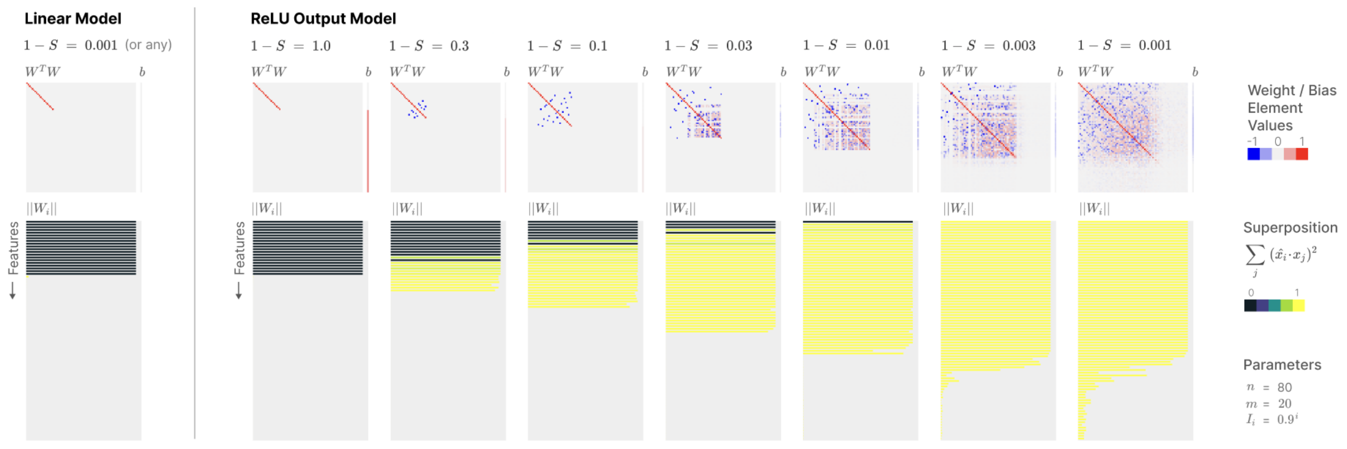
As our standard intuitions would expect, the linear model always learns the top-m most important features, analogous to learning the top principal components. The ReLU output model behaves the same on dense features ($1 - S = 1.0$), but as sparsity increases, we see superposition emerge. *The model represents 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: