Identifying Sparsely Active Circuits Through Local Loss Landscape Decomposition

Anonymous Authors[1]

Abstract

1. Background

Mechanistic interpretability is somewhat of an emerging field and aims to uncover the internal mechanisms responsible for seemingly black box behavior of large models so that developers can better understand, intervene on, and align model (Bereska & Gavves, 2024). Within mechanistic interptretabilty, decomposition methods aim decompose model behacvior into subcomponents that are less complex and more human interpretabile, but that still fully express model behavior. The most popular method in this space is Sparse Autoencoders, which have been successful in decomposing compressed features in the activation space of a model (Gao et al., 2024; Cunningham et al., 2023; Bricken et al., 2023).

1.1. Limitations of Activation Space-Based Methods

Sparse autoencoders rely on the activation space of a model, the output of a set of model's neurons during inference. Sparse autoencoders (SAEs) are designed to identify latent features by projecting a model's compressed activation space into a sparsely activated, overcomplete basis. These learned basis vectors represent distinct features, which can then be used to reconstruct the original activations.

Despite their utility, activation-based decomposition methods like SAEs have notable limitations. First, they assume that activation space consists of linear combinations of feature vectors, yet growing evidence suggests that even architectures designed to encourage linearity, such as residual streams, may encode non-linear features (Engels et al., 2024a;b). Second, activation-based approaches generally treat layers and blocks as independent computational modules, reconstructing activations from specific layers without capturing the possibility of cross-layer interactions and superposition (Merullo et al., 2024; Lindsey et al., 2024). This limitation is particularly problematic in architectures beyond transformers, such as recurrent networks, adversarial networks, diffusion models, and classic RL models, where the separation of activation spaces is even less well-defined (Pascanu, 2013; Goodfellow et al., 2014; Ho et al., 2020; Mnih et al., 2015). Finally, activation-based methods provide little insight into how features emerge during training, making it difficult to link learned behaviors to specific data or optimization dynamics. This lack of connection to the training process limits our ability to intervene in model behavior at an early stage or refine training data to steer a model toward more desirable properties.

1.2. Loss Landscape Geometry

An alternative to interpreting models through their activations is to interpret models through their loss landscape geometry, understanding how perturbing parameters, rather than activations, affects model output and behavior. Parameters are the fundamental entities updated during training, capturing information from data in a way that persists beyond individual inferences. Unlike activations, which are transient and input-dependent, the parameter space reflects the cumulative outcome of the training process, potentially providing deeper insights into how models generalize, store knowledge, and develop internal representations.

A growing body of research explores the relationship between parameter space, data, and model behavior. Singular learning theory (SLT) describes how the structure of high-dimensional parameter space influences generalization and feature formation (Watanabe, 2007; 2000; 2005; Wei et al., 2022), while developmental interpretability extends this by analyzing how parameter updates evolve over training(Sharkey et al., 2025). Studying parameter space has already led to key insights, such as the role of sharpness in grokking(Davies et al., 2023), phase transitions corresponding to important learning stages in language models(Wang et al., 2024; Hoogland et al., 2024), and distinctions between parameters associated with memorization vs. generalization(Bushnaq et al., 2024). These findings suggest that parameter-space analysis may uncover mechanisms that activation-based methods overlook, opening new avenues for interpretability and intervention.

1.3. Degenerate Loss Landscapes

We already know that loss landscapes have high levels of degeneracy. Mathematical degeneracy, data distribution degeneracy, etc. We hypothesize that

1.4. Contrastive Methods

Constrastive/paired methods are common in intepretability. They work

1.5. Loss Landscape Decomposition

We propose a method for identifying features and the circuits that produce them in large models by analyzing parameter space rather than activations. Our approach identifies directions in parameter space that correspond to specific circuits, drawing on insights from singular learning theory (SLT). Our method aims to identify principle directions in parameter space such that for a given sample, moving model parameters in most of these directions will not meaningfully change the model's output, but for a small subset of directions, the model's output will change dramatically. We hypotheize that these directions, or subnetworks, represent distinct computations.

To our knowledge, there have only been two prior methods attempting to decompose parameter space for interpretability. In one earlier work(Matena & Raffel, 2023), the authors decompose parameter space by computing principal directions of a per-sample Fisher Information matrix to resolve meaningful features. This approach aimed to identify parameter directions most sensitive to individual samples, capturing how different parts of the model contribute to specific tasks. However, due to computational constraints, it relied on diagonalization techniques to approximate key directions, which limited its ability to fully capture sparse, structured circuits. Another recent method, (Braun et al., 2025) to is Attribution Parameter Decomposition (APD), which uses a bi-optimization approach to identify subnetworks where (1) the sum of subnetwork weights is close to the original model parameters, and (2) for any given input, the sum of a smaller set of subnetworks - identified through topk attribution values - has low behavioral loss when compared to the original model.

Local loss landscape decomposition (L3D) aims to decompose a model's gradients of loss between output pairs with respect to parameter space 1. We seek to identify directions in parameter space such that (1) the set of directions can approximately reconstruct any gradient of paired sample divergences, and (2) for any given pair of samples, an even smaller subset of directions can be used in reconstructing this gradient. To reduce computational complexity, we learn low-rank representations of these parameter directions. In this work, we first describe the mathematical foundation of our approach. We then develop progressively more complex toy models to measure the efficacy of L3D and quantify its limitations.

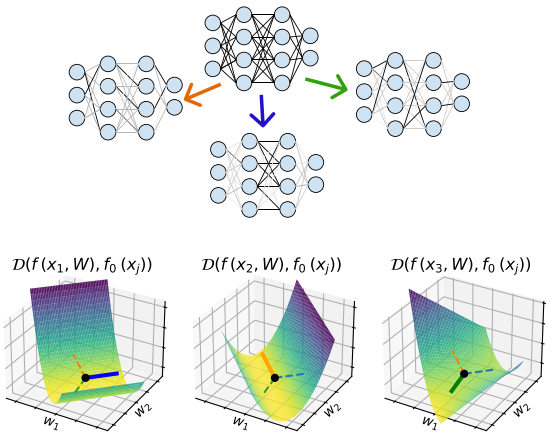


Figure 1: Decomposing a loss landscape into a set of principal directions , where (1) the set of directions can approximately reconstruct any per-sample Jacobian and (2) for any given sample, the curvature in most of the principle directions is approximately zero.

2. Methodology

2.1. Set up

Our project rests on the following premise: For a given input, there are many components of a model that are not involved in inference. Changing these parameters will not affect change the model's output. For the same given input, there are a smaller set of components in a model that are highly involved in inference. Changing these parameters will change the model's output, and will do so in a meaningful way - it will effectively turn a circuit "up" or "down", pushing the model's output towards the value that would be output if the model had never learned that specific circuit.

To formalize this, we will consider a model $f(x, W)$ $R^{n_s \times n_f} \mapsto R^{n_s \times n_o}$ that takes an input x and parameter values of W and outputs a vector of outputs values.

We wish to find a set of directions V in parameter space, such that for any given sample, a small set of these directions can be used to express the parameter directions that meaningfully change the model's output.

In the next sections, we will define "meaningful" changes, describe how to efficiently decompose parameters into these directions, explain a training algorithm for learning such a decomposition, and then explain how to use these decompo-

sitions to intervene on a model's behavior.

