EXPLAINABILITY, MECHANISTIC INTERPRETABILITY AND FEATURE GEOMETRY

REDMOND BOWCOTT

ABSTRACT. We consider first external explainability methods and their flaws, then turn to mechanistic interpretability and contextualise it within the framework of representation learning. We focus on how superposition accounts for polysemanticity and note the implications for improving explainability. Finally, using toy models, we expand on previous work to explore how networks can represent multi-dimensional features.

CONTENTS

1. INTRODUCTION

Explainability has grown into an area of focus in machine learning as we move towards broad-scale implementation of models and approach a point where alignment concerns might become realised. For a model to be explainable, we must be able to understand why a given input has generated the corresponding output. In particular, we must be able to provide an explanation in terms of human-understandable concepts. One motivation for having explainable models is that we can then ensure a given model is ready for use. To do so, we need to be able to verify for the given model that:

1. It has been trained faithfully: the model does not learn on misleading characteristics in the training data such as labels or watermarks,
2. It predicts fairly and does not perpetuate biases,
3. Its behaviour is safe and well-aligned: the model fulfils the purpose intended for it - in other words, the loss function is correctly designed - and does not attempting to mislead the user.

Another appeal is that explainable models present an opportunity to improve human understanding, too. For example, If we can understand the decision process of a successful model on a problem, then we can gain meaningful insights from the model and through this to improve our understanding of the initial problem.

However the ease of interpreting a model varies depending on its architecture. For example, a shallow decision tree is highly interpretable: the output comes from the composition of relatively few, individually-highly-interpretable case-based decisions. In contrast, a deep neural network is the composition of affine maps between high-dimensional spaces, interspersed with element-wise nonlinearities. This is too complex for an individual to justify the behaviours that arise from inspection of the parameters. Yet the use of deep networks in a rapidly increasing range of spaces has made learning how to interpret these models particularly important.

This paper considers external approaches to explainability in deep neural networks, ultimately converging on *mechanistic interpretability.* It builds up the framework underpinning this approach and summarises supporting empirical evidence for the theory of *superposition* of features, in particular. It finally examines how features with an underlying shared structure can be represented in superposition.

In Section 2, we consider local feature attribution methods. We argue that issues, both practical and theoretical, mean such methods cannot provide the properties we would desire from an explainable model. In Section 3, we give a broad overview of mechanistic interpretability, and contextualise the approach in Section 4 by discussing representation learning. In Section 5, we outline the challenge of polysemanticity to interpretability, provide a framework to help better conceptualise the issue and consider how superposition explains the phenomenon. Section 6 provides an overview of the results of *Toy Models* [1], which displayed superposition in simple single-layer networks. We mention how this work, built up from the insights of the previous sections, has led to current explainability techniques. In Section 7, we expand on the setup of [3] by examining in greater detail how networks represent a set of features with a 2-dimensional structure. Through this, we underline the information encoded in the structure learnt by the model. Finally, we identify regions where feature *discretization* occurs and note that this seems to be caused by the choice of Adam as our optimizer.

Acknowledgements

This research was funded by the London Mathematical Society and the University of Edinburgh. I would like to thank my supervisor Tim Cannings for his support and guidance throughout the project.

2. Local External Methods: SHAP, LIME and LRP

Given the low level of inherent interpretability of deep neural networks, a suite of learning techniques has been developed to instead provide *local*, *external*, *explanation models* for such networks. Local methods aim to accurately model the network's behaviour in a subset of the input space, often around a single known input; external techniques do not use the model's parameters. This set of approaches entails using local external techniques to construct an intermediate explanation model, which provides a simplified approximation of the original model's behaviour. We then look to map from this intermediate model to an explanation in terms of human-understandable concepts. By assimilating the information from multiple local approximations, we can then produce a global explanation of the model's behaviour.

A number of the most widely used external techniques such as LIME, SHAP and LRP are unified in [4] as various instances of *additive feature attribution methods*. These techniques all construct explanation models in which the model output is approximated by a linear function of a set of binary variables representing the presence or absence of a set of features. Different attribution methods are therefore ultimately different approaches to calculating the *effect* of each feature variable, as quantified by the function coefficients.

Lundberg and Lee argue for SHAP in [4] as the unique method which satisfies a set of conditions necessary for an explanation model to be an accurate approximation. Before enumerating these conditions, we first define *Shapley values*, explain how they are calculated and describe how they relate to SHAP:

Shapley values arise in game theory as a way to fairly weight the impact of each 'player' (in our context, feature) on the final 'payout' (prediction). We first decide the set of features we wish to consider. For a given input, let S be the set of those features present. To compute the Shapley values for this input, we train a model on the features present for each element in the power set of S. To find the effect for a given feature φ, we consider all pairs of subsets of S which differ only in whether φ is present. We then compute a weighted average (exact formula provided in [4]) of the difference in prediction within each pair to arrive at our effect value. Note importantly that by comparing over all subsets, we do not have to assume independence between features and so Shapley values account for correlations [6].

SHAP uses Shapley values to represent the change in the expected prediction of the model if conditioned on the presence of a given feature [4]. The properties satisfied by Shapley values include - uniquely among feature attribution methods - the following trio:

1. Local Accuracy: The explanation model provides the same prediction as the original model for the input we are looking to explain.
2. Missingness: A feature absent from the original input has no effect.
3. Consistency: When comparing two models, if a feature's presence is more significant (in the context of difference between subset pairs as described above) over all possible subsets of the input for one model, then its effect for that model should be greater, too.

However, practically implementing SHAP is a significant challenge. Since you have to train a power set of models, the computational cost increases exponentially with the number of features considered. Therefore exact computation is not practically viable and instead we evaluate each subset using our original model.

This only replaces one problem with another: removing features entirely is often not possible without retraining. Therefore we must approximate feature absences instead. KernelSHAP, the main practical implementation of SHAP, carries this out, but requires the strong and unlikely assumption of feature independence to do so [4]. Assuming independence can lead to out-of-distribution data points and inaccuracies, though. For example, KernelSHAP no longer accounts for correlations, undermining the underlying meaning of the feature effect values. This necessary approximation reduces some of the theoretical appeal of Shapley values.

More fundamentally, building an explanation model as an intermediate may be misaligned with our explainability goals altogether. If the union of local approximations which forms our global explanation model is not able to completely accurately encapsulate the model's behaviour, then the explanation model is unreliable on at least some subset of the data. Even it is an almost-perfect approximation, then we have no guarantee that the decision process in the explanation model is the same as that in the original network [7]. In fact this seems especially unlikely considering the simplified structure of our explanation model. In either case, though, the need to map from the explanation model to a correct interpretation compounds these challenges as another round of approximation and interpretation is required.

Considering the underlying issues with external explainability attempts and the flaws with the current best methods, we must consider other approaches to explainability. In particular, it seems necessary that these approaches refuse to treat models as black boxes: approximations are insufficient for the level of understanding we require. We therefore turn to the global, internal, model-specific approach of mechanistic interpretability.

3. Mechanistic Interpretability: An Overview

Mechanistic interpretability is a field which aims to reverse-engineer models to achieve explainability. Such work looks to build up to human-interpretable concepts from a model's learned weights and biases. The end goal here is finding a set of techniques which can be used to produce a complete and understandable description of the algorithms which map from input to output for a given network. Intuitively, we can view the relation of mechanistic interpretability to external methods in the broader setting of explainability as equivalent to that of cognitive science to behaviorism within psychology [8].

Mechanistic interpretability work for neural networks in particular looks to identify the *features* learned by a given network during training and the *circuits* these features form across layers. We define a feature as a human-interpretable variable which contains meaningful information about an input, as in [16]. Circuits, built up from both features and the weights between layers, provide interpretable descriptions of how features interact to form more complex concepts. Through this, we aim to be able to obtain from the model's activations a precise and causal explanation as to why, for a given input, the corresponding output was generated.

However this approach requires the assumption that features and circuits form the constituent parts of a network. To justify such a belief, we need to contextualise mechanistic interpretability within a more general framework of how neural networks learn.

4. Representation Learning

Several challenges in machine learning arise from the fact that volume increases exponentially with dimension [9]. This is known as the *curse of dimensionality.* We will discuss this later as a challenge for interpretability: how can we search over such a large space to fully understand the algorithms being learnt? However the dual of this appears first in the training of a network: how can we generalize from our training data without requiring an exponential amount of such data?

Deep neural networks can mitigate the curse of dimensionality [10]: they can generalise from data which has a high sparsity in the input space. To understand this ability, we need to recognise deep learning as a form of *representation learning*. Representation learning involves identifying a set of underlying features which describe the input data in a way useful for more efficient computation [11]. The set of independently-varying features forms a *distributed representation* of the data. (In contrast, a *local representation* would assign a code for each object independently, with no decomposition of the objects into features [17].) For example, in the board game 'Guess Who!' we form a distributed representation of the characters by describing them in terms of hair colour, eye wear, gender etc. Importantly, the dimension of the distributed representation can be (up to exponentially) smaller than the number of inputs we are looking to represent. An example of a maximally efficient representation is a binary sequence, which can represent exponentially more numbers than it has bits.

Another possible explanatory factor for deep learning's efficiency is the fact that in many real-life cases, the distribution of data is expected to be concentrated within a subset of the original space with much lower dimensionality [11]. This is known as the *manifold hypothesis*: data is often concentrated on a lower-dimensional manifold embedded in the high-dimensional input space.

Under this assumption, we can view part of the network's learning process as finding the structure of this manifold by identifying the features which best parameterise it. The model then escapes the curse of dimensionality by learning the best function in this lower-volume, higher-density *latent space*. Note that many simpler techniques can be recontextualised through the lens of manifold learning, too: for example PCA finds the best linear manifold to approximate the data.

Deep learning in particular is then so powerful because depth allows for these features to be re-used across layers, in multiple different circuits, allowing for more complex levels of representation.

Evidence of deep learning as representation learning is seen in *word embeddings*: representations of words in a high-dimensional space, as learnt by a model trained to predict the missing word from text samples. Concepts such as gender and semantic features like comparative adjectives are seen to have consistent directions in the trained space [12], as are contextual relationships like country to national food. Famously, for example, in the embedding space we have

$$\text{“King”} - \text{“Man”} + \text{“Woman”} \approx \text{“Queen”!}$$

At no point in the training process is gender identified as a feature, nor is the model equipped with prior knowledge from definitions, say. The structures emerge entirely through the optimization process.

Another strong set of indicators for representation learning arises in vision models. Using techniques to maximise the activation of a given neuron, we create a *feature visualisation* which provides some intuition into what that neuron might be looking for. Across layers, seemingly interpretable features arise. These include Gabor filters: basic edge and texture detectors which are seen in the first layer across vision models [20], and curve detectors in later - but still early - layers [21]. More specific objects are then often visualised when going deeper into the model. This aligns with the expectation of circuits building up from the most basic concepts over layers.

The *universality* of these features across models is also explained by representation learning. If models are not learning by identifying relevant features, the question of why feature visualisations imply that the same features are consistently seen in the same layers across models remains.

Understanding the need for representation learning provides a more intuitive perspective of how a neural network functions. This perspective provides us with an avenue to explainability: it shows that any successful model must have found some underlying compressed structure, since it has generalised past the training data and therefore bypassed the curse of dimensionality. Importantly, in representation learning, finding structure naturally arises as part of the training process. Therefore the assumption that it is possible to recover a set of meaningful concepts used by the network *post hoc* - after the model has already been trained - is justified. This is one of the core motivations for mechanistic interpretability. (Note, however, that this is not sufficient to justify the strongest goals of mechanistic interpretability. As we discuss later in Section 5, we need to make further assumptions on how these features interact in the network's algorithms for mechanistic interpretability to lead to anything more than a dimension reduction.)

5. Polysemanticity, Superposition & Linear Representations

Returning to word embeddings, note that the number of features that can vary independently is equal to the dimension of the embedding space, since independence is equivalent to orthogonality. The embeddings in [12] had dimension 640: far less than the number of independent features we would expect to need to represent to fully capture semantic and contextual relationships across a language. In other words, the size of the embedding dimension was a limiting factor with respect to the means of representation.

Therefore we now need to change perspective: instead of trying to understand learning within a space too large, we look to understand how to represent a set of features constrained by a space too small. To begin, it is useful to step back and consider various modes of representation in general, as is done in [13]:

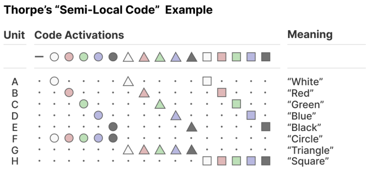

Figure 1. A compositional representation

On the right we see a semi-local code. Each unit corresponds to a unique feature. Objects are represented *compositionally* as the activation of their features. This allows for high expressivity - we can express the concept of any object, a single colour, or the combination of multiple shapes, for example - but requires more units, equivalent to requiring a higher-dimensional embedding space.

The 'highly distributed' code below is, like binary, an example of a maximally efficient representation. Each object is given by a unique unit pattern, but no representations exist for abstract concepts such as colours or shapes. The code cannot expand to objects outside the training data, such as a grey circle. As [13] argues through these examples, representations are a balance of expressivity and efficiency. The representation learned by a given model is a function of its own architecture as well as the data's underlying structure.

