TPLP: Page 1–. © The Author(s), . Published by Cambridge University Press
doi:

Decomposing Activations into Features: How Many and How do we Find Them? — A Survey

KAAREL HÄNNI*†
Notodai Research, SERI MATS, Caltech

JAKE MENDEL‡†
Notodai Research

KAY KOZARONEK*†
Notodai Research, SERI MATS

submitted 23 05 2023

Abstract

One of the main goals of interpretability is to reverse-engineer neural networks. For this, a reasonable subgoal is to determine the "variables in terms of which a neural network thinks". In this paper, we run with the assumption that these variables correspond to particular vectors in activation space, which we call 'features', whose linear combinations are the activation vectors on various inputs. We provide an exposition of certain techniques from the literature for reconstructing these features from activation data, discussing both approaches grounded in the assumption of there being many features, few of which are active on any one input; as well as approaches grounded in the assumption that there are relatively few features. We finish by suggesting experiments that might let us figure out which of these two options is closer to the truth.

KEYWORDS: interpretability, superposition, feature decomposition, AI alignment

1 Introduction

Even though we broadly understand how to train models that do well on certain tasks, our understanding of what algorithms these neural networks learn is currently poor. Although we can inspect the weights of a neural network, thereby obtaining a perfect low-level description of what a trained model does, we presently lack methods for understanding the algorithms implemented in trained models at a level of abstraction above that of giant inscrutable matrices — in particular, for reducing neural networks to human-understandable computer programs. AI interpretability sets out to develop such an understanding. We see this as a promising and important research direction for ensuring the safety and alignment of future (superintelligent) AI systems.

For translating neural networks into human-understandable computer programs, a promising first step is understanding the variables that would appear in these programs,

* funded by a 6-month research grant from LTFF
† equal contribution
‡ funded by Open Philanthropy Individual Grants Program

to borrow a framing outlined in Olah (2022). Olah suggests seeing activations as storing the values of these natural variables in a neural network's thinking. The question that then arises is how the variables are represented, or stored, in the activations of a model. In this paper, we will make the assumption that variables correspond to directions in activation space, or equivalently, to unit vectors $\mathbf{f_1}, \ldots, \mathbf{f_k}$ in activation space; we call these vectors the *features* of that activation space.[1] We make the further assumption that features are stored in activations linearly, by which we mean that the activation vector in each activation space of a model[2] is a linear combination of these feature vectors (for that particular activation space), with coefficients telling us which of the features are active. That is, $\mathbf{a} = \sum_{j=1}^{k} c_j \mathbf{f_j}$.[3] In the present paper, we will mostly be taking all of these assumptions for granted, and making the realistic assumption that we know a dataset of activation vectors $\mathbf{a_1}, \ldots, \mathbf{a_n}$, but the features $\mathbf{f_j}$ and coefficients c_{ij} in $\mathbf{a_i} = \sum_{j=1}^{k} c_{ij} \mathbf{f_j}$ are a priori unknown to us — these are exactly the hidden true variables we would like to be able to figure out. This brings us to the main two questions this paper hopes to address:

1) Given just a dataset of activation vectors, how might one reconstruct the features?
2) Are activation vectors made out of few features, or are they made out of many?[4]

Here is what we mean by the above more precisely. In the first question, we assume we have (generated by running some representative sample of inputs through the neural network up to a particular layer in which activations are recorded), and we are looking for ways to determine the correct linear decompositions of activation vectors, $\mathbf{a_i} = \sum_{j=1}^{k} c_{ij} \mathbf{f_j}$. That is, we want methods for finding such linear decompositions which are useful for understanding the neural network's computation. We will provide a (non-exhaustive) review of some methods proposed in the literature for doing this. All the methods we will consider will only use activation data — i.e., not the weights of the model (beyond using the weights to compute the activations). It will turn out that the methods can be neatly partitioned into those that are easiest to make sense of if one assumes there are **many** features, by which we mean that k is **at least** d, the dimension of the activation space — we discuss this case, sometimes referred to as 'superposition', in Section 2; and those that are easiest to make sense of if one assumes there are **few** features, by which we mean that k is **at most** d — we discuss this case in Section 3. As we see the superposition case as being relatively well-represented in recent interpretability discourse already, we have decided to dedicate more space to the case with few features in this paper.

As for the second question — i.e., whether there are few features, or many — we see it as roughly corresponding to the question of whether the superposition picture is correct, or more precisely as roughly corresponding to whether representations are superpositional

[1] We will use the convention that all vectors are row vectors by default.
[2] for instance, the activations in hidden layer 2 of a feedforward MNIST classifier, or the activations in the residual stream of a transformer language model at the 5th layer and 10th token position
[3] An example: when a vision model performs computation on a picture of a dog, the activation vector in a particular layer might be a linear combination of a "dog fur at this part of the picture" feature, a "patch of grassy texture in this position" feature, a "partial dog nose" feature (Nanda (2023)), and a number of other features, possibly including ones not as easily understandable to humans.
[4] possibly sparsely in the latter case — that is, we are not saying that any individual activation vector has many nonzero c_{ij}

or compositional (from Olah (2023)), though the focus of our framing is slightly different. We begin to address this question in Section 4 by proposing experiments that might let us figure out which of these two options is closer to the truth.

2 Decomposition methods: the case of many features

Let us first very briefly consider some methods for decomposing activations into features that make sense if we assume there are many features. In this setting, while there are typically many ways to decompose any (activation) vector as a linear combination of the features — for instance, one can typically write any activation vector as a linear combination of any d arbitrarily chosen features — it is canonically thought (Elhage *et al.* (2022)) that activation vectors are *sparse* linear combinations of features, meaning that in $\mathbf{a_i} = \sum_{j=1}^{k} c_{ij}\mathbf{f_j}$, only a small number of the coefficients c_{ij} should be nonzero. The thought is that while there are in principle many concepts that could be active in a model, on any single input, only a small number of the concepts are active. Restating this as an observation about the true features: we expect the set of features to have the property that any activation vector is a sparse (that is, with few nonzero entries) linear combination of the features. To our knowledge, the best idea from the literature for finding the true features is to simply look for a set of features that satisfies this property. Sharkey *et al.* (2023) search for such a sparse representation of activation vectors by training a sparse autoencoder. And in fact, they manage to closely recover the ground truth feature decomposition in a toy setting.

In the experiment with toy data, the authors first construct a set of random unit norm features $\mathbf{f_1}, \ldots, \mathbf{f_k}$, and then let the activation vectors be random sparse linear combinations of these features.[5] They then train a sparse autoencoder on activation data alone that aims to recover the features by looking for any sparse basis. The sparse autoencoder they use is a 2-layer MLP where the first and last layer have the same number of neurons, and the middle layer has more. More precisely, the architecture they use is $\tilde{\mathbf{a}} = \mathrm{ReLU}\left(\mathbf{a}\mathbf{W} + \mathbf{b}\right)\mathbf{D}$, where $\mathbf{W}, \mathbf{b}, \mathbf{D}$ are trainable parameters, with the constraint that the rows $\mathbf{d_1}, \ldots, \mathbf{d}_\ell$ of $\mathbf{D}$ are unit vectors.[6] Here, we think of the activation vector $\mathbf{a}$ as the input, and we denote the output $\tilde{\mathbf{a}}$. We do so because the loss we choose incentivizes $\tilde{\mathbf{a}}$ to be a close to $\mathbf{a}$:

$$L = L_{\text{reconstruction}} + L_{\text{sparsity}} = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{a_i} - \tilde{\mathbf{a}}_\mathbf{i}||_2^2 + \alpha\frac{1}{\ell}\,||\mathrm{ReLU}\left(\mathbf{a_i}\mathbf{W} + \mathbf{b}\right)||_1$$

The L^1 term in the loss incentivizes the mass in the middle layer to be spread sparsely. We can thus think of $\tilde{\mathbf{a}}_\mathbf{i} = \sum_{j=1}^{\ell} \mathrm{ReLU}\left(\mathbf{a}\mathbf{W} + \mathbf{b}\right)_j \mathbf{d_j}$ as the sparse linear decomposition of activation vector $\mathbf{a_i}$ in terms of features $\mathbf{d_j}$. The hope is that this lets us recover the ground truth features. And, at least in the toy setting, it does turn out to recover correct features quite well, as can be evaluated (assuming access to the ground truth features)

[5] There is more to be said about what this random process was like exactly — for instance, about what the authors did to make the presence of features be correlated and for the features to show up with varying frequencies — but we will not go into the details here.

[6] Recall that our convention is that all vectors are row vectors by default.

by noting that the average over ground truth features $\mathbf{f_j}$ of the maximum dot product with some $\mathbf{d_j}$ is close to 1, at least with the right choice of the hyperparameters ℓ and α.

When one tries to apply this to real models, one faces the problem of having to choose these two hyperparameters correctly without having access to ground truth features.[7] The authors come up with a few observable proxies that seem to correlate with whether correct features are recovered in toy models, though the results for real models remain mixed. For another example of sparse decomposition in a non-toy model, we refer the reader to Goh (2023).

3 Decomposition methods: the case of few features

In this section, we explore techniques for identifying interpretable directions in activation space in the case when the number of encoded features is *smaller* than d, the dimension of the activation space. In the special case when each neuron encodes a unique feature, these directions are just the basis vectors of the neuron basis. However, in general, even if there are fewer features than neurons, it is still possible that features are not encoded as directions in the neuron basis.[8] In this general case, we would like to establish tools for identifying the feature directions. In this section, we explore a family of 3 such methods: singular value decomposition (SVD), principal component analysis (PCA), and non-negative matrix factorization (NMF).

3.1 Mathematical Techniques

Suppose that the true number of features is $k < n$. That is, we hypothesize that each of the n activation vectors $\mathbf{a_i}$ (with $i = 1, \ldots, n$) can be reconstructed using a linear combination of feature vectors $\mathbf{f_j}$ (with $j = 1, \ldots, k$) with a set of k coefficients c_{ij}. This is quite a surprising property for a set of $n \gg d$ vectors to have. Again, in analogy with the reconstruction method in the many-features section, our plan is to try to find the correct features by just looking for a set of features which satisfies a surprising property about how activations should be composed of features, which the correct set of features satisfies. In this case, as the true features would let us recover all $\mathbf{a_i}$, we will be searching for a canonical set of features with this property.

3.1.1 SVD

The most straightforward approach to this problem is to find choices of features and coefficients which minimize the distance between estimates $\tilde{\mathbf{a}}_i = \sum_{j=0}^{k} c_{ij}\mathbf{f}_j$ and $\mathbf{a}_i$. If the distance metric we use is the sum of the magnitude of the vectors $\mathbf{a_i} - \tilde{\mathbf{a}}_\mathbf{i}$, then we are trying to solve the optimization problem $\min_{\{\mathbf{f}_j\}} \min_{\{c_{ij}\}} \sum_{i=1}^{n} \left|\left|\mathbf{a}_i - \sum_{j=1}^{k} c_{ij}\mathbf{f}_j\right|\right|^2$.

In order to solve this minimization problem, we want to find the k directions in activation space in which the summed squared components of the vectors $\mathbf{a}_i$ in these directions

[7] in particular, even if one assumes there are more than d features, this does not tell one exactly how many there are

[8] In fact, in certain cases — for instance, for the residual stream of a transformer — there is little privileging any basis.

are largest on average, so that we can subtract the components of the activation vectors in these directions (i.e. choose $c_{ij} = \mathbf{a}_i \cdot \mathbf{f}_j$) to leave only the $d-k$ components which are smallest.

We can write this problem in an equivalent way using matrices. If d is the dimension of the activation space, then define matrices $\mathbf{A}, \mathbf{F}, \mathbf{C}$ by $(\mathbf{A})_{il} = (\mathbf{a}_i)_l, (\mathbf{C})_{ij} = c_{ij}, (\mathbf{F})_{jl} = (\mathbf{f}_j)_l$ (with $l = 1, \dots d$) and the problem can be rewritten as finding $\mathbf{F}, \mathbf{C}$ to optimize:

$$\min_{\mathbf{F},\mathbf{C}} ||\mathbf{A} - \mathbf{C}\mathbf{F}||_F^2 \tag{1}$$

where the subscript F denotes Frobenius norm of the matrix. That is, we are trying to find an approximate factorization of the A matrix with shape $n \times d$ into a product of matrices with shapes $n \times k$ and $k \times d$, which necessarily is rank at most k.

