
Identifying Sparsely Active Circuits Through Local Loss Landscape Decomposition

Anonymous Authors¹

Abstract

1. Background

Mechanistic interpretability aims to uncover the internal mechanisms responsible for the so-called black box behavior or large models so that developers can better understand, intervene on, and align models (Bereska & Gavves, 2024). One goal of the field is to be able to decompose model behavior into subcomponents that are less complex and more human interpretable but that still fully express model behavior. The most popular method in this space are Sparse Autoencoders (SAEs) (Gao et al., 2024; Cunningham et al., 2023; Bricken et al., 2023) which have been successful in resolving compressed features from the activation space of a model. 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.

1.1. Limitations of Activation Space-Based Methods

However, decomposing the activation space of a model has various limitations. First, current SAE paradigm assumes that model’s represent features linearly in a single layer’s activation space. Growing evidence suggests that even architectures designed to encourage linearity, such as the residual stream in a transformer, may in fact encode non-linear features. (Engels et al., 2024a;b). Additionally, 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). Both of these limitations become even more pronounced in models with a less clearly defined read/write stream, such as diffusion models, recurrent networks, adversarial models, and RL models (Pascanu, 2013; Goodfellow et al., 2014; Ho et al., 2020; Mnih et al., 2015).

1.2. Loss Landscape Geometry

An alternative idea is to decompose model’s through their loss landscapes. Parameters are the fundamental objects updated during training, capturing information from data in a way that persists beyond a single inference step. Singular learning theory describes how the structure of high-dimensional parameter space influences generalization (Watanabe, 2007; 2000; 2005; Bushnaq et al., 2024; Davies et al., 2023), and developmental interpretability extends the work of SLT by characterizing phases of the training process. (Wang et al., 2024; Hoogland et al., 2024).

A key insight of SLT is that large models are highly degenerate (Wei et al., 2022) in the parameter space. Not only do modern architectures have inherent isotropies, but they are also degenerate with respect to the training data distribution. Models can have many different parameter configurations that result in minimum loss. In fact, gradient descent tends to converge on configurations that are at the intersection of many of these degenerate directions, called singularities. Our work takes this hypothesis one step further: if in relation to the loss on the full training distribution, models are highly singular then in relation to a subset of the training data, they are probably even more singular.

1.3. Paired Attribution Methods

The other phenomenon our method relies on is that in high-dimensional landscapes, local attribution methods seem to be good approximations of global relationships between pairs of samples. For example, attribution patching can successfully change the output of a model by targetting specific activations, determined by the first order gradients of paired outputs (Nanda, 2023; Kramár et al., 2024; Syed et al., 2023) Similarly, steering vectors, which are derived from differences in activations from paired samples, can effectively guide models toward specific behaviors even when applied beyond the original magnitude of activation differences (Turner et al., 2023; Subramani et al., 2022). Local and global attribution methods are good approximations of each other, especially when considering the space between paired outputs.

055 1.4. Loss Landscape Decomposition

056 Our goal in this paper is to identify directions in parameter
 057 space that correspond to computationally relevant subne-
 058 tworks.

059 To our knowledge, there have only been two prior methods
 060 attempting to decompose parameter space for interpretability.
 061 An earlier work(Matena & Raffel, 2023), decomposes
 062 parameter space by computing principal directions of a per-
 063 sample Fisher Information matrix to resolve meaningful
 064 features. A recent method, Attribution Parameter Decom-
 065 position (APD) (Braun et al., 2025) used a bi-optimization
 066 approach to identify subnetworks where (1) the sum of sub-
 067 network weights is close to the original model parameters,
 068 and (2) for any given input, the sum of the topk-attributed
 069 networks has low behavioral loss when compared to the
 070 original model.

071 Both of the approaches work with behavioral loss - seeking
 072 to understand circuits that when perturbed or ablated, would
 073 change a sample's overall predictions. Our method uses
 074 a slightly different approach, constraining the problem in
 075 such a way that allows us to work in first-order gradient
 076 space. When decomposing a model, we decidedly are
 077 only interested in circuits that move a model's predictions
 078 within the original subspace the outputs existed in. This is
 079 a more constrained problem than identifying circuits that
 080 merely change model output - we wish to identify to find
 081 circuits that change model outputs in the direction of another
 082 theoretical sample within the training distribution.

083 Our object of interest is therefore the gradient (with respect
 084 to the model parameters) of the divergence between a sam-
 085 ple's output and a reference output, sampled from our train-
 086 ing distribution. We seek to identify low-rank directions in
 087 parameter space such that for any pair of samples, a small
 088 number of these directions can be used to reconstruct their
 089 gradient of output divergence. We call our decomposition
 090 method Local Loss Landscape Decomposition (L3D). In
 091 this work, we first describe the mathematical foundation of
 092 our approach. We then develop progressively more complex
 093 toy models to measure the efficacy of L3D and quantify its
 094 limitations.

095 Local loss landscape decomposition (L3D) aims to decom-
 096 pose a model's gradients of loss between output pairs with
 097 respect to parameter space 1. We seek to identify directions
 098 in parameter space such that (1) the set of directions can
 099 approximately reconstruct any gradient of paired sample
 100 divergences, and (2) for any given pair of samples, an even
 101 smaller subset of directions can be used in reconstructing
 102 this gradient. To reduce computational complexity, we learn
 103 low-rank representations of these parameter directions. In
 104 this work, we first describe the mathematical foundation of
 105 our approach. We then develop progressively more complex

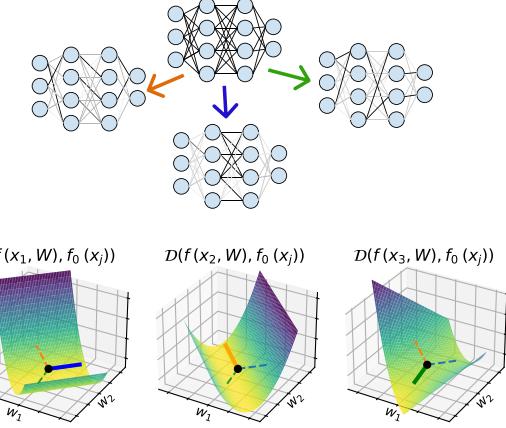


Figure 1: Decomposing a loss landscape into a set of parameter directions , where (1) the set of directions can approximately reconstruct any per-sample-pair gradient of divergence and (2) for any given sample, the curvature in most of the subnetworks directions is approximately zero.

toy models to measure the efficacy of L3D and quantify its limitations.

2. Methodology

In the next sections, we will formally setup our decomposition problem (Sec. 2.1), define the criteria we will use for our subnetworks/parameter directions (Sec. 2.2), describe how to efficiently decompose parameters into these directions (Sec. 2.3), walk through our training algorithm (Sec. 2.4), and then explain how to use these decompositions to intervene on a model's behavior (Sec. 2.5).

The terms parameter directions, subnetworks, and circuits can all be used interchangeably, and will refer to the directions in parameter space that we wish to learn.

2.1. Set up

Consider a model f that takes an batch of inputs X and parameter values of W , and outputs a batch of outputs.

$$f(x, W) : R^{n_s \times n_f} \mapsto R^{n_s \times n_o} \quad (1)$$

where n_s is the number of samples in the batch, n_f is the length of each input vector, and n_o is the length of each output vector.

Our project rests on the following hypothesis:

1. For a given input, there are many components of a model that are not involved in inference. Changing

110 these parameters will not affect change the model's
111 output.

- 112 2. For the same given input, there are a smaller set of
113 components in a model that are highly involved in
114 inference. Changing these parameters will change the
115 model's output.
- 116 3. We are interested in parameter directions that meaning-
117 fully change a model's output. We'd like to identify di-
118 rections that when perturbed, do not break the model's
119 output by moving it completely outside of realm of
120 realistic outputs, but rather turn decrease or enhance
121 the power of a specific computational circuit.

2.2. Divergences of Paired Outputs

To put point (3) a different way, intervening in a certain parameter direction should not move our outputs too far away from a reference output distribution. Therefore, we will be decomposing gradients of divergences between pairs of outputs - with the aim of finding directions that move the model's output within the same subspace that the reference output distribution lies.

Divergence of a pair of outputs can be defined as:

$$\nabla_W D(f(x_i, W), f(x_r, W_0))|_{W=W_0} \quad (2)$$

where $x_i, x_r \in X$

Here, 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 model's parameters fixed. Our toy models, which are regression-type models, so we use MSE as divergence.

Wee will abbreviate $f(x_r, W_0)$ as $f(x_r)$ and the expression in Eq. 2 as $\nabla_W D$.

2.3. Sparse Principal Directions

We want to transform our gradient into directions in parameter space, where each sample's gradient can be written as a linear combination of a small set of these components. We will do this by learning transforms $V^{in} \in R^{n_v \times n_w}$ and $V^{out} \in R^{n_w \times n_v}$ where n_w is the number of parameters in the model, and n_v is the number of components (subnetworks) we wish to use to represent the parameter space.

V^{in} effectively transforms a gradient from the parameter space to the subnetwork space, so that:

$$\nabla_V D = V^{in} \nabla_W D \quad (3)$$

We want to find V^{in} and V^{out} such that for any given pair of samples a small subset of subnetworks can approximately

reconstruct the gradient of divergence.

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

where $\mathcal{K} = \text{argTopFrac}_k \text{abs}(\nabla_{v_k} D)$

argTopFrac relies on a hyperparameter that controls the number of components we wish to use to reconstruct each sample. In practice, we use a batchTopK (Bussmann et al., 2024) and a fraction rather than an absolute number. If we use a batchTopFrac of .1 it means that we select the top 10% of $\nabla_V D$ magnitudes over v_k and x_i to reconstruct our batch of gradients.

2.3.1. LOW RANK PARAMETER DIRECTIONS

Learning a set of full rank parameter parameter directions would be extremely expensive (we would be learning $n_w n_v$ values). We also expect that modular, sparsely active circuits would be lower rank than their full-model counterparts (). Therefore, we use low-rank representations of our V^{in} and V^{out} , and correspondingly learn low-rank circuits. We describe the procedure for doing this in B.1.

2.4. Training

We wish to learn the decomposition-related transforms V^{in} and V^{out} that minimize the topK reconstruction loss of our divergence gradient described above. We use a (normalized) L2 norm loss.

$$L = \|\nabla_W D - V^{out}_{:, \mathcal{K}} V^{in}_{\mathcal{K}, :} \nabla_V D\|_2 / \quad (5)$$

For each batch of samples, we randomly select a reference sample x_r to be paired with each sample x_i in the batch. We then compute the gradient of divergence of x_i and x_r at the model's current parameters W_0 . We transform that gradient into the subnetwork space using V^{in} , and compute the topK components. We transform those components back into the original parameter space using V^{out} , and compute the loss between the reconstructed gradient and the original gradient. We apply a learning update to V^{in} and V^{out} with the goal of minizing this loss. We also normalize V^{out} to be a unit vector after each update.

2.5. Measuring and Intervening

Our learned subnetworks will just be the rows of V^{out} , reorganized into the same tensor structure as W . After identifying subnetworks, we may want to intervene on a specific circuit.

If we wish to intervene using a single subnetwork, we can update the parameters in by moving them in a scalar factor (δ) of that unit direction. To tune our model in the direction of subnetwork v_i and compute predictions on x , we evaluate:

Algorithm 1 Training Algorithm

```

165
166 1: for each epoch do
167 2:   for each  $X$  do
168 3:     for each  $x_i \in X$  do
169 4:       Randomly select  $x_r \in X$ 
170 5:        $\nabla_w D_i = \nabla_w D(f(x_i, W), f(x_r))|_{W=W_0}$ 
171 6:     end for
172 7:      $\nabla_v D = V^{in} \nabla_w$ 
173 8:      $\tau = \text{topK}(\text{abs}(\nabla_v D))$ 
174 9:      $\hat{\nabla}_w D = V^{out} (\nabla_v D \odot (\text{abs}(\nabla_v D) > \tau))$ 
175 10:     $L = \|\nabla_w D - \hat{\nabla}_w D\|_2$ 
176 11:     $L.\text{backward}()$ 
177 12:    Update  $V^{in}$  and  $V^{out}$ 
178 13:    Normalize  $V^{out}$  to be unit vectors.
179 14:  end for
180 15: end for

```

$$f(x, W + \delta v_i) \quad (6)$$

Going one step further, we may want to finetune our model, but constrain our finetuning to specific circuits for efficiency or performance reasons. While we do not use finetuning in this work, we describe what the process of finetuning would look like in Appendix B.2.

Additionally, we may want to identify which subnetworks were most relevant to the predictions of various samples, or identify which samples most relied on a given subnetwork. We describe this process in Appendix B.2.1

3. Results

Parameter space decomposition is extremely new, and there are not well-defined benchmarks by which to test our method. For this work, we focus on developing toy models that involve well-defined subnetworks, and evaluating L3D on decomposing these.

We designed several toy models to test the efficacy of L3D. Our toy models all consist of several well-characterized computations being performed by the same model, with an input space designed to rely on a single or small set of tasks. By designing our toy models this way, we have well characterized ground truth subnetworks by which to compare L3D’s decompositions.

Our toy models progressively test more complex types of circuits. Table 1 describes our 4 toy models and the different attributes of circuitry the are designed to capture. The specific hyperparameters used to train our toy models is described in Appendix B.3, as well as the hyperparamters used for each decomposition in Appendix B.4

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 an autoencoder with a single low-dimensional hidden layer in between and a relu activation at the output (Elhage et al., 2022). The model is trained on a dataset of samples where few features are active at a time, and “superimposes” these features in the hidden layer such that features embeddings in the hidden layer have minimal interference with each other S2. We train a toy model of superposition 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 independent circuit modules.

3.1.2. DECOMPOSITION

We decompose the TMS model into 15 subnetworks, using rank-1 parameter tensors. The network decomposes into subnetworks as we would expect: each subnetwork represents the embedding and reconstruction of a single input element, involving only the weights connecting the relevant input and output, and the final bias. Fig. 2 shows the subnetworks corresponding to each of the first 5 features, and Fig. S3 shows the full decomposition. One thing to note is that 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, the weights and biases in subnetwork 1 are in the opposite direction as the original network (Table 1).

This decomposition resulted in a reconstruction error of .19. 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. We expect decompositions of higher dimensional networks to exhibit less reconstruction error, as the amount of nearly orthogonal parameter vectors (non-interfering) that can be compressed into parameter space scales exponentially with dimension. We see this effect in later toy models.

3.1.3. INTERVENTION

Parameter vectors learned by L3D can be used to intervene on model behavior. 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 (See Appendix B.2). While we do not go the extent of finetuning a model, we explore the effect of perturbing a model’s parameter space in the direction of a subnetwork (by an increment of δ). If subnetworks do in fact represent sparse computations, we hope that intervening on a

Local Loss Landscape Decomposition

	Toy model of superposition	Circuit Superposition	Higher Rank Circuit Superposition	Complex Loss Landscape	
220					
221					
222					
223					
224					
225					
226					
227					
228					
229					
230					
231					
232	Feature Superposi-	$X \mapsto X$	$X \mapsto AX$	$X \mapsto AX$	$X \mapsto X^2$
233	tion	✓	✓	✓	✓
234	Circuit Superposition	✗	✓	✓	✓
235	Circuits > rank-2	✗	✗	✓	probably ✓
236	Complex Loss Land-	✗	✗	✗	✓
237	scape				

Table 1: Toy models and their properties (transposed)

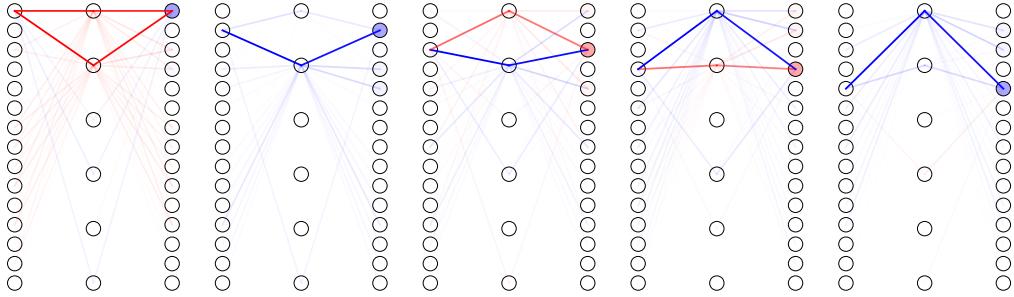


Figure 2: Selected 5 subnetworks that L3D decomposes the TMS-in-parallel model into

subnetwork has a strong effect on the predictions of relevant samples, and no effect on other samples. As shown in Fig. 3, moving the TMS-in-parallel model in the direction of a single subnetwork does in fact achieve just this. Perturbing in the direction of network 1 primarily affects samples where feature 1 was active, with a small effect on the inputs that had interference to feature 1’s embeddings. In fact, for TMS-in-parallel, we can successfully fully “turn off” a computation by moving far enough in the direction of the subnetwork. (Although for models with more complex loss landscapes, turning “off” a computation is not as straightforward, as we will later discuss).

3.2. Toy Model of Circuit Superposition

3.2.1. SETUP

TMS famously exhibits feature superposition - the input elements’ low dimensional embeddings are non-orthogonal. However, the sparse circuits in TMS are notably *not* in superposition - a given weight or parameter is only relevant for a single circuit and circuits. It seems highly unlikely that real world model circuits would decompose this way, since learning circuits composed of non-orthogonal parameter vectors would allow a model to put more expressivity 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. We define circuit superposition as a phenomenon by which subnetworks share parameter elements, and even more generally have non-orthogonal parameter vectors.

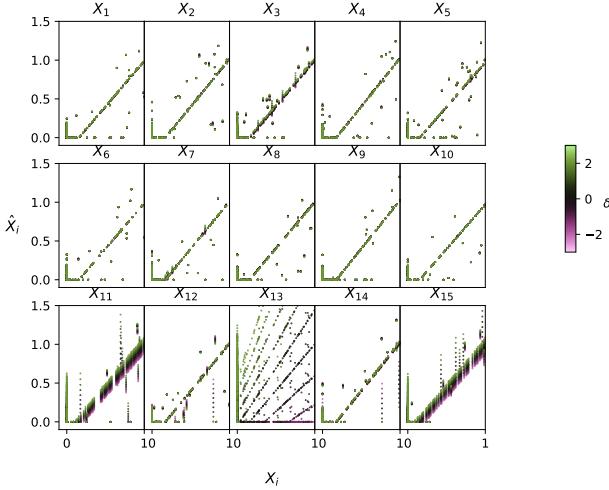


Figure 3: The effect of intervening on the TMS-in-parallel model in the direction of Network₁.

Our toy model of circuit superposition uses the same architecture and input data distribution as TMS, but is trained to predict linear combinations of the input features as its output ($X \mapsto AX$). We randomly select values of A between 0 and 3 to generate input-output pairs to train the toy model with. We use an input with 10 input features, and 10 output features (although such a model does not need to have the same number of input and outputs).

In this model, if we consider "subnetworks" to be parameter directions involved in inference of a small set of samples, then we would expect each circuit to be associated with a single input feature. If this is the case, then parameters would 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 $A_{1,1}, A_{2,1}, A_{3,1}, \dots$. Put another way, the circuits will interfere with each other - parameter directions associated with each circuit will be non-orthogonal.

3.2.2. DECOMPOSITION

We decompose TMCS into 10 subnetworks of rank-1 parameter tensors (4) with a reconstruction loss of .064. The subnetworks look like we expect, with each strongly corresponding to a single input feature.

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 :

$$j^* = argmax_j \|W^{dec}_{j,k}\|_2 \quad (7)$$

$$\hat{a}_{i,j} = W^{enc}_{i,j,k} W^{dec}_{i,j,k}$$

The parameter vectors are normalized to be unit vectors so we expect them to be a scalar multiple of the true A values. As seen in Figure 4, our derived \hat{a} have a very high correlation to the original A values ($r^2 = 0.92$).

3.3. Higher Rank Circuits

3.3.1. SETUP

Because each subnetwork in TMCS traces the path of a single input neuron, the underlying subnetworks should inherently have a rank of 1. In order to test the ability of L3D to learn higher rank circuits, we developed a toy model with inherently higher rank circuits. For this model, we use the same set up as TMCS, but we correlate the sparsities of sets of input features. We use 25 input features, and we filter our data to ensure that input features 1-5, 6-10, etc, are always active (> 0) or inactive (< 0) together. In this setup, circuits should always be associated with groups of 5 input features and so should have a rank of 5.

3.3.2. DECOMPOSITION

Although we expect the model to have 5 subcircuits, we use excess parameter tensors ($n=10$) in order to allow more flexibility in learning. We track the fraction of inputs for which a subnetwork was used in the topK reconstruction (P_{act}) to identify which were "dead circuits", and report the last epoch. Furthermore, 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 (Fig. S5). In Fig. 6, we show the decomposition of a rank-3 decomposition. L3D successfully learns a subnetwork corresponding to each of the 5 sets of input features, as well as a number of "dead". The higher and lower rank decompositions also learn similar subnetworks (Figure S6). When we trained L3D without these additional subnetworks, the reconstruction loss would get caught in a local minima. Similar to training SAEs (Cunningham et al., 2023), having extra degrees of freedom allows for better learning, even if at the end of training the extra subnetworks are never active.

3.4. Complex Loss Landscape

3.4.1. SETUP

The previous set of toy models all had relatively quadratic landscapes with respect to each element of W^{enc} or W^{dec} -

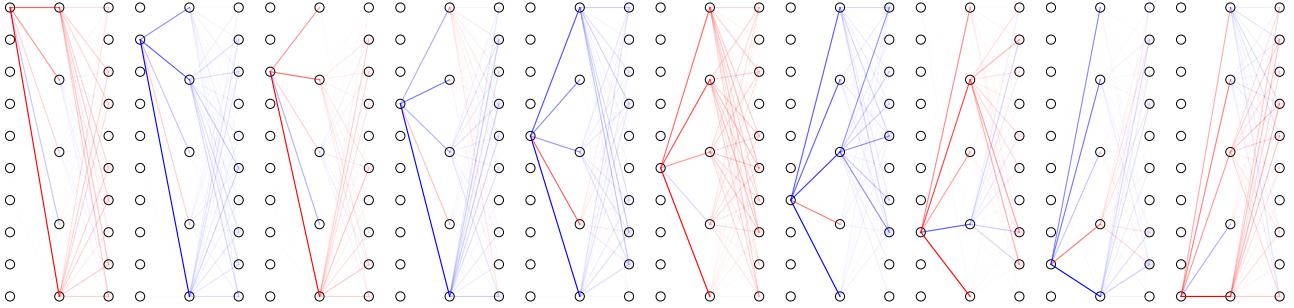


Figure 4: The learned subnetworks of TMCS

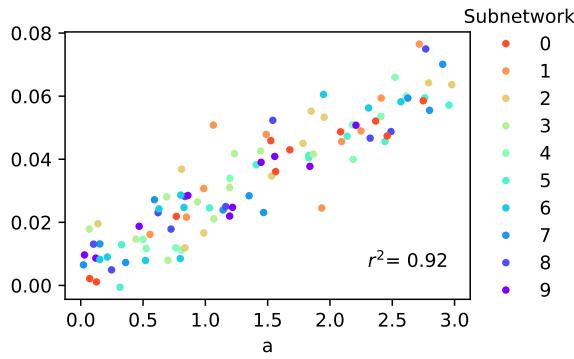


Figure 5: The coefficients derived from the subnetworks compared to actual coefficients

$dMSE/dW_i$ 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 is a good approximation for much of the global loss landscape, and we should expect intervening on a circuit to have well-behaved effects, even at large values of δ (as in Fig. 3).

However, we wanted to test the limitations of a L3D on a model with a more complex loss landscape, especially when it comes to intervening with a subnetwork.

We therefore trained a multi-layer model to predict multiple non-linear functions of input features at once. We train a GeLU network for $X_i \mapsto X_i^2$. We use a network with 4 hidden layers of 5 neurons each, and 5 input and output 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. Although it is less clear what rank the tensors of the underlying circuits should be, we expect that with a non-linear model they will likely be high rank.

3.4.2. DECOMPOSITION

We decompose our model into 5 rank-2 parameter tensors in our decomposition and instead of varying rank, we experiment with using different numbers of subnetworks to represent our model. In the 5-subnetwork decomposition (Figure 7), we see that subnetworks tracing the path of $X_i \mapsto X_i^2$ for each input feature i . However, this decomposition has a relatively high reconstruction error of .32. Much of this is probably because we kept our topFrac constant throughout all our models, using a topFrac of .1 for consistency. With only 5 subnetworks, this means that on average each sample's reconstruction will use 1 subnetwork - limiting the reconstruction error we can achieve.

We also experiment with holding rank constant (we drop to rank-1) and decompose the model into different numbers of subnetworks (3, 5, 10, and 15 subnetworks). 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 reduce reconstruction error (Figure S11). Each decomposition continues to learn input-output pair specific subnetworks, with the larger decompositions resulting in a few more dead subnetworks as well (Fig. 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 truly representative of global loss landscape in this way, then intervening on a single subnetwork should result in only a consistent small number of samples being affected, even if we move very far in that direction. Fig. 8 shows our results for these interventions. Even in this more complex toy model, local loss landscape is a relatively good approximation of the global loss landscape.

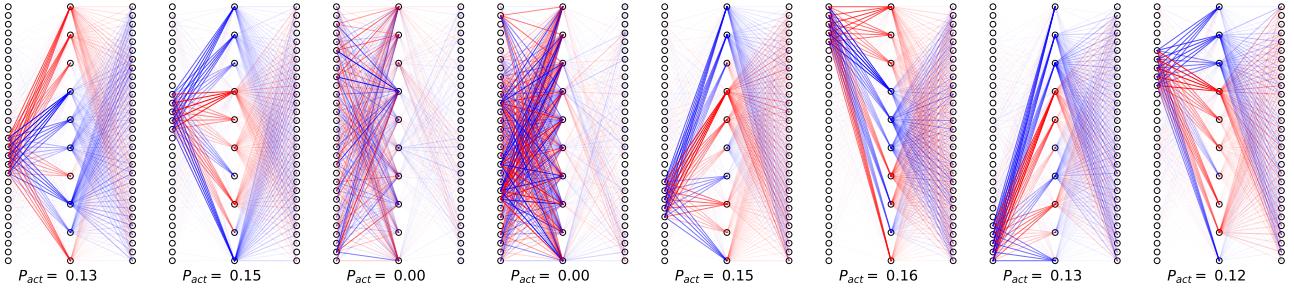


Figure 6: Parameter representations learned by L3D for the high rank circuit decomposition task.

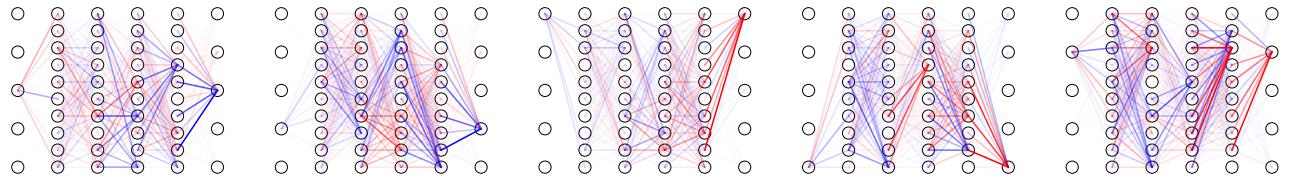


Figure 7: Subnetworks learned by L3D for the $X \mapsto X^2$ model, and effect of intervening on each subnetwork

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 (Fig. S7), we do begin to see 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 test if the hidden parameter directions it learns are also relevant circuits. Therefore, we perform the same intervention experiments but only intervening with the subnetwork components related to the model's hidden layers' weights and biases (S8). The results are very similar, where interventions 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 also wanted to understand 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 together, we expect this kind of interaction to become more common. We discuss more in the discussion section.

4. Discussion

L3D is one of the earliest parameter-based decomposition methods. For this reason, we have focused our work on demonstrating the fundamentals of L3D on toy models, and showcasing its promise with more complex models. Here we discuss what we believe are simple improvements to L3D that could enhance its performance and real-world use cases to which to extend L3D. Finally, we discuss unresolved challenges and limitations of L3D.

4.1. Simple Improvements

In this work, we did not focus on optimizing L3D, and we chose nearly identical hyperparameters for all of our decompositions.

Hyperparameter Choice

For all of our toy model decompositions, we always chose $\text{topFrac} = .1$, even when it was clear that certain toy models should have larger numbers of subnetworks activated per sample than others (For example, the $X \rightarrow X^2$ model with 5 inputs and 5 outputs, decomposed into 5 networks, should probably have $\text{topFrac} \geq .2$). Too low of topFrac choice is likely responsible for the high reconstruction loss

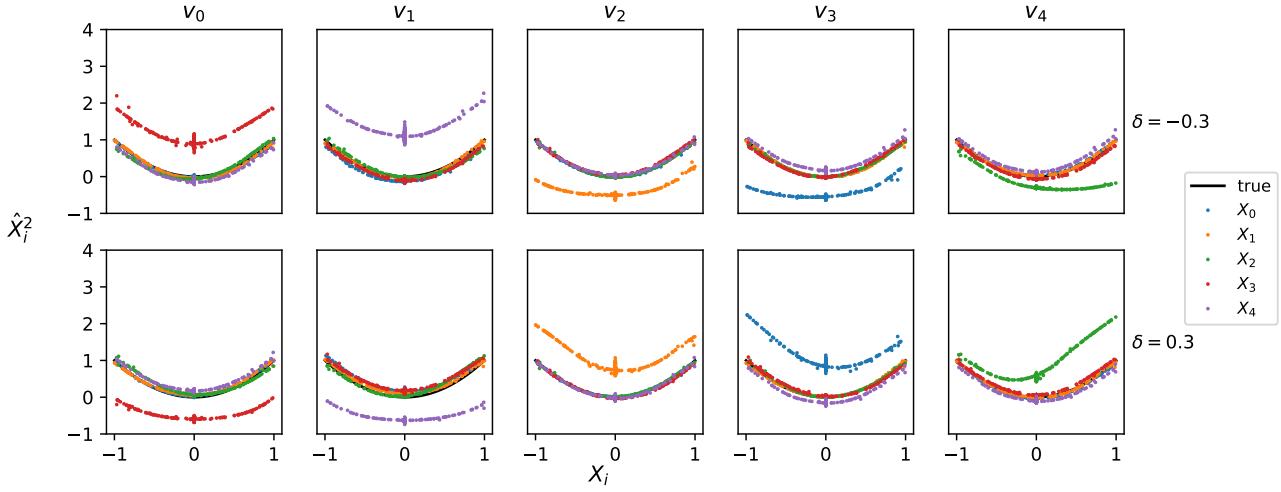


Figure 8: The effect of intervening on each subnetwork

of some of our models. Similarly, we always chose the ranks of the subnetworks somewhat arbitrarily. There has been some research done into understanding the relationship between rank and superposition budget (), and a better understanding of this relationship could help us choose better hyperparameters for L3D.

Flexible Rank Choices

Relatedly, the way L3D makes circuit decomposition computationally feasible is to represent each tensor in the subnetwork as a low-rank tensor. For simplicity, we use the same rank for all tensors. Theoretically, there could be more flexibility across tensors, and we could use different ranks for different blocks of the subnetwork if we believed certain parts of the model were composed of fundamentally higher rank circuits.

4.2. Extensions

There are also some higher effort extensions to L3D that may give it more real-world relevance.

Finetuning Our intervention experiments showed promise subnetworks of L3D could be perturbed in ways that only affect the predictions of relevant samples. As we described in Section B.2, this could be taken one step further by finetuning a model on a specific set of parameter directions. Using L3D networks, we could finetune a model on a specific set of parameter directions by freezing the current set of weights and learning an adaptor consisting of linear combinations of the subnetworks of choice. This could also benchmark the intervention capabilities of L3D versus other mechanistic intervention strategies such as SAE-derived steering vectors. For example, we might use L3D to identify various subnetworks involved in syncophancy, refusal, and other undesired

behaviors. After collecting curated data with the goal of finetuning away such behaviors, we could finetune L3D only in the direction of the behavior-related subnetworks and test how well the model achieves our desired output compared to other intervention strategies.

Identifying Specific Circuits with Contrastive Pairs We developed this method as an unsupervised decomposition method, with goals comparable to those of SAEs. However, the methods of L3D could be easily modified to use supervised signals to identify specific circuits of interest. Rather than using gradients of divergence of random pairs, we could decompose gradients of divergence between curated pairs of samples that isolate a behavior of interest. In this context, we could probably not require the topK constraint, but instead look decompose the network into a very small number of low-rank subnetworks involved in our behavior of interest.

4.3. Challenges

Although many of the improvements and extensions of L3D are highly addressable, we think there are some fundamental challenges with parameter-based decomposition methods that may not be easily resolved.

Local Loss Landscape L3D’s algorithm hinges on the somewhat surprising phenomenon that local gradient approximations work well as attribution methods. They clearly work to a great extent in the toy models we used for L3D and for the circuits we found in our real world models. However, do they work for all circuits? In our work, we use a randomly selected sample to be our “reference” output with which to compute divergence gradient. By using a randomly selected sample, rather than a single “reference output” such as the mean of the output distribution, we hope

495 that the random noise in the reference sample will average
496 out the effects of any non-convexity in the loss landscape.
497 However, perhaps, even in this setup, perhaps there are pa-
498 rameter directions that are highly non-convex and will be
499 difficult to perform local attribution on. Quantifying differ-
500 ent types of "dark matter" of parameter decomposition by
501 analyzing reconstruction loss (similar to recent work done
502 on SAEs) could better help us characterize these limitations.
503

504 **Relationship to overparameterized models** Going one
505 step further, we suspect that the reason local attribution
506 methods work so well is because large models are proba-
507 bly overparameterized. By this we mean that there exists a
508 much smaller network that performs identically to a model,
509 but higher dimensional models learn better, as they have less
510 local minima to get stuck into during the learning process
511 (Kawaguchi, 2016; Choromanska et al., 2015; Dauphin et al.,
512 2014; Soudry & Hoffer, 2017). Larger models may have
513 wider loss basins, or more degeneracies near their global
514 minima (Keskar et al., 2016; Sagun et al., 2017), making
515 local attribution methods less likely to break down as we
516 move through parameter space. If in the future, an learning
517 algorithm is developed that has fundamentally different
518 limitations that stochastic gradient descent and its relatives,
519 we might lose this property. Moreover, circuit activations
520 might no longer be sparse. A new learning process might
521 be able to compress subnetworks in such a way that subnet-
522 works have very high levels of interference with each other -
523 removing the degeneracy assumption that underlies L3D.

524 **Interpretation of a circuit**

525 Finally, we should address the definition of "circuits". It is
526 still not well agreed upon what a "feature" is in relation to
527 large networks, and the definition of what should constitute
528 a circuit or subnetwork is even less clear. Is our definition
529 of a circuit - sparsely active subnetworks that can move
530 outputs within the original output distribution - too restric-
531 tive? If there is a circuit that is relevant to every output, a
532 sort of "scaffolding" for more specific circuits - should it
533 be included in the decomposition? If, after identifying the
534 strcutre of subnetworks, we cannot interpret it beyond a
535 description of its end results, are circuits any more infor-
536 mative than they features they are computing? If parameter
537 decomposition is a viable strategy for understanding and
538 intervening with large networks, these questions will be
539 important for the mechanistic interpretability community to
540 address.
541

542
543
544
545
546
547
548
549

550 **5. Impact Statement**

551 This paper presents work whose goal is to advance the field
552 of Machine Learning. There are many potential societal
553 consequences of our work, none which we feel must be
554 specifically highlighted here.

