Taking features out of superposition with sparse autoencoders

Samson Koelle

May 20, 2024

Me

Marina Meila: 2017 - 2021

PhD thesis: Geometric algorithms for interpretable unsupervised learning

Uberduck: 2021 - 2024

Y Combinator, music generation for Quizlet, Cadbury Chocolate, David Guetta...

On job hunt

Taking features out of superposition with sparse autoencoders

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]University of Bristol, [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. Moreover, we show that with our learned set of features, we can pinpoint the features that are causally responsible for counterfactual behaviour on the indirect object identification task (Wang et al., 2022) to a finer degree than previous decompositions. 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.

ICLR 2024, available on Openreview

Summary

- Main idea: interpreting directions in the residual embedding space of transformer models is easier if we expand this latent space
- This presentation is an attempt to summarize what this means, why this is true, and how it is useful
- I think there are a lot of interesting analogies to more classical manifold learning research

Sections

Transformers

Transformer basics

- Transformers are a type of feedforward neural network distinguished by the presence of *transformer blocks*
- The original paper by Vaswani et al. [2017] has over 120,000 citations
- We'll focus on sequence-to-sequence transformers

$$T : \mathcal{H}^L \to \mathcal{H}^L$$

$L :=$ input sequence length, $\mathcal{H} :=$ set of tokens

```
[11] model.state_dict().keys()

odict_keys(['embed.W_E', 'pos_embed.W_pos', 'blocks.0.attn.W_Q', 'blocks.0.attn.W_O', 'blocks.0.attn.b_Q', 'blocks.0.attn.b_O', 'blocks.0.attn.W_K', 'blocks.0.attn.W_V', 'blocks.0.attn.b_K',
'blocks.0.attn.b_V', 'blocks.0.attn.mask', 'blocks.0.attn.IGNORE', 'blocks.0.mlp.W_in', 'blocks.0.mlp.b_in', 'blocks.0.mlp.W_out', 'blocks.0.mlp.b_out', 'blocks.1.attn.W_Q', 'blocks.1.attn.W_O',
'blocks.1.attn.b_Q', 'blocks.1.attn.b_O', 'blocks.1.attn.W_K', 'blocks.1.attn.W_V', 'blocks.1.attn.b_K', 'blocks.1.attn.b_V', 'blocks.1.attn.mask', 'blocks.1.attn.IGNORE', 'blocks.1.mlp.W_in',
'blocks.1.mlp.b_in', 'blocks.1.mlp.W_out', 'blocks.1.mlp.b_out', 'blocks.2.attn.W_Q', 'blocks.2.attn.W_O', 'blocks.2.attn.b_Q', 'blocks.2.attn.b_O', 'blocks.2.attn.W_K', 'blocks.2.attn.W_V',
'blocks.2.attn.b_K', 'blocks.2.attn.b_V', 'blocks.2.attn.mask', 'blocks.2.attn.IGNORE', 'blocks.2.mlp.W_in', 'blocks.2.mlp.b_in', 'blocks.2.mlp.W_out', 'blocks.2.mlp.b_out', 'blocks.3.attn.W_Q',
'blocks.3.attn.W_O', 'blocks.3.attn.b_Q', 'blocks.3.attn.b_O', 'blocks.3.attn.W_K', 'blocks.3.attn.W_V', 'blocks.3.attn.b_K', 'blocks.3.attn.b_V', 'blocks.3.attn.mask', 'blocks.3.attn.IGNORE',
'blocks.3.mlp.W_in', 'blocks.3.mlp.b_in', 'blocks.3.mlp.W_out', 'blocks.3.mlp.b_out', 'blocks.4.attn.W_Q', 'blocks.4.attn.W_O', 'blocks.4.attn.b_Q', 'blocks.4.attn.b_O', 'blocks.4.attn.W_K',
'blocks.4.attn.W_V', 'blocks.4.attn.b_K', 'blocks.4.attn.b_V', 'blocks.4.attn.mask', 'blocks.4.attn.IGNORE', 'blocks.4.mlp.W_in', 'blocks.4.mlp.b_in', 'blocks.4.mlp.W_out', 'blocks.4.mlp.b_out',
'blocks.5.attn.W_Q', 'blocks.5.attn.W_O', 'blocks.5.attn.b_Q', 'blocks.5.attn.b_O', 'blocks.5.attn.W_K', 'blocks.5.attn.W_V', 'blocks.5.attn.b_K', 'blocks.5.attn.b_V', 'blocks.5.attn.mask',
'blocks.5.attn.IGNORE', 'blocks.5.mlp.W_in', 'blocks.5.mlp.b_in', 'blocks.5.mlp.W_out', 'blocks.5.mlp.b_out', 'blocks.6.attn.W_Q', 'blocks.6.attn.W_O', 'blocks.6.attn.b_Q', 'blocks.6.attn.b_O',
'blocks.6.attn.W_K', 'blocks.6.attn.W_V', 'blocks.6.attn.b_K', 'blocks.6.attn.b_V', 'blocks.6.attn.mask', 'blocks.6.attn.IGNORE', 'blocks.6.mlp.W_in', 'blocks.6.mlp.b_in', 'blocks.6.mlp.W_out',
'blocks.6.mlp.b_out', 'blocks.7.attn.W_Q', 'blocks.7.attn.W_O', 'blocks.7.attn.b_Q', 'blocks.7.attn.b_O', 'blocks.7.attn.W_K', 'blocks.7.attn.W_V', 'blocks.7.attn.b_K', 'blocks.7.attn.b_V',
'blocks.7.attn.mask', 'blocks.7.attn.IGNORE', 'blocks.7.mlp.W_in', 'blocks.7.mlp.b_in', 'blocks.7.mlp.W_out', 'blocks.7.mlp.b_out', 'blocks.8.attn.W_Q', 'blocks.8.attn.W_O', 'blocks.8.attn.b_Q',
'blocks.8.attn.b_O', 'blocks.8.attn.W_K', 'blocks.8.attn.W_V', 'blocks.8.attn.b_K', 'blocks.8.attn.b_V', 'blocks.8.attn.mask', 'blocks.8.attn.IGNORE', 'blocks.8.mlp.W_in', 'blocks.8.mlp.b_in',
'blocks.8.mlp.W_out', 'blocks.8.mlp.b_out', 'blocks.9.attn.W_Q', 'blocks.9.attn.W_O', 'blocks.9.attn.b_Q', 'blocks.9.attn.b_O', 'blocks.9.attn.W_K', 'blocks.9.attn.W_V', 'blocks.9.attn.b_K',
'blocks.9.attn.b_V', 'blocks.9.attn.mask', 'blocks.9.attn.IGNORE', 'blocks.9.mlp.W_in', 'blocks.9.mlp.b_in', 'blocks.9.mlp.W_out', 'blocks.9.mlp.b_out', 'blocks.10.attn.W_Q', 'blocks.10.attn.W_O',
'blocks.10.attn.b_Q', 'blocks.10.attn.b_O', 'blocks.10.attn.W_K', 'blocks.10.attn.W_V', 'blocks.10.attn.b_K', 'blocks.10.attn.b_V', 'blocks.10.attn.mask', 'blocks.10.attn.IGNORE',
'blocks.10.mlp.W_in', 'blocks.10.mlp.b_in', 'blocks.10.mlp.W_out', 'blocks.10.mlp.b_out', 'blocks.11.attn.W_Q', 'blocks.11.attn.W_O', 'blocks.11.attn.b_Q', 'blocks.11.attn.b_O',
'blocks.11.attn.W_K', 'blocks.11.attn.W_V', 'blocks.11.attn.b_K', 'blocks.11.attn.b_V', 'blocks.11.attn.mask', 'blocks.11.attn.IGNORE', 'blocks.11.mlp.W_in', 'blocks.11.mlp.b_in',
'blocks.11.mlp.W_out', 'blocks.11.mlp.b_out', 'unembed.W_U', 'unembed.b_U'])
```

GPT2-small model parameter groups (number of blocks $B = 12$)

Transformer basics

- Start with a random embedding $T_* : \mathcal{H} \to \mathbb{R}^m$
- Blocks $T_b : \mathbb{R}^{m \times L} \to \mathbb{R}^{m \times L}$ progressively deform projection of tokens into $\mathbb{R}^{m \times L}$
- An unembedding $T_{-*} : \mathbb{R}^m \to \mathbb{R}^{|\mathcal{H}|}$ followed by softmax $s : \mathbb{R}^{|\mathcal{H}|} \to \mathbb{R}^{|\mathcal{H}|}$ is applied after the last block

$$T(h_L) = s^L \circ T_{-*}^L \circ T_{B-1} \circ T_{B-2} \circ \ldots \circ T_0 \circ T_*(h_L) \qquad (1)$$

- During training, target variables are shifted right by one version of input variables, and a mask is used on the *attention mechanism* so that the network only has knowledge of previous positions in the sequence
- During inference, last position in sequence is appended over repeated forward passes

Transformer blocks

- Each block consists of an attention mechanism A_b and *multilayer perceptron* M_b

$$T_b = M_b^L \circ A_b$$

- $A_b : \mathbb{R}^{m \times L} \to \mathbb{R}^{m \times L}$
- $M_b : \mathbb{R}^m \to \mathbb{R}^m$ independently for each $l \in [L]$

```
blocks.0.attn.W_Q: torch.Size([12, 768, 64])
blocks.0.attn.W_O: torch.Size([12, 64, 768])
blocks.0.attn.b_Q: torch.Size([12, 64])
blocks.0.attn.b_O: torch.Size([768])
blocks.0.attn.W_K: torch.Size([12, 768, 64])
blocks.0.attn.W_V: torch.Size([12, 768, 64])
blocks.0.attn.b_K: torch.Size([12, 64])
blocks.0.attn.b_V: torch.Size([12, 64])
blocks.0.attn.mask: torch.Size([1024, 1024])
blocks.0.attn.IGNORE: torch.Size([])
blocks.0.mlp.W_in: torch.Size([768, 3072])
blocks.0.mlp.b_in: torch.Size([3072])
blocks.0.mlp.W_out: torch.Size([3072, 768])
blocks.0.mlp.b_out: torch.Size([768])
```

Parameter shapes (alternate notation)

$H = 12$ (# heads)

$M = 768$ (embedding dimension)

$D = 64$ (attention dimension)

$Z = 3072$ (MLP dimension)

Multiheaded attention

$$A_b(x) = O_b^L((s^{HL_2}(\frac{(Q_b^L(x))_d(K_b^L(x))^d}{\sqrt{D}}))_{l_1}^D(V_b^L(x))^l)$$

- Einstein notation ($c_i x^i = \sum_i c_i x_i$)
- Can use two values of D (i.e. D_{VO}, D_{QK})
- l_1, l_2 two axes of *attention matrix* $s^{HL_2}(\frac{(Q_b^L(x))_d(K_b^L(x))^d}{\sqrt{D}})$

$$\frac{\text{softmax } s : \mathbb{R}^L \to \mathbb{R}^L}{Q_b : \mathbb{R}^{M\times H} \to \mathbb{R}^{D\times H}}$$

$$x \mapsto (W_{Q_b})_m x^m$$

$$W_{Q_b} \in \mathbb{R}^{M\times D\times H}$$

$$\frac{}{K_b : \mathbb{R}^{M\times H} \to \mathbb{R}^{D\times H}}$$

$$x \mapsto (W_{K_b})_m x^m$$

$$W_{K_b} \in \mathbb{R}^{M\times D\times H}$$

$$\frac{}{V_b : \mathbb{R}^{M\times H} \to \mathbb{R}^{D\times H}}$$

$$x \mapsto (W_{V_b})_m x^m$$

$$W_{V_b} \in \mathbb{R}^{M\times D\times H}$$

$$\frac{}{O_b : \mathbb{R}^{D\times H} \to \mathbb{R}^M}$$

$$x \mapsto (W_{O_b})_{hd} x^{hd}$$

$$W_{O_b} \in \mathbb{R}^{M\times D\times H}$$

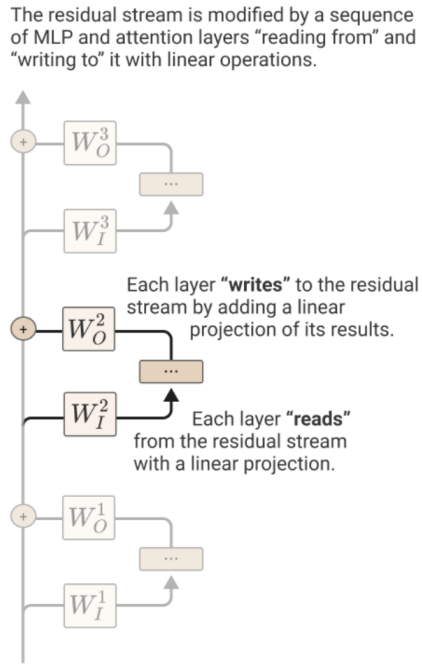
```python
class MultiHeadAttentionNew(nn.Module):
    def __init__(self, n_head, d_model, d_k, d_v, dropout=0.1):
        super().__init__()
        self.n_head = n_head

        self.w_qs = nn.Linear(d_model, n_head * d_k)
        self.w_ks = nn.Linear(d_model, n_head * d_k)
        self.w_vs = nn.Linear(d_model, n_head * d_v)

        nn.init.normal_(self.w_qs.weight, mean=0, std=np.sqrt(2.0 / (d_model + d_k)))
        nn.init.normal_(self.w_ks.weight, mean=0, std=np.sqrt(2.0 / (d_model + d_k)))
        nn.init.normal_(self.w_vs.weight, mean=0, std=np.sqrt(2.0 / (d_model + d_v)))

        self.fc = nn.Linear(n_head * d_v, d_model)
        nn.init.xavier_normal_(self.fc.weight)
        self.dropout = nn.Dropout(p=dropout)
        self.layer_norm = nn.LayerNorm(d_model)

    def forward(self, q, k, v, mask=None):
        residual = q
        q = rearrange(self.w_qs(q), 'b l (head k) -> head b l k', head=self.n_head)
        k = rearrange(self.w_ks(k), 'b t (head k) -> head b t k', head=self.n_head)
        v = rearrange(self.w_vs(v), 'b t (head v) -> head b t v', head=self.n_head)
        attn = torch.einsum('hblk,hbtk->hblt', [q, k]) / np.sqrt(q.shape[-1])
        if mask is not None:
            attn = attn.masked_fill(mask[None], -np.inf)
        attn = torch.softmax(attn, dim=3)
        output = torch.einsum('hblt,hbtv->hblv', [attn, v])
        output = rearrange(output, 'head b l v -> b l (head v)')
        output = self.dropout(self.fc(output))
        output = self.layer_norm(output + residual)
        return output, attn
```

ein

Residual stream

We may think of computation up to each block as an iteratively refined projection $\mathcal{H}^{m \times L} \to \mathbb{R}^{m \times L}$

- The output of each block is added to the input and fed to the next block
- Thus, we call the state of the L tokens in the m dimensional embedding space the *residual stream.*
- The attention dimension D constrains the subspace of the residual stream operated on by each head (i.e. the dimension of W_O and W_I is lower than the residual stream, so blocks may edit it orthogonally).

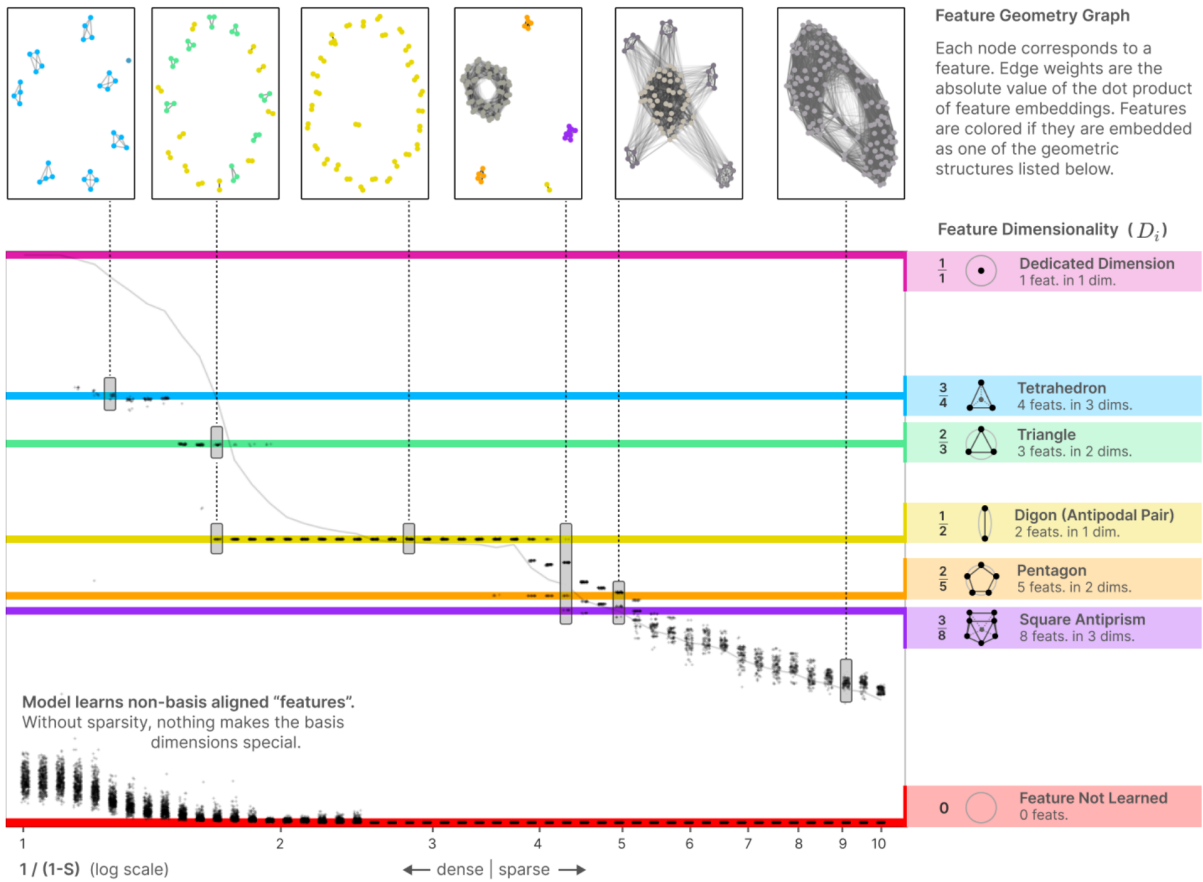


Structure across blocks [Nan]

Superposition

- Interpretability research for transformers tries to understand the relationship between textual concepts and positions in the residual stream.
- Neuron that fire in multiple contexts are known as *polysemantic*
- In this talk, we implicitly focus on neurons whose weights are W_O
- A residual stream with basis given by polysemantic neurons is said to exhibit *superposition*

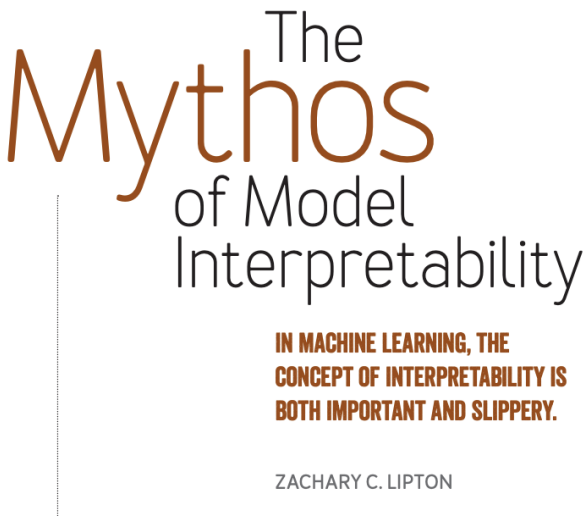
Superposition geometry depends on sparsity



Simulated data, one layer MLP, $D = \|W_i\|^2 / \sum_j (\langle \frac{W_i}{\|W_i\|}, W_j \rangle)^2$ [Elh]

Interpretability

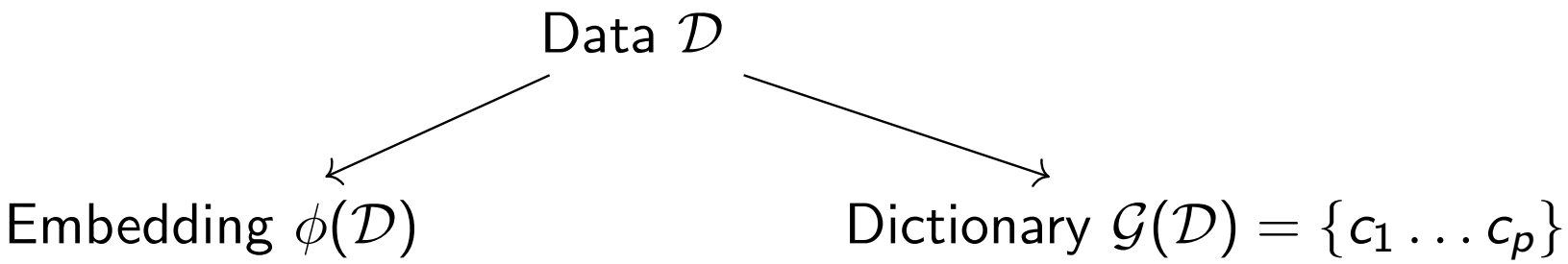
Why interpretability?



- Molnar [2023] is a great introduction to interpretability for neural networks

- Sample efficiency: interpretable models can learn from less data and generalize better
- Safety: For language models, a common goal is to guarantee how a model will respond in a certain situation
- There is a distinction between post hoc model interpretability and inherently interpretable models

Interpretability concepts

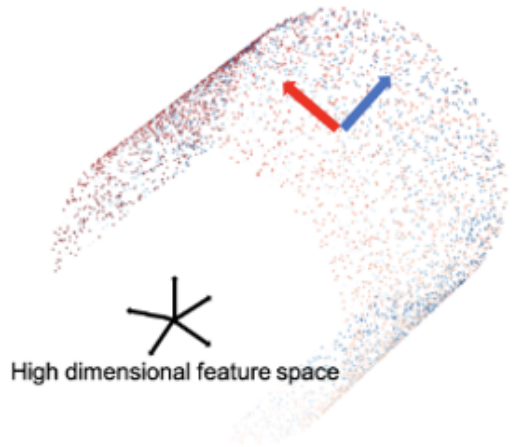


Concept	**Definition**	**Example**
Data	Raw input data	Molecular dynamics coordinates, gene expression, language model tokens
Embedding	Potentially learned transformation	Projection into latent spaces of neural networks, Principal Component Analysis (PCA) results, UMAP results
Dictionary	Interpretable functions	Bond rotation, cell-cycle, luminosity

Estimating interpretable representations

Interpretability algorithms attempt automatic association $\mathcal{D}$ or $\phi(\mathcal{D})$ with $\mathcal{G}(\mathcal{D})$

- Local Interpretable Model-agnostic Explanations [Ribeiro et al., 2016]
- Tangent Space Lasso [Koelle et al., 2024]



The TSLasso approach - dictionary functions colored on the data manifold

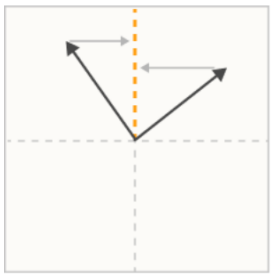
A large number of interpretability approaches are reviewed in Elh, Molnar [2023].

Vector space representations

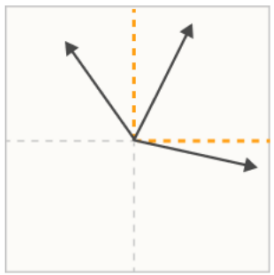
- It has been observed that (e.g. [Mikolov et al., 2013]) that neural networks create vector spaces of concepts in their latent space

$$\phi(\text{"}king\text{"}) - \phi(\text{"}man\text{"}) + \phi(\text{"}woman\text{"}) = \phi(\text{"}queen\text{"})$$

- But, these concepts are often not basis aligned, and more concepts are represented than there are dimensions in the space.



Polysemanticity is what we'd expect to observe if features were not aligned with a neuron, despite incentives to align with the privileged basis.



In the **superposition hypothesis**, features can't align with the basis because the model embeds more features than there are neurons. Polysemanticity is inevitable if this happens.

Elh: possible because of sparsity and noisy computation.

Another view on interpretability

- The notion of interpretability research expounded in Cunningham et al. [2023] comes from Elh
- Functional independence in Koelle et al. [2024] = decomposibility

Summary: A Hierarchy of Feature Properties

The ideas in this section might be thought of in terms of four progressively more strict properties that neural network representations might have.

- **Decomposability:** Neural network activations which are *decomposable* can be decomposed into features, the meaning of which is not dependent on the value of other features. (This property is ultimately the most important – see the role of decomposition in defeating the curse of dimensionality.)
- **Linearity:** Features correspond to directions. Each feature f_i has a corresponding representation direction W_i. The presence of multiple features $f_1, f_2 \ldots$ activating with values $x_{f_1}, x_{f_2} \ldots$ is represented by $x_{f_1} W_{f_1} + x_{f_2} W_{f_2} \ldots$.
- **Superposition vs Non-Superposition:** A linear representation exhibits superposition if $W^T W$ is not invertible. If $W^T W$ is invertible, it does not exhibit superposition.
- **Basis-Aligned:** A representation is basis aligned if all W_i are one-hot basis vectors. A representation is partially basis aligned if all W_i are sparse. This requires a privileged basis.

The first two (decomposability and linearity) are properties we hypothesize to be widespread, while the latter (non-superposition and basis-aligned) are properties we believe only sometimes occur.

Here, feature = dictionary function.

Constructing Dictionaries for Language Models

- One of the essential questions when considering interpretability w.r.t. a dictionary is what features should be in the dictionary
- A method for scoring features in language models is proposed by Bil.
- In this method, data points that have high values w.r.t. a particular feature c_j are fed to an "explainer" model (GPT4) that comes up with an explanation for the feature.
- That explanation is then used by an "interpretability scoring" model (GPT3.5) to predict feature values for held out data points.
- The idea is that simpler, more interpretable concepts will result in more accurate prediction.
- Summarized as interpretability score $\hat{I}(c_j) = \|c_j(h_i) - \hat{c}_j(h_i)\|$.

Sparse autoencoders

Problem setup

- We have access to an embedding matrix $\phi(\mathcal{D}) \in \mathbb{R}^{n \times m}$ at a certain block of the transformer residual stream.
- Directions in $\mathbb{R}^m$ correspond to various concepts.
- However, these directions are in superposition.
- Main idea: compare $\hat{I}(c_j)$ for various dictionary learning methods.

Sparse autoencoders

- Sparse autoencoders are a dictionary-learning method that reconstruct vectors from a sparse combination of learned dictionary features.
- Learn dictionary $c \in \mathbb{R}^p$ of functions from residual stream positions $x \in \mathbb{R}^m$
- Main idea: $p > m$ so that features are *monosemantic*.

Model:

$$\hat{x} = W^T c$$
$$c = \text{ReLU}(\text{Wx} + \text{b})$$

Objective function:

$$\|x - \hat{x}\|_2^2 + \lambda \|c\|_1$$

Results for selected concepts

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.

Cunningham et al. [2023]

Results for various dictionary learning methods

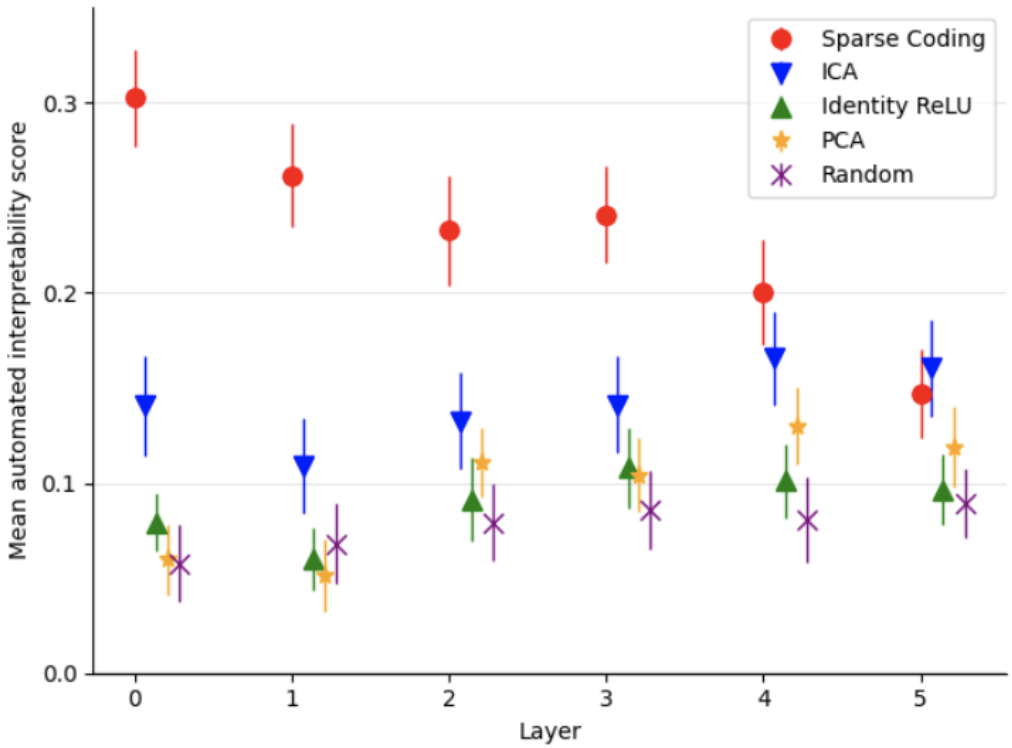


Figure 2: Average top-and-random autointerpretability score of our learned directions in the residual stream, compared to a number of baselines, using 150 features each. Error bars show 95% confidence intervals around means. The feature dictionaries used here were trained for 10 epochs using $\alpha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

Cunningham et al. [2023]

Circuit ablation study

- Transformers represent *circuits* that correspond to complex concepts.
- Indirect object identification [Wang et al., 2022]: "Then, Alice and Bob went to the store. Alice gave a snack to _".
- Conmy algorithm finds subset of most predictive features.

