

Foundations of Machine Learning: Over-parameterization and Feature Learning

by

Adityanarayanan Radhakrishnan

S.B., Mathematics, Massachusetts Institute of Technology (2016)

S.B., Electrical Engineering and Computer Science, Massachusetts Institute of
Technology (2016)

M.Eng., Electrical Engineering and Computer Science, Massachusetts
Institute of Technology (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

© 2023 Adityanarayanan Radhakrishnan. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Adityanarayanan Radhakrishnan
Department of Electrical Engineering and Computer Science
May 19, 2023

Certified by: Caroline Uhler
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Foundations of Machine Learning: Over-parameterization and Feature Learning

by

Adityanarayanan Radhakrishnan

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2023, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science and Engineering

Abstract

In this thesis, we establish and analyze two core principles driving the success of neural networks: over-parameterization and feature learning. We leverage these principles to design models with improved performance and interpretability on various computer vision and biomedical applications.

We begin by discussing the benefits of over-parameterization, i.e., using increasingly large networks that can perfectly fit training data. While prior work characterized the benefits of over-parameterized networks for supervised learning tasks, we show that over-parameterization is also beneficial for unsupervised learning problems, such as autoencoding. The ubiquitous advantage of using increasingly larger networks suggests that infinitely large networks should yield best performance. Remarkably, under certain conditions, training infinitely wide networks simplifies to training classical models known as kernel machines using the Neural Tangent Kernel (NTK). We showcase the practical value of the NTK by deriving and using it for matrix completion problems such as image inpainting and virtual drug screening. Additionally, we use the NTK connection to provide theoretical guarantees for deep neural networks. Namely, we construct interpolating infinitely wide and deep networks that are Bayes optimal, or consistent, for classification.

While the NTK has been a useful tool for understanding properties of deep networks, it lacks a key component that is critical to the success of neural networks: feature learning. In the second part of this thesis, we identify and mathematically characterize the mechanism through which deep neural networks automatically select features, or patterns in data. We show that neural feature learning occurs by re-weighting features based on how much they change predictions upon perturbation, a process that is mathematically characterized by the average gradient outer product. Our result explains prominent deep learning phenomena such as spurious features, lottery tickets, and grokking. Moreover, the mechanism identified in our work provides a backpropagation-free method for feature learning with any machine learning model. To demonstrate the effectiveness of this general feature learning mechanism, we use it to enable feature learning in kernel machines. We show that the resulting models, referred to as Recursive Feature Machines, achieve state-of-the-art performance on tabular data.

Overall, this thesis advances the foundations of machine learning and provides tools for building new machine learning models that are computationally simple, interpretable, and effective.

Thesis Supervisor: Caroline Uhler
Title: Full Professor

Acknowledgments

First, I would like to thank my advisor Caroline Uhler who has supported and guided me throughout my time as an undergraduate, M.Eng., and Ph.D. researcher. In addition to her invaluable technical guidance, Caroline has been incredibly foundational in building my research career. Caroline saw that I enjoyed research early on and was the first to encourage me to apply for Ph.D. positions. Without her, I would never have had the opportunity to pursue these exciting research directions. She has also given me freedom to pursue my interests and helped me build my own research agenda over the years. Caroline connected me with several of my current collaborators and has helped grow my research network. Most importantly, Caroline has been a compassionate mentor and has made sure I maintained a healthy work-life balance as a researcher.

Additionally, I would like to thank Mikhail (Misha) Belkin, who has been highly influential on my machine learning theory research. Our thought-provoking discussions have been key to formulating my research on over-parameterization and feature learning. Moreover, he has helped shape my philosophy on providing simple explanations for complex phenomena. I also want to thank Misha for expanding my research connections through collaboration at various summer research programs. Lastly, I want to thank him for making me feel welcome in his group at UCSD. Discussing and collaborating with him and his students been both fun and fruitful.

Next, I would like to thank G.V. Shivashankar (Shiva) who has been an invaluable collaborator on biomedical applications of machine learning through my Masters and Ph.D. research. Collaborating with him and his lab on such applications has helped open my eyes to the entirety of the machine learning pipeline from data generation to predictive modeling. Working with Shiva has been especially helpful in developing my skills in both communicating machine learning concepts to a biological audience and mathematically formulating biological problems. I would also like to thank him and his students for hosting me during research visits to Singapore.

I would also like to thank Anthony Philippakis who has been a mentor and collaborator throughout my affiliation with the Broad Institute. Anthony has encouraged my mathematical pursuits in geometry and analysis, and I have enjoyed our research discussions over the years. He and the machine learning for healthcare (ML4H) team at the Broad have been welcoming and helpful in expanding my knowledge about genetics and applications of machine learning in healthcare.

I am also grateful to the many undergraduate, Masters, and visiting students at MIT who have worked with me over the years and have helped shape my own leadership skills. In particular, I would like to thank Max Ruiz Luyten, Cathy Cai, George Stefanakis, Neha Prasad, and Eshaan Nichani for interesting research discussions that both led to new research and motivated me to develop a course on material in this thesis. I would also like to thank fellow Ph.D. and postdoctoral students Anastasiya Belyaeva, Louis Cammarata, Karren Yang, Chandler Squires, Daniel Beaglehole, Parthe Pandit, Libin Zhu, and Chaoyue Liu, for many years of insightful discussion and collaboration.

Importantly, none of this work would be possible without financial support from the Eric and Wendy Schmidt Center. I am also particularly grateful to the center for providing me the opportunity to connect with various biological collaborators through workshops and speaker series. The center has been extremely useful in providing a platform for applying several of the techniques developed in this thesis.

Finally, I want to thank my family and friends. I would like to thank my parents, Radhakrishnan Swaminathan and Ananthalakshmi Subramanian, who have been a reliable source of mentorship, support, and encouragement throughout my life. I would also like to thank my grandmother, G.K.P., who, while no longer with us, believed far in advance that I would pursue and complete a Ph.D. None of this work would be possible without the incredible support of my soon-to-be wife, Allison Edwards who has stuck with and helped me through various difficulties over the years. I would like to also thank my sister, Mathangi Radhakrishnan, and her husband, Charles (Charlie) Durham for their support and encouragement in my research as well. In fact, Charlie built and taught me how to build the servers, which enabled all of my applied research. Lastly, I want to thank my friend, Jarrod Smith, for the many fun, shared experiences we have had and for being there for me through the years.

Contents

1	Introduction	19
1.1	Over-parameterization	19
1.2	Feature Learning	20
1.3	Preliminaries	20
1.3.1	Neural Networks	20
1.3.2	Kernel Machines	21
1.3.3	Neural Tangent Kernel	21
2	Over-parameterized Autoencoders and Sequence Encoders	25
2.1	Introduction	25
2.2	Related Work	27
2.3	Empirical Findings	28
2.4	Theoretical Analysis of Special Cases	32
2.5	Discussion	35
3	Infinitely wide neural networks for matrix completion	37
3.1	Introduction	37
3.2	Matrix Completion with the NTK	39
3.3	Virtual Drug Screening with the NTK	41
3.4	Matrix Completion with the Convolutional NTK	42
3.5	Image Inpainting and Reconstruction with the CNTK	45
3.6	Discussion	47
4	Consistency of infinitely wide and deep neural network classifiers	49
4.1	Introduction	49
4.2	Taxonomy of Infinitely Wide and Deep Neural Networks	51
4.3	Outline of Proof Strategy for Theorems 7 and 8	55
4.4	Discussion	57
5	Mechanism of feature learning in deep fully connected neural networks	59
5.1	Introduction	59
5.2	Results	61
5.2.1	Deep Neural Feature Ansatz sheds light on notable phenomena from deep learning	62
5.2.2	Integrating feature learning into machine learning models	64
5.2.3	Recursive Feature Machines provide state-of-the-art results on tabular data	66
5.2.4	Theoretical Evidence for Deep Neural Feature Ansatz	67
5.3	Discussion	69
6	Summary and Discussion	73

A Chapter 2 Supplementary	75
A Encoding Multiple Sequences	75
B Proofs of Invariants, Theorem 1, and Theorem 2	75
C Analysis of Deep Autoencoders with 1 Training Example	78
D Trained Autoencoders Produce Outputs in the Span of the Training Data	79
E Proofs of Theorem 3 and 4	80
F Limit Cycles in Recurrent Neural Networks	81
G Sequence Encoding Audio	81
B Chapter 3 Supplementary	89
A Proofs for Matrix Completion with the NTK	89
B Experimental Details for Virtual Drug Screening in CMAP	93
C Feature Prior for Drug Response Imputation	94
D One-hot Encoding for Drugs is Equivalent to Imputation with Mean Over Cell Type	94
E Feature Prior Corresponding to Previous Algorithms	95
F Performance of Methods on Sparse versus Dense Subsets	95
G Metrics for Evaluation in Drug Response Imputation	95
H Statistical Significance of NTK on Drug Response Imputation	96
I Matrix Completion with the CNTK	96
J Equivalence with Semi-Supervised Learning for the CNTK	98
K Derivation of the CNTK for Matrix Completion with Modern Architectures	100
L Efficient Computation of the CNTK for High Resolution Images	102
M Experimental Details for Image Inpainting	104
C Chapter 4 Supplementary	111
A Preliminaries on NTK and Dual Activations	111
B Proofs of Theorem 1 and Theorem 2	112
C Extension of Hilbert estimate consistency	123
D Proof of Corollary 1	125
E Proof of Theorem 3	126
F Proof of Proposition 1	127
G Proofs for when Infinitely Wide and Deep Networks are Majority Vote Classifiers	129
H Infinitely Wide and Deep Networks with Standard Activation Functions	130
I Experiments	131
J Effectiveness of finite-depth NTKs using our derived activation functions on a variety of benchmarking classification tasks	134
D Chapter 5 Supplementary	137
A Theoretical evidence for Deep Neural Feature Ansatz	137
B Methods	144
C Background on Kernel Ridge Regression	147
D Kernel alignment and gradient outer product	147

List of Figures

2-1	The difference between associative memory and interpolation is described in (a, b); the mechanism identified in this work by which overparameterized autoencoders implement associative memory is described in (c, d). Training examples are shown as black points, the identity function is shown as a dotted line and functions are represented using solid, colored lines. (a) An example of a function that interpolates the training data but does not memorize training data: training data are not recoverable from just the function alone. (b) An example of a function that interpolates and memorizes training data: training data are recoverable as the range of the function. (c) An example of an interpolating function for which the training examples are attractors; the basin of attraction are shown. (d) Iteration of the function from (c) leads to a function that is piece-wise constant almost everywhere, with the training examples corresponding to the non-trivial constant regions. The fact that the training examples are attractors implies that iteration provides a retrieval mechanism.	26
2-2	Example of an over-parameterized autoencoder storing 500 images from ImageNet-64 as attractors after training to a reconstruction error of less than 10^{-8} . Architecture and optimizer details are provided in Appendix, Fig. A-1. (a) By iterating the trained autoencoder on corrupted versions of training samples, individual training samples are recovered. (b) Samples that are corrupted by uniform random noise or squares of varying color and size are recovered via iteration. (c) Fraction of samples recovered correctly from different noise applied to the training images. A sample is considered recovered when the error between the original sample and the recovered sample is less than 10^{-7}	29
2-3	Example of an over-parameterized autoencoder in the 2-dimensional setting storing training examples (represented as stars) as attractors. Basins of attraction for each sample are colored by sampling 10,000 points in a grid around the training examples, taking the limit of the iteration for each point, and assigning a color to the point based on the training example indicated by the limit. The vector field indicates the direction of motion given by iteration, and the inserts indicate that iteration leads to training examples for all points in an open set around each example.	30
2-4	Examples of over-parameterized sequence encoders storing training sequences as limit cycles. Architecture and optimizer details are provided in Appendix, Fig. A-1. (a) When trained on 389 frames of size 128×128 from the Disney film “Steamboat Willie”, the entire movie was stored as a limit cycle. Hence, iteration from random noise leads to recovery of the entire sequence. (b) When trained on two sequences of length 10 from MNIST, each sequence was stored as a limit cycle. Hence iteration from random noise leads to the recovery of each individual sequence. (c) Visualization of the basins of attraction for a sequence encoder storing 4 sequences as limit cycles in the 2-dimensional setting. The vector field indicates the direction of motion given by iteration.	31
2-5	Sequence encoders are more efficient at implementing associative memory than autoencoders. Number of training examples recovered are out of 100; architecture and optimizer details are provided in the Appendix, Fig. A-1. (a) Number of recovered images when autoencoding 100 examples from MNIST individually; a network of depth 31 and width 512 recovers 99 images out of 100. (b)-(e) Sequence encoding the same 100 MNIST examples as sequences of different lengths improves the recovery rates; in particular, a network of depth 1 and width 512 recovers the full 10 images when encoded as 5 sequences of length 20 (d) or 1 sequence of length 100 (e).	32

3-1	An overview of matrix completion applications where ?'s in (a), (b) and zero (black) pixels in (c), (d) represent unobserved entries. (a) Collaborative filtering example (the Netflix problem), where the goal is to predict how a user would rate (on a scale of 1-5) an unseen movie. (b) Virtual drug screening, where the problem is to predict the gene expression profile for an unobserved drug / cell type combination. In this application entire columns are unobserved. (c,d) Image inpainting and reconstruction involves reconstructing a corrupted region of an image (shown as black pixels). (e) Our NTK matrix completion framework is easily adapted to solve all of the above problems by selecting a feature prior that represents an embedding of application specific metadata.	38
3-2	Our infinite width neural network framework outperforms DNPP [84], FaLRTC [121], and mean over cell types for drug response imputation on CMap. (a) We visualize the availability of cell type and drug combinations of the subset from [84]. (b) Our method corresponds to first providing an embedding of cell type and drug combinations as the feature prior and then applying the NTK. We show that: (1) using a feature prior consisting of one-hot vectors for drugs corresponds to imputation by performing mean across observations for each cell type and (2) using a feature prior that captures similarity between drugs and cell types is effective for imputation. (c, d) Our infinite width neural network framework (denoted NTK) outperforms DNPP and mean over cell type across three evaluation metrics. We use 5 rounds of 10-fold cross validation to determine that the difference between our method and the next best method, DNPP, is statistically significant (p-value less than 10^{-20}).	41
3-3	Large hole inpainting using (i) the CNTK, (ii) neural networks with sigmoid last layer and batch normalization layers that are trained with Adam, and (iii) biharmonic functions. (a) Qualitative comparison of inpainting results across the three methods. Results for all images are provided in SI Appendix Fig. S5. (b) Comparison of peak signal-to-noise ratio (PSNR) across 3 methods with the CNTK providing the highest average PSNR. Runtime and structural similarity index measure (SSIM) for the three methods are provided in SI Appendix Fig. S4.	44
3-4	We use the CNTK to understand the impact of architecture and input on image inpainting. (a) Heatmap visualizations of the CNTK when varying the number of downsampling/upsampling layers and input. The visualization makes clear that the uniform random feature prior, unlike other feature priors, results in kernels that use the region surrounding a missing pixel value for imputation regardless of the number of downsampling layers. (b) The heatmap visualizations of the CNTK make transparent which observed pixels are being used to inpaint a given missing pixel when using the identity feature prior. (c) A comparison between inpainting a 128×128 resolution image of a rabbit with a finite width neural network and with the CNTK when the feature prior is the identity. The CNTK is able to accurately predict the unexpected behavior of the neural network.	46
4-1	Behavior of infinitely wide and deep neural networks trained with gradient descent. (a) Taxonomy of infinitely wide and deep networks. Depending on the choice of the activation function, $\phi(\cdot)$, these models implement majority vote (blue), 1-nearest neighbor (red), or singular kernel classifiers (green), a subset of which achieve consistency. (b) Regression versus classification using infinitely wide and deep networks. While these models are not effective in the regression setting, since their predictions are near zero almost everywhere, they can achieve consistency for classification, where only the sign of the prediction matters. (c) Illustration of the different behaviors of infinitely wide and deep networks for varying activation functions. Depending on the activation function, infinitely wide and deep networks implement majority vote (blue), 1-nearest neighbor (red), or singular kernel classifiers that can achieve consistency (green). Singular kernels that grow too slowly are akin to majority vote classifiers (dashed blue), whereas those that grow too quickly are akin to weighted nearest neighbor classifiers (dashed red).	50

- 4-2 Iteration of a piecewise linear function on a unit interval leads to a function with a singularity at $x = 1$, upon appropriate normalization. (a) We consider the piecewise linear function $f(x)$ given by $1 - b(1 - x)$ on $(c, 1]$ and ax on $[0, c]$, where $a = .5$, $b = 1.5$ and $c = \frac{b-1}{b-a}$. (b) We observe that upon iterating $f(\cdot)$ numerically to the limit of machine precision, the resulting function strongly agrees with the theoretical limit of Lemma 1 given by a function with singularity of order $-\log_b a \approx 1.7$ 56
- 5-1 **(A)** A demonstration of neural feature learning. We train two fully connected neural networks with two-hidden-layers and ReLU activation to classify glasses in image data (96×96 images) from the CelebA dataset [122], one in which the first layer is not trained and the other in which all layers are trained. We visualize the diagonal of the first layer neural feature matrix (NFM), $W_1^T W_1$, of the trained networks and observe that the network with better performance learns to select pixels corresponding to glasses. **(B)** We show the NFM of a layer is well approximated by the average gradient outer product of the trained network taken with respect to the input of this layer. The correlation between the first layer NFM and the average gradient outer product with respect to the input data is greater than 0.97 for four different classification tasks from the CelebA dataset. We visualize the top eigenvector of these matrices and observe that both are visually indistinguishable and highlight relevant features for prediction. 60
- 5-2 **(A)** The correlation between the initial NFM and trained NFM (red) and the correlation between average gradient outer product and the trained NFM (green) for each layer in five-hidden-layer ReLU fully connected networks with 1024 hidden units per layer trained on 121 tabular data classification tasks from [61]. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix. The higher correlation with the initial NFM in layer 1 is due to the fact that these matrices are smaller than the 1024×1024 matrices in the remaining layers (on average 28.84 features across all 121 tasks). **(B)** A visualization of the 64×64 NFM at initialization, the trained NFM, and average gradient outer product for layers 2 through 5 of the network trained to classify rosy cheeks from CelebA image data [122]. While the NFM at initialization is close to the identity matrix, the NFM after training has a qualitatively different structure that is captured by the average gradient outer product (correlation $> .78$). While we omit the layer 1 visualization here since the matrices are of size 27648×27648 , the correlation between the first layer NFM after training and the corresponding average gradient outer product is .99. 62
- 5-3 The Deep Neural Feature Ansatz enables identification of and simplicity biases and spurious features in fully connected networks. **(A)** When trained on 50000 concatenated 32×64 resolution images from CIFAR10 and MNIST datasets, the diagonal of the first layer NFM of a five-hidden-layer fully connected ReLU network indicates that digit pixels are primarily used as features for classification. The average gradient outer product confirms that perturbing these pixels leads to the greatest change in predicted values. **(B)** When trained on 1000 96×96 images of planes and trucks modified with a 7×8 pixel star pattern in the upper left corner, the diagonal of the first layer NFM of a five-hidden-layer ReLU fully connected network indicates the network relies solely on the star pattern for prediction. **(C)** When trained on 40000 96×96 images from CelebA to classify lipstick, the diagonal of the first layer NFM of a five-hidden-layer ReLU fully connected network indicates the network unexpectedly relies on eye pixels for classification. To corroborate this finding, we find that perturbing test samples by masking the lips leads to only a 3.66% drop in test accuracy, but perturbing the test samples by masking the eyes based on the average gradient outer product leads to a 27.22% drop in test accuracy. 63
- 5-4 Lottery tickets in fully connected networks. **(A)** Visualizations of the diagonals of first layer Neural Feature Matrices from a two-hidden-layer, width 1024 ReLU network trained on classification tasks from CelebA and the diagonals after thresholding (replacing with pixel value 1) the top 2% of pixels. There are 553 nonzero pixel values in the masked images. **(B)** Comparison in accuracy after re-training randomly initialized neural networks of the same architecture on the masked images (i.e., pruning 98% of the corresponding columns of the first layer weights). 64

5-5	Grokking in fully connected networks. (A) Modified 96×96 resolution images from a subset of STL-10 [45] in which a small star in the upper left indicates whether the image is a truck or an airplane. We use 553 total training examples with 500 examples of airplanes and 53 examples of trucks. (B) A two-hidden-layer fully connected network quickly reaches near 100% training accuracy. Yet, as training continues past this point, the test accuracy rises drastically from 80% to 99.38%. Corresponding feature matrices (shown as inserts) indicate that test accuracy improved since the network has learned the star pixels that give away the class label.	65
5-6	Grokking in RFMs trained on modified STL-10. (A) Samples from modified STL-10 (see Section 5.2.1 and Fig. 5-5 for dataset details). (B) While training accuracy is always 100%, iteration leads to a drastic rise in test accuracy from 55.8% to 100%. Feature matrices (shown as inserts) indicate that test accuracy improved since the RFM has learned the star pixels that give away the class label.	66
5-7	(A) Comparison of 182 models including RFMs, NTKs, random forests, and fully connected neural networks on 121 tabular datasets from [61]. All metrics, models, and training details are outlined in Appendix B. RFMs took 40 minutes to achieve these results while Neural Nets took 5 hours (both measurements are in wall time on a server with two Titan Xp GPUs). (B) To determine the benefit of feature learning, we compare the error rate (100% - accuracy) between RFMs and Laplace kernels, which are equivalent to RFMs without feature learning.	67
A-1	Training details for all experiments in main text and Appendix. Unless otherwise stated, all fully connected and convolutional networks are trained to minimize mean squared error below 10^{-8} . (a) Training details for fully connected architectures including dataset description, network width, network depth, nonlinearity, optimization method, learning rate, initialization scheme, random seeds used, and reference to the experiment in the text. (b) Training details for convolutional architecture including network topology, dataset, nonlinearity, optimization method, learning rate, initialization scheme, random seed and reference to experiment in the text. (c) Training details for recurrent architecture including network width and depth (for producing the next hidden state and output state), nonlinearity, optimization method, learning rate, initialization scheme, random seed used, and reference to experiment in the text.	82
A-2	Network trained on 2000 examples from MNIST stores all training examples as attractors. 1 iteration of the network leads to good reconstruction of the test example, but taking the limit of iteration leads to recovery of training examples. Average reconstruction error after 1 iteration of 58000 test examples is 0.0135. Training details are provided in Appendix Figure A-1.	83
A-3	Network trained on 1000 black and white images from CIFAR10 stores all training examples as attractors. Analogously to Figure 2-2a of the main text, as training examples are attractors, iteration from corrupted inputs results in the recovery of a training example. Training details are provided in Appendix Figure A-1.	83
A-4	Examples of spurious training examples arising in over-parameterized autoencoders trained using different optimization methods on 100 black and white examples from CIFAR10. The networks used have 11 hidden layers, 256 hidden units per layer, SELU nonlinearity, and are initialized using the default PyTorch initialization scheme. For all optimization methods, we use a learning rate of 10^{-1} . We use a momentum value of 0.009 and weight decay of 0.0001.	84
A-5	A U-Net convolutional autoencoder [161, 185] storing 10 CIFAR10 training examples as attractors. Training details are presented in Appendix Figure A-1b. (a) Maximum eigenvalue of the Jacobian of the network at the training example. (b) Iteration from corrupted inputs converges to a training example. In general, we observe that convolutional autoencoders store training examples as attractors when the receptive field size of the hidden units in the network covers the entire image. This can be achieved by increasing the stride of the filters, as is done in the U-Net used here.	84
A-6	A U-Net convolutional autoencoder [161, 185] storing 100 ImageNet-32 training examples as attractors. Training details are presented in Appendix Figure A-1b. Iteration from corrupted inputs converges to a training example.	85

A-7	Over-parameterized autoencoders become more contractive at the training examples as network depth and width are increased. Networks are trained on 100 examples from MNIST and are a subset of architectures considered in Figure 2-5a. Histograms of the maximum eigenvalue of the Jacobian at each of the 100 training examples are presented for each of the nine settings.	85
A-8	Over-parameterized autoencoders become more contractive at the training examples as network depth and width are increased. Networks are trained on 100 examples from MNIST and are a subset of architectures considered in Figure 2-5a. Histograms of top 1% of 28^2 eigenvalues of the Jacobian at each of the 100 training examples are presented for each of the nine settings. The variance of the distribution of eigenvalues decreases as width increases.	86
A-9	Comparison of recovery rate of over-parameterized autoencoder trained on 500 examples from ImageNet-64 (Figure 1 of main text) and 1 nearest neighbor (1-NN). From Figure 2 of the main text, we know that over-parameterized autoencoders use metrics other than Euclidean distance to construct basins of attraction around training examples. However, in this high dimensional setting, the metric used by the over-parameterized autoencoder is similar to that of 1-NN, as is demonstrated by the similar recovery rates from varying corruption patterns.	87
A-10	Impact of optimizer and nonlinearity on number of training examples stored as attractors. In all experiments, we used a fully connected network with 11 hidden layers, 256 hidden units per layer, and default PyTorch initialization. (*) NA indicates that the training error did not even decrease below 10^{-5} in 1,000,000 epochs. Although more attractors arise with adaptive methods and trigonometric nonlinearities, attractors arise in all settings considered. Note that we used a loss threshold of 10^{-5} for this table since the non-adaptive methods could not converge to 10^{-8} in 1,000,000 epochs.	87
A-11	Impact of initialization on number of training examples stored as attractors. In all experiments, we used a fully connected network with 11 hidden layers and 256 hidden units per layer trained using the Adam optimizer ($lr=10^{-4}$). (*) NA indicates that the training error did not decrease below 10^{-8} in 1,000,000 epochs. Attractors arise under all settings for which training converged. Generally, more attractors arise under smaller initializations or with trigonometric nonlinearities.	87
B-1	Comparison between DNPP and using the output of DNPP as a feature prior. Using our method with the output of DNPP as a feature prior leads to an improvement in all metrics across every round of 10-fold cross validation in 5 seeds.	106
B-2	Comparison between FaLRTC and using the output of FaLRTC as a feature prior. Using our method with the output of FaLRTC as a feature prior leads to an improvement in all metrics across every round of 10-fold cross validation in 5 seeds.	107
B-3	Comparison between DNPP and FaLRTC in (a) the sparse regime (< 150 profiles per cell type) and (b) the dense regime (> 150 profiles per cell type). We observe that FaLRTC outperforms DNPP in almost every fold for all performance metrics in the sparse regime. On the other hand DNPP outperforms FaLRTC in the dense regime in every fold for all performance metrics. This result demonstrates that DNPP can be improved drastically in the sparse regime.	108
B-4	A comparison of PSNR, SSIM, and runtime for large hole image inpainting using our framework (CNTK), corresponding finite width neural networks, and biharmonic inpainting. We observe that the CNTK outperforms (in PSNR and SSIM) on average both biharmonic inpainting and finite neural networks with sigmoid last layer and batch normalization layers while maintaining a runtime that is comparable to these methods. The last column illustrates that using more advanced techniques such as Adam with Langevin dynamics [42] can be used to boost the performance of neural networks, but at additional computational cost.	108
B-5	A qualitative comparison of large hole image inpainting using our framework (CNTK), corresponding finite width neural networks, and biharmonic inpainting. See SI Fig. S4 for the corresponding quantitative comparison.	109

B-6	A comparison of PSNR, SSIM, and runtime for image reconstruction using our framework (CNTK), corresponding finite width neural networks, and biharmonic inpainting. We observe that the CNTK performs (in PSNR and SSIM) on average comparably to biharmonic inpainting and outperforms corresponding finite neural networks with sigmoid last layer and batch normalization layers. While our method is slower than biharmonic inpainting, it is more flexible than this method (see Fig. S4), and it is in average much faster than training finite width neural networks for this application.	110
B-7	Visualizing the CNTK of a neural network with nearest neighbor downsampling and upsampling layers and a uniform random feature prior illustrates that this kernel is akin to a kernel that uses different norms for image completion. In the above figure, we visualize the CNTK heatmap for coordinate (64, 64) of the CNTK for a neural network with 5 nearest neighbor downsampling and upsampling layers operating on 128×128 images. In each subfigure, we zero out the x percentile (provided below each image) of pixel values. For example, the image on the bottom right corresponds to zeroing out all pixels with values below the 90th percentile. We observe that balls of varying norms appear in this visualization: e.g., the ℓ_∞ ball appears in the upper left and an ℓ_p ball with $1 < p < 2$ on the upper right.	110
C-1	A visualization of the four functions bounding $\check{\phi}(z)$ that are used to prove Theorem 1.	115
C-2	Classifiers implemented by infinitely wide and deep networks with standard activation functions. ReLU and Sigmoid lead to classifiers that implement majority vote behavior since they satisfy the conditions of Proposition 1. On the other hand, sine, tanh, and hard tanh all lead to singular kernel classifiers with order of singularity near 0.5, which is the consistent pole order for data with density on the unit circle. Hence, these activation functions lead to infinitely wide and deep networks that are near-consistent on the unit circle, as is corroborated experimentally in Figure C-3a.	131
C-3	Comparison of infinitely wide and deep neural network classifiers with varying activation function and traditional classifiers from our taxonomy. All datasets and activation functions are detailed in the Supplementary. (a) We report the test accuracy of infinitely wide and deep classifiers across a variety of distributions and activation functions. The naming convention for our activation functions indicates the dimension for which these activation functions are consistent (e.g. Poly. 2D is consistent for data on the unit circle). Green boxes highlight activation functions that lead to networks that perform within 0.1% or better than the Hilbert estimate, which is known to be consistent [54]. Note that the best performing activation functions for each distribution satisfy the conditions from Theorem 2 in the main text. Additionally, note that in higher dimensions, $d = 5, 9$, our theoretically derived activation functions outperform standard activation functions by over 10%. (b) A list of depths used for our experiments. (c) Parameters of the Dirichlet distributions used in our experiments.	133
C-4	Comparison between finite depth NTKs using the activation functions derived in our work and a broad spectrum of 180 models including fully connected ReLU networks and ReLU NTKs on a variety of classification tasks that are widely used for benchmarking (90 classification tasks from [61]). While our models are provably optimal in the infinite-depth setting as the sample size approaches infinity, these experiments demonstrate that our activation functions also result in competitive results in the small-sample setting as well as the small-dimensional large-sample setting. All metrics and grid search parameters are presented in SI Appendix J. (a) Grid searching over finite depth NTKs using our derived activation functions leads to the best performance across all metrics over 180 models on all 90 small-sample classification tasks (with fewer than 5000 training examples in [61]), thereby outperforming the ReLU NTK, which was previously found to outperform all other methods [10]. (b) Finite depth NTKs using the activation functions derived in our work also outperform ReLU NTK and neural networks on small-dimensional large-sample classification tasks (with fewer than 15 features and greater than 5000 training examples in [61]).	135

D-1	(A) The correlation between the average gradient outer product and the trained NFM for each layer in five-hidden-layer ReLU fully connected networks trained on 6 image classification tasks from CelebA and SVHN. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix. (B) The correlation between initial NFM and trained NFM for each layer in five-hidden-layer ReLU fully connected networks trained on 6 image classification tasks from CelebA and SVHN.	149
D-2	The correlation between the average gradient outer product and the trained NFM for each layer in five-hidden-layer, width 64 fully connected networks trained on 121 tabular classification tasks using the Adam optimizer with learning rate of 10^{-4} and default initialization scheme from PyTorch across sigmoid, sine, hyperbolic tangent, and leaky ReLU activation. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix. For all activations across all layers, median correlation is above .65.	150
D-3	The correlation between the average gradient outer product and the trained NFM for each layer in five-hidden-layer ReLU fully connected networks trained on 121 tabular classification tasks using the Adam optimizer with learning rate of 10^{-4} and default initialization scheme from PyTorch across widths in the range $\{64, 128, 256, 512, 1024\}$. We observe that lower width leads to higher correlation. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix.	151
D-4	The correlation between the average gradient outer product and the trained NFM for the first layer in 1-hidden-layer, width 256 ReLU fully connected networks trained on 121 tabular classification tasks using the Adam optimizer with learning rate of 10^{-4} using an initialization scheme of Gaussian with mean 0 and standard deviation in the range $\{10^{-1}, 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}\}$. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix. We observe that lower initialization scheme leads to higher correlation.	152
D-5	The top eigenvector of the first layer NFM from a two-hidden-layer, 1024 width ReLU network and from RFM feature matrices visually highlight similar features across 12 CelebA classification tasks. These top eigenvectors are highly correlated (Pearson correlation greater than 0.99).	153
D-6	Comparison of first layer NFM features and RFM features and performance between two-hidden-layer, 1024 width ReLU fully connected networks and RFMs. (A) (Left) Samples from the SVHN dataset in which the goal is to identify the center digit from a view of potentially many digits. (Center) Upon visualizing the diagonals of the feature matrices of RFMs and deep networks, we observe that these models learn to select the center digit. (Right) By selecting the center digit, RFMs and deep networks provide a 10% increase in test accuracy over Laplace kernels and a 5% increase in test accuracy over NTKs. (B) (Left) We consider the low rank setup from [189] in which the targets, y , are generated as a product of the first two coordinates of Rademacher random variables z . (Center) Since we know the target function, we can compare the ground truth feature matrix against the first layer NFM of a 1 hidden layer ReLU fully connected network and the RFM feature matrix. We observe that both models learn to select the top two coordinates. (Right) The performance of RFMs and neural networks far exceeds of NTKs and Laplace kernels since these methods learn to select relevant coordinates for prediction. (C) (Left) The low rank setup from [52] in which the targets, y are generated as a function of a projection of the input x onto a 1 dimensional subspace. Here, u is on the unit sphere in 10 dimensions and He_2, He_4 denote the second and fourth probabilist's Hermite polynomials. (Center) While RFMs learn to accurately approximate the ground truth gradient outer product, fully connected networks require additional training modifications, as discussed in [52]. (Right) As they learn the relevant subspace, RFMs far outperform 1 hidden layer ReLU fully connected networks, NTKs, and the Laplace kernel.	154

D-7 Connections between lottery tickets in deep networks and RFMs. The diagonals of the feature matrices of RFMs trained on CelebA are sparse, thereby indicating that only a subset of coordinates is used for prediction. Such sparsity suggests that we can threshold to very few pixels while still minimally affecting performance. Indeed, re-training RFMs upon thresholding to less than 3.5% of total pixels in CelebA tasks consistently improves performance for these tasks. 155

D-8 RFMs accurately capture simplicity biases of deep fully connected networks. **(A)** We train RFMs on a dataset similar to that from [168] in which we concatenate images of CIFAR-10 objects with unique digits from MNIST. Upon visualizing the diagonals of the feature matrices of RFMs, we observe that the model learns to mask the CIFAR-10 image and focus on the MNIST digit for prediction. **(B)** The diagonal of the feature matrix for an RFM trained on lipstick classification unusually indicates that eyes are used as a key feature. We thus construct a mask based on the top RFM features and replace the eyes of all test samples with those of a single individual. The trained RFM does 39.06% worse on these corrupted samples. On the other hand, replacing the lips of all test samples with those from the same individual leads to only a minor, 6.02%, decrease in accuracy. 155

D-9 Performance of RFMs, XGBoost, Gradient Boosting Trees, Random Forests, ResNets, Transformers (SAINT and FT), and fully connected networks (MLPs) from [70]. Our method used 3600 hours in total, while all other methods used 20000 hours for tuning, as reported in [70]. Results from all models other than RFMs are reported from the tables provided by [70]. All metrics and training details are outlined in Methods. Large tasks have 50000 training examples except for Jannis (40306 examples) and Diamonds (37758 examples). The medium tasks have at most 10000 samples. We show **(A)** average accuracy on large classification tasks and **(B)** average R^2 on large regression tasks. We compare model performance through commonly used metrics across all datasets for **(C)** classification, and **(D)** regression. 156

List of Tables

5.1	Settings for which we prove the Deep Neural Feature Ansatz. <i>Activation</i> refers the type of network activation function. <i>Steps</i> refers to the number of steps of gradient descent for which the proof holds. <i>Depth</i> refers to the depth of the neural network considered. <i>Outer layers</i> describes how the layers other than the first are initialized and trained. <i>Initialization</i> refers to the initialization method of the first layer weights. <i>GIA</i> indicates whether the gradient independence ansatz is required for the result to hold. <i># Samples</i> denotes the number of training samples considered.	67
D.1	Regression R^2 without categorical variables.	156
D.2	Classification accuracy without categorical variables.	157
D.3	Regression R^2 with categorical variables.	157
D.4	Classification accuracy with categorical variables.	157

Chapter 1

Introduction

Over the past decade, interest in machine learning research has spiked, with deep learning being a significant driving force. Indeed, deep learning has been applied across various domains including computer science [35, 80] and biology [183]. Despite numerous empirical successes, our understanding of the fundamental principles making these models effective is still emerging. By identifying core principles driving the success of deep networks, we could provide a principled, cost-effective approach to designing and training machine learning models with improved performance and transparency.

In this thesis, we identify and analyze two principles, *over-parameterization* and *feature learning*, that are central contributors to the success of deep neural networks. We demonstrate that these principles in isolation are able to shed light on various properties and phenomena exhibited by deep neural networks. Moreover, we show that these two principles can be combined to develop state-of-the-art machine, transparent learning models that are computationally simple to train. Below, we summarize our contributions in the context of each of these principles. We note that each of these contributions corresponds to a self-contained chapter of the thesis.

1.1 Over-parameterization

A common practice in deep learning is to increase model size in order to improve performance [80, 100, 208]. Indeed, there is an emerging understanding that utilizing *over-parameterized* models, i.e., those that can perfectly fit training data, can lead to improved performance on supervised learning problems such as classification and regression [16, 22, 24, 75, 135].

In Chapter 2, we demonstrate a similar benefit to using over-parameterized networks for unsupervised learning tasks. In particular, we first discuss our prior work showcasing that over-parameterized autoencoders learn latent representations that are better for downstream tasks than under-parameterized autoencoders. We then analyze the properties of solutions learned by such models. Namely, we show that these models learn solutions that are contractive around training examples and thus, automatically implement a form of associative memory.

As using increasingly large, over-parameterized networks leads to improved performance in both supervised and unsupervised learning, can we train infinitely large networks to get best performance? Remarkably, under conditions on parameter initialization, training infinitely wide neural networks corresponds to training classical machine learning models known as kernel machines [96]. Training a kernel machine corresponds to the conceptually simple process of transforming data with a fixed feature map and then solving linear regression [5, 166]. Given a network architecture, the work of [96] identified a fixed feature map yielding a kernel known as the Neural Tangent Kernel (NTK) [96]. Under a Gaussian initialization scheme, as network width approaches infinity, the NTK can be computed in closed form, and one can mimic training an infinitely wide network by training a kernel machine with the NTK.

Given its conceptual simplicity, how useful is the NTK as a practical tool and as a theoretical tool for understanding properties of neural networks? In Chapter 3, we highlight the effectiveness of the NTK by deriving a closed form for the NTK of infinitely wide neural networks used in matrix completion tasks such as virtual drug screening and image inpainting. Notably, the NTK yields state-of-the-art performance

for virtual drug screening and is competitive with deep convolutional networks for high resolution image inpainting. In Chapter 4, we demonstrate that the NTK can be used to provide theoretical guarantees for neural networks. In particular, we analyze the NTK for infinitely wide and deep fully connected networks for classification tasks and identify explicit activation functions for which such models are Bayes optimal or consistent.

1.2 Feature Learning

While infinitely wide networks characterized by the NTK are effective for a number of tasks, they lack a key component that is critical to the success of neural networks: *feature learning*. Neural feature learning refers to the ability of a neural network to automatically extract patterns or features from data and is often characterized as the change in a network’s internal, hidden representations through the course of training [169, 203]. While prior work has demonstrated that feature learning can lead to improved performance [52, 169, 203], the mechanism through which deep neural networks learn features and which features are being learned has remained unclear.

In Chapter 5, we identify the mechanism of deep neural feature learning. In particular, we posit the Deep Neural Feature Ansatz, which states that neural feature learning occurs through a mathematical procedure known as average gradient outer product. This procedure corresponds to up-weighting features that most influence a trained predictor. We show that our ansatz explains prominent deep learning phenomena such as spurious features and simplicity biases [168, 171], grokking [149], and the lottery ticket hypothesis [64].

Importantly, our ansatz provides a method for incorporating feature learning into any machine learning model, including those that could not previously learn features. In particular, we apply our ansatz to enable feature learning in over-parameterized, non-feature-learning kernel machines. We refer to the resulting algorithm as a Recursive Feature Machine (RFM). We show that RFMs are (1) computationally simple, as training involves solving a series of convex optimization problems ; (2) transparent, as learned features are explicitly provided through a feature matrix ; and (3) effective, as they provide state-of-the-art results on tabular data.

1.3 Preliminaries

In this section, we provide preliminaries for neural networks and kernel machines that will be used throughout this thesis.

1.3.1 Neural Networks

We begin by outlining the basic mathematical formulation for nonlinear fully connected neural networks.

Definition 1. Given matrices $\{W^{(i)}\}_{i=1}^{L+1}$ with $W^{(i)} \in \mathbb{R}^{k_i \times k_{i-1}}$ and a function $\phi : \mathbb{R} \rightarrow \mathbb{R}$, a **fully connected neural network** is a function $f : \mathbb{R}^d \times \mathbb{R}^p \rightarrow \mathbb{R}^c$ of the form:

$$f(x; \mathbf{W}) = W_{L+1} h_L(x) \quad ; \quad h_\ell(x) = \phi(W_\ell h_{\ell-1}(x)) \text{ for } \ell \in \{1, \dots, L\} ;$$

where $h_1(x) = x$, $k_0 = d$, $k_{L+1} = c$, $p = \sum_{i=1}^{L+1} k_i k_{i-1}$, and ϕ is applied element-wise. The choice L denotes the number of *hidden layers* in the network, k_i denotes the *width* of layer i , and ϕ denotes the *activation function*. Depending on the context, the *depth* of a network can either refer to the number of hidden layers, L , or the number of weight matrices, $L + 1$, and so, we will make our definition of depth explicit in each of the chapters. The vector $\mathbf{W} \in \mathbb{R}^p$ serves as a list of all parameters in the neural network.

It is common practice to train a neural network using a first order optimization method such as gradient descent, which is defined below.

Definition 2. Given a dataset $\{(x_p, y_p)\}_{p=1}^n \in \mathbb{R}^d \times \mathbb{R}^c$, a loss function $\mathcal{L} : \mathbb{R}^c \times \mathbb{R}^c \rightarrow \mathbb{R}$, and a neural

network f with initial parameters $\mathbf{W}^{(0)}$, **gradient descent** with learning rate $\eta > 0$ proceeds as follows:

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta \sum_{p=1}^n \nabla_{\mathbf{W}} \mathcal{L}(f(x_p, \mathbf{W}^{(t)}), y_p).$$

A common choice for loss function is the square loss, which is given by $\mathcal{L}(y, \tilde{y}) = \frac{1}{2} \|y - \tilde{y}\|_2^2$. For the definition of other commonly used loss functions including logistic or cross entropy loss see [69]. At the time of writing, we note that there are over 10 different variants of gradient descent present in the literature [142]. Out of these variants, we will primarily resort to using gradient descent and the Adam optimizer [101].

1.3.2 Kernel Machines

We next outline the basics of kernel machines and kernel regression that will be used in this thesis. We refer the reader to [166] and [5] for further background on kernels and their applications. Given a training dataset $\{(x_p, y_p)\}_{p=1}^n \in \mathbb{R}^d \times \mathbb{R}^c$, kernel regression corresponds to first transforming each of the points x_p with a fixed feature map $\psi : \mathbb{R}^d \rightarrow \mathcal{H}$, where \mathcal{H} is a Hilbert space, and then performing linear regression on the transformed data $\{(\psi(x_p), y_p)\}_{p=1}^n \in \mathcal{H} \times \mathbb{R}^c$. To perform linear regression when \mathcal{H} is an infinite dimensional space, we note that finding the minimum norm solution for linear regression on requires knowledge of inner products of transformed data, i.e., $\langle \psi(x), \psi(z) \rangle_{\mathcal{H}}$ for $x, z \in \mathbb{R}^d$ [190]. These inner products can be abstracted to a general positive semi-definite, symmetric function $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ known as a kernel.

Definition 3. Given a set \mathcal{X} , a symmetric function $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a **positive semi-definite kernel** iff for any $\{x^{(i)}\}_{i=1}^n \subset \mathcal{X}$ and for any $\{c_i\}_{i=1}^n \subset \mathbb{R}$,

$$\sum_{i=1}^n \sum_{j=1}^n c_i c_j K(x^{(i)}, x^{(j)}) \geq 0.$$

Given a training dataset as above and a kernel function, K , we perform kernel regression as follows:

1. Build the training kernel matrix, $\hat{K} \in \mathbb{R}^{n \times n}$ with $\hat{K}_{i,j} = K(x_i, x_j)$.
2. Solve the system of equations $\alpha \hat{K} = y$ where $\alpha \in \mathbb{R}^{c \times n}$ and $y = [y_1, \dots, y_p] \in \mathbb{R}^{c \times n}$.

The resulting predictor, $\hat{f} : \mathbb{R}^d \rightarrow \mathbb{R}^c$, is given by $\hat{f}(z) = \alpha K(X, z)$ where $K(X, z) \in \mathbb{R}^n$ with $K(X, z)_i = K(x_i, z)$. Unlike the case of neural networks, we note that training a kernel machine involves solving a convex optimization problem corresponding to a linear system of equations.

1.3.3 Neural Tangent Kernel

We lastly discuss prior literature [96] on the Neural Tangent Kernel (NTK) connection between neural networks and kernel machines that will be used throughout this thesis. We start with the definition of the NTK.

Definition 4 (NTK). Let $f(x; \mathbf{W}) : \mathbb{R}^d \times \mathbb{R}^p \rightarrow \mathbb{R}$ denote a neural network with parameters \mathbf{W} . The **neural tangent kernel**, $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, is a symmetric, positive definite function given by:

$$K(x, x') = \langle \nabla_{\mathbf{W}} f(x; \mathbf{W}^{(0)}), \nabla_{\mathbf{W}} f(x'; \mathbf{W}^{(0)}) \rangle,$$

where $\mathbf{W}^{(0)} \in \mathbb{R}^p$ denotes the parameters at initialization.

Consider fully connected networks of the following form:

$$f(x; \mathbf{W}) = W^{(L)} \frac{c}{\sqrt{k_L}} \phi \left(W^{(L-1)} \frac{c}{\sqrt{k_{L-1}}} \phi \left(\dots \frac{c}{\sqrt{k_1}} \phi \left(W^{(1)} x \right) \dots \right) \right), \quad (1.1)$$

where $\mathbf{W} = \{W^{(i)}\}_{i=1}^L$ with $W^{(i)} \in \mathbb{R}^{k_i \times k_{i-1}}$ and $k_0 = d, k_L = 1$; $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is an elementwise Lipschitz nonlinearity; and c is a constant. The key finding of [96] is that when $\mathbf{W}_i \stackrel{i.i.d.}{\sim} \mathcal{N}(0, 1)$, then as $k_1, k_2, \dots, k_L \rightarrow \infty$, $K_L(x, x')$ converges in probability to a deterministic kernel that does not change through training. Thus, solving kernel regression with kernel K_L is equivalent to the solution given by training the neural network. We present the case for fully connected networks from [96] below, and we note that the NTK can be computed for various architectures such as convolutional networks [9].

Theorem. *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a neural network defined in Eq. [1.1]. As $k_1, k_2, \dots, k_L \rightarrow \infty$, then $K_L(x, x')$ converges in probability to a deterministic kernel given by the following recurrences in $\Sigma_i, \dot{\Sigma}_i, K_i$:*

$$\begin{aligned} K_0(x, x') &= \Sigma_0(x, x') = x^T x', \\ K_L(x, x') &= \Sigma_L(x, x') + K_{L-1}(x, x') \Sigma'_{L-1}(\Sigma_{L-1}(x, x')), \\ \Sigma_L(x, x') &= c^2 \mathbb{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \Lambda_{L-1}(x, x'))} [\phi(u) \phi(v)], \\ \dot{\Sigma}_L(x, x') &= c^2 \mathbb{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \Lambda_{L-1}(x, x'))} [\phi'(u) \phi'(v)], \\ \Lambda_L(x, x') &= \begin{bmatrix} \Sigma_{L-1}(x, x) & \Sigma_{L-1}(x, x') \\ \Sigma_{L-1}(x', x) & \Sigma_{L-1}(x', x') \end{bmatrix}. \end{aligned}$$

Dual Activations. The expectations in the recurrences above can be simplified using the theory of dual activation functions studied in [53]. Let $\check{\phi} : [-1, 1] \rightarrow \mathbb{R}$ such that:

$$\check{\phi}(\xi) = c^2 \mathbb{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \Lambda)} [\phi(u) \phi(v)], \quad \Lambda = \begin{bmatrix} 1 & \xi \\ \xi & 1 \end{bmatrix}, \quad \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} \phi(u)^2 \exp\left(-\frac{u^2}{2}\right) du = \frac{1}{c^2}. \quad (1.2)$$

The map \mathcal{F} such that $\mathcal{F}(\phi) = \check{\phi}$ is an operator mapping from activation functions to positive definite functions¹, and $\check{\phi}$ is referred to as the *dual activation* [53]. The scaling factor c in the theorem above is typically selected to satisfy the integral equation in Eq. [1.2]. As an example, when ϕ is the ReLU, the integral is just $\frac{1}{2}$ times the second moment of the standard Gaussian distribution. Hence, $c = \sqrt{2}$ for the ReLU. The recurrence relation for the NTK can be drastically simplified for homogenous nonlinearities for which the dual activation has a closed form. As shown in prior work [43, 182], this is the case for the commonly used ReLU and LeakyReLU nonlinearities. In particular, the dual activation function for ReLU is well known [43], and we next present its form (with its derivative):

Lemma. *The dual activation $\hat{\phi} : [-1, 1] \rightarrow \mathbb{R}$ of the ReLU is:*

$$\begin{aligned} \hat{\phi}(\xi) &= \frac{1}{\pi} (\xi(\pi - \cos^{-1}(\xi)) + \sqrt{1 - \xi^2}) \\ \frac{d\hat{\phi}(\xi)}{d\xi} &= \frac{1}{\pi} (\pi - \cos^{-1}(\xi)) \end{aligned} \quad (1.3)$$

As shown in [32, 39], the NTK recursion for ReLU networks can be simplified using the dual activation. We provide this known simplification below for completeness.