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

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 Hobbahn, M. Using degeneracy in the loss landscape for mechanistic interpretability. *arXiv preprint arXiv:2405.10927*, 2024.
- Bussmann, B., Leask, P., and Nanda, N. Batchtopk sparse autoencoders. *arXiv preprint arXiv:2412.06410*, 2024.
- Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pp. 192–204. PMLR, 2015.
- 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.
- Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*, 27, 2014.
- 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.
- Kawaguchi, K. Deep learning without poor local minima. *Advances in neural information processing systems*, 29, 2016.
- Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.
- Kramár, J., Lieberum, T., Shah, R., and Nanda, N. AtP*: An efficient and scalable method for localizing llm behaviour to components. *arXiv preprint arXiv:2403.00745*, 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.
- Nanda, N. Attribution patching: Activation patching at industrial scale. URL: <https://www.neelnanda.io/mechanistic-interpretability/attribution-patching>, 2023.
- Pascanu, R. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2013.

- 660 Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou,
661 L. Empirical analysis of the hessian of over-parametrized
662 neural networks. *arXiv preprint arXiv:1706.04454*, 2017.
- 663 Soudry, D. and Hoffer, E. Exponentially vanishing sub-
664 optimal local minima in multilayer neural networks.
665 *arXiv preprint arXiv:1702.05777*, 2017.
- 666
- 667 Subramani, N., Suresh, N., and Peters, M. E. Extracting
668 latent steering vectors from pretrained language models.
669 *arXiv preprint arXiv:2205.05124*, 2022.
- 670
- 671 Syed, A., Rager, C., and Conmy, A. Attribution patching
672 outperforms automated circuit discovery. *arXiv preprint*
673 *arXiv:2310.10348*, 2023.
- 674
- 675 Turner, A. M., Thiergart, L., Leech, G., Udell, D., Vazquez,
676 J. J., Mini, U., and MacDiarmid, M. Steering lan-
677 guage models with activation engineering. *arXiv preprint*
678 *arXiv:2308.10248*, 2023.
- 679
- 680 Wang, G., Farrugia-Roberts, M., Hoogland, J., Carroll, L.,
681 Wei, S., and Murfet, D. Loss landscape geometry re-
682 veals stagewise development of transformers. In *High-*
683 *dimensional Learning Dynamics 2024: The Emergence*
684 *of Structure and Reasoning*, 2024.
- 685
- 686 Watanabe, S. Algebraic information geometry for learning
687 machines with singularities. *Advances in neural informa-*
688 *tion processing systems*, 13, 2000.
- 689
- 690 Watanabe, S. Algebraic geometry of singular learning ma-
691 chines and symmetry of generalization and training errors.
Neurocomputing, 67:198–213, 2005.
- 692
- 693 Watanabe, S. Almost all learning machines are singular. In
694 *2007 IEEE Symposium on Foundations of Computational*
695 *Intelligence*, pp. 383–388. IEEE, 2007.
- 696
- 697 Wei, S., Murfet, D., Gong, M., Li, H., Gell-Redman, J.,
698 and Quella, T. Deep learning is singular, and that’s good.
IEEE Transactions on Neural Networks and Learning
699 *Systems*, 34(12):10473–10486, 2022.
- 700
- 701
- 702
- 703
- 704
- 705
- 706
- 707
- 708
- 709
- 710
- 711
- 712
- 713
- 714

A. Definitions
A.0.1. DIMENSIONS

715 n_s : The number of samples in a batch of inputs
 716
 717 n_f : The dimensions of a single input vector to a model
 718
 719 n_o : The dimensions of a single output vector from a model
 720
 721 n_w : The number of parameters values in a model.
 722
 723 n_v : The number of subnetworks or parameter directions chosen to decompose a model.
 724
 725

A.0.2. MODEL SYNTAX

726 $X \in \mathbb{R}^{n_s \times n_f}, x \in \mathbb{R}^{n_f}$: Batch and individual input vectors to a model.
 727
 728 $W \in \mathbb{R}^{n_w}, w \in \mathbb{R}$: The set of and individual parameter values of a model
 729
 730 \sqsubseteq : The set of parameters corresponding to a specific tensor or block in a model.
 731
 732 $f : \mathbb{R}^{n_s \times n_f} \mapsto \mathbb{R}^{n_s \times n_o}$: A model mapping a set of input vectors to a set of output vectors.
 733
 734 $f(X, W)$: The output of model f with parameter values W on input X .
 735
 736 $f(X, W_0)$ or $f(X)$: The output of model f with fixed parameter values W_0 . W_0 is the set of learned parameter values from
 737 model training.
 738
 739 D : Divergence metric between two vectors. Typical divergence metrics are mean-squared error for regression-type outputs,
 740 and KL-divergence for probability-type outputs.
 741

A.0.3. DECOMPOSITION SYNTAX

742 $V(\text{or } V^{out}) \in \mathbb{R}^{n_v \times n_w}, v(\text{or } V^{out}) \in \mathbb{R}^{n_w}$: The set of or individual parameter directions that are used to decompose a
 743 model. V_{out} can be used to transform parameter directions in the subnetwork vector space back into the original parameter
 744 space of the model. The terms parameter direction, subnetwork, and circuit all have the same meaning.
 745
 746 $\mathcal{V}_{out}, \sqsubseteq_{out}$: Components of V^{out} or V^{out} related to a specific tensor or block in the original model.
 747
 748 $V^{in} \in \mathbb{R}^{n_w}, V^{in} \in \mathbb{R}$: Transforms the original parameter space of the model into the subnetwork vector space.
 749
 750 $\mathcal{V}_{in}, \sqsubseteq_{in}$: Components of V^{in} or V^{in} related to a specific tensor or block in the original model.
 751
 752 r : The rank of each component of the decomposition vectors corresponding to tensors in the original model.
 753

A.0.4. TRAINING

754 \mathcal{K} : The subset of indexes $i = \tau$ where τ is the threshold computed by using the top k absolute values in a tensor.
 755
 756 L : The L2 reconstruction loss used to optimize V^{in} and V^{out} .

A.0.5. MEASURING AND INTERVENTION

757 $I(x_i, x_j, v_k), I(x_i, v_k)$: The impact of subnetwork v_k on the divergence between samples x_i and x_j , or averaged across
 758 many x_j reference samples.
 759
 760 δ : A scalar value to move W in a specific direction.
 761

B. Additional Methods
B.1. Low-Rank Tensor Representation

765 We use low-rank representations of our V^{in} and V^{out} , and correspondingly learn low-rank circuits.
 766
 767
 768
 769

770 While W is a vector of all of the parameters in a model, typically model parameters are organized into tensors $W = \{w_i\}_i$.
 771 If our parameters are organized into tensors $W = \{w_i\}_i$, each subnetwork or parameter component can be organized as
 772 $V_i^{in} = \{v^{in}_i\}_i, V_i^{out} = \{v^{out}_i\}_i$ where we have parts of our subnetworks corresponding to each tensor in the original model
 773 parameters. We wish each of these tensors to be low rank, and we express them using the canonical polyadic decomposition
 774 () (a way to write 3+ dimentinoal tensors in terms of low-rank components).
 775

776
 777
 778 $v^{in}_{i,j} = \sum_{r=1}^R a_{i,j,r} \times b_{i,j,r} \times c_{i,j,r} \dots$ (8)
 779

780 Where R is the rank we wish to use to represent the parameter component, and the number of factors (a, b, c,...) in the
 781 factorization is equal to the number of dimensions in the tensor w_i .
 782

783 B.2. Finetuning

784 To finetune a model on a specific set of parameter directions, we would freeze the current set of weights and learn an
 785 adaptor consisting of linear combinations of the subnetworks of choice. During fine tuning, we would only need to learn the
 786 coefficients of these linear combinations greatly reducing cost.
 787

788
 789 $f(X, W_0 + W_{ft} V_K^{out})$ (9)
 790

791 B.2.1. QUANTIFYING IMPACT

792 We can quantify the impact of a subnetwork in two ways. First, we can compute the impact of a subnetwork on a pair of
 793 samples x_i, x_j , identifyng the subnetwork that, if intervened upon, would most strongly impact the the divergence of x_i 's
 794 outputs when compared to x_j . The impact of subnetwork v_k on such a pair of samples, $I(x_i, x_j, v_k)$ can be measured by:
 795

796
 797 $I(x_i, x_j, v_k) = |V_{k,:}^{in} \nabla_w D(f(x_i, W), f(x_j))|$ (10)
 798

799 Alternatively, we can average the impacts of a subnetwork v_k and an input x_i over many different reference samples to better
 800 quantify the impact of the subnetwork on a single sample's predictions overall. Although more computationally expensive,
 801 this can give a more robust measurement for the impact of a subnetwork on a specific sample.
 802

803
 804 $I(x_i, v_k) = \frac{1}{n_j} \sum_{j=1}^{n_j} I(x_i, x_j, v_k)$ (11)
 805
 806

807 B.3. Toy Model Training

808 For all of our toy models (except the $X \mapsto X^2$ model), we generate uniformly random inputs between 0 and 1. For
 809 $X \mapsto X^2$, we generate uniformly random inputs between -1 and 1. For all toy model data, we use a sparsity value of
 810 1-sparsity=.05. We generate 10000 datapoints and train for 1000 epochs with batch sizes of 32. We use an AdamW optimizer
 811 with a learning rate of 0.001.
 812

813
 814 $B.4. L3D Model Training$
 815
 816 To train L3D, we use the same training distributions as in each toy models. Although optimal hyperparameter values
 817 probably depend on the model size, and the rank and number of parameter tensors, we use the same hyperparameters for all
 818 of our models. We generate only 1000 datapoints, with a batch size of 32, and train for 1000 epochs. We use an AdamW
 819 optimizer with a learning rate of 0.01, and a learning decay rate of .8 every 100 steps. We always use a topFrac value of .1.
 820 We include all of the model's parameter tensors, including biases, in the decomposition.