2.2. Divergences of Paired Outputs

We wish to identify parameter directions that can be thought of as modular circuits. We would like to identify circuits that we can intervene on, and control individual computations, without fully breaking a model. Put another way, intervening in a certain parameter direction should not move our outputs too far away from the original output distribution.

The object we will be decomposing is the gradient of divergence between a pair of outputs.

Divergence of a pair of outputs can be defined as:

$$\nabla_w D(f(x_i, W), f(x_r, W_0))|_{W=W_0} \quad (1)$$

$$\text{where } x_i, x_r \in X \quad (2)$$

Note that we will abbreviate $f(x_r, W_0)$ as just $f(x_r)$ from here on out.

where D is a divergence measuer, f is our model, x_i is our input of interest, x_r is a randomly selected reference input , W is a set of parameters and W_0 is the model's parameters after training. For our toy models we use MSE divergence. x_i and x_r are fixed for a given pair.

We are interested in identifying directions in parameter space that move the model's output within the same subspace that the original model's output lies. This is why we look at divergence with a reference output x_r. We will be interested in decomposing gradient of this divergence with respect to the parameters W. For short, we will call this $\nabla_W D$

$$\nabla_W D = \nabla_W D(f(x_i, W), f(x_r))|_{W=W_0} \quad (3)$$

2.3. Sparse Principal Directions

We want to transform our gradient into components that represent directions in parameter space, where each sample's gradient can be written as a linear combination of a small set of these components. Formalizing this:

We want to find transforms V^{in} $R^{n_v \times n_w}$ and V^{out} $R^{n_w \times n_v}$ where n_w is the number of parameters in the model, and n_v is the number of components we wish to use to represent the parameter space.

$$\nabla_W D \approx {V^{out}}_{:,topK} {V^{in}}_{topK,:} \nabla_W D \quad (4)$$

$$\text{where } topK = \text{argtopK}abs(V^{in} \nabla_W D) \quad (5)$$

$topK$ is a hyperparameter that controls the number of components we wish to use to reconstruct each sample. In practice, we use a batchTopK () and a fraction rather than an absolute number.

2.3.1. Low Rank parameter directions

Learning full rank parameter circuits would be extremely expensive, and we expect that modular, sparsely active circuits would be lower rank that their full-model counterparts (). Therefore, we use low-rank representations of our V^{in} and V^{out}, and correspondingly learn low-rank circuits.

While W is a vector of all of the parameters in a model, typically model parameters are organized into tensors $W = \{w_i\}_i$.

If our parameters are organized into tensors $W = \{w_i\}_i$, each subnetwork or parameter component can be organized as $V_i^{in} = \{{v^{in}}_i\}_i, V_i^{out} = \{{v^{out}}_i\}_i$ where we have parts of our subnetworks corresponding to each tensor in the original model parameters. We wish each of these tensors to be low rank, and we express them using the canonical polyadic decomposition () (a way to write 3+ dimentinoal tensors in terms of low-rank components).

$${v^{in}}_{i,j} = \sum_{r=1}^{R} a_{i,j,r} \times b_{i,j,r} \times c_{i,j,r} ... \quad (6)$$

Where R is the rank we wish to use to represent the parameter component, and the number of factors (a, b, c,...) in the factorization is equal to the number of dimensions in the tensor w_i.

2.4. Training

We wish to learn the decomposition-related transforms V^{in} and V^{out} that minimize the topK reconstruction loss the gradient of divergence of pairs:

$$L = ||\nabla_W D - {V^{out}}_{:,topK} {V^{in}}_{topK,:} \nabla_W D||_2^2 \quad (7)$$

2.5. Measuring and Intervening

Our learned subnetworks will just be the rows of V^{out}, reorganized into the same tensor structure as W.

If we wish to intervene using a single subnetwork, we can do so by updating the parameters in the direction of the subnetwork, multiplied by a scalar δ. We can tune our model in the direction of subnetwork v_i, and compute predictions by evaluating:

$$f(x, W + \delta v_i) \quad (8)$$

We can quantify the impact of a subnetwork in two ways. First, we can compute the impact of a subnetwork on a pair

Algorithm 1 Training Algorithm

for each epoch **do**
  for each X **do**
    for each x_i **do**
      Randomly select x_r X
      $\nabla_w D[i] = \nabla_w D(f(x_i, W), f(x_r))|_{W=W_0}$
    end for
    $\nabla_v D = V^{in}{}_{topK,:} \nabla_w D$
    thresh = topK(abs($\nabla_v D$))
    $\hat{\nabla}_w D = V^{out}(\nabla_v D \times (abs(\nabla_v D) > thresh))$
    $L = ||\nabla_w D - \hat{\nabla}_w D||_2^2$
    L.backward()
  end for
end for

of samples x_i, x_j, identyfing the subnetwork that, if intervened upon, would most strongly impact the the divergence of x_i's outputs when compared to x_j. The impact of subnetwork v_i on such a pair of samples, $I(v_i, x_i, x_j)$ can be measured by:

$$I(v_i, x_i, x_j) = abs(V^{in}{}_{i,:} \nabla_w D(f(x_i, W), f(x_j))|_{W=W_0}) \tag{9}$$

Secondly, we can average the impacts of a subnetwork v_i and an input x_i over many different reference samples to quantify the impact of the subnetwork on a single sample's predictions.

$$I(v_i, x_i) = \frac{1}{n} \sum_{j=1}^{n} I(v_i, x_i, x_j) \tag{10}$$

3. Results

We designed several toy models to test the efficacy of L3D.

3.1. Toy Model of Superposition

3.1.1. Setup

We start off by validating our algorithm on a well-studied toy problem, the toy model of superposition (TMS), with a small variation. TMS is a single-layer autoencoder, where the number of input and output neurons are equal, and there is a smaller hidden layer in between, with a final relu activation function (Elhage et al., 2022). The model is trained on a dataset of samples where few features are active at a time, and has been shown to "superimpose" these features in the hidden layer such that features embeddings in the hidden layer have minimal interference with each other. We train a toy model of superpsition with 5 features and 2 hidden dimensions (with 1-sparsity=.05), and then place three such models in parallel, to test our method's ability to resolve superimposed features, as well as indpendent circuit modules.

3.1.2. Decomposition

We use L3D to decompose the TMS model into 15 principal vectors, using rank-1 parameter tensors. We use topFrac=.1 and train L3D for 1000 epochs on 1000 datapoints. The network decomposes into subnetworks as we would expect: each subnetworks represents the embedding and reconstruction of a input $\mapsto$ output node, involving only the weights connecting the two and the biases of the output nodes. Figure 2 shows the subnetworks corresponding to each of the first 5 features, and Figure S3 shows the full decomposition. One thing to note is that principal parameter vectors do not have a preferred direction. L3D is equally likely to identify a parameter vector in the direction of θ as it is in the direction of $-\theta$. This is why, for example, subnetwork 1 in Figure 2 has weights that are in the opposite direction as those in the original network.

This decomposition resulted in a reconstrcution error of . 15 components of rank-1 can successfully express each individual $X_i : \hat{X}_i$ circuit, but does not capture the interference between features when multiple features are active in the same sample. Note that we expect decompositions of higher dimensional networks to exhhibit less reconstruction error, as the amount of nearly orthogonal parameter vectors (non-interferring) that can be compressed into parameter space scales exponentially with dimension.

Pull number form WandB

3.1.3. Intervention

Additionally, the low-rank parameter vectors learned by L3D can be used to perturb the weight space of a model. In principle, we could finetune a model with an adaptor that only changes the parameters of a network in the direction of a specified set of parameter vectors. Finetuning with a small set of parameter vectors should only affect the predictions of the samples for which those parameter vectors were active subnetworks. While we do not fine tune a model, we explore the effect of perturbing a model's weight space in the direction of a subnetwork. Moving the TMS-in-parallel model in the direction of a single subnetwork at a time has a limited effect on the predictinos of most samples, but does result in a significant change in the model's predictions for samples where we expect the subnetwork to be active. In fact, for TMS-in-parallel, we can successfully fully "turn off" a computation by moving far enough in the direction of the subnetwork (For model's with more complex loss landscapes, turning "off" a computation is not as straightforward, as we will later discuss).

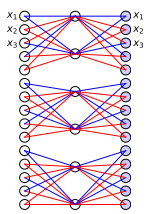
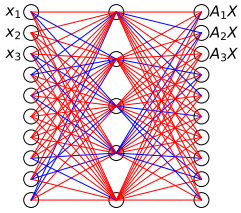
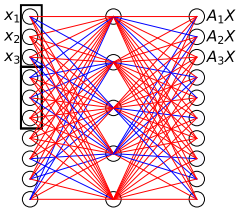
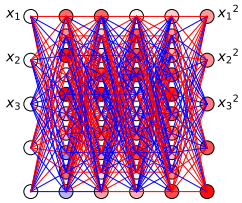
	Toy model of superposition	Circuit Superposition	Higher Rank Circuit Superposition	Complex Loss Landscape
				
	$X \mapsto X$	$X \mapsto AX$	$X \mapsto AX$	$X \mapsto X^2$
Feature Superposition	✓	✓	✓	✓
Circuit Superposition	×	✓	✓	✓
Circuits > rank-2	×	×	✓	probably ✓
Complex Loss Landscape	×	×	×	✓

Table 1: Toy models and their properties (transposed)

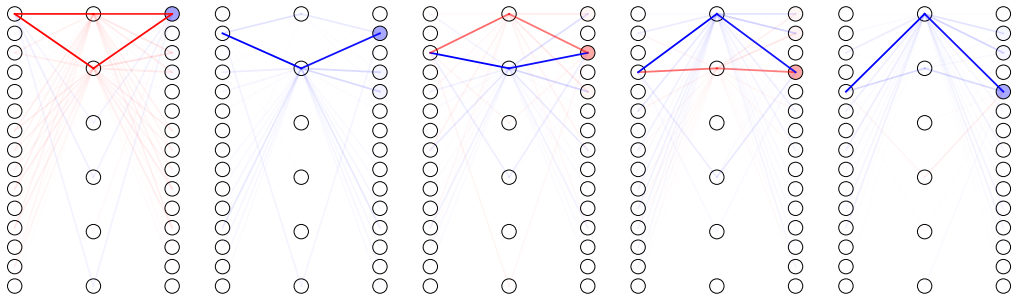


Figure 2: Selected 5 subnetworks that L3D decomposes the TMS-in-parallel model into

3.2. Toy Model of Circuit Superposition

3.2.1. Setup

The toy model of superposition exhibits feature superposition - where features are represented by non-orthogonal activation vectors. However, the sparse circuits in TMS are noteably **not** in superposition - a given weight or parameter is only relevant for a single circuit. It seems highly unlikely that real world model circuits would decompose this way, since learning circuits composed of non-orthogonal parameter vectors would allow the model to put more expressivitiy into a compressed space. We therefore develop a toy model of **circuit superposition** (TMCS) in order to analyze L3D's ability to resolve such circuits.

Our toy model of circuit superposition uses the same architeture and input data distribution as TMS, except our network is trained to predict linear combinations of the input features as its output, rather than reconstructions of each input feature ($X \mapsto AX$). We randomly select values of A between 0 and 3 to generate input, output pairs to train the toy model with.

In this model, if we consider "subnetworks" to be parameter directions involved in inference of a small set of the input data, then we would expect each circuit to be associated with a single input feature. If this is the case, then parameters might be involved in multiple circuits: ${W^{dec}}_{i,1}$ (the set of parameters connecting the hidden nodes to the first output node) will contain information about both $A_{3,1}, A_{2,1}, A_{3,1}$... Put another way, the circuits will interfere with each other - the parameter directions associated with each circuit will be non-orthogonal.

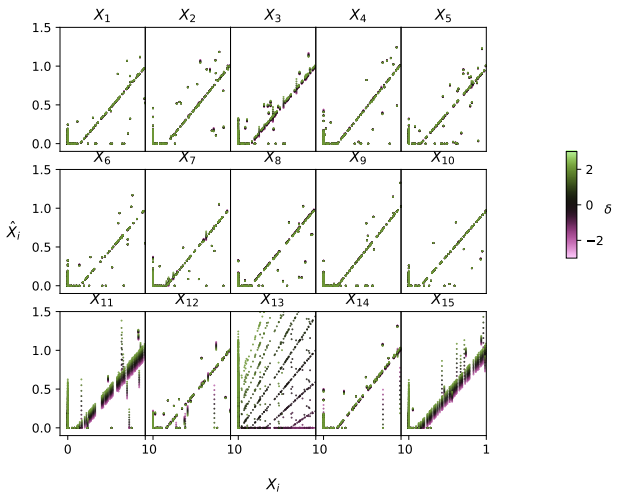


Figure 3: The effect of intervening on the TMS-in-parallel model in the direction of Network$_1$.

3.2.2. Decomposition

Using rank-1 parameter tensor and 15 principal vectors, we decompose TMSC into 3 subnetworks (4) (with reconstruction error). Since each subnetwork theoretically corresponds to the computations involved with a single input feature, we should be able to reconstruct the original A values from each subnetwork. To derive A from each subnetwork, we (1) identify the which column in the subnetwork's W^{dec} direction has the largest norm and then (2) trace the weights of the network through that path. That is for subnetwork k:

$$j^* = argmax_j ||{W^{dec}}_{jk}||_2 \tag{11}$$

$$\hat{\alpha}_{i,j} = {W^{enc}}_{i,j_k} {W^{dec}}_{i,j_k} \tag{12}$$

The parameter vectors are normalized to be unit vectors so we expect them to be a scalar multiple of the true α values. As seen in Figure 4, our derived $\hat{\alpha}$ have a very high correlation to the original α values ($r^2 = 0.9$).

3.3. Higher Rank Circuits

3.3.1. Setup

Because each subnetwork in TMSC traces the path of a single input neuron, the underlying subnetworks inherently have a rank of 1. In order to test the ability of L3D to learn higher rank circuits, and to understand the relationship between circuit rank and circuit superposition, we developed a toy model that inherently uses higher rank circuits (according to our definition of a circuit being sparsely involved in inference). For this model, we use the same set up as TMSC, but we correlate the sparsities of sets of input features. We use 24 input features, and we correlate the sparsities of the input features 1-5, 6-10, etc, ensuring that 1-3 are always active (¿0) or inactive (¡0) together. In this model, circuits should always be associated with groups of 5 input features and so should have a rank of 5.