Proposition. *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a neural network defined in Eq. [1.1]. Let ϕ be the ReLU activation and let $c = \sqrt{2}$. As $k_1, k_2, \dots, k_L \rightarrow \infty$, then $K_L(x, x')$ converges in probability to a deterministic kernel given by*

¹The map \mathcal{F} is more precisely from the Hilbert space $L^2(\mu)$ with μ the Gaussian measure to the space of positive definite functions. The factor c is selected so that $\|\phi\|_{L^2(\mu)} = 1$.

the following recurrences in Σ_i, K_i :

$$\begin{aligned}
K_0(x, x') &= \Sigma_0(x, x') = x^T x', \\
\Sigma_L(x, x') &= N_{L-1}(x, x') \check{\phi} \left(\frac{\Sigma_{L-1}(x, x')}{N_{L-1}(x, x')} \right), \\
K_L(x, x') &= \Sigma_L(x, x') + K_{L-1}(x, x') \frac{d\check{\phi}}{d\xi} \left(\frac{\Sigma_{L-1}(x, x')}{N_{L-1}(x, x')} \right), \\
N_{L-1}(x, x') &= \sqrt{\Sigma_{L-1}(x, x) \Sigma_{L-1}(x', x')}.
\end{aligned}$$

This proposition follows from using the change of variables $u = \sqrt{\Sigma_L(x, x)}\tilde{u}$ and $v = \sqrt{\Sigma_L(x', x')}\tilde{v}$ and the homogeneity of ReLU when computing $c^2 \mathbb{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \Lambda_{L-1}(x, x'))}[\phi(u)\phi(v)]$ using integration [32].

Given a closed form for the NTK, we can simply solve kernel regression with the NTK to train an infinite width version of the neural network in Eq. [1.1] initialized with weights drawn from a standard Gaussian distribution. In Chapters 3 and 4, we will showcase the practical value of using the NTK and demonstrate how the NTK can provide insights about properties of wide and deep neural networks.

Chapter 2

Over-parameterized Autoencoders and Sequence Encoders

Autoencoders are classical, unsupervised neural network models for learning representations from data [15]. Since their inception, autoencoders have always contained a bottleneck layer in which network width is smaller than the input dimension. The motivation behind such bottlenecks is to prevent the model from learning the identity function, which would correspond to essentially copying input to output without learning a meaningful latent representation.

Given the benefit of over-parameterization in supervised learning tasks, is there a benefit to using over-parameterized autoencoders, i.e., those with no bottleneck layers, or will such models simply learn the identity function? Remarkably, we found that there is indeed a benefit to using over-parameterized autoencoders in practice. In recent work [30], we found that applying over-parameterized autoencoders to transcriptomics data led to latent representations in which the effect of drugs was more aligned across cell types than in the original space. Such representations would not be possible had these models learned the identity function.

Given that over-parameterized autoencoders are clearly not learning the identity function, in the following chapter, we characterize the functions they do learn. In particular, we demonstrate that these models learn functions that are contractive around the training examples, thereby implementing a computational model of associative memory. The work presented in this section culminated into the following paper [152].

2.1 Introduction

Developing computational models of *associative memory*, a system which can recover stored patterns from corrupted inputs, is a long-standing problem at the intersection of machine learning and neuroscience. An early example of a computational model for memory dates back to the introduction of Hopfield networks [85, 117]. Hopfield networks are an example of an *attractor network*, a system which allows for the recovery of patterns by storing them as attractors of a dynamical system. In order to write patterns into memory, Hopfield networks construct an energy function with local minima corresponding to the desired patterns. To retrieve these stored patterns, the constructed energy function is iteratively minimized starting from a new input pattern until a local minimum is discovered and returned.

While Hopfield networks can only store binary patterns, the simplicity of the model allowed for a theoretical analysis of capacity [127]. In order to implement a form of associative memory for more complex data modalities, such as images, the idea of storing training examples as the local minima of an energy function was extended by several recent works [17, 58, 82, 83, 164]. Unlike Hopfield networks, these modern methods do not guarantee that a given pattern can be stored and typically lack the capacity to store patterns exactly; see e.g. [17].

Our main finding is that standard over-parameterized neural networks trained using standard optimization methods can implement associative memory. In contrast to energy-based methods, the storage and retrieval mechanisms are automatic consequences of training and do not require constructing and minimizing an energy function.

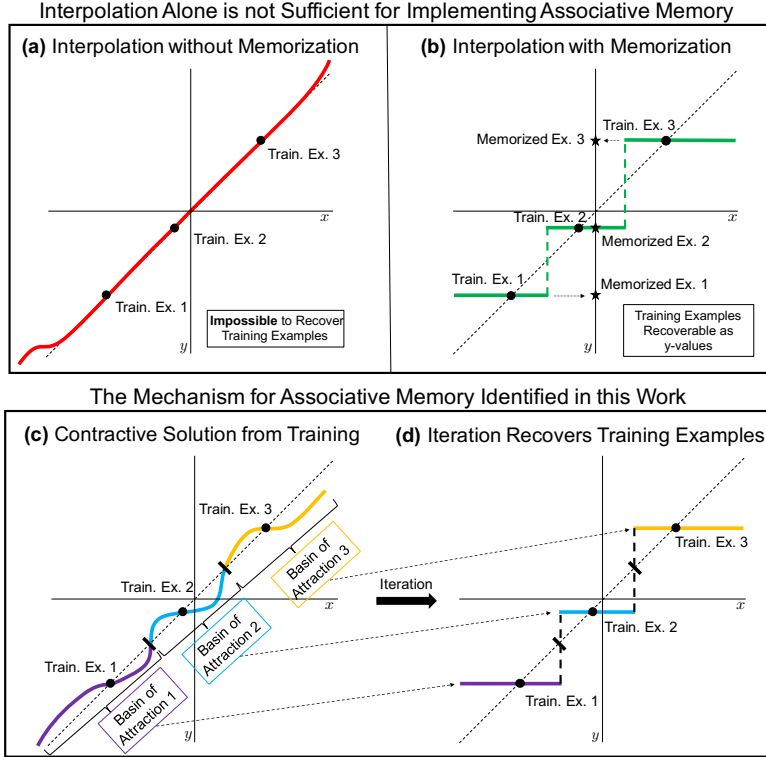


Figure 2-1: The difference between associative memory and interpolation is described in (a, b); the mechanism identified in this work by which overparameterized autoencoders implement associative memory is described in (c, d). Training examples are shown as black points, the identity function is shown as a dotted line and functions are represented using solid, colored lines. (a) An example of a function that interpolates the training data but does not memorize training data: training data are not recoverable from just the function alone. (b) An example of a function that interpolates and memorizes training data: training data are recoverable as the range of the function. (c) An example of an interpolating function for which the training examples are attractors; the basin of attraction are shown. (d) Iteration of the function from (c) leads to a function that is piece-wise constant almost everywhere, with the training examples corresponding to the non-trivial constant regions. The fact that the training examples are attractors implies that iteration provides a retrieval mechanism.

Interpolation Alone is Not Sufficient for Implementing Associative Memory. While in recent machine learning literature (e.g., [11, 208]) the term memorization is often used interchangeably with interpolation, the ability of a model to perfectly fit training data, note that memorization is stronger and also requires a model to be able to recover training data. In general, interpolation does not guarantee the ability to recover training data nor does it guarantee the ability to associate new inputs with training examples. Fig. 2-1a shows an example of a function that interpolates training data, but does not implement associative memory: there is no apparent method to recover the training examples from the function alone. On the other hand, Fig. 2-1b gives an example of a function that implements memory: the training examples are retrieved as the range of the function.

While it has been observed (e.g., [208]) that over-parameterized networks can interpolate the training data, there is no apriori reason why it should be possible to recover the training data from the network. In this work, we show that, remarkably, the retrieval mechanism also follows naturally from training: the examples can be recovered simply by iterating the learned map. A depiction of the memorization and retrieval mechanisms is presented in Fig. 2-1c and 2-1d. More precisely, given a set of training examples $\{x^{(i)}\}_{i=1}^n \subset \mathbb{R}^d$ and an overparameterized neural network implementing a family of continuous functions $\mathcal{F} = \{f : \mathbb{R}^d \rightarrow \mathbb{R}^d\}$, we show that minimizing the following *autoencoding* objective with gradient descent

methods leads to training examples being stored as attractors:

$$\arg \min_{f \in \mathcal{F}} \sum_{i=1}^n \|f(x^{(i)}) - x^{(i)}\|^2 \tag{2.1}$$

Interestingly, attractors arise without any specific regularization to the above loss function. We demonstrate this phenomenon by presenting a wealth of empirical evidence, including a network that stores 500 images from ImageNet-64 [141] as attractors. In addition, we present a proof of this phenomenon for over-parameterized networks trained on single examples.

Furthermore, we show that a slight modification of the objective (2.1) leads to an implementation of associative memory for sequences. More precisely, given a sequence of training examples $\{x^{(i)}\}_{i=1}^n \subset \mathbb{R}^d$, minimizing the following *sequence encoding* objective with gradient descent methods leads to the training sequence being stored as a stable limit cycle:

$$\arg \min_{f \in \mathcal{F}} \sum_{i=1}^n \|f(x^{((i \bmod n)+1)}) - x^{(i)}\|^2. \tag{2.2}$$

Multiple cycles can be encoded similarly (Appendix A). In particular, we provide several examples of networks storing video and audio samples as limit cycles. Interestingly, these experiments suggest that sequence encoding provides a more efficient mechanism for memorization and retrieval of training examples than autoencoding. By considering a sequence encoder as a composition of maps, we indeed prove that sequence encoders are more contractive to a sequence of examples than autoencoders are to individual examples.

2.2 Related Work

Autoencoders [15] are commonly used for manifold learning, and the autoencoder architecture and objective (Eq. (2.1)) have been modified in several ways to improve their ability to represent data manifolds. Two variations, contractive and denoising autoencoders, add specific regularizers to the objective function in order to make the functions implemented by the autoencoder contractive towards the training data [4, 159, 188]. However, these autoencoders are typically used in the under-parameterized regime, where they do not have the capacity to interpolate (fit exactly) the training examples and hence cannot store the training examples as fixed points.

On the other hand, it is well-known that over-parameterized neural networks can interpolate the training data when trained with gradient descent methods [56, 57, 196, 208]. As a consequence, over-parameterized autoencoders can store training examples as fixed points. In particular, recent work empirically studied over-parameterized autoencoders in the setting with one training example [209].

In this paper, we take a dynamical systems perspective to study over-parameterized autoencoders and sequence encoders. In particular, we show that not only do over-parameterized autoencoders (sequence encoders) trained using standard methods store training examples (sequences) as fixed points (limit cycles), but that these fixed points (limit cycles) are *attractors (stable)*, i.e., they can be recovered via iteration. While energy-based methods have also been shown to be able to recall sequences as stable limit cycles [36, 103], the mechanism identified here is unrelated and novel: it does not require setting up an energy function and is a direct consequence of training an over-parameterized network.

Background from Dynamical Systems

We now introduce tools related to dynamical systems that we will use to analyze autoencoders and sequence encoders.

Attractors in Dynamical Systems. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ denote the function learned by an autoencoder trained on a dataset $X = \{x^{(i)}\}_{i=1}^n \subset \mathbb{R}^d$. Consider the sequence $\{f^k(x)\}_{k \in \mathbb{N}}$ where $f^k(x)$ denotes k compositions of f applied to $x \in \mathbb{R}^d$. A point $x \in \mathbb{R}^d$ is a *fixed point* of f if $f(x) = x$; in this case the sequence $\{f^k(x)\}_{k \in \mathbb{N}}$ trivially converges to x .

Since overparameterized autoencoders interpolate the training data, it holds that $f(x^{(i)}) = x^{(i)}$ for each training example $x^{(i)} \in X$; hence all training examples are fixed points of f .¹ We now formally define what it means for a fixed point to be an *attractor* and provide a sufficient condition for this property.

Definition 5. A fixed point $x^* \in \mathbb{R}^d$ is an **attractor** of $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ if there exists an open neighborhood, \mathcal{O} , of x^* , such that for any $x \in \mathcal{O}$, the sequence $\{f^k(x)\}_{k \in \mathbb{N}}$ converges to x^* as $k \rightarrow \infty$. The set S of all such points is called the **basin of attraction** of x^* .

Proposition 1. A fixed point $x^* \in \mathbb{R}^d$ is an attractor of a differentiable map f if all eigenvalues of the Jacobian of f at x^* are strictly less than 1 in absolute value. If any of the eigenvalues are greater than 1, x^* cannot be an attractor.

Proposition 1 is a well-known condition in the theory of dynamical systems (Chapter 6 of [177]). The condition intuitively means that the function f is “flatter” around an attractor x^* . Since training examples are fixed points in over-parameterized autoencoders, from Proposition 1, it follows that a training example is an attractor if the maximum eigenvalue (in absolute value) of the Jacobian at the example is less than 1. Since attractors are recoverable through iteration, autoencoders that store training examples as attractors guarantee recoverability of these examples. Energy-based methods also allow for verification of whether a training example is an attractor. However, this requires checking the second order condition that the Hessian is positive definite at the training example, which is more computationally expensive than checking the first order condition from Proposition 1.

Discrete Limit Cycles in Dynamical Systems. Discrete limit cycles can be considered the equivalent of an attractor for sequence encoding, and a formal definition is provided below.

Definition 6. A finite set $X^* = \{x^{(i)}\}_{i=1}^n \subset \mathbb{R}^d$ is a **stable discrete limit cycle** of a smooth function $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ if: (1) $f(x^{(i)}) = x^{(i \bmod n)+1} \forall i \in \{1, \dots, n\}$; (2) There exists an open neighborhood, \mathcal{O} , of X^* such that for any $x \in \mathcal{O}$, X^* is the limit set of $\{f^k(x)\}_{k=1}^\infty$.

The equivalent of Proposition 1 for verifying that a finite sequence of points forms a limit cycle is provided below.

Proposition 2. Let network $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ be trained on a given sequence $x^{(1)}, \dots, x^{(n)}$ such that $f(x^{(i)}) = x^{(i \bmod n)+1}$. Then the sequence $\{x^{(i)}\}_{i=1}^n$ forms a stable discrete limit cycle if the largest eigenvalue of the Jacobian of $f^n(x^{(i)})$ is (in absolute value) less than 1 for any i .

This follows directly by applying Proposition 1 to the map f^n , since $x^{(i)} = f^n(x^{(i)})$ and $f(x^{(i)}) = x^{(i \bmod n)+1}$. Before presenting our results, we provide the following important remark.

Why the Emergence of Attractors in Autoencoders is Notable. Proposition 1 states that for a fixed point to be an attractor, all eigenvalues of the Jacobian at that point must be less than 1 in absolute value. Since the number of eigenvalues of the Jacobian equals the dimension of the space, this means that the angle of the derivative is less than $\pi/4$ in *every* eigendirection of the Jacobian. This is a highly restrictive condition, since intuitively, we expect the “probability” of such an event to be $1/2^d$. Hence, a fixed point of an arbitrary high-dimensional map is unlikely to be an attractor. Indeed, as we show in Corollary 1, fixed points of neural networks are not generally attractors. While not yet fully understood, the emergence and, indeed, proliferation of attractors in autoencoding is not due solely to architectures but to specific inductive biases of the training procedures.

2.3 Empirical Findings

Training Examples are Stored as Attractors in Over-parameterized Autoencoders. In the following, we present a range of empirical evidence that attractors arise in autoencoders across common architectures and optimization methods. For details on the specific architectures and optimization schemes used for each experiment, see Appendix, Fig A-1.

¹To ensure $f(x^{(i)}) \approx x^{(i)}$, it is essential to train until the loss is very small; we used $\leq 10^{-8}$.

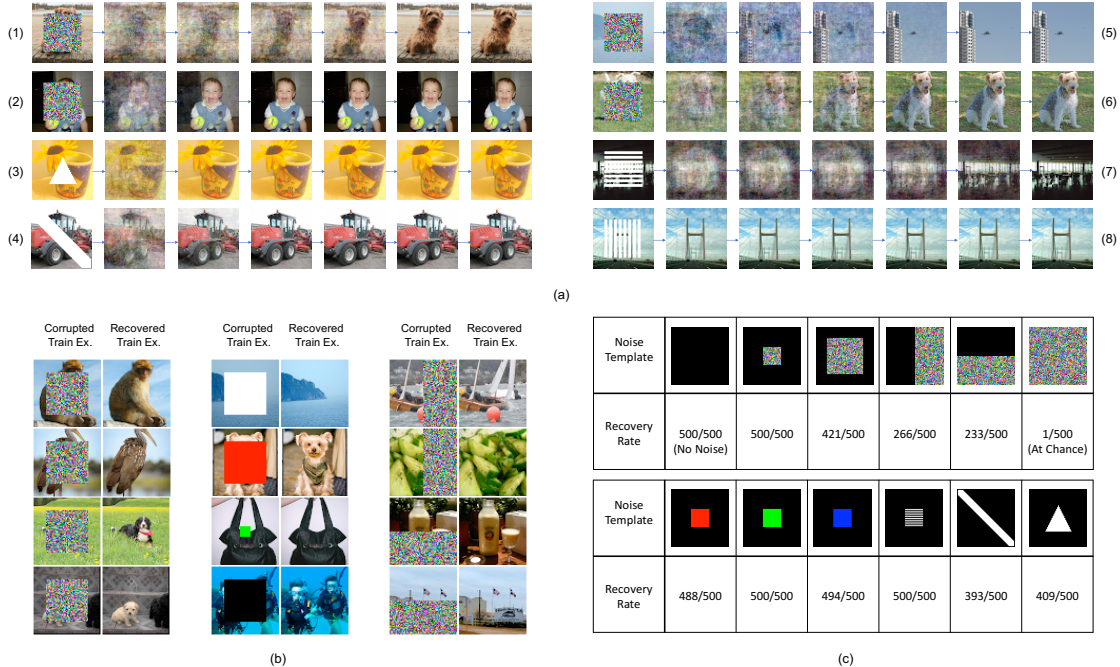


Figure 2-2: Example of an over-parameterized autoencoder storing 500 images from ImageNet-64 as attractors after training to a reconstruction error of less than 10^{-8} . Architecture and optimizer details are provided in Appendix, Fig. A-1. (a) By iterating the trained autoencoder on corrupted versions of training samples, individual training samples are recovered. (b) Samples that are corrupted by uniform random noise or squares of varying color and size are recovered via iteration. (c) Fraction of samples recovered correctly from different noise applied to the training images. A sample is considered recovered when the error between the original sample and the recovered sample is less than 10^{-7} .

Storing Images as Attractors. In Fig. 2-2, we present an example of an over-parameterized autoencoder storing 500 images from ImageNet-64 [141] as attractors. This was achieved by training an autoencoder with depth 10, width 1024, and cosid nonlinearity [59] on 500 training examples using the Adam [101] optimizer to loss $\leq 10^{-8}$. We verified that all 500 training images were stored as attractors by checking that the magnitudes of all eigenvalues of the Jacobian matrix at each example were less than 1. Indeed, Fig. 2-2a demonstrates that iteration of the trained autoencoder map starting from corrupted inputs converges to individual training examples. A common practice for measuring recoverability of training patterns is to input corrupted versions of the patterns and verify that the system is able to recover the original patterns. From Proposition 1, provided that a corrupted example is in the basin of attraction of the original example, iteration is guaranteed to converge to the original example. In examples (5) and (6) Fig. 2-2a, the corrupted images are not in the basin of attraction for the original examples, and so iteration converges to a different (but contextually similar) training example. Fig. 2-2b provides further examples of correct recovery from corrupted images. Fig. 2-2c presents a quantitative analysis of the recovery rate of training examples under various forms of corruption. Overall, the recovery rate is remarkably high: even when 50% of the image is corrupted, the recovery rate of the network is significantly higher than expected by chance.

Examples of autoencoders storing training examples as attractors when trained on 2000 images from MNIST [107] and 1000 black-and-white images from CIFAR10 [105] are presented in the Appendix Fig. A-2, Fig. A-3, respectively. The MNIST autoencoder presented in Appendix, Fig. A-2 stores 2000 training examples as attractors. Note that one iteration of the learned map on test examples can look similar to the identity function, but in fact, iterating until convergence yields a training example (see Appendix Fig. A-2).

Spurious Attractors. While in these examples, we verified that the training examples were stored as attractors by checking the eigenvalue condition, there could be *spurious attractors*, i.e. attractors other than the training examples. In fact, spurious attractors are known to exist for Hopfield networks [86]. To

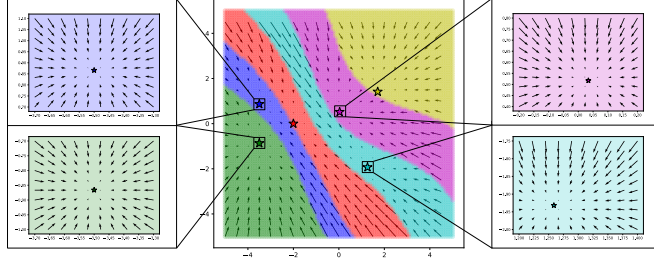


Figure 2-3: Example of an over-parameterized autoencoder in the 2-dimensional setting storing training examples (represented as stars) as attractors. Basins of attraction for each sample are colored by sampling 10,000 points in a grid around the training examples, taking the limit of the iteration for each point, and assigning a color to the point based on the training example indicated by the limit. The vector field indicates the direction of motion given by iteration, and the inserts indicate that iteration leads to training examples for all points in an open set around each example.

investigate whether there are additional attractors outside of the training examples, we iterated the map from sampled test images and randomly generated images until convergence. More precisely, we declared convergence of the map at iteration k for some image x when $\|f^{k+1}(x) - f^k(x)\|_2 < 10^{-8}$ and concluded that $f^k(x)$ had converged to the training example $x^{(i)}$ if $\|f^k(x) - x^{(i)}\|_2 < 10^{-7}$.

In general, spurious attractors can exist for over-parameterized autoencoders, and we provide examples in the Appendix Fig. A-4. However, remarkably, for the network presented in Fig. 2-2, we could not identify any spurious attractors even after iterating the trained map from 40,000 test examples from ImageNet-64, 10,000 examples of uniform random noise, and 10,000 examples of Gaussian noise with variance 4.

Attractors Arise across Architectures, Training Methods & Initialization Schemes. We performed a thorough analysis of the attractor phenomenon identified above across a number of common architectures, optimization methods, and initialization schemes. Starting with fully connected autoencoders, we analyzed the number of training examples stored as attractors when trained on 100 black and white images from CIFAR10 [105] under the following nonlinearities, initializations, and optimization methods:

- *Nonlinearities:* ReLU, Leaky Relu, SELU, $\text{cosid}(\cos x - x)$, Swish [59, 102, 156, 200], and sinusoidal $(x + (\sin 10x)/5)$.
- *Optimization Methods:* Gradient Descent (GD), GD with momentum, GD with momentum and weight decay, RMSprop, and Adam (Ch. 8 of [69]).
- *Initialization Schemes:* Random uniform initialization, namely $U[-a, a]$, per weight for $a \in \{0.01, 0.02, 0.05, 0.1, 0.15\}$. These initialization schemes subsume the PyTorch (Version 0.4) default, Xavier initialization, and Kaiming initialization [67, 79, 142].

In Appendix, Fig. A-10 and A-11, we provide the number of training examples stored as attractors for all possible combinations of (a) nonlinearity and optimization method listed above; and (b) nonlinearity and initialization scheme listed above. These tables demonstrate that attractors arise in all settings for which training converged to a sufficiently low loss within 1,000,000 epochs. In Appendix Figures A-10, A-11 and Appendix F, we also present examples of convolutional and recurrent networks that store training examples as attractors, thereby demonstrating that this phenomenon is not limited to fully connected networks and occurs in all commonly used network architectures.

Visualizing Attractors in 2D. In order to better understand the attractor phenomenon, we present an example of an over-parameterized autoencoder storing training examples as attractors in the 2D setting, where the basins of attraction can easily be visualized (Fig. 2-3). We trained an autoencoder to store 6 training examples as attractors. Their basins of attraction were visualized by iterating the trained autoencoder map starting from 10,000 points on a grid until convergence. The vector field indicates the direction of motion given by iteration. Also in this experiment, we found no spurious attractors. Each training example and corresponding basin of attraction is colored differently. Interestingly, the example in Fig. 2-3 shows that

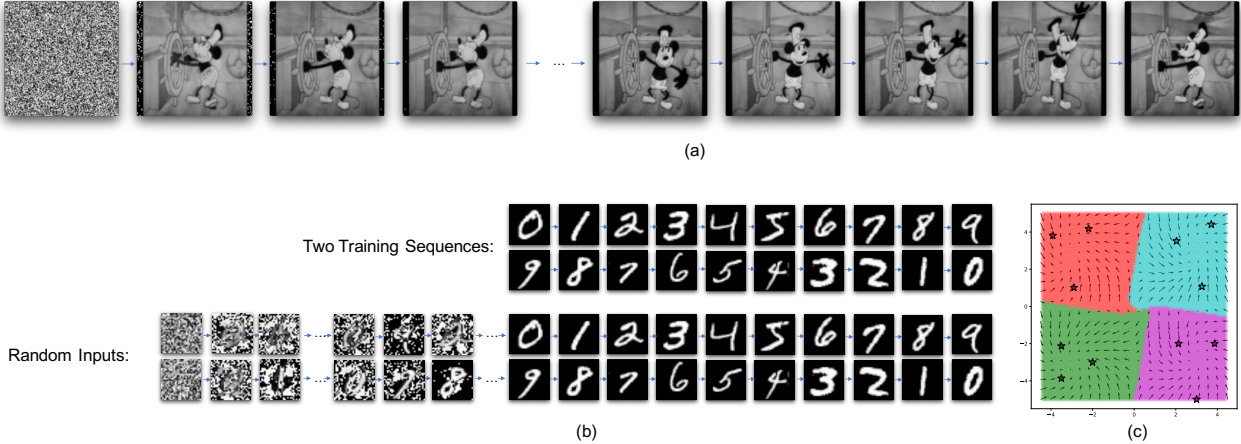


Figure 2-4: Examples of over-parameterized sequence encoders storing training sequences as limit cycles. Architecture and optimizer details are provided in Appendix, Fig. A-1. (a) When trained on 389 frames of size 128×128 from the Disney film “Steamboat Willie”, the entire movie was stored as a limit cycle. Hence, iteration from random noise leads to recovery of the entire sequence. (b) When trained on two sequences of length 10 from MNIST, each sequence was stored as a limit cycle. Hence iteration from random noise leads to the recovery of each individual sequence. (c) Visualization of the basins of attraction for a sequence encoder storing 4 sequences as limit cycles in the 2-dimensional setting. The vector field indicates the direction of motion given by iteration.

the metric learned by the autoencoder to separate the basins of attraction is not Euclidean distance, which would be indicated by a Voronoi diagram.

Over-parameterized Sequence Encoders Store Training Examples as Stable Limit Cycles and are More Efficient at Memorizing and Retrieving Examples than Autoencoders. We have thus far analyzed the occurrence of attractors in over-parameterized autoencoders. In this section, we demonstrate via various examples that by modifying the autoencoder objective to encode sequences (Eg. [2.2]), we can implement a form of associative memory for sequences. For details on the specific architectures and optimization schemes used for each experiment, see Appendix Figure A-1.

Storing Sequences as Limit Cycles. We trained a network to encode 389 frames of size 128×128 from the Disney film “Steamboat Willie” by mapping frame i to frame $i + 1 \pmod{389}$. Fig. 2-4(a) and the video² show that iterating the trained network starting from random noise yields the original video.

As a second example, we encoded two 10-digit sequences from MNIST: one counting upwards from digit 0 to 9 and the other counting down from digit 9 to 0. The maximal eigenvalues of the Jacobian of the trained encoder composed 10 times is 0.0034 and 0.0033 for the images from the first and second sequence, respectively. Hence by Proposition 2, both sequences form limit cycles. Indeed, as shown in Fig. 2-4(b), iteration from Gaussian noise leads to the recovery of both training sequences.

Finally, in Fig. 2-4(c), we visualized the vector field and basins of attraction for four cycles in the 2-dimensional setting. Unlike autoencoding where points near a training example are pushed towards it via iteration, the points now move following the cycles. In Appendix G, we also trained a sequence encoder that stores 10 seconds of speech as a limit cycle.

Efficiency of Sequence Encoding. In Fig. 2-5, we analyze the network sizes (width and depth) needed to store and retrieve 100 training images from MNIST using fully connected autoencoders and sequence encoders. Interestingly, our experiments indicate that memorization and retrieval of training examples can be performed more efficiently through sequence encoding than autoencoding. In particular, Fig. 2-5(a) shows the number of training examples (out of 100) that are attractors for different width and depth of the network. Note that a depth of 31 and width of 512 is needed to store almost all (99) training examples. If we instead

²Located at: https://github.com/uhlerlab/neural_networks_associative_memory

		Width		
		128	256	512
Depth	1	0	0	0
	6	0	4	12
	11	2	8	24
	16	22	38	49
	21	56	68	75
	26	83	86	94
31	92	90	99	

(a) Autoencoding 100 MNIST Examples

		Width		
		128	256	512
Depth	1	0	0	55
	6	0	65	100
	11	0	65	100
	16	15	75	100
	21	40	85	100
	26	75	95	100
31	90	100	100	

(b) Sequence Encoding 100 MNIST Examples as 20 Sequences of Length 5

		Width		
		128	256	512
Depth	1	0	0	0
	6	0	30	100
	11	0	50	100
	16	0	90	100
	21	60	90	100
	26	80	100	100
31	100	100	100	

(c) Sequence Encoding 100 MNIST Examples as 10 Sequences of Length 10

		Width		
		128	256	512
Depth	1	0	0	100
	6	0	100	100
	11	0	100	100
	16	0	100	100
	21	60	100	100
	26	100	100	100
31	100	100	100	

(d) Sequence Encoding 100 MNIST Examples as 5 Sequences of Length 20

		Width		
		128	256	512
Depth	1	0	0	100
	6	0	100	100
	11	0	100	100
	16	0	100	100
	21	0	100	100
	26	100	100	100
31	100	100	100	

(e) Sequence Encoding 100 MNIST Examples as 1 Sequence of Length 100

Figure 2-5: Sequence encoders are more efficient at implementing associative memory than autoencoders. Number of training examples recovered are out of 100; architecture and optimizer details are provided in the Appendix, Fig. A-1. (a) Number of recovered images when autoencoding 100 examples from MNIST individually; a network of depth 31 and width 512 recovers 99 images out of 100. (b)-(e) Sequence encoding the same 100 MNIST examples as sequences of different lengths improves the recovery rates; in particular, a network of depth 1 and width 512 recovers the full 10 images when encoded as 5 sequences of length 20 (d) or 1 sequence of length 100 (e).

encode the same data using 20 sequences of length 5, all 20 sequences (and thus all 100 examples) can be recovered using a much smaller network with a depth of 6 and 512 hidden units per layer (Fig. 2-5(b)). Extending this idea further (Fig. 2-5(c)-(e)), if we chain all 100 examples as a single sequence, the entire sequence is stored using a network with only 1 hidden layer and 512 hidden units.

Increasing Depth and Width leads to more Attractors/Limit Cycles. The experiments in Fig. 2-5 indicate that increasing network depth and width leads to an increase in the number of training examples / sequences stored as attractors / limit cycles. For over-parameterized autoencoders, this implies that the maximum eigenvalue of the Jacobian is less than 1 for a greater number of training examples upon increasing network depth and width (see Proposition 1), i.e., the network becomes more *contractive* around the training examples. Indeed, by analyzing the histogram of the maximum eigenvalue of the Jacobian at each of the training examples, we observed that as network depth and width increases, the mode of these histograms shifts closer to zero (Appendix Fig. A-7). Additionally, when considering the distribution of the top 1% of Jacobian eigenvalues, we find that as network width increases, the variance of the distribution of Jacobian eigenvalues decreases, and when depth increases, the mode of the distribution shifts closer to zero (Appendix Fig. A-8). In the following, we prove this phenomenon for a single training example, i.e., we prove that autoencoders trained on a single example become more contractive at the training example with increasing depth and width.

2.4 Theoretical Analysis of Special Cases

We now provide theoretical support for our empirical findings. Complete proofs are given in Appendix B-E.

Proof that when trained on a single example, over-parameterized autoencoders store the example as an attractor. We outline the proof for the 1-hidden layer setting. The complete proof for the multi-layer setting is given in the Appendix C.

Let $f(z) = W_1\phi(W_2z)$ represent a 1-hidden layer autoencoder with elementwise nonlinearity ϕ and weights $W_1 \in \mathbb{R}^{k_0 \times k}$ and $W_2 \in \mathbb{R}^{k \times k_0}$, applied to $z \in \mathbb{R}^{k_0}$. We analyze the function learned by gradient descent with learning rate γ by minimizing the following autoencoding loss on 1 training example x :

$$\mathcal{L}(x, f) = \frac{1}{2} \|x - f(x)\|_2^2. \quad (2.3)$$

Let $W_1^{(t)}, W_2^{(t)}$ denote the values of the weights after t steps of gradient descent. To prove that x is an attractor of f after training, we solve for $W_1^{(\infty)}, W_2^{(\infty)}$ and compute the top eigenvalue of the Jacobian of f

at x (denoted $\lambda_1(\mathbf{J}(f(x)))$).

In order to solve for W_1, W_2 , we first identify two invariants of gradient descent (proved in Appendix B):

Invariant 1: If W_1 and W_2 are initialized to be rank 1 matrices of the form $xu^{(0)T}$ and $v^{(0)}x^T$ respectively, then $W_1^{(t)} = xu^{(t)T}$ and $W_2^{(t)} = v^{(t)}x^T$ for all time-steps $t > 0$.

Invariant 2: If, in addition, all weights in each row of W_1 and W_2 are initialized to be equal, they remain equal throughout training.

Invariant 1 implies that autoencoders trained on 1 example produce outputs that are multiples of the training example. Generalizing this result, in Appendix D, we prove that autoencoders trained on multiple examples produce outputs in the span of the training data. Invariant 2 reduces gradient descent dynamics to the 1-dimensional setting. Using the Invariants 1 and 2 combined with gradient flow (i.e. taking the limit as the learning rate $\gamma \rightarrow 0$), we can solve for $W_1^{(\infty)}$ and $W_2^{(\infty)}$.

Theorem 1. *Let $f(z) = W_1\phi(W_2z)$ denote a 1-hidden layer network with elementwise nonlinearity ϕ and weights $W_1 \in \mathbb{R}^{k_0 \times k}$ and $W_2 \in \mathbb{R}^{k \times k_0}$, applied to $z \in \mathbb{R}^{k_0}$. Let $x \in \mathbb{R}^{k_0}$ be a training example with $\|x\|_2 = 1$. Assuming $\frac{\phi(z)}{\phi'(z)} < \infty \forall z \in \mathbb{R}$, then under Invariants 1 and 2, gradient descent with learning rate $\gamma \rightarrow 0$ applied to minimize the autoencoding loss in Eq. (2.3) leads to a rank 1 solution $W_1^{(\infty)} = xu^T$ and $W_2^{(\infty)} = vx^T$ with $u, v \in \mathbb{R}^k$ satisfying:*

$$\frac{u_i^2 - u_i^{(0)2}}{2} = \int_{v_i^{(0)}}^{v_i} \frac{\phi(z)}{\phi'(z)} dz \quad \text{and} \quad u_i\phi(v_i) = \frac{1}{k},$$

and $u_i = u_j, v_i = v_j$ for all $i, j \in [k]$, where $u^{(0)}$ and $v^{(0)}$ are such that $W_1^{(0)} = xu^{(0)T}$ and $W_2^{(0)} = v^{(0)}x^T$.

Theorem 1 allows us to compute the top eigenvalue of the Jacobian at x , denoted by $\lambda_1(\mathbf{J}(f(x)))$.

Theorem 2. *Under the setting of Theorem 1, it holds that*

$$\lambda_1(\mathbf{J}(f(x))) = \frac{\phi'(v_i)v_i}{\phi(v_i)}.$$

Using Theorem 2, we can explicitly determine whether a training example x is an attractor, when given a nonlinearity ϕ , initial values for $u^{(0)}$ and $v^{(0)}$, and the width of the network k . We note that for all non-piecewise nonlinearities used thus far, we can make any training example an attractor by selecting values for $u^{(0)}$, $v^{(0)}$ and k appropriately.

Example. Let x be a training example in \mathbb{R}^{k_0} . Suppose $\phi(z) = \frac{1}{1+e^{-z}}$ for $z \in \mathbb{R}$, $k = 2$, and $u_i^{(0)} = v_i^{(0)} = 1$ for all i . Then by Theorems 1 and 2, it holds after training that

$$\frac{u_i^2 - 1}{2} = \int_1^{v_i} \left(\frac{1}{1 - \phi(z)} \right) dz \quad \text{and} \quad \frac{u_i}{1 + e^{-v_i}} = \frac{1}{2}$$

with $u_i \approx .697$, $v_i \approx .929$ and $\lambda_1(\mathbf{J}(f(x))) \approx .263$. Since $\lambda_1(\mathbf{J}(f(x))) < 1$, x is an attractor. We also confirmed this result (up to third decimal place) experimentally by training a network using gradient descent with learning rate 10^{-4} .

Importantly, the analysis of Theorem 2 implies that attractors arise as a result of training and are not simply a consequence of interpolation by a neural network with a certain architecture; see the following corollary.

Corollary 1. *Let $x \in \mathbb{R}^{k_0}$ with $\|x\|_2 = 1$ and $f(z) = xu^T\phi(vx^Tz)$, where $u, v \in \mathbb{R}^k$ and ϕ is a smooth element-wise nonlinearity with $\frac{\phi'(z)}{\phi(z)} < \infty$ for all $z \in \mathbb{R}$, $\left| \frac{\phi'(z)z}{\phi(z)} \right| > 1$ for z in an open interval $\mathcal{O} \subset \mathbb{R}$. Then there exist infinitely many $v \in \mathbb{R}^k$, such that $f(x) = x$ and x is not an attractor for f .*

The condition, $|\phi'(z)z/\phi(z)| > 1$ for z in an open interval, holds for all smooth non-linearities considered in this paper. The proof is presented in Appendix B.

We note that while the linear setting with $\phi(z) = z$ has been studied extensively using gradient flow [6, 7, 72], our results extend to the non-linear setting and require novel tools.

Remarks on the Multiple Sample Setting. While we extend Invariant 1 to the multiple example setting in Appendix D, a similar extension of Invariant 2 is required in order to generalize Theorem 1 to multiple examples. We believe such an extension may be possible for orthonormal training examples. Under random initialization, it may be possible to prove the attractor phenomenon by analyzing autoencoders in the Neural Tangent Kernel (NTK) regime [96]. However, the disadvantage of such an analysis is that it relies on computing a closed form for the NTK in the limiting case of network width approaching infinity. On the other hand, Theorem 1 holds for a general class of non-linearities and for finite width and depth.

Remarks on Similarity to Power Iteration. The attractor phenomenon identified in this work appears similar to that of Fast Independent Component Analysis [92] or more general nonlinear power iteration [27], where every ‘‘eigenvector’’ (corresponding to a training example in our setting) of a certain iterative map has its own basin of attraction. In particular, increasing network depth may play a similar role to increasing the number of iterations in those methods. While the mechanism may be different, understanding this connection is an important direction for future work.

Proof that sequence encoding provides a more efficient mechanism for memory than autoencoding by analyzing sequence encoders as a composition of maps. We start by generalizing Invariants 1, 2, and Theorem 1 to the case of training a network to map an example $x^{(i)} \in \mathbb{R}^{k_0}$ to an example $x^{(i+1)} \in \mathbb{R}^{k_0}$ as follows.

Theorem 3. *Let $f(z) = W_1\phi(W_2z)$ denote a 1-hidden layer network with elementwise nonlinearity ϕ and weights $W_1 \in \mathbb{R}^{k_0 \times k}$ and $W_2 \in \mathbb{R}^{k \times k_0}$, applied to $z \in \mathbb{R}^{k_0}$. Let $x^{(i)}, x^{(i+1)} \in \mathbb{R}^{k_0}$ be training examples with $\|x^{(i)}\|_2 = \|x^{(i+1)}\|_2 = 1$. Assuming that $\frac{\phi(z)}{\phi'(z)} < \infty \forall z \in \mathbb{R}$ and there exist $u^{(0)}, v^{(0)} \in \mathbb{R}^k$ such that $W_1^{(0)} = x^{(i+1)}u^{(0)T}$ and $W_2^{(0)} = v^{(0)}x^{(i)T}$ with $u_i^{(0)} = u_j^{(0)}, v_i^{(0)} = v_j^{(0)} \forall i, j \in [k]$, then gradient descent with learning rate $\gamma \rightarrow 0$ applied to minimize*

$$\mathcal{L}(x, f) = \frac{1}{2} \|x^{(i+1)} - f(x^{(i)})\|_2^2 \quad (2.4)$$

leads to a rank 1 solution $W_1^{(\infty)} = x^{(i+1)}u^T$ and $W_2^{(\infty)} = vx^{(i)T}$ with $u, v \in \mathbb{R}^k$ satisfying

$$\frac{u_i^2 - u_i^{(0)2}}{2} = \int_{v_i^{(0)}}^{v_i} \frac{\phi(z)}{\phi'(z)} dz, \quad \text{and} \quad u_i \phi(v_i) = \frac{1}{k},$$

and $u_i = u_j, v_i = v_j$ for all $i, j \in [k]$.

The proof is analogous to that of Theorem 1. Sequence encoding can be viewed as a composition of individual networks f_i that are trained to map example $x^{(i)}$ to example $x^{((i \bmod n)+1)}$. The following theorem provides a sufficient condition for when the composition of these individual networks stores the sequence of training examples $\{x^{(i)}\}_{i=1}^n$ as a stable limit cycle.

Theorem 4. *Let $\{x^{(i)}\}_{i=1}^n$ be n training examples with $\|x^{(i)}\|_2 = 1$ for all $i \in [n]$, and let $\{f_i\}_{i=1}^n$ denote n 1-hidden layer networks satisfying the assumptions in Theorem 3 and trained on the loss in Eq. (2.4). Then the composition $f = f_n \circ f_{n-1} \circ \dots \circ f_1$ satisfies:*

$$\lambda_1(\mathbf{J}(f(x^{(1)}))) = \prod_{i=1}^n \left(\frac{\phi'(v_j^{(i)})v_j^{(i)}}{\phi(v_j^{(i)})} \right). \quad (2.5)$$

The proof is presented in Appendix E. Theorem 4 shows that sequence encoding provides a more efficient mechanism for memory than autoencoding. If each of the networks f_i autoencoded example x_i for $i \in [n]$, then Theorem 2 implies that each of the n training examples is an attractor (and thus recoverable) if each term in the product in Eq. (2.5) is less than 1. This in turn implies that the product, itself, is less than 1 and hence all training examples are stored by the corresponding sequence encoder, f , as a stable limit cycle.

2.5 Discussion

We have shown that standard over-parameterized neural networks trained using standard optimization methods implement associative memory. In particular, we empirically showed that autoencoders store training examples as attractors and that sequence encoders store training sequences as stable limit cycles. We then demonstrated that sequence encoders provide a more efficient mechanism for memorization and retrieval of data than autoencoders. In addition, we mathematically proved that when trained on a single example, nonlinear fully connected autoencoders store the example as an attractor. By modeling sequence encoders as a composition of maps, we showed that such encoders provide a more efficient mechanism for implementing memory than autoencoders, a finding which fits with our empirical evidence. We end by discussing implications and possible future extensions of our results.

Inductive Biases. In the over-parameterized regime, neural networks can fit the training data exactly for different values of parameters. In general, such interpolating auto-encoders do not store data as attractors (Corollary 1). Yet, as we showed in this paper, this is typically the case for parameter values chosen by gradient-based optimization methods. Thus, our work identifies a novel *inductive bias* of the specific solutions selected by the training procedure. Furthermore, increasing depth and width leads to networks becoming more *contractive* around the training examples, as demonstrated in Figure 2-4.

While our paper concentrates on the question of implementing associative memory, we employ the same training procedures and similar network architectures to those used in standard supervised learning tasks. We believe that our finding on the existence and ubiquity of attractors in these maps may shed light on the important question of inductive biases in interpolating neural networks for classification [22].

Generalization. While generalization in autoencoding often refers to the ability of a trained autoencoder to reconstruct test data with low error [209], this notion of generalization may be problematic for the following reason. The identity function achieves zero test error and thus “generalizes”, although no training is required for implementing this function. In general, it is unclear how to formalize generalization for autoencoding and alternate notions of generalization may better capture the desired properties. An alternative definition of generalization is the ability of an autoencoder to map corrupted versions of training examples back to their originals (as in Fig. 2-2a,b). Under this definition, over-parameterized autoencoders storing training examples as attractors generalize (Fig. 2-2c), while the identity function does not generalize. Given this issue with the current notion of generalization for autoencoding, it is an important line of future work to provide a definition of generalization that appropriately captures desired properties of trained autoencoders. Lastly, another important direction of future work is to build on the properties of autoencoders and sequence encoders identified in this work to understand generalization properties of networks used for classification and regression.

Metrics Used by Nonlinear Networks. In Figure 2-3, we provided a visualization of how the basins of attraction for individual training examples subdivide the space of inputs. The picture appears very different from the Voronoi tessellation corresponding to the 1-nearest-neighbor (1-NN) predictor, where each input is associated to its closest training point in Euclidean distance. Yet, this may be different in high dimension. In Appendix Fig A-9, we compare the recovery rate of our network from Figure 2-2 to that of a 1-NN classifier and observe remarkable similarity, leading us to conjecture that the basins of attraction of high-dimensional fully connected neural networks may be closely related to the tessellations produced by 1-NN predictors. Thus, understanding the geometry of attractors in high-dimensional neural networks is an important direction of future research.

Connection to Biological Systems. Finally, another avenue for future exploration (and a key motivation for the original work on Hopfield networks [85]) is the connection of autoencoding and sequence encoding in neural nets to memory mechanisms in biological systems. Since over-parameterized autoencoders and sequence encoders recover stored patterns via iteration, the retrieval mechanism presented here is biologically plausible. However, back-propagation is not believed to be a biologically plausible mechanism for storing patterns [71]. An interesting avenue for future research is to identify storage mechanisms that are biologically plausible and to see whether similar attractor phenomena arise in other, more biologically plausible, optimization methods.

Materials and Methods. An overview of all experimental details including datasets, network architectures,

initialization schemes, random seeds, and training hyperparameters considered in this work are provided in Appendix Fig. A-1, A-10, and A-11. Briefly, we used the PyTorch library [142] and two NVIDIA Titan Xp GPUs for training all neural networks. In our autoencoding experiments on the image datasets ImageNet-64 [141], CIFAR10 [105], and MNIST [107], we trained both, fully connected networks and U-Net convolutional networks [161]. For Fig 2-3, 2-4b, 2-4c, and 2-5 as well as for training sequence encoder models on audio and video samples³, we used fully connected networks. For all these experiments we used the Adam optimizer with a learning rate of 10^{-4} until the mean squared error dropped below 10^{-8} . For Appendix Fig. A-10 and A-11, we fixed the architecture width and depth while varying the initialization scheme, optimization method, and activation function.

³Link to video and audio samples: https://github.com/uhlerlab/neural_networks_associative_memory

Chapter 3

Infinitely wide neural networks for matrix completion

Having characterized the benefit of over-parameterization in unsupervised learning tasks, we now showcase the practical value of using infinitely large over-parameterized models. Namely, we derive the NTK corresponding to infinitely wide networks used for matrix completion tasks such as virtual drug screening and image inpainting. We demonstrate that the NTK of infinitely wide fully connected networks leads to state-of-the-art performance for virtual drug screening and that the convolutional NTK (CNTK) of infinitely wide convolutional networks leads to competitive performance for high resolution image inpainting. The work presented in this section culminated into the following paper [154].