To solve this minimization theorem we can use the Singular Value Decomposition of $\mathbf{A}$: $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\intercal$ where $\mathbf{U}, \mathbf{V}$ are Orthogonal matrices with sizes $n \times n$ and $d \times d$ respectively and Σ is a rectangular diagonal matrix of size $n \times d$ with non-negative entries on the diagonal (the singular values of A). Then, ordering the basis so that the elements of Σ are decreasing down the diagonal, we define $\tilde{\Sigma}, \tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ by taking only the first k columns (and rows in the case of $\tilde{\Sigma}$) from Σ, $\mathbf{U}$ and $\mathbf{V}$ respectively. Then, the Eckart-Young-Mirsky Theorem (Eckart & Young (1936)) states that the solution to the minimization problem (1) is $\mathbf{C} = \tilde{\mathbf{U}}\tilde{\Sigma}, \mathbf{F} = \tilde{\mathbf{V}}^\intercal$.

Intuitively, the Eckart-Young Theorem makes sense: if you want to find the best set of k features to describe the set of activation vectors $\mathbf{a}_i$, you should pick the directions (right singular vectors) with the top k singular values of the set $\mathbf{a}_i$. This also has the nice property that when we are unsure about the true number of features encoded by the model, adding more features won't change the features we have already found.

PCA is a close cousin of SVD that, in our framing, can be thought of as taking one of the features in every activation vector to be the mean of all the activation vectors to begin with, and then performing SVD on the vectors remaining to be decomposed.

3.1.2 NMF

Non-negative matrix factorization is an important modification of SVD in which the components of the feature vectors $(\mathbf{f}_j)_l$ and the coefficients with which they are added to reconstruct activations c_{ij} are required to be non-negative, i.e. we look to optimize $\min_{\mathbf{F},\mathbf{C}\geq 0} ||\mathbf{A} - \mathbf{C}\mathbf{F}||_F^2$ where $\mathbf{F}, \mathbf{C} \geq 0$ means that these matrices only have non-negative elements.

There are a few reasons why we might want to impose this restriction on $\mathbf{C}$ and $\mathbf{F}$:

- Assuming the nonlinearity in the neural network is ReLU, the components of each $\mathbf{a}_i$ are positive (in the neuron basis), which is compatible with it being a product of non-negative matrices. It may be hard for the model to encode meaning in directions which have negative components in the neuron basis. There is a sense in which this would imply that the components "do not have independent meaning", as it would be impossible for there to be an input on which only a negative feature is active.
- Insisting that activation vectors can only be composed out of positive linear combinations of features may result in features corresponding to more human-

interpretable concepts Olah et al. (2018); Alammar (2020), because these features can either be present or not present in an activation vector, but they cannot be present with a negative coefficient.

Based on these reasons and the empirical observation that NMF sometimes leads to more interpretable directions than the more general SVD, this technique has been often used (Hilton et al. (2020); McGrath et al. (2021); Olah et al. (2018)) to identify features in neural networks. Finding the global minimum of the NMF optimization problem is known to be NP-hard in general (Ding et al. (2005)), but local minima can be found by numerical methods. Unlike SVD, NMF comes with no guarantee that the feature directions we find when we assume there are $k+1$ features will contain the feature directions found under an assumption of k features. This means that our ability to identify with NMF the true features used for computation may depend on correctly identifying the number of features that the model is using.