821 C. Supplemental Figures

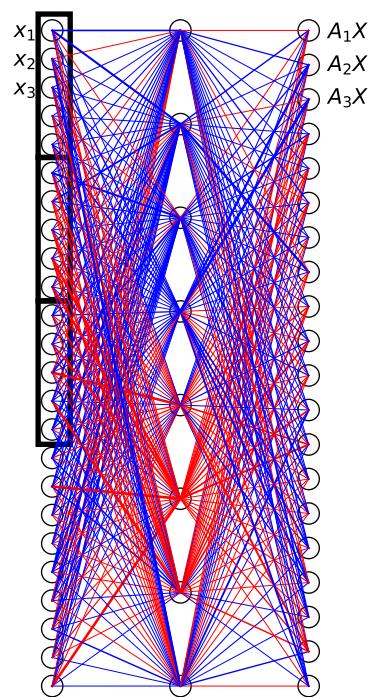


Figure S1: Full architecture of high rank circuit toy model.

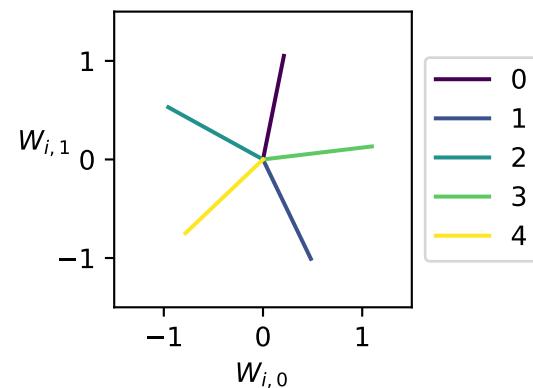


Figure S2: Encoder directions in TMS

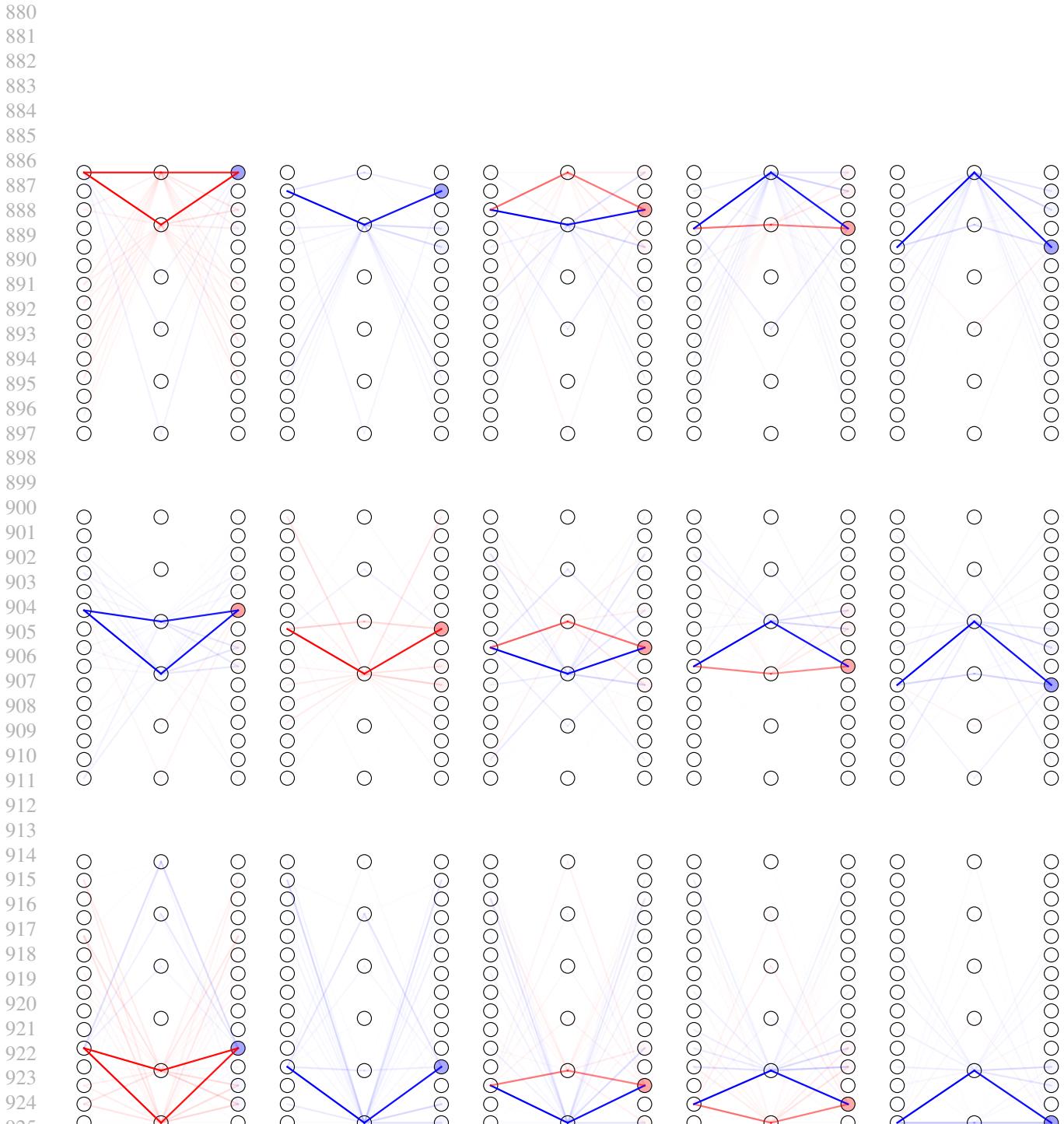
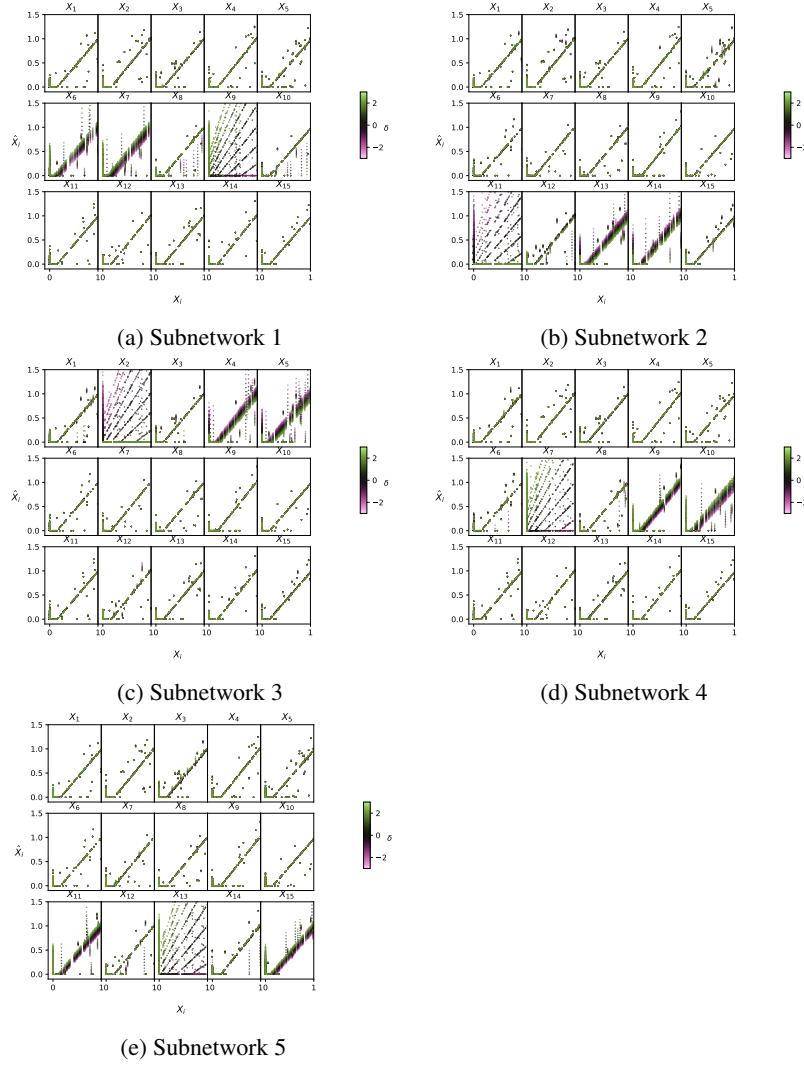