TEST

3.3.2. Decomposition

Although we expect the model to have 6 subcircuits, we use excess parameter tensors (n=10) in order to allow more flexibility in learning. Futhermore, although we expect the underlying subcircuits to be rank-5, we experiment with using different rank representations to see how well lower-rank parameter directions can represent the model. Interestingly, rank-1 representations of the parameter tensors are able to represent the model nearly as well as rank-5 representations (Figure S10). Using a 10 rank-3 parameter representations, L3D successfully learns a subnetwork corresponding to each of the 6 sets of input features, as well as a number of "dead" or noisy subnetworks (Figure S10). The higher and lower rank decompositions learn similar subnetworks (Figure S10).

3.4. Complex Loss Landscape

3.4.1. Setup

The previous set of toy models all had quadratic landscapes with respect to W^{enc} or W^{dec} where $dMSE/dW_i$ depends on W_i to the first order (or not at all, depending on whether the ReLU has fired). This suggests that the local loss landscape of the models will be a good approximation for much of the global loss landscape. However, we wanted to test the limitations of a L3D on a model with a more complex loss landscape, where we expect circuits to still be in superposition.

We therefore trained a multi-layer model to predict multiple non-linear functions of input features at once. We train a GeLU network for $X_i \mapsto X_i^2$. Specifically we use a network with 4 hidden layers of 5 neurons each, and 10 output and input input neurons. Once again, the input features are sparse, incentivizing the toy model to learn circuits in superposition whose interferences will cause minimal errors on the sparse input distribution.

We expect the model to have 5 subnetworks, one for each input feature. While it is not entirely clear what rank the subnetworks should have, because this is a non-linear model we expect each tensor in the subnetwork to be full rank.

3.4.2. Decomposition

Given that rank-1 parameter tensors seemed to give reasonable representations of subcircuits in the previous model, we again use rank-1 parameter tensors for this model. Instead of varying rank, we experiment with using different num-

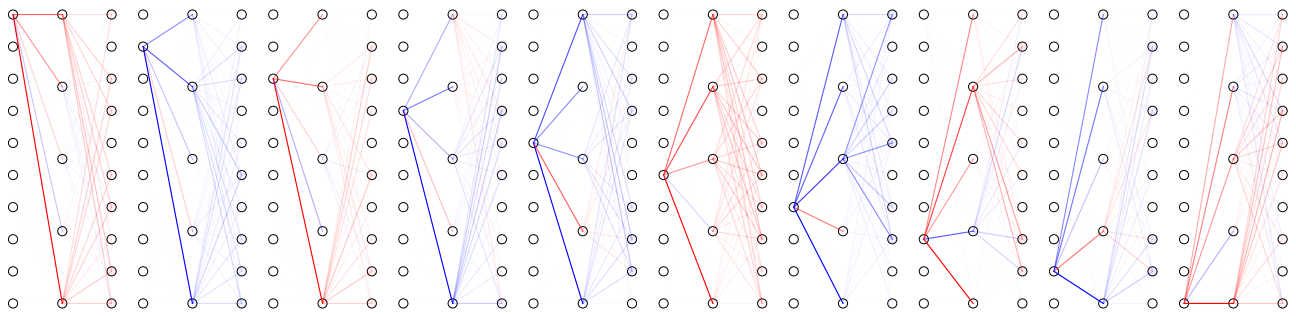


Figure 4: The learned subnetworks of TMCS

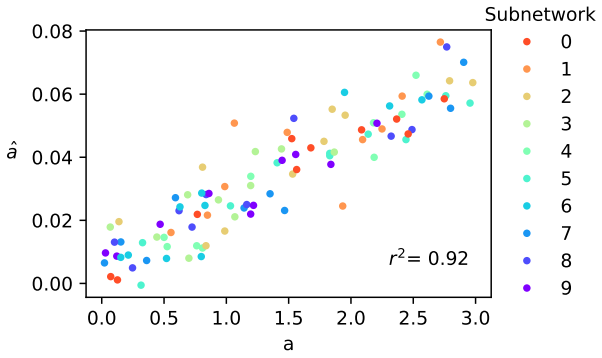


Figure 5: The coefficients derived from the subnetworks compared to actual coefficients

bers of subnetworks to represent our model. We decompose the model into sets of 3, 5, 10, and 15 subnetworks. In the 5-subnetwork decomposition (Figure 7), we see a subnetworks tracing the path of $X_i \mapsto X_i^2$ for each input feature i. In our 3-subnetwork decomposition, L3D still learned subnetworks corresponding to single input features, but can of course only represent 3 out of the 5 inputs. As we add more subnetworks, we are able to successfully learn more expressive decompositions of the model that lower reconstruction error (Figure S10). Interestingly, these subnetworks correspond to a combination of input and output-node specific subnetworks (ie network 0 in the 15-subnetwork decomposition), input-specific subnetworks (ie network 5), and output-specific subnetworks (ie network 8) S10.

3.4.3. Intervention

Intervening on these circuits helps us understand how much local loss landscape is representative of global loss landscape, particularly when it comes to inactive subnetworks remaining inactive as we move through parameter space. If local loss landscape is not representative of global loss landscape in this way, then intervening on a parameter direction of interest will result in a large number of samples being affected, rather than a small subset. 8 shows our results for perturbing parameter directions. Even in this more complex toy model, local loss landscape is a relatively good approximation of the global loss landscape. We can perturb the parameters in a direction of interest and have a large impact on the predictions of that sample and a minor impact on others. If we perturb far enough, we do see some effects on the predictions of other samples, but ratio of change in predictions to the relevant samples to those of the irrelevant samples is very high.

A careful reader may have noticed is that it would be very easy to identify parameter directions in the first or last layer of the $X \mapsto X^2$ model involved in a sparsely active circuit. The circuit for $X_i \mapsto X_i^2$ would just involve all weights connecting input node i to the first hidden layer, and all weights connecting the hidden node i to the output node i, as well as the bias of output node i. In order to make sure L3D is not just learning this trivial solution, we want to make sure the hidden parameter directions it learns are also relevant for circuits. Therefore, from the learned parameter directions, we perform intervention experiments only interveining on the model's middle weights and biases. The results are very similar, where inventions on a parameter direction only affect the output of a subset of samples - showing that the circuits that L3D has learned perform relevant computation even in their middle layers.

8 shows changes in predictions as we move in a single direction in parameter space. We'd also like to undrestand how subcircuits might interact with each other as we move through parameter space. In S9 we perturb multiple subnetworks at once, and measure the new predictions. For the most part, the subnetworks have little inference with each other: the relevant output values for each subnetwork move relatively independently of each other. For some parameter directions, we do see some unexpected interactions, such as when we move in the direction of subnetwork 1 and 2, we see a small effect on the predictions of a few samples not associated with either subnetwork. With larger models, especially those with more modular circuits that are working

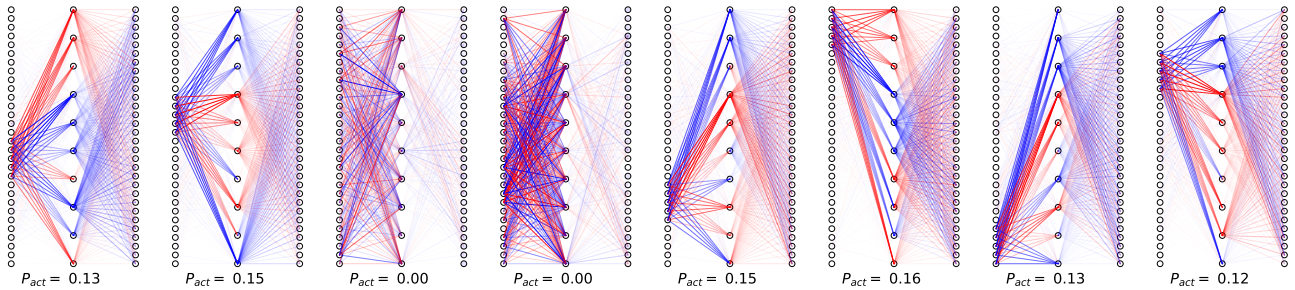


Figure 6: Parameter representations learned by L3D for the high rank circuit decomposition task.

together, we expect this kind of interaction to become more common. We discuss more in the discussion section.

