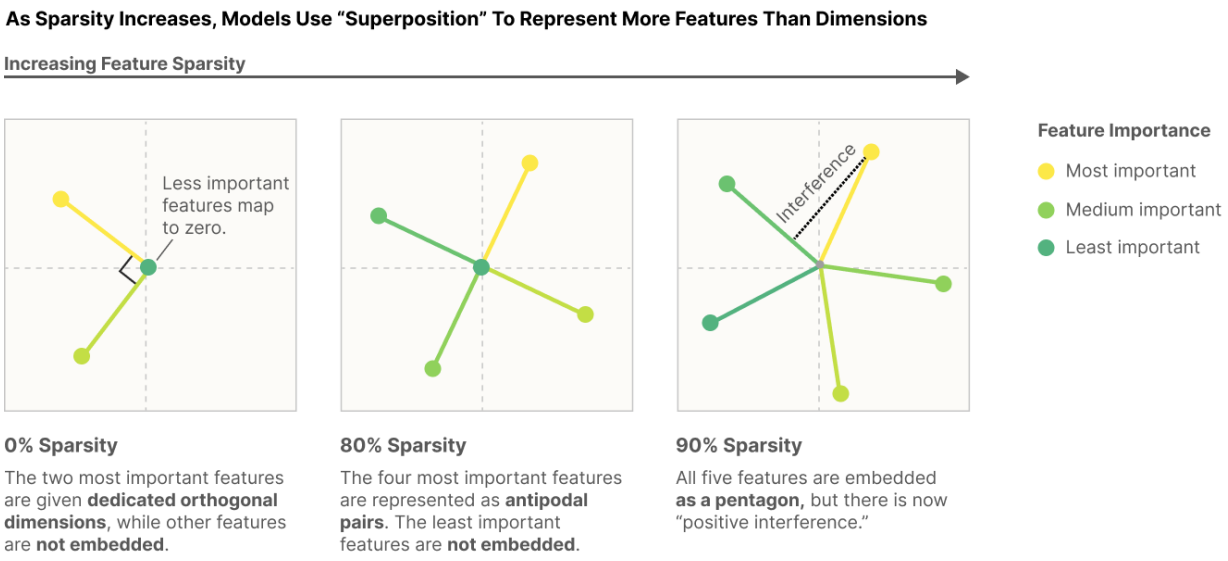
It would be very convenient if the individual neurons of artificial neural networks corresponded to cleanly interpretable features of the input. For example, in an “ideal” ImageNet classifier, each neuron would fire only in the presence of a specific visual feature, such as the color red, a left-facing curve, or a dog snout. Empirically, in models we have studied, some of the neurons do cleanly map to features. But it isn't always the case that features correspond so cleanly to neurons, especially in large language models where it actually seems rare for neurons to correspond to clean features. This brings up many questions. Why is it that neurons sometimes align with features and sometimes don't? Why do some models and tasks have many of these clean neurons, while they're vanishingly rare in others?

In this paper, we use toy models — small ReLU networks trained on synthetic data with sparse input features — to investigate how and when models represent more features than they have dimensions. We call this phenomenon **superposition** . When features are sparse, superposition allows compression beyond what a linear model would do, at the cost of "interference" that requires nonlinear filtering.

Consider a toy model where we train an embedding of five features of varying importanceWhere “importance” is a scalar multiplier on mean squared error loss. in two dimensions, add a ReLU afterwards for filtering, and vary the sparsity of the features. With dense features, the model learns to represent an orthogonal basis of the most important two features (similar to what Principal Component Analysis might give us), and the other three features are not represented. But if we make the features sparse, this changes:



This figure and a few others can be reproduced using the toy model framework Colab notebook in our Github repo

Not only can models store additional features in superposition by tolerating some interference, but we'll show that, at least in certain limited cases, *models can perform computation while in superposition*. (In particular, we'll show that models can put simple circuits computing the absolute value function in superposition.) This leads us to hypothesize that *the neural networks we observe in practice are in some sense noisily simulating larger, highly sparse networks*. In other words, it's possible that models we train can be thought of as doing "the same thing as" an imagined much-larger model, representing the exact same features but with no interference.

Feature superposition isn't a novel idea. A number of previous interpretability papers have considered it , and it's very closely related to the long-studied topic of compressed sensing in mathematics , as well as the ideas of distributed, dense, and population codes in neuroscience and deep learning . What, then, is the contribution of this paper?

For interpretability researchers, our main contribution is providing a direct demonstration that superposition occurs in artificial neural networks given a relatively natural setup, suggesting this may also occur in practice. That is, we show a case where interpreting neural networks as having sparse structure in superposition isn't just a useful post-hoc interpretation, but actually the "ground truth" of a model. We offer a theory of when and why this occurs, revealing a phase diagram for superposition. This explains why neurons are sometimes "monosemantic" responding to a single feature, and sometimes "polysemantic" responding to many unrelated features. We also discover that, at least in our toy model, superposition exhibits complex geometric structure.

But our results may also be of broader interest. We find preliminary evidence that superposition may be linked to adversarial examples and grokking, and might also suggest a theory for the performance of mixture of experts models. More broadly, the toy model we investigate has unexpectedly rich structure, exhibiting phase changes, a geometric structure based on uniform polytopes, "energy level"-like jumps during training, and a phenomenon which is qualitatively similar to the fractional quantum Hall effect in physics, among other striking phenomena. We originally investigated the subject to gain understanding of cleanly-interpretable neurons in larger models, but we've found these toy models to be surprisingly interesting in their own right.

Key Results From Our Toy Models

In our toy models, we are able to demonstrate that:

- **Superposition is a real, observed phenomenon**.
- **Both monosemantic and polysemantic neurons can form.**
- **At least some kinds of computation can be performed in superposition.**
- **Whether features are stored in superposition is governed by a phase change.**

- **Superposition organizes features into geometric structures** such as digons, triangles, pentagons, and tetrahedrons.

Our toy models are simple ReLU networks, so it seems fair to say that neural networks exhibit these properties in at least some regimes, but it's very unclear what to generalize to real networks.

Definitions and Motivation: Features, Directions, and Superposition

In our work, we often think of neural networks as having *features of the input* represented as *directions in activation space*. This isn't a trivial claim. It isn't obvious what kind of structure we should expect neural network representations to have. When we say something like "word embeddings have a gender direction" or "vision models have curve detector neurons", one is implicitly making strong claims about the structure of network representations.

Despite this, we believe this kind of "linear representation hypothesis" is supported both by significant empirical findings and theoretical arguments. One might think of this as two separate properties, which we'll explore in more detail shortly:

- **Decomposability:** Network representations can be described in terms of independently understandable features.
- **Linearity:** Features are represented by direction.

If we hope to reverse engineer neural networks, we *need* a property like decomposability. Decomposability is what allows us to reason about the model without fitting the whole thing in our heads! But it's not enough for things to be decomposable: we need to be able to access the decomposition somehow. In order to do this, we need to *identify* the individual features within a representation. In a linear representation, this corresponds to determining which directions in activation space correspond to which independent features of the input.

Sometimes, identifying feature directions is very easy because features seem to correspond to neurons. For example, many neurons in the early layers of InceptionV1 clearly correspond to features (e.g. curve detector neurons). Why is it that we sometimes get this extremely helpful

property, but in other cases don't? We hypothesize that there are really two countervailing forces driving this:

- **Privileged Basis:** Only some representations have a *privileged basis* which encourages features to align with basis directions (i.e. to correspond to neurons).
- **Superposition:** Linear representations can represent more features than dimensions, using a strategy we call *superposition*. This can be seen as neural networks *simulating larger networks*. This pushes features *away* from corresponding to neurons.

Superposition has been hypothesized in previous work , and in some cases, assuming something like superposition has been shown to help find interpretable structure . However, we're not aware of feature superposition having been unambiguously demonstrated to occur in neural networks before (demonstrates a closely related phenomenon of model superposition). The goal of this paper is to change that, demonstrating superposition and exploring how it interacts with privileged bases. If superposition occurs in networks, it deeply influences what approaches to interpretability research make sense, so unambiguous demonstration seems important.

The goal of this section will be to motivate these ideas and unpack them in detail.

It's worth noting that many of the ideas in this section have close connections to ideas in other lines of interpretability research (especially disentanglement), neuroscience (distributed representations, population codes, etc), compressed sensing, and many other lines of work. This section will focus on articulating our perspective on the problem. We'll discuss these other lines of work in detail in Related Work.

Empirical Phenomena

When we talk about "features" and how they're represented, this is ultimately theory building around several observed empirical phenomena. Before describing how we conceptualize those results, we'll simply describe some of the major results motivating our thinking:

- **Word Embeddings** - A famous result by *Mikolov et al.* found that word embeddings appear to have directions which correspond to semantic properties, allowing for embedding arithmetic vectors such as `V("king") - V("man") + V("woman") = V("queen")` (*but see*).
- **Latent Spaces** - Similar "vector arithmetic" and interpretable direction results have also been found for generative adversarial networks (e.g.).
- **Interpretable Neurons** - There is a significant body of results finding neurons which appear to be interpretable (*in RNNs* ; *in CNNs* ; *in GANs*), activating in response to some understandable property. This work has faced some skepticism .

In response, several papers have aimed to give extremely detailed accounts of a few specific neurons, in the hope of dispositively establishing examples of neurons which truly detect some understandable property (notably Cammarata *et al.* , but also).

- **Universality** - Many analogous neurons responding to the same properties can be found across networks .
- **Polysemantic Neurons** - At the same time, there are also many neurons which appear to not respond to an interpretable property of the input, and in particular, many *polysemantic neurons* which appear to respond to unrelated mixtures of inputs .

As a result, we tend to think of neural network representations as being composed of *features* which are *represented as directions*. We'll unpack this idea in the following sections.

What are Features?

Our use of the term "feature" is motivated by the interpretable properties of the input we observe neurons (or word embedding directions) responding to. There's a rich variety of such observed properties!In the context of vision, these have ranged from low-level neurons like and , to more complex neurons like or , to extremely abstract neurons corresponding to , , , and . In language models, researchers have found word embedding directions such as a male-female or singular-plural direction , low-level neurons disambiguating words that occur in multiple languages, much more abstract neurons, and "action" output neurons that help produce certain words . We'd like to use the term "feature" to encompass all these properties.

But even with that motivation, it turns out to be quite challenging to create a satisfactory definition of a feature. Rather than offer a single definition we're confident about, we consider three potential working definitions:

- **Features as arbitrary functions.** One approach would be to define features as any function of the input (as in). But this doesn't quite seem to fit our motivations. There's something special about these features that we're observing: they seem to in some sense be fundamental abstractions for reasoning about the data, with the same features forming reliably across models. Features also seem identifiable: cat and car are two features while cat+car and cat-car seem like mixtures of features rather than features in some important sense.
- **Features as interpretable properties.** All the features we described are strikingly understandable to humans. One could try to use this for a definition: features are the presence of human understandable "concepts" in the input. But it seems important to allow for features we might not understand. If AlphaFold

discovers some important chemical structure for predicting protein folding, it very well might not be something we initially understand!

- **Neurons in Sufficiently Large Models.** A final approach is to define features as properties of the input which a sufficiently large neural network will reliably dedicate a neuron to representing.This definition is trickier than it seems. Specifically, something is a feature if there a large enough model size such that it gets a dedicated neuron. This create a kind "epsilon-delta" like definition. Our present understanding – as we'll see in later sections – is that arbitrarily large models can still have a large fraction of their features be in superposition. However, for any given feature, assuming the feature importance curve isn't flat, it should eventually be given a dedicated neuron. This definition can be helpful in saying that something a feature – curve detectors are a feature because you find them in across a range of models larger than some minimal size – but unhelpful for the much more common case of features we only hypothesize about or observe in superposition. For example, curve detectors appear to reliably occur across sufficiently sophisticated vision models, and so are a feature. For interpretable properties which we presently only observe in polysemantic neurons, the hope is that a sufficiently large model would dedicate a neuron to them. This definition is slightly circular, but avoids the issues with the earlier ones.

We've written this paper with the final "neurons in sufficiently large models" definition in mind. But we aren't overly attached to it, and actually think it's probably important to not prematurely attach to a definition

Features as Directions

As we've mentioned in previous sections, we generally think of *features as being represented by directions*. For example, in word embeddings, "gender" and "royalty" appear to correspond to directions, allowing arithmetic like `V("king") - V("man") + V("woman") = V("queen")` . Examples of interpretable neurons are also cases of features as directions, since the amount a neuron activates corresponds to a basis direction in the representation

Let's call a neural network representation *linear* if features correspond to directions in activation space. In a linear representation, each feature f_i has a corresponding representation direction W_i. The presence of multiple features f_1, f_2… activating with values x_{f_1}, x_{f_2}… is represented by x_{f_1}W_{f_1} + x_{f_2}W_{f_2}.... To be clear, the features being represented are almost certainly nonlinear functions of the input. It's only the map from features to activation vectors which is linear. Note that whether something is a linear representation depends on what you consider to be the features.

We don't think it's a coincidence that neural networks empirically seem to have linear representations. Neural networks are built from linear functions interspersed with non-linearities. In some sense, the linear functions are the vast majority of the computation (for example, as measured in FLOPs). Linear representations are the natural format for neural networks to represent information in! Concretely, there are three major benefits:

- **Linear representations are the natural outputs of obvious algorithms a layer might implement.** If one sets up a neuron to pattern match a particular weight template, it will fire more as a stimulus matches the template better and less as it matches it less well.
- **Linear representations make features "linearly accessible."** A typical neural network layer is a linear function followed by a non-linearity. If a feature in the previous layer is represented linearly, a neuron in the next layer can "select it" and have it consistently excite or inhibit that neuron. If a feature were represented non-linearly, the model would not be able to do this in a single step.
- **Statistical Efficiency.** Representing features as different directions may allow *non-local generalization* in models with linear transformations (such as the weights of neural nets), increasing their statistical efficiency relative to models which can only locally generalize. This view is especially advocated in some of Bengio's writing (e.g.). A more accessible argument can be found in this blog post.

It is possible to construct non-linear representations, and retrieve information from them, if you use multiple layers (although even these examples can be seen as linear representations with more exotic features). We provide an example in the appendix. However, our intuition is that non-linear representations are generally inefficient for neural networks.

One might think that a linear representation can only store as many features as it has dimensions, but it turns out this isn't the case! We'll see that the phenomenon we call *superposition* will allow models to store more features – potentially many more features – in linear representations.

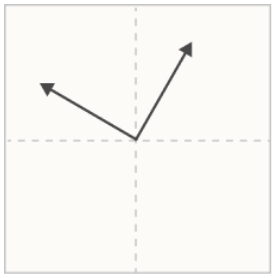
For discussion on how this view of features squares with a conception of features as being multidimensional manifolds, see the appendix “What about Multidimensional Features?”.

Privileged vs Non-privileged Bases

Even if features are encoded as directions, a natural question to ask is which directions? In some cases, it seems useful to consider the basis directions, but in others it doesn't. Why is this?

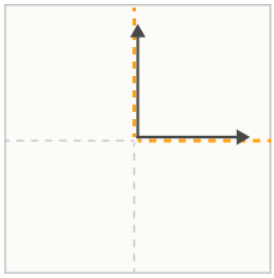
When researchers study word embeddings, it doesn't make sense to analyze basis directions. There would be no reason to expect a basis dimension to be different from any other possible direction. One way to see this is to imagine applying some random linear transformation M to the word embedding, and apply M^{-1} to the following weights. This would produce an identical model where the basis dimensions are totally different. This is what we mean by a *non-privileged basis*. Of course, it's possible to study activations without a privileged basis, you just need to identify interesting directions to study somehow, such as creating a gender direction in a word embedding by taking the difference vector between "man" and "woman".

But many neural network layers are not like this. Often, something about the architecture makes the basis directions special, such as applying an activation function. This "breaks the symmetry", making those directions special, and potentially encouraging features to align with the basis dimensions. We call this a privileged basis, and call the basis directions "neurons." Often, these neurons correspond to interpretable features.



In a **non-privileged basis**, features can be embedded in any direction. There is no reason to expect basis dimensions to be special.

Examples: word embeddings, transformer residual stream



In a **privileged basis**, there is an incentive for features to align with basis dimensions. This doesn't necessarily mean they will.

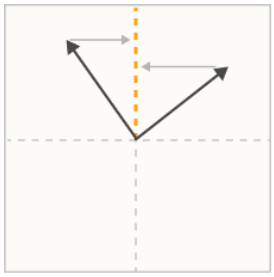
Examples: conv net neurons, transformer MLPs

From this perspective, it only makes sense to ask if a *neuron* is interpretable when it is in a privileged basis. In fact, we typically reserve the word "neuron" for basis directions which are in a privileged basis. (See longer discussion here.)

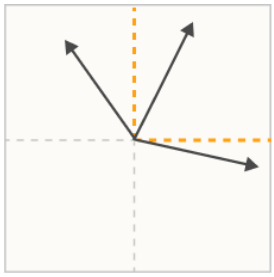
Note that having a privileged basis doesn't guarantee that features will be basis-aligned – we'll see that they often aren't! But it's a minimal condition for the question to even make sense.

The Superposition Hypothesis

Even when there is a privileged basis, it's often the case that neurons are "polysemantic", responding to several unrelated features. One explanation for this is the *superposition hypothesis*. Roughly, the idea of superposition is that neural networks "want to represent more features than they have neurons", so they exploit a property of high-dimensional spaces to simulate a model with many more neurons.



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

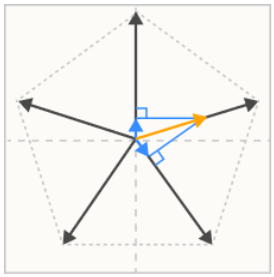


In the **superposition hypothesis**, features can't align with the basis because the model embeds more features than there are neurons. Polysemanticity is inevitable if this happens.

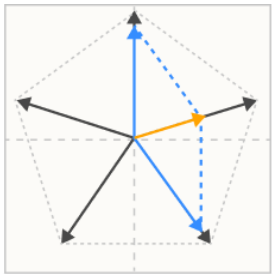
Several results from mathematics suggest that something like this might be plausible:

- **Almost Orthogonal Vectors.** Although it's only possible to have n orthogonal vectors in an n-dimensional space, it's possible to have \exp(n) many "almost orthogonal" (<\epsilon cosine similarity) vectors in high-dimensional spaces. See the Johnson–Lindenstrauss lemma.
- **Compressed sensing.** In general, if one projects a vector into a lower-dimensional space, one can't reconstruct the original vector. However, this changes if one knows that the original vector is sparse. In this case, it is often possible to recover the original vector.

Concretely, in the superposition hypothesis, features are represented as almost-orthogonal directions in the vector space of neuron outputs. Since the features are only almost-orthogonal, one feature activating looks like other features slightly activating. Tolerating this "noise" or "interference" comes at a cost. But for neural networks with highly sparse features, this cost may be outweighed by the benefit of being able to represent more features! (Crucially, sparsity greatly reduces the costs since sparse features are rarely active to interfere with each other, and non-linear activation functions create opportunities to filter out small amounts of noise.)

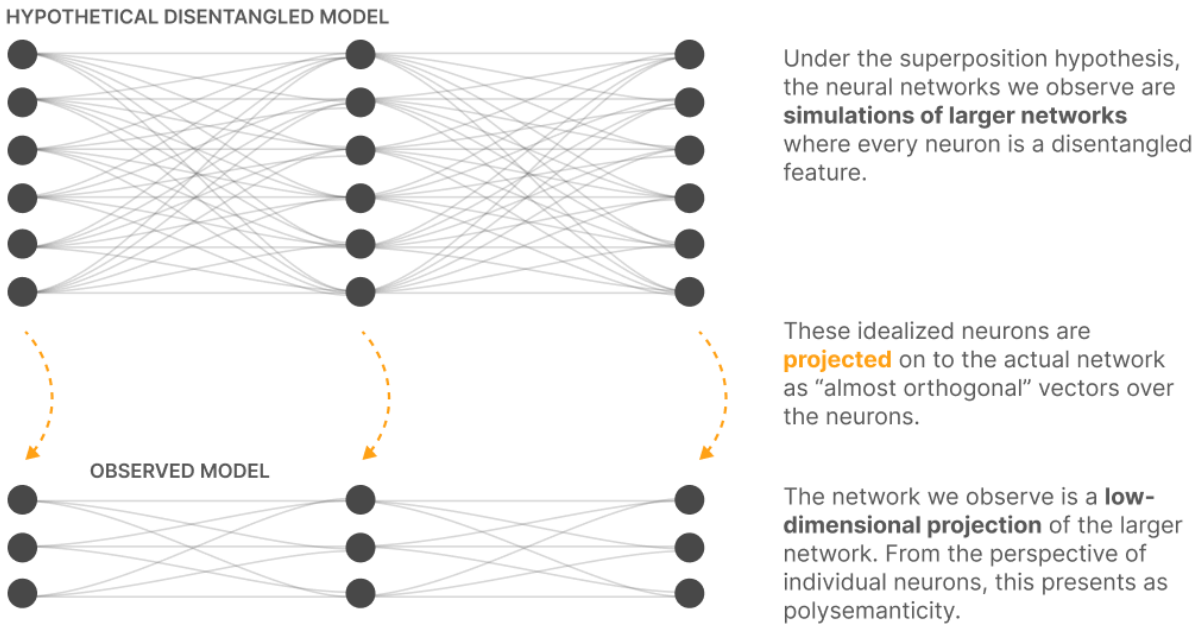


Even if only **one sparse feature** is active, using linear dot product projection on the superposition leads to **interference** which the model must tolerate or filter.



If the features aren't as sparse as a superposition is expecting, **multiple present features** can additively interfere such that there are multiple possible nonlinear reconstructions of an **activation vector**.

One way to think of this is that a small neural network may be able to noisily "simulate" a sparse larger model:



Although we've described superposition with respect to neurons, it can also occur in representations with an unprivileged basis, such as a word embedding. Superposition simply means that there are more features than dimensions.

Summary: A Hierarchy of Feature Properties

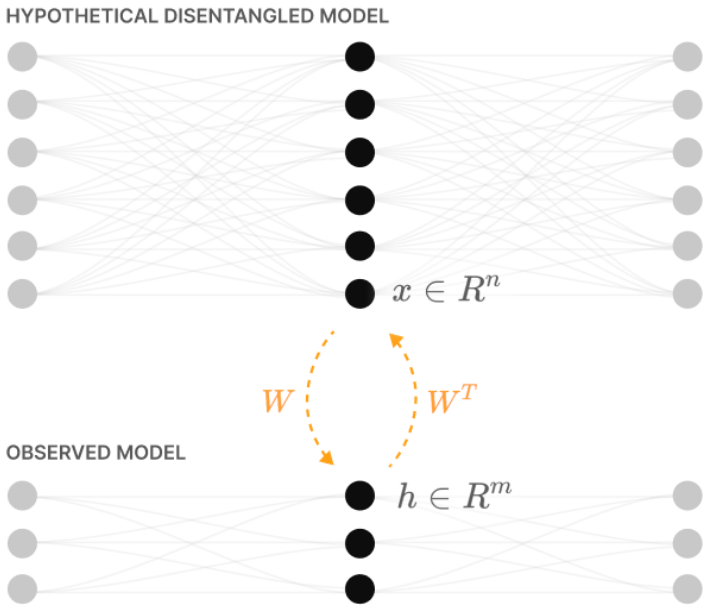
The ideas in this section might be thought of in terms of four progressively more strict properties that neural network representations might have.

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

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

Experiment Setup

Our goal is to explore whether a neural network can project a high dimensional vector x \in R^n into a lower dimensional vector h\in R^m and then recover it



Our first experiments will test the extent to which the idealized activations of an imagined larger model can be **stored** and **recovered** from a lower-dimensional space.

The Feature Vector (x)

We begin by describing the high-dimensional vector x: the activations of our idealized, disentangled larger model. We call each element x_i a "feature" because we're imagining features to be perfectly aligned with neurons in the hypothetical larger model. In a vision model, this might be a Gabor filter, a curve detector, or a floppy ear detector. In a language model, it might correspond to a token referring to a specific famous person, or a clause being a particular kind of description.

Since we don't have any ground truth for features, we need to create *synthetic data* for x which simulates any important properties we believe features have from the perspective of modeling them. We make three major assumptions:

- **Feature Sparsity:** In the natural world, many features seem to be sparse in the sense that they only rarely occur. For example, in vision, most positions in an image don't contain a horizontal edge, or a curve, or a dog head. In language, most tokens don't refer to Martin Luther King or aren't part of a clause describing music. This idea goes back to classical work on vision and the statistics of natural images (see e.g. Olshausen, 1997, the section "Why Sparseness?"). For this reason, we will choose a sparse distribution for our features.
- **More Features Than Neurons:** There are an enormous number of potentially useful features a model might represent.A vision model of sufficient generality might benefit from representing every species of plant and animal and every manufactured object which it might potentially see. A language model might benefit from representing each person who has ever been mentioned in writing. These are only scratching the surface of plausible features, but already there seem more than any model has neurons. In fact, large language models demonstrably do in fact know about people of very modest prominence – presumably more such people than they have neurons. This point is a common argument in discussion of the plausibility of "grandmother neurons" in neuroscience, but seems even stronger for artificial neural networks. This imbalance between features and neurons in real models seems like it must be a central tension in neural network representations.
- **Features Vary in Importance:** Not all features are equally useful to a given task. Some can reduce the loss more than others. For an ImageNet model, where classifying different species of dogs is a central task, a floppy ear detector might be one of the most important features it can have. In contrast, another feature might only very slightly improve performance

Concretely, our synthetic data is defined as follows: The input vectors x are synthetic data intended to simulate the properties we believe the true underlying features of our task have. We consider each dimension x_i to be a "feature". Each one has an associated sparsity S_i and importance I_i. We let $x_i=0$ with probability S_i, but is otherwise uniformly distributed between [0,1].The choice to have features distributed uniformly is arbitrary. An exponential or power law distribution would also be very natural. In practice, we focus on the case where all features have the same sparsity, $S_i = S$.

The Model (x \to x')

We will actually consider two models, which we motivate below. The first "linear model" is a well understood baseline which does not exhibit superposition. The second "ReLU output model" is a very simple model which does exhibit superposition. The two models vary only in the final activation function.

Linear Model	ReLU Output Model
$h = Wx$	$h = Wx$
$x' = W^T h + b$	$x' = \mathrm{ReLU}(W^T h + b)$
$x' = W^T W x + b$	$x' = \mathrm{ReLU}(W^T W x + b)$

Why these models?

The superposition hypothesis suggests that each feature in the higher-dimensional model corresponds to a direction in the lower-dimensional space. This means we can represent the down projection as a linear map h=Wx. Note that each column W_i corresponds to the direction in the lower-dimensional space that represents a feature x_i.

To recover the original vector, we'll use the transpose of the same matrix W^T. This has the advantage of avoiding any ambiguity regarding what direction in the lower-dimensional space really corresponds to a feature. It also seems relatively mathematically principledRecall that if is orthonormal. Although can't be literally orthonormal, our intuition from compressed sensing is that it will be "almost orthonormal" in the sense of Candes & Tao ., and empirically works.

We also add a bias. One motivation for this is that it allows the model to set features it doesn't represent to their expected value. But we'll see later that the ability to set a negative bias is important for superposition for a second set of reasons – roughly, it allows models to discard small amounts of noise.

The final step is whether to add an activation function. This turns out to be critical to whether superposition occurs. In a real neural network, when features are actually used by the model to

do computation, there will be an activation function, so it seems principled to include one at the end.

The Loss

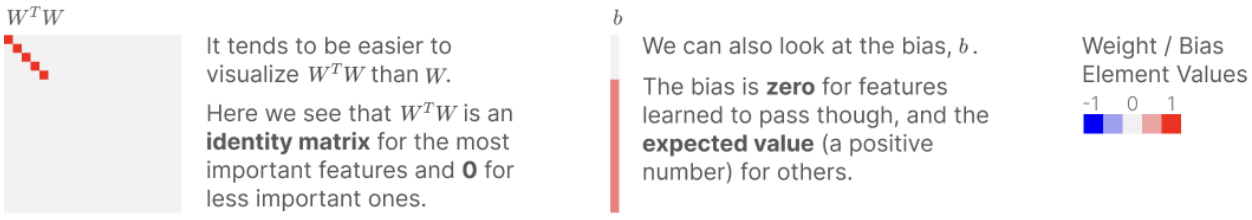
Our loss is weighted mean squared error weighted by the feature importances, I_i, described above:

$$L = \sum_x \sum_i I_i (x_i - x'_i)^2$$

Basic Results

Our first experiment will simply be to train a few ReLU output models with different sparsity levels and visualize the results. (We'll also train a linear model – if optimized well enough, the linear model solution does not depend on sparsity level.)

The main question is how to visualize the results. The simplest way is to visualize W^TW (a features by features matrix) and b (a feature length vector). Note that features are arranged from most important to least, so the results have a fairly nice structure. Here's an example of what this type of visualization might look like, for a small model model (n=20; ~m=5;) which behaves in the "expected linear model-like" way, only representing as many features as it has dimensions:

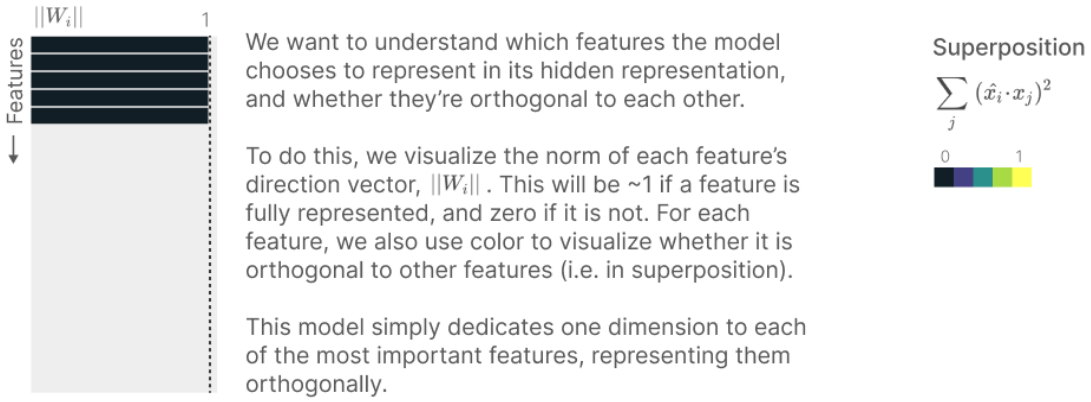


But the thing we really care about is this hypothesized phenomenon of superposition – does the model represent "extra features" by storing them non-orthogonally? Is there a way to get at it more explicitly? Well, one question is just how many features the model learns to represent. For

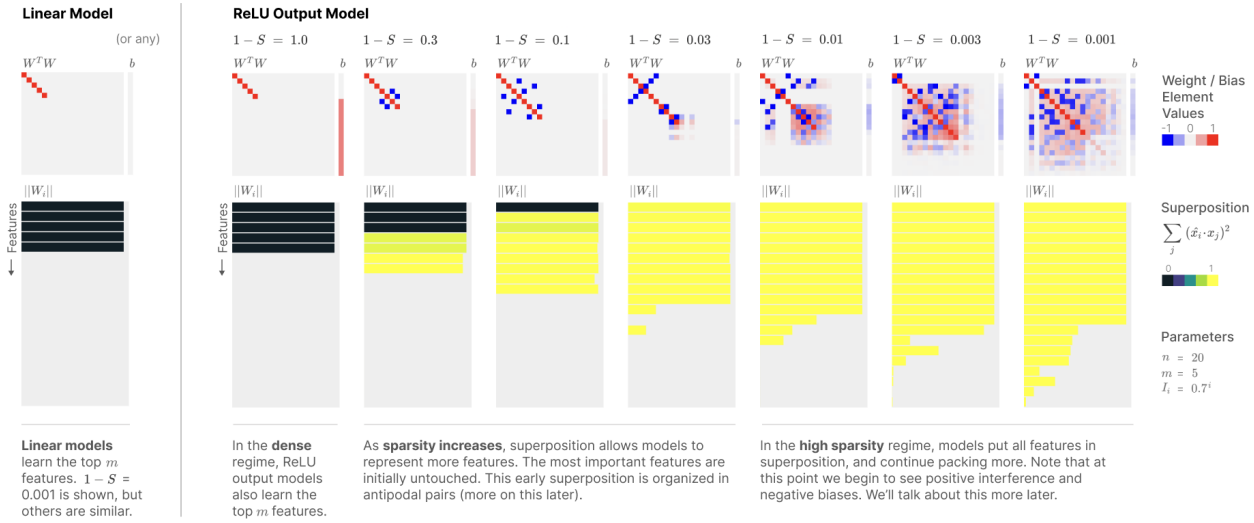
any feature, whether or not it is represented is determined by ||W_i||, the norm of its embedding vector.

We'd also like to understand whether a given feature shares its dimension with other features. For this, we calculate \sum_{j\neq i} (\hat{W_i}\cdot W_j)^2, projecting all other features onto the direction vector of W_i. It will be 0 if the feature is orthogonal to other features (dark blue below). On the other hand, values \geq 1 mean that there is some group of other features which can activate W_i as strongly as feature i itself!

We can visualize the model we looked at previously this way:

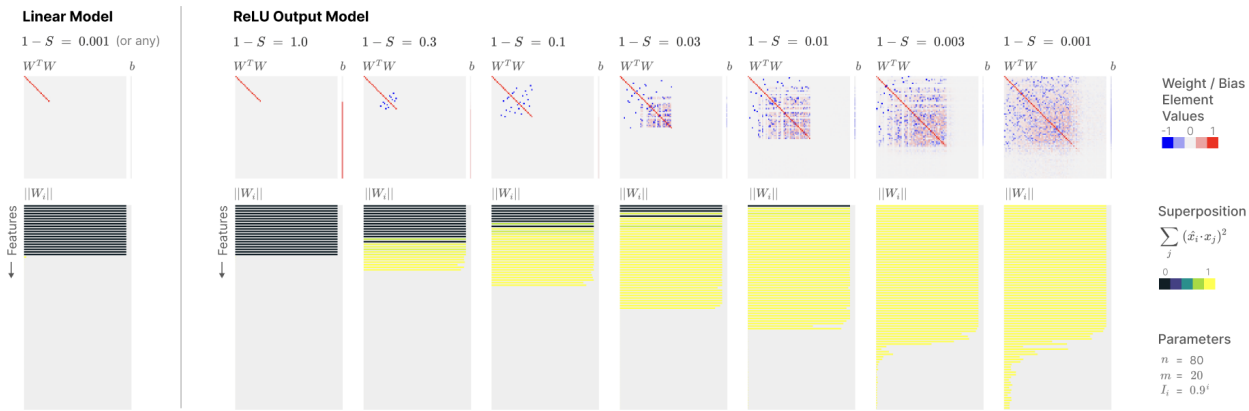


Now that we have a way to visualize models, we can start to actually do experiments. We'll start by considering models with only a few features (n=20; ~m=5;~ I_i=0.7^i). This will make it easy to visually see what happens. We consider a linear model, and several ReLU-output models trained on data with different feature sparsity levels:



As our standard intuitions would expect, the linear model always learns the top-m most important features, analogous to learning the top principal components. The ReLU output model behaves the same on dense features (1-S=1.0), but as sparsity increases, we see superposition emerge. *The model represents more features by having them not be orthogonal to each other.* It starts with less important features, and gradually affects the most important ones. Initially this involves arranging them in antipodal pairs, where one feature's representation vector is exactly the negative of the other's, but we observe it gradually transition to other geometric structures as it represents more features. We'll discuss feature geometry further in the later section, [The Geometry of Superposition](#).

