
Toy model of superposition control¹

Bartosz Rzepkowski

Independent researcher, Bartosz Rzepkowski Computing, 54-611 Wrocław, Poland

With

PIBBSS, Timaeus, & Apart Research

July 27, 2025

Abstract

One of the most important open questions posed by the authors of the famous "Toy models of superposition" paper [1] is **"how can we control whether superposition and polysemaniticity occur?"** In this research proposal, we present a novel approach, which we hope will be a step forward in answering this question. We will operate under the assumption that each feature corresponds to a single neuron within a neural network layer. This restriction was selected because it is the most desirable one from the perspective of interpretability of neural networks. Under this framework, we will demonstrate how to "handpick" desired features which we want to use in the next layer, and how to impose precise restrictions on how those features should be superposed in the hidden space. This also implies that it is possible to fully prevent superposition if desired.

We hope that the techniques shown in this proposal will allow developers to design blueprints clearly defining which features should be kept at any network depth, how those features should be superposed (if at all), and eventually dynamically adjust this specification. This will effectively allow developers to decide *a priori*, which circuits are allowed to form in the network, rather than discovering them after training is finished. The presented supervised learning algorithm will then force the neural network to follow this blueprint. We present how the whole procedure might work on a simple multilayer feedforward toy model.

The proposed method incorporates tensor network formalism along with Riemannian optimization techniques, both concepts widely used in the field of computational physics.

Keywords: AI safety, interpretability, model superposition, tensor networks, Riemannian optimization

1 Introduction

Neural networks to achieve their goals operate on various *features*. Those features might represent different concepts, depending on the design and purpose of the network. For example, in a Large Language Model a feature might correspond to a word "king", while in an image classifier a more meaningful feature could be a "shape of a dog's ear".

Those features can be defined as directions in an abstract *hidden space* whose size is dictated by the model's architecture. More precisely, at a given depth of the network, the size of this space is dictated by the number of neurons in a given layer. Each feature can be denoted as a direction W_i in this hidden space. For sufficiently large models, it is believed that they will use their capacity in order to not "mix" the features [1]. The "mixing" of the i -th and j -th features can be more formally defined as their *overlap*: $W_i \cdot W_j$. For large models, this overlap should not be present, that is, $W_i \cdot W_j = 0$ for $i \neq j$ and $W_i \cdot W_i = \|W_i\|_2^2$.

The *superposition of features* might appear at a given depth of a model when it tries to encode more features than it has dimensions. It might be the case, when the features are relatively sparse in

¹Research conducted at the AI Safety x Physics Grand Challenge, 2025

the training data. In such a scenario, we might observe that the overlap between different features becomes non-zero, i.e., $W_i \cdot W_j \neq 0$, for $i \neq j$. This superposition makes the interpretation of neural network behavior very complicated.

The mere detection of superposition in a given model is relatively easy. For a given layer in a simple multilayer neural network, the feature directions W_i are encoded as columns of the weight matrix W . Then, in order to detect the superposition one might simply calculate the sum of absolute values of overlaps for all pairs of different features, that is $\sum_{i < j} |W_i \cdot W_j|$. If the result is greater than 0, it is a proof that superposition is present in the model.

However, controlling the superposition in a model is a much harder task which, to the best of our knowledge, does not have a solution as of today. This research proposal tries to make a step forward in addressing this problem by creating a framework which might allow to design interpretable models. To achieve this, we employ various techniques widely used in computational physics and make the following observations:

- a neural network can be treated as a special kind of a tensor network,
- each weight matrix W can be understood as an object from a "weight matrices manifold", on which various constraints can be imposed,
- the $W^T W$ expression represents a "*features* $\times$ *features*" matrix; we can use it as inspiration to design a grid-blueprint allowing to define very precise constraints of the aforementioned manifold (details will be described in Sec. 3),
- using Riemannian optimization techniques it is possible to optimize just a single layer of a neural network at a time, according to the desired cost function,
- be "sweeping" through the neural network we can optimize all of it's layers (inspiration taken from the *Density Matrix Renormalization Group (DMRG)* [2] algorithm),
- the manifold used to optimize the weight matrix W can be arbitrarily adjusted at each step of the training.

