Notes on Toy Models of Superposition

Alexander van Spaendonck

February 2025

This brief summary does not go into all the details of the toy models that feature 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 some of the plots that the notebook produces and reflect on these results from a mathematical point of view.

1 Demonstrating Superposition

The model essentially maps an n-dimensional vector x into an m-dimensional space and back again, where $m < n$. It does this using a matrix W whose elements we call weights in the context of ML.

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

We interpret the input $x \in \mathbb{R}^n$ as containing information about n different "features" (simply scalars), 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 running the input x through the model, one obtains an output x' from which we can compute the loss function

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

As with any loss function, we train the model to minimize this quantity. The loss function is weighted by the 'importances' I_i of the features x_i. For convenience, we assume $I_i = \alpha^i$ with $0 < \alpha \leq 1$, meaning that feature x_1 is the most important feature and x_n the least.

Now we randomly sample vectors x, where their components (which are the 'features') are drawn from a uniform distribution $x_i \in [0, 1]$. We then train the

model with these randomly generated vectors and plot their resulting weights and biases in figures 1 and 2. We are looking at a heatmap for $W_i W_j$ – obtained from computing 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, as can be seen from the fact that WW^T restricted to its first four components is an identity matrix. Thus, the hidden layer passes along the four most important features of the input vector to the output. All other features are killed: their weights are set to zero and the biases (depicted in the column besides the grid) are set to cancel the expected values.

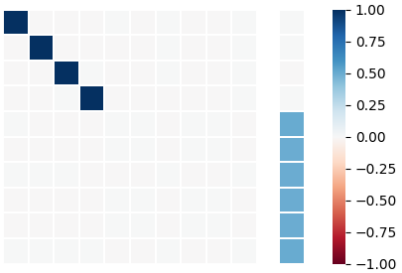

Figure 1: Linear model $(n, m) = (10, 4)$ with sparseness set to zero.

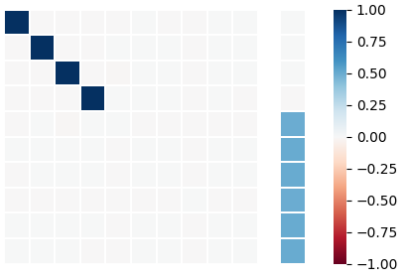

Figure 2: Nonlinear model $(n, m) = (10, 4)$ with sparseness set to zero.

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 *sparseness* 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 sparseness (which originally was zero) to values close to one, an interesting phenomenon takes place within the nonlinear models, seen in figures 3 and 4.

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 sparseness 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

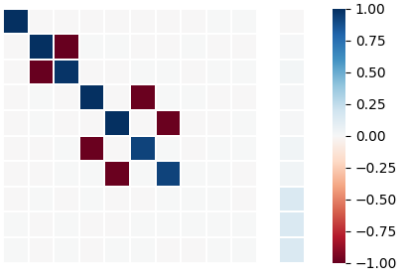

Figure 3: Nonlinear model $(n, m) = (10, 4)$ with sparseness set to 0.7.

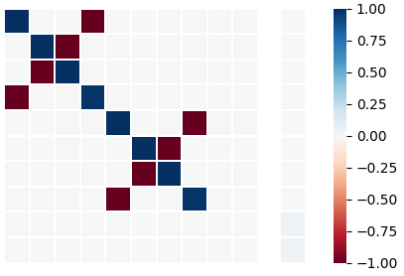

Figure 4: Nonlinear model $(n, m) = (10, 4)$ with sparseness set to 0.9.

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[1] 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 sparseness of our samples, the model finds it more attractive to put features in antipodal position. This is essentially a tradeoff: more antipodal pairings mean the model can pass more features to the output (thus lower its loss), but each co-occurrence of x_i, x_j creates errors that increase the loss. The balance between these two opposing

[1] or any two-dimensional rotation of this vector.

forces shifts in favor of admitting more features when we increase the sparseness. At some point (e.g. when $S = 0.9$ as shown in figure 4), a total of $2m$ features is mapped to antipodal pairs in the hidden layer. If we keep increasing sparseness, the model will try to fit even more than these $2m$ features into the hidden layer by considering even more complicated superpositions (see the paper). In our implementation, for example, we see this happening when sparseness is set to $S = 0.98$ (shown in figure 5) where now *all* features are passed along to the output.

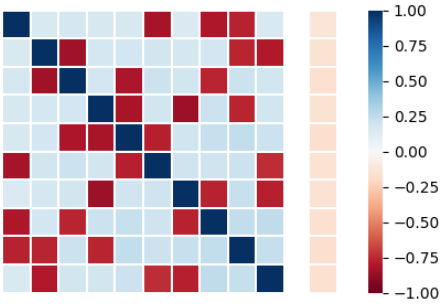

Figure 5: Nonlinear model $(n, m) = (10, 4)$ with sparseness set to 0.98.

2 A Privileged Basis

In the previous example, we have seen that with low sparseness, the most important features are mapped to orthogonal directions in the hiden layer. This basis, however, is not the canonical basis defined by the four neurons residing in the hidden layer, but some arbitrary rotation of it. In section 7 of the paper, another variation of the model is presented in which RelU activation functions are also 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:

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 sparseness 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

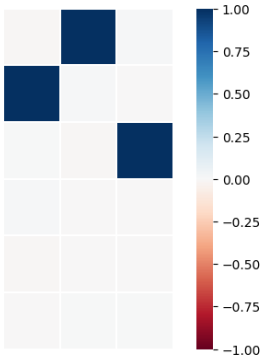

Figure 6: Hidden ReLU model $(n, m) = (6, 3)$ with sparseness set to zero.

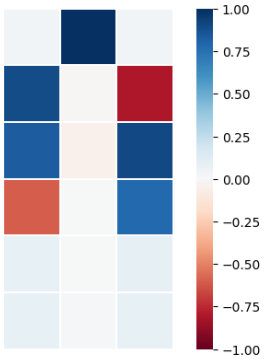

Figure 7: Hidden ReLU model $(n, m) = (6, 3)$ with sparseness set to 0.7.

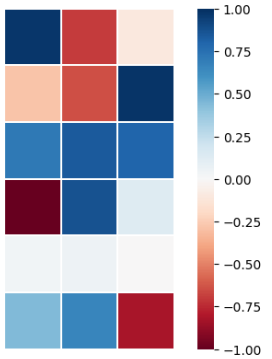

Figure 8: Hidden ReLU model $(n, m) = (6, 3)$ with sparseness set to 0.9.

neuron (middle column) is still monosemantic. When sparseness is increased to 0.9 even more features are added: now all neurons are polysemantic, meaning that they "respond" to multiple features.