The results are qualitatively similar for models with more features and hidden dimensions. For example, if we consider a model with m=20 hidden dimensions and n=80 features (with importance increased to I_i=0.9^i to account for having more features), we observe essentially a rescaled version of the visualization above:



Mathematical Understanding

In the previous section, we observed a surprising empirical result: adding a ReLU to the output of our model allowed a radically different solution – *superposition* – which doesn't occur in linear models.

The model where it occurs is still quite mathematically simple. Can we analytically understand why superposition is occurring? And for that matter, why does adding a single non-linearity make things so different from the linear model case? It turns out that we can get a fairly satisfying answer, revealing that our model is governed by balancing two competing forces –

feature benefit and *interference* – which will be useful intuition going forwards. We'll also discover a connection to the famous Thomson Problem in chemistry.

Let's start with the linear case. This is well understood by prior work! If one wants to understand why linear models don't exhibit superposition, the easy answer is to observe that linear models essentially perform PCA. But this isn't fully satisfying: if we set aside all our knowledge and intuition about linear functions for a moment, why exactly is it that superposition can't occur?

A deeper understanding can come from the results of Saxe et al. who study the learning dynamics of *linear neural networks* – that is, neural networks without activation functions. Such models are ultimately linear functions, but because they are the composition of multiple linear functions the dynamics are potentially quite complex. The punchline of their paper reveals that neural network weights can be thought of as optimizing a simple closed-form solution. We can tweak their problem to be a bit more similar to our linear case,We have the model be , but leave Gaussianaly distributed as in Saxe. revealing the following equation:

$$L \sim \sum_i I_i(1 - ||W_i||^2)^2 + \sum_{i \neq j} I_j(W_j \cdot W_i)^2$$

Feature benefit is the value a model attains from representing a feature. In a real neural network, this would be analagous to the potential of a feature to improve predictions if represented accurately.

Interference betwen x_i and x_j occurs when two features are embedded non-orthogonally and, as a result, affect each other's predictions. This prevents superposition in linear models.

The Saxe results reveal that there are fundamentally two competing forces which control learning dynamics in the considered model. Firstly, the model can attain a better loss by representing more features (we've labeled this "feature benefit"). But it also gets a worse loss if it represents more than it can fit orthogonally due to "interference" between features. In fact, this makes it never worthwhile for the linear model to represent more features than it has dimensions

Can we achieve a similar kind of understanding for the ReLU output model? Concretely, we'd like to understand $L = \int_x ||I(x - \mathrm{ReLU}(W^T W x + b))||^2 d\mathbf{p}(x)$ where x is distributed such that $x_i = 0$ with probability S.

The integral over x decomposes into a term for each sparsity pattern according to the binomial expansion of $((1-S)+S)^n$. We can group terms of the sparsity together, rewriting the loss as $L = (1-S)^n L_n + \ldots + (1-S)S^{n-1}L_1 + S^n L_0$, with each L_k corresponding to the loss when the input is a k-sparse vector. Note that as $S \to 1$, L_1 and L_0 dominate. The L_0 term, corresponding to the loss on a zero vector, is just a penalty on positive biases, $\sum_i \text{ReLU}(b_i)^2$. So the interesting term is L_1, the loss on 1-sparse vectors:

$$L_1 = \sum_i \int_{0 \le x_i \le 1} I_i(x_i - \text{ReLU}(||W_i||^2 x_i + b_i))^2 + \sum_{i \neq j} \int_{0 \le x_i \le 1} I_j \text{ReLU}(W_j \cdot W_i x_i + b_j)^2$$

If we focus on the case $x_i = 1$, we get something which looks even more analagous to the linear case:

$$= \sum_i I_i(1 - \text{ReLU}(||W_i||^2 + b_i))^2 + \sum_{i \neq j} I_j \text{ReLU}(W_j \cdot W_i + b_j)^2$$

Feature benefit is similar to before. Note that ReLU never makes things worse, and that the bias can help when the model doesn't represent a feature by taking on the expected value.

Interference is similar to before but ReLU means that negative interference, or interference where a negative bias pushes it below zero, is "free" in the 1-sparse case.

This new equation is vaguely similar to the famous Thomson problem in chemistry. In particular, if we assume uniform importance and that there are a fixed number of features with ||W_i|| = 1 and the rest have ||Wi||=0, and that b_i = 0, then the feature benefit term is constant and the interference term becomes a generalized Thomson problem – we're just packing points on the surface of the sphere with a slightly unusual energy function. (We'll see this can be a productive analogy when we resume our empirical investigation in the following sections!)

Another interesting property is that ReLU makes negative interference free in the 1-sparse case. This explains why the solutions we've seen prefer to only have negative interference when possible. Further, using a negative bias can convert small positive interferences into essentially being negative interferences.

What about the terms corresponding to less sparse vectors? We leave explicitly writing these out to the reader, but the main idea is that there are multiple compounding interferences, and the "active features" can experience interference. In a later section, we'll see that features often

organize themselves into sparse interference graphs such that only a small number of features interfere with another feature – it's interesting to note that this reduces the probability of compounding interference and makes the 1-sparse loss term more important relative to others.

Superposition as a Phase Change

The results in the previous section seem to suggest that there are three outcomes for a feature when we train a model: (1) the feature may simply not be learned; (2) the feature may be learned, and represented in superposition; or (3) the model may represent a feature with a dedicated dimension. The transitions between these three outcomes seem sharp. Possibly, there's some kind of phase change

One way to understand this better is to explore if there's something like a "phase diagram" from physics, which could help us understand when a feature is expected to be in one of these regimes. Although we can see hints of this in our previous experiment, it's hard to really isolate what's going on because many features are changing at once and there may be interaction effects. As a result, we set up the following experiment to better isolate the effects

As an initial experiment, we consider models with 2 features but only 1 hidden layer dimension. We still consider the ReLU output model, \text{ReLU}(W^T W x - b). The first feature has an importance of 1.0. On one axis, we vary the importance of the 2nd "extra" feature from 0.1 to 10. On the other axis, we vary the sparsity of all features from 1.0 to 0.01. We then plot whether the 2nd "extra" feature is not learned, learned in superposition, or learned and represented orthogonally. To reduce noise, we train ten models for each point and average over the results, discarding the model with the highest loss.

We can compare this to a theoretical "toy model of the toy model" where we can get closed form solutions for the loss of different weight configurations as a function of importance and sparsity. There are three natural ways to store 2 features in 1 dimension: W=[1,0] (ignore [0,1], throwing away the extra feature), W=[0,1] (ignore [1,0], throwing away the first feature to give the extra feature a dedicated dimension), and W=[1,-1] (store the features in superposition, losing the ability to represent [1,1], the combination of both features at the same time). We call this last solution “antipodal” because the two basis vectors [1, 0] and [0, 1] are mapped in opposite directions. It turns out we can analytically determine the loss for these solutions (details can be found in this notebook).

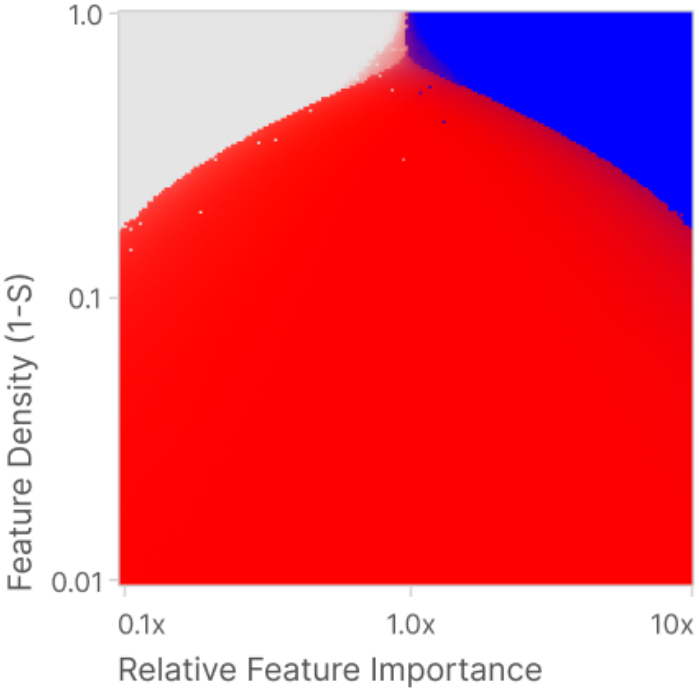
Sparsity-Relative Importance Phase Diagram (n=2, m=1)

What happens to an “extra feature” if the model can’t give each feature a dimension? There are three possibilities, depending on feature sparsity and the extra feature’s importance relative to other features:

- Extra Feature is Not Represented
- Extra Feature Gets Dedicated Dimension
- Extra Feature is Stored In Superposition

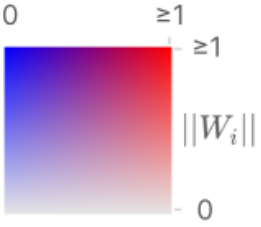
We can both study this empirically and build a theoretical model:

Empirical Version



Each configuration is colored by the norm and superposition of the extra feature.

$$\sum_j (\hat{x}_i \cdot x_j)^2$$



Theoretical Prediction

Not Represented
(Extra Feature is 0)

$$W = \begin{bmatrix} 1 & 0 \end{bmatrix}$$
$$W \perp \begin{bmatrix} 0 & 1 \end{bmatrix}$$

Dedicated Dimension
(Other Not Represented)

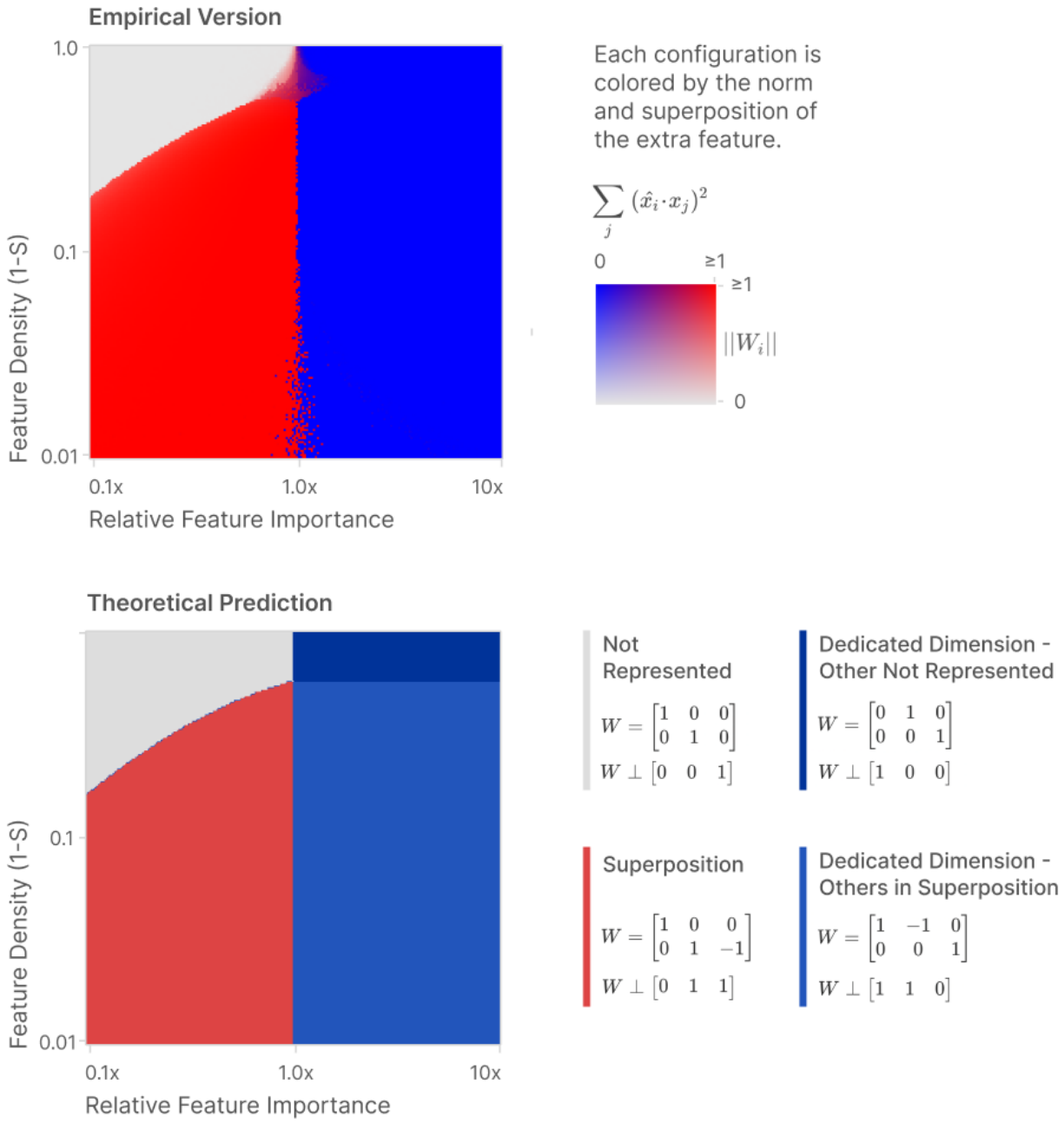
$$W = \begin{bmatrix} 0 & 1 \end{bmatrix}$$
$$W \perp \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Superposition
(Antipodal Pair)

As expected, sparsity is necessary for superposition to occur, but we can see that it interacts in an interesting way with relative feature importance. But most interestingly, there appears to be a real phase change, observed in both the empirical and theoretical diagrams! The optimal weight configuration discontinuously changes in magnitude and superposition. (In the theoretical model, we can analytically confirm that there's a first-order phase change: there's crossover between the functions, causing a discontinuity in the derivative of the optimal loss.)

We can ask this same question of embedding three features in two dimensions. This problem still has a single "extra feature" (now the third one) we can study, asking what happens as we vary its importance relative to the other two and change sparsity.

For the theoretical model, we now consider four natural solutions. We can describe solutions by asking "what feature direction did W ignore?" For example, W might just not represent the extra feature – we'll write this W \perp [0, 0, 1]. Or W might ignore one of the other features, W \perp [1, 0, 0]. But the interesting thing is that there are two ways to use superposition to make antipodal pairs. We can put the "extra feature" in an antipodal pair with one of the others (W \perp [0, 1, 1]) or put the other two features in superposition and give the extra feature a dedicated dimension (W \perp [1, 1, 0]). Details on the closed form losses for these solutions can be found in this notebook. We do not consider a last solution of putting all the features in joint superposition, W \perp [1, 1, 1].



These diagrams suggest that there really is a phase change between different strategies for encoding features. However, we'll see in the next section that there's much more complex structure this preliminary view doesn't capture.

The Geometry of Superposition

We've seen that superposition can allow a model to represent extra features, and that the number of extra features increases as we increase sparsity. In this section, we'll investigate this relationship in more detail, discovering an unexpected geometric story: features seem to organize themselves into geometric structures such as pentagons and tetrahedrons! In some ways, the structure described in this section seems "too elegant to be true" and we think there's a good chance it's at least partly idiosyncratic to the toy model we're investigating. But it seems worth investigating because if anything about this generalizes to real models, it may give us a lot of leverage in understanding their representations.

We'll start by investigating ***uniform superposition***, where all features are identical: independent, equally important and equally sparse. It turns out that uniform superposition has a surprising connection to the geometry of uniform polytopes! Later, we'll move on to investigate ***non-uniform superposition***, where features are not identical. It turns out that this can be understood, at least to some extent, as a deformation of uniform superposition.

Uniform Superposition

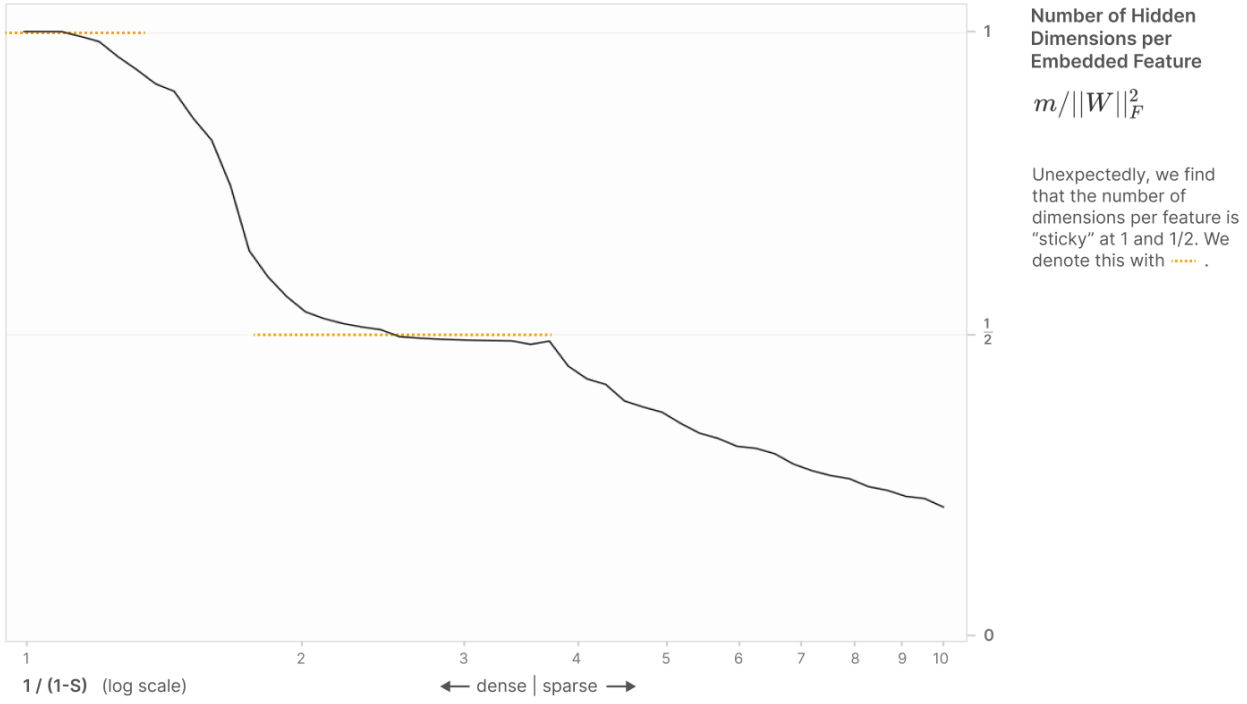
As mentioned above, we begin our investigation with uniform superposition, where all features have the same importance and sparsity. We'll see later that this case has some unexpected structure, but there's also a much more basic reason to study it: it's much easier to reason about than the non-uniform case, and has fewer variables we need to worry about in our experiments.

We'd like to understand what happens as we change feature sparsity, S. Since all features are equally important, we will assume without loss of generalityScaling the importance of all features by the same amount simply scales the loss, and does not change the optimal solutions. that each feature has importance I_i = 1 . We'll study a model with n=400 features and m=30 hidden dimensions, but it turns out the number of features and hidden dimensions doesn't matter very much. In particular, it turns out that the number of input features n doesn't matter as long as it's much larger than the number of hidden dimensions, n \gg m. And it also turns out that the number of hidden dimensions doesn't really matter as long as we're interested in the ratio of

features learned to hidden features. Doubling the number of hidden dimensions just doubles the number of features the model learns

A convenient way to measure the number of features the model has learned is to look at the Frobenius norm, ||W||_F^2. Since ||W_i||^2\simeq 1 if a feature is represented and ||W_i||^2\simeq 0 if it is not, this is roughly the number of features the model has learned to represent. Conveniently, this norm is basis-independent, so it still behaves nicely in the dense regime S=0 where the feature basis isn't privileged by anything and the model represents features with arbitrary directions instead.

We'll plot D^* = m / ||W||_F^2, which we can think of as the "dimensions per feature":



Surprisingly, we find that this graph is "sticky" at 1 and 1/2. (This very vaguely resembles the fractional quantum Hall effect – see e.g. this diagram.) Why is this? On inspection, the 1/2 "sticky point" seems to correspond to a precise geometric arrangement where features come in "antipodal pairs", each being exactly the negative of the other, allowing two features to be packed into each hidden dimension. It appears that antipodal pairs are so effective that the model preferentially uses them over a wide range of the sparsity regime.

It turns out that antipodal pairs are just the tip of the iceberg. Hiding underneath this curve are a number of extremely specific geometric configurations of features.

Feature Dimensionality

In the previous section, we saw that there's a sticky regime where the model has "half a dimension per feature" in some sense. This is an average statistical property of the features the model represents, but it seems to hint at something interesting. Is there a way we could understand what "fraction of a dimension" a specific feature gets?

We'll define the *dimensionality* of the ith feature, D_i, as:

$$D_i = \frac{||W_i||^2}{\sum_j (\hat{W}_i \cdot W_j)^2}$$

where W_i is the weight vector column associated with the ith feature, and \hat{W_i} is the unit version of that vector.