4. Discussion

Frankly crazy that local approximation works so well

4.1. Limitations

4.1.1. Hyperparameter Choice

4.1.2. Low Rank of blocks

4.1.3. Loss Landscape

4.2. Extensions

4.2.1. Finetuning

4.2.2. Identifying Specific Circuits with Constrastive Pairs

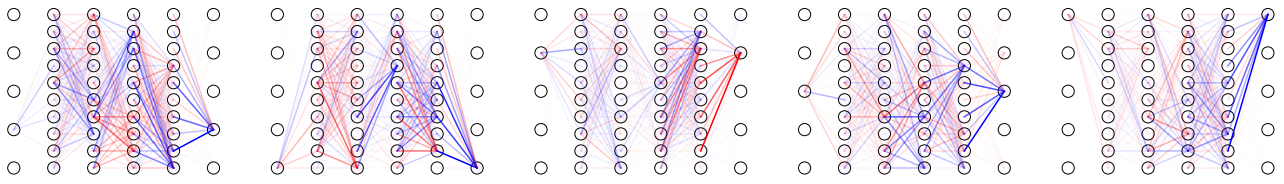


Figure 7: Subnetworks learned by L3D for the $X \mapsto X^2$ model, and effect of intervening on each subnetwork

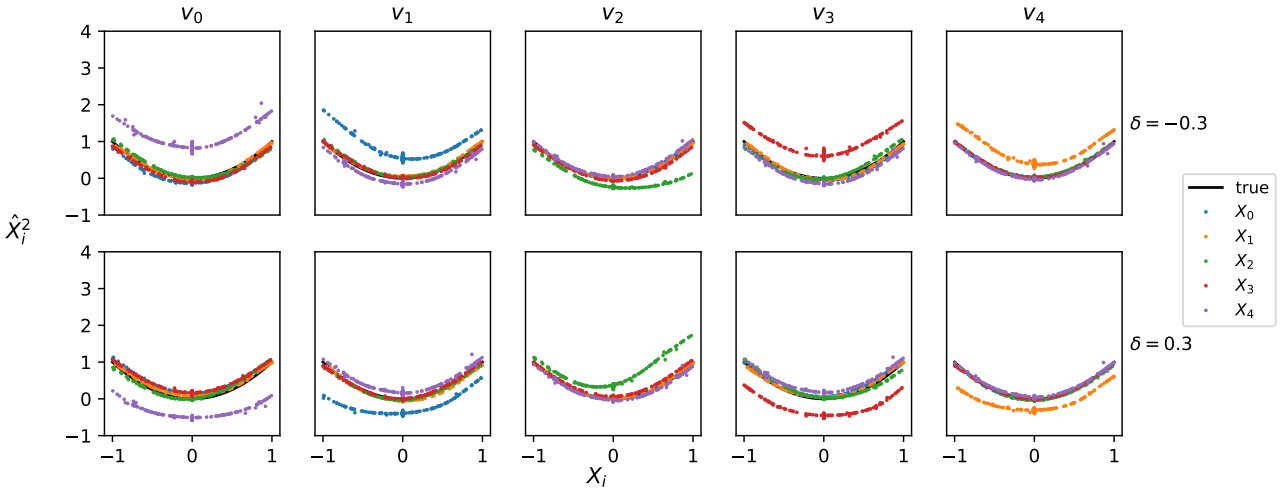


Figure 8: The effect of intervening on each subnetwork

Impact Statement

References

Bereska, L. and Gavves, E. Mechanistic interpretability for ai safety–a review. *arXiv preprint arXiv:2404.14082*, 2024.

Braun, D., Bushnaq, L., Heimersheim, S., Mendel, J., and Sharkey, L. Interpretability in parameter space: Minimizing mechanistic description length with attribution-based parameter decomposition. *arXiv preprint arXiv:2501.14926*, 2025.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Tamkin, A., and Carter, S. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. Accessed: 2025-02-17.

Bushnaq, L., Mendel, J., Heimersheim, S., Braun, D., Goldowsky-Dill, N., Hänni, K., Wu, C., and Hobbhahn, M. Using degeneracy in the loss landscape for mechanistic interpretability. *arXiv preprint arXiv:2405.10927*, 2024.

Cunningham, H., Ewart, A., Riggs, L., Huben, R., and Sharkey, L. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

Davies, X., Langosco, L., and Krueger, D. Unifying grokking and double descent. *arXiv preprint arXiv:2303.06173*, 2023.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Engels, J., Michaud, E. J., Liao, I., Gurnee, W., and Tegmark, M. Not all language model features are linear. *arXiv preprint arXiv:2405.14860*, 2024a.

Engels, J., Riggs, L., and Tegmark, M. Decomposing the dark matter of sparse autoencoders. *arXiv preprint arXiv:2410.14670*, 2024b.

Gao, L., la Tour, T. D., Tillman, H., Goh, G., Troll, R., Radford, A., Sutskever, I., Leike, J., and Wu, J. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Hoogland, J., Wang, G., Farrugia-Roberts, M., Carroll, L., Wei, S., and Murfet, D. The developmental landscape of in-context learning. *arXiv preprint arXiv:2402.02364*, 2024.

Lindsey, J., Templeton, A., Marcus, J., Conerly, T., Batson, J., and Olah, C. Sparse crosscoders for cross-layer features and model diffing. *Transformer Circuits Thread*, 2024.

Matena, M. and Raffel, C. Npeff: Non-negative per-example fisher factorization. *arXiv preprint arXiv:2310.04649*, 2023.

Merullo, J., Eickhoff, C., and Pavlick, E. Talking heads: Understanding inter-layer communication in transformer language models. *arXiv preprint arXiv:2406.09519*, 2024.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Pascanu, R. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2013.

Sharkey, L., Chughtai, B., Batson, J., Lindsey, J., Wu, J., Bushnaq, L., Goldowsky-Dill, N., Heimersheim, S., Ortega, A., Bloom, J., et al. Open problems in mechanistic interpretability. *arXiv preprint arXiv:2501.16496*, 2025.

Wang, G., Farrugia-Roberts, M., Hoogland, J., Carroll, L., Wei, S., and Murfet, D. Loss landscape geometry reveals stagewise development of transformers. In *High-dimensional Learning Dynamics 2024: The Emergence of Structure and Reasoning*, 2024.

Watanabe, S. Algebraic information geometry for learning machines with singularities. *Advances in neural information processing systems*, 13, 2000.

Watanabe, S. Algebraic geometry of singular learning machines and symmetry of generalization and training errors. *Neurocomputing*, 67:198–213, 2005.

Watanabe, S. Almost all learning machines are singular. In *2007 IEEE Symposium on Foundations of Computational Intelligence*, pp. 383–388. IEEE, 2007.

Wei, S., Murfet, D., Gong, M., Li, H., Gell-Redman, J., and Quella, T. Deep learning is singular, and that's good. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):10473–10486, 2022.

A. Definitions

B. Additional Derivations

C. SupplementalFigures

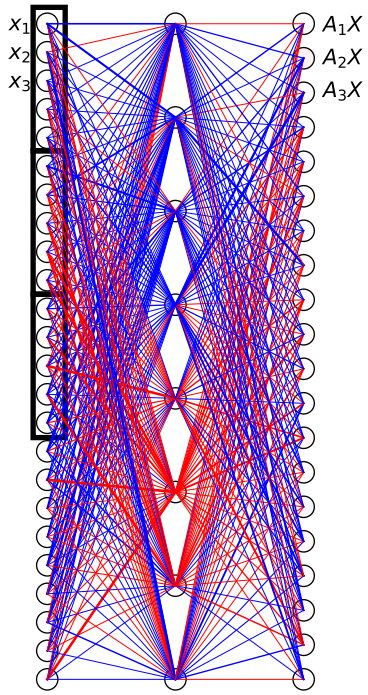


Figure S1: Full architecture of high rank circuit toy model.

CIRCUIT DECOMPOSITION RANK FIGURE

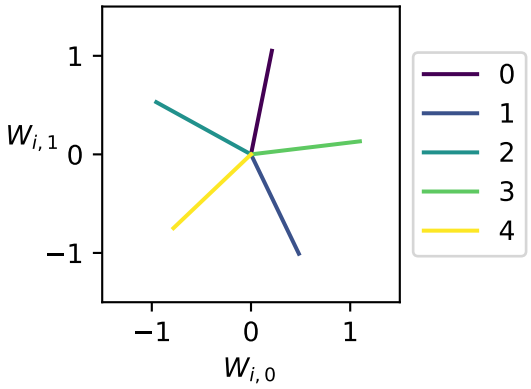


Figure S2: Encoder directions in TMS

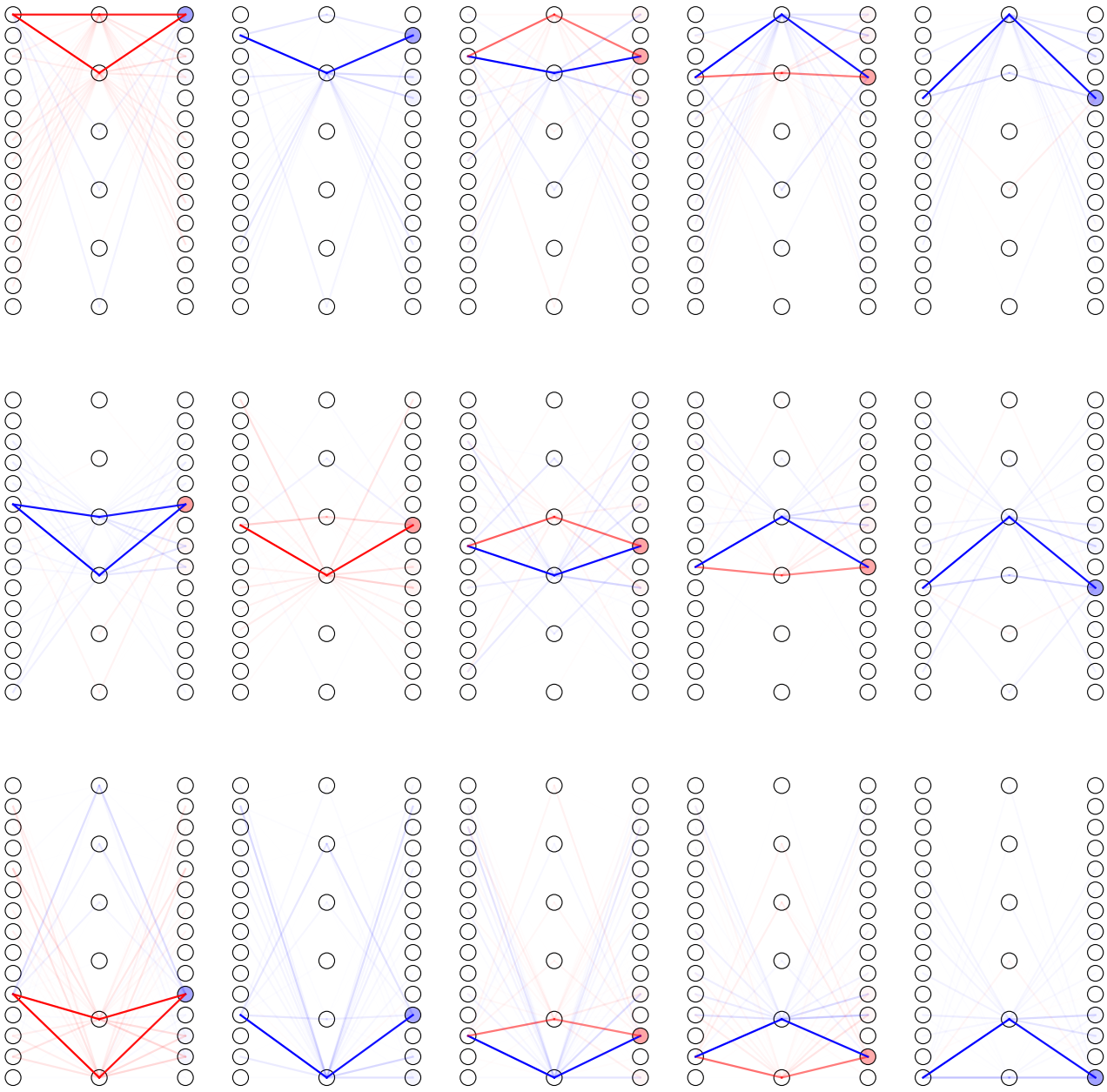
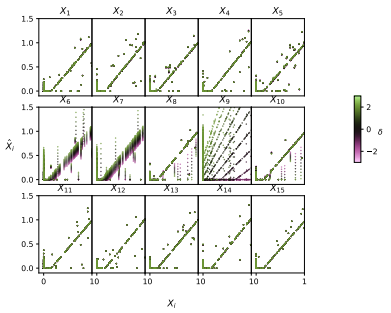
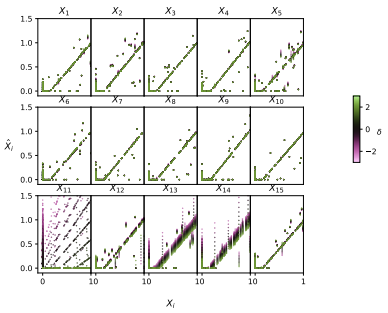


Figure S3: All subnetworks for the TMS decomposition

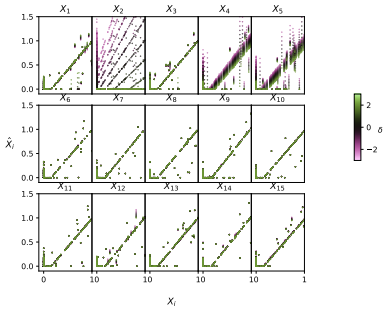
Figure S4: Effect of intervening on the first 5 subnetworks of the TMS-in-parallel model.



