Koopman Autoencoders Learn Neural Representation Dynamics

Nishant Suresh Aswani [1 2] **Saif Eddin Jabari** [1 2]

Abstract

This paper explores a simple question: can we model the internal transformations of a neural network using dynamical systems theory? We introduce Koopman autoencoders to capture how neural representations evolve through network layers, treating these representations as states in a dynamical system. Our approach learns a surrogate model that predicts how neural representations transform from input to output, with two key advantages. First, by way of lifting the original states via an autoencoder, it operates in a linear space, making editing the dynamics straightforward. Second, it preserves the topologies of the original representations by regularizing the autoencoding objective. We demonstrate that these surrogate models naturally replicate the progressive topological simplification observed in neural networks. As a practical application, we show how our approach enables targeted class unlearning in the Yin-Yang and MNIST classification tasks.

1 Introduction

Neural networks are defined by compositions. At each step, they transform their inputs, increasing the complexity of the overall transformation applied to data. Remarkably, these transformations have the effect of producing simple shapes at the output (Papyan et al., 2020), when quantified by topology (Naitzat et al., 2020). In fact, the neural representations (i.e., outputs of intermediate layers) of a network progressively simplify until a network arrives at the final output. This progression, along with the compositional nature of these networks, inspires an intuitive ‘path’ perspective (Lange et al., 2023). In other words, there is a notion of ‘traveling’ some distance from the input to the output, along the path defined by these neural representations. Our work further explores this path analogy by asking:

Can we discover a dynamics that generates this path?

And, when equipped with the dynamics, we press on, exploring:

Can we edit these dynamics to produce a different output than what was originally intended?

To elaborate on the significance of our second question: editing, updating, or unlearning specific knowledge contained within neural networks prevents expensive retraining or removes harmful undesired outputs for model alignment (Yao et al., 2023; Gupta et al., 2024).

Contributions. Our main contributions are as follows:

- We introduce Koopman autoencoder surrogates as a framework for interpolating and editing the neural representations of a trained neural network. Our Koopman autoencoders generate realistic dynamics, producing intermediate outputs which follow our established understanding of how neural representations topologically simplify as they progress through the layers of a neural network.
- We develop an encoder isometry objective to supplement the optimization process of Koopman autoencoders, preserving the original topology of neural representations in observable space.
- We demonstrate how our Koopman autoencoders can be used to edit neural representations in observable space, leading to fast, targeted class unlearning.

2 Related Work

Topology and dynamics. Our work is most closely aligned with literature that highlights topological and geometric perspectives in deep learning. Primarily inspired by Naitzat et al. (2020), we demonstrate how the shape of a data manifold can transform as it is processed by the layers of a neural network (NN). As advanced by Lange et al. (2023), we envision the outputs of each NN layer as forming a ‘path’, arising naturally from the compositional structure of NNs. Additionally, we put to work an established dynamics perspective in deep learning. With a spotlight on deep residual networks (ResNets) (He et al., 2016), there is growing evidence (Gai & Zhang, 2021; Li & Papyan, 2023) that treats ResNet activations as traveling on a ‘conveyor belt’ to their final output. This dynamics view plays nicely with the topological vantage, with Naitzat et al. (2020) positing

[1]New York University Tandon, Brooklyn, USA [2]New York University Abu Dhabi, Abu Dhabi, UAE. Correspondence to: Nishant Suresh Aswani <nishantaswani@nyu.edu>.

Published at the ICML 2025 Workshop on Machine Unlearning for Generative AI. Copyright 2025 by the author(s).