Figure S3: All subnetworks for the TMS decomposition

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



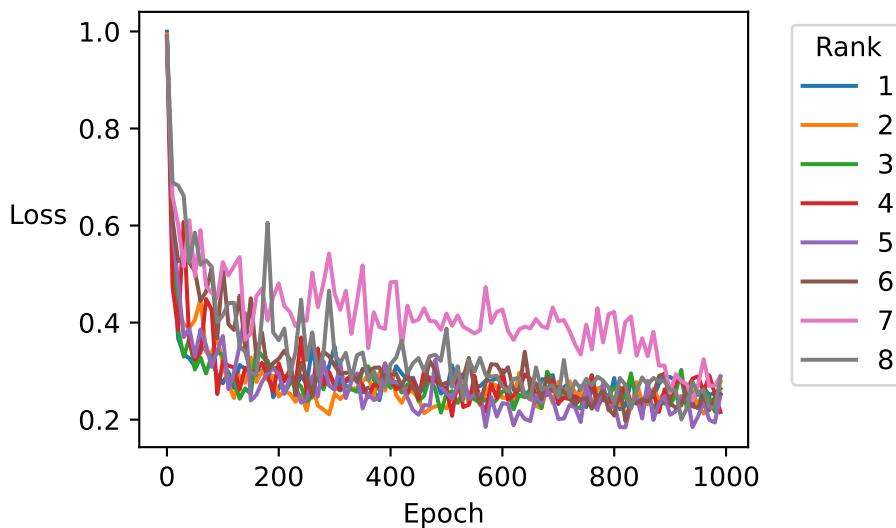
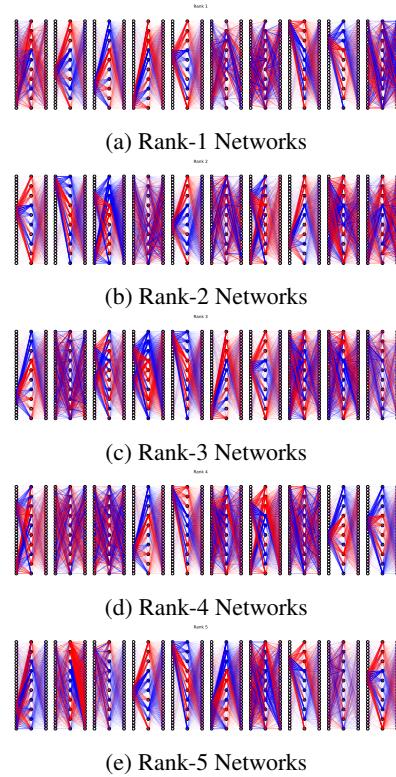


Figure S5: Loss vs Rank

Figure S6: Decomposing the toy model of high rank circuits into different numbers of subnetworks



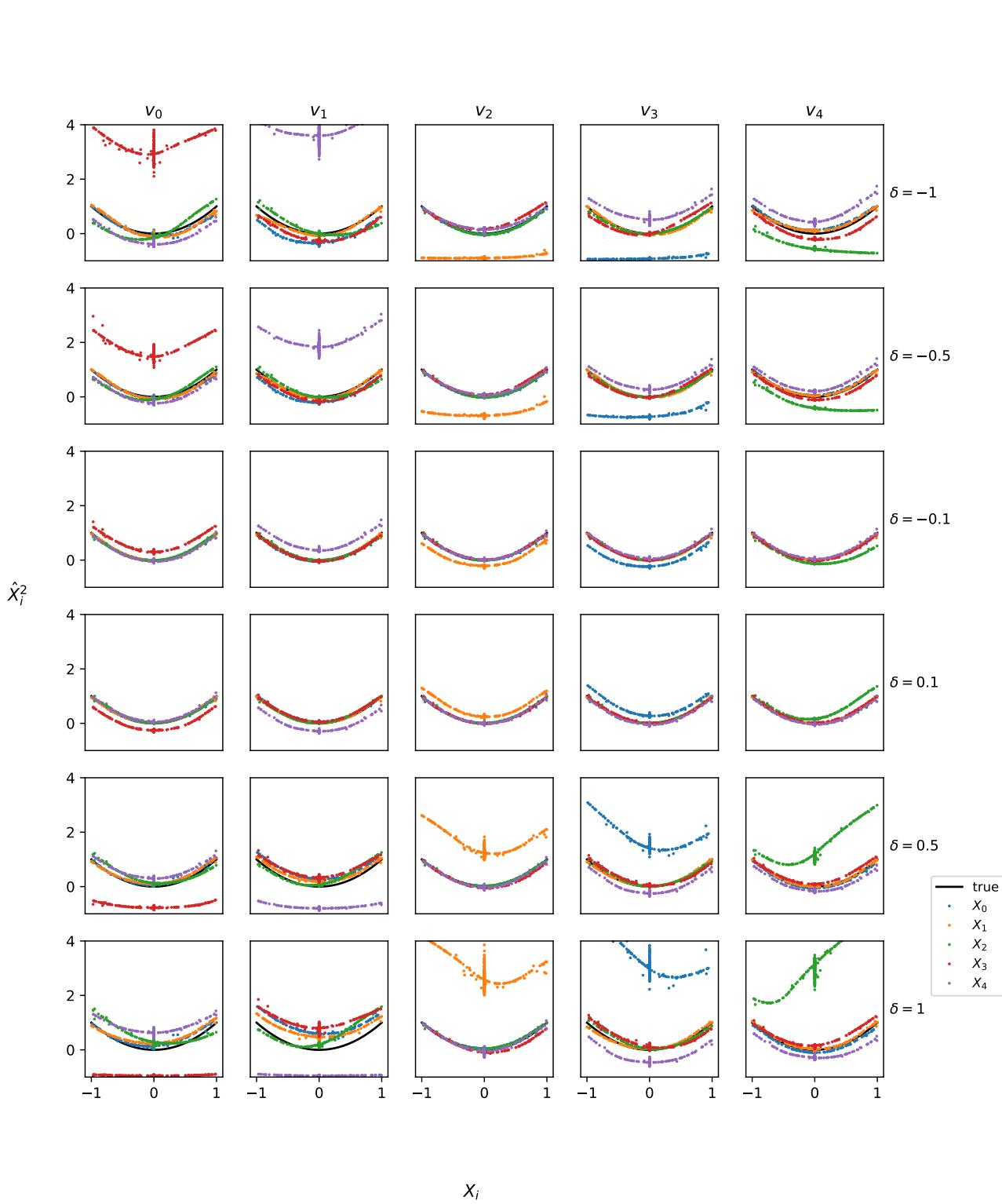


Figure S7: Effects of intervening on each of the 5 subnetworks of the $X \mapsto X^2$ model.

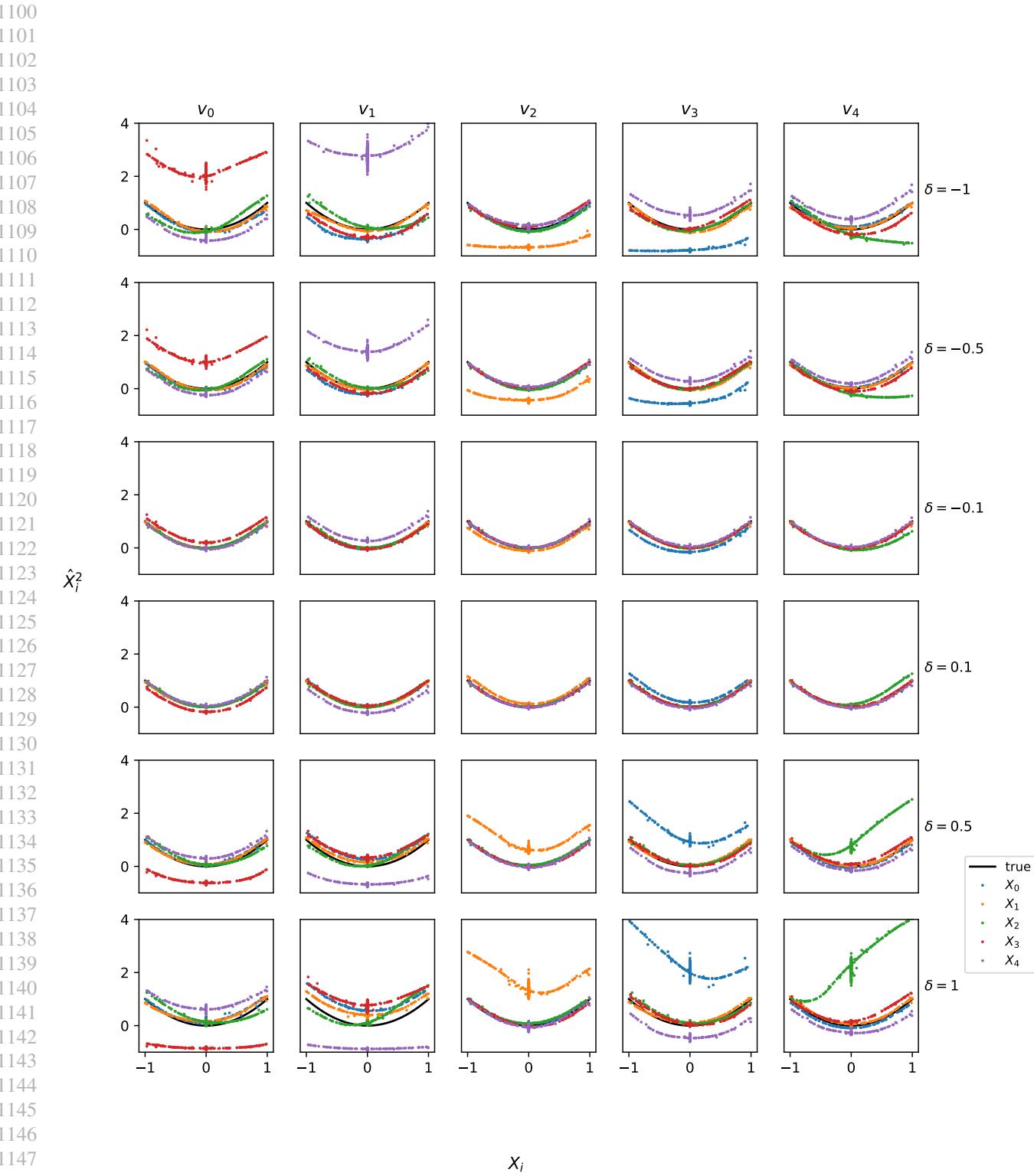


Figure S8: Intervening on just the weights and biases of the middle hidden layers in the $X \mapsto X^2$ network.

1155
1156
1157
1158
1159

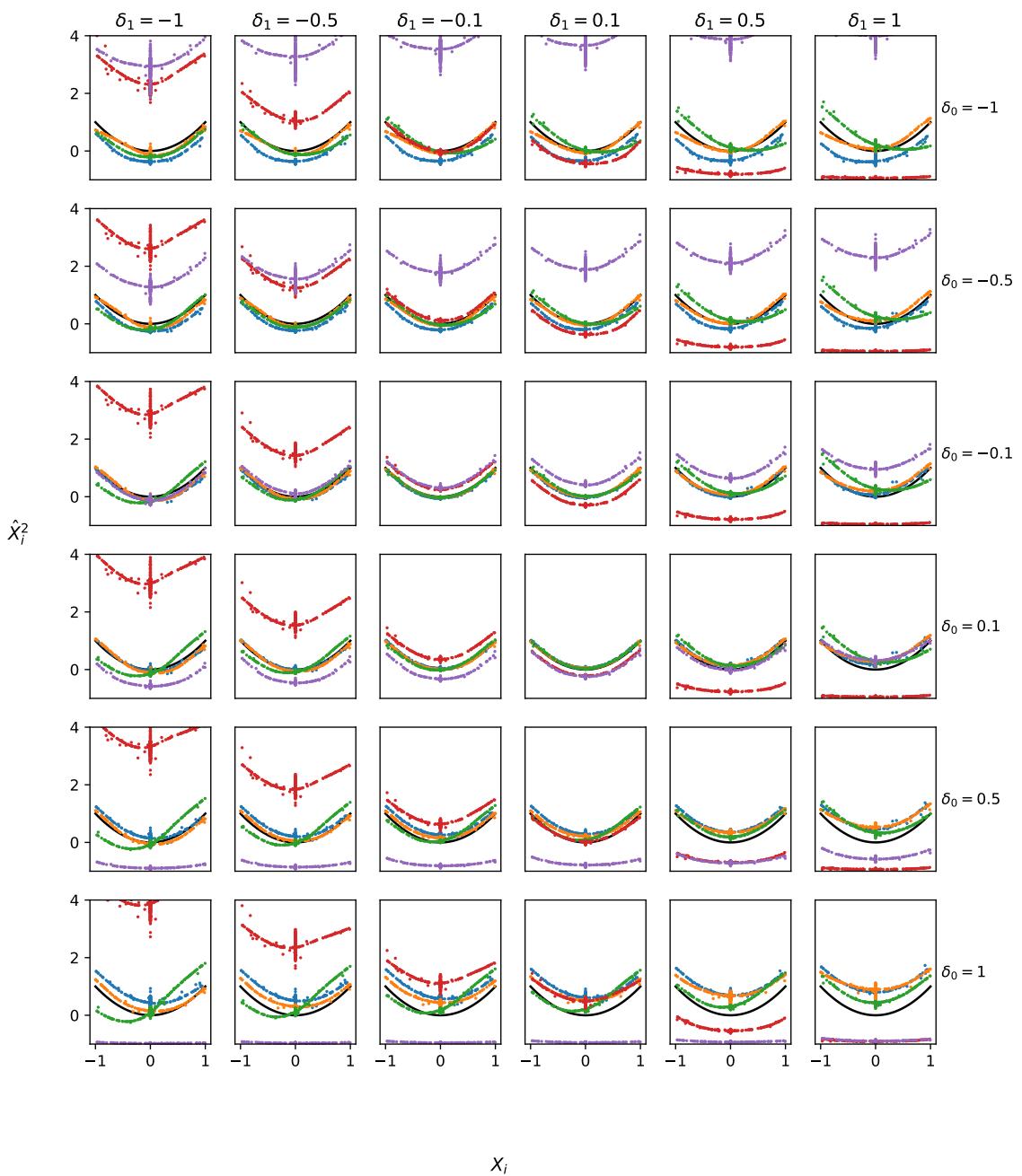


Figure S9: Intervening on multiple subnetworks at once.

Figure S10: Decomposing the $X \mapsto X^2$ model into different numbers of subnetworks

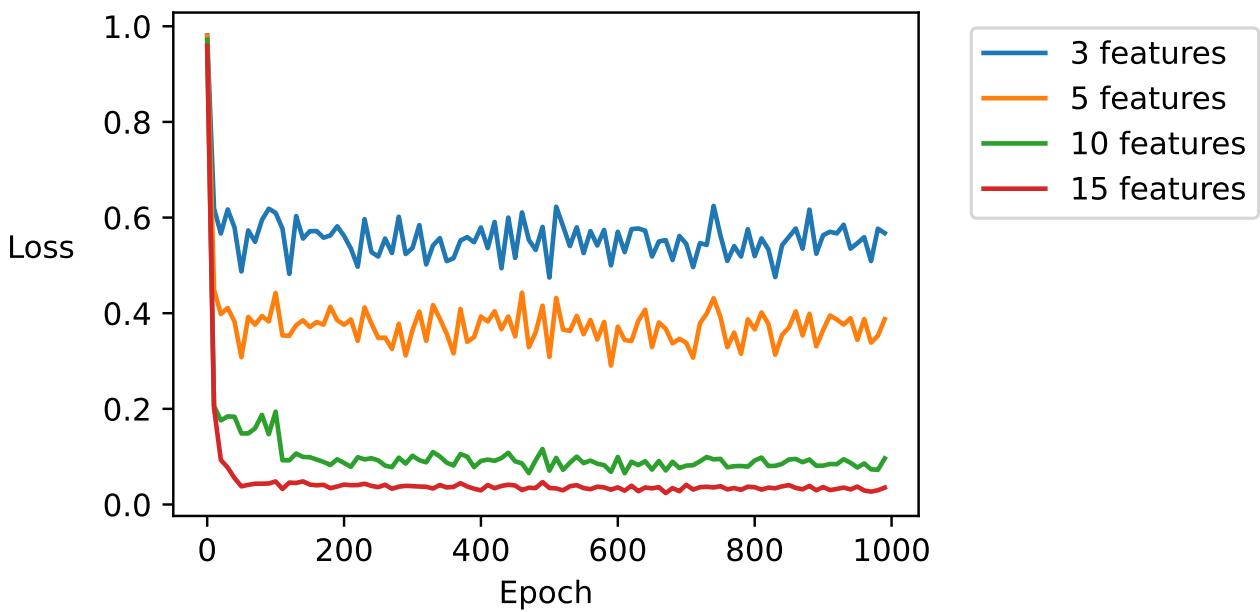
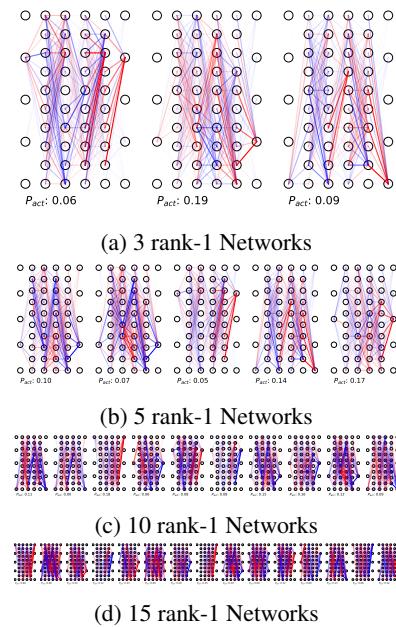


Figure S11: Training loss vs number of features for the $X \mapsto X^2$ model, and decompositions for a 3- and 5- subnetwork decomposition.