally important; however, we do not achieve $\sim$0 reconstruction loss, indicating that we have not fully captured the model's internal representations. We can also confirm this by measuring the perplexity of the model's predictions when a layer is substituted with its reconstruction. For instance, replacing layer 2's activations with our own increases the perplexity from 25 to 40. To this end, we would like to explore various other sparse autoencoder architectures and train on KL-divergence in place of reconstruction loss. Future efforts could also try to improve feature dictionary discovery by incorporating the weights of the model or dictionary features found in adjacent layers into the training process. Additionally, we were not able to learn very interpretable dictionaries for the middle of MLP activations (after the nonlinearity) and would like to pursue various methods to correct for this.

Our current methods for training sparse autoencoders are best suited to the residual stream, though there is evidence that they may be applicable to the MLPs. We would be excited about future work investigating what changes can be made to better understand the computations performed by the attention heads and MLP layers, each of which poses different challenges.

We would like to expand our automatic circuit-detection algorithm to include dictionaries trained on the outputs of the MLP and attention units to locate the specific computations of dictionary features. With this addition, and hopefully improved dictionaries in general, we would like to explore circuits found using learned dictionary features to study adversarial attacks, LLMs playing Othello/Chess, in-context learning, and preference/reward models.

We would like to study how network features develop across training by looking at feature formation across model checkpoints. We hope this will inform theories of deep learning by having non-toy, concrete examples of network feature formation.

7.2 CONCLUSION

Sparse autoencoders are a scalable, unsupervised approach to disentangling language model network features from superposition. We have demonstrated that the dictionary features they learn are more interpretable by autointerpretability, are better for performing precise model steering, and are more monosemantic than comparable methods.

The ability to find these dictionary features gives us a new, fully unsupervised tool to investigate model behaviour, allows us to make targeted edits, and can be trained using a manageable amount of computing power.

An ambitious dream in the field of interpretability is enumerative safety (Elhage et al., 2022): the ability to enumerate all features used by a network and thus understand the full set of computations that a model implements. If this were achieved, it could permit stronger guarantees that a model either is not able or is not inclined to perform certain dangerous actions, such as deception or advanced bioengineering. While this remains an ambition for now, dictionary learning hopefully marks a small step towards making it possible.

In summary, sparse autoencoders bring a new tool to the interpretability and editing of language models, which we hope others can build upon. The potential for innovations and applications is vast, and we're excited to see what happens next.

ACKNOWLEDGMENTS

We would like to thank the OpenAI Researcher Access Program for their grant of model credits for the autointerpretation and CoreWeave for providing EleutherAI with the computing resources for this project. We also thank Nora Belrose, Arthur Conmy, Jake Mendel, and the OpenAI Automated Interpretability Team (Jeff Wu, William Saunders, Steven Bills, Henk Tillman, and Daniel Mossing) for valuable discussions regarding the design of various experiments. We thank Wes Gurnee, Adam Jermyn, Stella Biderman, Leo Gao, Curtis Huebner, and William Saunders for their feedback on earlier versions of this paper. Thanks to Delta Hessler for proofreading. LR is supported by the

Long Term Future Fund. RH is supported by an Open Philanthropy grant. HC was greatly helped by the MATS program, funded by AI Safety Support.

REFERENCES

Gender by Name. UCI Machine Learning Repository, 2020. DOI: https://doi.org/10.24432/C55G7X.

Nora Belrose, David Schneider-Joseph, Shauli Ravfogel, Ryan Cotterell, Edward Raff, and Stella Biderman. Leace: Perfect linear concept erasure in closed form. *arXiv preprint arXiv:2306.03819*, 2023.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain neurons in language models. *URL https://openaipublic. blob. core. windows. net/neuron-explainer/paper/index. html.(Date accessed: 14.05. 2023)*, 2023.

Nick Cammarata, Gabriel Goh, Shan Carter, Chelsea Voss, Ludwig Schubert, and Chris Olah. Curve circuits. *Distill*, 2021. doi: 10.23915/distill.00024.006. https://distill.pub/2020/circuits/curve-circuits.

Arthur Conmy, Augustine N Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. *arXiv preprint arXiv:2304.14997*, 2023.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1, 2021.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Nelson Elhage, Robert Lasenby, and Chris Olah. Privileged bases in the transformer residual stream, 2023. URL `https://transformer-circuits.pub/2023/privileged-basis/index.html`. Accessed: 2023-08-07.

Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biol. Cybern.*, 20 (3–4):121–136, sep 1975. ISSN 0340-1200. doi: 10.1007/BF00342633. URL `https://doi.org/10.1007/BF00342633`.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Dan Hendrycks, Mantas Mazeika, and Thomas Woodside. An overview of catastrophic ai risks. *arXiv preprint arXiv:2306.12001*, 2023.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19, 2006.

Richard Ngo, Lawrence Chan, and Sören Mindermann. The alignment problem from a deep learning perspective. *arXiv preprint arXiv:2209.00626*, 2022.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

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.

Qing Qu, Yuexiang Zhai, Xiao Li, Yuqian Zhang, and Zhihui Zhu. Analysis of the optimization landscapes for overcomplete representation learning. *arXiv preprint arXiv:1912.02427*, 2019.

Shauli Ravfogel, Michael Twiton, Yoav Goldberg, and Ryan D Cotterell. Linear adversarial concept erasure. In *International Conference on Machine Learning*, pp. 18400–18421. PMLR, 2022.

Lee Sharkey, Dan Braun, and Beren Millidge. Taking features out of superposition with sparse autoencoders, 2023. URL `https://www.alignmentforum.org/posts/z6QQJbtpkEAX3Aojj/interim-research-report-taking-features-out-of-superposition`. Accessed: 2023-05-10.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

John Wright and Yi Ma. *High-dimensional data analysis with low-dimensional models: Principles, computation, and applications*. Cambridge University Press, 2022.

Zeyu Yun, Yubei Chen, Bruno A Olshausen, and Yann LeCun. Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors. *arXiv preprint arXiv:2103.15949*, 2021.

A AUTOINTERPRETATION PROTOCOL

The autointerpretability process consists of five steps and yields both an interpretation and an autointerpretability score:

1. On each of the first 50,000 lines of OpenWebText, take a 64-token sentence-fragment, and measure the feature's activation on each token of this fragment. Feature activations are rescaled to integer values between 0 and 10.
2. Take the 20 fragments with the top activation scores and pass 5 of these to GPT-4, along with the rescaled per-token activations. Instruct GPT-4 to suggest an explanation for when the feature (or neuron) fires, resulting in an interpretation.
3. Use GPT-3.5[6] to simulate the feature across another 5 highly activating fragments and 5 randomly selected fragments (with non-zero variation) by asking it to provide the per-token activations.
4. Compute the correlation of the simulated activations and the actual activations. This correlation is the autointerpretability score of the feature. The texts chosen for scoring a feature can be random text fragments, fragments chosen for containing a particularly high activation of that feature, or an even mixture of the two. We use a mixture of the two unless otherwise noted, also called 'top-random' scoring.
5. If, amongst the 50,000 fragments, there are fewer than 20 which contain non-zero variation in activation, then the feature is skipped entirely.

Although the use of random fragments in Step 4 is ultimately preferable given a large enough sample size, the small sample sizes of a total of 640 tokens used for analysis mean that a random sample will likely not contain any highly activating examples for all but the most common features, making top-random scoring a desirable alternative.

[6]While the process described in Bills et al. (2023) uses GPT-4 for the simulation step, we use GPT-3.5. This is because the simulation protocol requires the model's logprobs for scoring, and OpenAI's public API for GPT-3.5 (but not GPT-4) supports returning logprobs.

B Sparse Autoencoder Training and Hyperparameter Selection

To train the sparse autoencoder described in Section 2, we use data from the Pile (Gao et al., 2020), a large, public webtext corpus. We run the model that we want to interpret over this text while caching and saving the activations at a particular layer. These activations then form a dataset, which we use to train the autoencoders. The autoencoders are trained with the Adam optimiser with a learning rate of 1e-3 and are trained on 10-50M activation vectors for 1-3 epochs, with larger dictionaries taking longer to converge.

When varying the hyperparameter α which controls the importance of the sparsity loss term, we consistently find a smooth tradeoff between the sparsity and accuracy of our autoencoder, as shown in Figure 8. The lack of a 'bump' or 'knee' in these plots provides some evidence that there is not a single correct way to decompose activation spaces into a sparse basis, though to confirm this would require many additional experiments. Figure B shows the convergence behaviour of a set of models with varying α over multiple epochs.

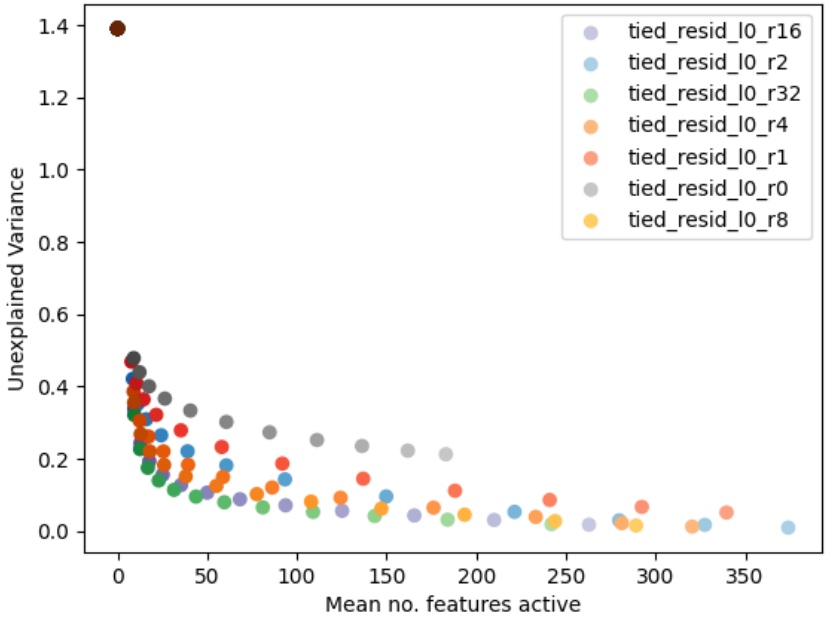

Figure 8: The tradeoff between the average number of features active and the proportion of variance that is unexplained for the MLP at layer 0.

C Further Autointerpretation Results

C.1 Interpretability is Consistent across Dictionary Sizes

We find that larger interpretability scores of our feature dictionaries are not limited to overcomplete dictionaries (where the ratio, R, of dictionary features to model dimensions is > 1), but occurs even in dictionaries that are smaller than the underlying basis, as shown in Figure 10. These small dictionaries are able to reconstruct the activation vectors less accurately, so with each feature being similarly interpretable, the larger dictionaries will be able to explain more of the overall variance.

C.2 High Interpretability Scores Are Not an Artefact of Top Scoring

A possible concern is that the autointerpretability method described in Section 3 combines top activating fragments (which are usually large) with random activations (which are usually small), mak-

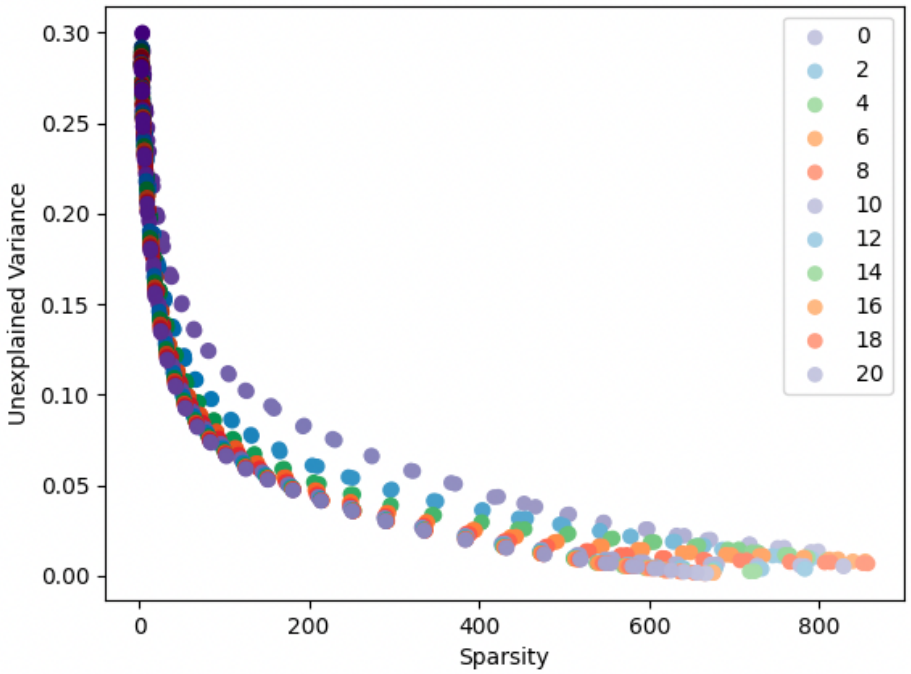

Figure 9: The tradeoff between sparsity and unexplained variance in our reconstruction. Each series of points is a sweep of the α hyperparameter, trained for the number of epochs given in the legend.

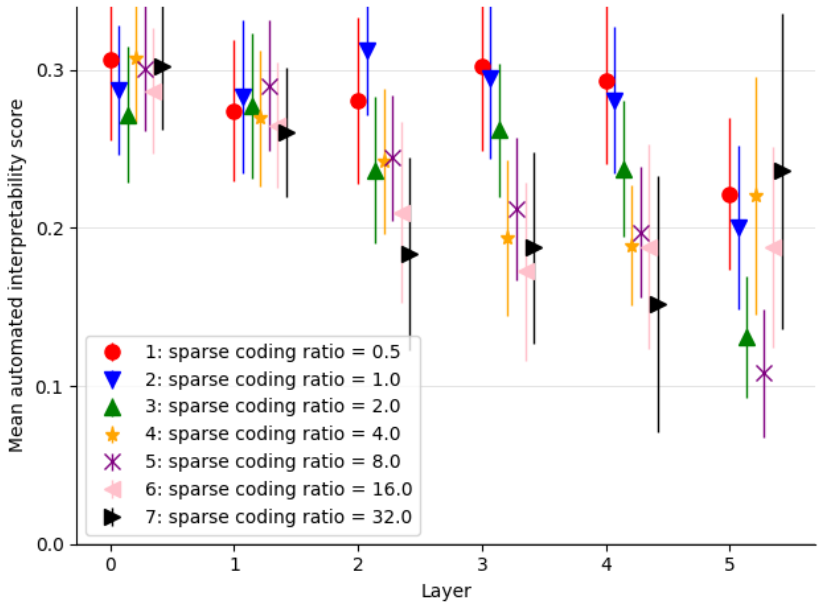

Figure 10: Comparison of average interpretability scores across dictionary sizes. All dictionaries were trained on 20M activation vectors obtained by running Pythia-70M over the Pile with $\alpha = .00086$.

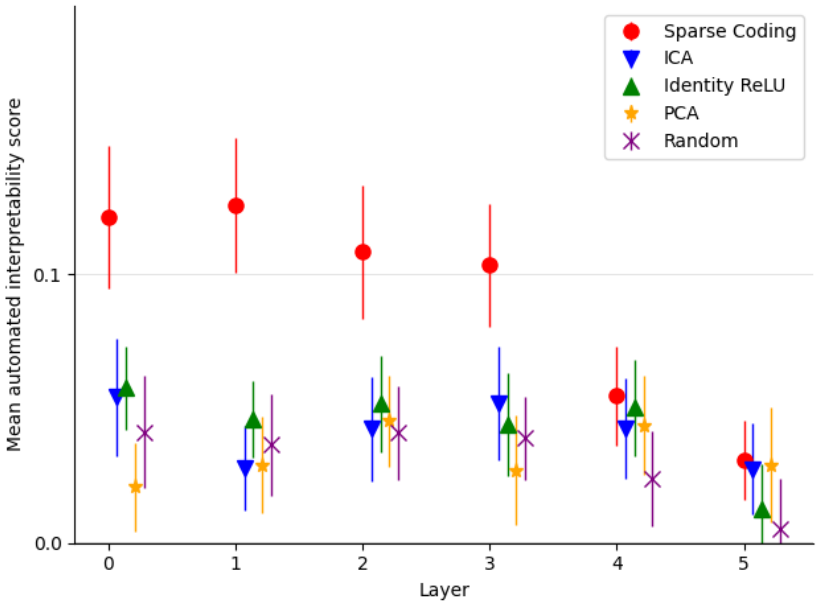

Figure 11: Random-only interpretability scores across each layer, a measure of how well the interpretation of the top activating cluster is able to explain the entire range of activations.

ing it relatively easy to identify activations. Following the lead of Bills et al. (2023), we control for this by recomputing the autointerpretation scores by modifying Step 3 using only randomly selected fragments. With large sample sizes, using random fragments should be the true test of our ability to interpret a potential feature. However, the features we are considering are heavy-tailed, so with limited sample sizes, we should expect random samples to underestimate the true correlation.

In Figure 11 we show autointerpretability scores for fragments using only random fragments. Matching Bills et al. (2023), we find that random-only scores are significantly smaller than top-and-random scores, but also that our learned features still consistently outperform the baselines, especially in the early layers. Since our learned features are more sparse than the baselines and thus, activate less on a given fragment, this is likely to underestimate the performance of sparse coding relative to baselines.

An additional potential concern is that the structure of the autoencoders allows them to be sensitive to less than a full direction in the activation space, resulting in an unfair comparison. We show in Appendix G that this is not the source of the improved performance of sparse coding.

While the residual stream can usually be treated as a vector space with no privileged basis (a basis in which we would expect changes to be unusually meaningful, such as the standard basis after a nonlinearity in an MLP), it has been noted that there is a tendency for transformers to store information in the residual stream basis(Dettmers et al., 2022), which is believed to be caused by the Adam optimiser saving gradients with finite precision in the residual basis(Elhage et al., 2023). We do not find residual stream basis directions to be any more interpretable than random directions.