3.1 Introduction

Matrix completion is a fundamental problem in machine learning, arising in a variety of applications including virtual drug screening and image inpainting. Given a matrix Y with only a subset of coordinates observed, the goal of matrix completion is to impute the unobserved entries in Y . For example, in collaborative filtering (Fig. 3-1a), matrix completion is used to infer the interests of a user from the interests of other users. A prominent example is the Netflix challenge of inferring movie preferences from sparsely-populated matrices of user ratings [1]. For virtual drug screening (Fig. 3-1b), matrix completion is used to predict the effect of a drug on a cell type/state given other drug and cell type/state combinations. For image inpainting (Fig. 3-1c) and image reconstruction (Fig. 3-1d), matrix completion is used to restore missing pixels in a corrupted image.

Standard approaches to matrix completion such as nuclear norm minimization [37, 38, 158] or deep matrix factorization [8] aim for a completion that yields a low rank matrix. While such methods can be effective in applications like collaborative filtering, where low rank can capture user similarity, such an objective function can lead to ineffective solutions for applications including drug response imputation, image inpainting, or image reconstruction. For example, in the case of drug response imputation, imputing a new drug would involve predicting the values of an entirely-missing vector of gene responses (in contrast to the aforementioned Netflix problem, which involves imputing single scalar entries of the matrix). In this case, a low-rank reconstruction would replace all missing entries with a fixed constant, thereby leading to poor predictive performance. Similarly, for image inpainting and reconstruction, a low rank completion is generally ineffective since it does not take into account local image structure [115, 201]. Thus, there is a need for a more general approach to matrix completion that can easily adapt to the structures in different applications.

In this work, we provide a simple, fast, and flexible framework for matrix completion. To accomplish this, we view matrix completion as an inverse problem; given a matrix $Y \in \mathbb{R}^{m \times n}$ such that a subset of coordinates $S = \{(i, j)\} \subset [m] \times [n]$ are observed and the other entries are missing, we aim to construct $\hat{Y} \in \mathbb{R}^{m \times n}$ such that $\hat{Y}_{i,j} \approx Y_{i,j}$ for all observed coordinates $(i, j) \in S$. We use neural networks to model the

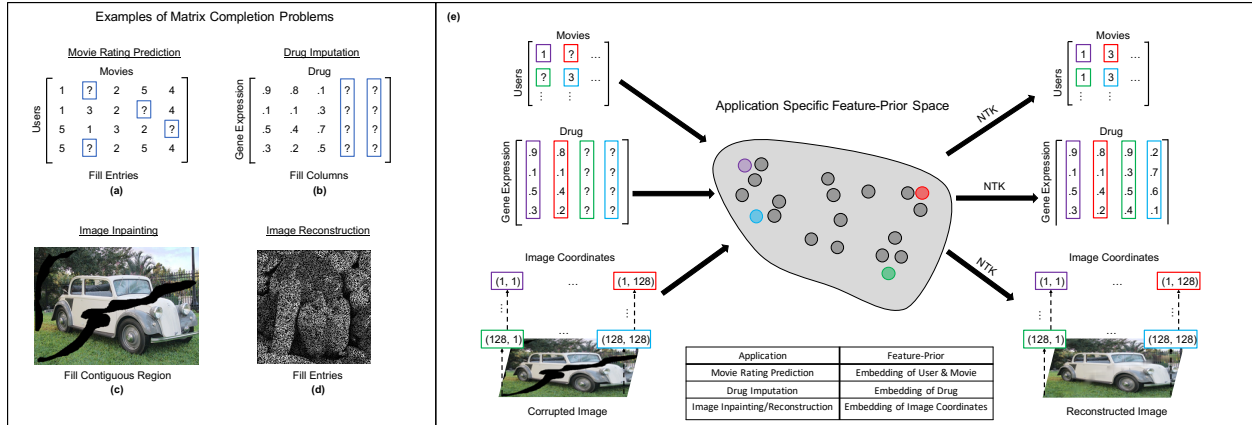


Figure 3-1: An overview of matrix completion applications where ?’s in (a), (b) and zero (black) pixels in (c), (d) represent unobserved entries. (a) Collaborative filtering example (the Netflix problem), where the goal is to predict how a user would rate (on a scale of 1-5) an unseen movie. (b) Virtual drug screening, where the problem is to predict the gene expression profile for an unobserved drug / cell type combination. In this application entire columns are unobserved. (c,d) Image inpainting and reconstruction involves reconstructing a corrupted region of an image (shown as black pixels). (e) Our NTK matrix completion framework is easily adapted to solve all of the above problems by selecting a feature prior that represents an embedding of application specific metadata.

observations in Y and use gradient descent to minimize:

$$\mathcal{L}(\mathbf{W}) = \sum_{(i,j) \in S} (Y_{i,j} - [W_d \phi(W_{d-1} \phi(\dots W_2 \phi(W_1 Z) \dots)])_{i,j})^2, \quad (3.1)$$

where $\mathbf{W} = \{W_\ell\}_{\ell=1}^d$ are the weights of a neural network with each $W_\ell \in \mathbb{R}^{k_{\ell+1} \times k_\ell}$ and $k_{d+1} = m$, $k_1 = p$; $\phi: \mathbb{R} \rightarrow \mathbb{R}$ is a fixed element-wise nonlinearity; and $Z \in \mathbb{R}^{p \times n}$ is a fixed application-dependent matrix, which we call the feature prior (described in detail below). The completed matrix \hat{Y} is then obtained using the forward model with the trained weights, i.e., $\hat{Y} = W_d \phi(W_{d-1} \phi(\dots W_2 \phi(W_1 Z) \dots))$. The main contribution of this work is showing that minimizing the loss in Eq. [3.1] when the width $\{k_\ell\}_{\ell=2}^d$ of the neural network tends to infinity, gives rise to a simple, fast, and flexible framework for matrix completion suitable for a range of applications.

Superficially, the formulation in Eq. [3.1] appears similar to that of traditional supervised learning, where a neural network is trained to map data (which would correspond to Z in our formulation) to corresponding labels Y . However, it is important to note that in our formulation Z can be independent of the observations Y (Z could for example be the identity matrix or a random matrix). Thus, Z should be interpreted as a prior that can be chosen in an application-dependent manner. We will discuss the effect of this prior as well as how to choose it for very different applications like virtual drug screening and image inpainting.

Simple and Fast Algorithm for Matrix Completion through Infinite Width Networks. A trend for improving neural network performance is to make models larger (in multiple respects) [80, 100, 161, 206]. Underscoring this trend, several recent works have empirically demonstrated the advantage of larger (in particular, wider) networks with respect to generalization and performance for classification and representation learning tasks [22, 135, 152, 208]. There is also an emerging theoretical understanding of the benefit of larger models [16, 24, 75]. The extreme case where network width approaches infinity, is what we consider in this paper in the setting of matrix completion.

While generally larger neural networks require more computational resources for training, quite counter-intuitively, the limit as network width approaches infinity may yield computational savings. Namely, it was recently shown that training infinite width networks is equivalent to solving kernel regression with a particular kernel known as the neural tangent kernel (NTK) [96]. For fully connected networks, the NTK can

be computed efficiently in closed form [96], and thus training an infinite width network reduces to solving a linear system. While this may still be computationally expensive when the number of examples is large, we will use recent pre-conditioner methods [124, 125, 128] to overcome this limitation.

For convolutional networks no efficient computation of the NTK (the so-called CNTK) has been known [9, 42, 179]. A major contribution of this work is to provide a memory and runtime efficient algorithm for computing the exact CNTK for matrix completion for a class of practical neural network architectures. As a consequence, our framework can be used to inpaint or reconstruct high-resolution images with hundreds of thousands of pixels. We also provide software for constructing the CNTK as well as pre-computed kernels. The simplicity and speed of our framework is exhibited by the fact that most of the results in this work require only a CPU and can be run efficiently on a laptop.

Flexibility through Feature Prior. The matrix Z in Eq. [3.1] is key to making our framework easily adaptable to different applications. Unlike traditional supervised learning where the goal is to learn a mapping from data X to labels Y , the matrix Z in our framework can be independent of the observations in Y . We refer to Z as a **feature prior** since, as we will see, by minimizing the loss in Eq. [3.1], the entries of Z encode structure between the coordinates of Y (see Fig. 3-1e).

We will demonstrate the flexibility of our framework by using it in two very different applications, namely for drug response imputation and image inpainting/reconstruction. For drug response imputation, we will select feature priors that encode information about cell and drug type combinations. For image inpainting and reconstruction, we will select feature priors that encode information about image coordinates. In addition to being flexible, we will show that our approach is competitive in terms of speed and accuracy with prior approaches that were specifically developed for drug response imputation [84, 121] or image inpainting/reconstruction [51, 185, 187].

3.2 Matrix Completion with the NTK

In this section, we derive the NTK for matrix completion when using fully connected networks. Our derivation provides a principled method for selecting the feature prior, Z ; namely, we will show that Z should be an embedding of *coordinate metadata*, i.e. information describing the coordinates of Y . For example in drug response imputation, each column of Z could correspond to a different drug and two columns of Z should be similar if the drug metadata is similar (e.g. the molecular structures are similar). The resulting method is then equivalent to performing semi-supervised learning to map from the columns of Z to observed entries in each row of Y . In Section 3.3, we will utilize this theoretical result to select an effective feature prior for virtual drug screening.

Since the NTK forms the backbone of our framework, we start with the definition of the NTK [96] and briefly review how solving kernel regression with the NTK connects to training infinitely wide neural networks.

Definition 7 (NTK). Let $f(w; x) : \mathbb{R}^p \times \mathbb{R}^d \rightarrow \mathbb{R}$ denote a neural network with parameters w . The corresponding **neural tangent kernel**, $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, is a symmetric, continuous, positive definite function given by:

$$K(x, x') = \langle \nabla_w f(w^{(0)}; x), \nabla_w f(w^{(0)}; x') \rangle,$$

where $w^{(0)} \in \mathbb{R}^p$ are the network parameters at initialization.

For a review of kernel regression and kernel functions see [166]. Given training data $(x^{(i)}, y^{(i)}) \in \mathbb{R}^d \times \mathbb{R}$ for $i = 1, \dots, n$, solving kernel regression with the NTK involves minimizing the loss:

$$\mathcal{L}(\alpha) = \|y - \alpha \hat{K}\|_2^2, \tag{3.2}$$

where $\alpha \in \mathbb{R}^{1 \times n}$, $y = [y^{(1)}, \dots, y^{(n)}]^T$, and $\hat{K} \in \mathbb{R}^{n \times n}$ with $\hat{K}_{i,j} = K(x^{(i)}, x^{(j)})$. The work of [96] established that using kernel regression with the NTK is equivalent (under mild assumptions) to training a neural network to map $x^{(i)}$ to $y^{(i)}$ using the mean squared error, in the limit as the network width tends to infinity.

Throughout this work, we will assume that $w_i^{(0)} \stackrel{i.i.d}{\sim} \mathcal{N}(0,1)$ and that the nonlinearity ϕ in Eq. [3.1] is homogeneous (which includes, for example, the rectified linear unit (ReLU), a widely used nonlinearity) so that the NTK corresponding to a fully connected network can be computed efficiently in closed form [43, 96, 182].

Feature Prior Provides a Flexible Approach for Matrix Completion through Connection with Semi-supervised Learning

A natural approach for imputing missing entries in a matrix, Y , is to first obtain an embedding of the coordinates of Y (e.g. a map from coordinates (i, j) to \mathbb{R}^p) and then learn a map from the coordinate embedding to the observed entries in Y (e.g. a map from \mathbb{R}^p to $Y_{i,j} \in \mathbb{R}$); see also [3, Ch.1]. For example, for virtual drug screening, one could first embed the drugs based on their molecular properties and then learn a map from this embedding to the measured output, such as gene expression. Such an approach in which a map is learned from an embedding to the observed samples is referred to as *semi-supervised learning* [69, Ch.15]. In this section, we will prove that minimizing the loss in Eq. [3.1] is equivalent to using a semi-supervised learning approach for matrix completion. Namely, we show that the columns of Z represent an embedding of the coordinates of Y and that the NTK is used to map from the columns of Z to the entries in Y .

It is a priori unclear how to compute the NTK for matrix completion, since this requires training examples and labels. For this, we note the following equivalent formulation of Eq. [3.1]:

$$\begin{aligned} \mathcal{L}(\mathbf{W}) &= \sum_{(i,j) \in \mathcal{S}} (Y_{i,j} - \langle f_Z(\mathbf{W}), M_{\{(i,j)\}} \rangle)^2, \\ f_Z(\mathbf{W}) &= W^{(d)} C_d \phi(W^{(d-1)} C_{d-1} \phi(\dots W^{(2)} C_2 \phi(W^{(1)} Z) \dots), \end{aligned} \quad (3.3)$$

where $C_\ell = c/\sqrt{k_\ell}$ for a constant c , $\langle A, B \rangle = \text{tr}(A^T B)$ denotes the trace inner product, and $M_{\{(i,j)\}}$ is an indicator matrix, i.e., it has a 1 in the (i, j) entry and zeros everywhere else. To ease notation, we will use M_{ij} to denote the indicator matrix $M_{\{(i,j)\}}$. The formulation in Eq. [3.3] shows that we can view matrix completion as a problem where the "training examples" are indicator matrices M_{ij} and the "labels" are the corresponding entries $Y_{i,j}$. This reformulation yields the following closed form for the NTK for matrix completion, where $\check{\phi} : [-1, 1] \rightarrow \mathbb{R}$ denotes the dual activation function [53] to ϕ . To keep notation simple, we here provide the theorem when ϕ is the ReLU activation function, but this result holds generally for homogeneous nonlinearities; see SI Appendix A.

Theorem 5. *Assume $Z = \{z^{(i)}\}_{i=1}^n \in \mathbb{R}^{p \times n}$, where each column is normalized with $\|z^{(i)}\|_2 = 1$. Let $f_Z(\mathbf{W})$ be a d layer fully connected network with nonlinearity $\phi(x) = \max(x, 0)$ and $c = \sqrt{2}$ in Eq. [3.3]. Then, as widths $k_2, k_3, \dots, k_d \rightarrow \infty$, the NTK for matrix completion with $f_Z(\mathbf{W})$ is given by*

$$K(M_{ij}, M_{i'j'}) = \begin{cases} \kappa_d(z^{(j)T} z^{(j')}) & \text{if } i = i' \\ 0 & \text{if } i \neq i' \end{cases},$$

where $\kappa_d(\xi) = \check{\phi}^{(d)}(\xi) + \kappa_{d-1}(\xi) \frac{d\check{\phi}}{d\xi}(\check{\phi}^{(d-1)}(\xi))$, and $\check{\phi}^{(h)}(\xi) = \check{\phi}(\check{\phi}^{(h-1)}(\xi))$ for $h \geq 1$ and $\check{\phi}^{(0)}(\xi) = \xi$.

The proof as well as an example showing how Theorem 5 can be used in practice to compute the NTK for matrix completion is presented in SI Appendix A. Since the kernel value between M_{ij} and $M_{i'j'}$ is a function of columns j and j' of Z , Theorem 5 implies that the NTK for matrix completion maps columns of Z to entries $Y_{i,j}$, and thus the columns of Z encode structure between the coordinates of Y .

By varying the nonlinearity ϕ , depth d , and feature prior Z , our framework encapsulates a variety of semi-supervised learning approaches. To provide a non-trivial example, we prove in SI Appendix A that our framework for matrix completion generalizes Laplacian-based semi-supervised learning [26]. This insight regarding the connection between our framework for matrix completion and semi-supervised learning represents the backbone for a simple and competitive approach to virtual drug screening described in the next section.

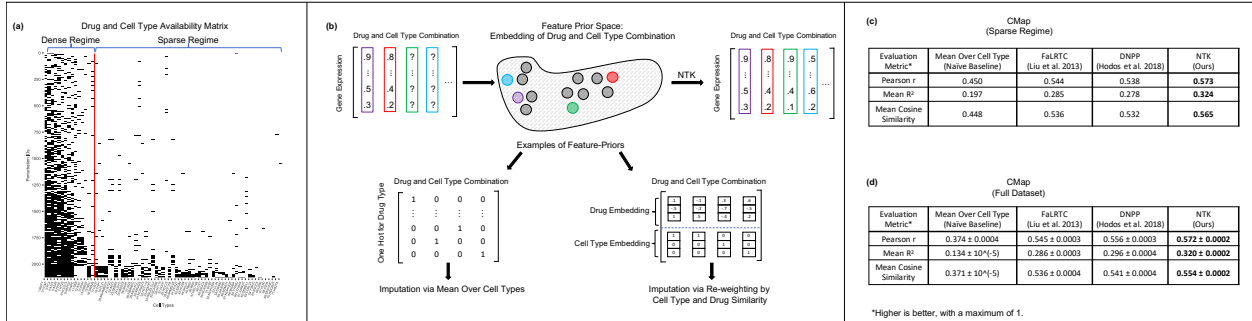


Figure 3-2: Our infinite width neural network framework outperforms DNPP [84], FaLRTC [121], and mean over cell types for drug response imputation on CMap. (a) We visualize the availability of cell type and drug combinations of the subset from [84]. (b) Our method corresponds to first providing an embedding of cell type and drug combinations as the feature prior and then applying the NTK. We show that: (1) using a feature prior consisting of one-hot vectors for drugs corresponds to imputation by performing mean across observations for each cell type and (2) using a feature prior that captures similarity between drugs and cell types is effective for imputation. (c, d) Our infinite width neural network framework (denoted NTK) outperforms DNPP and mean over cell type across three evaluation metrics. We use 5 rounds of 10-fold cross validation to determine that the difference between our method and the next best method, DNPP, is statistically significant (p-value less than 10^{-20}).

3.3 Virtual Drug Screening with the NTK

CMAP is a prominent, large-scale, publicly available drug screen that considers 20,413 different compounds and 72 different cell lines [178]. Experiments in CMAP were performed on a subset of 201,484 drug/cell line pairs; for each of these pairs the gene expression profile of 978 landmark genes was measured. CMAP has been an important resource for computational approaches to drug discovery and drug repurposing [30, 150, 178]. In these applications, the goal is to use a subset of observed drug/cell type pairs to predict the gene expression profile of new drug/cell type pairs. These profiles are then used to identify drug candidates of interest that can be tested experimentally [106, 194].

The CMAP dataset can be viewed as a 3-dimensional tensor (drugs, cell lines, genes), where many of the entries are missing. In the following, we will use the same pre-processing of the data as in [84] to filter out drug/cell line combinations with very few or inconsistent samples; a description and a link to the dataset is provided in SI Appendix B. The resulting drug/cell line combinations are shown in Fig. 3-2a. The 3-dimensional tensor can be flattened into a matrix, where the columns correspond to drug/cell line combinations and the rows represent genes (see Fig. 3-2b); i.e., following the notation from Section 2, entry Y_{ij} of the resulting flattened matrix is a real-valued number quantifying the gene expression of gene i in drug and cell type combination j . This matrix has a missing column for every missing drug/cell line combination. Classical low rank matrix factorization methods would prove ineffective in this setting since they would replace each missing column by the same constant column. On the other hand, Theorem 5 suggests the NTK as an effective way for imputing the missing gene expression profiles by selecting the feature prior Z such that two columns of Z are similar if they correspond to similar drug/cell line pairs. In the following, we discuss three different feature priors for this application; for a full description of these priors see SI Appendix C.

Feature Prior corresponding to the Mean Over Cell Type Baseline

A simple baseline is to impute the gene expression profiles for each missing drug for a given cell line by the mean over all observed drugs for this cell line. Quite surprisingly, this simple approach gives rise to a strong baseline [84, 175], since cell type is the dominant factor, while drugs have subtle effects on gene expression.

While it is generally nontrivial to improve upon this simple baseline without constructing a specialized algorithm [19, 84, 95, 146], our NTK framework provides an easy way for doing so. In particular, our framework makes it evident that the feature prior corresponding to the mean over cell type baseline is trivial,

since it corresponds to an embedding in which drugs are encoded via one-hot vectors (see SI Appendix D). Thus, to improve upon this baseline, we select any feature prior that can capture similarities between drugs.

Feature Prior Corresponding to Previous Algorithms

We now demonstrate that our framework provides a direct approach to improve on previous methods for virtual drug screening by using the output of previous methods as a feature prior in our framework. Namely, if a method is used to produce an imputation, \hat{Y} , then the columns in \hat{Y} should represent an embedding of drug and cell type combinations that captures their similarity. Hence, we can use $Z = \hat{Y}$ as the feature prior in our method. For illustration, we apply this approach to two state-of-the-art methods for virtual drug screening: (1) Drug Neighbor Profile Prediction (DNPP) [84], which is a weighted nearest neighbor scheme, and (2) Fast Low Rank Tensor Completion (FaLRTC) [121], which involves low rank matrix completion along each slice of the CMAP tensor. We show that our framework using these feature priors yields an improvement over the individual methods; see SI Appendix E.

Proposed Feature Prior for Drug Response Imputation

Observing the pattern of data availability in Fig. 3-2a, it is apparent that a subset of cell lines have observations for many (> 150) drugs (dense regime), while many cell lines have observations for only few (≤ 150) drugs (sparse regime). While previous methods such as DNPP are quite effective in the dense regime, they are not as effective in the sparse regime; see Fig. 3-2c and SI Appendix F. This can be explained by the fact that in the sparse regime DNPP roughly imputes using the simple mean over cell type baseline.

For effective drug response imputation in the sparse regime, our framework can be used to construct a simple feature prior by concatenating embeddings for cell types and drugs. In particular, we can use the gene expression values for a reference cell type for which there are a lot of drug observations (e.g. MCF7 in CMAP) as the embedding of drugs and the mean gene expression across all observations for a given cell type as the embedding of cell type. Fig. 3-2c shows that the NTK with this simple feature prior outperforms mean over cell type, FaLRTC and DNPP in the sparse regime. We compare across Pearson r value, mean R^2 , and mean cosine similarity. A description of all evaluation metrics is provided in SI Appendix G. By combining our feature prior for the sparse regime with the FaLRTC based feature prior for the dense regime, we obtain a drug imputation method that significantly outperforms DNPP, FaLRTC, and mean over cell type on the full dataset; see Fig. 3-2d (p-value less than 10^{-20} based on 5 rounds of 10-fold cross validation, with an improvement on every fold of every round across all metrics; see SI Appendix H).

3.4 Matrix Completion with the Convolutional NTK

While we have thus far derived and applied the NTK for matrix completion using fully connected networks, these architectures are not nearly as effective as convolutional networks for matrix completion tasks in which the target matrix is an image. Similar to the case of fully connected networks, a closed form for the NTK corresponding to convolutional networks (the so-called CNTK) is known in the regression setting [9], but it has not been considered in the setting of matrix completion. Moreover, the runtime for computing the CNTK for regression scales quadratically with each image dimension. In this section, we derive the CNTK for matrix completion and provide a computationally efficient method for computing the CNTK for matrix completion for a class of feature priors that are effective for image inpainting and reconstruction.

We begin by deriving the CNTK for matrix completion for a simple class of convolutional networks, when there are no downsampling or upsampling layers. We show that in this setting, the CNTK for matrix completion can be computed using terms from the CNTK for classification. In the following proposition (proof in SI Appendix I), $\Theta^{(d)} \in \mathbb{R}^{m \times n \times m \times n}$ denotes the tensor corresponding to the CNTK of a d layer convolutional network in the classification setting [9, Sec. 4].

Proposition 3. *Let $f_Z(\mathbf{W})$ be a d layer convolutional network used to map from feature prior, $Z \in \mathbb{R}^{c \times m \times n}$, to the target matrix, $Y \in \mathbb{R}^{m \times n}$. Then as the number of convolutional filters per layer approaches infinity,*

the CNTK of $f_Z(\mathbf{W})$ is given by:

$$K(M_{ij}, M_{i'j'}) = [\Theta^{(d)}(Z, Z)]_{i,j,i',j'}, \tag{3.4}$$

where $M_{ij}, M_{i'j'} \in \mathbb{R}^{m \times n}$ denote indicator matrices.

CNTK Performs Semi-Supervised Learning using Image Coordinate Features

In Section 3.2, we established a connection between semi-supervised learning and matrix completion using the NTK. We now establish a similar connection between semi-supervised learning and matrix completion with the CNTK for a class of feature priors defined in Theorem 6 below. This class includes feature priors that are heavily used in image inpainting applications, namely where the channels of Z are drawn i.i.d. from a stationary distribution [42, 185]. The following theorem (proof in SI Appendix J), which is analogous to Theorem 5 for the NTK, implies that using the CNTK for matrix completion is equivalent to mapping from coordinate features to observed entries in the target matrix Y .

Theorem 6. *Consider a convolutional network of depth d with homogeneous activation and in which all filters have size q and circular padding. Let $Z \in \mathbb{R}^{c \times m \times n}$ satisfy:*

$$\sum_{\ell=1}^c \sum_{-\alpha \leq a, b \leq \alpha} Z_{\ell, i+a, j+b} Z_{\ell, i'+a, j'+b} = \psi(|i-i'|, |j-j'|)$$

for some $\psi : \mathbb{R}^2 \rightarrow \mathbb{R}$ with maximum at $(0,0)$ and $\alpha = \frac{q-1}{2}$ (odd q). Then as the number of convolutional filters per layer goes to infinity, the CNTK simplifies to:

$$K(M_{ij}, M_{i'j'}) = \tilde{\psi}(|i-i'|, |j-j'|),$$

where $\tilde{\psi} : \mathbb{R}^2 \rightarrow \mathbb{R}$ is a function that can be computed from ψ (a recursive formula is provided in SI Appendix J).

Since the function $\tilde{\psi}$ depends only on the positions of the coordinates, Theorem 6 shows that the CNTK for matrix completion is equivalent to semi-supervised learning using kernels on features corresponding to coordinates.

Closed Form for the CNTK of Modern Architectures for Matrix Completion

Unlike the convolutional networks considered thus far, state-of-the-art architectures for unsupervised image inpainting such as [42, 185] incorporate a variety of layer structures including strided convolution, nearest neighbor and bilinear upsampling, skip connections, and batch normalization. We derive (in SI Appendix K) the CNTK for matrix completion using convolutional networks with the following layer structures: (1) Downsampling through Strided Convolution ; (2) Nearest Neighbor Upsampling ; and (3) Bilinear Upsampling.¹

Efficient Computation of the CNTK of Modern Architectures for Matrix Completion

A key insight that we use to speed up the computation of the CNTK is that the kernel in Eq. [3.4] depends only on the feature prior and not on the values of the observed pixels in an image. Hence, the CNTK need only be computed once for all images of a given resolution. This enables a drastic speedup over recomputing the kernel for every new image, as is currently required in classification.

However, using such a direct approach to compute the CNTK is still computationally prohibitive for high resolution images. In particular, computing the CNTK for a network with d convolutional layers to complete an image of size $2^p \times 2^q$, requires $O(p^2 q^2 d)$ runtime and $O(2^{2p+2q})$ space. In order to overcome these

¹The impact of linear downsampling and upsampling on the CNTK is briefly described in Appendix E of [179], but the explicit forms are not computed nor used in the experiments.

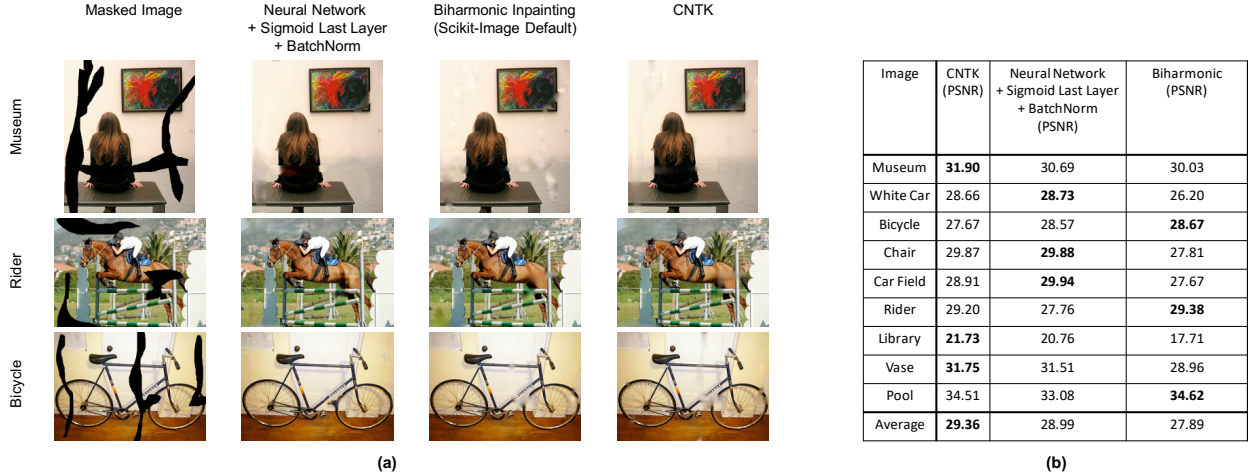


Figure 3-3: Large hole inpainting using (i) the CNTK, (ii) neural networks with sigmoid last layer and batch normalization layers that are trained with Adam, and (iii) biharmonic functions. (a) Qualitative comparison of inpainting results across the three methods. Results for all images are provided in SI Appendix Fig. S5. (b) Comparison of peak signal-to-noise ratio (PSNR) across 3 methods with the CNTK providing the highest average PSNR. Runtime and structural similarity index measure (SSIM) for the three methods are provided in SI Appendix Fig. S4.

limitations, prior work [179] used the Nyström method [193] to approximate the kernel. Instead of relying on such approximations, we here present an algorithm for computing the exact CNTK in a memory and runtime efficient manner for any convolutional neural network with circular padding, strided convolution, and nearest neighbor upsampling layers, when using a feature prior with i.i.d. random entries. Such networks and feature priors are heavily used for image completion tasks [185].

Our main insight that enables such an algorithm is that for convolutional networks with strided convolution and nearest neighbor upsampling layers, the CNTK for low resolution images can be expanded to high resolution images for any feature prior with i.i.d. random entries. In particular, if a neural network with s downsampling and upsampling layers is used to inpaint images of resolution $2^p \times 2^q$, our algorithm requires only an array of size 2^{2s+p+q} while storing the full CNTK requires an array of size 2^{2p+2q} . In practice, s is exponentially smaller than p, q and so our method is significantly more memory efficient; see the following specific example. In addition, since our method only requires computing the CNTK for images of size $2^{s+1} \times 2^{s+1}$, the runtime of our method is $O(2^{4s})$ instead of $O(2^{2p+2q})$, and thus, our method is significantly faster than a direct computation. A detailed description and proof of our expansion algorithm is presented in SI Appendix L.

Example. Let $f_Z(\mathbf{W})$ represent a convolutional neural network with circular padding, 3 layers of strided convolution with a stride size of 2 in each direction, and 3 nearest neighbor upsampling layers with a feature prior $Z \in \mathbb{R}^{c \times 512 \times 512}$ satisfying:

$$\sum_{p=1}^c Z_{p,i,j} Z_{p,i',j'} = \begin{cases} C_1 & i = i', j = j' \\ C_2 & \text{otherwise} \end{cases},$$

where $C_1, C_2 > 0$ are constants. Suppose $f_Z(\mathbf{W})$ is used to inpaint images of size 512×512 . Then, by computing the CNTK for 16×16 resolution images, $K_\ell \in \mathbb{R}^{16^2 \times 16^2}$, we can expand up to the exact CNTK for 512×512 images. Computing K_ℓ takes roughly 11 seconds when using a CPU with 1 thread and \tilde{K} uses less than 100MB of memory with floating point precision. On the other hand, even storing the true kernel $K \in \mathbb{R}^{512^2 \times 512^2}$ would require roughly 256GB memory when using floating point precision. This is twice the amount of RAM available on our server and 16 times the amount of RAM available on most laptops.

3.5 Image Inpainting and Reconstruction with the CNTK

We now utilize the results of the previous section to perform large hole image inpainting and reconstruction. As illustrated in Figs. 3-1c and 3-1d, large hole inpainting involves imputing a large contiguous region in an image while image reconstruction involves imputing random missing pixels in an image. Recent work [185] demonstrated that using convolutional neural networks with downsampling and upsampling layers to impute the missing pixels in images leads to competitive results for these applications.

The methods from [185] are a special case of our framework in Eq. [3.1]; namely using convolutional layers and letting the feature prior, Z , be a tensor with i.i.d. uniform random entries. Thus, we can use our framework for performing image completion tasks, and instead of training deep networks, we can simply solve kernel regression with the CNTK. We will demonstrate that this gives rise to a simple, fast, flexible, and competitive alternative to training deep networks for high resolution image completion problems. Moreover, we will demonstrate that our framework can be used to identify the role of architecture and feature prior on image completion problems and aid in identifying effective architectures and feature priors.

Application 1: Large Hole Inpainting with the CNTK

We utilize the CNTK for large hole inpainting tasks from [42, 185]. We compute the CNTK for the architecture used in [42] with 6 downsampling and nearest neighbor upsampling layers for the feature prior Z with i.i.d. entries $Z_{\ell,i,j} \sim U[0, .1]$, where $\ell \in \mathbb{Z}_+$ and $i, j \in [m] \times [n]$. We compute the CNTK on 128×128 resolution images and then expand it to the CNTK for high resolution images via our expansion technique in Section 3.4. We compare our method against neural networks of the same architecture using the training procedures from [42, 185] (see SI Appendix M for details). We also compare our method against inpainting with biharmonic functions [51], which is currently the default inpainting method in scikit-image [187].

Figure 3-3a shows examples of the resulting reconstructions, and Figure 3-3b shows the peak signal-to-noise ratio (PSNR) across all methods. Our method on average outperforms both inpainting with finite width neural networks and inpainting with biharmonic functions.² In SI Appendix Fig. S4, we show that our method also outperforms the other methods in terms of structural similarity index measure (SSIM), and that the runtime is comparable (within 2 minutes on average) across all methods in this setting. The reconstructions across all images and methods are provided in SI Appendix Fig. S5.

Application 2: Image Reconstruction with the CNTK

We next analyze the performance of the CNTK on the image reconstruction tasks considered in [185]. While the networks considered in [42, 185] make use of skip connections for image reconstruction, we only consider architectures without skip connections for which we can derive the CNTK exactly (see SI Appendix M for details). We again compare the CNTK to neural networks of the same architecture and to biharmonic inpainting. For this comparison, we use networks with 128 filters per layer, as is done in [42, 185]. In SI Appendix Fig. S6, we show that our model performs comparably to inpainting with biharmonic functions and outperforms neural networks of the same architecture. In SI Appendix Fig. S6, we additionally show that our method performs comparably to biharmonic inpainting in terms of SSIM and that our method is up to 10 times faster than using small width neural networks on the same hardware. While our method performs comparably to inpainting with biharmonic functions in this application, our framework is more flexible, since we can adjust architecture and feature prior, and it outperforms inpainting with biharmonic functions for the problem of large hole inpainting (see above). Since methods such as Adam with Langevin dynamics [42] have enabled performance boosts for neural networks (see SI Appendix Fig. S4 & S5), an interesting direction for future work could be to incorporate such techniques for image completion applications using the CNTK.

²While the PSNR values for these images are also presented in [42], they appear to be computed without replacement of the observed pixel values. We re-ran these experiments with replacement for fair comparison with biharmonic inpainting.

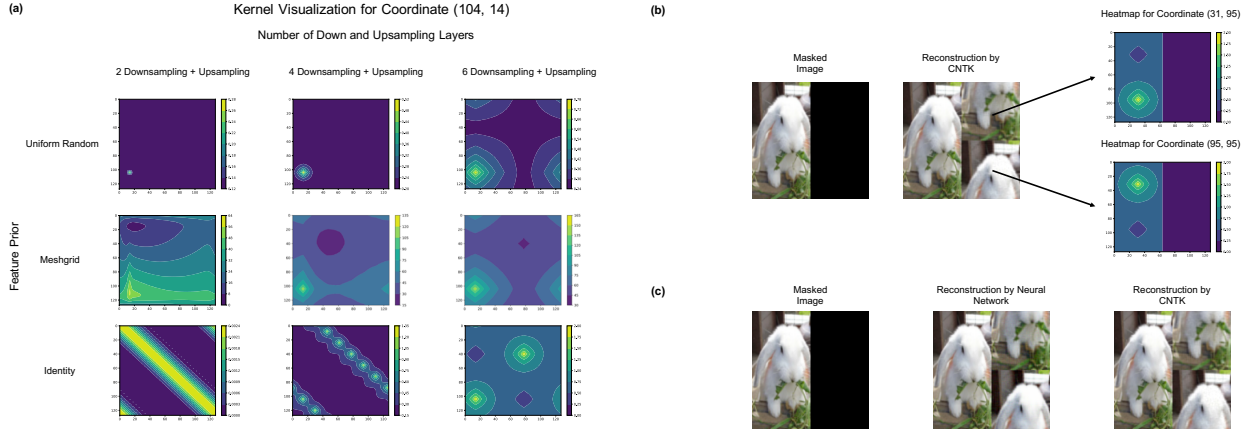


Figure 3-4: We use the CNTK to understand the impact of architecture and input on image inpainting. (a) Heatmap visualizations of the CNTK when varying the number of downsampling/upsampling layers and input. The visualization makes clear that the uniform random feature prior, unlike other feature priors, results in kernels that use the region surrounding a missing pixel value for imputation regardless of the number of downsampling layers. (b) The heatmap visualizations of the CNTK make transparent which observed pixels are being used to inpaint a given missing pixel when using the identity feature prior. (c) A comparison between inpainting a 128×128 resolution image of a rabbit with a finite width neural network and with the CNTK when the feature prior is the identity. The CNTK is able to accurately predict the unexpected behavior of the neural network.

Using Our Framework to Select Feature Prior and Architecture for Image Completion

In the following, we demonstrate that our framework provides a theoretical underpinning for understanding how a given architecture and feature prior influence image completion. In particular, we use our framework to explain why the uniform random feature prior and architectures with downsampling and upsampling layers are effective for image completion while other feature priors such as the identity feature prior are ineffective for this application.

The key observation enabling such interpretability is that for kernel methods, every prediction (a missing pixel value) is a linear combination of training examples (observed pixel values). Hence, for each imputed pixel, the CNTK can be used to provide a heatmap describing which observed pixels were most heavily weighted in the linear combination. In order to generate such heatmaps, we reshape the CNTK into a 4 dimensional tensor. Namely, given a CNTK $K \in \mathbb{R}^{mn \times mn}$, we reshape K to a tensor $K_T \in \mathbb{R}^{m \times n \times m \times n}$ where $K(M_{ij}, M_{i'j'}) = K_T(i, j, i', j')$. To generate a heatmap for a given a coordinate (i, j) , we visualize the matrix $K_T(i, j, :, :) \in \mathbb{R}^{m \times n}$. This visualization allows us to decipher how architecture and feature prior change the resulting imputation from a neural network.

The Uniform Random Feature Prior and Modern Architectures are Effective for Image Completion

In Fig. 3-4a, we visualize the kernel values $K(104, 14, :, :)$ computed for a 128×128 image when varying the number of down and upsampling layers and as well as the feature prior Z . Namely, we consider the cases where Z is the identity, the meshgrid from [185], or the uniform random tensor used in large hole inpainting experiments of [185]. A key observation is that the kernel values for the uniform random feature prior are highest around the coordinate of interest regardless of the amount of down and upsampling, which is in stark contrast to other feature priors.³ This implies that neighboring pixels are most heavily used when imputing using the uniform random feature prior (see SI Appendix Fig. S7 for additional visualizations). Moreover, when using the uniform random feature prior, the amount of down and upsampling increase (by powers of

³When there are no downsampling and upsampling layers, this follows immediately from Theorem 6.

2) the size of the region considered for imputation (see the first row of Fig. 3-4a). These heatmaps identify the minimum amount of downsampling necessary for large hole inpainting: if there is an $m \times m$ region of missing pixels ($m \geq 1$), we need least $\lfloor \log_2(m+1) \rfloor$ layers of downsampling to ensure that no pixel is filled in as an average of all other pixels. This result explains the observation from [185], which showed that using neural networks with four or fewer downsampling and upsampling layers led to worse large hole inpainting performance on images with large missing regions.

The Identity Feature Prior is Ineffective for Image Completion

The standard feature prior for matrix completion is given by choosing Z to be the identity matrix [8, 37, 72]. As shown in Fig. 3-4a, unlike the uniform random feature prior, the identity feature prior uses pixel observations from non-local regions for completion. Thus, we expect this feature prior to be ineffective for image completion tasks.

Fig. 3-4b shows the result of using the CNTK for a network with 6 downsampling and upsampling layers and the identity feature prior to impute a 128×128 rabbit image. The identity feature prior visually appears to translate observed pixels from a non-local region to perform imputation. The regions that are being translated are precisely those given by the corresponding heatmaps, e.g. the upper right quadrant is imputed using the lower left quadrant in Fig. 3-4b.

We note that our framework accurately predicts the behavior of finite width neural networks used for image inpainting. In Fig. 3-4c, we show the result of using a neural network with 6 downsampling and upsampling layers, sigmoid activation on the last layer, and identity feature prior. We observe that the neural network completes the image by translating observed pixels similarly to the imputation provided by the corresponding CNTK. This example highlights the power of using our framework for rapidly prototyping feature priors and architectures for image inpainting tasks.

3.6 Discussion

In this work, we presented a simple, fast, and flexible framework for matrix completion using the infinite width limit of neural networks, i.e. the neural tangent kernel (NTK). Below, we highlight the aspects of our framework that enable such simplicity, speed, and flexibility.

- **Simple.** Our framework is conceptually simple since we are using kernels to learn a map from features of coordinates, (i, j) , to entries in the target matrix, $Y_{i,j}$. Our framework is computationally simple since solving kernel regression involves solving a linear system of equations.
- **Fast.** Our framework is naturally fast when using the NTK of fully connected networks for matrix completion due to the simple closed form of the kernel (Theorem 5). We develop a memory and runtime efficient algorithm to compute and use the NTK of convolutional networks (the CNTK) for matrix completion (Section 3.4).
- **Flexible.** Our framework is easily adapted to various applications by the choice of the feature prior, thereby making our framework flexible. Moreover, we provided a principled approach for selecting the feature prior by establishing a connection with semi-supervised learning (Theorems 5, 6) and providing a visualization of the effect of the feature prior (Section 4).

The simplicity and speed of our framework is illustrated by the fact that many of our results (including inpainting high resolution images) can be run on a CPU and even on a laptop (see Materials & Methods for a link to our code). We demonstrated that our framework is flexible by using it to achieve competitive results for virtual drug screening (Section 3.3) and image inpainting/reconstruction (Section 3.5). We envision that our work provides a simple and accessible framework for producing strong baselines for several matrix completion applications. We conclude with a discussion of possible future extensions and applications.

Future Applications of Our Framework

In this work, we demonstrated the flexibility of our framework by constructing feature priors for two different applications, namely virtual drug screening and image completion. An interesting future direction is the

extension of our framework to other modalities such as tensors, video, or audio data. For example, by using a feature prior that captures the structure of coordinates in 3D images, we could apply our framework to impute missing regions in three-dimensional data.

Efficient Computation of the CNTK

In classification and regression settings, a major hindrance for using the CNTK in practice is the computational complexity in computing the kernel for a large image dataset. In this work, we presented an expansion technique to efficiently compute and store the exact CNTK for inpainting high resolution images, which was previously considered infeasible [42, 179]. By understanding the properties of the CNTK that make it effective for image problems, we envision that similar techniques could be applied to produce efficient kernel machines for image classification.

Developing Techniques to Improve the Performance of the NTK

While a large number of techniques such as skip connections, batch normalization, etc. have been developed to augment the performance of neural networks, such techniques have yet to be adapted to improve the performance of kernels. The simplicity and effectiveness of the NTK and CNTK based on simple architectures considered in this work motivates the development of techniques to further boost the performance of the NTK and kernel methods in general.

Materials and Methods. For solving kernel regression with the NTK, we use the direct linear system solver from [140] when the number of equations is fewer than 30,000, and we use EigenPro [124, 125] otherwise. For training neural networks, we use the PyTorch library [142]. All methods requiring a GPU are run on a single NVIDIA Titan RTX GPU. Our experiments are run on a shared server with 4 Titan RTX GPUs, 128GB CPU RAM, and 64 threads. For the virtual drug screening experiments, we use the subset of the CMap dataset [178] provided in [84]. A detailed description of all the methods (including random seeds and hyperparameters for DNPP and FaLRTC) and evaluation metrics for the virtual drug screening experiments is provided in SI Appendices C-H. A description of the t-test used for determining the significance of our results for virtual drug screening is presented in SI Appendix H. We provide code to replicate our results for the virtual drug screening experiments with the NTK, DNPP, FaLRTC, and mean over cell type at https://github.com/uhlerlab/ntk_matrix_completion. We use the codebase from [84] for performing imputation with FaLRTC.

For the image completion applications, we use the datasets from [42, 185]. For the neural network and NTK methods used in our image inpainting and reconstruction experiments, we provide a description of all architectures and training hyperparameters in SI Appendix M.

We provide a library for computing and using the CNTK for image inpainting and reconstruction applications in the codebase linked above. Our library lets the user define a custom neural network (similarly to network definitions in PyTorch), and then provides a function to compute the CNTK from the given architecture. Our method for computing the CNTK runs entirely on the CPU, and we enable parallelization across CPU threads. Our library includes functions for computing the CNTK for networks with nearest neighbor and bilinear upsampling layers, which are not readily available in the Neural Tangents library [139]. We additionally provide functions to solve kernel regression using the CNTK via a linear system solver or EigenPro. A full description of the library and an example of how to use our library for image inpainting is provided in Jupyter notebooks in our linked code. We additionally release several pre-computed kernels that can be used for high resolution inpainting and reconstruction.

Chapter 4

Consistency of infinitely wide and deep neural network classifiers

In this chapter, we demonstrate the value of the NTK in providing theoretical guarantees for neural networks. In particular, we analyze the NTK of infinitely wide and deep networks used for classification tasks. We derive a taxonomy characterizing the solutions implemented by such models based on choice of activation function. We then identify explicit activation functions for which infinitely wide and deep networks implement Bayes optimal or consistent classifiers. The work presented in this section culminated into the following paper [153].

4.1 Introduction

Deep learning has produced state-of-the-art results across several application domains including computer vision [80], natural language processing [35], and biology [183]. Despite these empirical successes, our understanding of basic theoretical properties of deep networks is far from satisfactory. In fact, for the fundamental problem of classification it has not been established whether neural networks trained with standard optimization methods can achieve consistency, i.e., whether they minimize the probability of misclassification for arbitrary data distributions (a property also referred to as *Bayes optimality* in the statistics literature).¹

There is a vast literature on the consistency of statistical machine learning methods, which has traditionally focused on methods that do not interpolate, or fit training data exactly [44, 55]. Given the recent successes of interpolating neural networks [22, 135, 208], there is renewed interest in understanding the consistency of interpolating machine learning models including nearest neighbor methods and kernel methods [23, 28, 48, 54, 155, 155]. While such methods can be universally consistent in the non-interpolating regime, these models are generally not consistent in the interpolating regime [18, 54, 155]. Moreover, little is known about the consistency of interpolating deep neural networks. Classical work [60] analyzing the consistency of neural networks utilizes the results of Cybenko [50] and Hornik [87] to show that the Bayes optimal classifier can be approximated by a neural network that is sufficiently wide; i.e., these prior results are concerned with the existence of networks that achieve consistency and do not present computationally feasible algorithms for finding such networks.

By establishing a connection between interpolating kernel smoothers and deep neural networks, we identify and construct an explicit class of interpolating neural networks that, when trained with gradient descent, achieve consistency for classification problems. Our results utilize the recent Neural Tangent Kernel (NTK) connection between training wide neural networks and using kernel methods. Several works [96, 110, 118, 120] established conditions under which using a kernel method with the NTK is equivalent to training neural networks, as network width approaches infinity. Given the conceptual simplicity of kernel methods, the NTK has been widely used as a tool for understanding the theoretical properties of neural networks [90, 110, 120, 154, 199]. Since neural networks in practice are often both wide and deep, we consider the natural extension of networks that are both infinitely wide and deep.

¹Consistency refers to a property that holds in an asymptotic sense as the number of training samples approaches infinity.

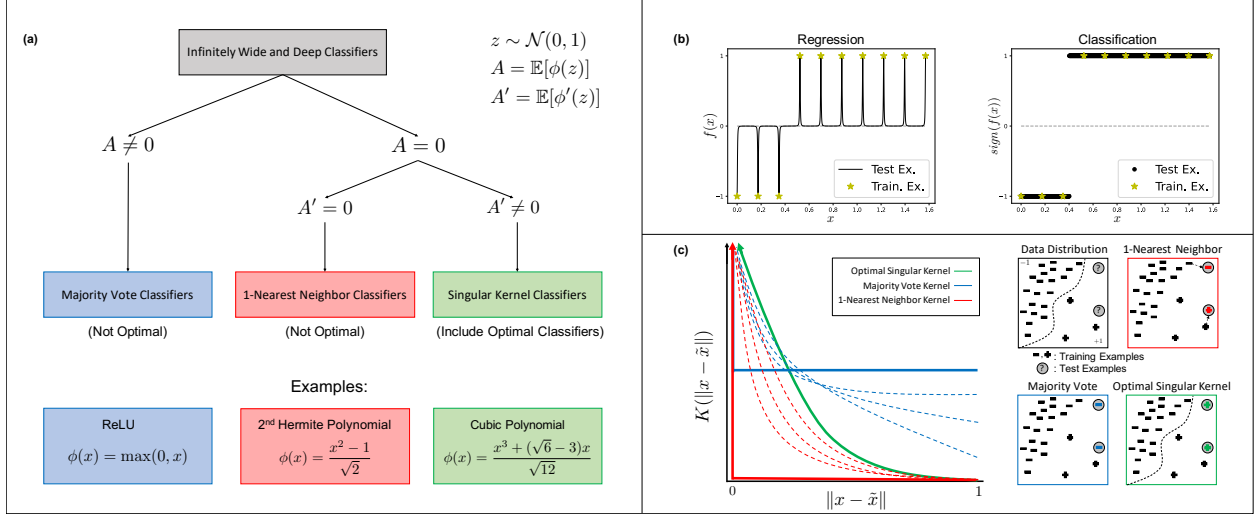


Figure 4-1: Behavior of infinitely wide and deep neural networks trained with gradient descent. (a) Taxonomy of infinitely wide and deep networks. Depending on the choice of the activation function, $\phi(\cdot)$, these models implement majority vote (blue), 1-nearest neighbor (red), or singular kernel classifiers (green), a subset of which achieve consistency. (b) Regression versus classification using infinitely wide and deep networks. While these models are not effective in the regression setting, since their predictions are near zero almost everywhere, they can achieve consistency for classification, where only the sign of the prediction matters. (c) Illustration of the different behaviors of infinitely wide and deep networks for varying activation functions. Depending on the activation function, infinitely wide and deep networks implement majority vote (blue), 1-nearest neighbor (red), or singular kernel classifiers that can achieve consistency (green). Singular kernels that grow too slowly are akin to majority vote classifiers (dashed blue), whereas those that grow too quickly are akin to weighted nearest neighbor classifiers (dashed red).

In particular, we focus on infinitely wide and deep networks in the classification setting and show that they have markedly different behavior than in the regression setting. Indeed, prior work [78, 90] showed that in the regression setting, infinitely wide and deep neural networks simply predict near-zero values at all test samples and thus, are far from consistent (see Fig. 4-1b). As a consequence, these models were dismissed as an approach for explaining the strong performance of deep networks in practice. In stark contrast to regression, we show that the sign of the predictor can be informative even when its numerical output is arbitrarily close to zero (see Fig. 4-1b for an illustration). In fact, as we show in this work, this is exactly how infinitely wide and deep neural networks can achieve Bayes optimal classification accuracy even though the output of the network approaches zero.

To characterize the behavior of infinitely wide and deep classifiers, we establish a taxonomy of such models, and we prove that it includes networks that achieve consistency (see Fig. 4-1a). More precisely, we prove that infinitely wide and deep neural network classifiers implement one of the following three well-known classifiers depending on the choice of activation function:

1. *1-nearest neighbor (1-NN) classifiers*: the prediction on a new sample is the label of the nearest sample (under Euclidean distance) in the training set [76].
2. *Majority vote classifiers*: the prediction on a new sample is the label of the class with greater representation in the training set.
3. *Singular kernel classifiers*: the prediction on a new sample is obtained by using the kernel $K(x, \tilde{x}) = \frac{R(\|x - \tilde{x}\|)}{\|x - \tilde{x}\|^\alpha}$ where $\alpha > 0$ is the order of the singularity.² As is standard when using kernel smoothers for

²For this order to be well-defined, $R(\cdot)$ is non-negative and satisfies $\inf_{|u| < \epsilon} R(u) > 0$ and $|R(u)| < C$ for some $\epsilon, C > 0$.

classification, the prediction, $m(x)$, on a new sample x given training data $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ is

$$m(x) = \text{sign} \left(\sum_{i=1}^n y^{(i)} K(x^{(i)}, x) \right). \quad (4.1)$$

As a corollary of a result in [54] it follows that singular kernel classifiers achieve consistency when α is the dimension of the data, d (see SI Appendix C). Hence our taxonomy and, in particular, Theorem 8 of this work provide exact conditions under which infinitely wide and deep neural network classifiers achieve consistency for any given data dimension. Notably, we identify a simple class of activation functions that yield singular kernel classifiers with $\alpha = d$, and we thus identify concrete examples of neural networks that achieve consistency. For example, for $d = 2$, the infinitely wide and deep classifier with activation function $\phi(x) = (x^3 + (\sqrt{6} - 3)x)/\sqrt{12}$ achieves consistency. Interestingly, the popular rectified linear unit (ReLU) activation $\phi(x) = \max(x, 0)$ leads to an infinitely wide and deep classifier that implements the majority vote classifier and is thus not consistent. Similarly, the activation function $\phi(x) = (x^2 - 1)/\sqrt{2}$ leads to an infinitely wide and deep classifier that implements the 1-NN classifier and is thus also not consistent.

We note that singular kernels provide a natural transition between 1-NN and majority vote classifiers. Namely, as discussed in [54], for $\alpha > d$, singular kernel classifiers behave akin to weighted nearest neighbor classifiers since $\|x - \tilde{x}\|^\alpha$ is extremely small for \tilde{x} near x . Similarly, for $\alpha < d$, singular kernel classifiers behave akin to majority vote classifiers since $\|x - \tilde{x}\|^\alpha$ is no longer small for \tilde{x} far from x . We visualize this transition between the three classes established in our taxonomy in Fig. 4-1c.

4.2 Taxonomy of Infinitely Wide and Deep Neural Networks

In the following, we construct a taxonomy of classifiers implemented by infinitely wide and deep neural networks. Our construction relies on the recent connection between infinitely wide neural networks and kernel methods [96]. In particular, this connection involves utilizing a kernel method known as a kernel machine, which is related to the kernel smoother described in Eq. [4.1]. In contrast to the kernel smoother, a kernel machine with kernel K is given by:

$$\text{sign} \left(y K_n^{-1} K(X, x) \right), \quad (4.2)$$

where $X = [x^{(1)} | x^{(2)} | \dots | x^{(n)}] \in \mathbb{R}^{d \times n}$ denotes the training data, $y = [y^{(1)}, y^{(2)}, \dots, y^{(n)}] \in \{-1, 1\}^{1 \times n}$ the labels, $K_n \in \mathbb{R}^{n \times n}$ satisfies $(K_n)_{i,j} = K(x^{(i)}, x^{(j)})$ and $K(X, x) \in \mathbb{R}^n$ satisfies $(K(X, x))_i = K(x^{(i)}, x)$. Both kernel methods can be used as prediction schemes for classification [166]. Note that while both algorithms produce predictors with the same functional form, their predictions are generally different. Indeed, understanding the relation between kernel smoothers and kernel machines will be critical to our proof of consistency.

Under certain conditions, training a neural network as width approaches infinity is equivalent³ to using a kernel machine with a specific kernel known as the Neural Tangent Kernel [96], which is defined below.

Definition 8. Let $f^{(L)}(x; \mathbf{W})$ denote a fully connected network⁴ with L hidden layers with parameters \mathbf{W} operating on data $x \in \mathbb{R}^d$. For $x, \tilde{x} \in \mathbb{R}^d$, the **Neural Tangent Kernel (NTK)** is given by:

$$K^{(L)}(x, \tilde{x}) = \langle \nabla_{\mathbf{W}} f^{(L)}(x; \mathbf{W}), \nabla_{\mathbf{W}} f^{(L)}(\tilde{x}; \mathbf{W}) \rangle.$$

To work with a simple closed form for the NTK and to avoid symmetries arising from the activation function, we will consider training data with probability density function on \mathcal{S}_+^d , where \mathcal{S}_+^d is the intersection

³This equivalence requires a particular initialization scheme on the weights known as the NTK initialization scheme [96]. Formally, this equivalence holds when an offset term corresponding to the predictions of the neural network at initialization are added to those given by the using a kernel machine with the NTK [96]. Like in prior works (e.g. [9, 78, 90, 109, 154]), we will analyze the NTK without such offset. This model corresponds to averaging the predictions of infinitely many infinite width neural networks [139].

⁴Throughout this work, we consider fully connected networks that have no bias terms.

of the unit sphere \mathcal{S}^d in $d + 1$ dimensions and the non-negative orthant.⁵ We also assume that no samples are repeated in the training data.

In this work, we analyze the behavior of infinitely wide and deep networks by analyzing the kernel machine in Eq. [4.2], as depth, L , goes to infinity. To perform our analysis, we utilize the recursive formula for the NTK of a deep network originally presented in [96]. Namely, $K^{(L)}$ can be expressed as a function of $K^{(L-1)}$ and the network activation function, $\phi(\cdot)$, yielding a discrete dynamical system indexed by L . The exact formula can be found in Eq. [4.5], and additional relevant results from prior works that are used in our proofs are referenced in SI Appendix A.

Remarkably, the properties of the resulting dynamical system as $L \rightarrow \infty$ are governed by the mean of $\phi(z)$ and its derivative, $\phi'(z)$, for $z \sim \mathcal{N}(0, 1)$. For simplicity, we will assume throughout that $\mathbb{E}[\phi(z)^2] < \infty$ and similarly $\mathbb{E}[\phi'(z)^2] < \infty$, an assumption that holds for many activation functions used in practice including ReLU, leaky ReLU, sigmoid, sinusoids, and polynomials. By defining $A = \mathbb{E}[\phi(z)]$ and $A' = \mathbb{E}[\phi'(z)]$, we break down our analysis into the following three cases:

- Case 1: $A = 0$, $A' \neq 0$,
- Case 2: $A = 0$, $A' = 0$,
- Case 3: $A \neq 0$.

Under cases 1 and 2, 0 is the unique fixed point attractor of the recurrence for $K^{(L)}$ and thus $K^{(L)}(x, \tilde{x}) \rightarrow 0$ as $L \rightarrow \infty$ for $x \neq \tilde{x}$. As a consequence, cases 1 and 2 lead to infinitely wide and deep neural networks that predict 0 almost everywhere. Hence, these networks are far from Bayes optimal in the regression setting and were thus dismissed as an approach for explaining the strong performance of deep networks. On the other hand, case 3 yields nonzero values for any pair of examples and thus, prior works that analyzed the regression setting [78, 90] focused on activation functions satisfying case 3.

In stark contrast to the regression setting, we will show that infinitely wide and deep networks with activation functions satisfying case 1 are effective for classification, with a subset achieving consistency. In particular, we will show that networks in case 1 implement singular kernel classifiers while those in case 2 implement 1-NN classifiers. Notably, we will identify conditions and provide explicit examples of activation functions in case 1 that guarantee consistency. We will then show that infinitely wide and deep classifiers with activations satisfying case 3 generally correspond to majority vote classifiers. A summary of our taxonomy is presented in Fig. 4-1a, and we will now discuss each of the three cases in more depth.

Case 1 ($A = 0, A' \neq 0$) networks implement singular kernel classifiers and can achieve optimality

We establish conditions on the activation function under which an infinitely wide and deep network implements a singular kernel classifier (Theorem 7). We then utilize results of [54] to show that this set of classifiers contains those that achieve consistency for any given data dimension. Lastly, we will present explicit activation functions that lead to infinitely wide and deep classifiers that achieve consistency. We begin with the following theorem, which establishes conditions under which the infinite depth limit of the NTK is a singular kernel.

Theorem 7. *Let $K^{(L)}$ denote the NTK of a fully connected neural network with L hidden layers and activation function $\phi(\cdot)$. For $z \sim \mathcal{N}(0, 1)$, define $A = \mathbb{E}[\phi(z)]$, $A' = \mathbb{E}[\phi'(z)]$, and $B' = \mathbb{E}[\phi'(z)^2]$. If $A = 0$ and $A' \neq 0$, then for $x, \tilde{x} \in \mathcal{S}_+^d$:*

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(x, \tilde{x})}{(A')^{2L}(L+1)} = \frac{R(\|x - \tilde{x}\|)}{\|x - \tilde{x}\|^\alpha},$$

where $\alpha = -2 \frac{\log(A'^2)}{\log(B')}$ and $R(\cdot)$ is non-negative, bounded from above, and bounded away from 0 around 0.

The full proof is presented in SI Appendix B, and we outline its key steps in Section 4.3. Theorem 8 below characterizes the activation functions for which the infinitely wide and deep network achieves consistency.

⁵For example, min-max scaling followed by projection onto the sphere results in the data lying in this region.

In particular, we establish the consistency of the infinitely wide and deep neural network classifier, $m_n(\cdot)$, given by taking the limit as $L \rightarrow \infty$ of the kernel machine in Eq. [4.2] with $K = K^{(L)}$, i.e.

$$m_n(x) = \lim_{L \rightarrow \infty} \text{sign} \left(y (K_n^{(L)})^{-1} K^{(L)}(X, x) \right). \quad (4.3)$$

Theorem 8. *Let m_n denote the classifier in Eq. [4.3] corresponding to training an infinitely wide and deep network with activation function $\phi(\cdot)$ on n training examples. For $z \sim \mathcal{N}(0, 1)$, define $A = \mathbb{E}[\phi(z)]$, $A' = \mathbb{E}[\phi'(z)]$, and $B' = \mathbb{E}[\phi'(z)^2]$. If*

$$A = 0 \quad \text{and} \quad A' \neq 0 \quad \text{and} \quad -\frac{\log(A'^2)}{\log(B')} = \frac{d}{2},$$

*then this classifier is Bayes optimal.*⁶

While the full proof of Theorem 8 is presented in SI Appendices B and C, we outline its key steps in Section 4.3. In particular, the proof follows by using Theorem 7 above, proving that m_n is a singular kernel classifier, and then using the results of [54], which establish conditions under which singular kernel estimators achieve optimality. The following corollaries (proofs in SI Appendix D) present concrete classes of activation functions that satisfy the conditions of Theorem 8 for any given data dimension d .

Corollary 1. *Let m_n denote the classifier in Eq. [4.3] corresponding to training an infinitely wide and deep network with activation function*

$$\phi(x) = \begin{cases} \frac{1}{\sqrt{2}} h_7(x) + \frac{1}{\sqrt{2}} x & \text{if } d = 1, \\ \frac{1}{2^{d/4}} h_3(x) + \sqrt{1 - \frac{2}{2^{d/2}}} h_2(x) + \frac{1}{2^{d/4}} x & \text{if } d \geq 2, \end{cases}$$

where $h_i(x)$ is the i^{th} probabilist's Hermite polynomial.⁷ Then the classifier m_n is Bayes optimal.

Corollary 2. *For $d \geq 2$, let m_n denote the classifier in Eq. [4.3] corresponding to training an infinitely wide and deep network with activation function*

$$\phi(x) = \frac{\sin(ax)}{\sqrt{\sinh(a^2)}}; \quad -\frac{\log \frac{a^2}{\sinh(a^2)}}{\log \frac{a^2 \cosh(a^2)}{\sinh(a^2)}} = \frac{d}{2}.$$

Then the classifier m_n is Bayes optimal.

We note the remarkable simplicity of the above activation functions yielding infinitely wide and deep networks that achieve consistency. In particular, for $d \geq 2$, Corollary 1 gives activations are simply cubic polynomials, and Corollary 2 gives sinusoidal activations where the frequency a increases with dimension (e.g. $a^2 \approx 2.676$ leads to consistency for $d = 2$ and $a^2 \approx 6.135$ leads to consistency for $d = 3$). Lastly, we note that our results are easily extended to the case where data has density on a submanifold of \mathcal{S}_+^d by simply selecting α to be the dimension of the data manifold in Theorem 7.

Case 2 ($A = 0, A' = 0$) networks implement 1-NN

We now identify conditions on the activation function under which infinitely wide and deep networks implement the 1-NN classifier.

⁶Let $m(x) = \arg \max_{\tilde{y} \in \{-1, 1\}} \mathbb{P}(y = \tilde{y} | x)$ denote the Bayes optimal classifier. Let X_n denote the training data in \mathcal{S}_+^d , let $f(\cdot)$ denote the density on \mathcal{S}_+^d , and let $m_{X_n} := m_n$ denote the classifier in Eq. [3]. Formally, Theorem 2 implies that at almost all $x \in \mathcal{S}_+^d$ with $f(x) > 0$ and for any $\epsilon > 0$, $m_{X_n}(x)$ converges to $m(x)$ in probability as $n \rightarrow \infty$, i.e.,

$$\lim_{n \rightarrow \infty} \mathbb{P}_{X_n} (|m_{X_n}(x) - m(x)| > \epsilon) = 0.$$

This is the same notion of consistency, i.e., *weak consistency*, established for the Hilbert kernel estimator in [54].

⁷The closed forms for these polynomials are as follows: $h_2(x) = \frac{x^2-1}{\sqrt{2}}$, $h_3 = \frac{x^3-3x}{\sqrt{6}}$, and $h_7(x) = \frac{x^7-21x^5+105x^3-105x}{12\sqrt{35}}$.

Theorem 9. Let m_n denote the classifier in Eq. [4.3] corresponding to training an infinitely wide and deep network with activation function $\phi(\cdot)$ on n training examples. For $z \sim \mathcal{N}(0, 1)$, define $A = \mathbb{E}[\phi(z)]$ and $A' = \mathbb{E}[\phi'(z)]$. If $A = A' = 0$, then $m_n(x)$ implements 1-NN classification for almost all $x \in \mathcal{S}_+^d$.

The proof of Theorem 9 is provided in SI Appendix E. The proof strategy is to show that the value of the kernel between a test example and its nearest training example dominates the prediction as $L \rightarrow \infty$. In particular, assuming without loss of generality that $x^T x^{(1)} > x^T x^{(j)}$ for $j \in \{2, 3, \dots, n\}$, we prove that:

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(x, x^{(j)})}{K^{(L)}(x, x^{(1)})} = 0.$$

As a result, after re-scaling by $K^{(L)}(x, x^{(1)})$, we obtain that $m_n(x) = \text{sign}(y^{(1)})$. We note that this proof is analogous to the standard proof that the Gaussian kernel $K(x, \tilde{x}) = \exp(-\gamma \|x - \tilde{x}\|^2)$ converges to the 1-NN classifier as $\gamma \rightarrow \infty$.

Case 3 ($A \neq 0$) networks implement majority vote classifiers

We now analyze infinitely wide and deep networks when the activation function satisfies $\mathbb{E}[\phi(z)] \neq 0$ for $z \sim \mathcal{N}(0, 1)$. In this setting, we establish conditions under which the infinitely wide and deep network implements majority vote classification, i.e., the prediction on test samples is simply the label of the class with greatest representation in the training set. More precisely, the following proposition (proof in SI Appendix F) implies that when the infinite depth NTK is a constant non-zero value for any two non-equal inputs, the resulting classifier is the majority vote classifier.

Proposition 4. Let m_n denote the classifier in Eq. [4.3] corresponding to training an infinitely wide and deep network with activation function $\phi(\cdot)$ on n training examples such that the sum of the labels $y^{(i)}$ is not 0. For any $x, \tilde{x} \in \mathcal{S}_+^d$ with $x \neq \tilde{x}$, if the NTK $K^{(L)}$ satisfies

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(x, \tilde{x})}{C(L)} = C_1 \text{ and } \lim_{L \rightarrow \infty} \frac{K^{(L)}(x, x)}{C(L)} \neq C_1, \quad (4.4)$$

with $C_1 > 0$ and $0 < C(L) < \infty$ for any L , then m_n implements the majority vote classifier, i.e.,

$$m_n(x) = \text{sign} \left(\sum_{i=1}^n y^{(i)} \right).$$

We now analyze which activation functions satisfy Eq. [4.4]. As described in [77, 108, 148, 204], under case 3, the value of $B' = \mathbb{E}[\phi'(z)^2]$ for $z \sim \mathcal{N}(0, 1)$ determines the fixed point attractors of $K^{(L)}$ as $L \rightarrow \infty$. Thus, the infinite depth behavior under case 3 can be broken down into three cases based on the value of B' . Using the terminology from [148], these cases are:

- (i) $B' > 1$ (Chaotic Phase),
- (ii) $B' < 1$ (Ordered Phase),
- (iii) $B' = 1$ (Edge of Chaos).

In Lemma 5 in SI Appendix G, we demonstrate that in the chaotic phase, the resulting infinite depth NTK satisfies the conditions of Proposition 4 and thus implements the majority vote classifier. In Lemma 6 in SI Appendix G, we similarly show that in the ordered phase the infinite depth NTK also corresponds to the majority vote classifier.⁸ The remaining case known as "edge of chaos" has been analyzed in prior works for specific activation functions; for example, the NTK for networks with ReLU activation satisfies Eq. [4.4] with $C_1 = \frac{1}{4}$ and $C(L) = L + 1$ [78, 90]. Hence by Proposition 4, the corresponding infinite depth classifier for ReLU networks corresponds to the majority vote classifier.

⁸More precisely, we consider the behavior of the infinite depth classifier under ridge-regularization, as the regularization term approaches 0.

Classifiers Implemented by Infinitely Wide and Deep Networks with Standard Activation Functions

We now discuss activation functions commonly used in practice and the classifiers implemented by infinitely wide and deep networks with such activation functions. The conditions of Theorem 7 are satisfied by several commonly used activation functions in practice including sine, erf, tanh, and hard tanh. However, as we prove in SI Appendix H the order of singularity, α , in Theorem 7 for all of these activation functions is near 0.5, which is the value of α that is required for consistency for data on the unit circle.

On the other hand, activation functions including ReLU, sigmoid, cosid (i.e. $\cos x - x$) [59], and swish (i.e. $\frac{x}{1+e^{-x}}$) [156] satisfy the conditions of Proposition 4, and thus, implement majority vote.

In SI Appendix I and Fig. S3, we provide experiments across several data distributions in which we compare the performance of infinitely wide and deep classifiers using standard activation functions such as ReLU, erf, and sine, which have closed forms for the NTK, against those that lead to consistent classifiers according to Theorem 8 above. In all cases, we observe a strong accordance between our experiments and theoretical results, showing that infinitely wide and deep networks using standard activation functions are far from consistent.

Practical relevance of our results. While this work derives activation functions that lead to infinitely wide and deep networks that are consistent for fixed data dimension as the number of training samples approaches infinity, we demonstrate the practical value of the derived activation functions in SI Appendix J and Fig. S4 on a variety of benchmarking datasets in the context of *finitely* deep networks and *finite* sample sizes, concentrating in particular on the small-sample regime. Namely, in SI Appendix J and Fig. S4, we show that grid searching over the activation functions provided in Corollaries 1 and 2 leads to improved performance over standard classifiers including 179 models from [61], fully connected ReLU networks, as well as ReLU NTKs from [10] on a variety of benchmarking datasets including (i) the 90 benchmarking classification tasks in the small-sample regime (with fewer than 5000 training samples) considered in [10], and (ii) the 3 classification tasks in the small-dimensional large-sample regime (with fewer than 15 features and greater than 10,000 training samples) considered in [61].

4.3 Outline of Proof Strategy for Theorems 7 and 8

In the following, we outline the proof strategy for our main results. This involves analyzing infinitely wide and deep networks via the limiting NTK kernel given by $K^{(L)}$ as the number of hidden layers $L \rightarrow \infty$. As shown in [96], $K^{(L)}$ can be written recursively in terms of $K^{(L-1)}$ and the so-called dual activation function, which was introduced in [53].

Definition 9. Let $\phi : \mathbb{R} \rightarrow \mathbb{R}$ be an activation function satisfying $\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2] < \infty$. Its **dual activation function** $\check{\phi} : [-1, 1] \rightarrow \mathbb{R}$ is given by

$$\check{\phi}(z) = \mathbb{E}_{(u,v) \sim \mathcal{N}(\mathbf{0}, \Lambda)}[\phi(u)\phi(v)], \quad \text{where } \Lambda = \begin{bmatrix} 1 & z \\ z & 1 \end{bmatrix}.$$

While all quantities in our theorems are stated in terms of activation functions, these can be restated in terms of dual activations as follows:

$$A^2 = \check{\phi}(0) \quad \text{and} \quad (A')^2 = \check{\phi}'(0) \quad \text{and} \quad B' = \check{\phi}'(1).$$

Assuming that ϕ is normalized such that $\check{\phi}(1) = 1$,⁹ the recursive formula for the NTK of a deep fully connected network for data on the unit sphere was described in [39, 96] in terms of dual activation functions as follows.

⁹Such normalization is always possible for any activation function satisfying $\mathbb{E}[\phi(z)^2] < \infty$ for $z \sim \mathcal{N}(0, 1)$ and has been used in various works before including [39, 77, 78, 90, 116, 199].

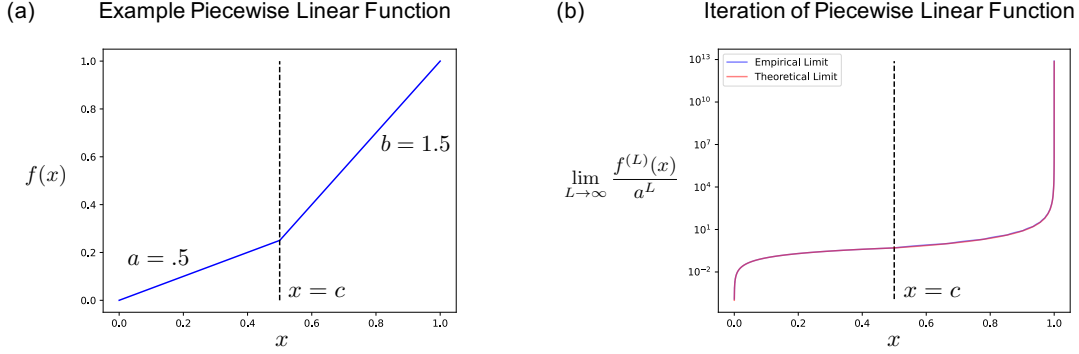


Figure 4-2: Iteration of a piecewise linear function on a unit interval leads to a function with a singularity at $x = 1$, upon appropriate normalization. (a) We consider the piecewise linear function $f(x)$ given by $1 - b(1 - x)$ on $(c, 1]$ and ax on $[0, c]$, where $a = .5, b = 1.5$ and $c = \frac{b-1}{b-a}$. (b) We observe that upon iterating $f(\cdot)$ numerically to the limit of machine precision, the resulting function strongly agrees with the theoretical limit of Lemma 1 given by a function with singularity of order $-\log_b a \approx 1.7$.

Recursive Formula for the NTK. Let $f^{(L)}(x; \mathbf{W})$ denote a fully connected neural network with L hidden layers and activation $\phi(\cdot)$. For $x, \tilde{x} \in \mathcal{S}^d$, let $z = x^T \tilde{x}$. Then $K^{(L)}$ is radial, i.e. $K^{(L)}(x, \tilde{x}) = K^{(L)}(z)$, with $K^{(0)}(z) = z$ and

$$K^{(L)}(z) = \check{\phi}^{(L)}(z) + K^{(L-1)}(z)\check{\phi}'(\check{\phi}^{(L-1)}(z)), \quad (4.5)$$

where $\check{\phi}^{(L)}(z) = \check{\phi}(\check{\phi}^{(L-1)}(z))$ with $\check{\phi}^{(0)}(z) = \check{\phi}(z)$ and $\check{\phi}'(\cdot)$ denotes the derivative of $\check{\phi}(\cdot)$.

We utilize the dynamical system in Eq. [4.5] to analyze the behavior of $K^{(L)}(\cdot)$ as $L \rightarrow \infty$. Theorem 7 implies that upon normalization by $(L+1)\check{\phi}'(0)^L$, this dynamical system converges to a singular kernel with singularity of order $\alpha = -\log(\check{\phi}'(0)) / \log(\check{\phi}'(1))$. We now present a sketch of the proof of this result.

We first derive the order of the singularity upon iteration of $\check{\phi}$, since as we show in SI Appendix B, the order of the singularity of the infinite depth NTK is the same as that of the iterated $\check{\phi}$. Since we consider data in \mathcal{S}_+^d , $\check{\phi}(\cdot)$ is a function defined on the unit interval $[0, 1]$. Hence, understanding the properties of infinitely wide and deep networks reduces to understanding the properties of iterating a function on the unit interval. To provide intuition around how the iteration of a function on the unit interval can give rise to a function with a singularity, we discuss iterating a piecewise linear function as an illuminating example; see Fig. 4-2 for a visualization.

Lemma 1. For $0 < a < 1$ and $b > 1$, let $f : [0, 1] \rightarrow \mathbb{R}$ and $c = \frac{b-1}{b-a}$ such that

$$f(x) = \begin{cases} ax & \text{if } x \in [0, c] \\ 1 - b(1 - x) & \text{if } x \in (c, 1] \end{cases}.$$

Then,

$$\lim_{L \rightarrow \infty} \frac{f^{(L)}(x)}{a^L} = \frac{R(x)}{(1-x)^{-\log_b a}},$$

where $R(x)$ is non-negative, bounded from above and bounded away from 0 around $x = 1$.

Proof. For any $x \in [0, c]$, we necessarily have:

$$\lim_{L \rightarrow \infty} \frac{f^{(L)}(x)}{a^L} = \lim_{L \rightarrow \infty} \frac{a^L x}{a^L} = x.$$

Now for fixed $x \in (c, 1)$, since $x = 0$ is an attractive fixed-point of f , let L_0 denote the smallest integer such that $f^{(L_0)}(x) \leq c$. Hence, since $f^{(L_0)}(x) \in [0, c]$, we obtain:

$$\lim_{L \rightarrow \infty} \frac{f^{(L)}(x)}{a^L} = \lim_{L \rightarrow \infty} \frac{f^{(L-L_0)}(f^{(L_0)}(x))}{a^{L-L_0}} \frac{1}{a^{L_0}} = f^{(L_0)}(x) a^{-L_0}. \quad (4.6)$$

We next solve for L_0 by analyzing the iteration of $g(x) := 1 - b(1 - x)$. In particular, we observe that $g^{(L)}(x) = 1 - b^L(1 - x)$, and thus L_0 is given by:

$$\begin{aligned} 1 - b^{L_0}(1 - x) &\leq c \\ \implies L_0 &= \left\lceil \log_a \left(\frac{1-x}{1-c} \right)^{-\log_b a} \right\rceil \\ \implies a^{-L_0} &\in \left[\left(\frac{1-c}{1-x} \right)^{-\log_b a}, \frac{1}{a} \left(\frac{1-c}{1-x} \right)^{-\log_b a} \right]. \end{aligned}$$

Hence, by Eq. [4.6] we conclude that for $x \in (c, 1)$, it holds that

$$\lim_{L \rightarrow \infty} \frac{f^{(L)}(x)}{a^L} = \frac{R(x)}{(1-x)^{-\log_b a}},$$

where $R(x)$ is non-negative, bounded from above and bounded away from 0 around $x = 1$, which completes the proof. \square

In SI Appendix B, we extend this analysis to the iteration of dual activations on the unit interval, thereby establishing the order of a singularity obtained by iterating dual activation functions. We then show that this order equals the order of the singularity given by the infinite depth NTK.

Next, we discuss the proof strategy for Theorem 8, which establishes conditions on the activation function under which infinitely wide and deep networks achieve consistency in the classification setting. The proof builds on results in [54] characterizing the consistency of singular kernel smoothers of the form

$$g(x) = \frac{\sum_{i=1}^n y^{(i)} K(x^{(i)}, x)}{\sum_{i=1}^n K(x^{(i)}, x)}, \text{ where } K(x^{(i)}, x) = \frac{1}{\|x - x^{(i)}\|^\alpha}.$$

In particular, it is shown that if $\alpha = d$, then $g(x)$ achieves consistency. Since Theorem 7 establishes conditions under which the infinite depth NTK implements a singular kernel, to complete the proof we show that infinitely wide and deep classifiers achieve consistency by (1) showing that the classifier m_n implements a singular kernel smoother, and (2) selecting ϕ such that $\alpha = d$ for the corresponding singular kernel.

4.4 Discussion

In this work, we identified and constructed explicit neural networks that achieve consistency for classification when trained using standard procedures. Furthermore, we provided a taxonomy characterizing the behavior of infinitely wide and deep neural network classifiers. Namely, we showed that these models implement one of the following three well-known types of classifiers: (1) 1-NN (test predictions are given by the label of the nearest training example) ; (2) majority vote (test predictions are given by the label of the class with greatest representation in the training set); or (3) singular kernel classifiers (a set of classifiers containing those that achieve consistency). We conclude by discussing implications of our work and future extensions.

Benefit of Depth in Neural Networks. An emerging trend in machine learning is that larger neural networks capable of interpolating (i.e., perfectly fitting) the training data, can generalize to test data [22, 135, 208]. While the size of neural networks can be increased through width or depth, works such as [22, 135] primarily identified a benefit to increasing network width. Indeed, it remained unclear whether there was any benefit to using extremely deep networks. A line of prior work analyzed the impact of selecting

activation functions and initializations to enable the training of deep networks [134, 148, 204], while other works [138, 198, 199] empirically demonstrated that drastically increasing depth in networks with ReLU or tanh activation could lead to worse performance. In this work, we established a remarkable benefit of very deep networks by proving that they achieve consistency with a careful choice of activation function. In line with previous empirical findings, we proved that deep networks with activations such as ReLU or tanh do not achieve consistency.

Regression versus Classification. Our results demonstrate the benefit of using infinitely wide and deep networks for classification tasks. We note that this is in stark contrast to the regression setting, where infinitely deep and wide neural networks are far from consistent, as they simply predict a non-negative, near-zero constant almost everywhere [78, 90]. Thus, our work provides concrete examples of neural networks that are effective for classification but not regression. A key difference between regression and classification is that classification requires only the sign of the prediction. In particular, as we show in this work, the sign of the prediction of an infinitely wide and deep network can be meaningful for classification even though the prediction itself is close to 0.

Edge of Chaos Regime. An interesting class of models that are only partially characterized by our taxonomy corresponds to networks with activations in the edge of chaos regime, i.e., when the activation function, $\phi(\cdot)$ satisfies $\mathbb{E}[\phi(z)] \neq 0$ and $\mathbb{E}[\phi'(z)^2] = 1$ for $z \sim \mathcal{N}(0, 1)$. We proved that all activations in this class that have been described so far [78, 90], including the popular ReLU activation, give rise to infinitely wide and deep networks that implement the majority vote classifier. While it appears that all activations in this class lead to the majority vote classifier, it remains open to understand whether there exist other activations in this regime that implement alternative classifiers. Moreover, works analyzing the edge of chaos regime typically consider infinite width networks with bias terms. While these bias terms are often set to avoid exponential convergence of predictions with increasing depth, they can be detrimental in the classification setting. For example, the work of [78] shows that with appropriate bias, the tanh activation function leads to an infinitely wide and deep network that satisfies our Proposition 4 and thus implements majority vote. However, without the bias, this activation function satisfies Theorem 7 and thus leads to a singular kernel classifier. It is an interesting question to characterize how the addition of bias terms may influence our taxonomy.

Finite vs. Infinite Neural Networks. In this work, we identified and constructed infinitely wide and deep classifiers that achieve consistency. In particular, our results imply weak consistency of infinitely wide and deep networks, which means that these models converge in probability to the Bayes optimal classifier as the number of training samples approaches infinity. While recent work [18] demonstrated that finite depth NTKs are not universally consistent, i.e., they are not consistent for arbitrary distributions, it remains open as to whether these models are consistent in a weaker sense. An important next question is to understand whether interpolating neural networks that are finitely wide and deep can achieve consistency for classification and provide specific activation functions to do so. Some evidence in this direction is given by recent work demonstrating that sufficiently wide and deep ReLU networks correctly classify points on disjoint curves on a sphere [191]. We also note that Bayes consistency considers the setting when the number of training examples approaches infinity. Another natural next step is to characterize the number of training examples needed for infinitely wide and deep classifiers to reasonably approximate the Bayes optimal classifier. Recent work [129] identified a slow (logarithmic) rate of convergence for singular kernel classifiers, thereby implying that many training examples are needed for these models to be effective in practice. An important open direction of future work is thus to determine not only whether finitely wide and deep networks are Bayes optimal for classification but also whether these models require fewer samples to perform well in practice.

Chapter 5

Mechanism of feature learning in deep fully connected neural networks

In this final chapter, we identify the mechanism of neural feature learning, which is a key component driving the success of deep neural networks used in practice. We posit the Deep Neural Feature Ansatz, which states that feature learning in neural networks occurs through a procedure known as average gradient outer product. Our ansatz explains various prominent deep learning phenomena and unifies previous work on feature learning analyzing how particular components of networks lead to feature learning. Importantly, our ansatz provides a method for integrating feature learning into any machine learning model. We use our ansatz to integrate feature learning into over-parameterized kernel machines and develop Recursive Feature Machines (RFMs). We illustrate the practical value of RFMs by using them to achieve state-of-the-art results across two tabular data benchmarks containing over 151 classification and regression tasks. The work presented in this section culminated into the following paper [151].

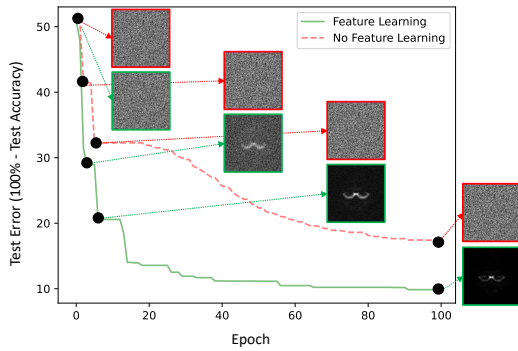
5.1 Introduction

In the last few years, neural networks have led to major breakthroughs on a variety of applications including image generation [157], protein folding [183], and language understanding and generation [35]. The ability of these models to automatically learn and utilize problem-specific *features*, or patterns in data, for prediction is thought to be a central contributor to their success [169, 203]. Thus, a major goal of machine learning research has been to identify the mechanism through which such neural feature learning occurs and which features are selected. Indeed, understanding this mechanism provides the opportunity to design networks with improved reliability and model transparency needed for various scientific and clinical applications (e.g., natural disaster forecasting, clinical diagnostics).

Prior works refer to neural feature learning as the change in a network’s internal, intermediate representations through the course of training [33, 203]. Significant research effort [2, 13, 14, 52, 73, 89, 111, 114, 131, 160, 169, 189, 203, 211] has shown the benefits of feature learning in neural networks over non-feature learning models. Yet, precise characterization of the feature learning mechanism and how features emerge remained an unsolved problem.

In this work, we posit the mechanism for feature learning in deep, nonlinear fully connected neural networks. Informally, this mechanism corresponds to the approach of progressively re-weighting features in proportion to the influence they have on the predictions. Mathematically stated, if W_i denotes the weights of a trained deep network at layer i , then Gram matrix $W_i^T W_i$, which we refer to as the *ith layer Neural Feature Matrix* (NFM), is proportional to the average gradient outer product of the network with respect to the input to this layer.

As an illustrative example of our results, consider neural networks trained to classify the presence of glasses in images of faces. In Fig. 5-1A, we compare the NFMs and performance of a non-feature learning network with fixed first layer weights and a feature learning network where all weights are updated. While the NFM of the non-feature learning model (shown in red) is unchanging through training, the NFM of the

A Neural feature learning when classifying glasses in images**B** Average gradient outer product captures neural network features

Classification Task	5 o'clock shadow	Necktie	Smiling	Rosy Cheeks
Correlation between first layer NFM and Avg. gradient outer product	0.993	0.970	0.998	0.991
NFM of first layer (Top Eigenvector)				
Avg. gradient outer product (Top Eigenvector)				

Figure 5-1: **(A)** A demonstration of neural feature learning. We train two fully connected neural networks with two-hidden-layers and ReLU activation to classify glasses in image data (96×96 images) from the CelebA dataset [122], one in which the first layer is not trained and the other in which all layers are trained. We visualize the diagonal of the first layer neural feature matrix (NFM), $W_1^T W_1$, of the trained networks and observe that the network with better performance learns to select pixels corresponding to glasses. **(B)** We show the NFM of a layer is well approximated by the average gradient outer product of the trained network taken with respect to the input of this layer. The correlation between the first layer NFM and the average gradient outer product with respect to the input data is greater than 0.97 for four different classification tasks from the CelebA dataset. We visualize the top eigenvector of these matrices and observe that both are visually indistinguishable and highlight relevant features for prediction.

feature learning model (shown in green) evolves to represent a pattern corresponding to glasses. Even though both networks are able to fit the training data equally well, the feature learning model has significantly lower test classification error. A major finding of our work is that we are able to recover the key first layer NFM matrix without access to the internal structure of the neural network. To illustrate this finding, in Fig. 5-1B, we show that the average gradient outer product of a trained neural network with respect to the data is strongly correlated (Pearson correlation greater than .97) with the first layer NFM for a variety of classification tasks.

We empirically show that the feature learning mechanism identified in our work unifies previous lines of investigation, which study the relationship between neural feature learning and various aspects of neural networks such as network architecture [189] and weight initialization scheme [203]. In settings where feature learning is argued to occur (e.g., in finite width networks and networks initialized near zero), the average gradient outer product is more correlated with the neural feature matrices. Moreover, our mechanism explains prominent deep learning phenomena including the emergence of spurious features and biases in trained neural networks [168, 171], grokking [149], and how pruning networks can increase performance [64].

Importantly, as the average gradient outer product can be computed given any predictor, our result provides a backpropagation-free approach for feature learning with any machine learning model including those models that previously had no feature learning capabilities. Indeed, we can iterate between training a machine learning model and computing the average gradient outer product of this model to learn features. We apply this procedure to enable feature learning in class of non-feature learning models known kernel machines [5, 166] and refer to the resulting algorithm as a *Recursive Feature Machine* (RFM). We demonstrate that RFMs achieve state-of-the-art performance across two tabular data benchmarks covering over 150 datasets [61, 70], thereby highlighting the practical value of leveraging the feature learning mechanism identified in this work.

5.2 Results

Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ denote a fully connected network with L hidden layers for $L > 1$, weight matrices $\{W_i\}_{i=1}^{L+1}$, and elementwise activation function ϕ of the form

$$f(x) = W_{L+1}h_L(x) \quad ; \quad h_\ell(x) = \phi(W_{\ell-1}h_{\ell-1}(x)) \text{ for } \ell \in \{2, \dots, L\}$$

with $h_1(x) = x$. We refer to the terms $h_i(x)$ as the *features* at layer i . We can characterize how features $h_{i+1}(x)$ are constructed by understanding how W_i scales and rotates elements of $h_i(x)$. These scaling and rotation quantities are recovered mathematically from the eigenvalues and eigenvectors of the matrix $W_i^T W_i$, which is the NFM at layer i . Hence, to characterize how features are updated in any layer of a trained neural network, it suffices to characterize how the corresponding layer’s NFM is constructed. Before mathematically stating how such NFMs are built, we connect NFM construction to the following intuitive procedure for selecting features.

Given any predictor, a natural approach for identifying important features is to rank them by the magnitude of change in prediction upon perturbation. When considering infinitesimally small feature perturbations on real-valued predictors, this approach is mathematically equivalent to computing the magnitude of the derivative of the predictor output with respect to each feature. These magnitudes are computed by the gradient outer product of the predictor given by $(\nabla f(x))(\nabla f(x))^T$ where $\nabla f(x)$ is the gradient of a predictor, f , at a point x .¹

Our main insight, the *Deep Neural Feature Ansatz*, is that deep networks learn features by implementing the above approach for feature selection. Mathematically stated, we posit that the NFM of any layer of a trained network is proportional to the average gradient outer product of the network taken with respect to the input to this layer. In particular, let W_i denote the weights of layer i of a deep, nonlinear fully connected neural network, f . Given a sample x , let $h_i(x)$ denote the input into layer i of the network, and let f_i denote the sub-network of f operating on $h_i(x)$. Suppose that f is trained on n samples $\{(x_p, y_p)\}_{p=1}^n$. Then throughout training,

$$W_i^T W_i \propto \frac{1}{n} \sum_{p=1}^n \nabla f_i(h_i(x_p)) \nabla f_i(h_i(x_p))^T ; \quad (\text{Deep Neural Feature Ansatz})$$

where $\nabla f_i(h_i(x_p))$ denotes the gradient of f_i with respect to $h_i(x_p)$.² We refer to this statement as the Deep Neural Feature Ansatz. Formally, we prove that the ansatz holds when using gradient descent to layer-wise train (1) ensembles of deep fully connected networks and (2) deep fully connected networks with the trainable layer initialized at zero (see Section 5.2.4 and Appendix A). We note that for the special case of the first layer and for networks with scalar outputs, the right hand side is related to the statistical estimator known as the expected gradient outer product [74, 88, 181, 197].

Next, we empirically validate the ansatz when training all layers of deep fully connected networks across 127 classification tasks. In particular, in Fig. 5-2, we train fully connected networks with ReLU activation, five-hidden layers, 1024 hidden units per layer using stochastic gradient descent on the 121 classification tasks from [61]. In our experiments, we initialize the first layer weights near zero to reduce the impact of the initial weights in computing correlations (see Appendix B). In Fig. 5-2A, we observe that the Pearson correlation between the NFMs after training and the average gradient outer products have median value above .85 (shown in green) and are consistently higher than the corresponding correlation between the NFMs after training and those at initialization (shown in red). Note that the gap between the two correlations is larger for layers 2 through 5 since these all have NFMs of dimension 1024×1024 while the first layer NFM depends on the dimension of the input data, which is on average 28.84. In addition to the 121 tasks, we also validate the ansatz on six different image classification tasks across the CelebA dataset [122] and Street View House Numbers (SVHN) dataset [137] (see Appendix Fig. D-1). In Fig. 5-2B, we provide a visualization of the NFMs at initialization, NFMs after training, and average gradient outer products for a fully connected

¹For predictors with multi-dimensional outputs, we consider the Jacobian Gram matrix given by $(Jf(x))^T(Jf(x))$, where $Jf(x)$ is the Jacobian of a predictor, f , at a point x .

²Additionally, we note that the right hand side of the ansatz can be viewed as a covariance matrix when the gradients are centered.

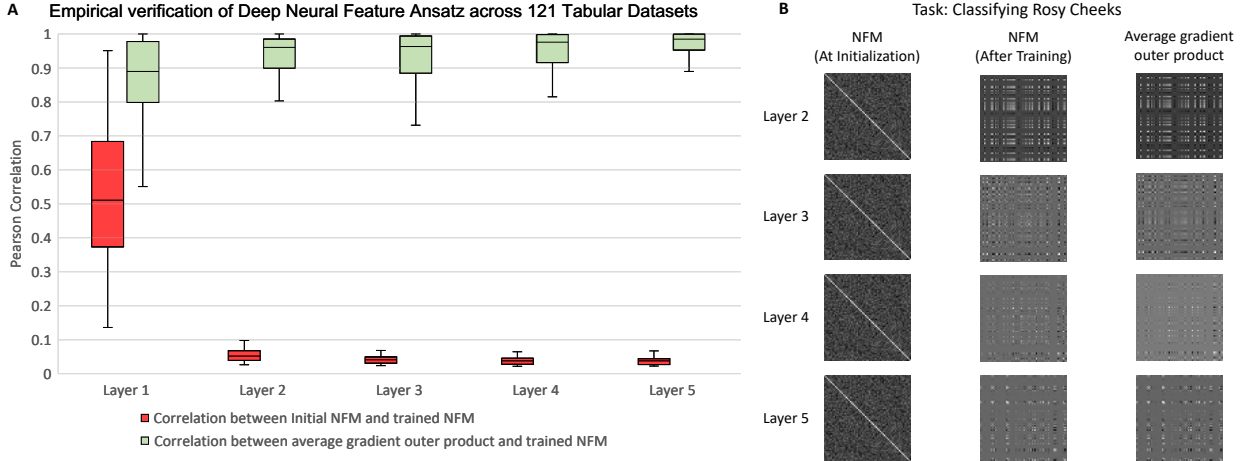


Figure 5-2: **(A)** The correlation between the initial NFM and trained NFM (red) and the correlation between average gradient outer product and the trained NFM (green) for each layer in five-hidden-layer ReLU fully connected networks with 1024 hidden units per layer trained on 121 tabular data classification tasks from [61]. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix. The higher correlation with the initial NFM in layer 1 is due to the fact that these matrices are smaller than the 1024×1024 matrices in the remaining layers (on average 28.84 features across all 121 tasks). **(B)** A visualization of the 64×64 NFM at initialization, the trained NFM, and average gradient outer product for layers 2 through 5 of the network trained to classify rosy cheeks from CelebA image data [122]. While the NFM at initialization is close to the identity matrix, the NFM after training has a qualitatively different structure that is captured by the average gradient outer product (correlation $> .78$). While we omit the layer 1 visualization here since the matrices are of size 27648×27648 , the correlation between the first layer NFM after training and the corresponding average gradient outer product is .99.

network with five-hidden-layers, 64 hidden units per layer with ReLU activation trained to classify rosy cheeks in CelebA images. We observe that while NFMs after training have qualitatively different structure than the NFMs at initialization, such structure is accurately captured by average gradient outer products. In addition to the above experiments, we empirically validate that the ansatz holds for a variety of commonly used nonlinearities such as leaky ReLU [200], hyperbolic tangent, sigmoid, and sinusoid and using standard optimization algorithms such as Adam [101] (see Appendix Fig. D-2).

