An Exploration of Sparse Autoencoders and Random Walks in Latent Space

Farhan Baig FARHAN.BAIG@YALE.EDU
Department of Computer Science
Yale University
New Haven, CT 06511, USA

Nathan Mak NATHAN.MAK@YALE.EDU
Department of Statistics and Data Science
Yale University
New Haven, CT 06511, USA

Abstract

The following paper will begin with a brief introduction to machine interpretability at a large scale then dive into a subfield of said research concerning Sparse Autoencoder Models (SAEs). The general principle of such models is to extract features from learned activations in a model. We will explore exactly how such models work in a simple manner to enhance comprehension. Our paper will mainly revolve around a few notable articles in the field pioneered by Anthropic, and then we will consider a case study on a smaller model. To this model, we have ourselves added code for the exploration of specific topics in feature manipulation. Additionally, we have added a novel experiment regarding random walks in the latent space of SAE models, which we believe gives some insight and intuition into how LLMs construct concepts. Lastly, we hope to tie the ideas of SAE models back into the larger research area of machine interpretability and examine related papers and concepts such as Rank One Model Editing (a technique which introduces perturbations to a model's embeddings and observes resulting outputs to discover causality) to further uncover the black box of large language models.

Keywords: Sparse Autoencoders, features, LLMs, Rank One Model Editing

1 Introduction

Machine interpretability is the field of ML research concerned with how computers think. Over the last few years, we have seen a boom in large language models and associated models, which have given rise to a massively expanded creative landscape for the average person. However, the aforementioned average person and machine learning researchers alike are not exactly sure how they work. We know that some combination of data and mathematics result in a meaningful output, but how? Not only is this field of research driven by inherent curiosity about how models work, but also by things such as safety concerns. Researchers want to ensure that models have safe outputs and are not being biased or hurtful (Bricken et al. (2023)). The question of understanding models and tweaking their outputs is incredibly relevant as models get progressively larger and more complex.

Researchers have come up with a multitude of ways to explore machine interpretability; however, for the purposes of our paper, we will mainly focus on Sparse Autoencoder Models

(SAEs). These models have two separate parts: the encoder and decoder. The encoder is trained on other models' activations–an LLM for the purposes of this paper–and gives us learned features. Features can be imagined as vectors or directions in a model's activation space which encode some information. A good example of this is to suppose that two vectors encode the idea for "king" and "queen" respectively. Then, the difference between the said vectors could theoretically encode some sort of gender direction, and if we look at the word "brother" in another part of our activation space, we could somewhat accurately determine where the word "sister" would lie.

Thus, SAE models are based upon the idea of dictionary learning which states that concepts are encoded more or less discretely within LLMs. We can construct any given idea by "looking up" associated concepts in a constructed dictionary of sorts. Essentially, this is the same as directional encoding as is described above, which can be more formally stated as the linear representation hypothesis. Dictionary learning seems to be true to a significant extent in various empirical studies (Bricken et al. (2023)), and, as such, it is a fairly safe basis on which to begin our understanding of LLMs. This is not to say that this is certainly how LLMs think, but in order for SAEs to make any sense at all, we rely on this idea as a crutch.

The reason SAE models are concerned with higher dimensional features rather than individual neurons is that, oftentimes in models, we find that neurons are polysemantic (seem to encode multiple ideas). This relates to the superposition hypothesis–the idea that neural networks project to a higher dimensional space representing a larger "disentangled" network (Bricken et al. (2023)). Thus, we yield more features than neurons, which means that neurons themselves are not necessarily the best basis on which to understand a model's knowledge. This is an issue with features as well, but to a lesser extent assuming that the SAE is trained properly (such features are known as dense features which we will talk about more later on) (Bloom (2024)).

Before discussing how SAEs work mathematically under the hood, we should first qualify what makes an SAE "good". Ideally, we want an SAE which gives us sparse features. This means that features are separate enough from other features to strongly represent a concept. If our model does not give us a sufficiently sparse feature space, then we are left with lots of dead or dense features (Bloom (2024)). Dead features are, as their name implies, features which represent little to no meaning. Such features slow down our training process and result in meaningless output. Too many dead features implies that we are over penalizing with L1 loss (Bloom (2024)). Dense features are features which are activated on many inputs, thus encoding many things and thus encoding nothing specific. Again, these features are detrimental and do not allow us to truly understand a model's associations. Ideally, we want to minimize the number of dead and dense features.

One interesting way that researchers have solved the issue of too many dead features is ghost gradients. Ghost gradients work by applying a constant loss to the result of a loss function and shifting dead features towards the direction that the SAE is not fitting (the autoencoder residual). This is because this space is unoccupied and we are likely to yield some use of the dead neurons this way (Bloom (2024)). The actual process of calculating the loss term is rather complicated and is outlined in the following Anthropic paper regarding ghost gradients (Templeton et al. (2024)). Despite enhancing compute cost due to the process of calculating the loss term, the trade off in terms of the number of dead features is almost certainly worth it in smaller models (Templeton et al. (2024)). The second part of an SAE model is called the decoder. This portion takes in our learned features and outputs activations which are meant to closely match our original activations passed in to the encoder. The model thus has two loss functions for each step, which will be outlined in the following section.

2 Important Math

In neural networks, activation functions determine if a neuron should be activated or not, and they add non-linearity to the model. Nonlinearity allows the network to learn complex, non-linear, patterns. One of the most common and simple activation functions is the Rectified Linear Unit (ReLU), and it is defined as the following:

$$\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x}), \text{ where } \mathbf{x} \text{ is the input to a neuron}$$

Its linear component has the advantage of making the neural network easier to optimize, and the gradient for learning does not disappear/go to zero (vanishing effect) as with other activation functions (Glorot et al. (2011)). Also, it is computationally easy since it is just a maximum function. The relevance for SAEs is that ReLU can output zero, which encourages sparsity. Some potential issues are the floor at zero, which means a neuron might not be able to learn if there are many negative inputs, and the unbounded activation which might increase the weight so much that it never gets activated or learned again (Glorot et al. (2011)).