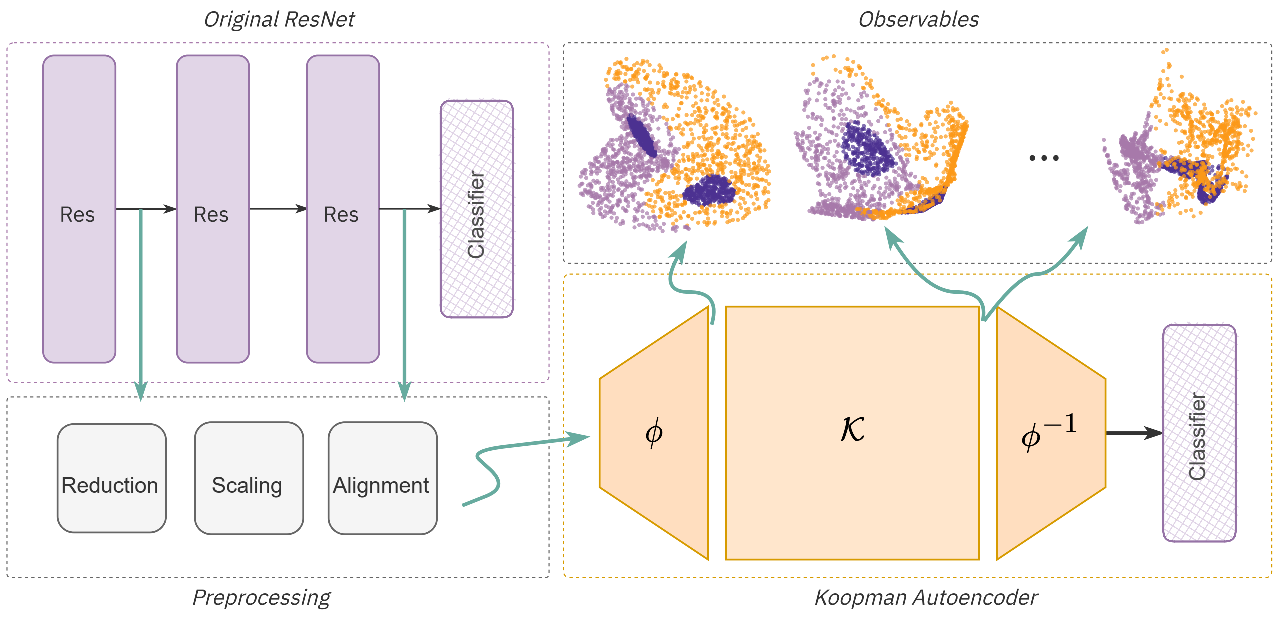

Figure 1. A summary of our framework presented in Section 3. We gather neural representations from a trained, residual network and preprocess them to bring them into the same space. Afterwards, we train a Koopman autoencoder on a pair of the representations, resulting in predictive autoencoder with manipulable and visualizable observabe space.

that "[network] depth plays the role of time," in the sense that additional layers "afford additional time to transform the data."

Koopman-based approaches. At the heart of our method is a Koopman autoencoder (KAE). KAEs have been employed in machine learning problems to forecast physical systems (Takeishi et al., 2017; Lusch et al., 2018; Azencot et al., 2020), disentangle latent factors in sequential datasets (Berman et al., 2023), and generate time-series (Naiman et al., 2024). Traditionally, Koopman approaches find application in control tasks due to their predictive nature. Generally, practical approaches (Budišić et al., 2012; Brunton et al., 2022), developed atop Koopman theory (Koopman, 1931), work within a latent space equipped with linear dynamics allowing one to study, and potentially shape, these dynamics via linear control and spectral tools. Our work is unique in proposing a KAE to interpolate between and manipulate the topology of neural representations.

We provide more background on both topics in Appendix A.

3 Koopman Autoencoders as Surrogates

Consider a trained neural network $\mathcal{N}^L$ composed of $L \in \mathbb{Z}^+$ layers, where each layer f_i is indexed by $i \in \{1, 2, ..., L\}$. The network is defined by successive compositions, giving rise to the form

$$\mathcal{N}^L(x) = f_L \circ \dots f_2 \circ f_1(\mathbf{x}_0), \tag{1}$$

where $\mathbf{x}_0$ is an input. The output of f_i is the i-th neural representation $\mathbf{x}_i \in \mathbb{R}^{d_{i+1}}$, where d_{i+1} is the input dimension of the subsequent layer f_{i+1}. Inspired by Li & Papyan (2023), we work with deep multi-layer perceptrons (MLPs) comprised of residual blocks, a form of residual networks (ResNets). Figure 3 plots the top three principal components of neural representations from each residual block, visualizing how the data transform across the layers of a residual network.

We evoke a dynamical systems perspective of these ResNets, treating the neural representations $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$ of the trained network as the states generated by a complex, nonlinear system. Within this context, we introduce a Koopman autoencoder, consisting of an encoder $\phi : \mathbb{R}^{d_{i+1}} \to \mathbb{R}^p$, a decoder $\phi^{-1} : \mathbb{R}^p \to \mathbb{R}^{d_{i+1}}$, and a linear operator $\mathcal{K} : p \to p$. In concert, they operate as

$$\mathbf{x}_j = \phi^{-1} \circ \mathcal{K} \circ \phi(\mathbf{x}_i), \; \forall i, j \in \{1, 2, \dots, L\} : i < j \tag{2}$$

In Equation 2, ϕ embeds a neural representation into a (typically) higher-dimensional observable, after which $\mathcal{K}$ 'advances' the observable. Finally, ϕ^{-1} returns the observable to the state space. We implement ϕ and ϕ^{-1} as symmetric, but untied, MLPs and define $\mathcal{K}$ as a learnable square matrix. Hence, the KAE produces a dynamic in the observable space, governed by the linear operator. We elaborate on our KAE formulation in Appendix C, including the preprocessing steps prior to training the KAE.

4 Experiments

We work with two residual MLPs, trained on the Yin-Yang (Kriener et al., 2022) and the MNIST classification tasks (Lecun et al., 1998). Each of the MLPs consist of residual blocks (see Appendix B for details). In all our experiments,

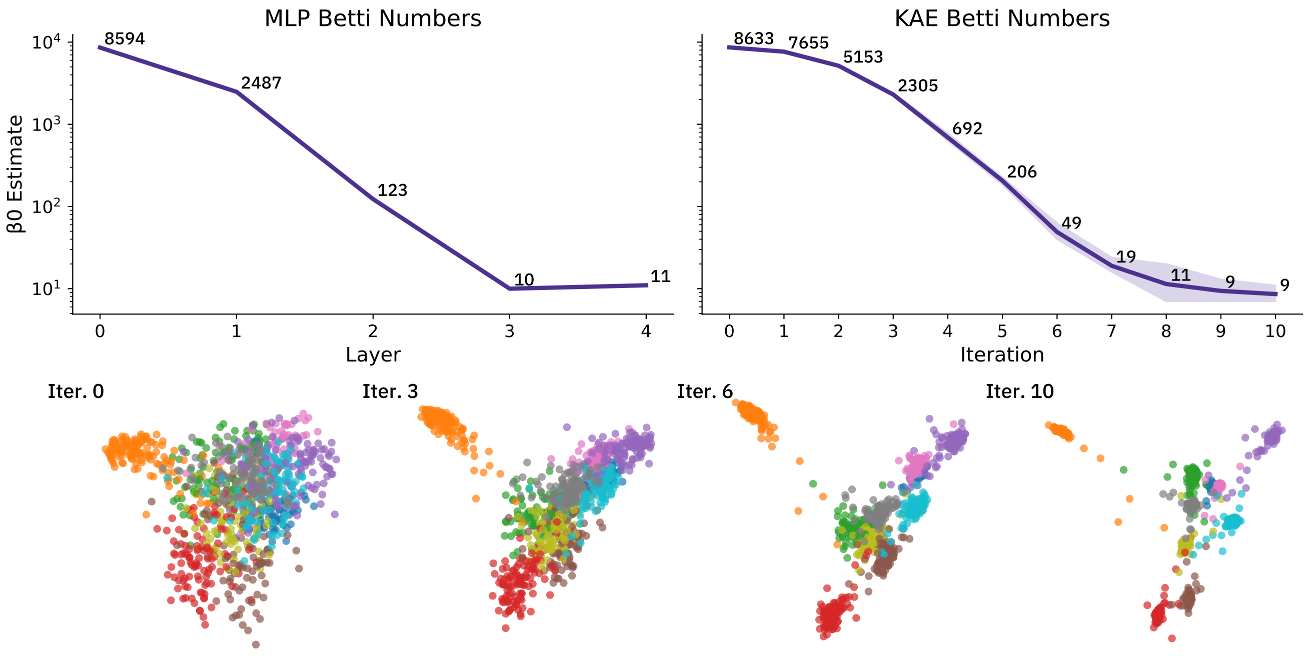

Figure 2. (Top left) The β_0 Betti numbers of the neural representations from each residual block of a residual MLP trained on MNIST. The Betti numbers are computed using the Vietoris-Rips complex at a filtration $\epsilon = 0.166$. (Top right) The average β_0 Betti numbers of intermediate outputs, projected into state space, for five KAEs trained on the first and penultimate layer representations of the residual MLP. The Betti numbers are computed using the Vietoris-Rips complex at a filtration $\epsilon = 0.14$. (Bottom) Select intermediate outputs from an MNIST KAE, projected into the state space. At each successive iteration, the topology is simplified until it arrives at the penultimate layer representations.

we set $\hat{\mathbf{x}}_i$ as the first layer neural representations and $\hat{\mathbf{x}}_j$ as the penultimate layer representations of the residual MLP. Here $\hat{\mathbf{x}}$ refers to the processed neural representation $\hat{\mathbf{x}}$ produced after applying the steps in Appendix C.3. Thus, when given $\hat{\mathbf{x}}_i$ as input, our KAEs are trained to predict $\hat{\mathbf{x}}_j$.

Given the parameterization described in Appendix C.1, our KAEs can predict $k-1$ intermediate representations in observable space, before finally predicting $\hat{\mathbf{x}}_j$. Each of these observable space predictions can be decoded into state space via the KAE decoder for analysis. Ultimately, the output $\hat{\mathbf{x}}_j$ is fed into the final MLP layer, resulting in a class prediction. So, our KAEs can act as surrogate models, handling the intermediate computations. The classification accuracy provides a way to measure the surrogate quality of our KAE. Table 1 demonstrates that our KAEs are able to faithfully produce the penultimate layer representations for both datasets. We provide more details of the KAE architecture and their training in Appendix C.

4.1 Simplifying Topology

Given the parameterization described in Section C.1, our KAEs can interpolate between $\mathbf{x}_i$ and $\mathbf{x}_j$ to produce intermediate representations. Remarkably, we demonstrate that the dynamics within our observable space naturally produce intermediate representations similar to those from the original MLP. To support this claim, we decode the observables into state space and quantify their topology. In Figure 2A, on the left, we present the β_0 Betti numbers of the neural representations from each block of a residual MLP trained to classify MNIST. As established in Naitzat et al. (2020), and evidenced by our plot, successive network layers generate increasingly simple topologies. In comparison, we also plot the β_0 Betti numbers of the decoded, intermediate outputs of five KAEs. Despite having no knowledge of the MLP's intermediate representations and their topologies, our KAEs still naturally simplify in topology at every step. As a visual aid, Figure 2B plots the top three principal components of selected iterations from one of the KAEs.

The dynamics learnt by the KAEs produce a trajectory of neural representations with sound topologies, in line with what is found within a residual MLP. When paired with dimensionality reduction techniques, they provide an approximate visualization of how data is being transformed within a neural network. We hypothesize that the KAE dynamics can be made more faithful to the original residual network by regularizing the KAE's intermediate representations; for example, the KAE could be trained to predict all the neural representations from a residual network.

Table 1. Summary of results demonstrating KAE prediction quality. KAEs were trained with five different seeds.

Dataset	MLP Top-1 % Acc.	KAE Top-1 % Acc. (StDev.)	Target Class	Edited Acc. (StDev.)
Yin-Yang	99.31	98.75 (0.15)	Class 0 (Yin)	98.78 (1.18) $\rightarrow$ 85.01 (1.90)
			Class 1 (Yang)	98.27 (0.21) $\rightarrow$ 78.88 (8.53)
			Class 2 (Dots)	99.97 (0.05) $\rightarrow$ 62.52 (1.35)
MNIST	99.03	98.53 (0.04)	Class 1	99.23 (0.04) $\rightarrow$ 0.0 (0.0)
			Class 4	98.29 (0.08) $\rightarrow$ 0.0 (0.0)
			Class 7	98.01 (0.18) $\rightarrow$ 0.0 (0.0)

4.2 Application: Model Editing

The penultimate layer representations of well-trained classification models experience neural collapse (NC) (Papyan et al., 2020), effectively 'clustering' outputs, as seen at the bottom of Figure 2. In our case, the encoder isometry helps preserve this NC topology in observable space. As a result, identifying a class of 'undesired' outputs in the penultimate layer is a straightforward task. Further, the dynamics that generate the outputs in observable space are governed by a linear operator. Hence, finding the undesired inputs, corresponding to the unwanted outputs, is a matter of applying the inverse operator $\mathcal{K}^{-1}$. To summarize, in observable space, we can quickly identify the unwanted outputs in a neural representation (due to NC) along with their corresponding inputs (by applying the inverse linear operator). Then, with the aid of a model editing algorithm, such as EMMET (Gupta et al., 2024), we can learn an edited linear operator which generates an updated representation—sans the unwanted outputs. If the edited linear operator can maintain the rest of the topology, we can unlearn a specific class without affecting the model's performance on the other classes. We elaborate on our methodology in Appendix E.

Table 1 reports our model editing efforts for two datasets, with starkly different results, highlighting the importance of the neural collapse property. For the Yin-Yang dataset, we use the most strongly regularized KAE (see Figure 5). Despite performing sufficient class separation, the neural representation of the original MLP (and the KAEs), do not exhibit neural collapse; there is a large within-class variance in the penultimate layer. On the other hand, the representations of the MLP (and our KAEs) trained on MNIST exhibit strong neural collapse (see Figure 2). As a result, model editing is successful on the MNIST dataset but performs poorly on the Yin-Yang dataset. In Figure 6, we show the top three principal components of the penultimate representations before and after the linear operator is edited. Here, we edit the operator to remove class 4 (violet) by redirecting it to the class 9 (light blue) cluster, effectively merging the two classes. As a result, the KAE surrogate unlearns class 4. We found that the modified representations do not affect the performance of the KAE decoder and the subsequent MLP classifier on the remaining classes.

5 Limitations and Future Work

Tying together interpretability insights from the perspectives of topology and dynamical systems, our work introduces Koopman autoencoders as surrogate models, which learn the dynamics underlying a deep network's neural representations. By parameterizing the linear operator, we can interpolate an arbitrary number of steps between neural representations. And, our experiments validate that the generated interpolation follows the established principle of progressively simplifying topology. Additionally, we demonstrate how linear dynamics in observable space can enable editing the neural representations, leading to class unlearning. For future work, several directions emerge:

- **Representation regularization:** Currently, our approach is limited to interpolating between two neural representations. How do we regularize the dynamics to interpolate through all the intermediate representations of a model?
- **Operator interpretability:** Given that a Koopman operator governs our dynamics, does spectral analysis of the operator offer insights into the original model's mechanism?
- **Observable space shaping:** Since we have the freedom to shape how neural representations look in observable space, are there other favorable topologies that enable certain goals (e.g., disentanglement, interpretability, unlearning)?
- **Architecture extensions:** Extending our approach to models with different architectures (e.g., convolutional layers, transformer blocks, etc.) could enable more sophisticated model editing applications beyond classification tasks. Can we extend our framework to unlearn concepts in language models?

In conclusion, our work demonstrates how Koopman theory can provide a practical framework for working with neural representations, opening new avenues for analyzing deep networks through the lens of dynamical systems.