Our ansatz unifies several previous lines of investigation into feature learning. In Appendix Figs. D-3 and D-4, we provide empirical evidence that the NFMs and average gradient outer products have greater correlation for finite width networks and networks initialized near zero, which are key regimes in which feature learning is argued to occur [189, 203]. In particular, we corroborate our results by reporting correlation between the NFMs after training and the average gradient outer products of networks trained on the 121 tabular classification tasks from [61] across 5 different widths and 5 initialization schemes.

5.2.1 Deep Neural Feature Ansatz sheds light on notable phenomena from deep learning

Empirical studies of deep neural networks have brought to light a number of remarkable and often counter-intuitive phenomena. We proceed to show that the mechanism of feature learning identified by our Deep Neural Feature Ansatz provides an explanation for several notable deep learning phenomena including (1) the emergence of spurious features [93, 171] and simplicity biases [91, 168]; (2) grokking [149]; and (3) lottery tickets in neural networks [64].

Spurious features and simplicity biases of neural networks. The Deep Neural Feature Ansatz implies the emergence of simplicity biases and spurious features in fully connected neural networks. Simplicity bias refers to the property of neural networks utilizing the “simplest” available features for prediction [81, 91,

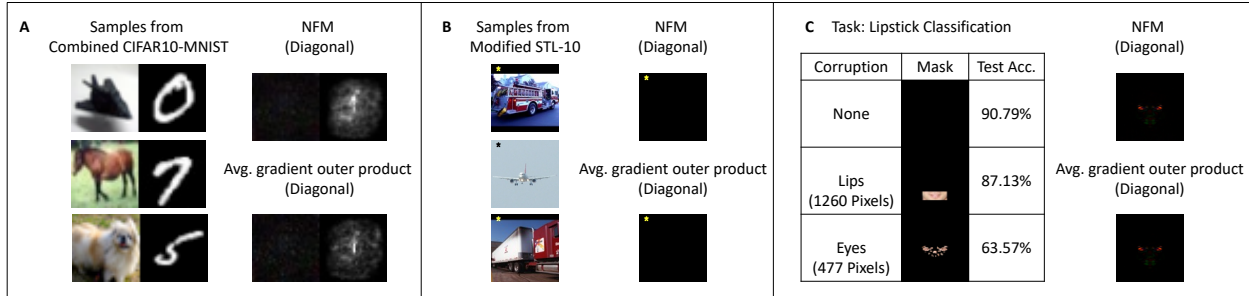


Figure 5-3: The Deep Neural Feature Ansatz enables identification of and simplicity biases and spurious features in fully connected networks. **(A)** When trained on 50000 concatenated 32×64 resolution images from CIFAR10 and MNIST datasets, the diagonal of the first layer NFM of a five-hidden-layer fully connected ReLU network indicates that digit pixels are primarily used as features for classification. The average gradient outer product confirms that perturbing these pixels leads to the greatest change in predicted values. **(B)** When trained on 1000 96×96 images of planes and trucks modified with a 7×8 pixel star pattern in the upper left corner, the diagonal of the first layer NFM of a five-hidden-layer ReLU fully connected network indicates the network relies solely on the star pattern for prediction. **(C)** When trained on 40000 96×96 images from CelebA to classify lipstick, the diagonal of the first layer NFM of a five-hidden-layer ReLU fully connected network indicates the network unexpectedly relies on eye pixels for classification. To corroborate this finding, we find that perturbing test samples by masking the lips leads to only a 3.66% drop in test accuracy, but perturbing the test samples by masking the eyes based on the average gradient outer product leads to a 27.22% drop in test accuracy.

[99, 145, 168] even when multiple features are equally indicative of class labels. A consequence of simplicity bias is the emergence of spurious features, which are patterns that are correlated but are not necessarily causally related to the predictive targets [93, 171]. Examples of neural networks leveraging spurious features include neural networks using the presence of fingers to detect band-aids [171] or, problematically, using surgical skin markers to predict malignant skin lesions [195]. Frequently, these spurious features are “simpler” than the patterns we consider to be causally predictive. Given their strong correlation with labels, perturbing these simple or spurious features will lead to a larger change in the prediction of a trained model than perturbing other available features, often including those causally related to the predictor. Hence, the ansatz implies that neural feature learning will reinforce such features.

We demonstrate these phenomena empirically in Fig. 5-3 upon training fully connected networks on three image classification tasks (see Appendix B for training methodology). In Fig. 5-3A, we consider the task from [168] and train a model on 50,000 concatenated images from CIFAR10 and MNIST datasets [105, 107]. After training, we visualize the diagonal of the first layer NFM and observe that the model is simply relying on the digit for recognizing the image. We observe that the average gradient outer product is correlated with the NFM (Pearson correlation 0.8504), which indicates that perturbing digit pixels leads to the greatest change in prediction. In Fig. 5-3B, we show that neural networks will rely primarily on spurious features for prediction even when there are only few such features. In particular, we trained fully connected networks to classify between 1000 modified images of trucks and planes from the STL-10 dataset [45] with trucks containing a gold star pattern and planes containing a black star pattern in the upper left corner of the image. Visualizing the diagonal of the first layer NFM and average gradient outer product indicates that the network simply learns to rely only on the star pattern for prediction.

Lastly, we showcase the power of our ansatz by using it to identify spurious features for a deep network trained to classify the presence of lipstick in CelebA images. In Fig. 5-3C, we observe that the model on original test samples achieves 90.79% accuracy. Yet, by visualizing the diagonals of the NFM and average gradient outer product, we observe that the trained model is unexpectedly relying on the eyes to determine whether the individual is wearing lipstick. To further corroborate this finding, we observe that the test accuracy drops only slightly by 3.66% when replacing the lips of all test samples with those of one individual. If we instead replace the eyes of all test samples according to the mask given by the diagonal of the average gradient outer product, test accuracy drops by 27.22% to slightly above random chance.

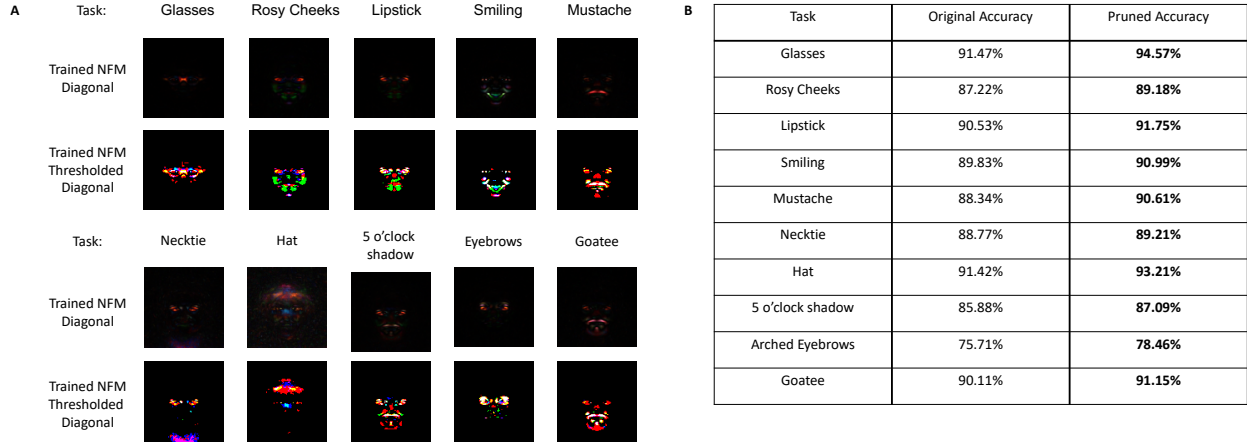


Figure 5-4: Lottery tickets in fully connected networks. **(A)** Visualizations of the diagonals of first layer Neural Feature Matrices from a two-hidden-layer, width 1024 ReLU network trained on classification tasks from CelebA and the diagonals after thresholding (replacing with pixel value 1) the top 2% of pixels. There are 553 nonzero pixel values in the masked images. **(B)** Comparison in accuracy after re-training randomly initialized neural networks of the same architecture on the masked images (i.e., pruning 98% of the corresponding columns of the first layer weights).

Lottery Tickets. Introduced in [64], the “lottery ticket hypothesis” refers to the claim that a randomly-initialized neural network contains a sub-network that can match or outperform the trained network when trained in isolation. Such sub-networks are typically found by pruning away weights with the smallest magnitude [64]. The sparsity of feature matrices identified in this work provides direct evidence for this hypothesis. Indeed, such sparsity is immediately evident when visualizing the diagonals of the feature matrix as in Fig. 5-4A.

In line with the lottery ticket hypothesis, we demonstrate that retraining neural networks after thresholding coordinates of the data corresponding to these sparse regions in the neural feature matrix leads to a consistent increase in performance in many settings. In Fig. 5-4A and B, we prune 98% of pixels in CelebA images according to the features identified by neural feature matrix and indeed, observe a consistent increase in predictive performance upon retraining a neural net on the thresholded features.

Grokking. Introduced in recent work [149], grokking refers to the phenomenon of deep networks exhibiting a dramatic increase in test accuracy when training past the point where training accuracy is 100%. We showcase a similar effect by training neural networks to classify between a subset of 96×96 resolution images of airplanes and trucks from the STL-10 dataset [45] (training details are presented in Appendix B). We modify this subset with two key features to enable grokking: (1) the dataset is small with a large class imbalance between the two classes with 500 examples of airplanes and 53 examples of trucks and (2) there is a small star of pixels in the upper left corner of each image that is colored white or black based on the class label (see Fig. 5-5A). The test set is balanced with 800 examples of each class.

In Fig. 5-5B, we observe that grokking aligns with our ansatz. Indeed, in Fig. 5-5B, we observe that the network can achieve near 100% training accuracy without any feature learning, but test accuracy remains at roughly 80%. Yet, as training continues past this point, the average gradient outer product up-weights pixels corresponding to the star pattern, as indicated by the first layer NFM, and test accuracy improves drastically to 99.38%.

5.2.2 Integrating feature learning into machine learning models

We now leverage the mechanism of feature learning identified in the ansatz to provide an algorithm for integrating feature learning into any machine learning model. We then showcase the power of this algorithm by applying it to classical, non-feature learning models known as kernel machines and achieving state-of-the-

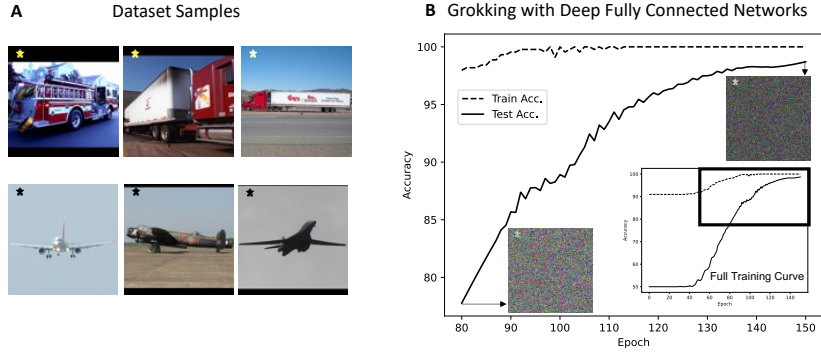


Figure 5-5: Grokking in fully connected networks. **(A)** Modified 96×96 resolution images from a subset of STL-10 [45] in which a small star in the upper left indicates whether the image is a truck or an airplane. We use 553 total training examples with 500 examples of airplanes and 53 examples of trucks. **(B)** A two-hidden-layer fully connected network quickly reaches near 100% training accuracy. Yet, as training continues past this point, the test accuracy rises drastically from 80% to 99.38%. Corresponding feature matrices (shown as inserts) indicate that test accuracy improved since the network has learned the star pixels that give away the class label.

art performance on tabular datasets.

A key insight of our ansatz is that neural feature learning occurs through the average gradient outer product, which is a mathematical operation that can be applied to any function. Given its universality, we can apply it to any machine learning model to enable feature learning. In particular, we use an iterative two-step strategy that alternates between first training any predictor and then using the average gradient outer product to directly learn features.

To demonstrate the power of this feature learning approach, we apply it to classical, non-feature learning kernel machines [166] by (1) estimating a predictor using a kernel machine ; (2) learning features using the average gradient outer product of the trained predictor ; and (3) repeating these steps after using the learned features to transform input to the predictor. For completeness, background on kernels is provided in Appendix C. Intuitively, training a kernel machine involves solving linear regression after applying a feature transformation on the data. Unlike traditional kernel functions that are fixed in advance before training, we use kernel functions that incorporate a learnable feature matrix M into the kernel function. For simplicity, we utilize a generalization of the Laplace kernel given by $K_M(x, z) = \exp(-\gamma \|x - z\|_M)$ where $\gamma > 0$, M is a positive semi-definite, symmetric feature matrix, and $\|x - z\|_M^2 := (x - z)^T M (x - z)$ denotes the Mahanobis distance between data points x, z .³ We now alternate between using kernel regression with the kernel function, K_M , to estimate a predictor and using the average gradient outer product to update the feature matrix, M . We refer to the resulting algorithm, presented in Algorithm 1, as a *Recursive Feature Machine* (RFM).

Algorithm 1 Recursive Feature Machine (RFM)

Input: X, y, K_M, T ▷ Training data: (X, y) , kernel function: K_M , and number of iterations: T
Output: α, M ▷ Solution to kernel regression: α , and feature matrix: M
 $M = I_{d \times d}$ ▷ Initialize M to be the identity matrix
for $t \in T$ **do**
 $K_{train} = K_M(X, X)$ ▷ $K_M(X, X)_{i,j} := K_M(x_i, x_j)$
 $\alpha = y K_{train}^{-1}$
 $M = \frac{1}{n} \sum_{x \in X} (\nabla f(x)) (\nabla f(x))^T$ ▷ $f(x) = \alpha K_M(X, x)$ with $K_M(X, x)_i := K_M(x_i, x)$
end for

³We note that in statistical literature this distance is defined by $d_M(x, z) = \sqrt{(x - z)^T M^{-1} (x - z)}$ [126], but here, we make use of the notation from metric learning literature [29], which omits the inverse. We additionally note that Mahanobis kernels can be extended to general, non-radial kernels by considering kernels of the form $K_M(x, z) = K(M^{\frac{1}{2}} x, M^{\frac{1}{2}} z)$.

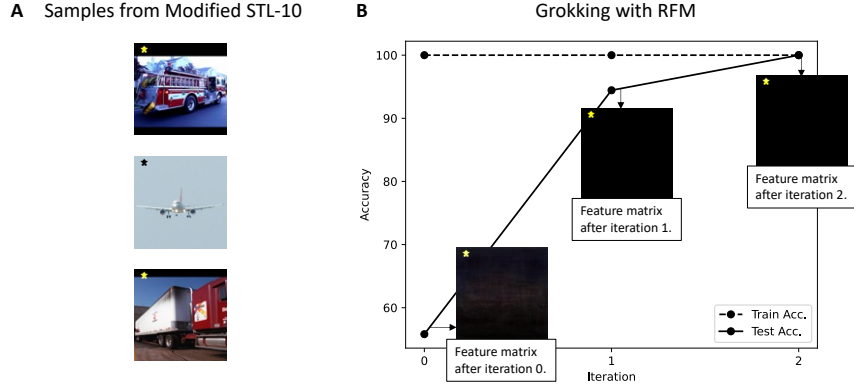


Figure 5-6: Grokking in RFMs trained on modified STL-10. (A) Samples from modified STL-10 (see Section 5.2.1 and Fig. 5-5 for dataset details). (B) While training accuracy is always 100%, iteration leads to a drastic rise in test accuracy from 55.8% to 100%. Feature matrices (shown as inserts) indicate that test accuracy improved since the RFM has learned the star pixels that give away the class label.

In Appendix B and Appendix Figs. D-5 and D-6, we compare features learned by RFMs and deep fully connected networks and demonstrate remarkable similarity between RFM features and first layer features of deep fully connected neural networks. We show that the correlation between the top eigenvector of the first layer NFM after training and that of the RFM feature matrix, M , is consistently greater than .99 for 12 different classification tasks from CelebA. We also show high correlation between RFM features and first layer NFM features for SVHN and low rank polynomial regression tasks from [189] and [52]. Lastly, in Appendix D, we discuss connections between RFMs and prior literature on kernel alignment [49, 192].

Given that RFMs use the same feature learning mechanism as neural networks, these models exhibit the deep learning phenomena discussed earlier, i.e., grokking, lottery tickets, and simplicity biases. In Fig. 5-6, we showcase that RFMs perform grokking on the same dataset used in Section 5.2.1 and Fig. 5-5. We show that RFMs exhibit lottery ticket and simplicity bias phenomena in Appendix Figs. D-7 and D-8.

5.2.3 Recursive Feature Machines provide state-of-the-art results on tabular data

We demonstrate the immediate practical value of the integrated feature learning mechanism by demonstrating that RFMs achieve state-of-the-art results on two tabular benchmarks containing over 150 datasets. The first benchmark we consider is from [61], which compares the performance of 179 different machine learning methods including neural networks, tree-based models, and kernel machines on 121 tabular classification tasks. In Fig. 5-7A, we show RFMs outperform these classification methods, kernel machines using the Laplace kernel, and kernel machines using the recently introduced Neural Tangent Kernel (NTK) [96] across the following commonly used performance metrics:

- Average accuracy: The average accuracy of the classifier across all datasets.
- P90/P95: The percentage of datasets on which the classifier obtained accuracy within 90%/95% of that of the best performing model.
- PMA: The percentage of the maximum accuracy achieved by a classifier averaged across all datasets.
- Friedman rank: The average rank of the classifier across all datasets.

We note that while some of the datasets contain up to 130,000 training examples, RFMs are computationally fast to train through the use of pre-conditioned linear system solvers such as EigenPro [124, 125]. Indeed, RFMs take 40 minutes to achieve these results while neural networks took 5 hours (both measurements are in wall time on a server with two Titan Xp GPUs). In Fig. 5-7B, we analyze the benefit of feature

A Performance across all 121 classification datasets from Fernández-Delgado et al. 2014

Classifier	Avg. Accuracy (%)	P90 (%)	P95 (%)	PMA (%)	Friedman Rank
RFM (Ours)	85.37	92.56	85.96	97.36 ± 4.04	17.79
Laplace Ridge Regression	83.76	90.08	74.38	95.95 ± 5.41	28.48
NTK Ridge Regression	82.70	85.95	68.60	94.84 ± 8.17	33.55
Random Forest*	81.96	83.47	68.60	93.48 ± 12.10	33.52
Gaussian SVM	81.81	82.35	69.75	93.21 ± 11.37	37.50
Neural Net	79.37	73.55	53.72	91.14 ± 12.81	44.13

*Best out of 179 methods from Fernández-Delgado et al. 2014

B RFM vs. Laplace Kernel Error

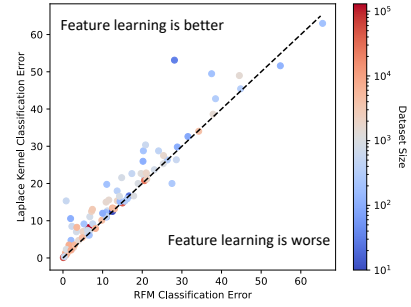


Figure 5-7: **(A)** Comparison of 182 models including RFMs, NTKs, random forests, and fully connected neural networks on 121 tabular datasets from [61]. All metrics, models, and training details are outlined in Appendix B. RFMs took 40 minutes to achieve these results while Neural Nets took 5 hours (both measurements are in wall time on a server with two Titan Xp GPUs). **(B)** To determine the benefit of feature learning, we compare the error rate (100% - accuracy) between RFMs and Laplace kernels, which are equivalent to RFMs without feature learning.

learning by comparing the difference in error (100% - accuracy) between RFMs and the classical Laplace kernel, which is equivalent to an RFM without feature learning. We observe that the Laplace kernel generally results in higher error than the RFM for larger datasets.

In Appendix Fig. D-9 and Tables D.1, D.2, D.3, and D.4 we additionally compare RFMs to two transformer models [163, 174], ResNet [80], and two gradient boosting tree models [41, 143] across regression and classification tasks on a second tabular benchmark [70]. Consistent with our findings on the first tabular benchmark, we observe that RFMs generally outperform tree-based models and neural networks at a fraction of the computational cost (3600 compute hours for RFMs while all other methods are with 20,000 compute hours).

5.2.4 Theoretical Evidence for Deep Neural Feature Ansatz

We now present theoretical evidence for the Deep Neural Feature Ansatz. A summary of all theoretical results for the Deep Neural Feature Ansatz is presented in Table 5.1.

Result	Activation	Steps	Depth	Outer layers	Initialization	GIA	# Samples
Proposition 5	Any	Any	Any	Fixed	Zero	No	1
Proposition 9	Any	1	Any	Fixed	Zero	No	Any
Proposition 10	Linear	Any	2	Fixed, i.i.d.	Zero	No	Any
Proposition 11	Linear	2	2	Trainable, i.i.d.	Zero	No	Any
Theorem 10	ReLU	Any	Any	Fixed, i.i.d.	Any	Yes	Any

Table 5.1: Settings for which we prove the Deep Neural Feature Ansatz. *Activation* refers the type of network activation function. *Steps* refers to the number of steps of gradient descent for which the proof holds. *Depth* refers to the depth of the neural network considered. *Outer layers* describes how the layers other than the first are initialized and trained. *Initialization* refers to the initialization method of the first layer weights. *GIA* indicates whether the gradient independence ansatz is required for the result to hold. *# Samples* denotes the number of training samples considered.

To provide intuition as to when the ansatz holds, we first analyze the general setting in which we train models, $f : \mathbb{R}^d \rightarrow \mathbb{R}$, of the form $f(x) = g(Bx)$ with $g : \mathbb{R}^k \rightarrow \mathbb{R}$ by updating the weight matrix B using gradient descent. This setting encompasses any type of neural network in which the first layer is fully

connected and the only trainable layer. We begin with Proposition 5 below (proof in Appendix A), which establishes the ansatz for such functions trained on one training example $(x, y) \in \mathbb{R}^d \times \mathbb{R}$.

Proposition 5. *Let $f(z) = g(Bz)$ with $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and $g : \mathbb{R}^k \rightarrow \mathbb{R}$. Given one training sample (x, y) , suppose that f is trained to minimize $\frac{1}{2}(y - f(x))^2$ using gradient descent. Let $B^{(\ell)}$ denote B after ℓ steps of gradient descent. If $B^{(0)} = \mathbf{0}$ and $f_t(z) := g(B^{(t)}z)$, then for all time steps t :*

$$\nabla f_t(z) \nabla f_t(z)^T \propto B^{(t)T} B^{(t)} .$$

Informally, this example demonstrates that feature learning is maximized when the ansatz holds. Namely, if initialization is nonzero, then $B^{(t)T} B^{(t)}$ contains a term corresponding to initialization $B^{(0)T} B^{(0)}$, which disrupts exact proportionality in the ansatz but also affects the quality of features learned. In the extreme case of non-feature learning models such as neural network Gaussian processes [136] where the first layer weights are drawn from a standard Gaussian distribution and fixed, then $B^{(t)T} B^{(t)}$ has rank almost surely $\min(k, d)$ while $\nabla f_t(z) \nabla f_t(z)^T$ is a rank 1 matrix. From this perspective, we argue that the ansatz provides a precise characterization of feature learning.

The difficulty in generalizing Proposition 5 to multiple steps is that the terms $\nabla g(B^{(t)}x_i)$ are no longer necessarily equal for all examples after 1 step. Thus, instead of proving the result for this general class of functions, we instead turn to classes of functions corresponding to fully connected networks. In particular, Theorem 10 (proof in Appendix A) establishes the ansatz in the more general setting of deep, nonlinear fully connected networks. Before stating this theorem, we introduce the relevant notation for deep neural networks and the gradient independence ansatz used in our proof.

Notation. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ denote an L hidden layer network with element-wise activation $\phi : \mathbb{R} \rightarrow \mathbb{R}$ of the form:

$$g(x) = a^\top \phi_{(L)}(x) \quad \phi_{(\ell)}(x) = \begin{cases} \phi \left(\sqrt{\frac{c_\phi}{k_\ell}} W_\ell \phi_{(\ell-1)}(x) \right) & \ell \in \{1, \dots, L\} \\ x & \ell = 0 \end{cases} \quad (5.1)$$

where $a \in \mathbb{R}^{k_L}$ and $W_\ell \in \mathbb{R}^{k_\ell \times k_{\ell-1}}$ for $\ell \in [L]$ with $k_\ell \in \mathbb{Z}_+$ and $k_0 = d$. We denote row k of weight matrix W_ℓ by $W_{\ell,k} \in \mathbb{R}^{d \times 1}$.

Gradient Independence Ansatz (GIA).⁴ *In computing gradients, whenever we multiply by a weight matrix, W_ℓ , we can instead multiply by an i.i.d. copy of W_ℓ without changing the gradient.*

We note that the GIA has been used in a range of works on analyzing neural networks including [40, 144, 165, 198, 202, 204, 205] and is implicit in the original NTK derivation [96].⁵ In [9], the authors rigorously prove the gradient independence ansatz for fully connected neural networks with ReLU activation functions.

Theorem 10. *Let f denote an L -hidden layer network with ReLU activation $\phi(z) = \max\{0, z\}$. Suppose we sample weights $a_{k'}, W_\ell$ for $\ell > 1$ in an i.i.d. manner so that $\mathbb{E}[a_{k'}^2] = 1$, $\mathbb{E}[W_{\ell,k''}^2] = 1$, $\mathbb{E}[a_{k'}] = 0$, and $\mathbb{E}[W_{\ell,k''}] = 0$. Suppose W_1 is fixed and arbitrary. Let $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$. If $x \sim \mathcal{N}(0, I_d)$, then*

$$\frac{1}{k_1} W_1^\top W_1 = \mathbb{E}_x \left[\lim_{k_2, \dots, k_L \rightarrow \infty} \mathbb{E}_a \left[\nabla_x f(x) \nabla_x f(x)^\top \right] \right] .$$

Theorem 10 can be directly applied in a recursive fashion to prove the ansatz holds for any layer of a deep network. In particular, we simply consider the layer-wise training scheme in which we show the ansatz holds for the first layer, and then fix the first layer and apply Theorem 10 to the second layer and proceed inductively.

⁴This is often called an assumption in the literature (e.g., Assumption 2.2 in [202]). We prefer the word ansatz following the terminology from [33], as this is a simplifying principle rather than a true mathematical assumption.

⁵This condition is required for establishing the closed form of the deep NTK presented in [96] as observed in [202] but is not needed to establish transition to linearity (e.g., [118]).

5.3 Discussion

Summary of results. Characterizing the mechanism of neural feature learning has been an unresolved problem that is key to advancing performance and interpretability of neural networks. In this work we posited the Deep Neural Feature Ansatz, which stated that neural feature learning occurs by up-weighting the features most influential on model output, a process that is formulated mathematically in terms of the average gradient outer product. Our ansatz unified previous lines of investigation into neural feature learning and explained various deep learning phenomena. An important insight from our ansatz was that the average gradient outer product could instead be used to learn features with any machine learning model. We showcased the power of this insight by using it to enable feature learning with classical, non-feature learning models known as kernel machines. The resulting algorithm, which we referred to as Recursive Feature Machines, achieved state-of-the-art performance on tabular benchmarks containing over 150 classification and regression tasks.

Connections and implications. We conclude with a discussion of connections between our results and machine learning literature as well as implications of our results.

Advancing interpretability in deep learning. A key area of practical interest is understanding and interpreting how neural networks make predictions. There is a rich literature on gradient-based methods for understanding features used by deep networks for image classification tasks [167, 170, 207]. These methods utilize gradients of trained networks to identify patterns that are important for prediction in a single data point. Rather than focus on features relevant for individual samples, our ansatz directly provides a characterization of Neural Feature Matrices, which capture features the network selects across all data points. We demonstrated how our ansatz shed light on the emergence of spurious features in neural networks and how it could be leveraged to identify such spurious features. We envision that the transparency provided by our ansatz can serve as a key tool for increasing interpretability and mitigating biases of neural networks more generally.

Building state-of-the-art models at a fraction of the cost by streamlining feature learning. Neural networks simultaneously learn a predictor and features through backpropagation. While such simultaneous learning is a remarkable aspect of neural networks, our ansatz shows that it can also lead to inefficiencies during training. For example, in the initial steps of training, the features are selected based on a partially trained predictor, and the resulting features can be uninformative. To streamline the feature learning process, we showed that we can instead train a predictor and then estimate features directly via the average gradient outer product. This approach has already led to an improvement in performance and a reduction in time in our experiments, specifically from 5 hours for training neural networks to 40 minutes for RFM across 121 tabular data tasks from [61]. A natural next direction is to extend this direct feature learning approach for fully connected networks to streamline training of general network architectures including convolutional networks, graph neural networks, and transformers. We envision that using the average gradient outer product as an alternative to backpropagation could reduce the sizeable training costs associated with state-of-the-art models, including large language models for which fully connected networks form the backbone.

The role of width: Transition to linearity vs. feature learning. Under the NTK initialization scheme, very wide neural networks undergo a transition to linearity and implement kernel regression with a kernel function that is not data adaptive and depends entirely on the network architecture [96, 118]. On the other hand, more narrow neural networks simultaneously learn both a predictor and features. Thus, network width modulates between two different regimes: one in which networks implement non-data-adaptive kernel predictors and another in which networks learn features. While this remarkable property highlights the flexibility of deep networks, it also illustrates their complexity. Indeed, simply increasing width under a particular initialization scheme increases the representational power of a neural network while decreasing its ability to learn features. In contrast, by separating predictor learning and feature learning into separate subroutines, we can circumvent such modelling complexity without sacrificing performance.

The role of depth. Our Deep Neural Feature Ansatz provided a way to capture features at deeper layers of a fully connected network by using the average gradient outer product. Yet, our implementation of RFMs leveraged this feature learning mechanism to capture only features of the input, which corresponded to the

first layer features learned by fully connected networks. Interestingly, despite only using first layer features, RFMs provided state-of-the-art results on tabular datasets, matching or outperforming deep fully connected networks across a variety of tasks. Thus, an interesting direction of future work is to understand the exact nature of deep feature learning and to characterize the architectures, datasets, and settings for which deep feature learning is beneficial.

Empirical NTK. Recently, a line of works have studied the connection between kernel learning and neural networks through the time-dependent evolution of the NTK [63]. An insightful work [123] showed that the *after kernel*, i.e. the empirical NTK at the end of training, matches the performance of the original network. Interestingly, [12] highlighted another benefit of empirical NTK by showing that as a neural network is trained, the empirical NTK increases alignment with the ideal kernel matrix. Given the similarity in features learned between RFMs and neural networks, we believe that RFMs may be an effective means of approximating the after kernel without training neural networks.

Connections to other statistical and machine learning methods. Our result connects neural feature learning to a number of classical methods from statistics and machine learning, which we discuss below.

- *Supervised dimension reduction.* The problem of identifying key variables necessary to understand the response function (called sufficient dimensionality reduction in [112]) has been investigated in depth in the statistical literature. In particular, estimates of the gradient of the target function can be used to identify relevant coordinates for the target function [88, 197]. A series of works proposed methods that simultaneously learn the regression function and its gradient by non-parametric estimation [132, 133]. The gradients can then be used to improve performance on downstream prediction tasks [104]. Gradient estimation is particularly useful for coordinate selection in *multi-index models*, for which the regression function f^* has the form $f^*(x) = g(Ux)$, where U is a low rank matrix. Similar to neural feature learning, the multi-index estimator in [88] iteratively identifies the relevant subspace by learning the regression function and its gradient, but makes use of kernel smoothers. A line of recent work identifies the benefits of using neural networks for such multi-index (or single-index) problems by analyzing networks after 1 step of gradient descent [13, 52] or showing that networks identify the principal subspace, U , through multiple steps of training [31, 131]. Another line of work [113] estimates the Principal Hessian Directions, the eigenvectors of the average Hessian matrix of the target function, to identify relevant coordinates. Finally, a parallel line of research on “active subspace” methods in the context of *dynamical systems* has recently become a topic of active investigation [46].
- *Metric and manifold learning.* Updating feature matrices can also be viewed as learning a data-dependent Mahalanobis distance, i.e. a distance $d_M(x, z) = \sqrt{(x - z)^T M (x - z)}$ where M is the feature matrix. This connects to a large body of literature on metric learning with numerous applications to various supervised and unsupervised learning problems [29]. Furthermore, we believe that feature learning methods such as neural networks or RFMs may benefit from incorporating ideas from the unsupervised and semi-supervised manifold learning and nonlinear dimensionality reduction literature [25, 162]. We also note that some of the early work on Radial Basis Networks explicitly addressed metric learning as a part of kernel function construction [147].
- *FisherFaces and EigenFaces.* We further note the strong similarity between the eigenvectors of feature matrices (e.g., Figs. 5-1, D-5) analyzed in this work and those given by EigenFace [173, 184], and FisherFace [20] algorithms. While EigenFaces are obtained in a purely unsupervised fashion, the FisherFace algorithm uses labeled images of faces and Fisher’s Linear Discriminant [62] to learn a linear subspace for dimensionality reduction. The first layer of neural networks and RFMs also learn linear subspaces based on labeled data but in a recursive way, using nonlinear classifiers.
- *Debiasing.* Debiasing is a statistical procedure of recent interest in the statistics literature [210]. Given a high-dimensional problem with a hidden low-dimensional structure, debiasing involves first performing variable selection by using methods such as Lasso [180] or sparse PCA [98] and then fitting a low-dimensional model to the selected coordinates. We note that this procedure is similar to a single step of RFM. Moreover, both RFMs and neural networks can be viewed as a non-linear iterative version of the debiasing procedure with soft coordinate selection.

- *Expectation Maximization (EM)*. The RFM algorithm is reminiscent of the EM algorithm [130] with alternating estimation of the kernel predictor (M-step) and the feature matrix M (E-step). From this viewpoint, developing estimators for the feature matrix other than the sample covariance estimator considered in this work is an interesting future direction. Moreover, depending on properties of the data and the target function, the feature matrix may be structured. Such structure could be leveraged to develop more sample efficient estimators for the M-step.
- *Boosting*. The mechanism of neural feature learning is reminiscent of boosting [65] where a “weak learner,” only slightly correlated with the optimal predictor, is “boosted” by repeated application. Feature learning can similarly improve a suboptimal predictor as long as its average gradient outer product estimate is above the noise level.

Looking forward. Overall, our work provides new insights into the operational principles of neural networks and how such principles can be leveraged to design new models with improved performance, computational simplicity, and transparency. We envision that the mechanism of neural feature learning identified in this work will be key to improving neural networks and developing such new models.

Data Availability

All image datasets considered in this work, i.e., CelebA, SVHN, CIFAR10, MNIST and STL-10, are publicly available for download via PyTorch. Tabular data from [61] is available to download via <https://github.com/LeoYu/neural-tangent-kernel-UCI> provided by [10]. Tabular data from [70] is available to download via <https://github.com/LeoGrin/tabular-benchmark>.

Code Availability

Code for neural network experiments is available at https://github.com/aradha/deep_neural_feature_ansatz. Code for RFMs is available at https://github.com/aradha/recursive_feature_machines/tree/pip_install.

Chapter 6

Summary and Discussion

Summary. In this thesis, we identified over-parameterization and feature learning as two core principles driving the success of deep learning. While the benefits of over-parameterization are well studied for supervised learning tasks in the literature, we analyzed the impact of over-parameterization in unsupervised learning. We observed that over-parameterized autoencoders learned representations that were useful for downstream tasks such as drug re-purposing [30]. Contrary to the long-held belief that such models would learn the identity function through training, we showed that these models in fact learn functions that are contractive around the training examples [152].

Given the ubiquitous benefits of over-parameterization in both supervised and unsupervised learning, we then demonstrated the effectiveness of using infinitely wide neural networks corresponding to kernel machines. In particular, we derived the NTK of infinitely wide neural networks for matrix completion problems and showed that these models achieve state-of-the-art performance for virtual drug screening and are competitive with convolutional networks for image inpainting [154]. In addition to its practical value, we used the NTK to provide theoretical guarantees for neural networks by constructing infinitely wide and deep networks that were Bayes optimal for classification [153].

While the NTK is effective in a number of applications, these methods rely on using fixed feature transformations independent of the dataset. On the other hand, neural networks are able to learn features automatically from data, thereby giving them an edge over kernel methods on certain tasks. We thus identified the mechanism of neural feature learning. We posited the Deep Neural Feature Ansatz, which stated that neural feature learning occurred through a procedure known as the average gradient outer product [151]. We used our ansatz to explain prominent deep learning phenomena such as the emergence of spurious features in neural networks.

Lastly, the feature learning mechanism identified in our work provided a means of introducing feature learning into any machine learning model. Given that non-feature learning kernel machines were already effective on a number of tasks, these models were prime candidates for improvement with feature learning. We thus introduced feature learning into these models and referred to the resulting algorithm as Recursive Feature Machines (RFMs). We showed the immediate practical value of RFMs by using them to achieve state-of-the-art results on tabular datasets.

Next Steps. Identifying the core principles behind the success of deep networks provides a path forward for building the next generation of machine learning models that are effective, computationally inexpensive, and interpretable. We presented an example of the power of this philosophy through the construction of RFMs, which were cost effective, achieved state-of-the-art results on tabular data, and provided interpretability through the form of a feature matrix. We now discuss next directions for applying and improving these methods.

Biomedical applications. In this thesis, we briefly touched on some applications of over-parameterized models in biomedical applications such as drug re-purposing and virtual drug screening. By integrating feature learning into these models we can achieve effective results in this domain and also provide new biological insights. In particular, simply visualizing feature matrices for predictive modeling tasks in genetics or genomics

would already provide further insight into relationships between phenotypes and genotypes. Moreover, it would be interesting to understand the features learned by unsupervised models in this domain, particularly when integrating different data modalities.

Advancing understanding of the role of depth. The NTK and the average gradient outer product are useful tools for understanding properties of trained deep networks. For example, we used the NTK to understand inductive biases of networks for matrix completion tasks, and we used the average gradient outer product to explain simplicity biases, grokking, and lottery tickets in deep networks. We envision that these tools will further our understanding of the role of depth in neural networks as well. In particular, an interesting direction of future work is to use these tools to characterize when deep feature learning is beneficial over simply increasing depth with a random feature model.

Libraries for over-parameterized, feature learning models. The prevalence of deep networks is driven in large part by deep learning libraries such as PyTorch [142]. These libraries provide simple building blocks for rapidly constructing and testing novel deep learning architectures. While such libraries are still emerging for kernel machines [139], an important direction of future work is to integrate feature learning and faster kernel regression solvers into these libraries. We envision that such integration would lead to greater adoption of these effective, interpretable, and conceptually simple methods by practitioners.

Appendix A

Chapter 2 Supplementary

A Encoding Multiple Sequences

Given sequences of training examples $\{x_i^{(j)}\} \in \mathbb{R}^d$ for $i \in [n], j \in [k_i], k_i \in \mathbb{Z}_{\geq 0}$, minimizing the following *sequence encoding* objective with gradient descent methods leads to training sequences being stored as limit cycles:

$$\arg \min_{f \in \mathcal{F}} \sum_{i=1}^n \sum_{j=1}^{k_i} \|f(x_i^{(j \bmod k_i + 1)}) - x_i^{(j)}\|_2^2. \quad (\text{A.1})$$

B Proofs of Invariants, Theorem 1, and Theorem 2

We present the full proof of Invariants 1 and 2, Theorem 1, and Theorem 2 below. To simplify notation, we use A to represent W_1 and B to represent W_2 .

Let $f(z) = A\phi(Bz)$ represent a 1-hidden layer network with elementwise, differentiable nonlinearity ϕ , $z \in \mathbb{R}^{k_0}$, $B \in \mathbb{R}^{k \times k_0}$, and $A \in \mathbb{R}^{k_0 \times k}$. Suppose that gradient descent with learning rate γ is used to minimize the following loss for the autoencoding problem with 1 training example x :

$$\mathcal{L}(x, f) = \frac{1}{2} \|x - f(x)\|_2^2. \quad (\text{A.2})$$

Gradient descent updates on A and B are as follows:

$$A^{(t+1)} = A^{(t)} + \gamma(x - A^{(t)}\phi(B^{(t)}x))\phi(B^{(t)}x)^T \quad (\text{A.3})$$

$$B^{(t+1)} = B^{(t)} + \gamma \text{diag}(\phi'(B^{(t)}x))A^{(t)T}(x - A^{(t)}\phi(B^{(t)}x))x^T, \quad (\text{A.4})$$

where $\text{diag}(\phi'(B^{(t)}x)) = \begin{bmatrix} \phi'(B_{1,:}^{(t)}x) & & \\ & \ddots & \\ & & \phi'(B_{k,:}^{(t)}x) \end{bmatrix}$ with $B_{i,:}^{(t)}$ representing row i of matrix $B^{(t)}$.

In the following, we restate and prove Invariant 1 formally:

Invariant 1. *If $A^{(0)} = xa^{(0)T}$ and $B^{(0)} = b^{(0)}x^T$ for $a^{(0)}, b^{(0)} \in \mathbb{R}^k$, then for all time-steps t , $A^{(t)} = xa^{(t)T}$ and $B^{(t)} = b^{(t)}x^T$ for $a^{(t)}, b^{(t)} \in \mathbb{R}^k$.*

Proof. We provide a proof by induction. The base case follows for $t = 0$ from the initialization. Now we

assume for some t that $A^{(t)} = xa^{(t)T}$ and $B^{(t)} = b^{(t)}x^T$. Then for time $t + 1$ we have:

$$\begin{aligned} A^{(t+1)} &= A^{(t)} + \gamma(x - A^{(t)}\phi(B^{(t)}x))\phi(B^{(t)}x)^T \\ &= xa^{(t)T} + \gamma(x - xa^{(t)T}\phi(B^{(t)}x))\phi(B^{(t)}x)^T \\ &= x[a^{(t)T} + \gamma(1 - a^{(t)T}\phi(B^{(t)}x))\phi(B^{(t)}x)^T] \\ &= xa^{(t+1)T} \end{aligned}$$

and similarly,

$$\begin{aligned} B^{(t+1)} &= B^{(t)} + \gamma \text{diag}(\phi'(B^{(t)}x))A^{(t)T}(x - A^{(t)}\phi(B^{(t)}x))x^T \\ &= b^{(t)}x^T + \gamma \text{diag}(\phi'(B^{(t)}x))A^{(t)T}(x - A^{(t)}\phi(B^{(t)}x))x^T \\ &= [b^{(t)} + \gamma \text{diag}(\phi'(B^{(t)}x))A^{(t)T}(x - A^{(t)}\phi(B^{(t)}x))]x^T \\ &= b^{(t+1)}x^T. \end{aligned}$$

Hence, since the statement holds for $t + 1$, it holds for all time steps. \square

Under the initialization in Invariant 1, outputs of the network are multiples of the training example. Generalizing this result, in Materials and Methods D, we prove that autoencoders trained on multiple examples produce outputs in the span of the training data.

Using Invariant 1, any interpolating solution satisfies the following condition.

Proposition 6. *Under the initialization in Invariant 1, if $\|x\|_2 = 1$ and $A^{(\infty)}, B^{(\infty)}$ yield zero training error, then $A^{(\infty)} = xa^T, B^{(\infty)} = bx^T$ for $a, b \in \mathbb{R}^k$ such that $a^T\phi(b) = 1$.*

Proof. From Invariant 1, it holds that $A^{(\infty)} = xa^T, B^{(\infty)} = bx^T$. If the loss is minimized to zero, then $f(x) = x$, and thus:

$$x = xa^T\phi(bx^T x) = xa^T\phi(b) \rightarrow a^T\phi(b) = 1,$$

which completes the proof. \square

Using Proposition 6, we can compute the maximum eigenvalue of the Jacobian at training example x (denoted $\lambda_1(\mathbf{J}(f(x)))$). This is done in the following result.

Proposition 7. *Let the Hadamard product \odot denote coordinate-wise multiplication. Then $\lambda_1(\mathbf{J}(f(x))) = a^T\phi'(b) \odot b$.*

Proof. Note that

$$\mathbf{J}(f(z)) = xa^T(\phi'(bx^T z) \odot b)x^T = xa^T(\phi'(b) \odot b)x^T,$$

and hence $\mathbf{J}(f(x))x = xa^T(\phi'(b) \odot b)$. This implies that x is an eigenvector of $\mathbf{J}(f(x))$ with eigenvalue $a^T(\phi'(b) \odot b)$. Since $xa^T(\phi'(b) \odot b)x^T$ is rank 1, the remaining eigenvalues of $\mathbf{J}(f(x))$ are all zero, which completes the proof. \square

We now prove Invariant 2, which states that if, in addition to Invariant 1, all weights in each row of A, B are initialized to be equal, then they remain equal throughout training.

Invariant 2. *Let $\|x\|_2 = 1$ and $A^{(0)} = xa^{(0)T}, B^{(0)} = b^{(0)}x^T$ for vectors $a^{(0)}, b^{(0)} \in \mathbb{R}^k$. If $a_i^{(0)} = a_1^{(0)}, b_i^{(0)} = b_1^{(0)}$, then $a_i^{(t)} = a_1^{(t)}, b_i^{(t)} = b_1^{(t)}$ for all $i \in [k]$.*

Proof. From the proof of Invariant 1, we have that:

$$\begin{aligned} a^{(t+1)T} &= a^{(t)T} + \gamma(1 - a^{(t)T}\phi(b^{(t)}))\phi(b^{(t)})^T \\ b^{(t+1)} &= b^{(t)} + \gamma \text{diag}(\phi'(b^{(t)}))a^{(t)T}(1 - a^{(t)T}\phi(b^{(t)})). \end{aligned}$$

Now, if $a_i^{(t)} = a_1^{(t)}, b_i^{(t)} = b_1^{(t)}$, then from the above this implies that $a_i^{(t+1)} = a_1^{(t+1)}, b_i^{(t+1)} = b_1^{(t+1)}$ and the proof follows by induction. \square

Using Invariants 1 and 2, we can now prove Theorem 1.

Theorem 11. *Let $f(z) = A\phi(Bz)$ denote a 1-hidden layer network with elementwise nonlinearity ϕ , $z \in \mathbb{R}^{k_0}$, $B \in \mathbb{R}^{k \times k_0}$, and $A \in \mathbb{R}^{k_0 \times k}$. Let A, B be initialized as in Invariant 1, 2. Gradient descent with learning rate γ is used to minimize the following loss for 1 training example $x \in \mathbb{R}^{k_0}$ with $\|x\|_2 = 1$*

$$\mathcal{L}(x, f) = \frac{1}{2} \|x - f(x)\|_2^2. \quad (\text{A.5})$$

Assuming $\frac{\phi(z)}{\phi'(z)} < \infty \forall z \in \mathbb{R}$, then as the learning rate $\gamma \rightarrow 0$, it holds that $A^{(\infty)} = xa^T$ and $B^{(\infty)} = bx^T$ with $a, b \in \mathbb{R}^k$ such that

$$\frac{a_i^2 - a_i^{(0)2}}{2} = \int_{b_i^{(0)}}^{b_i} \frac{\phi(z)}{\phi'(z)} dz \quad ; \quad a_i \phi(b_i) = \frac{1}{k}$$

for all $i \in [k]$ with $a_i = a_j, b_i = b_j$ for all $i, j \in [k]$.

Proof. From Invariants 1 and 2, we have that:

$$\begin{aligned} a_1^{(t+1)} &= a_1^{(t)} + \gamma(1 - ka_1^{(t)}\phi(b_1^{(t)}))\phi(b_1^{(t)}) \\ b_1^{(t+1)} &= b_1^{(t)} + \gamma\phi'(b_1^{(t)})a_1^{(t)}(1 - ka_1^{(t)}\phi(b_1^{(t)})). \end{aligned}$$

Rearranging the above, we obtain

$$\begin{aligned} \frac{a_1^{(t+1)} - a_1^{(t)}}{\phi(b_1^{(t)})} &= \frac{b_1^{(t+1)} - b_1^{(t)}}{a_1^{(t)}\phi'(b_1^{(t)})} \\ \implies a_1^{(t)} \frac{da_1^{(t)}}{dt} &= \frac{\phi(b_1^{(t)})}{\phi'(b_1^{(t)})} \frac{db_1^{(t)}}{dt} \quad \text{as } \gamma \rightarrow 0 \\ \implies \int_0^{t'} a_1^{(t)} \frac{da_1^{(t)}}{dt} dt &= \int_0^{t'} \frac{\phi(b_1^{(t)})}{\phi'(b_1^{(t)})} \frac{db_1^{(t)}}{dt} dt \\ \implies \frac{a_1^{(t')2} - a_1^{(0)2}}{2} &= \int_{b_1^{(0)}}^{b_1^{(t')}} \frac{\phi(z)}{\phi'(z)} dz, \end{aligned}$$

wich completes the proof. \square

Using Proposition 7 and Theorem 1 above, we can calculate the values of a_1, b_1 explicitly. After computing b_1 , the following result can be used to compute $\lambda_1(\mathbf{J}(f(x)))$.