This research proposal falls into the **Problem Area 1: Theory/Practice Gap for AI Interpretability**.

2 Motivations

Below we list three reasons that drove this research:

- The main motivation for this work was a "specification first" approach, which means that developers should be able to precisely define **which** features the neural network could use at any depth and **how** to superpose them in the hidden space (if at all). Ideally, this specification should correspond to the clear interpretability of the model.
- This blueprint should not be set in stone, allowing modifications during training, because the initial assumptions might have been wrong.
- Finally, it has been shown that the presence of superposition in the model might increase its robustness [3], which might encourage developers to design smaller models with higher superposition to achieve better performance. We hope that the methods outlined in this proposal will help counterbalance these trends, instead encouraging developers to design more interpretable models, even if it comes at the cost of larger architectures (which effectively increase the hidden space size).

3 Proposed Research

In this research proposal we will focus on a toy feedforward neural network consisting of just a few layers, each followed by an activation function. Example of such a structure is shown in Fig. 1.

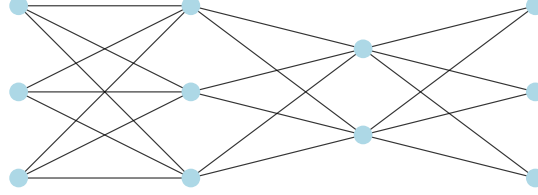


Figure 1: Example architecture of the toy model we will be investigating. The weight matrix W which we will be analyzing can be thought of as "laying between the second and third layer" (see the main text).

The core purpose of this proposal, stated plainly, is to allow developers to impose two conditions on this neural network. Firstly, each neuron should correspond to just a single feature. Secondly, the developer should have a way to precisely define how those features will be superposed in the hidden space of the model and suppress this phenomenon if desired.

3.1 Extracting feature directions from a neural network

Let us assume that each neuron in the second layer of the model in Fig. 1 represents just a single feature. Then, because the next layer has just two neurons, the model might be implied to "compress" the three input features in its lower-dimensional hidden space. The way in which this happens is dictated by the weight matrix W , which describes the transition between the second and third layer.

As mentioned in Sec. 1 features can be described as directions in the hidden space of the model. In our case, those directions are simply encoded as columns of the matrix W . Because the second layer has three neurons, while the third one has only two, the most general form of the weight matrix W will look as follows:

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}. \quad (1)$$

Then, the first feature will be represented by the vector $(w_{11}, w_{21})^T$, the second by the $(w_{12}, w_{22})^T$, and the third by $(w_{13}, w_{23})^T$.

3.2 "Handcrafting" interactions between features

The authors of the paper "Toy models of superposition" [1] give a very intuitive way to see how the features will "interact" with each other in the hidden space by plotting the $W^T W$ matrix. This matrix is the result of two operations. Firstly, W compresses the input features into the lower-dimensional hidden space (or up-scales them, if the next layer has more neurons than the previous

one), and then W^T tries to restore them in their original space. Because of this, the W^TW matrix is of $features \times features$ size.

Below we present a user-friendly way of "handcrafting" different kinds of interaction between various features. (In Appendix A we go through a set of examples, progressing from basic interactions to increasingly sophisticated ones. We highly encourage the reader to go through these examples to get a good intuition on how to interpret the W^TW matrix; however, it is not required to grasp the main idea behind this research proposal).

From the developer perspective, we can for now set aside the origins of the W^TW matrix, and just use it as an inspiration to define a colorful $features \times features$ grid-blueprint, allowing us to impose different restrictions on the interaction of features. The color at the intersection of the i -th row and j -th column will tell us whether the i -th feature should interfere with the j -th one in a constructive way (color red), destructive one (color blue), or whether to suppress the interference at all (white color). Such a grid-blueprint can be represented as a matrix, where red indicates positive values and blue indicates negative ones; stronger hues reflect greater magnitudes.

Using this approach, we show in Fig. 2 four examples of possible interactions. Fig. 2a corresponds to a case in which only the first five features were kept, with no interference among them. In the scenario shown in Fig. 2b the six different features were kept, not in order, also without interference among them. In Fig. 2c, we allow the first five features for some destructive interference. Finally, in Fig. 2d we show a mixture where multiple features were kept and both constructive and destructive interference between them was allowed.

Two important remarks should be made at this point. First, the restrictions imposed by the grid-blueprint on the model cannot be arbitrarily picked and are strongly dependent on the underlying network architecture. Secondly, a restriction given by the color red in the cell located in the i -th row and j -th column might be interpreted as follows: "the i -th and j -th features should interfere constructively with the predefined strength" (given by the color intensity in the diagram). It is an extremely strict constraint. A neural network following such a blueprint during training might not achieve the desired performance. Because of that, **we allow for an extension of the formalism**

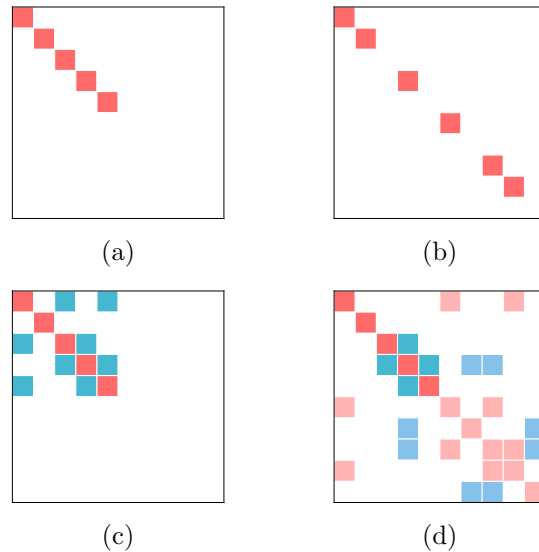


Figure 2: Four different mixtures of interactions among features presented in a grid-blueprint. For details see the main text.

shown in Fig. 2 by saying that it is possible to assign a spectrum of colors to a given cell in the grid-blueprint. Now, assigning a spectrum ranging from light-red through white to light-blue (again in the cell in the i -th row and j -th column) might be interpreted as: "we allow for some constructive, destructive, or no interference at all between these two features". This loosening of constraints might allow the neural network to find suitable parameters allowing it to achieve better performance. For details on the first remark made in this paragraph, see Appendix A, while for the second, see Appendix B.

3.3 Imposing "handcrafted" features interactions on the model

The above formalism might look very convenient, but the reader might ask how do we force the neural network to follow these rules. The answer to this question comes from two observations.

First, the weight matrix W can be interpreted as an object on a Riemannian manifold. Moreover, the specification given by the grid-blueprint can be directly translated into the definition of the manifold of interest! For example, the grid shown in Fig. 2a can be used to define the following manifold of weight matrices:

$$\mathcal{W} = \left\{ W \in \mathbb{R}^{m \times n} \mid W^T W = \begin{bmatrix} I_5 & 0 \\ 0 & 0 \end{bmatrix} \right\}, \quad (2)$$

with $m \geq 5$ and $n \geq 5$. We can interpret the structure of the above manifold as a "set of matrices W , for which $W^T W$ is the same as our grid-blueprint". Then, by optimizing the weights using the Riemannian optimization methods (for example, via the *CDOpt* toolbox [4, 5]) over this manifold, we force the neural network to follow the established specification. However, it should be noted that, typically, a *bias* is added to each layer in a neural network, which is not grasped by the manifold with a structure similar to the one presented in Eq. 2. However, this can be easily fixed by operating in the manifold $\mathcal{M}$ being a product of the weight matrices manifold $\mathcal{W}$ and the bias vectors manifold $\mathcal{B}$, that is, $\mathcal{M} = \mathcal{W} \times \mathcal{B}$.