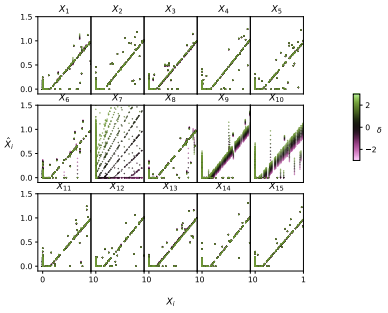
(a) Subnetwork 1



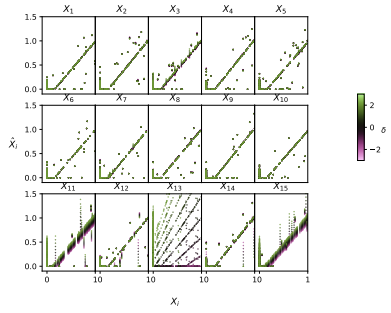
(b) Subnetwork 2



(c) Subnetwork 3



(d) Subnetwork 4



(e) Subnetwork 5

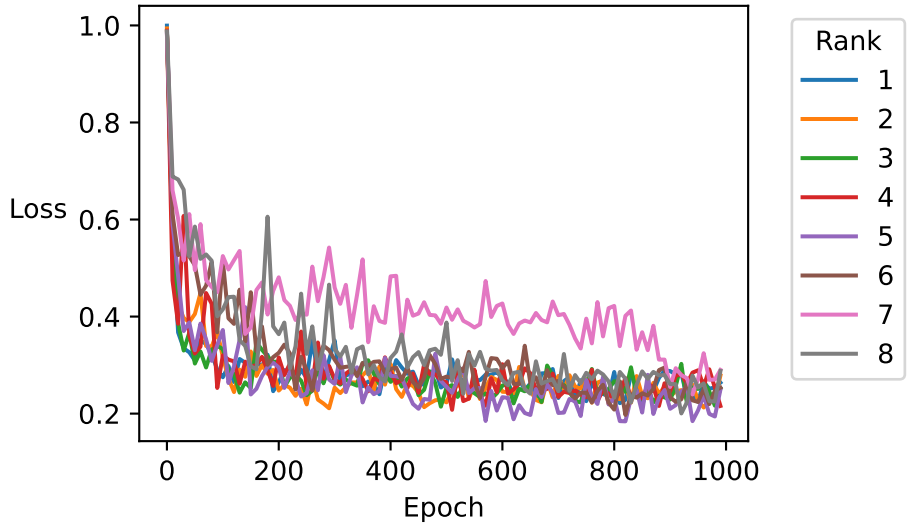


Figure S5: Loss vs Rank

Figure S6: Decomposition of the high rank circuit model using different rank parameter directions

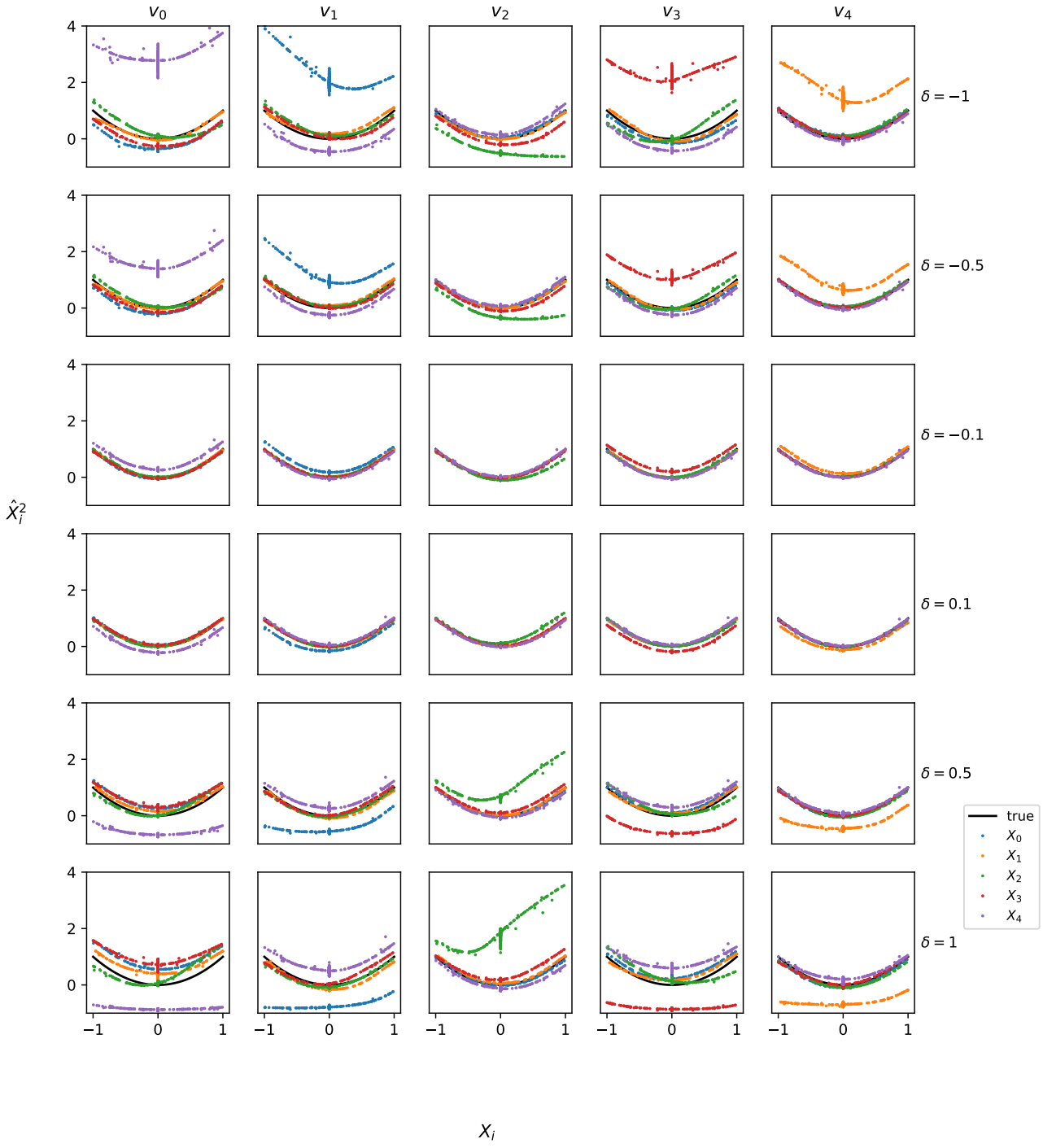


Figure S7: Effects of intervening on each of the 5 subnetworks of the $X \mapsto X^2$ model.