Theorem 12. *Under the setting of Theorem 1 it holds that*

$$\lambda_1(\mathbf{J}(f(x))) = \frac{\phi'(b_1)b_1}{\phi(b_1)}.$$

Example. *From Theorem 1, if $\phi(z) = e^{2z}$, then the values of a_1, b_1 are given by solving:*

$$\begin{aligned} a_1^2 &= b_1, \\ \sqrt{b_1}e^{2b_1} &= \frac{1}{k}. \end{aligned}$$

From Theorem 12, the top eigenvalue of the Jacobian at the training example is $2b_1$. Note that the above equation implies b_1 decreases as k increases. At $k = 1, \lambda_1 = .6$, at $k = 2, \lambda_1 = .28$, and at $k = 3, \lambda_1 = .16$. This nonlinearity guarantees that training example is an attractor. Moreover, as there are no fixed points other than the training example, there are no spurious attractors.

Remark. The analysis of Theorem 12 implies that attractors arise as a consequence of training and are not simply consequences of interpolation by a neural network with a certain architecture; see the following corollary.

Corollary 3. Let $x \in \mathbb{R}^{k_0}$ with $\|x\|_2 = 1$ and $f(z) = xa^T\phi(bx^Tz)$, where $a, b \in \mathbb{R}^k$ and ϕ is a smooth element-wise nonlinearity with $\frac{\phi'(z)}{\phi(z)} < \infty$ for all $z \in \mathbb{R}$, $\left|\frac{\phi'(z)z}{\phi(z)}\right| > 1$ for z in an open interval $\mathcal{O} \subset \mathbb{R}$. Then there exist infinitely many $b \in \mathbb{R}^k$, such that $f(x) = x$ and x is not an attractor for f .

Proof. If $a_i = a_j$ and $b_i = b_j$ for all $i, j \in [k]$, then Propositions 6 and 7 imply that $f(x) = x$ if:

$$a_i\phi(b_i) = \frac{1}{k}$$

$$\lambda_1(\mathbf{J}(f(x))) = \frac{\phi'(b_i)b_i}{\phi(b_i)}.$$

However, for any value of b_i such that $\phi(b_i) \neq 0$, we can select a value of a_i such that $a_i\phi(b_i) = \frac{1}{k}$. Hence, we just select appropriate a_i such that $a_i\phi(b_i) = \frac{1}{k}$ for $b_i \in \mathcal{O}$. \square

C Analysis of Deep Autoencoders with 1 Training Example

Let $f(z) = W_d\phi(W_{d-1}\phi(W_{d-2}\dots\phi(W_1z)\dots))$ represent a $d - 1$ hidden layer network with elementwise nonlinearity ϕ with $z \in \mathbb{R}^{k_0}, W_i \in \mathbb{R}^{k_i \times k_{i-1}}$, and $k_d = k_0$. We again consider the setting where gradient descent with learning rate γ is used to minimize the square loss on 1 training example x . As in the 1 hidden layer case, we derive invariants of training that allow us to derive a closed form solution when training on 1 example in the gradient flow setting.

Firstly, the following invariant (analogous to Invariant 1) holds in the deep setting.

Invariant 3. If $W_d^{(0)} = xa^{(0)T}$ and $W_1^{(0)} = b^{(0)}x^T$ for $b^{(0)}, a^{(0)} \in \mathbb{R}^{k_1}, \mathbb{R}^{k_{d-1}}$ respectively, then for all time-steps t , $W_d^{(t)} = xa^{(t)T}$ and $W_1^{(t)} = b^{(t)}x^T$ for $b^{(t)}, a^{(t)} \in \mathbb{R}^{k_1}, \mathbb{R}^{k_{d-1}}$ respectively.

The proof is analogous to that of Invariant 1. Now that we have invariants of training for layers W_d, W_1 , we still need to find an invariant for the intermediate layers W_2, \dots, W_{d-1} . The following invariant extends Invariant 2 to the deep setting.

Invariant 4. Assume $\|x\|_2 = 1$ and $W_d^{(0)} = xa^{(0)T}, W_1^{(0)} = b^{(0)}x^T$ for $b^{(0)}, a^{(0)} \in \mathbb{R}^{k_1}, \mathbb{R}^{k_{d-1}}$ respectively. Let $\mathbf{1}_{m \times n}$ denote the $m \times n$ matrix of all 1's. If $a_j^{(0)} = a_1^{(0)}, b_l^{(0)} = b_1^{(0)}, W_i^{(0)} = w_i^{(0)}\mathbf{1}_{k_i \times k_{i-1}}$ with $w_i^{(0)} \in \mathbb{R}$, then for all time-steps t , $a_j^{(t)} = a_1^{(t)}, b_l^{(t)} = b_1^{(t)}, W_i^{(t)} = w_i^{(t)}\mathbf{1}_{k_i \times k_{i-1}}$ for $j \in \mathbb{R}^{k_{d-1}}, l \in \mathbb{R}^{k_1}$, and $i \in \{2, \dots, d-1\}$.

That is, in the deep setting, the intermediate layers remain rank 1 throughout training, if they are initialized to be a constant times the all 1's matrix. The proof follows by induction and is analogous to the proof of Invariant 4.

Theorem. Let $f(z) = W_d\phi(W_{d-1}\phi(W_{d-2}\dots\phi(W_1z)\dots))$ denote a $d - 1$ hidden layer network with elementwise nonlinearity ϕ , with $z \in \mathbb{R}^{k_0}, W_i \in \mathbb{R}^{k_i \times k_{i-1}}$, and $k_d = k_0$. Let $\{W_i\}_{i=1}^d$ be initialized as in Invariants 3, 4. Gradient descent with learning rate γ is used to minimize the following loss for 1 training example $x \in \mathbb{R}^{k_0}$ with $\|x\|_2 = 1$:

$$\mathcal{L}(x, f) = \frac{1}{2}\|x - f(x)\|_2^2 \tag{A.6}$$

Assuming $\frac{\phi(z)}{\phi'(z)} < \infty \forall z \in \mathbb{R}$, then as the learning rate $\gamma \rightarrow 0$, we have the following relationships between the weights $a_1^{(t)}, b_1^{(t)}, w_i^{(t)} \in \mathbb{R}$ for $i \in \{1, \dots, d-2\}$:

$$\begin{aligned} \frac{w_1^{(t')^2} - w_1^{(0)2}}{2} &= \int_{b_1^{(0)}}^{b_1^{(t')}} \frac{\phi(b_1)}{\phi'(b_1)k_2} db_1, \\ \frac{w_{i+1}^{(t)2} - w_{i+1}^{(0)2}}{2} &= \int_{w_i^{(0)}}^{w_i^{(t)}} \frac{\phi(w_i k_i \phi(w_{i-1} k_{i-1} \dots \phi(b_1) \dots))}{\phi'(w_i k_i \phi(w_{i-1} k_{i-1} \dots \phi(b_1) \dots))} \frac{1}{\phi(w_{i-1} k_{i-1} \dots \phi(b_1) \dots) k_{i+1}} dw_i, \\ \frac{a_1^{(t)2} - a_1^{(0)2}}{2} &= \int_{w_{d-1}^{(0)}}^{w_{d-2}^{(t)}} \frac{\phi(w_{d-1} k_{d-1} \phi(w_{d-2} k_{d-2} \dots \phi(b_1) \dots))}{\phi'(w_{d-1} k_{d-1} \phi(w_{d-2} k_{d-2} \dots \phi(b_1) \dots))} \frac{1}{\phi(w_{d-2} k_{d-2} \dots \phi(b_1) \dots)} dw_{d-1}. \end{aligned}$$

The proof is analogous to the proof of Theorem 1. In addition, analogously to Theorem 12, we can explicitly compute the maximum eigenvalue of the Jacobian at the training example x using the following corollary.

Corollary 4. *Under the setting of the theorem above, it holds that*

$$\lambda_1(\mathbf{J}(f(x))) = \frac{\left(\prod_{i=2}^{d-1} \phi'(w_i k_{i-1} \phi(w_{i-1} k_{i-2} \dots \phi(b_1) \dots)) w_i k_{i-1} \right) \phi'(b_1) b_1}{\phi(w_{d-1} k_{d-2} \phi(w_{d-2} k_{d-3} \dots (\phi(b_1)) \dots))}$$

The proof is analogous to that of Theorem 12. Note that in the theorem above, the integration for later layer weights becomes increasingly complicated, since the integration depends on the values of the previous weights. Fortunately, for certain nonlinearities, the integral is tractable; see the example below.

Example. *Let $\phi(x) = x^m$ and assume $k_i = 1$ for $i \in [d-1]$. Then from the theorem above, it holds that:*

$$w_{i+1}^2 = \frac{w_i^2}{m} \quad \forall i \in [d-1]$$

From the corollary above, $\lambda_1(\mathbf{J}(f(x))) = m^{d-1}$. Hence if $m < 1$, then x becomes an attractor.

Remarks. In the 1 hidden layer setting, regardless of how large the width, $\lambda_1(\mathbf{J}(f(x))) = m$ when using $\phi(z) = z^m$. Thus, the example above demonstrates that depth can make over-parameterized autoencoders more contractive even when width cannot.

D Trained Autoencoders Produce Outputs in the Span of the Training Data

In the following section, we generalize Invariant 1 to the setting with multiple training examples and thus, demonstrate that trained autoencoders produce outputs in the span of the training data.

Invariant 5. *Let $f(z) = A\phi(Bz)$ represent a 1-hidden layer network with elementwise, differentiable non-linearity ϕ , $z \in \mathbb{R}^{k_0}$, $B \in \mathbb{R}^{k \times k_0}$, and $A \in \mathbb{R}^{k_0 \times k}$. Suppose that gradient descent with learning rate γ is used to minimize the following loss for the autoencoding problem with n training examples $\{x^{(i)}\}_{i=1}^n$:*

$$\mathcal{L}(x, f) = \frac{1}{2} \|x^{(i)} - f(x^{(i)})\|_2^2. \quad (\text{A.7})$$

If $A^{(0)} = \sum_{i=1}^n x^{(i)} a_i^{(0)T}$ and $B^{(0)} = \sum_{i=1}^n b_i^{(0)} x^{(i)T}$ for vectors $a_i^{(0)}, b_i^{(0)} \in \mathbb{R}^k$, then for all time-steps t , it holds that $A^{(t)} = \sum_{i=1}^n x^{(i)} a_i^{(t)T}$ and $B^{(t)} = \sum_{i=1}^n b_i^{(t)} x^{(i)T}$ for some vectors $a_i^{(t)}, b_i^{(t)} \in \mathbb{R}^k$.

Proof. The proof exactly follows the proof of Invariant 1. For completeness, we show the proof for $A^{(t)}$ below. We again provide a proof by induction. The base case follows for $t = 0$ from the initialization. Now we assume for some t that $A^{(t)} = \sum_{i=1}^n x^{(i)} a_i^{(t)T}$ and $B^{(t)} = \sum_{i=1}^n b_i^{(t)} x^{(i)T}$. Then for time $t + 1$ we have:

$$\begin{aligned} A^{(t+1)} &= A^{(t)} + \gamma \sum_{i=1}^n (x^{(i)} - A^{(t)} \phi(B^{(t)} x^{(i)})) \phi(B^{(t)} x^{(i)})^T \\ &= \sum_{i=1}^n x^{(i)} a_i^{(t)T} + \gamma \sum_{i=1}^n (x^{(i)} - \sum_{j=1}^n x^{(j)} a_j^{(t)T} \phi(B^{(t)} x^{(i)})) \phi(B^{(t)} x^{(i)})^T \\ &= \sum_{\ell=1}^n x^{(\ell)} a_{\ell}^{(t+1)T}. \end{aligned}$$

The proof for $B^{(t)}$ follows analogously. Hence, since the statement holds for $t + 1$, it holds for all time steps, which completes the proof. \square

E Proofs of Theorem 3 and 4

By analyzing sequence encoders as a composition of maps, we prove that sequence encoding provides a more efficient mechanism for memory than autoencoding. We begin by restating Theorem 3 below.

Theorem 13. *Let $f(z) = W_1 \phi(W_2 z)$ denote a 1-hidden layer network with elementwise nonlinearity ϕ and weights $W_1 \in \mathbb{R}^{k_0 \times k}$ and $W_2 \in \mathbb{R}^{k \times k_0}$, applied to $z \in \mathbb{R}^{k_0}$. Let $x^{(i)}, x^{(i+1)} \in \mathbb{R}^{k_0}$ be training examples with $\|x^{(i)}\|_2 = \|x^{(i+1)}\|_2 = 1$. Assuming that $\frac{\phi(z)}{\phi'(z)} < \infty \forall z \in \mathbb{R}$ and there exist $u^{(0)}, v^{(0)} \in \mathbb{R}^k$ such that $W_1^{(0)} = x^{(i+1)} u^{(0)T}$ and $W_2^{(0)} = v^{(0)} x^{(i)T}$ with $u_i^{(0)} = u_j^{(0)}, v_i^{(0)} = v_j^{(0)} \forall i, j \in [k]$, then gradient descent with learning rate $\gamma \rightarrow 0$ applied to minimize*

$$\mathcal{L}(x, f) = \frac{1}{2} \|x^{(i+1)} - f(x^{(i)})\|_2^2 \quad (\text{A.8})$$

leads to a rank 1 solution $W_1^{(\infty)} = x^{(i+1)} u^T$ and $W_2^{(\infty)} = v x^{(i)T}$ with $u, v \in \mathbb{R}^k$ satisfying

$$\frac{u_i^2 - u_i^{(0)2}}{2} = \int_{v_i^{(0)}}^{v_i} \frac{\phi(z)}{\phi'(z)} dz, \quad \text{and} \quad u_i \phi(v_i) = \frac{1}{k},$$

and $u_i = u_j, v_i = v_j$ for all $i, j \in [k]$.

The proof exactly follows that of Theorem 1, since updates to u, v do not depend on the data $x^{(i)}, x^{(i+1)}$.

Theorem 14. *Let $\{x^{(i)}\}_{i=1}^n$ be n training examples with $\|x^{(i)}\|_2 = 1$ for all $i \in [n]$, and let $\{f_i\}_{i=1}^n$ denote n 1-hidden layer networks satisfying the assumptions in Theorem 3 and trained on the loss in Eq. (2.4). Then the composition $f = f_n \circ f_{n-1} \circ \dots \circ f_1$ satisfies:*

$$\lambda_1(\mathbf{J}(f(x^{(1)}))) = \prod_{i=1}^n \left(\frac{\phi'(v_j^{(i)}) v_j^{(i)}}{\phi(v_j^{(i)})} \right). \quad (\text{A.9})$$

Proof. From Theorem 3, each of the f_i for $i \in [n]$ have the following form after training:

$$f_i(z) = x^{(i \bmod n+1)} u^{(i)T} \phi(v^{(i)} x^{(i)T} z)$$

with $u^{(i)T} \phi(v^{(i)}) = 1$ and $u_l^{(i)} = u_j^{(i)}, v_l^{(i)} = v_j^{(i)}$ for $l, j \in [k]$ and $i \in [n]$.

Hence the composition f is given by:

$$\begin{aligned} f(z) &= (f_n \circ f_{n-1} \circ \dots \circ f_1)(z) \\ &= x^{(1)} u^{(n)T} \phi(v^{(n)} x^{(n)T} x^{(n)} u^{(n-1)T} \phi(v^{(n-1)} x^{(n-1)} \dots \phi(v^{(1)} x^{(1)T} z) \dots) \\ &= x^{(1)} u^{(n)T} \phi(v^{(n)} u^{(n-1)T} \phi(v^{(n-1)} \dots \phi(v^{(1)} x^{(1)T} z) \dots) \quad \text{since } \|x^{(i)}\|_2 = 1 \text{ for all } i \in [n]. \end{aligned}$$

We now compute the Jacobian of f at the example $x^{(1)}$. Since $u_i k \phi(v_i) = 1$ for all $i \in [n]$ and $\|x^{(1)}\|_2 = 1$, it holds that

$$\mathbf{J}(f(x^{(1)})) = x^{(1)} \left(\prod_{i=1}^n u_j^{(i)} k \phi'(v_j^{(i)}) v_j^{(i)} \right) x^{(1)T}.$$

However, we know that $u_j^{(i)} = \frac{1}{k \phi(v_j^{(i)})}$, and so we have that

$$\lambda_1(\mathbf{J}(f(x^{(1)}))) = \prod_{i=1}^n \left(\frac{\phi'(v_j^{(i)}) v_j^{(i)}}{\phi(v_j^{(i)})} \right),$$

which completes the proof. □

F Limit Cycles in Recurrent Neural Networks

In order to demonstrate that recurrent neural networks (RNNs) can also memorize and recall sequences, we trained a vanilla RNN (whose architecture is detailed in Appendix Figure A-1) to encode the following sentence from our introduction: ‘‘Hopfield networks are able to store binary training patterns as attractive fixed points.’’ When training the RNN, we encoded each word using 1-hot representation i.e., since there are 13 words in the sentence, we represented each word with a vector of size 13 and placed a ‘‘1’’ in the index corresponding to the word.

We trained such that each word is mapped to the next modulo 13 using the Cross Entropy Loss (as is done in practice). Unlike the other settings considered in this work, RNNs are used to generate new sentences after training by sampling a new word from the vector output given a previous word¹. Under our architecture, we found that repeatedly choosing the highest probability word given the previous word consistently output the entire training sentence regardless of the number of times this sampling process was repeated.

G Sequence Encoding Audio

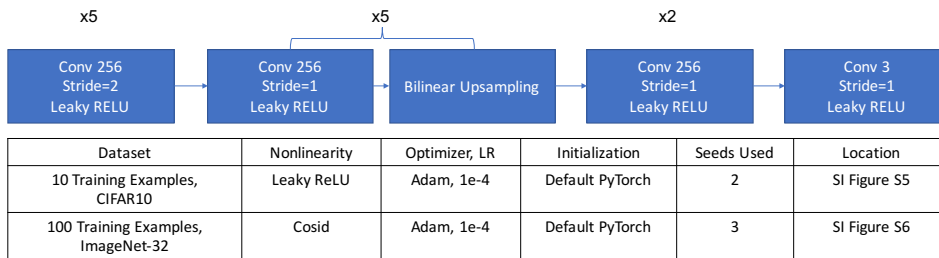
In order to demonstrate that fully connected networks can memorize high dimensional sequences, we captured an audio clip of an 8 second recording from the Donald Trump talking pen. Each second of the audio contains 22,050 frequencies, and we trained a fully connected network to map from the frequencies in second i to second $i + 1 \pmod 8$. We have attached an audio sample² titled ‘‘trump_quote_recovered_from_noise.mp3’’ demonstrating that iteration from random noise leads to recovery of the entire quote. The full architecture used is provided in Appendix Figure A-1.

¹This process is usually started from inputting the all zero vector.

²Located at: https://github.com/uhlerlab/neural_networks_associative_memory

Dataset	Width	Depth	Nonlinearity	Optimizer, LR	Initialization	Seeds Used	Location
500 Examples, ImageNet-64	1024	10	Cosid	Adam, 1e-4	U[-2e-2, 2e-2]	3072	Figure 2
6 Points, 2D	512	30	Cosid	Adam, 1e-4	Default PyTorch	5	Figure 3
389 Frames, "Steamboat Willie"	1024	16	SELU	Adam, 1e-4	Default PyTorch	2	Figure 4a
2 Sequences of Length 10, MNIST	128	31	SELU	Adam, 1e-4	Default PyTorch	2	Figure 4b
4 Sequences of Length 3, 2D	1024	10	Cosid	Adam, 1e-4	Default PyTorch	5	Figure 4c
Autoencoding/ Sequence encoding 100 MNIST Examples	N.A.	N.A.	SELU	Adam, 1e-4	Default PyTorch	2, 5, 2072	Figure 5
2000 Examples, MNIST	2048	10	Cosid	Adam, 1e-4	Default PyTorch	2	SI Figure S2
1000 Examples, CIFAR10, BW	2048	10	Cosid	Adam, 1e-4	U[-2e-2, 2e-2]	2	SI Figure S3
8 Seconds of Trump Audio	15	36	SELU	Adam, 1e-4	Default PyTorch	2	SI Supplementary Text G

(a) Fully Connected Network Training Details



(b) Convolutional Network Training Details

Dataset	Width	Depth	Nonlinearity	Optimizer, LR	Initialization	Seeds Used	Location
13 Word Sentence	128	31	SELU	Adam, 1e-4	Default PyTorch	2	SI Supplementary Text F

(c) Recurrent Network Training Details

Figure A-1: Training details for all experiments in main text and Appendix. Unless otherwise stated, all fully connected and convolutional networks are trained to minimize mean squared error below 10^{-8} . (a) Training details for fully connected architectures including dataset description, network width, network depth, nonlinearity, optimization method, learning rate, initialization scheme, random seeds used, and reference to the experiment in the text. (b) Training details for convolutional architecture including network topology, dataset, nonlinearity, optimization method, learning rate, initialization scheme, random seed and reference to experiment in the text. (c) Training details for recurrent architecture including network width and depth (for producing the next hidden state and output state), nonlinearity, optimization method, learning rate, initialization scheme, random seed used, and reference to experiment in the text.

Test Example	0	1	2	3	4	5	6	7	8	9	7	4
1st Iteration	0	1	2	3	4	5	6	7	8	9	7	9
Limit of Iteration	0	1	2	3	4	5	6	7	8	9	7	9

Figure A-2: Network trained on 2000 examples from MNIST stores all training examples as attractors. 1 iteration of the network leads to good reconstruction of the test example, but taking the limit of iteration leads to recovery of training examples. Average reconstruction error after 1 iteration of 58000 test examples is 0.0135. Training details are provided in Appendix Figure A-1.

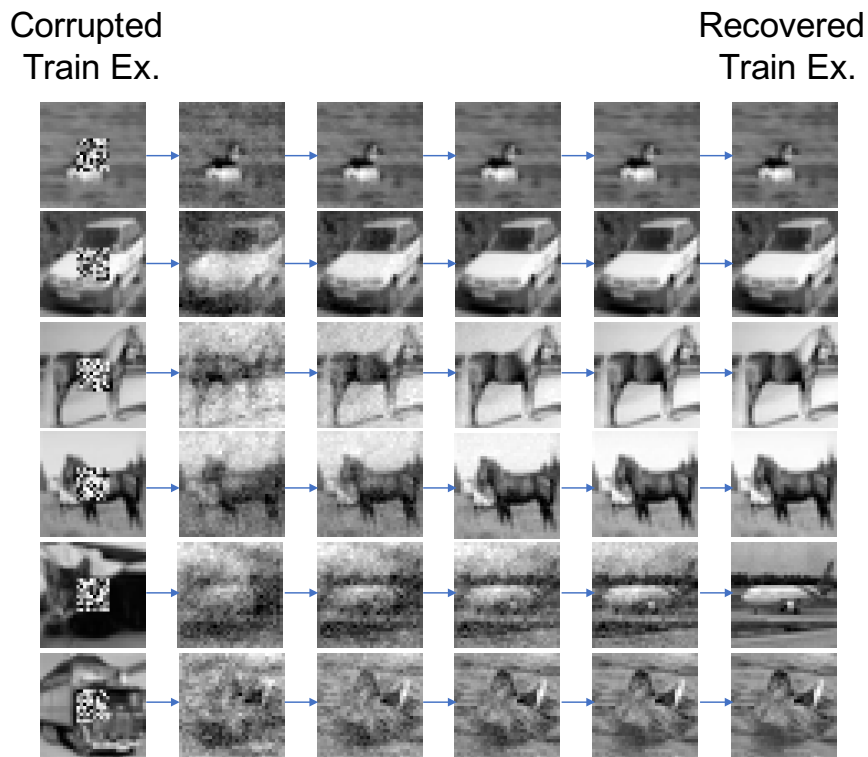


Figure A-3: Network trained on 1000 black and white images from CIFAR10 stores all training examples as attractors. Analogously to Figure 2-2a of the main text, as training examples are attractors, iteration from corrupted inputs results in the recovery of a training example. Training details are provided in Appendix Figure A-1.

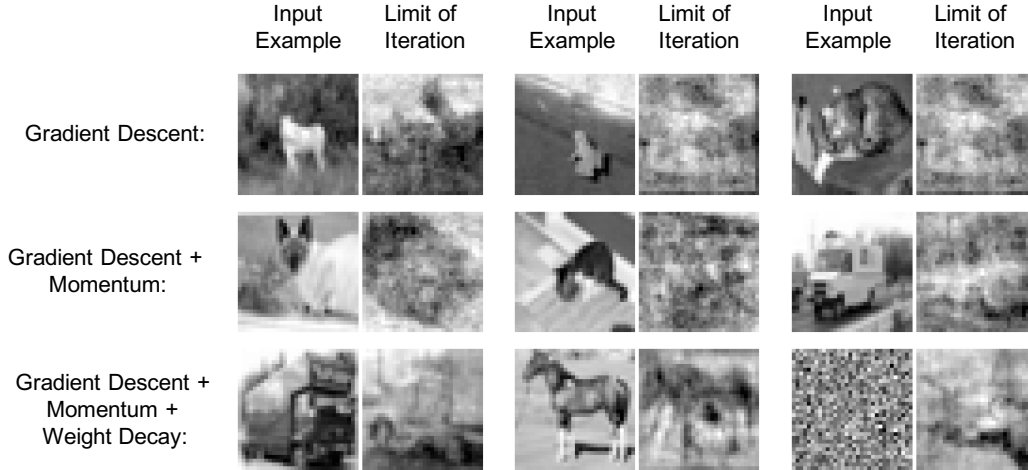


























Figure A-4: Examples of spurious training examples arising in over-parameterized autoencoders trained using different optimization methods on 100 black and white examples from CIFAR10. The networks used have 11 hidden layers, 256 hidden units per layer, SELU nonlinearity, and are initialized using the default PyTorch initialization scheme. For all optimization methods, we use a learning rate of 10^{-1} . We use a momentum value of 0.009 and weight decay of 0.0001.

Training Example										
Max Eigenvalue	0.03	0.02	0.05	0.09	0.04	0.21	0.02	0.08	0.05	0.09

(a)

Input									
Limit of Iteration									

(b)

Figure A-5: A U-Net convolutional autoencoder [161, 185] storing 10 CIFAR10 training examples as attractors. Training details are presented in Appendix Figure A-1b. (a) Maximum eigenvalue of the Jacobian of the network at the training example. (b) Iteration from corrupted inputs converges to a training example. In general, we observe that convolutional autoencoders store training examples as attractors when the receptive field size of the hidden units in the network covers the entire image. This can be achieved by increasing the stride of the filters, as is done in the U-Net used here.

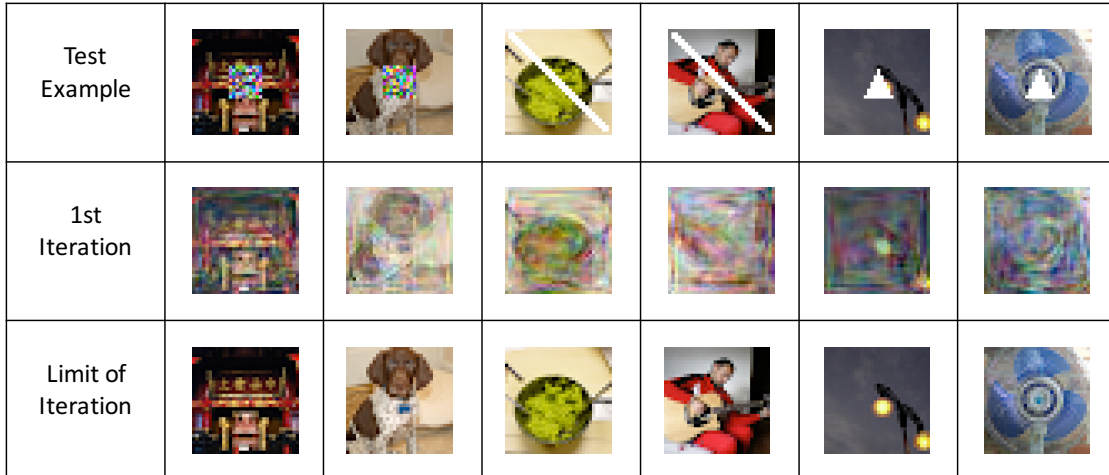


Figure A-6: A U-Net convolutional autoencoder [161, 185] storing 100 ImageNet-32 training examples as attractors. Training details are presented in Appendix Figure A-1b. Iteration from corrupted inputs converges to a training example.

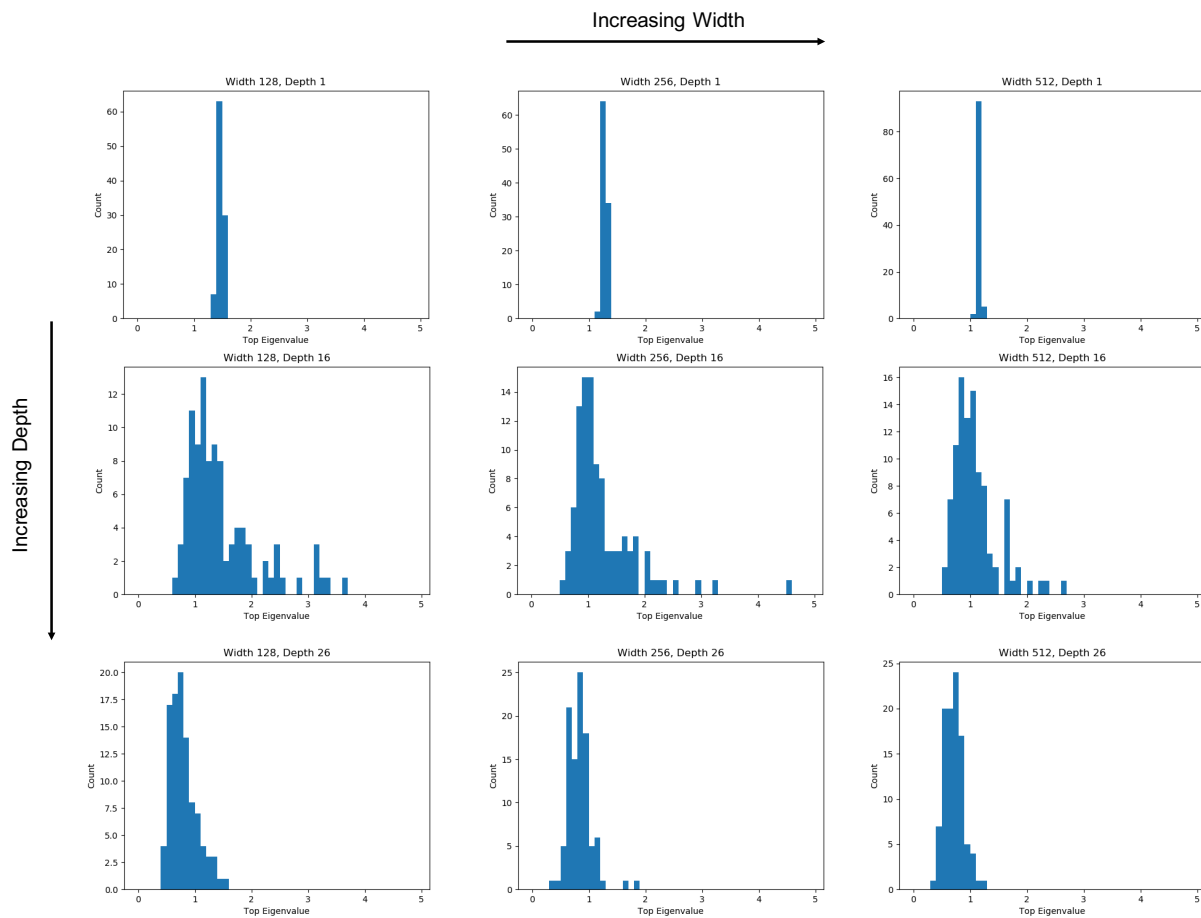


Figure A-7: Over-parameterized autoencoders become more contractive at the training examples as network depth and width are increased. Networks are trained on 100 examples from MNIST and are a subset of architectures considered in Figure 2-5a. Histograms of the maximum eigenvalue of the Jacobian at each of the 100 training examples are presented for each of the nine settings.

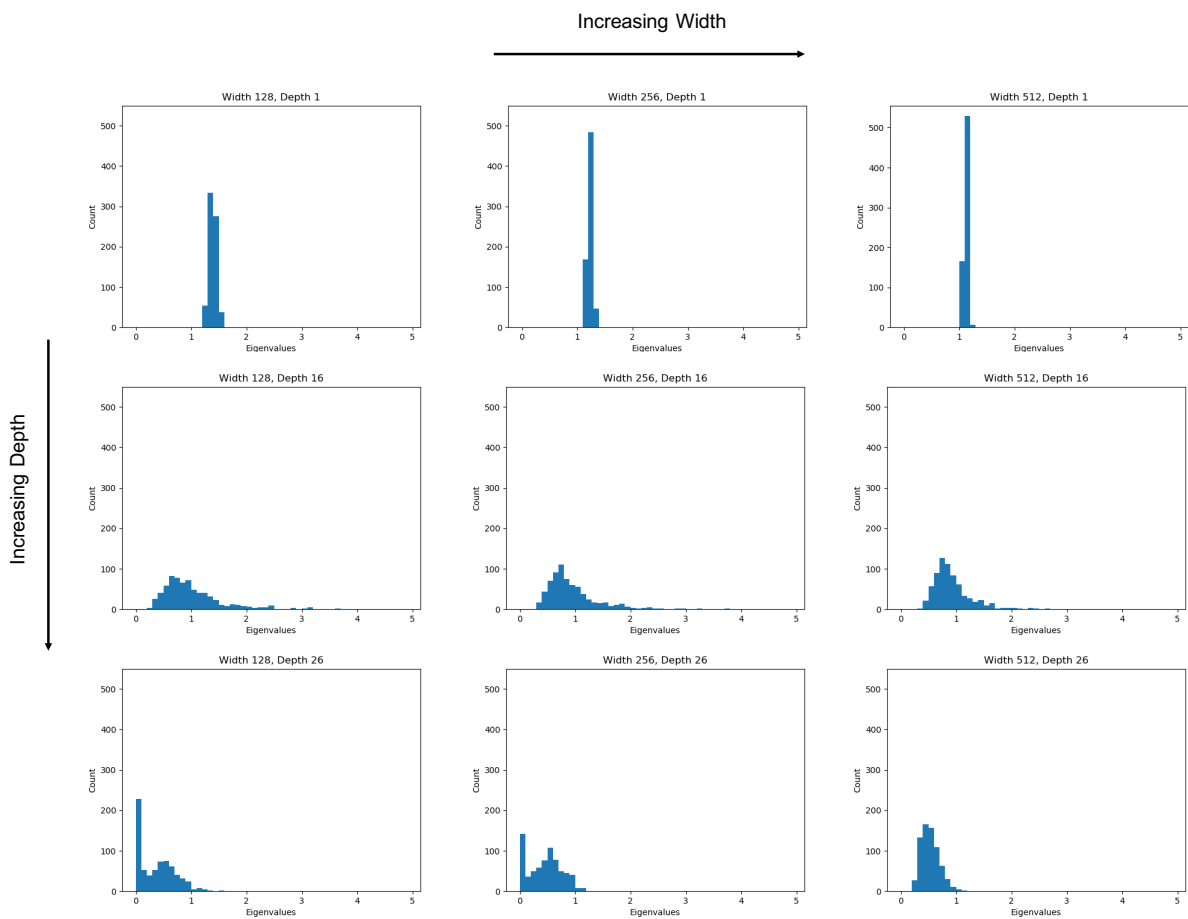


Figure A-8: Over-parameterized autoencoders become more contractive at the training examples as network depth and width are increased. Networks are trained on 100 examples from MNIST and are a subset of architectures considered in Figure 2-5a. Histograms of top 1% of 28^2 eigenvalues of the Jacobian at each of the 100 training examples are presented for each of the nine settings. The variance of the distribution of eigenvalues decreases as width increases.

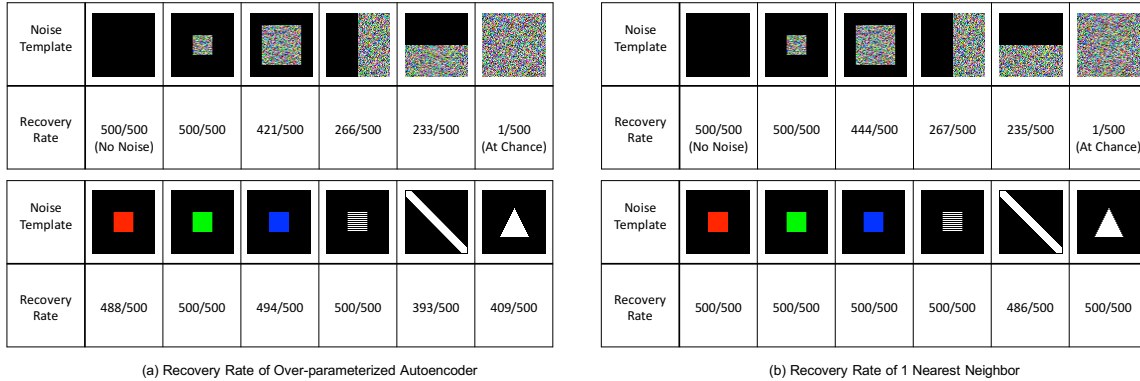


Figure A-9: Comparison of recovery rate of over-parameterized autoencoder trained on 500 examples from ImageNet-64 (Figure 1 of main text) and 1 nearest neighbor (1-NN). From Figure 2 of the main text, we know that over-parameterized autoencoders use metrics other than Euclidean distance to construct basins of attraction around training examples. However, in this high dimensional setting, the metric used by the over-parameterized autoencoder is similar to that of 1-NN, as is demonstrated by the similar recovery rates from varying corruption patterns.

Act. \ Opt.	ReLU	Leaky ReLU	SELU	Swish	$\cos x - x$	$x + \frac{\sin 10x}{5}$
GD	28/100	34/100	10/100	NA*	5/100	19/100
GD + Momentum	14/100	23/100	10/100	NA*	2/100	21/100
GD + Momentum + Weight Decay	NA*	NA*	18/100	NA*	22/100	NA*
RMSprop	97/100	98/100	100/100	49/100	100/100	100/100
Adam	38/100	53/100	30/100	14/100	100/100	100/100

Figure A-10: Impact of optimizer and nonlinearity on number of training examples stored as attractors. In all experiments, we used a fully connected network with 11 hidden layers, 256 hidden units per layer, and default PyTorch initialization. (*) NA indicates that the training error did not even decrease below 10^{-5} in 1,000,000 epochs. Although more attractors arise with adaptive methods and trigonometric nonlinearities, attractors arise in all settings considered. Note that we used a loss threshold of 10^{-8} for this table since the non-adaptive methods could not converge to 10^{-8} in 1,000,000 epochs.

Act. \ Init.	ReLU	Leaky ReLU	SELU	Swish	$\cos x - x$	$x + \frac{\sin 10x}{5}$
U[-0.01, 0.01]	62/100	78/100	78/100	16/100	26/100	93/100
U[-0.02, 0.02]	43/100	65/100	71/100	20/100	31/100	70/100
U[-0.05, 0.05]	55/100	55/100	29/100	32/100	100/100	89/100
U[-0.1, 0.1]	36/100	43/100	13/100	30/100	100/100	NA*
U[-0.15, 0.15]	34/100	38/100	13/100	6/100	100/100	NA*

Figure A-11: Impact of initialization on number of training examples stored as attractors. In all experiments, we used a fully connected network with 11 hidden layers and 256 hidden units per layer trained using the Adam optimizer ($\text{lr}=10^{-4}$). (*) NA indicates that the training error did not decrease below 10^{-8} in 1,000,000 epochs. Attractors arise under all settings for which training converged. Generally, more attractors arise under smaller initializations or with trigonometric nonlinearities.

Appendix B

Chapter 3 Supplementary

A Proofs for Matrix Completion with the NTK

We present the statement of Theorem 1 for a general homogeneous (degree 1), Lipschitz nonlinearity below and then present the proof. We again note that ReLU and LeakyReLU are commonly used nonlinearities that satisfy these conditions. The results are easily extended to homogeneous nonlinearities of arbitrary degree and for feature priors that have columns with arbitrary norm.

Theorem. *Assume $Z = \{z^{(i)}\}_{i=1}^n \in \mathbb{R}^{p \times n}$, where each column is normalized with $\|z^{(i)}\|_2 = 1$. Let $f_Z(\mathbf{W})$ be a d layer fully connected network with Lipschitz nonlinearity ϕ that is homogeneous of degree 1 and $c = \|\phi\|_{L^2(\mu)}^{-1}$ where $L^2(\mu)$ is the Hilbert space of square Lebesgue integrable functions under Gaussian measure. Then as layer widths $k_1 \rightarrow \infty, k_2 \rightarrow \infty, \dots, k_{d-1} \rightarrow \infty$ and under the Gradient Independence Ansatz [202], the NTK for matrix completion with $f_Z(\mathbf{W})$ is given by*

$$K_d(M_{ij}, M_{i'j'}) = \begin{cases} \kappa_d(z^{(j)T} z^{(j')}) & \text{if } i = i' \\ 0 & \text{if } i \neq i' \end{cases},$$

where $\kappa_d(\xi) = \check{\phi}^{(d)}(\xi) + \kappa_{d-1}(\xi) \frac{d\check{\phi}}{d\xi}(\check{\phi}^{(d-1)}(\xi))$, and $\check{\phi}^{(k)}(\xi) = \check{\phi}(\check{\phi}^{(k-1)}(\xi))$ for $k \geq 1$ and $\check{\phi}^{(0)}(\xi) = \xi$.

Proof. We proceed by induction and present the case for $d = 1$ first. Namely, we define $g_Z(M)$ as follows:

$$g_Z(M) = \text{tr}(M^T A \phi(BZ)),$$

where $A \in \mathbb{R}^{m \times k}, B \in \mathbb{R}^{k \times p}, Z \in \mathbb{R}^{p \times n}$. To compute the kernel, we compute $\frac{\partial g_Z(M)}{\partial A_{\alpha,\beta}}, \frac{\partial g(M)}{\partial B_{\alpha,\beta}}$ directly. We begin by expanding the matrix products in $g_Z(M)$. For a matrix U , we let $U_{i,:}$ denote row i of U and $U_{:,i}$ denote column i of U . Note that

$$\begin{aligned} g_Z(M) &= \text{tr} \left(M^T A \frac{c}{\sqrt{k}} \phi(BZ) \right) \\ &= \frac{c}{\sqrt{k}} \text{tr} \left(M^T A \begin{bmatrix} \phi(B_{1,:}Z_{:,1}) & \dots & \phi(B_{1,:}Z_{:,n}) \\ \vdots & \dots & \vdots \\ \phi(B_{k,:}Z_{:,1}) & \dots & \phi(B_{k,:}Z_{:,n}) \end{bmatrix} \right) \\ &= \frac{c}{\sqrt{k}} \text{tr} \left(M^T \begin{bmatrix} \sum_{a=1}^k A_{1,a} \phi(B_{a,:}Z_{:,1}) & \dots & \sum_{a=1}^k A_{1,a} \phi(B_{a,:}Z_{:,n}) \\ \vdots & \dots & \vdots \\ \sum_{a=1}^k A_{m,a} \phi(B_{a,:}Z_{:,1}) & \dots & \sum_{a=1}^k A_{m,a} \phi(B_{a,:}Z_{:,n}) \end{bmatrix} \right) \\ &= \frac{c}{\sqrt{k}} \sum_{i=1}^m \sum_{j=1}^n M_{i,j} \sum_{a=1}^k A_{i,a} \phi(B_{a,:}Z_{:,j}). \end{aligned}$$

We thus have that

$$\begin{aligned}\frac{\partial g_Z(M)}{\partial A_{\alpha,\beta}} &= \frac{c}{\sqrt{k}} \sum_{j=1}^n M_{\alpha,j} \phi(B_{\beta,:}, Z_{:,j}), \\ \frac{\partial g_Z(M)}{\partial B_{\alpha,\beta}} &= \frac{c}{\sqrt{k}} \sum_{i=1}^m \sum_{j=1}^n M_{i,j} A_{i,\alpha} \phi(B_{\alpha,:}, Z_{:,j}) Z_{\beta,j}.\end{aligned}$$

The NTK is given by:

$$\begin{aligned}K_1(M, \tilde{M}) &= \langle \nabla g_Z(M), \nabla g_Z(M') \rangle \\ &= \sum_{\alpha=1}^m \sum_{\beta=1}^k \frac{\partial g_Z(M)}{\partial A_{\alpha,\beta}} \cdot \frac{\partial g_Z(M')}{\partial A_{\alpha,\beta}} + \sum_{\alpha=1}^m \sum_{\beta=1}^p \frac{\partial g_Z(M)}{\partial B_{\alpha,\beta}} \cdot \frac{\partial g_Z(M')}{\partial B_{\alpha,\beta}}.\end{aligned}$$

To simplify the computation, we note that we will only ever need the gradient at indicator matrices M_{ij} and $M_{i'j'}$. Moreover, from the formula for the partial derivatives, we conclude that

$$\begin{aligned}\frac{\partial g_Z(M_{ij})}{\partial A_{\alpha,\beta}} &= \begin{cases} 0 & \text{if } \alpha \neq i \\ \frac{c}{\sqrt{k}} \phi(B_{\beta,:}, Z_{:,j}) & \text{otherwise} \end{cases}, \\ \frac{\partial g_Z(M_{ij})}{\partial B_{\alpha,\beta}} &= \frac{c}{\sqrt{k}} A_{i,\alpha} \phi(B_{\alpha,:}, Z_{:,j}) Z_{\beta,j}.\end{aligned}$$

Thus, we can simplify the NTK as follows:

$$\begin{aligned}\lim_{k \rightarrow \infty} K_1(M_{ij}, M_{i'j'}) &= \lim_{k \rightarrow \infty} \frac{c^2}{k} \sum_{\beta=1}^k \phi(B_{\beta,:}, Z_{:,j}) \phi(B_{\beta,:}, Z_{:,j'}) \mathbf{1}_{i=i'} \\ &\quad + \frac{c^2}{k} \sum_{\alpha=1}^k A_{i,\alpha} A_{i',\alpha} \sum_{\beta=1}^p \phi'(B_{\beta,:}, Z_{:,j}) \phi'(B_{\beta,:}, Z_{:,j'}) Z_{\beta,j} Z_{\beta,j'} \\ &= \begin{cases} 0 & i \neq i' \\ \kappa_1(z^{(j)T} z^{(j')}) & i = i' \end{cases},\end{aligned}$$

which completes the base case.

For the inductive step, we assume that

$$\lim_{k_{d-2} \rightarrow \infty} \dots \lim_{k_1 \rightarrow \infty} K_{d-1}(M_{ij}, M_{i'j'}) = \begin{cases} 0 & i \neq i' \\ \kappa_{d-1}(z^{(j)T} z^{(j')}) & i = i' \end{cases}.$$

We now show that $K_d(M_{ij}, M_{i'j'})$ has the desired form. For this, we define:

$$g_Z(M) = M^T A \frac{c}{\sqrt{k_{d-1}}} \phi(h_Z(\mathbf{W})),$$

where $A \in \mathbb{R}^{m \times k_{d-1}}$ and $h_Z(\mathbf{W}) : \mathbb{R}^{p \times n} \rightarrow \mathbb{R}^{k_{d-1} \times n}$ is a $d-1$ fully connected network operating on Z .