References

Azencot, O., Erichson, N. B., Lin, V., and Mahoney, M. Forecasting sequential data using consistent koopman autoencoders. In *International Conference on Machine Learning*, pp. 475–485. PMLR, 2020.

Berman, N., Naiman, I., and Azencot, O. Multifactor sequential disentanglement via structured koopman autoencoders. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

Brunton, S. L., Budišić, M., Kaiser, E., and Kutz, J. N. Modern koopman theory for dynamical systems. *SIAM Review*, 64(2):229–340, 2022.

Budišić, M., Mohr, R., and Mezić, I. Applied koopmanism. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(4), 2012.

Gai, K. and Zhang, S. A mathematical principle of deep learning: Learn the geodesic curve in the wasserstein space. *arXiv preprint arXiv:2102.09235*, 2021.

Gupta, A., Sajnani, D., and Anumanchipalli, G. A unified framework for model editing. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15403–15418, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Koopman, B. O. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

Kriener, L., Göltz, J., and Petrovici, M. A. The yin-yang dataset. In *Neuro-Inspired Computational Elements Conference*, NICE 2022, pp. 107–111. ACM, 2022.

Lange, R. D., Kwok, D., Matelsky, J. K., Wang, X., Rolnick, D., and Kording, K. Deep networks as paths on the manifold of neural representations. In Doster, T., Emerson, T., Kvinge, H., Miolane, N., Papillon, M., Rieck, B., and Sanborn, S. (eds.), *Proceedings of 2nd Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML)*, volume 221 of *Proceedings of Machine Learning Research*, pp. 102–133. PMLR, 28 Jul 2023.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, J. and Papyan, V. Residual alignment: uncovering the mechanisms of residual networks. *Advances in Neural Information Processing Systems*, 36:57660–57712, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Lusch, B., Kutz, J. N., and Brunton, S. L. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

Naiman, I., Erichson, N. B., Ren, P., Mahoney, M. W., and Azencot, O. Generative modeling of regular and irregular time series data via koopman vaes. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Naitzat, G., Zhitnikov, A., and Lim, L.-H. Topology of deep neural networks. *Journal of Machine Learning Research*, 21(184):1–40, 2020.

Papyan, V., Han, X. Y., and Donoho, D. L. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

Takeishi, N., Kawahara, Y., and Yairi, T. Learning koopman invariant subspaces for dynamic mode decomposition. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Yao, Y., Wang, P., Tian, B., Cheng, S., Li, Z., Deng, S., Chen, H., and Zhang, N. Editing large language models: Problems, methods, and opportunities. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore, 2023. Association for Computational Linguistics.

A Preliminaries

A.1 Topology

Our concrete measure of an object's topology refers to its *Betti numbers*. For a k-dimensional manifold, one can compute k Betti numbers, defining its topological signature. The zero-th Betti number, β_0, of a manifold refers to the number of unconnected components. The k-th Betti number, for $k \geq 1$, quantifies the number of k-dimensional holes in the manifold. This manifests in the popular, though counterintuitive, quip that 'a donut is topologically equivalent to a coffee mug.' Both objects have one connected component, a single 1-D hole, and zero 2-D holes, giving them the Betti number sequence $\beta = \{1, 1, 0\}$.

When working with discrete manifolds, such as neural representations from a network, quantifying topology relies on *persistence homology*. Very simply, the approach computes k-dimensional *simplices* (e.g., points, lines, triangles, tetrahedra, etc.) of an object at varying scales, which determine an object's *homologies*. These homologies are closely related to the Betti numbers; by tracking these homology groups across scales, one can make claims about an object's topology. We rely on the *Vietoris-Rips (VR) complex*, a particular method of computing the simplices, which in turn informs the Betti numbers. The VR complex requires a distance metric (in our case Euclidean) and a scale parameter ϵ. For a more detailed background on algebraic topology we refer to Naitzat et al. (2020).

A.2 Koopman theory

In a typical discrete dynamical system, we observe measurements of a state $\mathbf{x}_t \in \mathcal{M} \subseteq \mathbb{R}^N$ at time $t \in \mathbb{Z}^+$, which evolve under a mapping $\mathcal{T} : \mathcal{M} \to \mathcal{M}$, such that

$$\mathbf{x}_{k+1} = \mathcal{T}(\mathbf{x}_k). \tag{3}$$

When $\mathcal{T}$ is nonlinear, these systems are often analyzed using linear approximations near fixed points, often to control the underlying nonlinear system.

Koopman operator theory suggests an alternative global linearization of the dynamics by finding a map in the *observable space*, $\phi(x_k) : \mathcal{M} \to \mathcal{F} \subseteq \mathbb{C}$. In this space, the linear map $\mathcal{K} : \mathcal{F} \to \mathcal{F}$, which evolves the observables, is defined as the *Koopman operator.* If we assume our observables as vectors, we obtain the form

$$\phi(\mathbf{x}_{k+1}) = \mathcal{K} \circ \phi(\mathbf{x}_k), \tag{4}$$

where ϕ "lifts" our original system states into the observable space resulting in a system that evolves under a linear operator. The forecast can be obtained in the state space by applying an inverse operation $\phi^{-1} : \mathcal{F} \to \mathcal{M}$ to the result of the forward dynamic. Brunton et al. (2022) provide a fuller view of modern applications of Koopman theory, along with its rich history in machine learning.