A similar activation function, the Gaussian Error Linear Unit (GELU), was used in Neel Nanda's gelu-1l transformer model, which trained the SAE model. GELU is defined as the following:

$$\text{GELU}(x) = x\,\Phi(x) \quad \text{where} \quad \Phi(x) \text{ is the CDF of the standard normal distribution}$$

Some important differences between ReLU and GELU is that GELU is curved and can output negative values (Glorot et al. (2011)). The negative values possibility could solve the potential ReLU learning issue from having a strict floor at zero, and the curvature could allow for better modeling of complex patterns. However, it is more computationally intensive than ReLU.

Loss functions measure how well a model performs by measuring the distance between the model's prediction and the actual value in a dataset. Anthropic's SAE's loss function combines an "L2 penalty on the reconstruction loss and an L1 penalty on feature activations" (Templeton et al. (2024)).

The L1 penalty encourages sparsity since it pushes some activations/weights towards zero by minimizing the sum of the absolute values of the weights. The L2 penalty is used to minimize error in the reconstruction.

Another norm that is used to encourage sparsity but is not used in Anthropic's loss function is the L0 norm which minimizes the number of non-zero weights (Louizos et al. (2018)). However, it is more difficult to optimize this norm since it is not differentiable.

Since SAE training is computationally intensive, it should be optimized for the best quality of results within a computational budget constraint. This means optimizing hyperparameters which are set before training. Anthropic applies the “scaling law” to find the optimal number of features being learned and the number of steps used to train the SAE since these two seem to have greater compute costs compared to other hyperparameters (Kaplan et al. (2020)). They did this by altering/sweeping through potential hyperparameters for the number of steps and features while keeping the other ones fixed, minimizing the loss given a compute budget, and then choosing the numbers that suited their purpose best.

Bias is the “systematic error from erroneous assumptions in the learning algorithm,” and it can lead to inaccurate predictions by the model (Mikołajczyk-Bareła and Grochowski (2023)). The Anthropic equation for $\hat{\mathbf{x}}$ includes this bias in the equation to account for it if there is any. While the goal is to eliminate any bias, this can be difficult due to the many potential sources like data collection and model choice.

3 Feature Analysis

The natural next step upon arriving at a working SAE model is feature exploration. We have defined features earlier, but note again that they are more or less directions within a model’s activation space which correspond to some concept. We can somewhat quantify a feature’s association to a concept through two constructs: feature specificity and feature sensitivity. Feature specificity is measured through the features likelihood of firing on some given input. For example, we can construct a likelihood score of a feature and a specific construct as follows: $log(P(s|concept)/P(s))$ where s is a string and $P(s)$ can be calculated using the individual probabilities of each constituent token $P(t)$) (Bricken et al. (2023)). Feature sensitivity is similar, indicating that features, despite being associated with some content, do not necessarily fire on all related ideas. The example given in the text is an Arabic script feature which fires on most Arabic script tokens but not on specific ones such as the article transliterated as ”al”. See Figure 1.

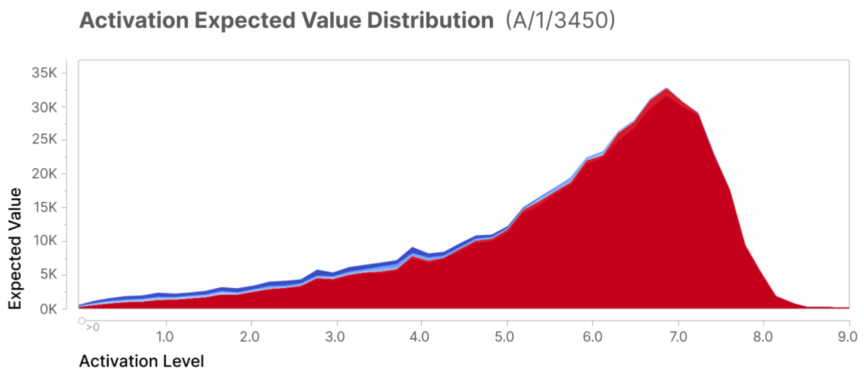


Figure 1: Bricken et al. (2023)

There is a lot that can be done with features, as is outlined in various papers regarding the topic; however, for the sake of this paper, we will mainly focus on a few interesting points:

feature neighborhoods, feature ablations, and feature splitting. We have built upon Neel Nanda's gelu-1L model findings to explore such phenomena. However, prior to discussing the code, it would be valuable to outline exactly what these terms refer to.

Feature grouping is rather straightforward. We observe that features are similar to other features which represent related information and cluster in feature neighborhoods. For example, features relating to San Francisco are grouped together, with the "Golden Gate Bridge" feature being slightly further from the "San Francisco" feature than say the "San Francisco 49ers" feature (Templeton et al. (2024)). See Figure 2.