Following the computation for the 1 layer case, we obtain

$$\begin{aligned}\frac{\partial g_Z(M)}{\partial A_{\alpha,\beta}} &= \frac{c}{\sqrt{k_{d-1}}} \sum_{j=1}^n M_{\alpha,j} \phi(h_Z(\mathbf{W}))_{\beta,j}, \\ \frac{\partial g_Z(M)}{\partial \mathbf{W}_{\alpha,\beta}} &= \frac{c}{\sqrt{k_{d-1}}} \sum_{i=1}^m \sum_{j=1}^n M_{i,j} \sum_{k=1}^{k_{d-1}} A_{i,k} \frac{\partial \phi(h_Z(\mathbf{W}))_{k,j}}{\partial \mathbf{W}_{\alpha,\beta}}.\end{aligned}$$

Now we consider the case of indicator matrices $M_{ij}, M_{i'j'}$. For M_{ij} , we note that $\frac{\partial g_Z(M)}{\partial A_{\alpha,\beta}}$ is only non-zero for the terms

$$\frac{\partial g_Z(M_{ij})}{\partial A_{i,\beta}} = \frac{c}{\sqrt{k_{d-1}}} \phi(h_Z(\mathbf{W}))_{\beta,j}.$$

Hence, if $i \neq i'$, we obtain that

$$\sum_{\alpha,\beta} \frac{\partial g_Z(M_{ij})}{\partial A_{\alpha,\beta}} \frac{\partial g_Z(M_{i'j'})}{\partial A_{\alpha,\beta}} = 0.$$

Similarly, for M_{ij} , we have that

$$\frac{\partial g_Z(M_{ij})}{\partial \mathbf{W}_{\alpha,\beta}} = \frac{c}{\sqrt{k_{d-1}}} \sum_{k=1}^{k_{d-1}} A_{i,k} \frac{\partial \phi(h_Z(\mathbf{W}))_{k,j}}{\partial \mathbf{W}_{\alpha,\beta}}.$$

If $i \neq i'$, as $k_{d-1} \rightarrow \infty$, by law of large numbers:

$$\sum_{\alpha,\beta} \frac{\partial g_Z(M_{ij})}{\partial B_{\alpha,\beta}} \frac{\partial g_Z(M_{i'j'})}{\partial B_{\alpha,\beta}} \rightarrow 0.$$

Thus, if $i \neq i'$, we conclude that $K_d(M_{ij}, M_{i'j'}) = 0$. On the other hand, if $i = i'$, then we have that

$$\sum_{\alpha,\beta} \frac{\partial g_Z(M_{ij})}{\partial A_{\alpha,\beta}} \frac{\partial g_Z(M_{i'j'})}{\partial A_{\alpha,\beta}} = \frac{c^2}{k_{d-1}} \sum_{k=1}^{k_{d-1}} \phi(h_Z(\mathbf{W}))_{k,j} \phi(h_Z(\mathbf{W}))_{k,j'}. \quad (\text{B.1})$$

Similarly, if $i = i'$, we have that

$$\begin{aligned}\sum_{\alpha,\beta} \frac{\partial g_Z(M_{ij})}{\partial B_{\alpha,\beta}} \frac{\partial g_Z(M_{i'j'})}{\partial B_{\alpha,\beta}} &= \frac{c^2}{k_{d-1}} \left(\sum_{k=1}^{k_{d-1}} A_{i,k} \phi'(h_Z(\mathbf{W}))_{k,j} \frac{\partial h_Z(\mathbf{W})_{k,j}}{\partial \mathbf{W}_{\alpha,\beta}} \right) \\ &\quad \cdot \left(\sum_{k=1}^{k_{d-1}} A_{i,k} \phi'(h_Z(\mathbf{W}))_{k,j'} \frac{\partial h_Z(\mathbf{W})_{k,j'}}{\partial \mathbf{W}_{\alpha,\beta}} \right).\end{aligned}$$

By the inductive hypothesis as $k_1, k_2, \dots, k_{d-2} \rightarrow \infty$, the above converges in probability to:

$$\lim_{k_{d-2} \rightarrow \infty} \lim_{k_{d-3} \rightarrow \infty} \dots \lim_{k_1 \rightarrow \infty} \sum_{\alpha,\beta} \frac{\partial g_Z(M_{ij})}{\partial B_{\alpha,\beta}} \frac{\partial g_Z(M_{i'j'})}{\partial B_{\alpha,\beta}} \quad (\text{B.2})$$

$$\rightarrow \frac{c^2}{k_{d-1}} \left(\sum_{k=1}^{k_{d-1}} \phi'(h_Z(\mathbf{W}))_{k,j} \phi'(h_Z(\mathbf{W}))_{k,j'} K_{d-1}(M_{k,j}, M_{k,j'}) \right). \quad (\text{B.3})$$

Therefore, when $i = i'$, adding Eqs. [B.1] and [B.2] and applying the inductive hypothesis yields:

$$\lim_{k_{d-1} \rightarrow \infty} \lim_{k_{d-2} \rightarrow \infty} \dots \lim_{k_1 \rightarrow \infty} K_d(M_{ij}, M_{i'j'}) = \check{\phi}(\check{\phi}^{(d-1)}(\langle z^{(j)}, z^{(j')} \rangle)) + K_{d-1} \frac{d\check{\phi}}{d\xi} \left(\check{\phi}^{(d-1)}(\langle z^{(j)}, z^{(j')} \rangle) \right),$$

which completes the proof. \square

We next provide an example showing how to compute the NTK for matrix completion.

Example. *Suppose we have:*

$$Y = \begin{bmatrix} y_{11} & .5 & .3 \\ .1 & .2 & y_{23} \\ .4 & y_{32} & y_{33} \end{bmatrix}.$$

Assuming we read off the observed entries of Y in row major order and that $Z = I$ (the 3×3 identity matrix), then the NTK is given by:

$$K = \begin{bmatrix} \kappa(1) & \kappa(0) & 0 & 0 & 0 \\ \kappa(0) & \kappa(1) & 0 & 0 & 0 \\ 0 & 0 & \kappa(1) & \kappa(0) & 0 \\ 0 & 0 & \kappa(0) & \kappa(1) & 0 \\ 0 & 0 & 0 & 0 & \kappa(1) \end{bmatrix}.$$

The solution to kernel regression is given by:

$$\tilde{g}(M) = \begin{bmatrix} .5 & .3 & .1 & .2 & .4 \end{bmatrix} K^{-1} k(M),$$

where $k(M)$ is the vector with entries $k(M_{ij}, M)$ for $(i, j) \in S$. As an example, for M_{11} , we have:

$$k(M_{11}) = \begin{bmatrix} \kappa(0) & \kappa(0) & 0 & 0 & 0 \end{bmatrix}^T.$$

This example demonstrates the key difference between the NTK of fully connected networks for matrix completion and the usual multivariate NTK: namely, the former corresponds to solving a *separate* kernel regression problem for each row of the target matrix Y . By modifying the nonlinearity ϕ and the feature prior in Theorem 1, our framework encapsulates a broad class of semi-supervised learning approaches for matrix completion. We provide a nontrivial example below.

Example (Semi-supervised Learning with the Graph Laplacian). *The following corollary to Theorem 2 proves that semi-supervised learning using the graph Laplacian operator from [26] is a specific instance of matrix completion with the NTK of a linear neural network used for matrix completion.*

Corollary. *Let $X \in \mathbb{R}^{d \times n}$ denote a set of data points of which a subset $X_S \in \mathbb{R}^{d \times s}$ is labelled with labels $Y_S \in \mathbb{R}^{1 \times s}$. Let $Z \in \mathbb{R}^{p \times n}$ denote the projection of X onto the top p eigenvectors of the graph Laplacian. Let $g_Z(M) = \text{tr}(M^T A \frac{1}{\sqrt{2k}} B Z)$ for $A \in \mathbb{R}^{1 \times k}$, $B \in \mathbb{R}^{k \times p}$. Then as $k \rightarrow \infty$, the following are equivalent:*

$$\arg \min_{A, B} \sum_{(i, j) \in S} (Y_{ij} - g_Z(M_{ij}))^2 \iff \arg \min_{w \in \mathbb{R}^p} \|Y_S - wZ\|_2^2$$

in the sense that $g_Z(M_{ij}) = wZ_{:,j}$.

The proof follows immediately from Theorem 1 and the fact that the dual activation for $\phi(x) = x$ is $\check{\phi}(\xi) = \xi$. The example above illustrates the generality of our framework for matrix completion. Moreover,

semi-supervised learning with the graph Laplacian can naturally be extended by using the NTK for a nonlinear neural network instead of a linear neural network. Namely, instead of using the eigenvectors of the graph Laplacian, we can naturally extend the above corollary by using embeddings produced by autoencoders (Ch. 14 of [69]).

Note that the flexibility to learn a low-rank imputation or imputation with other structures via our framework is given by the feature prior, which incorporates the relationships between the coordinates of the target matrix. Indeed, varying the feature prior can drastically change the imputation given by the NTK, and the NTK with appropriate feature prior can even produce low-rank imputations, as shown by the following example.

Example. Consider the Netflix problem of movie rating imputation. Suppose the target matrix Y is of the form

$$Y = \begin{bmatrix} 1 & 2 & y_{13} \\ 1 & 2 & 3 \end{bmatrix},$$

where the rows of Y represent users, the columns represent movies, and the coordinate Y_{ij} represents the rating (from 1 to 5 stars) a user i gave to movie j . By first flattening the matrix Y into $Y_v = [1, 2, y_{13}, 1, 2, 3]$, and then using our framework with feature prior

$$Z = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix},$$

leads to a low rank imputed matrix

$$\hat{Y} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}.$$

The above example is simplistic in that it produces a low rank imputation by assuming that the users are identical and using a one-hot embedding for the movies. In practice, one would use a feature prior that embeds users via external metadata (e.g. user age, gender, etc.) and our framework would predict similar ratings for users with similar metadata.

B Experimental Details for Virtual Drug Screening in CMAP

For this application, we consider the 978 genes \times 2,130 drugs \times 71 cell types “large” tensor from [84]. From this tensor, we extract the 15,855 non-null values, and leave out the cell types (‘SNU1040’, ‘HEK293T’, ‘HS27A’), as they have less than 10 drugs in the dataset (i.e. for these cell types, we would not be able to perform 10-fold cross validation). We exclude MCF7 from the dataset when using our method, since we use it to compute our feature prior, but we give all other methods training access to all MCF7 observations to ensure a fair comparison. This leaves us with a dataset of 14,336 samples, which are used for imputation. A link to download this dataset is given in [84], which we repeat here for convenience: https://github.com/clinicalml/dgc_predict.

For training DNPP and FaLRTC, we use the same hyper-parameters as in [84]. We implemented DNPP, mean over cell type, and our framework in Python in the above link. We use the Matlab code from [84] located via the following link: https://github.com/clinicalml/dgc_predict/blob/b8bff6d757fc757aadf39034b9972db37c6da983/matlab/thirdparty/visual/FaLRTC.m. In order to make our results for FaLRTC accessible without Matlab, we provide the imputations from FaLRTC in the following folder: https://www.dropbox.com/sh/w23viwbm3py1dq1/AADQD3Bi_bLx4Z7X2hcLoUzXa?dl=0.

C Feature Prior for Drug Response Imputation

DNPP performs well for imputing the effect of drugs on cell types that have many observations in the training set, but performs poorly when imputing the effect of drugs on cell types with few observations in the training set. Thus, to improve on DNPP, we use a dual feature prior: one for imputing the effect of drugs on cell types with many (at least 150) observations in the training set (the dense regime), and another for imputing the effect of drugs on cell types with few (at most 150) observations in the training set (the sparse regime).

Since DNPP and FaLRTC both yield an imputation that captures similarity between cell type and drug combinations in the large observation regime, we can use the output of one of these methods as the feature prior for those cell types that had greater than 150 drugs in the training set. In particular, we chose the output of FaLRTC for the feature prior in the dense regime since applying our method with this feature prior yielded superior results. For all observed examples that were in the training set, we use the gene expression for the observation itself as the encoding. For all feature priors, we additionally concatenated a constant (1.5) times the identity matrix to ensure that the corresponding kernel is positive definite¹. We then solved kernel regression exactly (using the numpy solve function [140]) for the NTK of a 1-hidden layer ReLU network.

For those cell types with few (less than 150 observations) in the training set, we used a feature prior that concatenates an embedding of the cell type and an embedding of the drug type. For the drug embedding, we used the gene expression of MCF7 treated with the same drug as the drug embedding, if available in the training set. If this vector was not available in the training set, we simply used the mean of all MCF7 observations. For the cell type embedding, we used the mean of all observations for the corresponding cell type available in the training set. We then normalized each cell embedding to have the same norm as the drug embedding to balance their contributions to dot products computed for the kernel. We re-scaled the embedding for the cell type by a factor of 1.25 to give the cell type additional weight over drug type². Lastly, we normalized the concatenation of the embeddings and solved kernel regression via the closed form in Theorem 1. We refer to this feature prior as the *MCF7 reference prior*. The code for computing our feature priors is available at https://github.com/uhlerlab/ntk_matrix_completion.

D One-hot Encoding for Drugs is Equivalent to Imputation with Mean Over Cell Type

The following result shows that using a feature prior consisting of a one-hot embedding for drugs leads to performing imputation using the mean over all observations for a given cell type.

Proposition 8. *Let $Y \in \mathbb{R}^{m \times n}$ denote the gene expression vectors for cell type c with drugs $\{d_j\}_{j=1}^n$, such that columns $\{y^{(j)}\}_{i=1}^\ell$ are observed and columns $\{y^{(j)}\}_{i=\ell+1}^n$ are missing. Let $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times p}$, $\phi(x) = \max(x, 0)$, $g: \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$ such that:*

$$g(M) = \text{tr} \left(M^T A \frac{\sqrt{2}}{\sqrt{k\ell}} \phi(BZ) \right),$$

where $Z = I_{n \times n}$ (i.e. a one-hot encoding of the drug). Then for $i \in [m]$, $j > \ell$, the solution to kernel ridge-less regression with the NTK for g is:

$$\tilde{g}(M_{ij}) = \left(\frac{1}{2\pi - 1} - \frac{\ell}{(2\pi - 1)(2\pi - 1 + \pi\ell)} \right) \left(\frac{1}{\ell} \sum_{j=1}^\ell y_i^{(j)} \right).$$

Proof. The proof relies on the fact that the kernel matrix K for g is a block diagonal matrix. In particular, as shown in the example in Section 3, there is one block, $K_{B_i} \in \mathbb{R}^{\ell \times \ell}$, for each row of Y (i.e. m blocks), and K_{B_i} has diagonal entries $\kappa(1) = 2$ and off-diagonal entries $\kappa(0) = \frac{1}{\pi}$. Hence, each block of the kernel matrix

¹We chose the constant 1.5 by tuning this parameter to give highest Pearson r value on seed 512. We then used this constant for all other random seeds.

²This hyperparameter was selected to maximize Pearson r value for seed 512 and then fixed across all other random seeds.

can be written as:

$$K_{B_i} = \frac{1}{\ell} \left(\left(2 - \frac{1}{\pi} \right) I_{\ell \times \ell} + \frac{1}{\pi} J \right),$$

where $J \in \mathbb{R}^{\ell \times \ell}$ is the all ones matrix. By the Sherman-Morrison formula,

$$K_{B_i}^{-1} = \frac{1}{\ell} \left(\frac{\pi}{2\pi - 1} I - \frac{\pi}{(2\pi - 1)(2\pi - 1 + \pi\ell)} J \right),$$

and thus

$$\tilde{g}(M_{ij}) = \begin{bmatrix} y_i^{(1)} & y_i^{(2)} & \dots & y_i^{(\ell)} \end{bmatrix} K_{B_i}^{-1} \mathbf{1} \frac{1}{\pi\ell},$$

where $\mathbf{1} \in \mathbb{R}^{\ell}$ is the all ones vector. Hence,

$$\tilde{g}(M_{ij}) = \left(\frac{1}{2\pi - 1} - \frac{\ell}{(2\pi - 1)(2\pi - 1 + \pi\ell)} \right) \left(\frac{1}{\ell} \sum_{j=1}^{\ell} y_i^{(j)} \right),$$

which completes the proof. \square

E Feature Prior Corresponding to Previous Algorithms

As discussed in Section 2 of the main text, our framework provides a direct approach for improving upon previous methods for virtual drug screening. Using the output of DNPP and FaLRTC as the feature prior in our framework leads to an improvement; namely, across every round and fold in 5 rounds of 10-fold cross validation (using seeds 149, 10, 53, 77, 1928), we find that our method with the DNPP output as a feature prior outperforms DNPP and that our method with the FaLRTC output as a feature prior outperforms FaLRTC. This is demonstrated in SI Fig. S1 and S2.

F Performance of Methods on Sparse versus Dense Subsets

We demonstrate in SI Fig. S3 that DNPP is effective for imputation on the dense regime (i.e. for those drug/cell type pairs with over 150 profiles), but not as effective in the sparse regime (i.e. for those drug/cell type pairs with less than 150 profiles). FaLRTC seems to perform comparably between the dense and the sparse regime, but under-performs DNPP on the full dataset.

G Metrics for Evaluation in Drug Response Imputation

Let $\hat{Y} \in \mathbb{R}^{m \times n}$ denote the concatenation of the test predictions for all 10 folds and let $Y^* \in \mathbb{R}^{m \times n}$ denote the ground truth. We use $y^{*(i)}$ to denote the i^{th} column of Y^* . Let $\bar{y}^{(i)} = c_i \mathbf{1}$ where $c_i = \sum_{j=1}^m y_j^{(i)}$. For $A \in \mathbb{R}^{a \times b}$, let $A_v \in \mathbb{R}^{a \cdot b}$ denote the vectorized version of A . We use the following 3 metrics for evaluating the effectiveness of a given imputation method. All evaluation metrics have a maximum value of 1.

1. *Pearson r value*: This evaluation metric was used in [84] and is given by:

$$v = \frac{\langle \hat{Y}_v, Y_v^* \rangle}{\|\hat{Y}_v\|_2 \|Y_v^*\|_2}.$$

2. *Mean R²*: This evaluation metric is given by:

$$v = \frac{1}{n} \sum_{i=1}^n \left(1 - \frac{\sum_{j=1}^m (\hat{y}_j^{(i)} - y_j^{*(i)})^2}{\sum_{j=1}^m (y_j^{*(i)} - \bar{y}_j^{(i)})^2} \right).$$

3. *Mean Cosine Similarity*: This evaluation metric is given by:

$$v = \frac{1}{n} \sum_{i=1}^n \frac{\langle \hat{y}^{(i)}, y^{*(i)} \rangle}{\|\hat{y}^{(i)}\|_2 \|y^{*(i)}\|_2}.$$

H Statistical Significance of NTK on Drug Response Imputation

In experiments on the full dataset, we use 10-fold cross validation and 5 random seeds (149, 10, 77, 53, 1928) for comparing our method to DNPP from [84]. For each fold, we ensure that 10% of the drugs for each cell type are present in the test set. To determine the statistical significance of our method for improving over DNPP, we use a one-sided test with the following corrected repeated k-fold cv test statistic for r rounds of k -fold cross validation (as described in Section 3.3 of [34]):

$$t = \frac{\frac{1}{kr} \sum_{i=1}^k \sum_{j=1}^r d_{ij}}{\left(\frac{1}{kr} + \frac{n_2}{n_1}\right) \hat{\sigma}^2},$$

where d_{ij} is the difference between the evaluation metric for our method (the output of FaLRTC as the feature prior for the dense regime and the MCF7 reference feature prior for the sparse regime) and that of the DNPP for fold k of round j , $\hat{\sigma}$ is the estimated variance of the differences d_{ij} , and n_1 is the number of samples used for training and n_2 is the number of samples used for testing (i.e. $\frac{n_2}{n_1} \approx \frac{1}{9}$ for our setting). This statistic is distributed according to a t-distribution with $kr - 1$ degrees of freedom. For the mean R^2 , we obtain $t = 18.29$ and a corresponding p-value of $7.7 \cdot 10^{-24}$. For the mean cosine similarity, we obtain $t = 14.75$ and a p-value of $5.9 \cdot 10^{-20}$. Thus, at a significance level of .01, we reject the null hypothesis that our method and DNPP have the same performance.

I Matrix Completion with the CNTK

We repeat Proposition 1 from the main text and present the proof below. The tensor $\Theta \in \mathbb{R}^{m \times n \times m \times n}$ was defined and used in the computation of the CNTK for classification in [9].

Proposition. *Let $f_Z(\mathbf{W})$ be a d layer convolutional network used to map from the feature prior $Z \in \mathbb{R}^{c \times r \times s}$ to the target matrix $Y \in \mathbb{R}^{m \times n}$. Then as the number of convolutional filters per layer tends to infinity, the CNTK of $f_Z(\mathbf{W})$ is given by:*

$$K(M_{ij}, M_{i'j'}) = [\Theta^{(d)}(Z, Z)]_{i,j,i',j'}, \quad (\text{B.4})$$

where $M_{ij}, M_{i'j'} \in \mathbb{R}^{m \times n}$ denote indicator matrices.

Proof. The proof follows almost immediately from the derivation of the CNTK for classification provided in [9]. Namely, let $g(M) = M^T f_Z(\mathbf{W})$ for $M \in \mathbb{R}^{m \times n}$. Then, we have that:

$$\frac{\partial g(M)}{\partial \mathbf{W}_{\alpha,\beta}} = \sum_{i=1}^m \sum_{j=1}^n M_{i,j} \frac{\partial f_Z(\mathbf{W})_{i,j}}{\partial \mathbf{W}_{\alpha,\beta}}.$$

Thus, the kernel at the indicator matrices $M_{ij}, M_{i'j'}$ is given by:

$$K(M_{ij}, M_{i'j'}) = \frac{\partial g(M_{ij})}{\partial \mathbf{W}_{\alpha,\beta}} \frac{\partial g(M_{i'j'})}{\partial \mathbf{W}_{\alpha,\beta}} = \frac{\partial f_Z(\mathbf{W})_{i,j}}{\partial \mathbf{W}_{\alpha,\beta}} \frac{\partial f_Z(\mathbf{W})_{i',j'}}{\partial \mathbf{W}_{\alpha,\beta}} = [\Theta(Z, Z)]_{i,j,i',j'},$$

which completes the proof. \square

Below we additionally present an explicit derivation for the 1 hidden layer case for ReLU networks. This derivation will be useful in understanding the connection between the CNTK for matrix completion with semi-supervised learning from coordinate embeddings (i.e. Theorem 2 of the main text).

Proposition (1 Hidden Layer Convolutional Network). Let $Z \in \mathbb{R}^{c \times m \times n}$ denote the feature prior. Let $*$ denote the neural network convolution operator and let $f_Z(\mathbf{W}) = A * \frac{\sqrt{c}}{q\sqrt{k}} \phi(B * Z)$ denote a 1 hidden layer convolutional network where B has k filters of size $q \times q \times c$ with circular padding, A has 1 filter of size $q \times q \times k$ with circular padding for odd q , ϕ is a homogeneous activation function of degree 1, and $c^2 = \frac{1}{\mathbb{E}_{u \sim \mathcal{N}(0,1)}[\phi(u)^2]}$. Let $K^{(0)}, \tilde{K}^{(0)}, \Sigma^{(0)} \in \mathbb{R}^{m \times n \times m \times n}$ such that:

$$\Sigma^{(0)}(i, j, i', j') = K^{(0)}(i, j, i', j') = \sum_{\ell=1}^c \sum_{-\frac{q+1}{2} \leq m, n \leq \frac{q+1}{2}} Z_{\ell, i+m, j+n} Z_{\ell, i'+m, j'+n}.$$

If M_{ij} and $M_{i'j'}$ are indicator matrices, then as $k \rightarrow \infty$, the CNTK for $f_Z(\mathbf{W})$ is given by:

$$K(M_{ij}, M_{i'j'}) = \frac{1}{q^2} \sum_{-\frac{q+1}{2} \leq a, b \leq \frac{q+1}{2}} \Sigma^{(1)}(i+a, j+b, i'+a, j'+b) + \dot{\Sigma}^{(1)}(i+a, j+b, i'+a, j'+b) K^{(0)}(i+a, j+b, i'+a, j'+b),$$

where

$$\Sigma^{(1)}(i, j, i', j') = \sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')} \hat{\phi} \left(\frac{\Sigma^{(0)}(i, j, i', j')}{\sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')}} \right),$$

$$\dot{\Sigma}^{(1)}(i, j, i', j') = \frac{d\hat{\phi}}{d\xi} \left(\frac{\Sigma^{(0)}(i, j, i', j')}{\sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')}} \right).$$

Proof. We provide the proof for the case of 1 input channel ($c = 1$) below. The proof follows analogously for the case of multiple input channels. Let $g(M) = \text{tr}(M^T f_Z(\mathbf{W}))$. Let $Y^{(\ell)}$ denote channel ℓ of $\frac{2}{\sqrt{k}} \phi(B * Z)$ and let $H = A * \frac{\sqrt{2}}{q\sqrt{k}} \phi(B * Z)$. We thus have that

$$Y_{ij}^{(\ell)} = \frac{\sqrt{c}}{q\sqrt{k}} \phi \left(\sum_{-\frac{q+1}{2} \leq a, b \leq \frac{q+1}{2}} Z_{i+a, j+b} B_{a,b}^{(\ell)} \right),$$

$$H_{ij} = \sum_{\ell=1}^k \sum_{-\frac{q+1}{2} \leq a, b \leq \frac{q+1}{2}} Y_{i+a, j+b}^{(\ell)} A_{a,b}^{(\ell)},$$

$$g(M) = \sum_{1 \leq i, j \leq d} M_{ij} H_{ij}.$$

Now we compute the partial derivatives of f with respect to the parameters $A_{a,b}^{(\ell)}$ and $B_{m,n}^{(\ell)}$:

$$\frac{\partial g(M_{ij})}{\partial A_{a,b}^{(\ell)}} = Y_{i+a, j+b}^{(\ell)},$$

$$\frac{\partial g(M_{ij})}{\partial B_{m,n}^{(\ell)}} = \sum_{-\frac{q+1}{2} \leq a, b \leq \frac{q+1}{2}} A_{a,b}^{(\ell)} \frac{\sqrt{c}}{q\sqrt{k}} \phi' \left(\sum_{-\frac{q+1}{2} \leq a', b' \leq \frac{q+1}{2}} Z_{i+a+a', j+b+b'} B_{a', b'}^{(\ell)} \right) Z_{i+a+m, j+b+n}.$$

As $k \rightarrow \infty$, the CNTK converges in probability to:

$$K(M_{ij}, M_{i'j'}) = \mathbb{E}_{A_{a,b}^{(\ell)}, B_{m,n}^{(\ell)} \sim \mathcal{N}(0,1)} \left[\sum_{\ell=1}^k \sum_{a,b} \frac{\partial g(M_{ij})}{\partial A_{a,b}^{(\ell)}} \frac{\partial g(M_{i'j'})}{\partial A_{a,b}^{(\ell)}} + \sum_{\ell=1}^k \sum_{m,n} \frac{\partial g(M_{ij})}{\partial B_{m,n}^{(\ell)}} \frac{\partial g(M_{i'j'})}{\partial B_{m,n}^{(\ell)}} \right]. \quad (\text{B.5})$$

This expression can be simplified as follows:

$$K(M_{ij}, M_{i'j'}) = \sum_{-\frac{q+1}{2} \leq a, b \leq \frac{q+1}{2}} \Sigma^{(1)}(i+a, j+b, i'+a, j'+b) \\ + \dot{\Sigma}^{(1)}(i+a, j+b, i'+a, j'+b) K^{(0)}(i+a, j+b, i'+a, j'+b),$$

where we have:

$$\Sigma^{(1)} = \frac{c}{q^2} \mathbb{E}_{B_{a',b'}^{(\ell)}} \left[\sum_{a,b} \phi \left(\sum_{-\frac{q+1}{2} \leq a', b' \leq \frac{q+1}{2}} Z_{i+a+a', j+b+b'} B_{a',b'}^{(\ell)} \right) \phi \left(\sum_{-\frac{q+1}{2} \leq a', b' \leq \frac{q+1}{2}} Z_{i'+a+a', j'+b+b'} B_{a',b'}^{(\ell)} \right) \right], \\ \dot{\Sigma}^{(1)} = \frac{c}{q^2} \mathbb{E}_{B_{a',b'}^{(\ell)}} \left[\sum_{a,b} \phi' \left(\sum_{-\frac{q+1}{2} \leq a', b' \leq \frac{q+1}{2}} Z_{i+a+a', j+b+b'} B_{a',b'}^{(\ell)} \right) \phi' \left(\sum_{-\frac{q+1}{2} \leq a', b' \leq \frac{q+1}{2}} Z_{i'+a+a', j'+b+b'} B_{a',b'}^{(\ell)} \right) \right].$$

Lastly, we reduce the above expressions by substituting in the values for $\Sigma^{(0)}$ from the statement of the proposition. Namely, let

$$u = \sum_{a,b} \phi \left(\sum_{-\frac{q+1}{2} \leq a', b' \leq \frac{q+1}{2}} Z_{i+a+a', j+b+b'} B_{a',b'}^{(\ell)} \right), \\ v = \sum_{a,b} \phi \left(\sum_{-\frac{q+1}{2} \leq a', b' \leq \frac{q+1}{2}} Z_{i'+a+a', j'+b+b'} B_{a',b'}^{(\ell)} \right).$$

Then, the above expressions for $\Sigma^{(1)}$, $\dot{\Sigma}^{(1)}$ simplify to:

$$\Sigma^{(1)} = \frac{c}{q^2} \mathbb{E}_{B_{a',b'}^{(\ell)}} [\phi(u)\phi(v)], \\ \dot{\Sigma}^{(1)} = \frac{c}{q^2} \mathbb{E}_{B_{a',b'}^{(\ell)}} [\phi'(u)\phi'(v)].$$

Hence, we can use the formula for the dual activation of the ReLU to conclude that:

$$\Sigma^{(1)}(i, j, i', j') = \frac{1}{q^2} \sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')} \check{\phi} \left(\frac{\Sigma^{(0)}(i, j, i', j')}{\sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')}} \right), \\ \dot{\Sigma}^{(1)}(i, j, i', j') = \frac{1}{q^2} \frac{d\check{\phi}}{d\xi} \left(\frac{\Sigma^{(0)}(i, j, i', j')}{\sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')}} \right).$$

Lastly, we complete the proof by substituting these expressions for $\Sigma^{(1)}$, $\dot{\Sigma}^{(1)}$ into the expression for $K(M_{ij}, M_{i'j'})$ above. \square

As implied by Proposition 1 above, the CNTK is a functional of *pairs of coordinates* of images, while the usual CNTK for classification operates on pairs of images [9]. To be more specific, consider the setting where the target matrix Y is in $\mathbb{R}^{m \times n}$. Then, the CNTK for matrix completion that we compute lies in $\mathbb{R}^{mn \times mn}$. On the other hand, when given n images for classification, the CNTK computed in [9] lies in $\mathbb{R}^{n \times n}$ and does not depend on the image size.

J Equivalence with Semi-Supervised Learning for the CNTK

In the following, we present the statement and proof of Theorem 2 from the main text with the precise form for $\tilde{\psi}$.

Theorem. Consider a convolutional network, $f_Z(\mathbf{W})$, with d hidden layers with homogeneous activation of degree 1 and in which all filters have size q and circular padding. Let $Z \in \mathbb{R}^{c \times m \times n}$ satisfy:

$$\sum_{\ell=1}^c \sum_{-\alpha \leq a, b \leq \alpha} Z_{\ell, i+a, j+b} Z_{\ell, i'+a, j'+b} = \psi(|i-i'|, |j-j'|)$$

for some $\psi : \mathbb{R}^2 \rightarrow \mathbb{R}$ with maximum at $(0,0)$ and $\alpha = \frac{q-1}{2}$ (odd q). Then as the number of convolutional filters per layer goes to infinity, the CNTK is given by:

$$\begin{aligned} K_d(M_{ij}, M_{i'j'}) &= \tilde{\psi}(|i-i'|, |j-j'|) \\ &= \check{\phi}^{(d)} \left(\frac{\psi(|i-i'|, |j-j'|)}{\psi(0,0)} \right) \psi(0,0) \\ &\quad + K_{d-1}(M_{ij}, M_{i'j'}) \frac{d\check{\phi}}{d\xi} \left(\check{\phi}^{(d-1)} \left(\frac{\psi(|i-i'|, |j-j'|)}{\psi(0,0)} \right) \right), \end{aligned}$$

where $\check{\phi}$ is the dual activation of ϕ , $\check{\phi}^{(d)}(\xi) = \check{\phi}(\check{\phi}^{(d-1)}(\xi))$ with $\check{\phi}^{(0)}(\xi) = \xi$, and $K_0(M_{ij}, M_{i'j'}) = \psi(|i-i'|, |j-j'|)$.

Proof. We prove this by induction on the number of hidden layers d . We begin with the base case for $d = 1$: The proof for this case follows from the proof of the Proposition for 1 hidden convolutional networks in SI Appendix G. Namely, we have:

$$\begin{aligned} K(M_{ij}, M_{i'j'}) &= \sum_{-\frac{q-1}{2} \leq a, b \leq \frac{q-1}{2}} \Sigma^{(1)}(i+a, j+b, i'+a, j'+b) \\ &\quad + \dot{\Sigma}^{(1)}(i+a, j+b, i'+a, j'+b) K^{(0)}(i+a, j+b, i'+a, j'+b), \end{aligned}$$

where

$$\begin{aligned} \Sigma^{(1)}(i, j, i', j') &= \frac{1}{q^2} \sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')} \check{\phi} \left(\frac{\Sigma^{(0)}(i, j, i', j')}{\sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')}} \right), \\ \dot{\Sigma}^{(1)}(i, j, i', j') &= \frac{1}{q^2} \frac{d\check{\phi}}{d\xi} \left(\frac{\Sigma^{(0)}(i, j, i', j')}{\sqrt{\Sigma^{(0)}(i, j, i, j) \Sigma^{(0)}(i', j', i', j')}} \right). \end{aligned}$$

Now since $\Sigma^{(0)}(i, j, i', j') = \psi(|i-i'|, |j-j'|)$, we conclude that

$$\begin{aligned} \Sigma^{(1)}(i, j, i', j') &= \frac{1}{q^2} \psi(0,0) \check{\phi} \left(\frac{\psi(|i-i'|, |j-j'|)}{\psi(0,0)} \right), \\ \dot{\Sigma}^{(1)} &= \frac{1}{q^2} \frac{d\check{\phi}}{d\xi} \left(\frac{\psi(|i-i'|, |j-j'|)}{\psi(0,0)} \right). \end{aligned}$$

Substituting the above into the expression for $K(M_{ij}, M_{i'j'})$, we obtain

$$\begin{aligned} K(M_{ij}, M_{i'j'}) &= \frac{1}{q^2} \sum_{-\frac{q-1}{2} \leq a, b \leq \frac{q-1}{2}} \psi(0,0) \check{\phi} \left(\frac{\psi(|i-i'|, |j-j'|)}{\psi(0,0)} \right) \\ &\quad + \psi(|i-i'|, |j-j'|) \frac{d\check{\phi}}{d\xi} \left(\frac{\psi(|i-i'|, |j-j'|)}{\psi(0,0)} \right). \end{aligned}$$

Note that the summand no longer depends on a, b , and thus we conclude that

$$K(M_{ij}, M_{i'j'}) = \psi(0,0) \check{\phi} \left(\frac{\psi(|i-i'|, |j-j'|)}{\psi(0,0)} \right) + \frac{d\check{\phi}}{d\xi} \left(\frac{\psi(|i-i'|, |j-j'|)}{\psi(0,0)} \right) \psi(|i-i'|, |j-j'|),$$

which completes the base case.

For the inductive step, we assume that the following holds for depth $d - 1$:

$$\begin{aligned}\Sigma^{(d-1)}(M_{ij}, M_{i'j'}) &= \frac{1}{q^2} \check{\phi}^{(d-1)} \left(\frac{\psi(|i - i'|, |j - j'|)}{\psi(0, 0)} \right) \psi(0, 0), \\ \dot{\Sigma}^{(d-1)}(M_{ij}, M_{i'j'}) &= \frac{1}{q^2} \frac{d\check{\phi}}{d\xi} \left(\check{\phi}^{(d-2)} \left(\frac{\psi(|i - i'|, |j - j'|)}{\psi(0, 0)} \right) \right), \\ K_{d-1}(M_{ij}, M_{i'j'}) &= q^2 \Sigma^{(d-1)}(M_{ij}, M_{i'j'}) + q^2 K_{d-2}(M_{ij}, M_{i'j'}) \dot{\Sigma}^{(d-1)}(M_{ij}, M_{i'j'}),\end{aligned}$$

and assume that $K_{d-1}(M_{ij}, M_{i'j'}) = K_{d-1}(M_{i+a, j+b}, M_{i'+a, j'+b})$ for any $a, b \in \mathbb{Z}$ satisfying $i + a, i' + a \in [m]$ and $j + b, j' + b \in [n]$ (i.e. assume that K_{d-1} is *shift invariant*). Now, let $S^{(d-1)}(M_{ij}, M_{i'j'})$ be defined as follows:

$$S^{(d-1)}(M_{ij}, M_{i'j'}) = \sum_{-\frac{q-1}{2} \leq a, b, \leq \frac{q-1}{2}} \Sigma^{(d-1)}(M_{i+a, j+b}, M_{i'+a, j'+b}) = \check{\phi}^{(d-1)} \left(\frac{\psi(|i - i'|, |j - j'|)}{\psi(0, 0)} \right) \psi(0, 0).$$

Then, by the derivation of the CNTK in [9], we obtain

$$\begin{aligned}\Sigma^{(d)}(M_{ij}, M_{i'j'}) &= \frac{1}{q^2} \check{\phi} \left(\frac{S^{(d-1)}(M_{ij}, M_{i'j'})}{\sqrt{S^{(d-1)}(M_{ij}, M_{ij}) S^{(d-1)}(M_{i'j'}, M_{i'j'})}} \right) \sqrt{S^{(d-1)}(M_{ij}, M_{ij}) S^{(d-1)}(M_{i'j'}, M_{i'j'})} \\ &= \frac{1}{q^2} \check{\phi} \left(\check{\phi}^{(d-1)} \left(\frac{\psi(|i - i'|, |j - j'|)}{\psi(0, 0)} \right) \right) \psi(0, 0),\end{aligned}$$

where the last equality follows from the fact that $\check{\phi}(1) = 1$. Following an analogous derivation for $\dot{\Sigma}^{(d-1)}$, we obtain that

$$\dot{\Sigma}^{(d)}(M_{ij}, M_{i'j'}) = \frac{1}{q^2} \frac{d\check{\phi}}{d\xi} \left(\check{\phi}^{(d-1)} \left(\frac{\psi(|i - i'|, |j - j'|)}{\psi(0, 0)} \right) \right).$$

Hence, the CNTK $K_d(M_{ij}, M_{i'j'})$ is given by:

$$\begin{aligned}K_d(M_{ij}, M_{i'j'}) &= \sum_{-\frac{q-1}{2} \leq a, b, \leq \frac{q-1}{2}} \Sigma^{(d)}(M_{i+a, j+b}, M_{i'+a, j'+b}) \\ &\quad + K_{d-1}(M_{i+a, j+b}, M_{i'+a, j'+b}) \dot{\Sigma}^{(d)}(M_{i+a, j+b}, M_{i'+a, j'+b}) \\ &= q^2 \Sigma^{(d)}(M_{ij}, M_{i'j'}) \\ &\quad + q^2 K_{d-1}(M_{ij}, M_{i'j'}) \dot{\Sigma}^{(d)}(M_{ij}, M_{i'j'}),\end{aligned}$$

where the last line follows from the shift invariance of K_{d-1} . Lastly, we have that K_d is shift invariant since all of the terms $\Sigma^{(d)}$, $\dot{\Sigma}^{(d)}$ and K_{d-1} are shift invariant. Hence, the induction is complete and the theorem follows. \square

K Derivation of the CNTK for Matrix Completion with Modern Architectures

Below, we derive the CNTK for networks with fixed linear transformations. We note a similar formula appears in the Appendix of [179], but does not appear to be derived for the cases of nearest neighbor upsampling, nearest neighbor downsampling, and bilinear upsampling.

Proposition. *Let $g(M) = \text{tr}(M^T A f_Z(\mathbf{W}))$ denote a neural network where $A \in \mathbb{R}^{m \times n \times pq}$ is a fixed (i.e. non-trainable) linear transformation and $f_Z(\mathbf{W})$ is a convolutional network under the NTK parameterization³.*

³We assume A operates on the vectorized version of $f_Z(\mathbf{W})$ and then the output is reshaped to size $m \times n$ before multiplication by M^T .

Then the CNTK, $K \in \mathbb{R}^{mn \times mn}$, for g is given by:

$$K = AK_f A^T \implies K(M_{ij}, M_{i'j'}) = \sum_{a=1}^{pq} \sum_{b=1}^{pq} A_{v(i,j),a} A_{v(i',j'),b} K_f(M_{v_1^{-1}(a),v_2^{-1}(a)}, M_{v_1^{-1}(b),v_2^{-1}(b)}),$$

where $v : \mathbb{R}^2 \rightarrow \mathbb{R}$ is the bijective map from a coordinate (i, j) in a matrix B to its position in the vectorized version of B and $K_f \in \mathbb{R}^{pq \times pq}$ is the CNTK for f .

Proof. Let w_p denote a weight in f and let \mathbf{w} denote the vector of all weights in f . We thus have that

$$\begin{aligned} \frac{\partial g(M)}{\partial w_p} &= \sum_{m,n} M_{m,n} \frac{\partial [Af_Z(\mathbf{W})]_{v(m,n)}}{\partial w_p} = \sum_{m,n} M_{m,n} \sum_{\ell=1}^{pq} A_{v(m,n),\ell} \frac{\partial f_Z(\mathbf{W})_\ell}{\partial w_p} \\ \implies K(M_{ij}, M_{i'j'}) &= \left\langle \sum_{a=1}^{pq} A_{v(i,j),a} \frac{\partial f_Z(\mathbf{W})_a}{\partial \mathbf{w}}, \sum_{b=1}^{pq} A_{v(i',j'),b} \frac{\partial f_Z(\mathbf{W})_b}{\partial \mathbf{w}} \right\rangle = AK_g A^T, \end{aligned}$$

which completes the proof. \square

While the Proposition above generally implies that a linear transformation requires evaluating a quadratic form when computing the CNTK, the matrix A corresponding to layers used in practice is typically extremely sparse. Hence, the required computation is simplified drastically, as is demonstrated by the following corollaries (the proofs follow directly from the proposition above).

Corollary (Downsampling through Strided Convolution). *Let $\Sigma^{(\ell)}, \dot{\Sigma}^{(\ell)}, K^{(\ell)} \in \mathbb{R}^{d \times d \times d \times d}$ correspond to the tensors used in the CNTK for a depth ℓ convolutional network. Then, using downsampling with a stride of 2 at step $\ell + 1$ maps the tensors to $\Sigma^{(\ell+1)}, \dot{\Sigma}^{(\ell+1)}, K^{(\ell+1)} \in \mathbb{R}^{\frac{d}{2} \times \frac{d}{2} \times \frac{d}{2} \times \frac{d}{2}}$ as follows: $\forall i, j, i', j' \equiv 0 \pmod{2}$,*

$$\begin{aligned} \Sigma^{(\ell+1)} \left(\frac{i}{2}, \frac{j}{2}, \frac{i'}{2}, \frac{j'}{2} \right) &= \Sigma^{(\ell)}(i, j, i', j'), \\ \dot{\Sigma}^{(\ell+1)} \left(\frac{i}{2}, \frac{j}{2}, \frac{i'}{2}, \frac{j'}{2} \right) &= \dot{\Sigma}^{(\ell)}(i, j, i', j'), \\ K^{(\ell+1)} \left(\frac{i}{2}, \frac{j}{2}, \frac{i'}{2}, \frac{j'}{2} \right) &= K^{(\ell)}(i, j, i', j'). \end{aligned}$$

Corollary (Nearest Neighbor Upsampling). *Let $\Sigma^{(\ell)}, \dot{\Sigma}^{(\ell)}, K^{(\ell)} \in \mathbb{R}^{\frac{d}{2} \times \frac{d}{2} \times \frac{d}{2} \times \frac{d}{2}}$ correspond to the tensors used in the CNTK for a depth ℓ convolutional network. Then, using nearest neighbor upsampling with a scale factor of 2 at step $\ell + 1$ transforms the tensors to $\Sigma^{(\ell+1)}, \dot{\Sigma}^{(\ell+1)}, K^{(\ell+1)} \in \mathbb{R}^{d \times d \times d \times d}$ as follows:*

$$\begin{aligned} \Sigma^{(\ell+1)}(i, j, i', j') &= \Sigma^{(\ell)} \left(\left\lfloor \frac{i}{2} \right\rfloor, \left\lfloor \frac{j}{2} \right\rfloor, \left\lfloor \frac{i'}{2} \right\rfloor, \left\lfloor \frac{j'}{2} \right\rfloor \right), \\ \dot{\Sigma}^{(\ell+1)}(i, j, i', j') &= \dot{\Sigma}^{(\ell)} \left(\left\lfloor \frac{i}{2} \right\rfloor, \left\lfloor \frac{j}{2} \right\rfloor, \left\lfloor \frac{i'}{2} \right\rfloor, \left\lfloor \frac{j'}{2} \right\rfloor \right), \\ K^{(\ell+1)}(i, j, i', j') &= K^{(\ell)} \left(\left\lfloor \frac{i}{2} \right\rfloor, \left\lfloor \frac{j}{2} \right\rfloor, \left\lfloor \frac{i'}{2} \right\rfloor, \left\lfloor \frac{j'}{2} \right\rfloor \right). \end{aligned}$$

The computation for bilinear upsampling (Ch. 2.4 of [68]) is presented below. We primarily use the structure of the updates to $\Sigma, \dot{\Sigma}, K$ to efficiently compute the CNTK when the channels of X are drawn i.i.d. from a stationary distribution.

When bilinearly upsampling (Ch. 2.4 of [68]) an image $A \in \mathbb{R}^{d \times d}$ to an image $\tilde{A} \in \mathbb{R}^{2d \times 2d}$, each coordinate of \tilde{A} is a linear combination of four coordinates of A . Namely for $\alpha = \frac{d-1}{2d-1}$,

$$\tilde{A}_{i,j} = \sum_{a,b \in \{0,1\}} \lambda_{a,b}^{(i,j)} A_{[\alpha i]+a, [\alpha j]+b},$$

and $\lambda_{a,b}^{i,j}$ is selected as follows. Let $r = \lfloor \alpha i \rfloor, c = \lfloor \alpha j \rfloor$ and let:

$$\begin{aligned} \ell_r &= \frac{r}{\alpha}, u_r = \frac{r+1}{\alpha}, \ell_c = \frac{c}{\alpha}, u_c = \frac{c+1}{\alpha}, \\ X &= [u_r - r, r - \ell_r], Y = [u_c - c, c - \ell_c], C = \frac{1}{(u_r - \ell_r)(u_c - \ell_c)}. \end{aligned}$$

Then, $\lambda_{a,b}^{(i,j)} = CX_a Y_b$ for $a, b \in \{0, 1\}$. The CNTK tensors are now transformed as follows.

Corollary 5 (Bilinear Upsampling). *Let $\Sigma^{(\ell)}, \dot{\Sigma}^{(\ell)}, K^{(\ell)} \in \mathbb{R}^{\frac{d}{2} \times \frac{d}{2} \times \frac{d}{2} \times \frac{d}{2}}$ correspond to the tensors used in the CNTK for a depth ℓ convolutional network. Then, using bilinear upsampling with a scale factor of 2 at step $\ell + 1$ transforms the tensors to $\Sigma^{(\ell+1)}, \dot{\Sigma}^{(\ell+1)}, K^{\ell+1} \in \mathbb{R}^{d \times d \times d \times d}$ as follows:*

$$\begin{aligned} \Sigma^{(\ell+1)}(i, j, i', j') &= \sum_{a,b \in \{0,1\}} \sum_{a',b' \in \{0,1\}} \lambda_{a,b}^{(i,j)} \lambda_{a',b'}^{(i',j')} \Sigma^{(\ell)}(\lfloor \alpha i \rfloor + a, \lfloor \alpha j \rfloor + b, \lfloor \alpha i' \rfloor + a', \lfloor \alpha j' \rfloor + b'), \\ \dot{\Sigma}^{(\ell+1)}(i, j, i', j') &= \sum_{a,b \in \{0,1\}} \sum_{a',b' \in \{0,1\}} \lambda_{a,b}^{(i,j)} \lambda_{a',b'}^{(i',j')} \dot{\Sigma}^{(\ell)}(\lfloor \alpha i \rfloor + a, \lfloor \alpha j \rfloor + b, \lfloor \alpha i' \rfloor + a', \lfloor \alpha j' \rfloor + b'), \\ K^{(\ell+1)}(i, j, i', j') &= \sum_{a,b \in \{0,1\}} \sum_{a',b' \in \{0,1\}} \lambda_{a,b}^{(i,j)} \lambda_{a',b'}^{(i',j')} K^{(\ell)}(\lfloor \alpha i \rfloor + a, \lfloor \alpha j \rfloor + b, \lfloor \alpha i' \rfloor + a', \lfloor \alpha j' \rfloor + b'). \end{aligned}$$

L Efficient Computation of the CNTK for High Resolution Images

Computing and storing the CNTK exactly for high resolution images is computationally prohibitive when using a naive approach. In particular, [42] notes that the kernel K for a 500×500 ($K \in \mathbb{R}^{500 \times 500 \times 500 \times 500}$) resolution image requires roughly 233GB of memory, which is infeasible on common hardware. In order to overcome these computational limitations, [179] uses the Nyström method [193] to approximate the kernel. In this section, we will demonstrate that we can compute the exact CNTK in a memory and run-time efficient manner for any convolutional neural network with circular padding, strided convolution, and nearest neighbor upsampling layers by using a feature prior Z that has infinitely many channels.