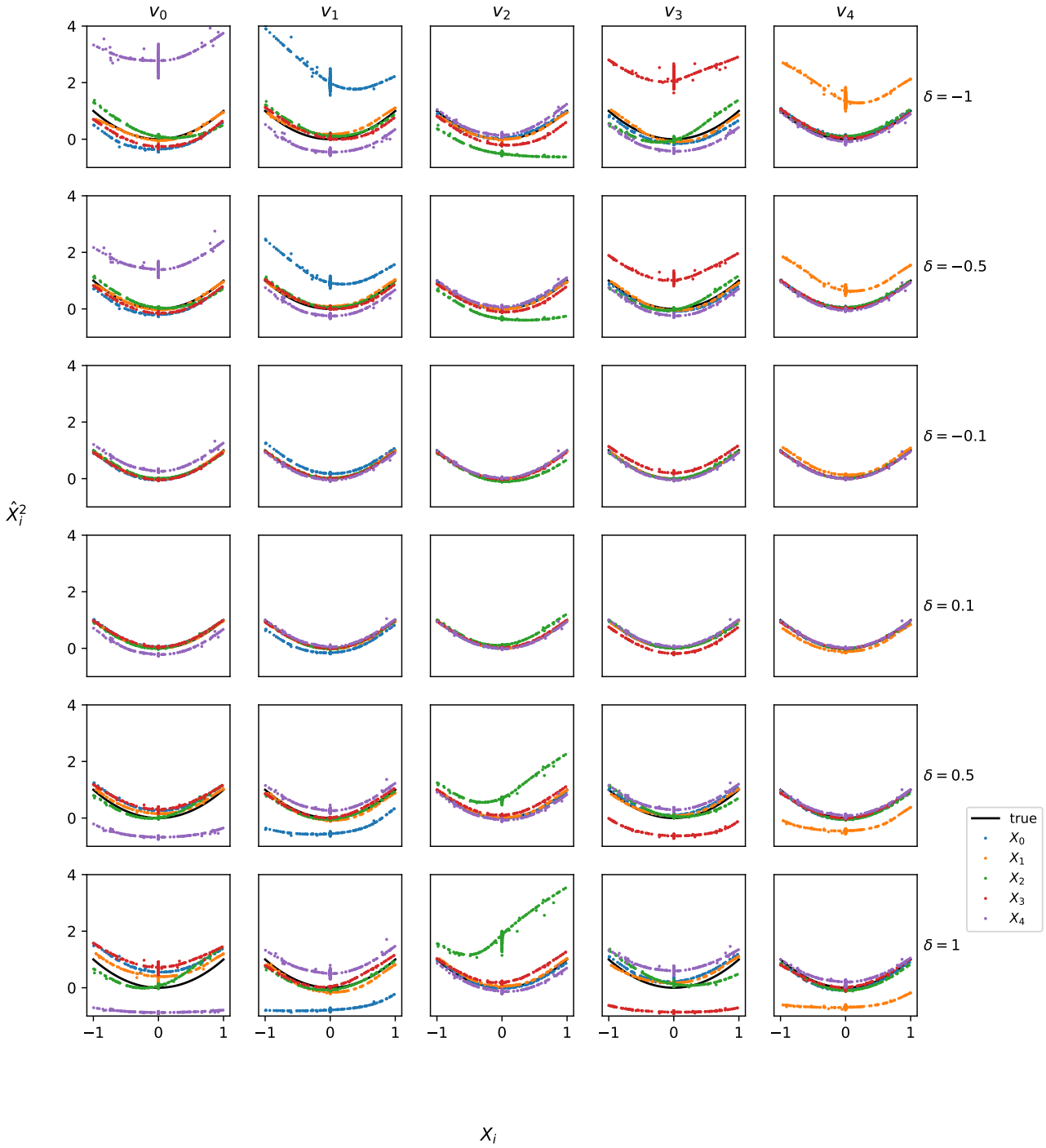


Figure S8: Intervening on just the weights and biases of the middle hidden layers in the $X \mapsto X^2$ network.

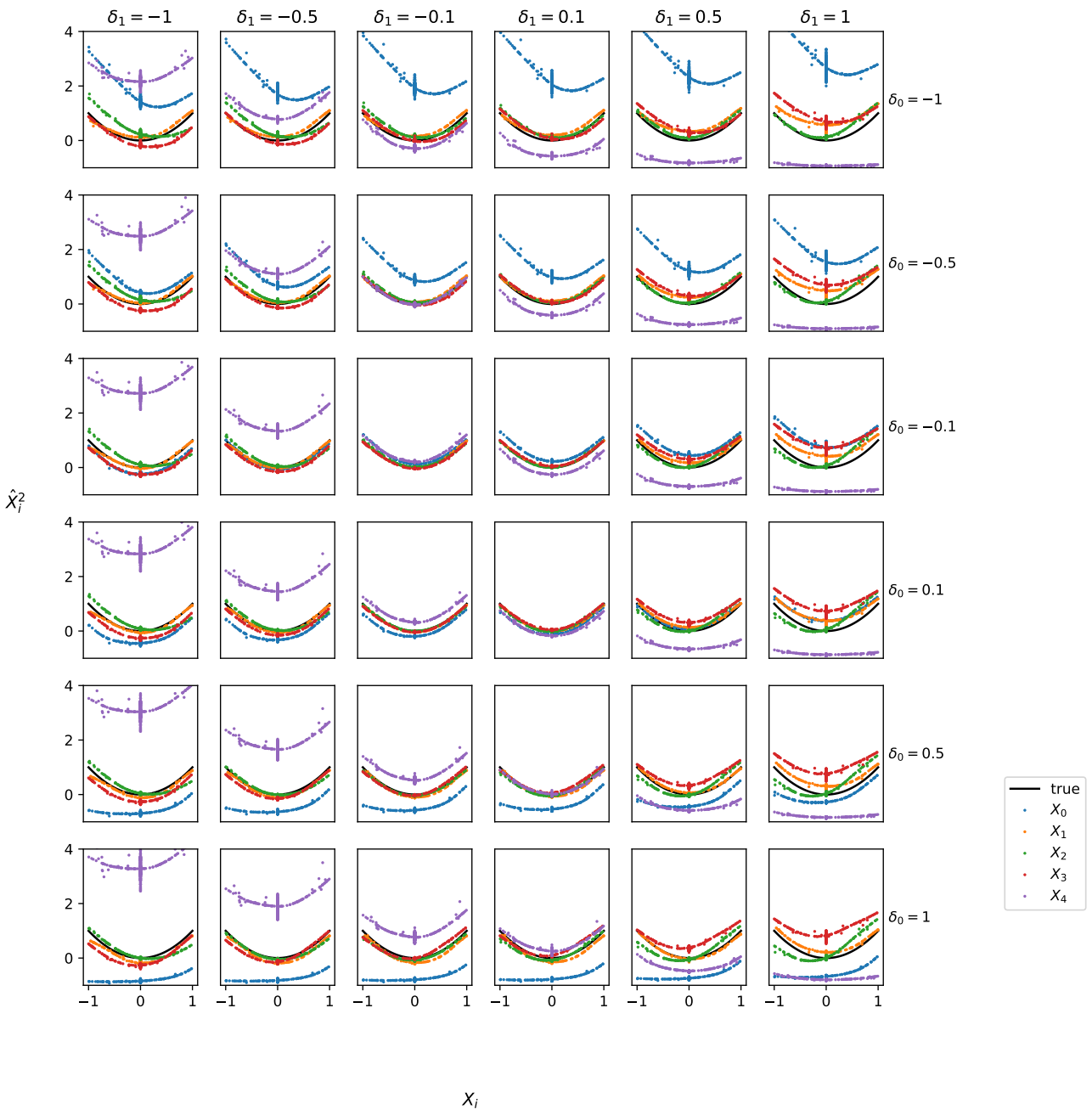


Figure S9: Intervening on multiple subnetworks at once.

Figure S10: Decompositions when using different numbers of subnetworks

Figure S11: Training loss vs number of features for the $X \mapsto X^2$ model, and decompositions for a 3- and 5- subnetwork decomposition.