C.3 Interpreting the MLP Sublayer

Our approach of learning a feature dictionary and interpreting the resulting features can, in principle, be applied to any set of internal activations of a language model, not just the residual stream. Applying our approach to the MLP sublayer of a transformer resulted in mixed success. Our approach still finds many features that are more interpretable than the neurons. However, our architecture also learns many dead features, which never activate across the entire corpus. In some cases, there are so many dead features that the set of living features does not form an overcomplete basis. For example, in a dictionary with twice as many features as neurons, less than half might be active enough to

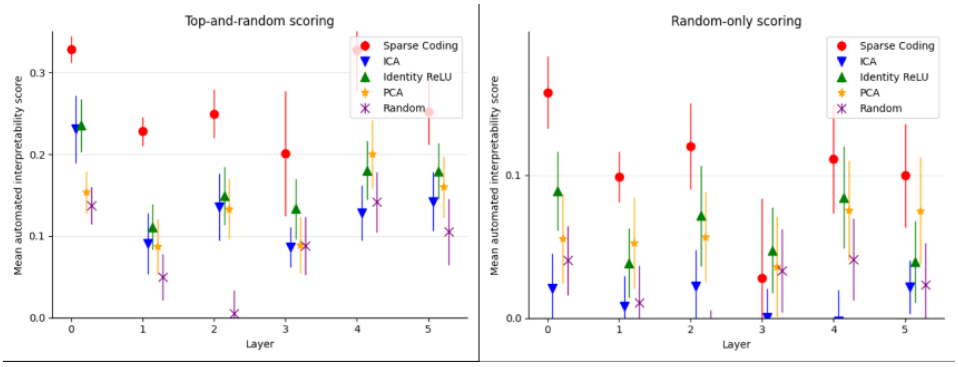

Figure 12: Top-and-random and random-only interpretability scores for across each MLP layer, using an ℓ^1 coefficient $\alpha = 3.2e-4$ and dictionary size ratio $R = 1$.

perform automatic interpretability. The exceptions to this are the early layers, where a large fraction of them are active.

For learning features in MLP layers, we find that we retain a larger number of features if we use a different matrix for the encoder and decoder, so that Equations 1 and 2 become

$$\mathbf{c} = ReLU(M_e\mathbf{x} + \mathbf{b}) \quad (4)$$

$$\hat{\mathbf{x}} = M_d^T\mathbf{c} \quad (5)$$

We are currently working on methods to overcome this and find truly overcomplete bases in the middle and later MLP layers.

C.4 Interpretability Scores Correlate with Kurtosis and Skew of Activation

It has been shown that the search for sparse, overcomplete dictionaries can be reformulated in terms of the search for directions that maximise the ℓ^4-norm (Qu et al., 2019).

We offer a test of the utility of this by analysing the correlation between interpretability and a number of properties of learned directions. We find that there is a correlation of 0.19 and 0.24 between the degree of positive skew and kurtosis respectively that feature activations have and their top-and-random interpretability scores, as shown in Table 2.

Moment	Correlation with top-random interpretability score
Mean	-0.09
Variance	0.02
Skew	0.20
Kurtosis	0.15

Table 2: Correlation of interpretability score with feature moments across residual stream results, all layers, with dictionary size ratios $R \in \{0.5, 1, 2, 4, 8\}$.

This also accords with the intuitive explanation that the degree of interference due to other active features will be roughly normally distributed by the central limit theorem. If this is the case, then features will be notable for their heavy-tailedness.

This also explains why Independent Component Analysis (ICA), which maximises the non-Gaussianity of the found components, is the best performing of the alternatives that we considered.

D Measuring the Disruptiveness of our Concept Erasure

We measure the disruption to model behaviour by calculating the KL-divergence of the ablated model from the base model on the first 10,000 sequences from the Pile dataset (Gao et al., 2020).

We find that erasing the relevant dictionary features causes significantly less KL-divergence than difference-in-means projection or LEACE (Figure 13).

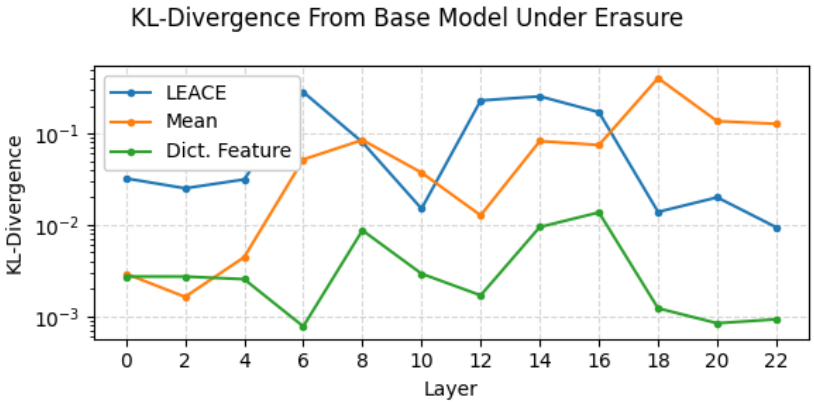

Figure 13: KL-divergence of the ablated model from the base model on the Pile, versus the layer the concept erasure intervention is performed at. The last datapoint for the difference-in-means edit has been ommitted as it is multiple orders of magnitude larger than all other datapoints.

E Qualitative Feature Analysis

E.1 Residual Stream Basis

Figure 14 gives a token activation histogram of the residual stream basis. Connecting this residual stream dimension to the apostrophe feature from Figure 6, this residual dimension was the 10th highest dimension read from the residual stream by our feature[7].

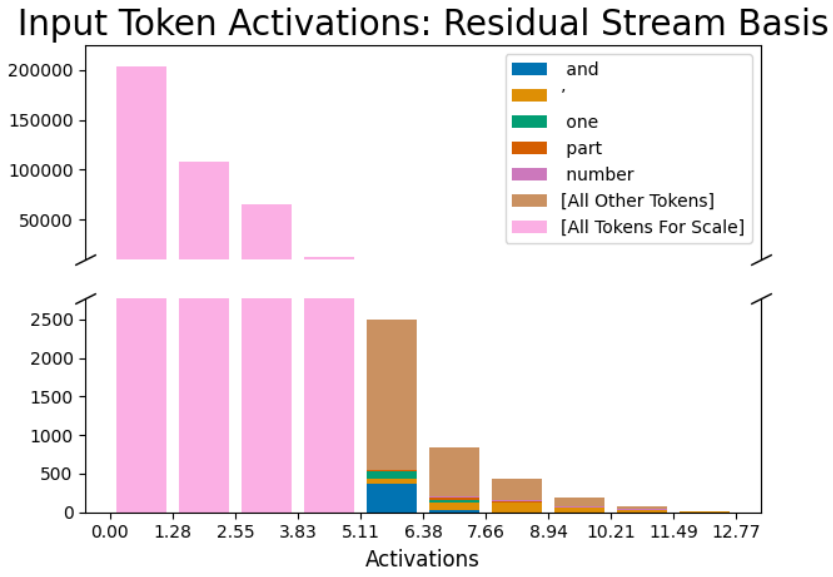

Figure 14: Histogram of token counts in the neuron basis. Although there are a large fraction of apostrophes in the upper activation range, this only explains a very small fraction of the variance for middle-to-lower activation ranges.

[7]The first 9 did not have apostrophes in their top-activations like dimension 21.

E.2 Examples of Learned Features

Other features are shown in Figures 15, 16, 17, and 18.