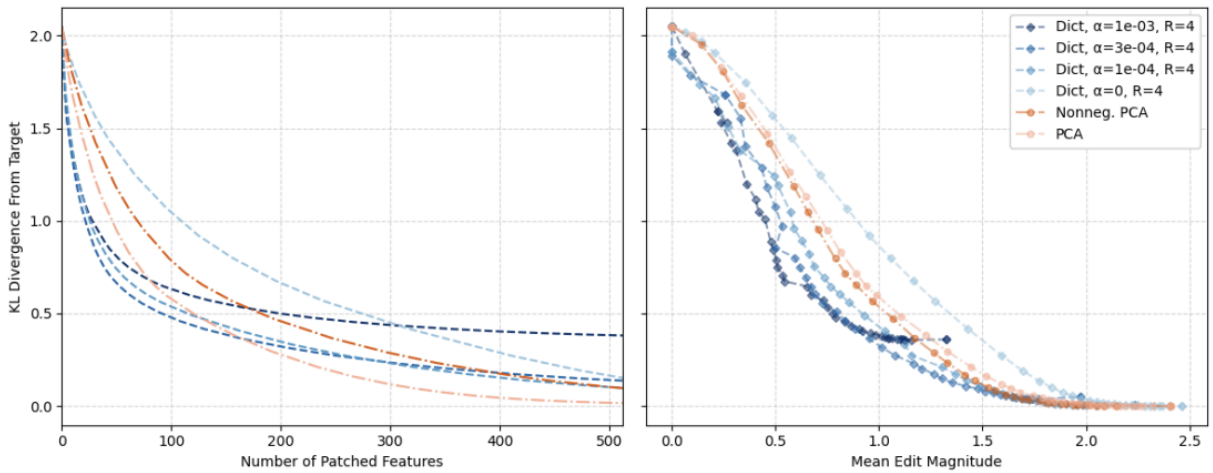


Figure 3: (Left) Number of features patched vs KL divergence from target, using various residual stream decompositions. We find that patching a relatively small number of dictionary features is more effective than patching PCA components and features from the non-sparse $\alpha = 0$ dictionary. (Right) Mean edit magnitude vs KL divergence from target as we increase the number of patched features. We find that our sparse dictionaries improve the Pareto frontier of edit magnitude vs thoroughness of editing. In both figures, the feature dictionaries were trained on the first 10,000 elements of the Pile (Gao et al., 2020) (approximately 7 million activations) using the indicated α and R values, on layer 11 of Pythia-410M (see Appendix F for results on other layers).

Discussion

What really is superposition?

- Superposition is a phenomenon of latent spaces where the number of independent features that are encoded is greater than the dimension of the latent space.
- Noisy computation interpretation: intrinsic dimension of data manifold is higher than the dimension of the embedding space
- Sparsity interpretation: residual stream encodes tangent space of data fiber bundle.

Future directions

- Investigate at what happens layer by layer.
- Investigate MLP and/or attention mechanism.
- Embed substrings (i.e. don't just score interpretability of entire input string)
- Tangent space lasso on most interpretable features
- Dimension estimation by stability analysis of p [Robert_AIZI].
- Improved circuit discovery [Bri].

References

Language models can explain neurons in language models. https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html. Accessed: 2024-5-13.

Towards monosemanticity: Decomposing language models with dictionary learning. https://transformer-circuits.pub/2023/monosemantic-features. Accessed: 2024-5-13.

Toy models of superposition. https://transformer-circuits.pub/2022/toy_model/index.html. Accessed: 2024-5-13.

A mathematical framework for transformer circuits. https://transformer-circuits.pub/2021/framework/index.html. Accessed: 2024-5-13.

Writing better code with pytorch and einops. https://einops.rocks/pytorch-examples.html. Accessed: 2024-5-15.

Arthur Conmy. Automatic-Circuit-Discovery.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. September 2023.

Samson J Koelle, Hanyu Zhang, Octavian-Vlad Murad, and Marina Meila. Consistency of Dictionary-Based manifold learning. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 4348–4356. PMLR, 2024.

Thanks!