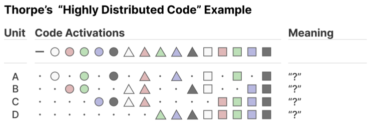

FIGURE 2. A non-compositional representation

It remains to discuss how to formalise these concepts for neural networks. Let l be a layer of the network and suppose l has n nodes. Then the *activation space* $\mathcal{A}_l$ for that layer is the n-dimensional space describing the activations of the neurons across l. The learned weights from the previous layer l' to layer l then show us where features are mapped to in $\mathcal{A}_l$ from $\mathcal{A}_{l'}$, taking into account biases and non-linearities between the layers. We can understand how features are represented in the network by the positions they are mapped to in each layer's activation space.

Note that within an activation space there can exist a *privileged basis*: a basis which is meaningful. The privileged basis that arises in neural networks is the standard, neuron-aligned basis. This happens if there is an element-wise nonlinearity directly after a layer (e.g. ReLU) or some sort of regularization constraint (for example an L_0 norm directly forces neuron alignment). Non-linearities incentivize features to align with the privileged basis by encouraging independence from other features. For example, suppose we want to store the value of two features in a hidden layer of a network using two neurons, and there is a ReLU directly after the layer. Having each neuron correspond to the value of a single feature is better than having one neuron equal to their average and another to their difference. In the first case, the value of one feature does not affect how well we store the value of the other, whereas in the second if either value is negative then the information is lost since its activation is set to 0 by the ReLU.

In contrast, if a given layer has no non-linearity or regularization pressure, then any rotation of the projection matrix mapping to that layer, followed by the reverse rotation in the projection to the next layer and a suitable adjustment of the second bias term, would produce the same representation. Therefore no privileged basis for that layer exists.

Reframing our explainability goals in these terms, the path we will pursue towards interpretablity begins with recovering features by identifying activation patterns in each activation space. We can then build towards circuits by understanding the transformations between activation spaces. The activation values for a given input would then show us the features identified by the model, and their role in the model's prediction.

One confounding factor in this pursuit, however, is the presence of *polysemanticity* in networks. Polysemanticity is a property of the neurons within a model: a neuron is polysemantic if it activates on multiple distinct features. For example in the vision model InceptionV1 [18], we see a neuron firing for both cats and cars, similarly we find a neuron firing for both HTTP requests and Korean text [19] in a language model. Importantly, these both show neurons activating on features sufficiently distinct that it seems unlikely there is some underlying unifying feature between them.

The simplest explanation for this would be that features are orthogonal in the activation space, but the layer does not have a privileged basis. Privileged bases could arise infrequently, or just be absent in the layers in which these neurons are located. This could be because the effects of regularization and non-linearities were not strong enough to force a privileged basis, for example. Were this the case, we would have to find the basis directions in which our features vary independently, but once this basis had been found, we would be able to extract features fairly easily.

However, as mentioned in the example of word embeddings, it may often be the case that the network would like to represent more features than there are dimensions in activation space. This leads to the *superposition hypothesis*: under certain conditions, networks will represent more features that there are neurons in a layer - equivalently than there are dimensions in activation space - by placing features in *superposition*: by projecting features to be *almost-orthogonal* to each other in activation space. The number of features a network can then represent in this manner increases exponentially, as the number of almost-orthogonal vectors does, with dimension.

It is important to note here that both of these explanations make the assumption that networks represent features compositionally, as in the semi-local code above (Fig 1). This forms part of the *linear representation hypothesis*: networks represent the presence of features by addition in activation space, and the intensity of features by scaling [14]. The linear representation hypothesis is an underlying assumption of the superposition hypothesis.

The composition of features as addition in activation space is necessary for us to be able to discuss how individual features are represented. In turn, this is incredibly fortunate for interpretability. Even highly efficient representations will have too many features to consider in unison: recall the range of features found in [12], for example. Compositional representations allow us to work atomically, though, gradually combining individual features into a full mechanistic understanding.

Intensity as scaling allows us to assume meaning is constant in each direction in activation space. Consistency of meaning is integral due to the nature of superposition. Superposition allows more features to be represented but, since features are no longer orthogonal, also causes *interference*: the presence of one feature causes an activation which has some magnitude in the direction of other features. However with bias terms and non-linearities like ReLU layers to filter noise, the loss caused by interference can be mitigated. Provided that features are mutually present sufficiently rarely - in other words, provided that features are *sparse* - having additional features compensates for the effects of interference. This is because higher sparsity reduces the rate of significant interference, as might be caused by multiple features all firing at once.

To provide contrast with other possible representations, suppose a network uses magnitude to differentiate between features. Consider two features represented in the same direction in activation space, but with the magnitude corresponding to each different. (More precisely, the network learns weights such that the expected activation induced in that layer has the same direction for both features but differs in magnitude depending on the feature). Then a low-intensity activation of one would be similar to a high-intensity activation of the other, and vice versa. If features are represented linearly, interference takes the form of a low-magnitude activation of features and its effect can be reduced as described above. In this nonlinear example, though, low level noise has a completely different meaning. Prediction is more affected as a result.

Moreover, we would expect features to be sparse in the training data for large language models or vision models. For example, InceptionV1 was trained on a subset of ImageNet with 1000 categories varying from guinea pigs to parachutes [15]. The proportion of the total set of features active for a given input is likely very low. So under these hypotheses, we would indeed expect superposition to occur in those models mentioned above with observed polysemanticity. To briefly note: it might be the case, though, that some features occur with a high frequency across a set of training samples, or are very important for prediction. Then a network might represent these features orthogonally to all others, but place the less-important others in superposition, still.

6. Visualising Superposition & Further Steps

The clearest and most detailed examination of superposition is given in [1]. The paper used a simplified toy model - a feedforward network with a single hidden layer - and trained it on synthetic data, generated according to a pre-defined distribution, to investigate how a network might represent features when there are more features than dimensions in activation space.

The network was trained as an *autoencoder*: it learnt how to best compress and recover the original input, which here was a collection of features varying independently. Thus we effectively start with a learned representation, and are interested in how this representation is stored within the network under space constraints. The network had to project down from the initial input space to the lower-dimensional activation space, and then back up to the original space. This was done with both a linear model and a model with a ReLU layer after the projection back to the original space.

The input was a vector x, with each coordinate representing a feature and the network was trained to learn a projection W so as to best approximate x: by $W^TWx + b$, for the linear model, or $\mathrm{ReLU}(W^TWx + b)$ for the nonlinear one.

In the main set of experiments, features were independent and importance was varied amongst features by assigning a coefficient I_i to each feature and defining the loss function as

$$\mathcal{L} = \sum_i I_i ||x_i - \hat{x}_i||_2^2.$$

Sparsity of features varied across trials by varying the probability p with which a given feature would be inactive, with the feature otherwise uniformly distributed over $[0, 1]$. The choice of uniform distribution was arbitrary.

The following results were observed:

1. The linear network did not exhibit superposition in any situation. For a hidden layer with k neurons, they represented the features with the $k-$highest importance coefficients orthogonally. This is as we would expect.
2. In the networks with a non-linearity, superposition did occur. Superposition increased with p: more features were represented almost-orthogonally as the sparsity of features increased.
3. The extent to which a feature was placed in superposition was a function of its sparsity and relative importance. Higher importance encouraged representation with less interference, as did a higher frequency of activity.

4. The transitions between a feature being placed orthogonal to all others, being represented in superposition and not being represented at all occurred suddenly, in a manner similar to 'phase changes'.
5. The weights corresponding to each feature all had an equal magnitude. This aligns with our expectation since they all had the same distribution. See Section 7 for a more detailed discussion of this.
6. In a slight variation of our initial experiment, with another ReLU layer placed between the hidden layer and final output, the features aligned with the neurons: in other words, the non-linearity created a privileged basis in the activation layer.

This work therefore provided strong support for both the linear representation hypothesis and the superposition hypothesis. By showing the dynamics of feature representation, it paved the way for the current largest-scale interpretability work on LLMs ([19], [16]). In particular, showing the relationship between superposition and polysemanticity provided a new direction of approach: trying to find an overcomplete basis in the activation space via a *sparse autoencoders* to recover the features of the learned representation [19].

7. Investigating Multi-Dimensional Features

At the time of [1], the linear representation hypothesis was assumed to be stronger. Features were assumed to be solely one-dimensional. However multi-dimensional features have since been extracted from models. In [2], for example, researchers showed that circular representations for days of the week and months of the year were present in large language models. We looked to expand on the setup of [3] to further explore how a 2-dimensional feature was represented under different sparsity and correlation conditions.

The setup focused on feature recovery as in the original *Toy Models* paper. We treat our set of features as evenly-spaced points around the perimeter of a circle. Let x be our input vector and x_ϕ denote the feature at angle ϕ. For each given trial, we fix a scale ℓ. For each training sample, we generate an angle θ in $[0, 2\pi]$ and a magnitude m uniformly distributed over [0, 1). The value of each feature x_ϕ is then given by:

$$x_\phi = \begin{cases} m\cos(\frac{\phi-\theta}{\ell}) & \text{if } \frac{|\phi-\theta|}{\ell} \leq \frac{\pi}{2} \text{ or } \frac{2\pi-(\phi-\theta)}{\ell} \leq \frac{\pi}{2} \\ 0 & \text{otherwise} \end{cases}$$

(Note that this is a slight adjustment to the original setup, done to create a circular underlying structure.)

The hidden layer is 3-dimensional and, as before, the model learns W so as to best approximate x by $\hat{x} = \text{ReLU}(W^TWx + b)$.

To visualise the location of features in activation space, we graph the columns of W. Since all features have the same distribution, we do not need to account for different expected magnitudes and the columns of W are suitable for understanding the relative feature positions in activation space. All visualisations in this section come from [22].

For the first part of this discussion, we consider the progression from $\ell = 1.0$ to $\ell = 0.1$, at which point there are still always at least two features firing. As the scale factor decreases, we see the initial smooth organization of features become more distorted:

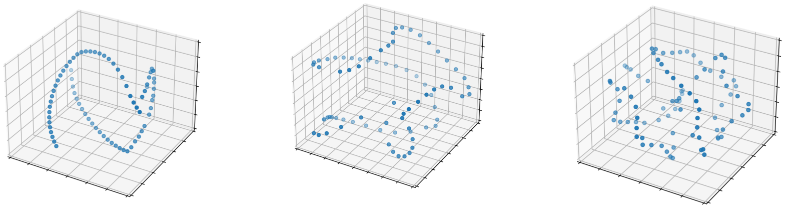

FIGURE 3. Network Weights Visualised with $\ell = 1.0,\ 0.3,\ 0.15$.

To provide some intuition and aid with interpretation, it is useful to briefly expand on how the final representations are efficient compressions of our features. First, note that the organisation of weights does roughly follow the geometric organisation of features as might be expected. In other words, the columns with most-similar weight vectors have corresponding features closest together with respect to angular distance (Fig 4). (Some discontinuities often arise, likely just from initialization, although this has not been looked into).

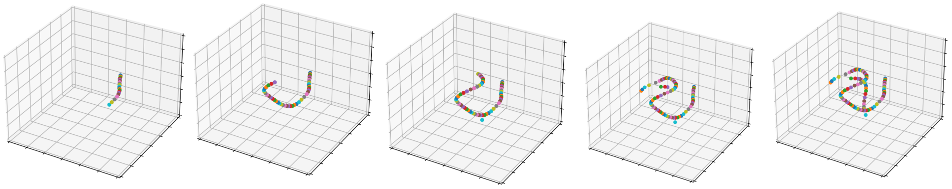

FIGURE 4. First k columns of W for $k = 20,\ 45,\ 60,\ 75,\ 90$. $N = 90$ total features, $\ell = 0.8$.

Next, consider the function of the structure of the weights matrix. Let W_ϕ denote the column in W corresponding to the feature x_ϕ. To begin with, suppose that only x_ϕ is active. Then the activation recovered is given by $|W_\phi|^2 x_\phi$. Next, suppose that x_ψ, close to x_ϕ with respect to our geometric definition of the features, is also active. Then the activation recovered for x_ϕ also has the term $W_\phi^T W_\psi x_\psi = \langle W_\phi, W_\psi \rangle x_\psi > 0$ since, as discussed above, the angle between the column vectors is small and by construction activations are always positive.

More generally, let us define features close together in the angular sense as *nearby features*. The organisation we see arise is valuable because nearby features are positively correlated between $\ell = 1.0$ and $\ell = 0.1$. Having these features close together in activation space allows positive interference to occur between them.

This explains the uniformly negative bias the network learns for all features. Feature activations are set to 0 through the ReLU layer if the signal from the feature and positive interference from those features around it is not strong enough. The magnitude of the bias also works to modulate the interference and recover the correct magnitude of activation.