B Dataset and model details

B.1 Yin-Yang task

The Yin-Yang dataset (Kriener et al., 2022) is a task with two-dimensional inputs consisting of three classes, allowing for easy visualization of the model's decision boundary and topology. For our experiments, we use a residual MLP architecture

$$\text{Residual MLP} : \mathbb{R}^2 \to \text{Linear}(2 \to 10, \text{ReLU}) \to 4 \times [\text{ResBlock}(10, \text{ReLU})] \to \text{Linear}(10 \to 2)$$

We generate a training dataset of 5×10^3 samples, with roughly equal distribution among the three classes. For the test dataset, we generate another set of 5×10^3 samples with a different seed. The network is trained to a test accuracy of 99.31% using SGD with momentum (set to 0.9) for 500 epochs. We use a batch size of 512 samples, a weight decay set to 5×10^{-4}, and a cyclic learning rate peaking at 10^{-1}. Figure 3 shows the neural activations for each output layer from the Yin-Yang dataset.

B.2 MNIST task

For the MNIST task (Lecun et al., 1998), we train a residual MLP with four blocks

$$\text{Residual MLP} : \mathbb{R}^2 \to \text{Linear}(2 \to 784, \text{ReLU}) \to 4 \times [\text{ResBlock}(784, \text{ReLU})] \to \text{Linear}(784 \to 2)$$

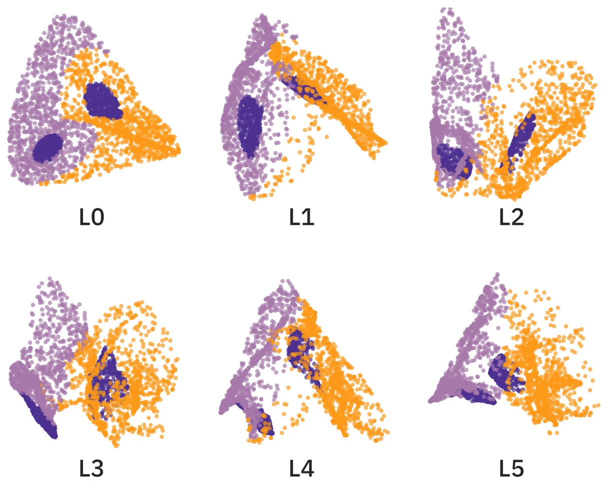

Figure 3. The top three principal components of the neural representations from the first layer (L0) and all residual blocks (L1-5) of a multi-layer perceptron (MLP) with a ResNet-style architecture. Each plot contains 2×10^3 points and undergoes the preprocessing steps outlined in Section C.3 before PCA for plotting. The model is trained on the Yin-Yang dataset (Kriener et al., 2022), a three-way classification task. See Appendix B for details on architecture and dataset.

The model is trained to a test accuracy of 99.03% using SGD with momentum (set to 0.9) for 30 epochs on a batch size of 128 samples, a weight decay set to 5×10^{-4}, and a cyclic learning rate peaking at 10^{-1}.

Similar to Figure 3, we show the neural activations from each output layer of the MNIST model in Figure 4.

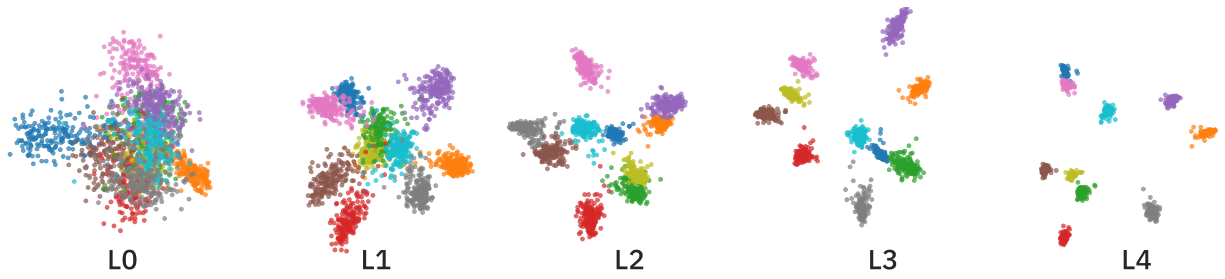

Figure 4. The top three principal components of the neural representations from the first layer (L0) and all residual blocks (L1-4) of a residual multi-layer perceptron (MLP). Each plot consists of 2×10^3 points and undergoes the preprocessing steps outlined in Section C.3 before PCA. The model is trained on the MNIST digits task.

C Koopman autoencoder details

C.1 Architecture

Table 2 outlines the architecture of the Koopman autoencoders used in both tasks.

Table 2. KAE architecture

Component	Yin-Yang	MNIST
Encoder	batch $\times\mathbb{R}^{10}$ Linear($10 \to 30$) $\to$ LeakyReLU Linear($30 \to 20$)	batch $\times\mathbb{R}^{784}$ Linear($784 \to 1000$) $\to$ LeakyReLU Linear($1000 \to 800$)
Koopman Matrix	batch $\times\mathbb{R}^{20}$ Linear($20 \to 20$)	batch $\times\mathbb{R}^{800}$ Linear($800 \to 800$)
Decoder	batch $\times\mathbb{R}^{20}$ Linear($20 \to 30$) $\to$ LeakyReLU Linear($30 \to 10$)	batch $\times\mathbb{R}^{800}$ Linear($800 \to 1000$) $\to$ LeakyReLU Linear($1000 \to 784$)

We parameterize the Koopman operator as

$$\mathcal{K} = \exp\left(\mathcal{G}/k\right)^{k}, \tag{5}$$

where $\mathcal{G}$ is another linear operator of the same shape and k determines the number of steps that $\hat{\mathbf{x}}_i$ is advanced in observable space. When coupled with dimensionality reduction, this parameterization allows for a smooth k-step transformation of the neural activations, enabling an explicit visualization of topological changes. The parameterization is not restrictive: we can obtain the final prediction by directly applying the k-powered matrix.

C.2 Objectives

The KAE is optimized with the objective functions

$$\mathcal{L}_{\text{recon}} = \left\|\mathbf{x}_{\{i,j\}} - \phi^{-1} \circ \phi(\mathbf{x}_{\{i,j\}})\right\|^2, \tag{6}$$

$$\mathcal{L}_{\text{linear}} = \left\|\phi(\mathbf{x}_j) - \mathcal{K} \circ \phi(\mathbf{x}_i)\right\|^2, \tag{7}$$

$$\mathcal{L}_{\text{state}} = \left\|\mathbf{x}_j - \phi^{-1} \circ \mathcal{K} \circ \phi(\mathbf{x}_i)\right\|^2, \tag{8}$$

$$\mathcal{L}_{\text{dist}} = \left\|\left\|\mathbf{x}_{\{i,j\}}\right\|^2 - \left\|\phi(\mathbf{x}_{\{i,j\}})\right\|^2\right\|^2, \tag{9}$$

resulting in a combined loss

$$\mathcal{L}_{\text{total}} = \lambda_1\mathcal{L}_{\text{recon}} + \lambda_2\mathcal{L}_{\text{linear}} + \lambda_3\mathcal{L}_{\text{state}} + \lambda_4\mathcal{L}_{\text{dist}}. \tag{10}$$

The $\{\lambda_i\}_{i=1}^4$ act as weighting hyperparameters. We use the AdamW optimizer (Loshchilov & Hutter, 2019) to train our KAEs. Table 3 presents the hyperparameter choices.

Equation 6 encourages the KAE to reconstruct states in the absence of any dynamics, promoting autoencoding. The linear prediction loss (Eq. 7) ensures that the observables evolve linearly in the latent space, while the state prediction loss (Eq. 8) aids end-to-end prediction accuracy when mapping back to the state space. Finally, the encoder isometry (Eq. 9) encourages preservation of inter-point distances even in the observable space. We discuss the significance of encoder isometry in Section D.

Table 3. KAE hyperparameter details

Dataset	batch	observable dim.	#epochs	λ_{recon}	λ_{linear}	λ_{state}	λ_{dist}	learning rate	weight decay
Yin-Yang	1024	20	1000	1	1	1	1	1×10^{-1}	5×10^{-4}
MNIST	512	800	100	1	1	1	10^{-3}	5×10^{-3}	5×10^{-4}

C.3 Preprocessing Representations

Given we are working with neural representations, we draw from tools in RSA metrics literature. Permitting intra-layer comparison, these metrics first require embedding neural representations in a common space $\mathbb{R}^q$. Only then is a distance metric defined. Lange et al. (2023) detail the intricacies and variations in this class of approaches.

Our work is concerned solely with the initial embedding step. To avoid confusion with 'embedding' in the context of Koopman approaches, we refer to this as *preprocessing*. To elaborate, we apply the following preprocessing to $\mathbf{x_i}, \mathbf{x_j}$, before they are fed into a KAE:

1. **Mean-centering**: $\hat{\mathbf{x}} = \mathbf{x} - \mathbb{E}[\mathbf{x}]$ (11)
2. **Projection**: $\hat{\mathbf{x}} = \hat{\mathbf{x}} U_{:q}$, given $U \Sigma V^\top = \text{svd}(\hat{\mathbf{x}})$ (12)
3. **Normalizing**: $\hat{\mathbf{x}} = \hat{\mathbf{x}} / \|\hat{\mathbf{x}}\|$ (13)
4. **Procrustes alignment**: $\hat{\mathbf{x}} = \hat{\mathbf{x}} R$, where $R \in \mathcal{O}(q)$ solves $\min_R \|\hat{\mathbf{x}} - \hat{\mathbf{y}} R\|_F$ (14)

Overall, we shift, project, and scale the representations before finding the best (rotational) alignment, making the representations more suited for comparison. In addition to affording us invariance properties, the preprocessing allows for learning a KAE on neural representations with originally non-uniform dimensions; i.e., outputs of differently-sized NN layers. However, we do not include models with non-uniform dimensions in our experiments.