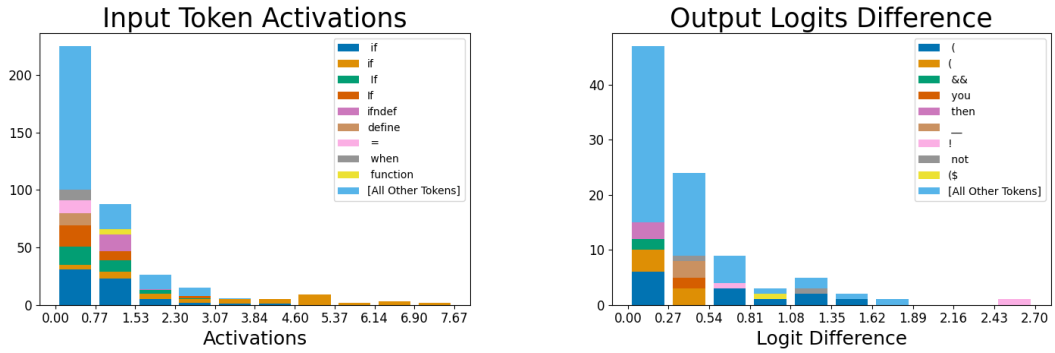

Figure 15: 'If' feature in coding contexts

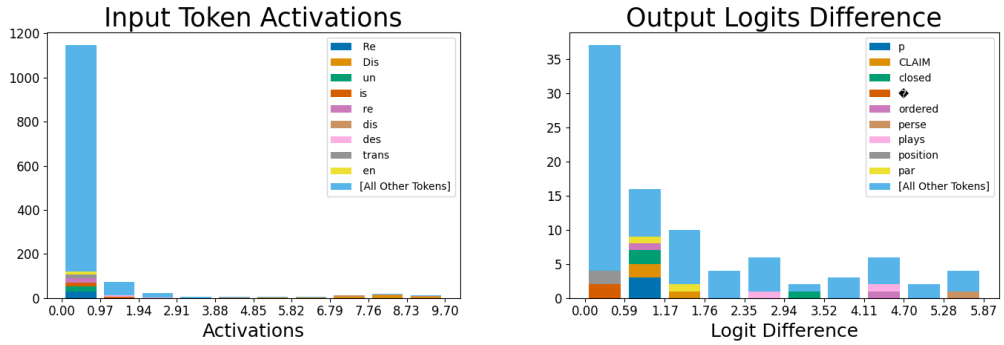

Figure 16: 'Dis' token-level feature showing bigrams, such as 'disCLAIM', 'disclosed', 'disordered', etc.

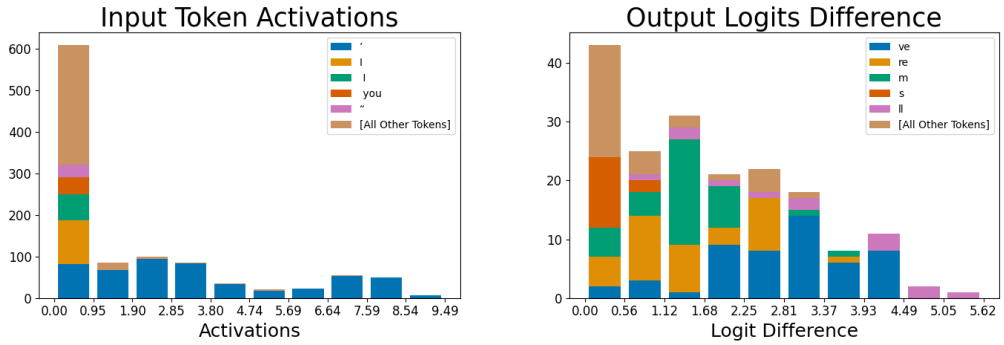

Figure 17: Apostrophe feature in "I'll"-like contexts.

E.3 Feature Search Details

We searched for the apostrophe feature using the sentence " I don't know about that. It is now up to Dave"', and seeing which feature (or residual stream dimension) activates the most for the last apostrophe token. The top activating feature in our dictionary was an outlier dimension feature (i.e., a feature direction that mainly reads from an outlier dimension of the residual stream), the apostrophes

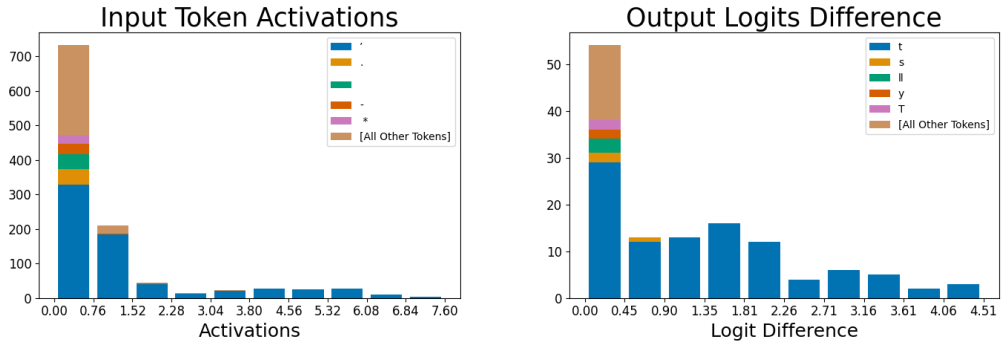

Figure 18: Apostrophe feature in "don't"-like contexts.

after O (and predicted O'Brien, O'Donnell, O'Connor, O'clock, etc), then the apostrophe-preceding-s feature.

For the residual basis dimension, we searched for max and min activating dimensions (since the residual stream can be both positive and negative), where the top two most positive dimensions were outlier dimensions, the top two negative dimensions were our displayed one and another outlier dimension, respectively.

E.4 Failed Interpretability Methods

We attempted a weight-based method going from the dictionary in layer 4 to the dictionary in layer 5 by multiplying a feature by the MLP and checking the cosine similarity with features in layer 5. There were no meaningful connections. Additionally, it's unclear how to apply this to the Attention sublayer since we'd need to see which position dimension the feature is in. We expected this failed by going out of distribution.