Second, a matrix is a kind of tensor (second-order one), so we might take inspiration from tensor network algorithms such as DMRG [2], in which we optimize only a single tensor at a time, while freezing all other tensors. Using this approach in our framework, we might optimize the weights and biases of a single layer of the model, according to some loss function $\mathcal{L}(w, x, y)$, while freezing the weights and biases of all other layers. Then, after the optimization converges, or the predefined number of allowed steps is exceeded, we move to the next layer, which we optimize in the same fashion. We proceed in this way through all the layers in the network, optimizing each layer multiple times, until the desired performance is achieved.

3.4 Implementing the "one feature per neuron" constraint

So far we have only imposed restrictions on the way in which the features of one layer are represented in the hidden space (associated with the W matrix) of the next one. However, for the whole framework to work, we also need to force each neuron in the next layer to encode just a single new feature. This in turn relates to the *activation space*. The relationship between the hidden and activation spaces can be stated as

$$A = \phi(Wx + b), \quad (3)$$

where ϕ is the activation function and b is bias. Subsequently, we aim to establish the constraint $AA^T = \mathbf{I}$. This can be easily achieved while optimizing the weights of a given layer by extending the loss function by adding the penalization of multiple neurons firing at the same time:

$$\mathcal{L}_{orth}(w, x, y) = \mathcal{L}(w, x, y) + \lambda \|AA^T - \mathbf{I}\|_F^2, \quad (4)$$

where λ is a hyperparameter. Although we are using the manifold formalism throughout this work, we wanted to draw a clear line separating the ways in which we implement the constraints on the hidden and activations spaces. This prompted us to incorporate the restrictions imposed on the latter in the loss function, rather than to extend the definition of the manifold $\mathcal{M}$. (Moreover, we suppose that it might not be trivial to define a manifold, which at the same time imposes certain restrictions on W , b and A at the same time, due to the nonlinear function ϕ .)

3.5 The training algorithm

In the following, we outline a high-level algorithm that combines all of the previously described methods². It can be used to train a simple multilayer feedforward network within a supervised learning setting. We extend the notation used up to this point by using the upper index to indicate that we are working with the i -th layer of the network. Thus, W^i would correspond to the weight matrix of the i -th layer, while $\mathcal{L}^i$ would be its associated loss function.

Algorithm 1: Training algorithm

Data: a feedforward neural network with n layers, ϵ - threshold on the loss function,
 X - input vectors, Y - expected output vectors
// while desired loss is not yet achieved
while $\mathcal{L} > \epsilon$ **do**
 // iterate through all the layers
 for $i \in \{1, 2, \dots, n\}$ **do**
 1. Design a grid-blueprint for the i -th layer
 2. Define a manifold $\mathcal{M}^i$ according to the grid-blueprint
 3. Define a loss function $\mathcal{L}_{orth}^i$
 4. Using W^i and b^i as a starting point, optimize the weights
 and biases of the current layer over the manifold $\mathcal{M}^i$, according to $\mathcal{L}_{orth}^i$,
 while freezing all other weights and biases in the model

4 Discussion and Conclusion

We hope that the framework presented in this work might be a step towards answering the open question of how we can control whether superposition and polysemanticity occur in neural networks and encourage developers to design more interpretable models. However, a potential drawback of this method could be convergence issues during optimization of each layer. The restrictions imposed on the manifolds might be very strict, which can translate into a landscape in which finding a minimum by the optimizer might appear difficult. Numerical tests should assess the practical applicability of this method.

²For more details and some additional remarks on the training algorithm see the Appendix C.

References

- [1] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy Models of Superposition. September 2022. arXiv:2209.10652 [cs].
- [2] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, January 2011.
- [3] Yizhou Liu, Ziming Liu, and Jeff Gore. Superposition Yields Robust Neural Scaling. May 2025. arXiv:2505.10465 [cs].
- [4] Nachuan Xiao, Xin Liu, and Kim-Chuan Toh. Constraint dissolving approaches for Riemannian optimization. *arXiv preprint arXiv:2203.10319*, 2022.
- [5] Xiaoyin Hu, Nachuan Xiao, Xin Liu, and Kim-Chuan Toh. A constraint dissolving approach for nonsmooth optimization over the Stiefel manifold. *arXiv preprint arXiv:2205.10500*, 2022.

5 Appendix

A Examples of ”handcrafting” features interactions

Below, we present a set of examples showing how different assumptions about the feature interaction are reflected by the $W^T W$ matrix. In all of these scenarios, we will begin with three features f_1, f_2 and f_3 , because there are three neurons in the second layer of the toy model shown in Fig. 1. We will try to map them onto a two-dimensional space, as there are only two vectors in the third layer of the network.

A.1 Picking a subset of features

Developers may decide that they want to keep only the features f_1 and f_2 and discard f_3 . At the same time, they might want to make f_1 and f_2 ”as independent as possible”, which can be achieved by restricting the overlap of their direction vectors W_{f_1} and W_{f_2} to be equal to 0, that is, $W_{f_1} \cdot W_{f_2} = 0$. Good candidates for fulfilling these assumptions are the vectors $W_{f_1} = (1, 0)^T$ and $W_{f_2} = (0, 1)^T$ (these vectors have two entries, because there are two neurons in the third layer). To discard the feature f_3 we assign to it the vector $W_{f_3} = (0, 0)^T$. Then, the W matrix will look as follows:

$$W = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad (5)$$

and the $W^T W$ matrix will take the following form:

$$W^T W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} \textcolor{red}{1} & 0 & 0 \\ 0 & \textcolor{red}{1} & 0 \\ 0 & 0 & 0 \end{pmatrix}. \quad (6)$$

The entry from the i -th row and j -th column in the $W^T W$ matrix tells us how the features f_i and f_j interfere (how they overlap). This also implies that the matrix $W^T W$ is symmetric along

its diagonal (f_i has to interfere with f_j in the same way as f_j with f_i)³. The diagonal entries tell us how the given feature interferes with itself (so these values will be simply $\|W_{f_i}\|_2^2$), whereas the off-diagonal entries show us how different features interfere. Because of that, **we are particularly interested in the off-diagonal entries of the $W^T W$ matrix**. Above, we also used the red color to highlight positive entries in the $W^T W$. For negative ones, we will use a blue color.

In the above example we picked the $W_{f_1} = (1, 0)^T$ and $W_{f_2} = (0, 1)^T$ vectors. However, there is nothing special about those two vectors, and a different choice might be as good as the one we made previously. Then, let us instead choose $W_{f_1} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $W_{f_2} = \frac{1}{\sqrt{2}}(1, -1)^T$ (we keep W_{f_3} unchanged). Then we have

$$W = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 1 \end{pmatrix}, \quad W^T W = \begin{pmatrix} \textcolor{red}{1} & 0 & 0 \\ 0 & \textcolor{red}{1} & 0 \\ 0 & 0 & 1 \end{pmatrix}. \quad (7)$$

We can see that the $W^T W$ matrix is exactly the same, although we picked different W_{f_1} and W_{f_2} . This tells us that the grid-blueprint introduced in the main text should be understood as a specification of how the features relate to each other, irrespectively to any predefined basis.

Now, let us slightly modify the above example, by keeping just the features f_1 and f_3 , and discarding f_2 , by assigning $W_{f_1} = (1, 0)^T$, $W_{f_2} = (0, 0)^T$, and $W_{f_3} = (0, 1)^T$. This translates into

$$W = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad W^T W = \begin{pmatrix} \textcolor{red}{1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \textcolor{red}{1} \end{pmatrix}. \quad (8)$$

Thus, we can see that a $W^T W$ matrix with some nonzero entries on its diagonal corresponds to a case in which we keep only a subset of the features. We will use this case as our starting point and expand from here.

A.2 Introducing interference

Let us slightly modify the first example introduced in the previous subsection. We keep the $W_{f_1} = (1, 0)^T$, $W_{f_2} = (0, 1)^T$ unchanged, but modify $W_{f_3} = (-1, 0)^T$. With this setup we have

$$W = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \quad W^T W = \begin{pmatrix} \textcolor{red}{1} & 0 & \textcolor{blue}{-1} \\ 0 & \textcolor{red}{1} & 0 \\ \textcolor{blue}{-1} & 0 & \textcolor{red}{1} \end{pmatrix}. \quad (9)$$

For this particular choice we can see that there is a destructive interference between the f_1 and f_3 features (when summed, those two vectors would result in a $(0, 0)^T$ vector).

A.3 General form of the $W^T W$ matrix

The above simple examples showed how to pick some features and how to allow for their interference. However, we cannot arbitrarily choose the form of the $W^T W$ matrix. For the architecture of the network presented in Fig. 1 it is impossible to define the following $W^T W$ matrix

³One might be encouraged to interpret the $W^T W$ as a density matrix, i.e., an object representing the state of a quantum system. The density matrices do not have to be symmetric along their diagonal, allowing for more exotic quantum interactions. However, we believe that the way the $W^T W$ matrix is constructed implies its symmetry along the diagonal and leave the open question whether it is possible to extend the approach presented in this work to also allow for these quantum properties to emerge. On one hand, this could be interesting from the perspective of studying the expressiveness of neural networks, but on the other hand, it is not recommended from the standpoint of mechanistic interpretability.

$$W^T W = \begin{pmatrix} \textcolor{red}{1} & 0 & 0 \\ 0 & \textcolor{red}{1} & 0 \\ 0 & 0 & \textcolor{red}{1} \end{pmatrix}, \quad (10)$$

because there are too few neurons in the third layer. More precisely, in a two-dimensional space we cannot find three linearly independent (orthogonal) vectors. To arrive at the structure of the matrix $W^T W$ shown in Eq. 10, we would need to introduce the third neuron into the third layer of the network.

This gives us the following rule of thumb: **whenever we try to use more features than there are neurons in the layer, for which we are creating the $W^T W$ matrix, we have to introduce some interference.**

The problem of defining a proper $W^T W$ matrix becomes simpler when a given layer in the neural network has more neurons than the previous one. In such a case, we are "up-scaling" the input features into a higher-dimensional space, which allows for a more robust form of the $W^T W$ matrix. This is also true when consecutive layers have the same number of neurons (however, in this case we cannot talk about the "up-scaling" of features).

B The extended version of the grid-blueprint

To translate the simple version of the grid-blueprint (with just a single color per cell allowed) into the definition of a weight matrices manifold, we can use the following general formula:

$$\mathcal{W} = \{W \in \mathbb{R}^{m \times n} \mid W^T W = G\}, \quad (11)$$

where G denotes the grid-blueprint. Now, in order to allow for a spectrum of colors to be assigned to a single cell, we might want to extend the above definition as follows

$$\mathcal{W} = \left\{ W \in \mathbb{R}^{m \times n} \mid W^T W \stackrel{\text{e.w.}}{\geq} G_{min} \wedge W^T W \stackrel{\text{e.w.}}{\leq} G_{max} \right\}, \quad (12)$$

where G_{min} (G_{max}) corresponds to a grid-blueprint, in which we chose the smallest (largest) possible value in each cell, while $\stackrel{\text{e.w.}}{\leq}$ and $\stackrel{\text{e.w.}}{\geq}$ operators indicate that we are comparing the matrices element-wise.

C Details of the training algorithm

The following remarks are in place for the proposed training algorithm:

1. During optimization of a given layer of the network, only a subset of X and Y should be taken into account, as including all the training data at the same time would result in an extremely costly computation of the loss function.
2. In the main text, we presented a simplified version of "sweeping" through the neural network, i.e., we suggested that in each sweep we should go from the first layer up to the last one and then start again from the first one. However, a better approach would be to sweep back and forth through the layers, as closely located layers are likely to exert a stronger influence on one another, effectively resulting in a better convergence of the whole method.