Superposition in a Privileged Basis

So far, we've explored superposition in a model *without a privileged basis*. We can rotate the hidden activations arbitrarily and, as long as we rotate all the weights, have the exact same model behavior. That is, for any ReLU output model with weights W, we could take an arbitrary orthogonal matrix O and consider the model $W' = OW$. Since $(OW)^T(OW) = W^TW$, the result would be an identical model!

Models without a privileged basis are elegant, and can be an interesting analogue for certain neural network representations which don't have a privileged basis – word embeddings, or the transformer residual stream. But we'd also (and perhaps primarily) like to understand neural network representations where there are neurons which do impose a privileged basis, such as transformer MLP layers or conv net neurons.

Our goal in this section is to explore the simplest toy model which gives us a privileged basis. There are at least two ways we could do this: we could add an activation function or apply L1 regularization to the hidden layer. We'll focus on adding an activation function, since the representation we are most interested in understanding is hidden layers with neurons, such as the transformer MLP layer.

This gives us the following "ReLU hidden layer" model:

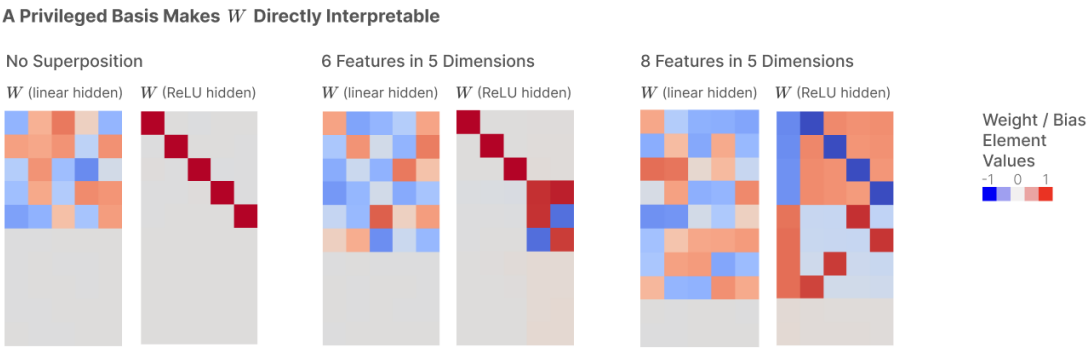
$$h = \text{ReLU}(Wx)$$

$$x' = \text{ReLU}(W^Th + b)$$

We'll train this model on the same data as before.

Adding a ReLU to the hidden layer radically changes the model from an interpretability perspective. The key thing is that while W in our previous model was challenging to interpret (recall that we visualized W^TW rather than W), W in the ReLU hidden layer model can be directly interpreted, since it connects features to basis-aligned neurons.

We'll discuss this in much more detail shortly, but here's a comparison of weights resulting from a linear hidden layer model and a ReLU hidden layer model:



Recall that we think of basis elements in the input as "features," and basis elements in the middle layer as "neurons". Thus W is a map from features to neurons.

What we see in the above plot is that *the features are aligning with neurons in a structured way*! Many of the neurons are simply dedicated to representing a feature! (This is the critical property that justifies why neuron-focused interpretability approaches – such as much of the work in the original Circuits thread – can be effective in some circumstances.)

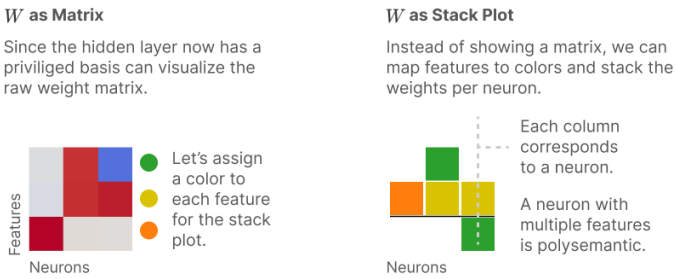
Let's explore this in more detail.

VISUALIZING SUPERPOSITION IN TERMS OF NEURONS

Having a privileged basis opens up new possibilities for visualizing our models. As we saw above, we can simply inspect W. We can also make a per-neuron stacked bar plot where, for every neuron, we visualize its weights as a stack of rectangles on top of each other:

- Each column in the stack plot visualizes one column of W.
- Each rectangle represents one weight entry, with height corresponding to the absolute value.
- The color of each rectangle corresponds to the feature it acts on (i.e. which row of W it's in).
- Negative values go below the x-axis.
- The order of the rectangles is not significant.