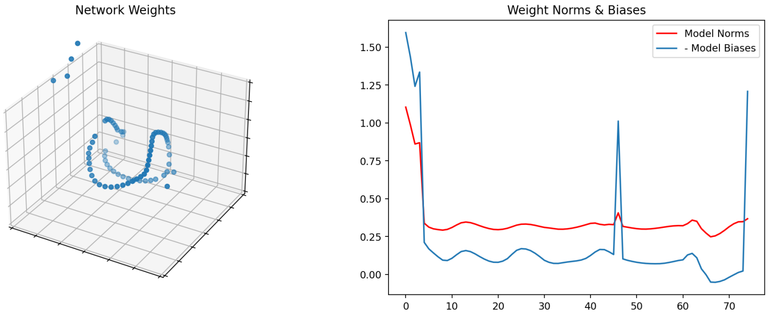

Figure 5. Network Weights (left), Norm of Weights and Negative of the Bias for each feature (right) with $\ell = 0.6$, $N = 75$ features.

As scale decreases, the activity of nearby features becomes a weaker indicator of activity of a given feature. We suspect the sequence in Fig 3 shows that as scale decreases, the curvature of the representation increases, resulting in a greater angular distance between nearby features and thus less positive interference between them.

This shows the role interference plays in encoding information about the relationship between features in a higher-dimensional representation. It is not just a tolerated side-effect of the representation that arises. Therefore, looking towards interpretability, we see that techniques which look to extract a discrete set of directions must also use the relations between these directions to construct an accurate explanation. In other words, we see that it is necessary to recover the structure of multi-dimensional representations to understand the relationship between the constituent features. While the relationship between day or month features is clear, this may not always be the case, especially in domains other than language.

Turning to the representations that arise when $\ell < 0.1$, we observe that the structure of the weights matrix progresses towards *discretization* as scale decreases: weights for different features converge to the same values to form clusters. This progression occurs as the correlation between nearby points decreases first to 0, then becomes negative until eventually all features are mutually exclusive (when the scale is such that at most one feature is active at once). To understand why this sequence of representations arises, it is necessary to consider both the structures that arise and the broader training regime:

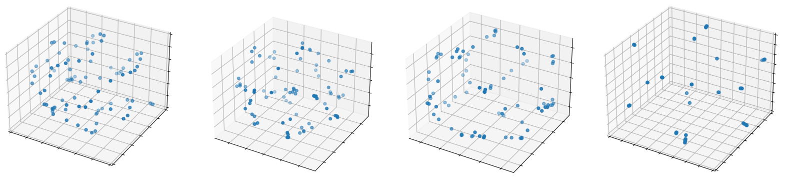

Figure 6. Weights matrix for $\ell = 0.09, 0.05, 0.04, 0.02$ with $N = 90$ features.

On the one hand, with $n = 90$ features, feature activations become mutually exclusive when $\ell < \frac{2}{45} \approx 0.044$. Past this point, there is no underlying structure for the network to represent and so no efficient representation to be found. This is evidenced by an almost-unchanged loss curve over the training regime below this scale boundary. Yet there remains a consistent pattern of first nonlinear representations, then discretization, emerging. An example is shown in the first training visualisation of Fig 7. This suggests that the use of Adam as optimizer plays a strong role in pushing towards discretization. This is supported by the fact that this discretization does not occur when using SGD with momentum as optimizer instead.

On the other hand, increasing discretization in the representations learnt seems to occur as we move towards the scale boundary from above, too. This type of representation does have a reduction on the loss above the scale boundary, as we see in the second paired graph of Fig 7.

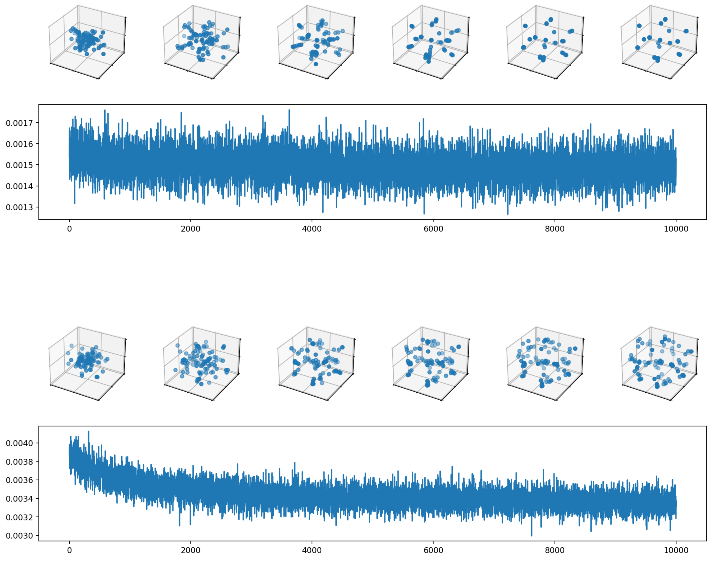

Figure 7. Training Weights & Corresponding Loss Curves for $\ell = 0.02$ (above) and $\ell = 0.05$ (below).

Consequently we must consider that discretization in itself is valuable as a compression technique when features are independent or all negatively correlated, as here, and when the number of features is much higher than the number of dimensions. Under these circumstances, the dimension compression is much more punishing since no efficient representation can be found. Significant information is necessarily lost when projecting down to the hidden layer.

One consideration explored was that the purpose of discretization is to divide activation space into distinct regions, with each region containing a collection of features. This creates a coarser set of features, allowing for a less precise, but consistent, rough recovery of the initial feature values. However it did not seem to be the case that nearby features were represented in the same regions, which we would expect if this theory was accurate. Nor did this explanation fit when considering that our loss function was the mean squared error between original and recovered feature vectors.

Another possible explanation is that the loss reduction we see at small scales is instead be the consequence of other characteristics of the representation. For example, it might occur during training before discretization begins, as the network learns to represent all weights with approximately equal magnitudes to avoid significant interference. Then Adam forces discretization, while loss remains unchanged. Why there is no change following discretization however must then be considered.

Overall, we believe something similar to this final explanation is the most likely account of the different representations which arise at small scales. However we say this with a high degree of uncertainty still. Although we are not definitive, we suspect that discretization is indeed an artifact of using Adam as optimizer.

References

[1] Elhage et al. Toy Models of Superposition. https://arxiv.org/pdf/2209.10652#

[2] Engels et al. Not All Language Model Features are Linear. https://arxiv.org/pdf/2405.14860

[3] Chris Olah, Josh Batson. Feature Manifolds. https://transformer-circuits.pub/2023/may-update/index.html#feature-manifolds

[4] Scott S. Lunberg, Su-In Lee. https://arxiv.org/pdf/1705.07874

[5] A. Hanif, X. Zhang and S. Wood. "A Survey on Explainable Artificial Intelligence Techniques and Challenges," 2021 IEEE 25th International Enterprise Distributed Object Computing Workshop, pp. 81-89.

[6] I. Cover, S. Lunberg, Su-In Lee. https://jmlr.csail.mit.edu/papers/volume22/20-1316/20-1316.pdf pg. 23.

[7] Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell 1, 206–215 (2019). https://doi.org/10.1038/s42256-019-0048-x

[8] L. Bereska, E. Gaves. Mechanistic Interpretability for AI Safety, A Review. https://arxiv.org/pdf/2404.14082

[9] C. Olah. Mechanistic Interpretability, Variables and the Importance of Interpretable Bases. https://transformer-circuits.pub/2022/mech-interp-essay/index.html

[10] Y. Bengio. Neural Net Language Models. http://www.scholarpedia.org/article/Neural_net_language_models

[11] Y. Bengio, A. Courville, P. Vincent. Representation Learning: A Review and New Perspectives. https://arxiv.org/pdf/1206.5538

[12] T. Mikolov et al. Efficient Estimations of Word Representations in Vector Space. https://arxiv.org/pdf/1301.3781

[13] C. Olah. Distributed Representations: Composition & Superposition. https://transformer-circuits.pub/2023/superposition-composition/index.html

[14] C. Olah. What is a Linear Representation? https://transformer-circuits.pub/2024/july-update/index.html#linear-representations

[15] https://image-net.org/challenges/LSVRC/2014/browse-synsets

[16] Templeton et al. Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet. https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html#computational/

[17] Hinton et al. Distributed Representations. https://stanford.edu/~jlmcc/papers/PDP/Chapter3.pdf

[18] Olah et al. Feature Visualization. https://distill.pub/2017/feature-visualization/

[19] Bricken et al. Towards Monosemanticity: Decomposing Language Models with Dictionary Learning. https://transformer-circuits.pub/2023/monosemantic-features/index.html

[20] Olah et al. An Overview of Early Vision in Inception V1. https://distill.pub/2020/circuits/early-vision/

[21] Cammarata et al. Curve detectors. https://distill.pub/2020/circuits/curve-detectors/

[22] https://colab.research.google.com/drive/1EZk7jN8fagL1SPOG9T1BMz3UU8MDPYXe?usp=sharing