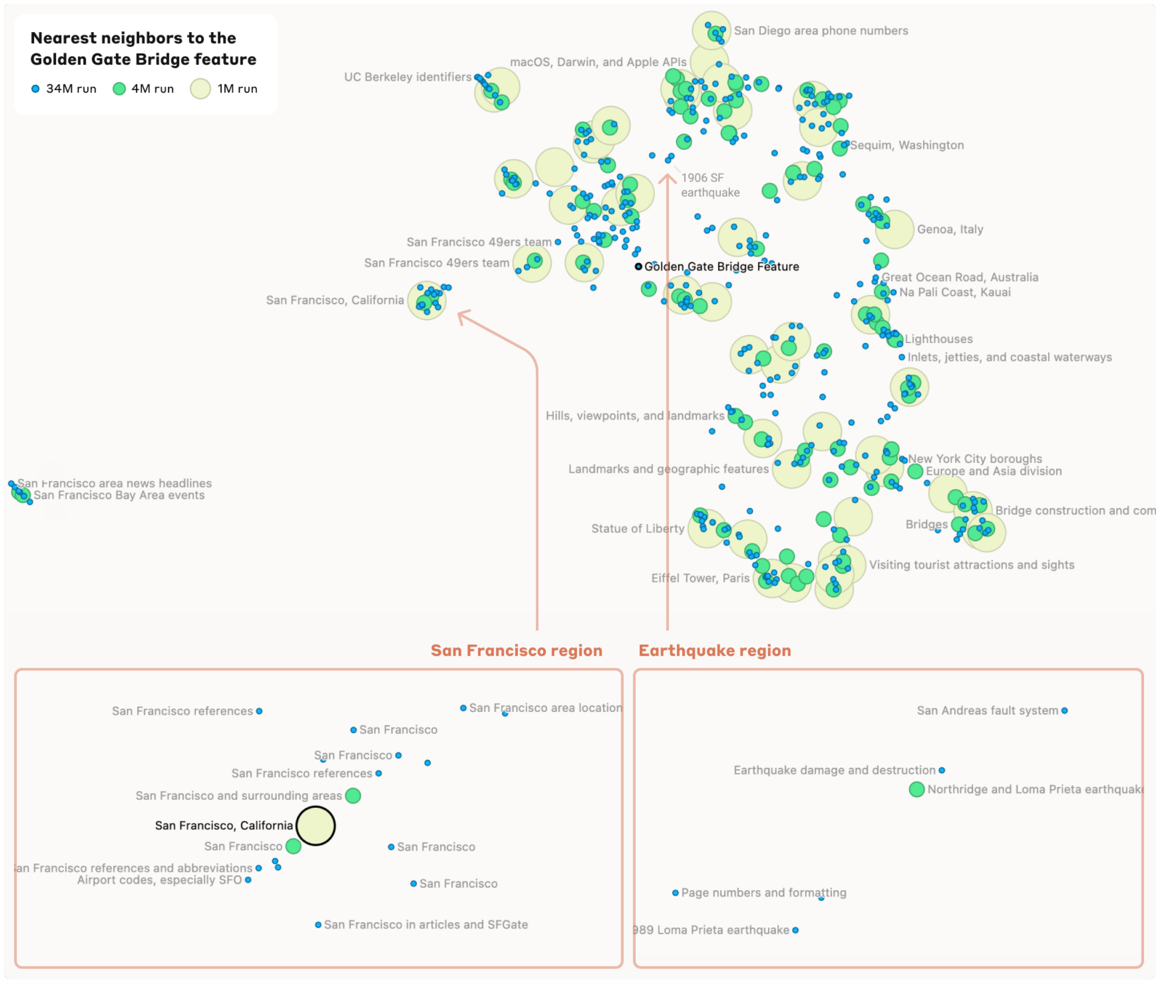
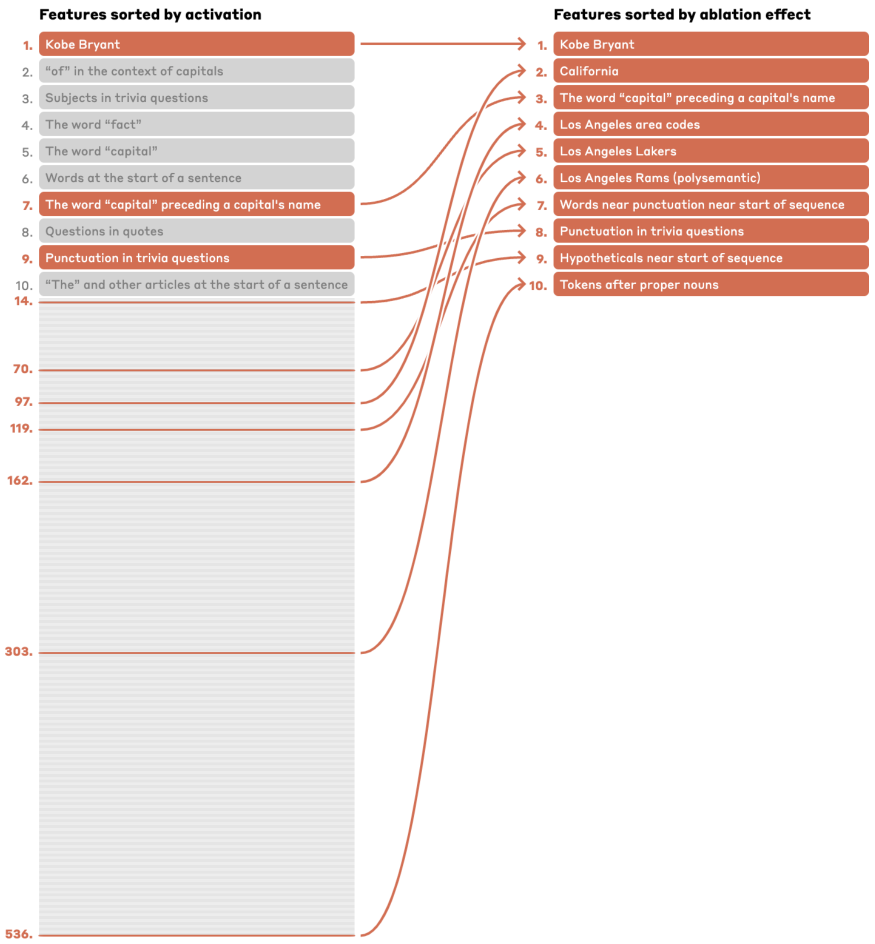


Figure 2: Templeton et al. (2024)

This pattern is promising as it tells us that our model is in fact learning associations to some degree. Feature splitting naturally follows from feature grouping. Feature splitting is the idea that in larger SAE models, features will break down into more specific features but still remain close together in grouping. In the earlier example, we see that in our smallest model, we have one "San Francisco" feature, but as the model grows, it splits into features like "San Francisco" and "San Francisco surrounding areas", and as the model grows again, the features split into even more specifics like “San Francisco references” and “Airport Codes” (Templeton et al. (2024)). See Figure 2.

Feature ablation is slightly different and is interesting because it allows for us to almost control a model’s output to some extent. To perform feature ablation, we fix a feature to zero in a specific token on the forward pass of the model and observe the resulting output (Templeton et al. (2024)). The idea here is that we are essentially telling a feature to not fire which changes the association that our model has and thus manipulates its output. An interesting experiment that can be conducted as a result of this, which was done by Anthropic researchers, is to measure how much individual features are important to a model’s output for some given input. So, given some input, we garner a list of the most highly activated features, and then we perform ablation on each of these. From this we garner a new list which measures which of the features, when missing, had the largest impact on the semantic meaning of the text. It was then observed that of the top 10 features by ablation effect, only 3 were within the top 10 on the basis of activation (Templeton et al. (2024)). See Figure 3.



Features sorted by activation
1. Kobe Bryant
2. “of” in the context of capitals
3. Subjects in trivia questions
4. The word “fact”
5. The word “capital”
6. Words at the start of a sentence
7. The word “capital” preceding a capital's name
8. Questions in quotes
9. Punctuation in trivia questions
10. “The” and other articles at the start of a sentence
14.
70.
97.
119.
162.
303.
536.
Features sorted by ablation effect
1. Kobe Bryant
2. California
3. The word “capital” preceding a capital's name
4. Los Angeles area codes
5. Los Angeles Lakers
6. Los Angeles Rams (polysemantic)
7. Words near punctuation near start of sequence
8. Punctuation in trivia questions
9. Hypotheticals near start of sequence
10. Tokens after proper nouns

Figure 3: Templeton et al. (2024)

Another useful concept to introduce, which is also discussed within the Anthropic paper, is feature completeness. Ideally, when we train an SAE, we want to know whether the features we extract cover a wide enough range of topics to properly describe varied ideas. For example, if we have some model which has a feature for Beijing, do we necessarily have features corresponding to other Chinese cities like Chongqing or Guangzhou? Researchers at Anthropic found that as the size of the SAE model increased, the breadth of feature coverage also increased. For example, the 1M parameter run had 14 less features for representing periodic table elements than the 4M parameter run (Templeton et al. (2024)). See Figure 4.

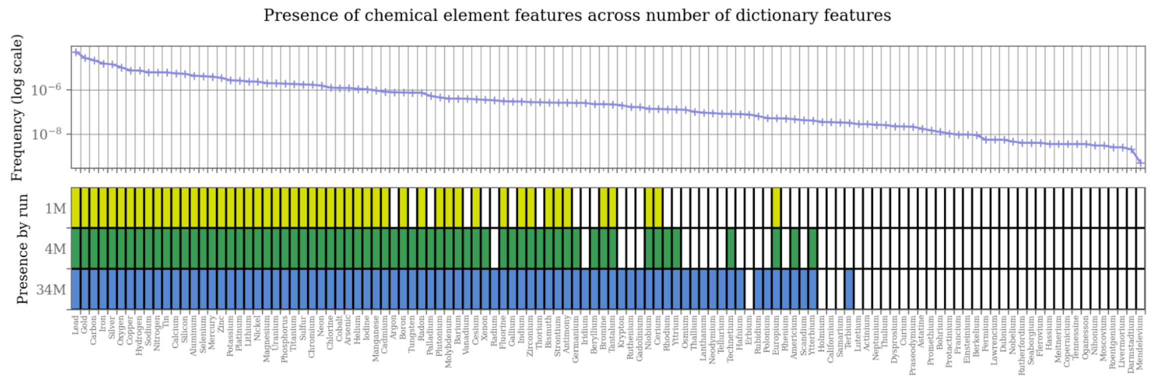


Figure 4: Templeton et al. (2024)

For the sake of our exploration, many of these concepts will be simplified mainly for two reasons. Firstly, we have not trained our own SAE model due to a lack of time; however, we are aiming to clean up Neel Nanda's training codebase. Secondly, experimenting with a single SAE and a smaller single layer model inherently leads to some simplification of concepts such as feature completeness. See Figure 4.

4 Exploration Setup

The SAE model we will be working with is set up as follows. It is built upon the `gelu-1L` model, which is a one-layer simple transformer (Nanda (2023)). The training process minimizes the summation of the L1 loss and L2 loss, defined as follows:

$$f_i(x) = \mathrm{ReLU}\left(\mathbf{W}^{\mathrm{enc}}_{i,\cdot} \cdot \mathbf{x} + b^{\mathrm{enc}}_i\right) \qquad \hat{\mathbf{x}} = \mathbf{b}^{\mathrm{dec}} + \sum_{i=1}^{F} f_i(\mathbf{x})\mathbf{W}^{\mathrm{dec}}_{\cdot,i}$$

$$\text{L1 Loss: } \lambda \sum_i f_i(\mathbf{x}) \cdot \left\| \mathbf{W}^{\mathrm{dec}}_{\cdot,i} \right\|_2 \qquad \text{L2 Loss: } \frac{1}{F} \sum_{i=1}^{F} \|x_i - \hat{x}_i\|_2^2$$

Our sparsity penalty for L1 loss is set as $l1_{\mathrm{coeff}}$. We initialize our encoder weights using the Kaiming Uniform Initialization and decoder weights with the unit L2-norm (Nanda (2023)).