F Number of Active Features

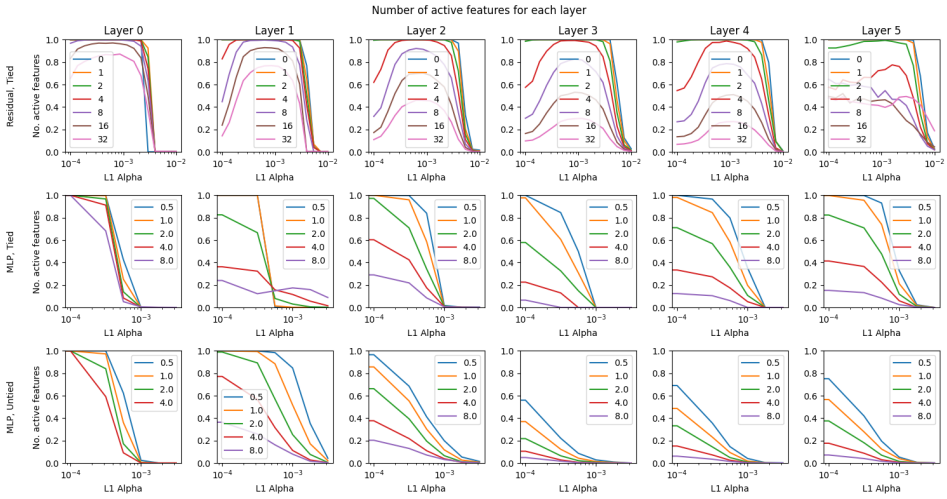

Figure 19: The number of features that are active, defined as activating more than 10 times across 10M datapoints, changes with sparsity hyperparamter α and dictionary size ratio R.

In Figure 19 we see that, for residual streams, we consistently learn dictionaries that are at least 4x overcomplete before some features start to drop out completely, with the correct hyperparameters. For MLP layers you see large numbers of dead features even with hyperparameter $\alpha = 0$. These

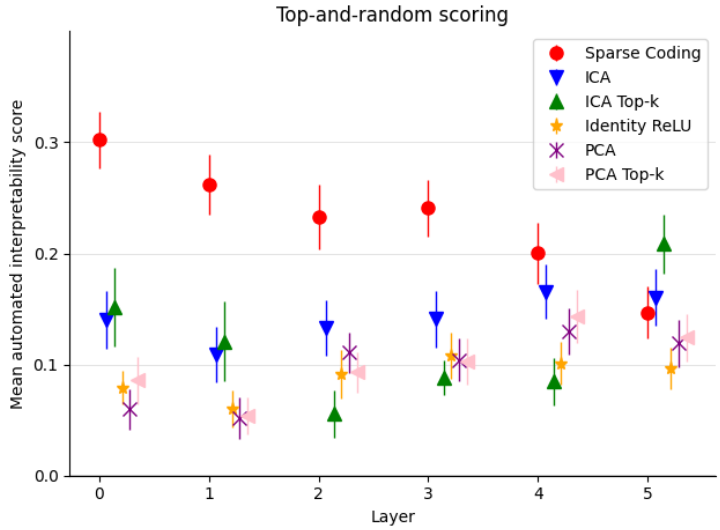

Figure 20: Autointerpretation scores across layers for the residual stream, including top-K baselines for ICA and PCA.

figures informed the selection of $\alpha = 8.6e-4$ and $\alpha = 3.2e-4$ that went into the graphs in Section 3 for the residual stream and MLP respectively. Due to the large part of the input space that is never used due to the non-linearity, it is much easier for MLP dictionary features to become stuck at a position where they hardly ever activate. In future we plan to reinitialise such 'dead features' to ensure that we learn as many useful dictionary features as possible.

G TOP K COMPARISONS

As mentioned in Section 3, the comparison directions learnt by sparse coding and those in the baselines are not perfectly even. This is because, for example, a PCA direction is active to an entire half-space on one side of a hyperplane through the origin, whereas a sparse coding feature activates on less than a full direction, being only on the far side of a hyperplane that does not intersect the origin. This is due to the bias applied before the activation, which is, in practice, always negative. To test whether this difference is responsible for the higher scores, we run a variant of PCA and ICA in which we have a fixed number of directions, K, which can be active for any single datapoint. We set this K to be equal to the average number of active features for a sparse coding dictionary with ratio $R = 1$ and $\alpha = 8.6e-4$ trained on the layer in question. We compare the results in Figure 20, showing that this change does not explain more than a small fraction of the improvement in scores.