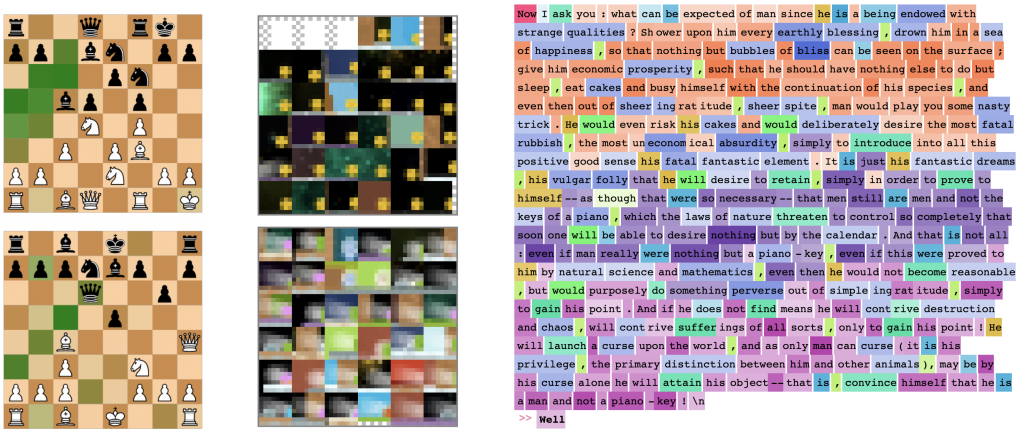
3.2 Applications

The techniques introduced in this section have been used several times to identify human-interpretable feature directions in the activation space of models. In this section we demonstrate how to do so, following the procedure used by McGrath et al. (2021) to find interpretable features in the activations of AlphaZero. AlphaZero has a Resnet architecture with 20 blocks followed by a policy head and a value head. Each block consists of (Conv, ReLU) $\times 2$ and a skip connection (add, ReLU), and each convolution has 256 channels and a kernel size of 3×3 with stride 1. Therefore the output of each block is a tensor of activations of size $8 \times 8 \times 256$, or equivalently, a 256 dimensional vector for each board square. To identify feature directions, the authors followed the following steps:

1. Pick a particular set of activations (the output of one of the blocks) to study.
2. Sample activations for 10000 game states and form a set of $8 \times 8 \times 10000 = 640000$ activation vectors. Stack these vectors into a matrix $\mathbf{A}$ with shape (640000, 256).
3. Choose a number of features to decompose the activation vectors into. After experimenting with a range of possible numbers, the authors chose 36 features.
4. Use NMF to decompose $\mathbf{A}$ into a 640000×36 matrix of coefficients $\mathbf{C}$ and a 36×256 matrix $\mathbf{F}$ of 36 feature vectors.
5. To visualise these features, pick a board state and a particular feature, and extract the elements of $\mathbf{C}$ which correspond to the coefficients of that feature for each of the 64 squares on the board. Then visualise these coefficients by superimposing them onto the board state. An example of such a visualization is shown in figure 1.

This process is not specific to studying AlphaZero at all. For example, Hilton et al. (2020) and Alammar (2020) used NMF to interpret RL vision and language models respectively, with some small modifications:

- In step (1), Hilton et al. (2020) uses the activations at the outputs of one of the convolutional layers. Alammar (2020) uses the activations in the MLP in one of the layers of the transformer.
- In step (5), Hilton et al. (2020) visualises a particular feature by finding images (observations) in which some part of the image has a high coefficient associated with



(a) (b) (c)

Fig. 1: Visualising features in (A) AlphaZero, (B) an RL agent trained on Coinrun, and (C) a Language Model. Many of the features are at least somewhat human-interpretable. (1a) Coefficients of an identified feature for each of the 64 activation vectors for two given board states, superimposed on the board. The higher the coefficient, the darker the shade of green. This feature is 'developed diagonal moves for opponent' (McGrath et al. (2021)). (1b) The region of the RL agent's observations which corresponds to an activation vector with high coefficient of a given feature, visualised for multiple observations. Since each convolution mixes together nearby activations, the region of the observation identified is more than one pixel. The feature visualised at the top is 'coin' and at the bottom 'agent or enemy moving right' (Hilton et al. (2020)). (1c) Each token input into a transformer coloured by which feature is present in that token's activation vector with highest coefficient. Features include words at the beginning, auxiliary verbs, pronouns, and so on (Alammar (2020)).

the feature, and showing those parts of the observation. Alammar (2020) visualises features by superimposing colours on top of each token corresponding to which identified feature is present with highest coefficient (figure 1).

3.3 Modifications

Other sources use modified versions of the techniques introduced here to identify feature directions:

Using relevance to model prediction. Graziani et al. (2023) identify feature directions in Inception V3 (a 42 layer Convnet trained on ImageNet) not based solely on singular directions with the largest singular values, but by finding which singular directions the output depends on most. Having performed SVD on $\mathcal{A}$, they calculate the rate of change of the model output with respect to each of the singular directions. That is, defining c_{ij} as the coefficient of the jth singular vector in activation $\mathbf{a}_i$, they multiply the jth singular value σ_j by $\frac{1}{n}\sum_{i=1}^{n}\frac{\partial\psi}{\partial c_{ij}}$ where ψ is the output of the model, which for now we assume is one-dimensional (but the technique generalises to higher dimensional outputs – see the paper for more details). Keeping the directions with the k highest values of

$\sigma_j \cdot \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \psi}{\partial c_{ij}}$ now corresponds to keeping the singular directions whose variance is responsible for the highest variance in the output.

Using Canonical Correlation Analysis. Another variation is to use a technique from this section to identify which subspace of activation space to project down to, but to use a different technique to identify feature directions in this subspace. Raghu et al. (2017) used SVD on activation vector sets in 2 different layers to reduce the dimension of each activation space as much as they could while retaining 99% of the variance. Then, they use Canonical Correlation Analysis (Hardoon et al. (2004)) to chose a basis in each subspace to maximize the correlation between basis vectors.

4 How could we tell whether there are few features or many?

There are conceptual reasons, as well as empirical evidence (including important evidence not mentioned in the previous two sections), for there being many features, and for there being few. But we will not present a discussion of this here. Instead, we will very briefly state a list of ideas for experiments that might distinguish between the two hypotheses.

- Train a model with n neurons at layer l. Then retrain the network, holding all weights fixed except $W_{l-1,l}$ and $W_{l,l+1}$ which are re-initialised. Do small k techniques like SVD find the same features both times (as measured by cosine similarity of features sorted by the canonical ordering of each singular vector set)? What about large k techniques like sparse autoencoding?
- Repeat the first experiment, but use $D > d$ neurons at layer l when retraining. Now we want the decomposition in the narrower layer to be the same as the decomposition into the top k features in the wide layer (in generalm we expect that the number of features learned in the wide model is likely to be larger than in the narrow model). Does this work better when our decomposition is with a small k technique or a large k technique?
- Train pairs of (model with ReLU at layer l, model with ReLU and a filter which sets to zero all but the top m activations at layer l) for a range of values of d. If the top-m filter forces features to be aligned to the neuron basis, then the models should only have similar performance when the number of features $k \leq d$.
- We'd like to improve our confidence that sparse autoencoders have found the real features. If we train the sparse autoencoder twice, do we get the same decomposition of activation vectors? If we fix some weights on the sparse autencoder to hardcode some random features into the model, but we learn their biases, does the model choose to ignore them from its feature set by setting their bias really low so the neurons never activate?
- Is there something correct in both the large-k and small-k pictures? For instance, instead of doing SVD and also instead of using a sparse autoencoder for feature decomposition, train an autoencoder in which some number $\delta < d$ of neurons have no L^1 penalty, and the rest do. The thought is that these δ features might pick out the top δ compositional directions, and the rest will be used for capturing the features in superposition. Will we find that the features are more interpretable this way than in either the dense or sparse extreme?

Acknowledgments

This is a project of Notodai Research, an AI alignment research team consisting of Kaarel Hänni, Georgios Kaklamanos, Kay Kozaronek, Walter Laurito, and Jonathan Ng, with Jake Mendel as a visiting researcher. We would particularly like to thank Joe Benton, Aryan Bhatt, Lucius Bushnaq, Vera Gahlen, Georgios Kaklamanos, David Reber, Joshua Reiners, Lee Sharkey, and Jörn Stöhler, for valuable conversations.

References

Alammar, J. 2020. Interfaces for Explaining Transformer Language Models.

Ding, Chris, He, Xiaofeng, Simon, Horst, & Jin, Rong. 2005. On the Equivalence of Nonnegative Matrix Factorization and K-means- Spectral Clustering. *Proceedings of the 2005 SIAM International Conference on Data Mining*.

Eckart, Carl & Young, Gale. 1936. The approximation of one matrix by another of lower rank. *Psychometrika 1*, 3, 211–218.

Elhage, Nelson *et al.*. 2022. Toy Models of Superposition. *Transformer Circuits Thread*.

Goh, Gabriel. Decoding The Thought Vector. *gabgoh.github.io*. Available at:

https://gabgoh.github.io/ThoughtVectors/. Accessed on 23 May 2023.

Graziani, Mara, Nguyen, An-phi, O'Mahony, Laura, Müller, Henning, & Andrearczyk, Vincent. 2023. Concept discovery and Dataset exploration with Singular Value Decomposition. In ICLR 2023 Workshop on Pitfalls of limited data and computation for Trustworthy ML.

Hardoon, David Roi, Szedmák, Sándor, & Shawe-Taylor, John. 2004. Canonical Correlation Analysis: An Overview with Application to Learning Methods. *Neural Computation 16*, 2639–2664.

Hilton, Jacob, Cammarata, Nick, Carter, Shan, Goh, Gabriel, & Olah, Chris. 2020. Understanding RL Vision. *Distill*.

Li, Yixuan, Yosinski, Jason, Clune, Jeff, Lipson, Hod, & Hopcroft, John. 2016. Convergent Learning: Do different neural networks learn the same representations? *arXiv preprint arXiv:1511.07543*.

McGrath, Thomas, Kapishnikov, Andrei, Tomasev, Nenad, Pearce, Adam, Hassabis, Demis, Kim, Been, & Paquet, Ulrich. 2021. Acquisition of Chess Knowledge in AlphaZero. *CoRR abs/2111.09259*.

Millidge, Beren. 2023. Deep learning models might be secretly (almost) linear. *lesswrong.com*. Available at: https://www.lesswrong.com/s/yivyHaCAmMJ3CqSyj/p/o6ptPu7arZrqRCxyz. Accessed on 23 May 2023.

Nanda, Neel. 2023. 200 COP in MI: Exploring Polysemanticity and Superposition. *lesswrong.com*. Available at:

https://www.lesswrong.com/s/yivyHaCAmMJ3CqSyj/p/o6ptPu7arZrqRCxyz. Accessed on 23 May 2023.

Olah, Chris, Satyanarayan, Arvind, Johnson, Ian, Carter, Shan, Schubert, Ludwig, Ye, Katherine, & Mordvintsev, Alexander. 2018. The Building Blocks of Interpretability. *Distill*.

Olah, Chris, Cammarata, Nick, Schubert, Ludwig, Goh, Gabriel, Petrov, Michael, & Carter, Shan. 2020. Zoom In: An Introduction to Circuits. *Distill*.

Olah, Chris. 2022. Mechanistic Interpretability, Variables, and the Importance of Interpretable Bases. *Transformer Circuits Thread*. Available at: https://transformer-circuits.pub/2022/mech-interp-essay/index.html. Accessed on 23 May 2023.

Olah, Chris. 2023. Distributed Representations: Composition & Superposition. *Transformer Circuits Thread*. Available at: https://transformer-circuits.pub/2023/superposition-composition/index.html. Accessed on 23 May 2023.

Raghu, Maithra, Gilmer, Justin, Yosinski, Jason, & Sohl-Dickstein, Jascha. 2017. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability.

Sharkey, Lee, Braun, Dan & Millidge, Beren. 2023. [Interim research report] Taking features out of superposition with sparse autoencoders. *alignmentforum.org*. Available at: https://www.alignmentforum.org/posts/z6QQJbtpkEAX3Aojj/interim-research-report-taking-features-out-of-superposition. Accessed on 23 May 2023.

Sharkey, Lee, Braun, Dan & Millidge, Beren. 2023. A small update to the Sparse Coding interim research report. *alignmentforum.org*. Available at: https://www.alignmentforum.org/posts/DezghAd4bdxivEknM/a-small-update-to-the-sparse-coding-interim-research-report. Accessed on 23 May 2023.