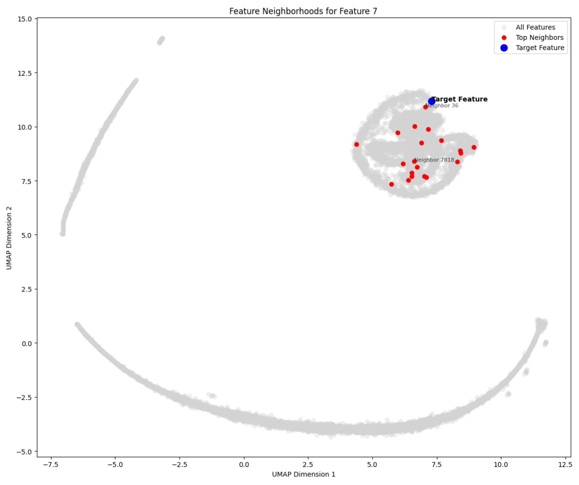


Figure 5: 2D Feature Neighborhoods

One notable step in our model is a gradient adjustment step, which occurs just after backpropagation. This can be modeled as:

$$W_{\text{dec}}^{\text{grad}} \leftarrow W_{\text{dec}}^{\text{grad}} - \text{Proj}_{W_{\text{dec}}}(W_{\text{dec}}^{\text{grad}})$$

This step ensures that the gradients of W_{dec} remain orthogonal to the weight vector (Nanda (2023)). We are running the model on a GPU configuration, and for the sake of our experiment, we are utilizing weights from `run1` of Nanda's training process.

5 Replication

We begin our replication with feature neighborhoods. We implemented two functions: one to visualize feature neighborhoods projected onto a 2D plane and one where the feature distribution was viewed in 3D space. The methodology was to accumulate a feature matrix, learned from the weight of our encoder model, then plot the features in both 2D and 3D. There is an argument to be made that the hidden layer features should be utilized for this visualization. However, for the sake of simplicity, we argue that the weights of the encoder are learned from our input activations and can theoretically represent features in a higher dimensional space (Nanda (2023)). This is actually an interesting question regarding the definition of features. But again, for the sake of our visualization, we will be using this as it was the simplest for us to code. Future research into the similarities between "weight features" (or some reclassification of this concept) and "hidden layer features" would be an interesting extension to the field.

Features seem to naturally group, but we can also see with metrics like cosine similarity, that the closest features to one another reside within the same part of latent space. This analysis can be seen in Figures 5 & 6.

We see fairly clearly that features seem to group even in lower dimensional representations. See Figure 5 & 6.

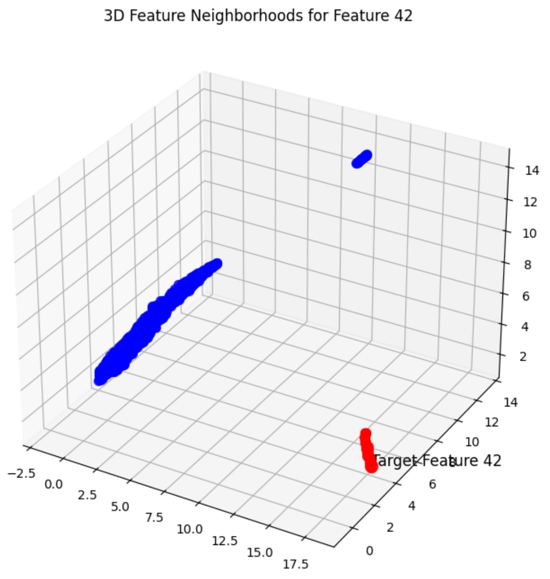


Figure 6: 3D Feature Neighborhoods

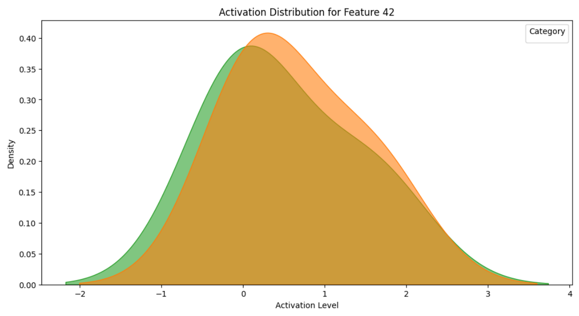
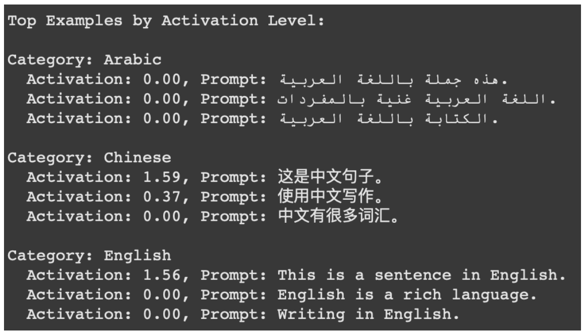


Figure 7: Feature Specificity Visualization

We then explored feature specificity. We plotted the relative activation of sentences in Chinese, Arabic, and English of a specific feature. We notice that the activation is similar for both the Chinese and English text, but zero for the equivalent sentence in Arabic. This indicates that feature 42 fires on a similar sentence structure in both Chinese and English and thus weakly encodes something in the sentence (all sentences are of the form "This is a sentence in ..." in their respective language). However, due to feature specificity, it does not necessarily fire on the same structure in Arabic. We can see this both visually in the plot as well as from the output which tells us the activation levels for each prompt. See Figure 7 & 8.

Lastly, we wanted to see the effects of feature ablation on a smaller 1 layer model. Our findings were rather tame, but they could probably be exaggerated by spending more time picking features that have larger ablation effects. To study feature ablation, we first passed our input data through the model to obtain our latent representation (think features here) and ran two forms of ablation: clamping and noise addition. In clamping, we set the feature



```
Top Examples by Activation Level:

Category: Arabic
  Activation: 0.00, Prompt: هذه جملة باللغة العربية.
  Activation: 0.00, Prompt: اللغة العربية غنية بالمفردات.
  Activation: 0.00, Prompt: الكتابة باللغة العربية.

Category: Chinese
  Activation: 1.59, Prompt: 这是中文句子。
  Activation: 0.37, Prompt: 使用中文写作。
  Activation: 0.00, Prompt: 中文有很多词汇。

Category: English
  Activation: 1.56, Prompt: This is a sentence in English.
  Activation: 0.00, Prompt: English is a rich language.
  Activation: 0.00, Prompt: Writing in English.
```

Figure 8: Feature Specificity Metrics

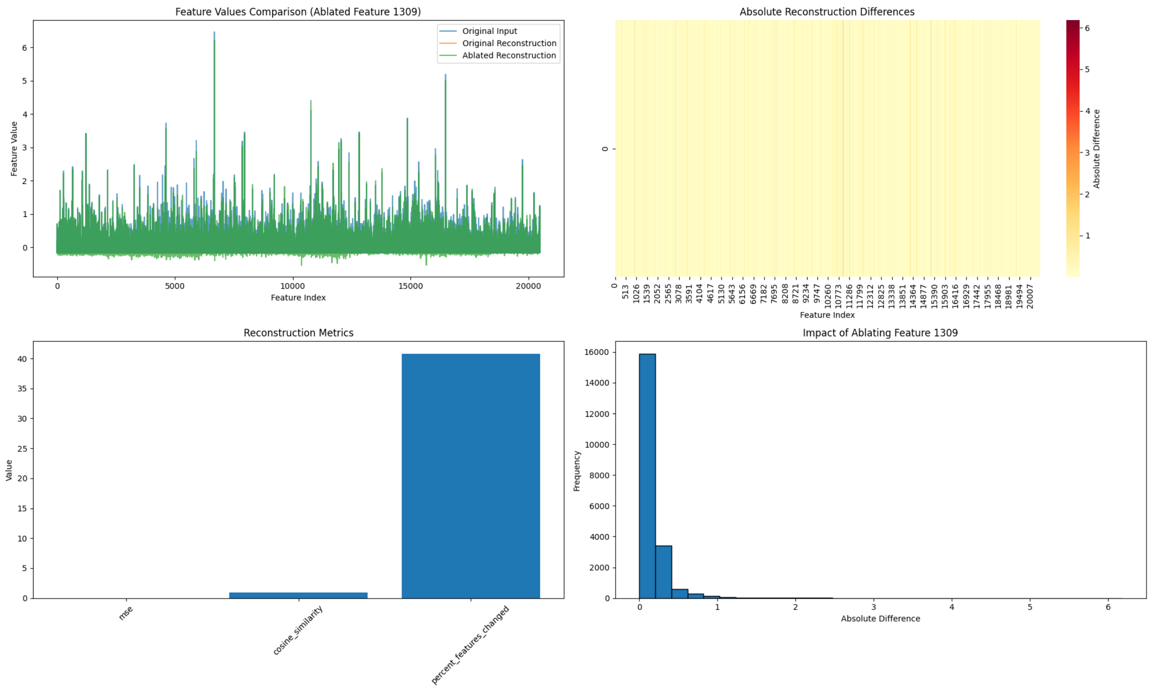


Figure 9: Feature Ablation Visualizations

to zero and decoded our modified latent representation back into an output which was then compared to the original using MSE and cosine similarity. The same process was done for noise, but instead of setting the feature to zero, we set it to be some random noise. We then visualized our results as follows. See Figure 9.

For the following two paragraphs, when we refer to ”plots”, we are referring to Figure 9. The plot in the top left shows the relative activations between the original reconstruction and the ablated reconstruction. The overlap seems to be fairly consistent, indicating that none of the features we surveyed had massive ablation effects. One thing that should be noted here is that there is a relative effect between features when we ablate. This is something we felt that the Anthropic researchers left out of their study–the impact of ablated features on other feature activations. Shown in our heatmap in the top left, for the most part, ablating a single feature did not have a significant effect on other features. However, for certain features, it changed the feature's activation in the reconstruction quite significantly which implies that these features affect our reconstruction more heavily following the ablation.

The two plots on the bottom more or less tell us the same thing, which is that the feature we ablated was not particularly important to the rest of the model. Despite the percentage of features changed being above 40%, we still see a cosine similarity of 0.93 and a MSE of essentially zero, which tells us that features did not change much even if they did change. We should expect change in a lot of features, but minimal changes imply that the ablated feature is not particularly impactful. We see this again in the plot on the bottom right, which shows that the vast majority of our features had an absolute difference of less than 1, with a smaller proportion changing any more than that. This is similar to the visualization of the heatmap, but it is sorted to provide some more insight into the individual impact of a feature. We spent some time trying to find a feature which had a larger ablation effect, but due to the vague nature of what our features encode, this was rather difficult. Future research should explore the features with most impact. Another idea would be to explore their feature neighborhoods to see how proximity in the latent space corresponds to "relevance" in the rest of the model.

6 Our Novel Experiment

We wanted to not only replicate the findings of the Anthropic paper, but also seek to develop some sort of novel insight of our own. We were inspired by a paper regarding random walks in Stable Diffusion where researchers proposed that generative image models like Stable Diffusion understand visual representations as a latent manifold. This implies that images are points in space and you can get from any single image to another by following some path in the latent space (Team (2022)). Before proceeding, let us define latent space. The latent space of a model is the abstraction of a model's inner workings. Returning to our earlier example of "king" and "queen", these lie within the latent space of an LLM similar to how images lie within the latent space of a diffusion model. Additionally, features lie within the latent space of a model. Our thought is that if we follow a method similar to the Stable Diffusion paper, we can learn more about how concepts are represented in latent space. We can also learn how well we can traverse from one idea to another based on the manipulation of our features. So, if we have some input and take a random iterative walk in latent space, the hope is that our model has learned a rigid enough latent space representation such that our resulting output will not be unintelligible.

An example will serve well here. Imagine we start with some input: "The fox ran around the bush". As we take a random walk in latent space, we should observe slight changes. For example, on the second iteration, we could get "The dog ran under the bush". The hope is that after a few iterations, we preserve some meaning and sentence structure. If we do not observe this, we know that small changes in our latent space can result in massive changes in our output. This implies some faulty learned features or a weakly constructed latent space where entirely unrelated concepts are very close together.

The following section of the paper will discuss our methodology and the results we yielded from our experiment.

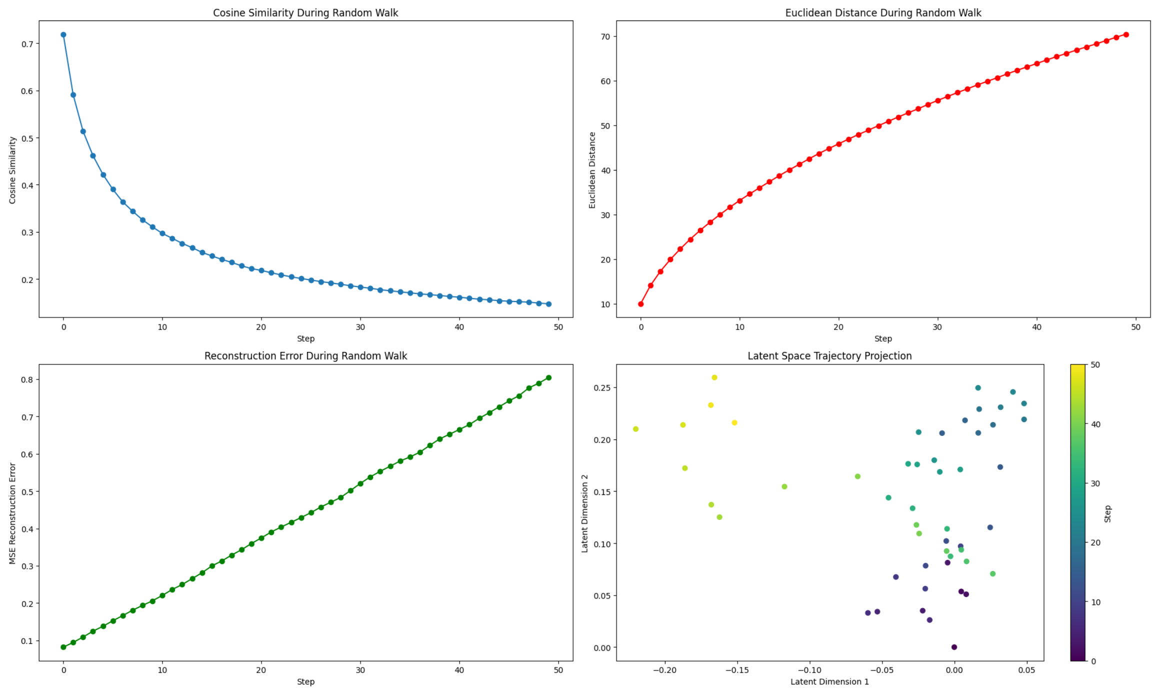


Figure 10: Latent Space Random Walk Results

Our code starts with a piece of the input data which is our starting point in latent space. We then iterate some set number of times calculating random noise (directionality) and moving in that direction away from our starting point by a fixed step size. We want to observe how quickly our latent point degrades from the starting point. We visualized this through calculating reconstruction loss and cosine similarity at each step in the process and plotting them in the left two plots. For the remaining part of this section, we will be referring to the plots in Figure 10.

We see that as we progress in our steps/iterations, reconstruction loss increases and cosine similarity decreases. This tracks, as the further away we get from our original point, the more nonsensical our current point should be. What is interesting to us, however, is that there is not much variability in the loss and similarities. Even after running the code multiple times on multiple starting points, we see roughly the same plots. Increasing and decreasing the step size did seem to make the reconstruction loss plot less and more erratic respectively. There were other unexpected findings as well. For example, at least in theory, the euclidean distance graph should be far less smooth since we are moving randomly. Further research into this peculiarity would certainly be a goal moving forward.

We would also like to spend some time discussing the plot on the bottom right, which displays a 2D projection of the actual walk that our code took through the latent space. Earlier points are lighter in color, and later points are darker. We see that we are actually moving from one part of latent space to another. In the first few moves, we seem to be in a similar part of the latent space, thus explaining our low reconstruction loss and high cosine similarity, but after a while, we do move further, and it is at this point our latent point degrades.

We hope that this experiment offers some basis for intuition of the inner workings of SAE models and what exactly features describe, as well as the space in which features exist. There is certainly more research to be done in this area, but we feel this was a good introductory problem from which further research can be launched.

7 Future Research and Applications

Despite the bulk of this paper focusing on SAE models and how they work, we also want to discuss some of the real-world uses for SAE models. As mentioned earlier, as models get progressively more complex, multimodal, and knowledgeable, we want to ensure that model outputs are safe. For example, we do not want models consistently or even irregularly disseminating memory unsafe code (Bricken et al. (2023)). So, to prevent such things from happening, we need to develop some framework by which we can understand a model's comprehension of dangerous concepts which can then be tweaked. Insights from SAEs can also be used to develop more descriptive and efficient models.

Despite SAEs being incredibly powerful, we still run into some roadblocks when working with them. For example, evaluating the trade off between our reconstruction loss and sparsity is a difficult, if not impossible, balance to strike when trying to understand exactly how models think. Another issue arises in simply scaling SAEs. We discussed this briefly earlier on, but as SAE models grow and the models which they try to describe grow, the compute necessary to create such insights will be massive and potentially unfeasible (Templeton et al. (2024)). Additionally, even if we do create an incredibly descriptive SAE model, we still do not necessarily truly see how the model represents learned concepts. The perfect example of this is the idea of feature completeness. At what point do we have a large enough SAE? Perhaps one of the most limiting things about SAEs is that we do not know for certain whether features are actually a representative way to understand machine thought (Bricken et al. (2023)). The entire basis of SAEs is the idea of dictionary learning and the superposition hypothesis, which seem to be backed empirically but cannot be proven (Templeton et al. (2024)). So, perhaps the entire SAE experiment is a convenient trend and actually provides no significant information.

This is likely not the case, but it should motivate us to look into other subfields within machine interpretability. One that we feel is particularly interesting is a group of methods based on a central principle known as causal tracing. Causal tracing is the idea that we can introduce some sort of corruptions to a models inputs (embeddings) and observe the models resulting output. From this, we can subsequently learn how certain parts of the model impact its factual associations (Meng et al. (2022)). See Figure 11. A popular technique which arose from this idea is known as Rank One Model Editing (ROME). The purpose of ROME is to be able to edit LLMs factual associations (Meng et al. (2022)). So, similar to SAEs, it can allow us to make models safer and more accurate.

ROME works by treating individual MLPs (multilayer perceptrons) within a model as key value stores. The idea is that certain MLPs store certain generalizable or specific facts, and by tweaking an MLPs weights, we can rewrite a key-value pair (Meng et al. (2022)). In the context of causal tracing, we introduce corruptions to our input embeddings, then

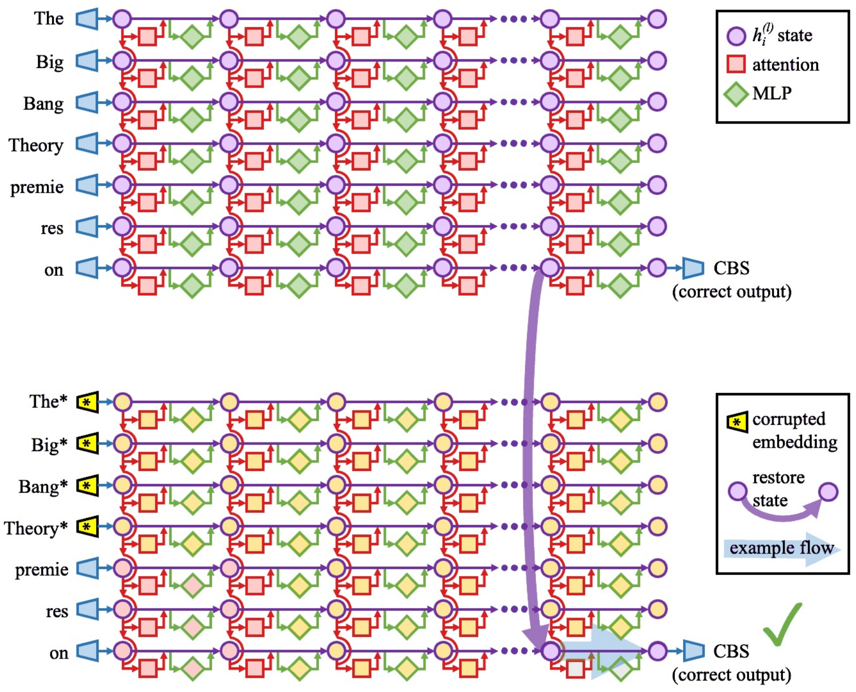


Figure 11: Meng et al. (2022)

in reverse restore the states of our MLPs and see which particular MLP or subset of MLPs seems to associate with the fact we want to change. See Figure 11. What is interesting about this approach is that it is not really reliant on an assumption of dictionary learning. Despite this, we can still tweak a model in such a way that we can affect its factual association. What seems to be the case too is that we are not just changing the factual association in some surface level associative way. Rather, there seems to be an actual change in the knowledge of the model, as when researchers prompted the model in similar contexts, it seemed to preserve knowledge (Meng et al. (2022)).

Another interesting bit of information from the ROME article is that factual associations seem to show up at two sites within an MLP (an early site and a late site). This tells us that knowledge retrieval seems to happen earlier on in a model's processing of a token sequence and then attention mechanisms carry that information forward to the late site (Meng et al. (2022)). Granted, the experiments conducted in this paper did prompt the model in a set "knowledge tuple" sequence of tokens where they had the subject, relation, and object in that order for any given concept (Meng et al. (2022)). ROME was employed on GPT-2, but an interesting experiment for any reader would be to expand that methodology to another model.

Techniques similar to ROME have been used by researchers at the same lab to conduct memory editing at a far larger scale as well as memory editing within Diffusion models (Meng et al. (2022)). We recommend that the reader look into both MEMIT and ROME as not only are they fascinating model editing methods, but they also are rather dense, and contribute heavily to one's understanding of transformer models as a whole.

We have reached the conclusion of this paper. For further research into the field, we recommend that the reader look into the code associated with this tutorial, Neel Nanda's training codebase, and the references for a list of valuable resources. Additionally, if the idea of random walks particularly intrigues you, please reference the Diffusion paper (Team (2022)).

References

Joseph Bloom. Open source sparse autoencoders for all residual stream layers of gpt2 small. https://www.alignmentforum.org/posts/f9EgfLSurAiqRJySD/open-source-sparse-autoencoders-for-all-residual-stream, 2024.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/glorot11a.html.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization, 2018. URL https://arxiv.org/abs/1712.01312.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems*, 36, 2022. arXiv:2202.05262.

Agnieszka Mikołajczyk-Bareła and Michał Grochowski. A survey on bias in machine learning research, 2023. URL https://arxiv.org/abs/2308.11254.

Neel Nanda. Open-source replication and commentary on anthropic's results. https://www.lesswrong.com/posts/fKuugaxt2XLTkASkk/open-source-replication-and-commentary-on-anthropic-s, 2023. Accessed: 2024-12-14.

Keras Team. Random walks with stable diffusion. https://keras.io/examples/generative/random_walks_with_stable_diffusion/, 2022. Accessed: 2024-12-14.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.