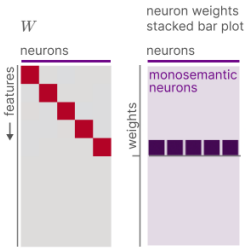
This stack plot visualization can be nice as models get bigger. It also makes polysemantic neurons obvious: they simply correspond to having more than one weight.



We'll now visualize a ReLU hidden layer toy model with $n = 10;\ m = 5; I^i = 0.75^i$ and varying feature sparsity levels. We chose a very small model (only 5 neurons) both for ease of visualization, and to circumvent some issues with this toy model we'll discuss below.

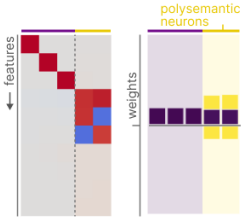
However, we found that these small models were harder to optimize. For each model shown, we trained 1000 models and visualized the one with the lowest loss. Although the typical solutions are often similar to the minimal loss solutions shown, selecting the minimal loss solutions reveals even more structure in how features align with neurons. It also reveals that there are ranges of sparsity values where the optimal solution for all models trained on data with that sparsity have the same weight configurations.

The solutions are visualized below, both visualizing the raw W and a neuron stacked bar plot. We color features in the stacked bar plot based on whether they're in superposition, and color neurons as being monosemantic or polysemantic depending on whether they store more than one feature. Neuron order was chosen by hand (since it's arbitrary).



All Monosemantic Neurons (approximately 0.5 ≤ 1-S)

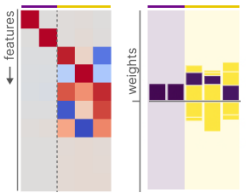
In this regime, every feature gets a dedicated neuron. That is, the model consists entirely of monosemantic neurons.



Three Features in Two Neurons (approximately 0.2 ≤ 1-S ≤ 0.5)

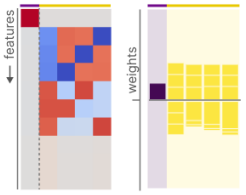
In this regime, the three most important features still get dedicated neurons, but the next three are represented in a kind of binary code by two neuron's activations.

These two neurons form a binary code for three features. Note that the (1,1) feature is orthogonal to the others, while the other two are an antipodal pair.



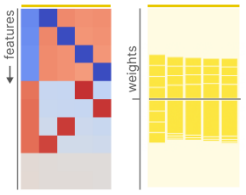
Five Features in Three Neurons (1-S = 0.15)

At this sparsity level (which seems quite narrow), we still see two monosemantic neurons, and three polysemantic neurons. The three polysemantic neurons imlement a code that doesn't have any very simple explanation.



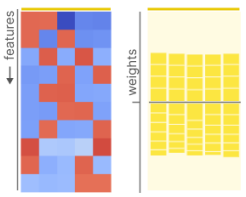
Six Features in Four Neurons (1-S = 0.12)

In this regime, we have one monosemantic neuron. The four polysemantic neurons implement an interesting code. One neuron seems to distinguish between important and unimportant features. Important features are then encoded as sets of two of the other three neurons, while unimportant features are represented by one neuorn.



Eight Features in Five Neurons (0.05 ≤ 1-S ≤ 0.08)

In this regime, all neurons are polysemantic. The code is a kind of extension of the one described previously: one distinguishes imporant and unimportant features. Then either sets of three of the other neurons, or a single other neuron, are used to distinguish the specific feature.

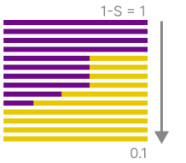


Features are Pairs of Neurons (1-S ≤ 0.04)

In this regime, each feature simply correspond to a *pair of neurons*.

Presumably if there were more features, increasing sparsity would eventually produce a dense binary code. But with only 10 features, this is the densest code that forms.

As feature sparsity increases, we see neurons shift from being monosemantic to being polysemantic.



Individual features are colored based on whether they're in superposition