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

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 principled[9] , 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

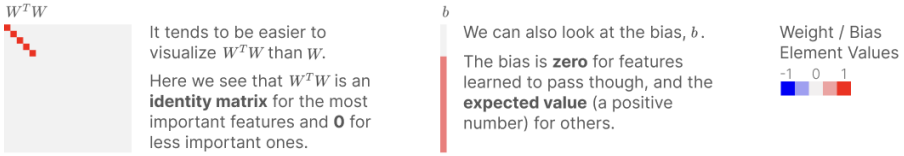
Our loss is weighted mean squared error weighted by the feature importances, I_i, described above:

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

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.)

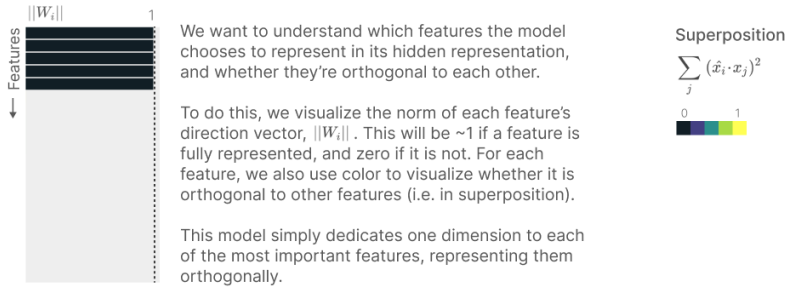
The main question is how to visualize the results. The simplest way is to visualize $W^T W$ (a features by features matrix) and b (a feature length vector). Note that features are arranged from most important to least, so the results have a fairly nice structure. Here's an example of what this type of 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:



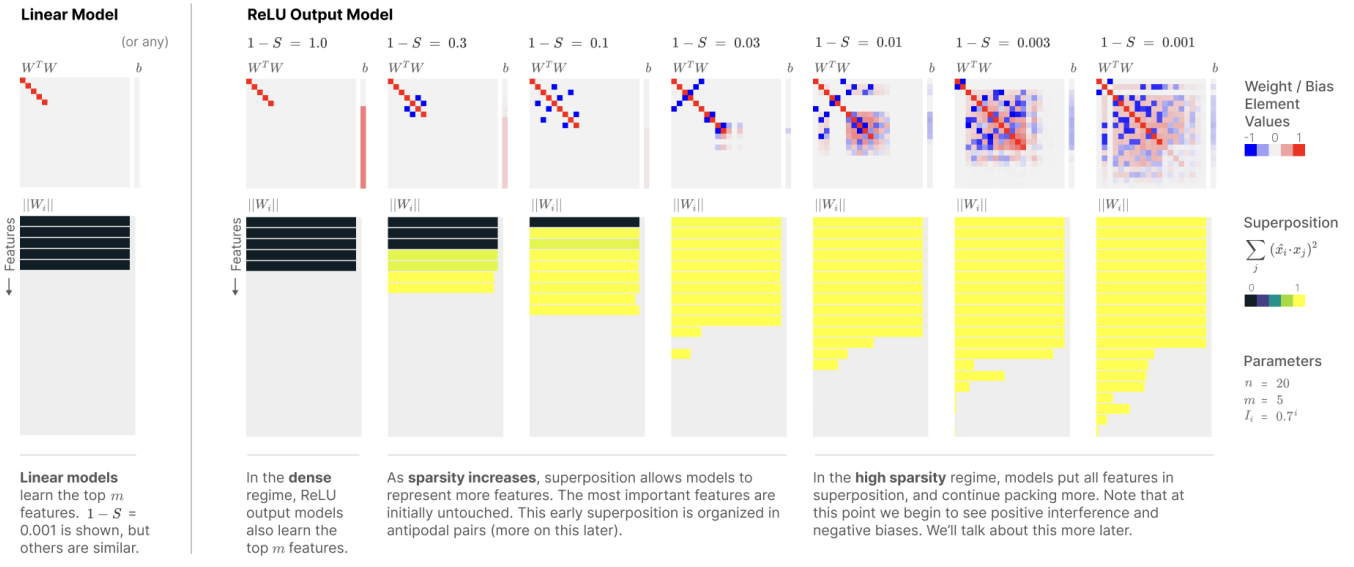
But the thing we really care about is this hypothesized phenomenon of superposition – does the model represent "extra features" by storing them non-orthogonally? Is there a way to get at it more explicitly? Well, one question is just how many features the model learns to represent. For any feature, whether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

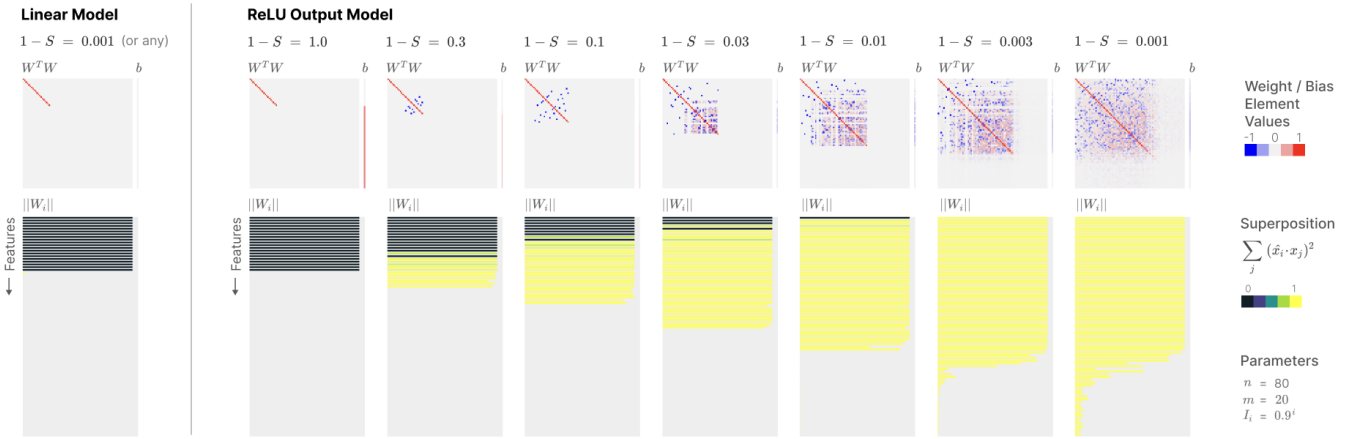


Now that we have a way to visualize models, we can start to actually do experiments. We'll start by considering models with only a few features ($n = 20;\ m = 5;\ I_i = 0.7^i$). This will make it easy to visually see what happens. We consider a linear model, and several ReLU-output models trained on 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. [26] 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,[10] revealing the following equation: