Notes on Toy Models of Superposition

Alexander van Spaendonck

February 2025

This brief summary will not go into all details of the toy model that features in "Toy Models of Superposition", as the authors provide a very thorough and readable explanation of their work. However, I would like to use this summary to briefly explain the graphs that the notebook produces and reflect on these results from mathematical point of view.

The model essentially maps an n-dimensional vector $\vec{x}$ into an m-dimensional space and back again, where $m < n$.

$$\mathbb{R}^n \xrightarrow{W} \mathbb{R}^m \xrightarrow{W^T} \mathbb{R}^n \tag{1}$$

We interpret the input $\vec{x} \in \mathbb{R}^n$ as containing information about n different features, and we are interested in seeing if our model can recover these features from the hidden layer ($\mathbb{R}^m$). The paper considers two particular variations of this model: one without activation functions (called 'linear') and one with (called 'nonlinear'):

$$\begin{aligned} \text{Linear:} \qquad & x' = xWW^T + b \\ \text{Nonlinear:} \qquad & x' = \text{ReLU}(xWW^T + b) \end{aligned} \tag{2}$$

After doing this, we compute a loss function

$$L(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} I_i(x_i - x_i')^2 \tag{3}$$

which the model tries to minimize. Note that the features are weighted in the loss function and we assume that $I_0 > I_1 > I_2 > .....$

Now let us randomly sample vectors $\vec{x}$, where its components (or 'features') are drawn from a uniform distribution $x_i \in [0, 1]$. We then train the model with these randomly generated vectors and after a while find the following diagrams appear for the linear and nonlinear models:

We are looking at a heatmap for W_iW_j – obtained from the matrix WW^T. What the heatmap indicates, is that within the matrix W the model creates a small orthonormal matrix of four by four (WW^T is an identity matrix) which passes along the four most important features of the input vector to the output. All other features are killed: the weights are set to zero and the biases (depicted in the column besides the grid) are set to cancel the expected values.

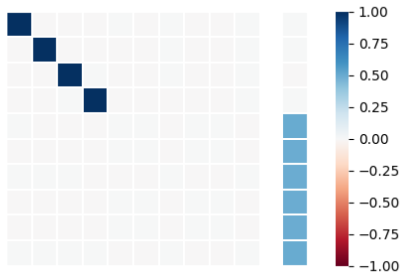

Figure 1: Linear model with sparsity set to zero.

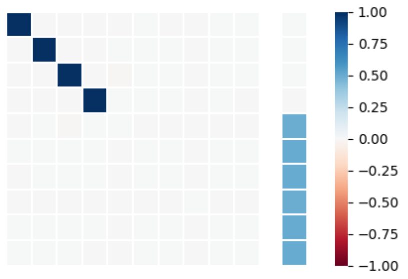

Figure 2: Nonlinear model with sparsity set to zero.

This is what one would expect from a linear algebra: the first transformation W is a projection and the second transformation W^T is an embedding; hence information is lost along the way. Since the first four vectors carry the highest weights I_i, the model decides that in order to minimize the loss function it should simply pass these four vectors along and ignore the others.

What is interesting is when we start introducing *sparsity* S. Most features we can identify in neural networks are sparse, and in order to mimic that behaviour here, we set each feature x_i of the input to zero with a probability of S. When we now increase the sparsity (which originally was zero) to values close to one, an interesting phenomenon takes place within the nonlinear models:

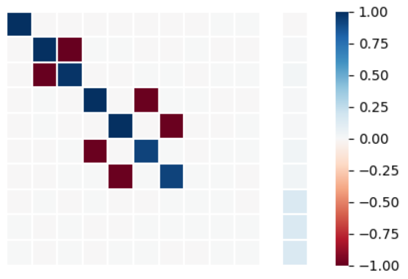

Figure 3: Nonlinear model with sparsity set to 0.7.

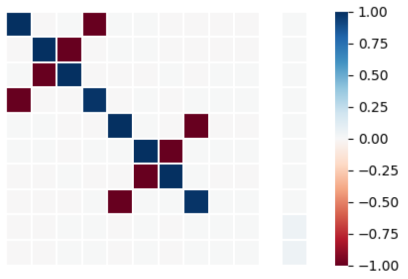

Figure 4: Nonlinear model with sparsity set to 0.9.

The hidden layer starts passing along more than just the four most important features. How does it do this? By producing *antipodal pairs*: The dark red squares in the heatmap indicate weights for which $W_i W_j \simeq -1$ which implies that rather than perpendicular, these features x_i and x_j are mapped to the same dimension in the hidden layer, but opposite to each other. It does this starting with the least important features, but as we increase sparsity the model dares to put more important features in antipodal position as well. This mechanism, where the model maps different features to non-orthogonal directions is called superposition.

To understand why the model does this, one can study the loss function as is discussed in the paper, but there is another argument which I find particularly nice. Let us zoom in on two features x_i and x_j that form an antipodal pair and see how they get mapped from input to output.

$$\mathbb{R}^2 \xrightarrow{w} \mathbb{R}^1 \xrightarrow{w^T} \mathbb{R}^2 \tag{4}$$

The mapping we use is $w = (1, -1)$ resulting in

$$\begin{pmatrix} x_i \\ x_j \end{pmatrix} \xrightarrow{w} (x_i - x_j) \xrightarrow{w^T} \begin{pmatrix} x_i - x_j \\ x_j - x_i \end{pmatrix} \xrightarrow{\text{ReLU}} \text{ReLU} \begin{pmatrix} x_i - x_j \\ x_j - x_i \end{pmatrix} \tag{5}$$

Now when we compute the loss function, what do we find? This ofcourse depends on the values of x_i and x_j which are either zero (with probability S) or a number uniformly drawn from the interval $[0, 1]$. Let's sketch these four cases

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{\text{nonlinear model}} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \text{hence} \quad L = 0$$

$$\begin{pmatrix} x_i \\ 0 \end{pmatrix} \xrightarrow{\text{nonlinear model}} \begin{pmatrix} x_i \\ 0 \end{pmatrix} \qquad \text{hence} \quad L = 0$$

$$\begin{pmatrix} 0 \\ x_j \end{pmatrix} \xrightarrow{\text{nonlinear model}} \begin{pmatrix} 0 \\ x_j \end{pmatrix} \qquad \text{hence} \quad L = 0$$

$$\begin{pmatrix} x_i \\ x_j \end{pmatrix} \xrightarrow{\text{nonlinear model}} \text{ReLU} \begin{pmatrix} x_i - x_j \\ x_j - x_i \end{pmatrix} \qquad \text{hence} \quad L > 0$$

In conclusion, three out of the four cases shown above lead to a perfect recovery of the original features and a loss of zero. Only when both features are nonzero do they interfere with eachother leading to a nonzero training loss. But now comes the important point: the odds of encountering such a sample scales with $P(x_i > 0, x_j > 0) = (1 - S)^2$. Therefore, as we increase the sparsity of our samples, the model finds it more attractive to put features in antipodal position. This is essentially a tradeoff: more antipodal pairings means the model can pass more features to the output (thus loss goes down), but each co-occurence of x_i, x_j creates errors that increase the loss. The balance between these two opposing forces shifts in favor of admitting more features as the sparsity increases.

Finally, when we increase the sparsity even more, we find that configurations even more complicated than antipodal pairs emerge (see the paper for an extensive discussion on this). In our implementation, we see this happening for example when sparsity is set to $S = 0.98$.

Now *all* features are passed along to the output.

1 A priviliged basis

In the previous example, we have seen that with low sparsity the most important features are mapped to orthogonal directions. This basis, however, is not the

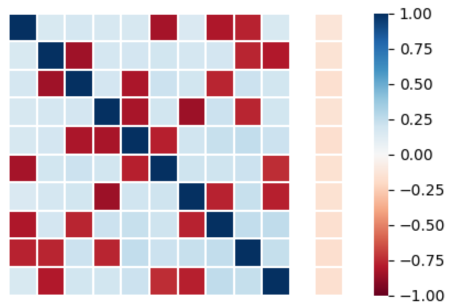

Figure 5: Nonlinear model with sparsity set to 0.98.

canonical basis defined by the four neurons in the hidden layer, but some random rotation of it. In section 7 of the paper, another variation of the model presented in which RelU activitation functions are added after the hidden layer.

$$x' = \text{ReLU}(hW^T + b) \qquad \text{where} \qquad h = \text{ReLU}(xW) \tag{6}$$

The effect of this addition is that the orthogonal basis that the model creates will have a tendency to align with the canonical basis described by the neurons. As a result, one can plot W rather than WW^T and find the following plots:

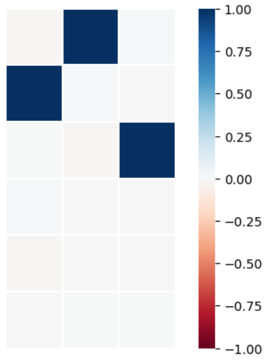

Figure 6: Hidden ReLU model with sparsity set to zero.

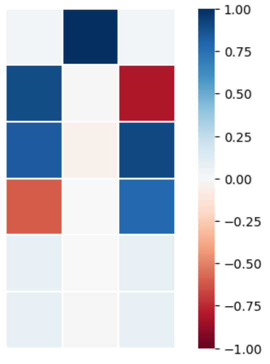

Figure 7: Hidden ReLU model with sparsity set to 0.7.

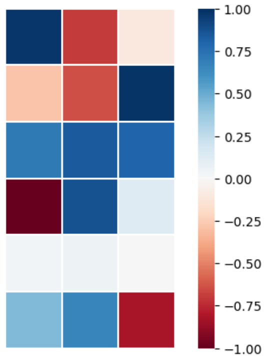

Figure 8: Hidden ReLU model with sparsity set to 0.9.

The models are – as explained in the paper – harder to optimize, hence we consider smaller models here with six features and three hidden neurons. In the first example ($S = 0$) we find that the three most important features align with the three neurons, meaning that each neuron can be identified with one feature (this is called "monosemanticity"). When sparsity increases to 0.7 the model adds a fourth feature, which it puts in superposition with features two and three.

Note that the first and most important feature (top row) is still left untouched. In this case we see that two out of the three neurons (left and right columns) now get contributions from different features (these are "polysemantic") whereas one neuron (middle column) is monosemantic. When sparsity is increased to 0.9 even more features are added: now all neurons are polysemantic, meaning that they "fire" to multiple features.