D Encoder Isometry

Typical implementations of KAEs (Takeishi et al., 2017; Lusch et al., 2018; Azencot et al., 2020; Berman et al., 2023) do not consider encoder isometry. However, neural representations are topological objects; our isometry objective (Eq. 9) promotes the observables to carry over the original shape of the representation.

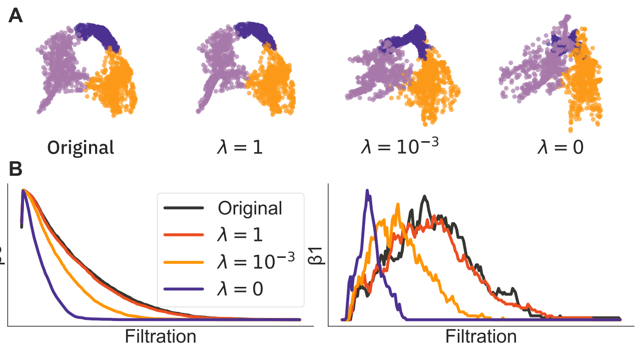

Figure 5. (A) Each scatter plot displays 2×10^3 points projected onto the top three principal components (PCs) derived from representations in the penultimate layer. The leftmost plot shows PCs from the original MLP representations, while the remaining show PCs computed after embedding the representations into observable space via different KAEs. All PCs are aligned via the orthogonal Procrustes problem. (B) Betti curves, for $\beta 0$ and $\beta 1$, across a filtration threshold of $\epsilon = 4$ for the penultimate layer representations of the original model (black) and the observable space representations via different KAEs.

To demonstrate, we train 3 KAE variants with different penalization strengths ($\lambda_4 = \{0, 10^{-3}, 1\}$) on the encoder isometry objective. The KAEs are trained to predict (and reconstruct) the penultimate layer representations of a residual MLP. Figure 5A displays the top three principal components of the penultimate layer representations in observable space. Figure 5B presents the *Betti curves* of these same models, demonstrating that the most strongly penalized encoder (red) exhibits the closest topological similarity to the original model (black). These results indicate that increasing λ_4 leads to more topologically faithful representations in observable space. As a result, we expect that topological edits in the observable space will also be reflected in the state space.

E Model Editing

We outline the steps of our model editing approach in Algorithm 1.

Algorithm 1 Model Editing with KAEs

```
**Input:** trained KAE $\{\phi, \mathcal{K}, \phi^{-1}\}$, reprs. $\{\mathbf{x}_i, \mathbf{x}_j\}$, target class $c$
**Output:** Updated output reprs. $\hat{\mathbf{x}}_j$

// Identify unwanted outputs
$Z_{del} \leftarrow \{\phi(\mathbf{x}_j) \mid \mathbf{x}_j$ belongs to class $c\}$
$Z_{keep} \leftarrow \{\phi(\mathbf{x}_j) \mid \mathbf{x}_j$ not in class $c\}$

// Compute corresponding inputs
$X_{mem} \leftarrow \mathcal{K}^{-1} \circ Z_{del}$
$X_{keep} \leftarrow \{\mathbf{x}_i \mid \mathbf{x}_i$ not in $X_{mem}\}$

// Select alternative outputs
$Z_{new} \leftarrow$ alt_output$(X_c)$

// Edit operator
$\mathcal{L} \leftarrow$ EMMET$(\mathcal{K}, \{X_{mem}, Z_{new}\}, \{X_{keep}, Z_{keep}\})$

// Update reprs.
$\hat{\mathbf{x}}_j \leftarrow \mathcal{L} \circ \mathbf{x}_i$
```

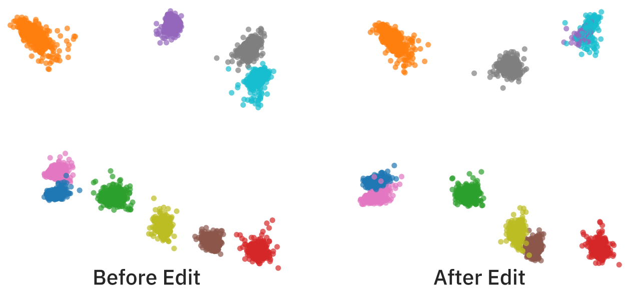

Figure 6. 10^4 points projected on the top three principal components of the neural representations produced by the Koopman operator in observable space before editing (left) and after editing (right). The KAE is trained on the first and penultimate-layer representations of a MNIST classifier. The operator is edited to forget class 4 (violet) by merging the outputs of that class with those of class 9 (light blue). The result of the merge is visible on the top right corner.