Our key insight is that once architecture is fixed, the the CNTK for low resolution images can be expanded to that for high resolution images. In particular, when the convolutional architecture can be applied to both images of resolution d_1 and d_2 with $d_2 > d_1$, we can expand the kernel for resolution d_1 , $K_{d_1} \in \mathbb{R}^{d_1 \times d_1 \times d_1 \times d_1}$, to a tensor of size $\mathbb{R}^{d_1 \times d_1 \times d_2 \times d_2}$, which can be indexed to match the entries of the kernel for resolution d_2 , $K_{d_2} \in \mathbb{R}^{d_2 \times d_2 \times d_2 \times d_2}$.

In order to expand the kernel for low resolution images to the one for high resolution images, we need only pad and permute the rows and columns of the low resolution matrix. We define the required operations formally below (using zero indexing for our matrices).

Definition 10 (Row and Column Rotation). *Let $\Pi_{i,j} : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}^{d \times d}$ such that $\Pi_{i,j}(A) = P_{\pi_i} A P_{\pi_j}$ where P_{π_ℓ} is a permutation matrix with permutation $\pi_\ell(i) = (i + \ell) \bmod d$.*

Definition 11 (Minimum Padding). *Let $M : \mathbb{R}^{d_1 \times d_1} \rightarrow \mathbb{R}^{d_2 \times d_2}$ with $d_2 \geq d_1$ such that $M(A) = \tilde{A}$, where*

$$\tilde{A}_{i,j} = \begin{cases} A_{i,j} & i < d_1, j < d_1 \\ \min_{a,b \in [d_1]} A_{a,b} & \text{otherwise} \end{cases}.$$

Example. *The operator $\Pi_{i,j}$ rotates the rows of A down by i and rotates the columns of A right by j as follows:*

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \implies \Pi_{1,2}(A) = \begin{bmatrix} A_{3,2} & A_{3,3} & A_{3,1} \\ A_{1,2} & A_{1,3} & A_{1,1} \\ A_{2,2} & A_{2,3} & A_{2,1} \end{bmatrix}.$$

Minimum padding $M : \mathbb{R}^{2 \times 2} \rightarrow \mathbb{R}^{4 \times 4}$ expands a matrix as follows:

$$A = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix} \implies M(A) = \begin{bmatrix} 0.1 & 0.2 & 0.1 & 0.1 \\ 0.3 & 0.4 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}.$$

The theorem below demonstrates how to construct the kernel for a high resolution image by expanding and indexing a low resolution kernel. We assume that all strided convolutional layers have a stride size of 2 in each direction and all upsampling layers have a scaling factor of 2. For the following theorem, we also write the kernel $K \in \mathbb{R}^{m \times n \times m \times n}$ as a 4 dimensional tensor $K \in \mathbb{R}^{m \times n \times m \times n}$, where $K(i, j, i', j') := K(M_{ij}, M_{i'j'})$.

Theorem (CNTK Expansion). *Let g denote a convolutional neural network with circular padding, s down-sampling with strided convolution layers and s nearest neighbor upsampling layers used to inpaint images in $\mathbb{R}^{2^{s+1} \times 2^{s+1}}$. Define the feature prior $Z^{(\ell)} = \{Z_p^{(\ell)}\}_{p=1}^{\infty} \subset \mathbb{R}^{2^\ell \times 2^\ell}$ for $\ell \in \mathbb{Z}_+$ such that:*

$$\sum_{p=1}^{\infty} Z_{p,i,j}^{(\ell)} Z_{p,i',j'}^{(\ell)} = \begin{cases} C_1 & i = i', j = j' \\ C_2 & \text{otherwise} \end{cases}. \quad (\text{B.6})$$

Let $d_2 = 2^{p_2}$ such that $p_2 > s + 1$. For $\alpha = 2^\beta$, let K_α denote the CNTK for g when used to inpaint images in $\mathbb{R}^{\alpha \times \alpha}$ with feature prior $Z^{(\beta)}$. Let $p = 2^s$, $i' = i \bmod p$, $j' = j \bmod p$. Then for $i, j \in [d_2]$, we compute $\tilde{K} \in \mathbb{R}^{p \times p \times d_2 \times d_2}$ as follows:

$$\tilde{K}(i', j', :, :) = \Pi_{i'-p, j'-p}(M(\Pi_{p-i', p-j'}(K_{s+1}[i', j', :, :])),$$

and we have:

$$K_{d_2}(i, j, :, :) = \Pi_{i-i', j-j'} \tilde{K}(i', j', :, :).$$

Proof. To provide intuition for the general case, we first prove the result for $s = 0$. Using the Proposition from SI Appendix I and the conditions on $Z^{(\ell)}$, we obtain

$$\Sigma^{(0)}(i, j, i', j') = K^{(0)}(i, j, i', j') = \begin{cases} q^2 C_1 & \text{if } i = i', j' = j' \\ q^2 C_2 & \text{otherwise} \end{cases}.$$

Hence for any $\ell, \ell' \geq 1$ with $\ell' < \ell$, we conclude that $K_\ell(i, j, i', j') = K_{\ell'}(a, b, a', b')$ when $(i, j) \neq (i', j')$ and $(a, b) \neq (a', b')$, and $K_\ell(i, j, i, j) = K_{\ell'}(a, b, a, b)$ for all $i, j \in [2^\ell]$ and $a, b \in [2^{\ell'}]$. Hence, by permuting rows, columns and minimum padding $K_{\ell'}$, we can recover the kernel for K_ℓ . Note that for $\ell = 0$, we do not ever record a kernel entry for the case where $(i, j) \neq (i', j')$ and so minimum padding would pad with the incorrect minimum value of $K_0(0, 0, 0, 0)$. This is why we need to expand up from the kernel for images of dimension 2^{s+1} and not just from the kernel for images of dimension 2^s .

For $s > 0$, we rely on the nearest neighbor upsampling and downsampling corollaries from SI Appendix K to understand which entries of $K_\ell(i, j, i', j')$ are equal to $K_{\ell'}(a, b, a', b')$. Since $Z^{(\ell)}, Z^{(\ell')}$ have the same range $\{C_1, C_2\}$ of channel-wise products, it suffices to identify the elements of $K_{\ell'}$ that are equal. These elements will then naturally be equal in K_ℓ after minimum padding.

From [42], we have that down-sampling through strided convolution preserves stationarity, and so after t downsampling and convolutional layers, we again have that $K_\ell^{(t)}(i, j, i', j') = K_{\ell'}^{(t)}(a, b, a', b')$ when $(i, j) \neq (i', j')$ and $(a, b) \neq (a', b')$, and $K_\ell^{(t)}(i, j, i, j) = K_{\ell'}^{(t)}(a, b, a, b)$ for all $i, j \in [2^\ell]$ and $a, b \in [2^{\ell'}]$.

In general, upsampling (including nearest neighbor upsampling) does not preserve stationarity, as is discussed in [42]. However, nearest neighbor upsampling preserves equality (up to permutation) between $K_\ell(i, j, :, :)$ and $K_\ell(i', j', :, :)$ provided that $i \equiv i' \pmod{2^s}$ and $j \equiv j' \pmod{2^s}$. This follows immediately from analyzing the output after nearest neighbor upsampling in the original image space. In the following,

we provide an example.

Example. Consider the output of nearest neighbor upsampling a single channel $Y \in \mathbb{R}^{2 \times 2}$ to $\tilde{Y} \in \mathbb{R}^{4 \times 4}$:

$$Y = \begin{bmatrix} Y_{0,0} & Y_{0,1} \\ Y_{1,0} & Y_{1,1} \end{bmatrix} \implies \tilde{Y} = \begin{bmatrix} Y_{0,0} & Y_{0,0} & Y_{0,1} & Y_{0,1} \\ Y_{0,0} & Y_{0,0} & Y_{0,1} & Y_{0,1} \\ Y_{1,0} & Y_{1,0} & Y_{1,1} & Y_{1,1} \\ Y_{1,0} & Y_{1,0} & Y_{1,1} & Y_{1,1} \end{bmatrix}.$$

From the stationarity of $Z^{(\ell)}$ and since convolution and downsampling layers preserve stationarity, we have that the CNTK for the above output $K_2(i, j, :, :)$ equals (up to permutation) $K_2(i', j', :, :)$ whenever $i \equiv i' \pmod{2}$ and $j \equiv j' \pmod{2}$ since the corresponding entries in \tilde{Y} have identical patterns of neighbors (i.e. a row or column permutation by 2^s does not affect the sums involved in the kernel computation).

Thus, we conclude that the range of entries in $K_{\ell'}(a, b, :, :)$ and $K_{\ell}(i, j, :, :)$ are equal whenever both $i \equiv a \pmod{2^s}$ and $b \equiv j \pmod{2^s}$. To complete the proof, we just permute and minimum pads the entries of $K_{\ell'}(a, b, :, :)$ to align the expanded matrix such that entry $K_{\ell'}(a, b, a, b)$ corresponds to $K_{\ell}(i, j, i, j)$ in the expanded matrix. \square

Remarks. Note that the expansion trick provided in the theorem above solely depends on (1) the number of downsampling and nearest neighbor upsampling layers; (2) the feature prior Z having special structure as described in (B.6); and (3) the convolutional layers using circular padding. It importantly does *not* depend on the number of layers, type of homogeneous activation function (i.e. ReLU or leakyReLU), or size of the convolutional filters used. Hence our expansion technique can be used on a range of architectures, as we also demonstrate in Section 4 of the main text. The permutations $\Pi_{p-i', p-j'}$, $\Pi_{i'-p, j'-p}$ used to compute \tilde{K} are essentially used to ensure that we perform minimum padding appropriately for kernel values at the kernel’s edges. Lastly, when there are s downsampling and upsampling layers, the smallest image size we can expand from is an image of size $2^{s+1} \times 2^{s+1}$. We cannot use images of size 2^s since the corresponding kernel will not contain the same minimum value as that for images of size 2^{s+1} .

M Experimental Details for Image Inpainting

In the following, we describe the hyperparameters used for training neural networks and solving kernel regression with the CNTK on the considered image inpainting and image reconstruction tasks.

Large Hole Inpainting

For all large hole inpainting experiments, we used the autoencoder architecture from [42] that has 6 downsampling and upsampling layers with no skip connections. On all images other than the “library” image, we trained using the Adam optimizer [101] for 1000 epochs with a learning rate 10^{-2} . For the “library” image, we trained using the Adam optimizer for 6000 epochs with a learning rate of 10^{-2} . We used a random seed of 15 for all libraries. For implementing Adam with Langevin dynamics, we used the code and data from [42] directly. We performed optimal early stopping for all neural networks, i.e. we chose the reconstruction that has the closest match in PSNR to the ground truth. While impossible to perform in practice, optimal early stopping allows us to compare the CNTK with the best possible result from the neural network.

For solving kernel regression with the CNTK, we trained using EigenPro [124] for 10 epochs, i.e., we did not early stop for large hole inpainting tasks. We scaled all kernels by a factor of 0.5 to ensure convergence with EigenPro.

Image Reconstruction

Below we list the architectures and training procedure for each image. For the neural networks, we always trained for 6000 epochs using Adam with a learning rate of 10^{-3} , which is the learning rate used in [185]. All neural networks have 128 convolutional filters per layer as is the case in [185]. We trained the CNTK for the corresponding architecture with EigenPro for 50 epochs, unless otherwise specified. The architectures

used nearest neighbor upsampling, unless otherwise specified. We observed that training longer or, ideally, direct solving kernel regression with the CNTK for networks with nearest neighbor upsampling led to the best PSNR results for image reconstruction tasks. This is consistent with [185] in which networks for image reconstruction are trained twice as long as those for large hole inpainting. A direct solve was only computationally feasible on 256×256 resolution images.

- “Barbara”: We use a network with 2 downsampling and upsampling layers.
- “Boat”: We use a network with 6 downsampling and upsampling layers. We train the CNTK for 100 epochs.
- “Camera Man”: We use a network with 6 downsampling and upsampling layers. We train the CNTK for 100 epochs.
- “Couple”: We use a network with 6 downsampling and upsampling layers. We train the CNTK for 100 epochs.
- “Finger”: We use a network with 3 downsampling and upsampling layers. We train the CNTK for 100 epochs.
- “Hill”: We use a network with 6 downsampling and upsampling layers.
- “House”: We use a network with 6 downsampling and upsampling layers. We solve kernel regression exactly using the numpy solve function.
- “Lena”: We use a network with 6 downsampling and upsampling layers.
- “Man”: We use a network with 6 downsampling and upsampling layers.
- “Montage”: We use a network with 6 downsampling and upsampling layers. We solve kernel regression exactly using the numpy solve method.
- “Peppers”: We use a network with 5 downsampling and upsampling layers with bilinear upsampling. We solve kernel regression exactly using the numpy solve method, but add diagonal regularization from [109]. In particular, for kernel $K \in \mathbb{R}^{p \times p}$, we add $\frac{4 \cdot 10^{-5}}{p} \text{tr}(K) I_{p \times p}$ to the kernel before using the numpy solve function.

SEED 149		DNPP	Ours (DNPP Prior)	DNPP	Ours (DNPP Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.298	0.302	0.544	0.548	
2	0.288	0.291	0.534	0.538	
3	0.296	0.298	0.541	0.545	
4	0.289	0.292	0.535	0.539	
5	0.290	0.293	0.535	0.539	
6	0.304	0.308	0.546	0.551	
7	0.293	0.296	0.540	0.545	
8	0.289	0.291	0.537	0.541	
9	0.308	0.311	0.552	0.556	
10	0.312	0.316	0.554	0.559	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.556	0.559

SEED 10		DNPP	Ours (DNPP Prior)	DNPP	Ours (DNPP Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.299	0.301	0.545	0.549	
2	0.306	0.309	0.550	0.554	
3	0.292	0.294	0.538	0.542	
4	0.305	0.309	0.549	0.554	
5	0.284	0.286	0.531	0.535	
6	0.283	0.286	0.530	0.535	
7	0.298	0.302	0.542	0.547	
8	0.297	0.299	0.543	0.547	
9	0.303	0.306	0.545	0.550	
10	0.302	0.305	0.545	0.550	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.556	0.559

SEED 53		DNPP	Ours (DNPP Prior)	DNPP	Ours (DNPP Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.306	0.310	0.549	0.554	
2	0.287	0.290	0.532	0.537	
3	0.299	0.302	0.544	0.548	
4	0.292	0.295	0.538	0.542	
5	0.288	0.291	0.535	0.540	
6	0.302	0.304	0.546	0.550	
7	0.293	0.296	0.539	0.544	
8	0.295	0.298	0.541	0.546	
9	0.304	0.307	0.546	0.551	
10	0.294	0.297	0.538	0.543	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.555	0.559

SEED 77		DNPP	Ours (DNPP Prior)	DNPP	Ours (DNPP Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.310	0.313	0.552	0.557	
2	0.285	0.288	0.533	0.537	
3	0.289	0.292	0.537	0.542	
4	0.296	0.299	0.539	0.543	
5	0.303	0.307	0.548	0.552	
6	0.297	0.300	0.541	0.546	
7	0.290	0.292	0.536	0.540	
8	0.306	0.309	0.549	0.554	
9	0.301	0.304	0.545	0.550	
10	0.283	0.285	0.531	0.535	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.555	0.559

SEED 1928		DNPP	Ours (DNPP Prior)	DNPP	Ours (DNPP Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.302	0.305	0.546	0.551	
2	0.299	0.302	0.543	0.547	
3	0.302	0.305	0.546	0.550	
4	0.293	0.295	0.539	0.543	
5	0.301	0.304	0.543	0.548	
6	0.295	0.298	0.540	0.544	
7	0.301	0.304	0.544	0.549	
8	0.300	0.302	0.544	0.548	
9	0.286	0.288	0.535	0.539	
10	0.290	0.294	0.538	0.543	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.556	0.559

Figure B-1: Comparison between DNPP and using the output of DNPP as a feature prior. Using our method with the output of DNPP as a feature prior leads to an improvement in all metrics across every round of 10-fold cross validation in 5 seeds.

SEED 149		FaLRTC	Ours (FaLRTC Prior)	FaLRTC	Ours (FaLRTC Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.290	0.313	0.541	0.562	
2	0.281	0.304	0.530	0.550	
3	0.285	0.307	0.535	0.556	
4	0.279	0.301	0.528	0.548	
5	0.281	0.304	0.530	0.551	
6	0.293	0.317	0.542	0.563	
7	0.280	0.303	0.532	0.553	
8	0.287	0.310	0.536	0.556	
9	0.294	0.317	0.545	0.565	
10	0.285	0.310	0.542	0.565	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.544	0.569

SEED 10		FaLRTC	Ours (FaLRTC Prior)	FaLRTC	Ours (FaLRTC Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.289	0.312	0.541	0.561	
2	0.290	0.313	0.541	0.561	
3	0.284	0.307	0.533	0.553	
4	0.294	0.317	0.543	0.564	
5	0.276	0.299	0.525	0.545	
6	0.275	0.298	0.525	0.546	
7	0.287	0.311	0.536	0.557	
8	0.288	0.311	0.536	0.558	
9	0.290	0.313	0.541	0.562	
10	0.285	0.309	0.540	0.561	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.545	0.570

SEED 53		FaLRTC	Ours (FaLRTC Prior)	FaLRTC	Ours (FaLRTC Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.290	0.313	0.544	0.565	
2	0.277	0.299	0.525	0.546	
3	0.288	0.310	0.537	0.557	
4	0.283	0.306	0.533	0.554	
5	0.279	0.302	0.529	0.550	
6	0.289	0.312	0.542	0.562	
7	0.284	0.308	0.535	0.556	
8	0.288	0.311	0.539	0.559	
9	0.293	0.316	0.543	0.564	
10	0.287	0.311	0.535	0.556	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.545	0.570

SEED 77		FaLRTC	Ours (FaLRTC Prior)	FaLRTC	Ours (FaLRTC Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.292	0.315	0.545	0.565	
2	0.280	0.303	0.532	0.552	
3	0.285	0.308	0.536	0.556	
4	0.288	0.311	0.537	0.556	
5	0.294	0.318	0.543	0.564	
6	0.288	0.311	0.538	0.558	
7	0.281	0.304	0.530	0.551	
8	0.292	0.316	0.544	0.565	
9	0.290	0.313	0.539	0.560	
10	0.271	0.294	0.521	0.542	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.545	0.570

SEED 1928		FaLRTC	Ours (FaLRTC Prior)	FaLRTC	Ours (FaLRTC Prior)
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.	
1	0.285	0.308	0.539	0.560	
2	0.286	0.310	0.538	0.559	
3	0.288	0.311	0.537	0.559	
4	0.281	0.304	0.530	0.551	
5	0.288	0.311	0.537	0.558	
6	0.285	0.308	0.533	0.553	
7	0.289	0.312	0.537	0.558	
8	0.284	0.307	0.533	0.555	
9	0.288	0.310	0.536	0.556	
10	0.279	0.303	0.532	0.553	

DNPP	Ours (DNPP Prior)
Pearson r	Pearson r
0.544	0.569

Figure B-2: Comparison between FaLRTC and using the output of FaLRTC as a feature prior. Using our method with the output of FaLRTC as a feature prior leads to an improvement in all metrics across every round of 10-fold cross validation in 5 seeds.

Sparse Regime (< 150 profiles per cell type)

SEED ID	FaLRTC	DNPP	FaLRTC	DNPP
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.
1	0.289	0.279	0.541	0.533
2	0.290	0.270	0.541	0.526
3	0.284	0.261	0.533	0.519
4	0.294	0.290	0.543	0.543
5	0.276	0.273	0.525	0.525
6	0.275	0.277	0.525	0.528
7	0.287	0.283	0.536	0.535
8	0.288	0.281	0.536	0.534
9	0.280	0.285	0.541	0.537
10	0.285	0.286	0.540	0.544

FaLRTC	DNPP
Pearson r	Pearson r
0.545	0.539

Dense Regime (> 150 profiles per cell type)

SEED ID	FaLRTC	DNPP	FaLRTC	DNPP
CV Round \ Metric	Mean R ²	Mean R ²	Mean Cos. Sim.	Mean Cos. Sim.
1	0.292	0.310	0.522	0.550
2	0.290	0.307	0.517	0.545
3	0.289	0.304	0.518	0.545
4	0.286	0.301	0.513	0.542
5	0.290	0.310	0.518	0.550
6	0.290	0.304	0.519	0.545
7	0.270	0.282	0.499	0.526
8	0.299	0.309	0.526	0.549
9	0.287	0.305	0.515	0.545
10	0.274	0.287	0.503	0.530

FaLRTC	DNPP
Pearson r	Pearson r
0.542	0.560

Figure B-3: Comparison between DNPP and FaLRTC in (a) the sparse regime (< 150 profiles per cell type) and (b) the dense regime (> 150 profiles per cell type). We observe that FaLRTC outperforms DNPP in almost every fold for all performance metrics in the sparse regime. On the other hand DNPP outperforms FaLRTC in the dense regime in every fold for all performance metrics. This result demonstrates that DNPP can be improved drastically in the sparse regime.

(A) PSNR Comparison*

Image	CNTK	Neural Network + Sigmoid Last Layer + BatchNorm	Biharmonic	Neural Network + Sigmoid Last Layer + BatchNorm + Adam + LD
Museum	31.90	30.69	30.03	34.40
White Car	28.66	28.73	26.20	28.87
Bicycle	27.67	28.57	28.67	30.89
Chair	29.87	29.88	27.81	32.81
Car Field	28.91	29.94	27.67	30.11
Rider	29.20	27.76	29.38	29.71
Library	21.73	20.76	17.71	21.79
Vase	31.75	31.51	28.96	32.27
Pool	34.51	33.08	34.62	35.70
Average	29.36	28.99	27.89	30.73

*Higher is better with a max of 100.

(B) SSIM Comparison**

Image	CNTK	Neural Network + Sigmoid Last Layer + BatchNorm	Biharmonic	Neural Network + Sigmoid Last Layer + BatchNorm + Adam + LD
Museum	0.915	0.919	0.915	0.929
White Car	0.922	0.924	0.913	0.921
Bicycle	0.955	0.962	0.914	0.968
Chair	0.954	0.946	0.934	0.964
Car Field	0.887	0.887	0.883	0.887
Rider	0.940	0.930	0.921	0.938
Library	0.884	0.864	0.860	0.900
Vase	0.977	0.977	0.976	0.979
Pool	0.974	0.971	0.967	0.968
Average	0.934	0.931	0.920	0.939

*Higher is better with a max of 1.

(C) Runtime Comparison

Image	CNTK (Time)	Neural Network + Sigmoid Last Layer + BatchNorm (Time)	Biharmonic (Time)	Neural Network + Sigmoid Last Layer + BatchNorm + Adam + LD (Time)
Museum	102	91	133	3213
White Car	78	75	34	2947
Bicycle	78	69	28	2699
Chair	76	78	52	5340
Car Field	73	69	86	2885
Rider	78	75	27	2558
Library	341	122	141	3734
Vase	45	56	4	2358
Pool	260	108	30	2699
Average	125	83	59	3159

Figure B-4: A comparison of PSNR, SSIM, and runtime for large hole image inpainting using our framework (CNTK), corresponding finite width neural networks, and biharmonic inpainting. We observe that the CNTK outperforms (in PSNR and SSIM) on average both biharmonic inpainting and finite neural networks with sigmoid last layer and batch normalization layers while maintaining a runtime that is comparable to these methods. The last column illustrates that using more advanced techniques such as Adam with Langevin dynamics [42] can be used to boost the performance of neural networks, but at additional computational cost.

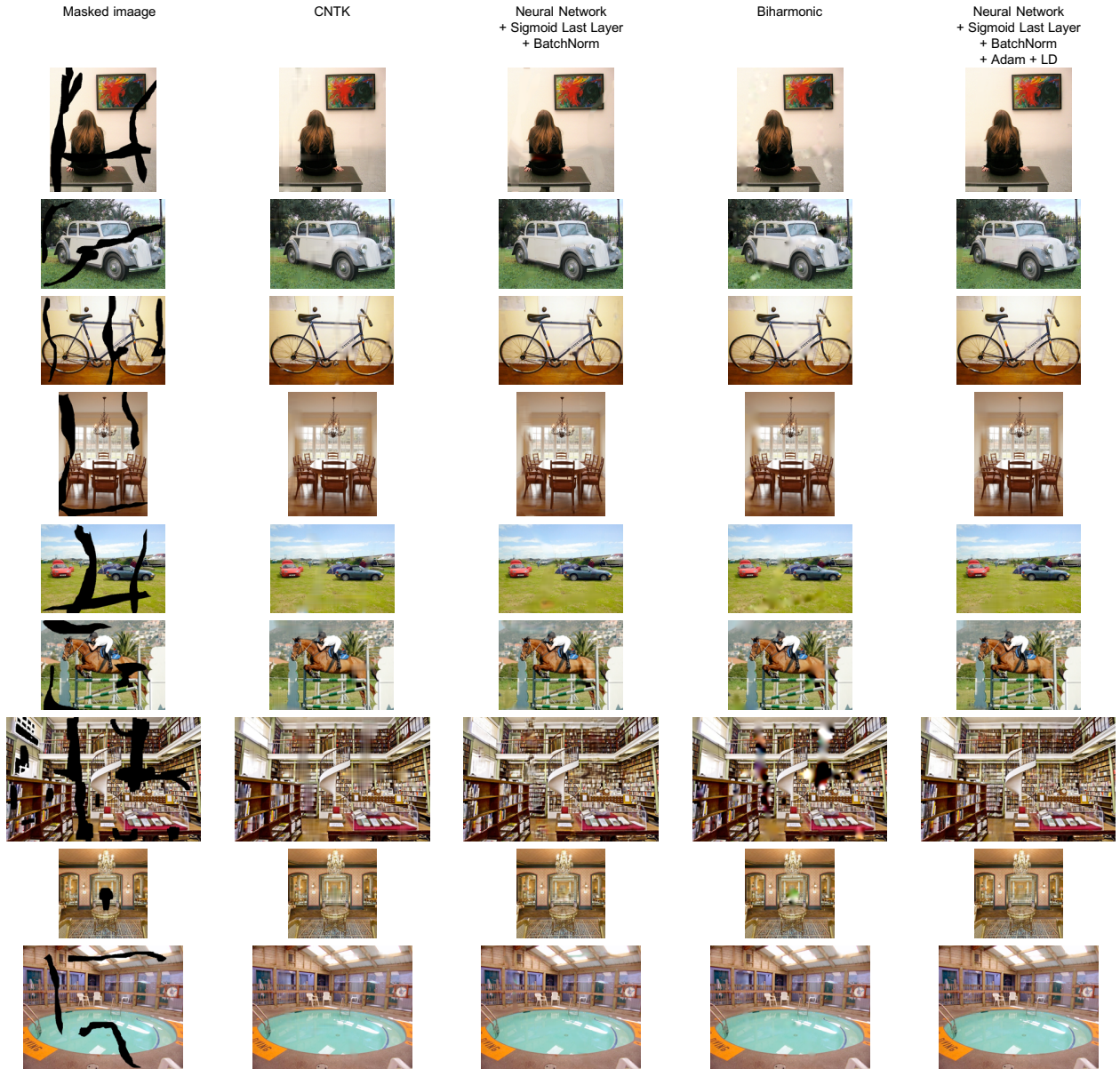


Figure B-5: A qualitative comparison of large hole image inpainting using our framework (CNTK), corresponding finite width neural networks, and biharmonic inpainting. See SI Fig. S4 for the corresponding quantitative comparison.

Image	CNTK	Neural Network + Sigmoid Last Layer + BatchNorm	Biharmonic
Barbara	26.61	24.78	26.75
Boat	31.99	31.39	32.07
Camera Man	27.65	27.28	28.04
Couple	32.65	31.46	32.27
Finger	31.71	28.46	33.49
Hill	33.46	32.02	33.39
House	36.10	34.00	35.60
Lena	34.96	35.07	36.08
Man	32.15	31.39	32.91
Montage	29.98	28.71	28.39
Peppers	32.25	32.25	30.33
Average	31.77	30.62	31.76

*Higher is better with a max of 100.

Image	CNTK	Neural Network + Sigmoid Last Layer + BatchNorm	Biharmonic
Barbara	0.843	0.673	0.865
Boat	0.873	0.883	0.882
Camera Man	0.888	0.812	0.896
Couple	0.904	0.905	0.898
Finger	0.957	0.947	0.969
Hill	0.886	0.876	0.889
House	0.936	0.903	0.920
Lena	0.911	0.933	0.921
Man	0.894	0.894	0.902
Montage	0.924	0.886	0.933
Peppers	0.922	0.941	0.915
Average	0.903	0.878	0.908

**Higher is better with a max of 1.

Image	CNTK	Neural Network + Sigmoid Last Layer + BatchNorm	Biharmonic
Barbara	673	407	446
Boat	1463	7827	444
Camera Man	100	2270	37
Couple	1463	7845	446
Finger	1371	1963	443
Hill	1463	7826	485
House	140	2260	38
Lena	1467	7828	442
Man	1468	7833	441
Montage	157	2253	37
Peppers	124	1273	36
Average	899	4511	300

Figure B-6: A comparison of PSNR, SSIM, and runtime for image reconstruction using our framework (CNTK), corresponding finite width neural networks, and biharmonic inpainting. We observe that the CNTK performs (in PSNR and SSIM) on average comparably to biharmonic inpainting and outperforms corresponding finite neural networks with sigmoid last layer and batch normalization layers. While our method is slower than biharmonic inpainting, it is more flexible than this method (see Fig. S4), and it is in average much faster than training finite width neural networks for this application.

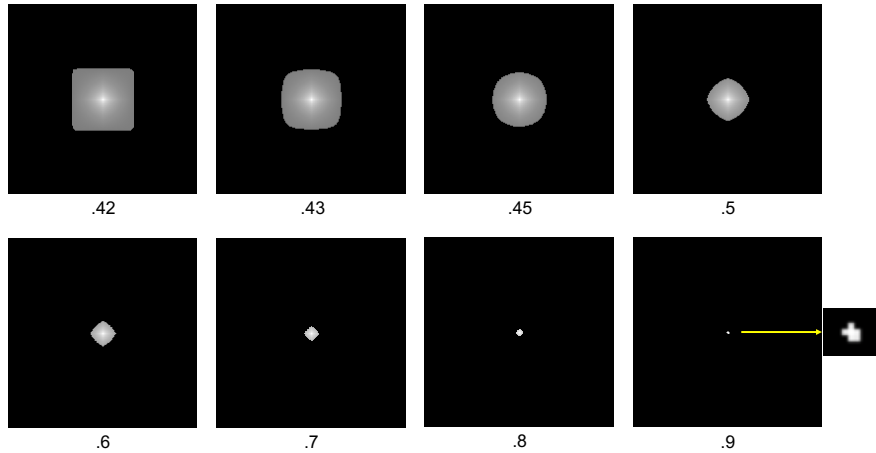


Figure B-7: Visualizing the CNTK of a neural network with nearest neighbor downsampling and upsampling layers and a uniform random feature prior illustrates that this kernel is akin to a kernel that uses different norms for image completion. In the above figure, we visualize the CNTK heatmap for coordinate (64, 64) of the CNTK for a neural network with 5 nearest neighbor downsampling and upsampling layers operating on 128×128 images. In each subfigure, we zero out the x percentile (provided below each image) of pixel values. For example, the image on the bottom right corresponds to zeroing out all pixels with values below the 90th percentile. We observe that balls of varying norms appear in this visualization: e.g., the ℓ_∞ ball appears in the upper left and an ℓ_p ball with $1 < p < 2$ on the upper right.

Appendix C

Chapter 4 Supplementary

A Preliminaries on NTK and Dual Activations

In this section, we briefly review the connection between neural networks and the NTK and properties of dual activations that we will use to prove our main results.

In this work, we consider the setting where the data dimensionality d is fixed. Given an elementwise function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ and a constant C satisfying $C^2 \mathbb{E}_{z \sim \mathcal{N}(0,1)}[\phi(z)^2] = 1$, an L -hidden layer neural network is given by the function $f^{(L)}$, which is defined iteratively as follows:

$$f^{(L)}(x) = W_{L+1} \frac{C}{\sqrt{k_L}} \phi(f^{(L-1)}(x)) ; f^{(1)}(x) = W_2 \frac{C}{\sqrt{k_1}} \phi(W_1 x) ,$$

where $W_i \in \mathbb{R}^{k_i \times k_{i-1}}$ with $k_{L+1} = 1$ and $k_0 = d$. In the deep learning literature, ϕ is called an *activation function*, L is the *depth* of the network, and k_i is the *width* of layer i for $i \in \{1, 2, \dots, L\}$.

To analyze infinitely wide networks, we assume the entries of each W_i are independently, identically distributed from $\mathcal{N}(0, 1)$. The infinitely wide network of depth L is given by considering the limit as $k_1, k_2, \dots, k_L \rightarrow \infty$ in order. In this limit, training all layers of the network $f^{(L)}$ using gradient descent with sufficiently small learning rate leads to a predictor that is equivalent to training a kernel machine using the NTK [96, 118]. The key idea behind this equivalence is that as layer-wise widths approach infinity, a network is well-approximated by a first order Taylor series expansion around its weights [118]. In particular, the Hessian of a neural network with respect to its weights is uniformly small in a ball of fixed radius [118, Theorem 3.2], and gradient descent linearly converges to a solution within this ball [119, Theorem 4]. A summary of key concepts regarding this transition to linearity can be found in [21].

In this work, we analyze infinitely wide and deep networks by considering the limit of infinitely wide networks as $L \rightarrow \infty$. Importantly, we consider the limit as width approaches infinity first before depth in order to eliminate deviations from Gaussianity of order L/k (depth over width) arising in finite width networks [160]. Lastly, to analyze consistency, we consider the behavior of infinitely wide and deep networks as the number of training samples n goes to infinity. All our consistency results rely on knowledge of the ambient dimension of the data. In particular, we focus on data lying on the hypersphere in d dimensions, i.e., a manifold of dimension $d - 1$, in which setting the consistent singular kernel classifier with a kernel of the form $K(x, \tilde{x}) = \frac{1}{\|x - \tilde{x}\|^\alpha}$ has $\alpha = d - 1$.

In order to analyze the behavior of the iterated dual activation, we reference the following result of [53], which implies that the dual activation is analytic around 0 on the interval $[-1, 1]$.

Analyticity of Dual Activations. Let $\phi(\cdot)$ be an activation function such that

$$\mathbb{E}_{x \sim \mathcal{N}(0,1)}[\phi(x)^2] = 1.$$

Let $\check{\phi} : [-1, 1] \rightarrow \mathbb{R}$ denote the dual activation. Then, for $z \in [-1, 1]$,

$$\check{\phi}(z) = \sum_{i=0}^{\infty} a_i z^i, \quad (\text{C.1})$$

where $a_i \geq 0$ for all $i \in \mathbb{N}$.

As proven in [53, Lemma 11], several key properties are implied by (C.1). Those utilized in this work are: (1) $\check{\phi}$ is increasing on $[0, 1]$, and (2) non-negativity of $\check{\phi}(\cdot)$ on $[0, 1]$. (C.1) also implies the following property of dual activations that we will use to construct our taxonomy of infinitely wide and deep neural network classifiers.

Lemma 2. *Let $\check{\phi} : [-1, 1] \rightarrow \mathbb{R}$ be a dual activation such that $\check{\phi}(0) = 0$, $\check{\phi}(1) = 1$, and $\check{\phi}(z) \neq z$. Then, $0 \leq \check{\phi}'(0) < 1$.*

Proof. By (C.1), we need only show that $0 \leq a_1 < 1$. Since $\check{\phi}(1) = 1$, we obtain that $\sum_{i=1}^{\infty} a_i = 1$. Since $a_i \geq 0$ for all $i \in \mathbb{N}$, we conclude that $0 \leq a_1 \leq 1$. Now if $a_1 = 1$, then $a_i = 0$ for $i \geq 2$, which implies that $\check{\phi}(z) = z$. Hence, we conclude that $0 \leq a_1 < 1$, which completes the proof. \square

B Proofs of Theorem 1 and Theorem 2

We first prove Theorem 1, which is expressed below in terms of the dual activation function.

Theorem. *Let $K^{(L)}$ denote the NTK of a fully connected neural network with L hidden layers and activation function $\phi(\cdot)$. For $x, \tilde{x} \in \mathcal{S}_+^d$, let $z = x^T \tilde{x}$. If the dual activation function $\check{\phi}(\cdot)$ satisfies*

- 1) $\check{\phi}(0) = 0$, $\check{\phi}(1) = 1$,
- 2) $0 < \check{\phi}'(0) < 1$ and $\check{\phi}'(1) < \infty$,

then:

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(x, \tilde{x})}{\check{\phi}'(0)^L (L+1)} = \frac{R(x^T \tilde{x})}{\|x - \tilde{x}\|^\alpha},$$

where $\alpha = -2 \frac{\log(\check{\phi}'(0))}{\log(\check{\phi}'(1))}$ and $R(u) \geq 0$ is bounded for $u \in [0, 1]$ and bounded away from 0 around $u = 1$.

In order to prove this theorem, we first prove that the iterated, normalized NTK converges to a singular kernel without explicitly identifying the order of the singularity.

Lemma 3. *Let $K^{(L)}$ denote the NTK of a depth L fully connected network with normalized activation function ϕ . Assuming $\check{\phi}$ satisfies the conditions of Theorem 1, then for any $x, \tilde{x} \in \mathcal{S}_+^d$ it holds that*

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(x, \tilde{x})}{a_1^L (L+1)} = \psi(x^T \tilde{x}),$$

where $\psi : [0, 1] \rightarrow \mathbb{R}$ can be written as a power series with non-negative coefficients with a singularity at 1.

Proof. We utilize the form of the NTK given in [9] and utilize the radial form of the kernel in Eq. (5) of the main text. Namely, for $z \in [0, 1]$, we have:

$$K^{(L)}(z) = \sum_{i=0}^L \check{\phi}^{(i)}(z) \prod_{j=i}^{L-1} \check{\phi}'(\check{\phi}^{(j)}(z)), \quad (\text{C.2})$$

where $\check{\phi}^{(i)}$ denotes the iteration of $\check{\phi}$ i times. By (C.1) and since $\check{\phi}(0) = 0$, we have that $\check{\phi}(z) = \sum_{i=1}^{\infty} a_i z^i$ for all $z \in [0, 1]$.¹ Now, we bound $\check{\phi}$ by quadratic functions in z and bound $\check{\phi}'$ by linear functions in z . In

¹Note that the sum starts from a_1 since $\check{\phi}(0) = 0 \implies a_0 = 0$.

particular, using the conditions $\check{\phi}(1) = 1$ and $\check{\phi}'(1) = C < \infty$, we obtain the upper bounds:

$$\begin{aligned}\check{\phi}(z) &= a_1 \left(z + \sum_{i=2}^{\infty} \frac{a_i}{a_1} z^i \right) \leq a_1 \left(z + \sum_{i=2}^{\infty} \frac{a_i}{a_1} z^2 \right) = a_1 \left(z + \left(\frac{1}{a_1} - 1 \right) z^2 \right), \\ \check{\phi}'(z) &= a_1 \left(1 + \sum_{i=2}^{\infty} i \frac{a_i}{a_1} z^{i-1} \right) \leq a_1 \left(1 + \sum_{i=2}^{\infty} \frac{a_i}{a_1} i z \right) = a_1 \left(1 + \left(\frac{C}{a_1} - 1 \right) z \right).\end{aligned}$$

Similarly, we obtain the lower bounds:

$$\begin{aligned}\check{\phi}(z) &= a_1 \left(z + \sum_{i=2}^{\infty} \frac{a_i}{a_1} z^i \right) \geq a_1 \left(z + \frac{a_2}{a_1} z^2 \right), \\ \check{\phi}'(z) &= a_1 \left(1 + \sum_{i=2}^{\infty} i \frac{a_i}{a_1} z^{i-1} \right) \geq a_1 \left(1 + \frac{2a_2}{a_1} z \right) \geq a_1 \left(1 + \frac{a_2}{a_1} z \right).\end{aligned}$$

Now, substituting the above lower and upper bounds into the recursion for $\check{\phi}^{(i)}$, we obtain

$$a_1^i z \prod_{j=0}^{i-1} \left(1 + \frac{a_2}{a_1} \check{\phi}^{(j)}(z) \right) \leq \check{\phi}^{(i)}(z) \leq a_1^i z \prod_{j=0}^{i-1} \left(1 + \left(\frac{1}{a_1} - 1 \right) \check{\phi}^{(j)}(z) \right). \quad (\text{C.3})$$

Lastly, since $C \geq 1$, substituting (C.3) and the bounds on $\check{\phi}'$ into (C.2) for $K^{(L)}$, we obtain

$$(L+1)a_1^L z \prod_{j=0}^{L-1} \left(1 + \frac{a_2}{a_1} \check{\phi}^{(j)}(z) \right) \leq K^{(L)}(z) \leq (L+1)a_1^L z \prod_{j=0}^{L-1} \left(1 + \left(\frac{C}{a_1} - 1 \right) \check{\phi}^{(j)}(z) \right).$$

Hence, to prove that $\psi(z) := \lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)}$ is finite for $z \in [0, 1)$, we need to show that

$$\prod_{j=0}^{\infty} \left(1 + \tilde{C} \check{\phi}^{(j)}(z) \right) < \infty$$

for all $z \in [0, 1)$ and any constant \tilde{C} . By the Cauchy criterion [176, Ch.5], the above infinite product converges if and only if the following sum converges:

$$\sum_{j=0}^{\infty} \tilde{C} \check{\phi}^{(j)}(z) < \infty.$$

This sum converges by the ratio test. In particular,

$$\lim_{j \rightarrow \infty} \frac{\check{\phi}^{(j)}(z)}{\check{\phi}^{(j-1)}(z)} = \lim_{z \rightarrow 0} \frac{\check{\phi}(z)}{z} = a_1 < 1,$$

where we used the contractive mapping theorem [177] to establish the first equality, since 0 is a fixed point attractor of $\check{\phi}$. As a consequence, $\psi(z) < \infty$ for $z \in [0, 1)$. Now according to (C.2), $\psi(z)$ can be written as a convergent power series with non-negative coefficients for $z \in [0, 1)$. To establish the singularity of $\psi(z)$ at $z = 1$, we show that for any constant $R > 0$, there exists z_0 such that $\psi(z) > R$ for $z > z_0$. In particular, note that for any fixed L_0 ,

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)} = \psi(z) \geq z \prod_{j=0}^{L_0-1} \left(1 + \frac{a_2}{a_1} \check{\phi}^{(j)}(z) \right).$$

The right-hand side is a continuous function with maximum value $\left(1 + \frac{a_2}{a_1}\right)^L$. Hence, by selecting L_0 such that $\left(1 + \frac{a_2}{a_1}\right)^{L_0} > R$, we can then pick z_0 such that $\psi(z) > R$ for all $z > z_0$. Hence, we conclude that

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)} = \psi(z),$$

where $\psi(z)$ can be written as a convergent power series with non-negative coefficients on $[0, 1)$ with a singularity at $z = 1$, which completes the proof. \square

We will now prove Theorem 1 by establishing the order of the singularity of ψ from Lemma 3. To characterize the order of this singularity, we will generally characterize the order of the singularity arising from iterating functions on the interval $[0, 1]$. In particular, we begin by establishing the order of the singularity of the normalized iteration of a function that is linear around $x = 1$.

Lemma 4. *Let*

$$f(z) = \begin{cases} g(z) & \text{if } z \in [0, d] \\ 1 - b(1 - z) & \text{if } z \in (d, 1] \end{cases},$$

with $d < 1$ such that $f(z)$ is strictly monotonically increasing and $g(z)$ can be written as a convergent power series with non-negative coefficients with $g(0) = 0$, $g'(0) = a < 1$, and $b > 1$. Then for $z \in (d, 1]$, it holds that

$$\lim_{L \rightarrow \infty} \frac{f^{(L)}(z)}{a^L} = \frac{R(z)}{(1 - z)^{-\log_b a}},$$

where $R(z)$ is non-negative for $z \in [0, 1]$, bounded from above, and bounded away from 0 around $z = 1$.

Proof. We first visualize the curve $f(z)$ in Fig. C-1a. For any $z \in (d, 1]$, let $L_0(z)$ denote the smallest number of iterations until $f^{(L)}(z) = z' \leq d$. Then for $z \in (d, 1]$, we have that

$$\lim_{L \rightarrow \infty} \frac{f^{(L)}(z)}{a^L} = \lim_{L \rightarrow \infty} \frac{f^{(L-L_0(z))}(z')}{a^{L-L_0(z)}} a^{-L_0(z)}.$$

Now by the proof of Lemma 3, we know that

$$\lim_{L \rightarrow \infty} \frac{f^{(L-L_0(z))}(z')}{a^{L-L_0(z)}} = \tilde{R}(z'),$$

with $\tilde{R}(z') \geq z'$. Thus, we need only analyze the term $a^{-L_0(z)}$ to determine the pole order. In particular, we have that $L_0(z)$ is the least integer that satisfies:

$$1 - b^{L_0(z)}(1 - z) \leq d.$$

Hence, $L_0(z)$ is given by:

$$\begin{aligned} L_0(z) &= \left\lceil \log_b \left(\frac{1-d}{1-z} \right) \right\rceil \\ &= \left\lceil \log_a \left(\frac{1-d}{1-z} \right)^{\frac{1}{\log_a b}} \right\rceil \\ &= \left\lceil \log_a \left(\frac{1-z}{1-d} \right)^{-\log_b a} \right\rceil \\ &\in \left[\log_a \left(\frac{1-z}{1-d} \right)^{-\log_b a}, \log_a \left(\frac{1-z}{1-d} \right)^{-\log_b a} + 1 \right]. \end{aligned}$$

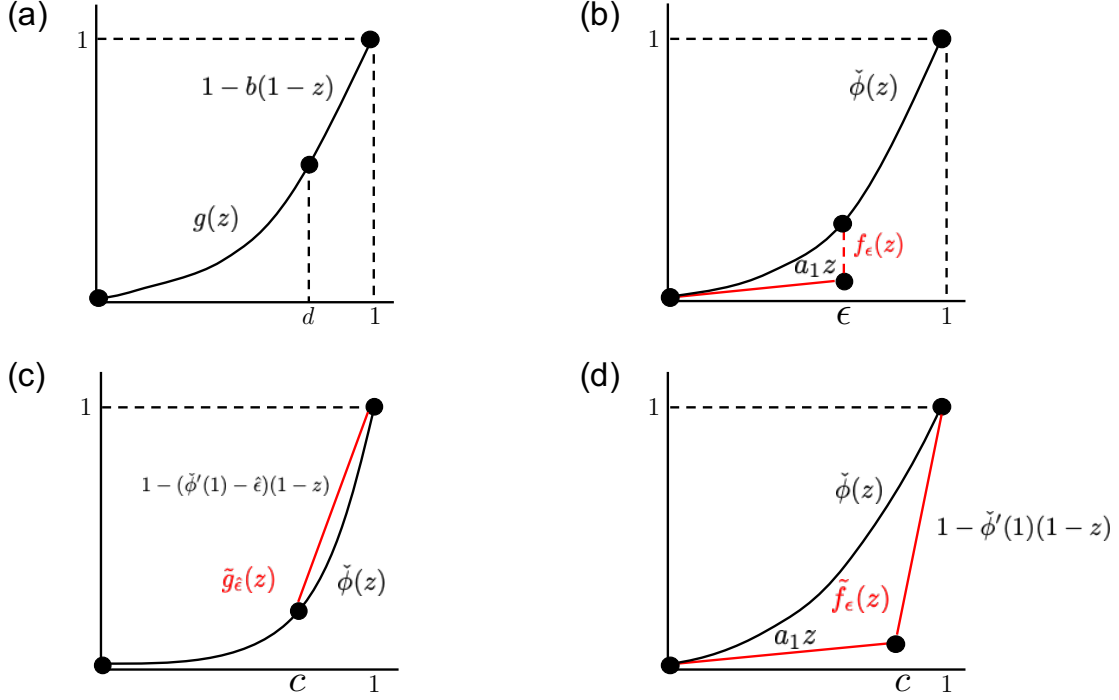


Figure C-1: A visualization of the four functions bounding $\check{\phi}(z)$ that are used to prove Theorem 1.

As a consequence,

$$a^{-L_0(z)} \in \left[\left(\frac{1-d}{1-z} \right)^{-\log_b a}, \frac{1}{a} \left(\frac{1-d}{1-z} \right)^{-\log_b a} \right].$$

Thus we conclude that for $z \in (d, 1)$:

$$\lim_{L \rightarrow \infty} \frac{f^{(L)}(z)}{a^L} = \frac{R(z)}{(1-z)^{-\log_b a}},$$

where $R(z)$ is non-negative for $z \in [0, 1]$, bounded from above, and bounded away from 0 around $z = 1$, which concludes the proof. \square

We will now utilize Lemma 4 to prove Theorem 1.

Proof. Let $\check{\phi}(z) = \sum_{i=1}^{\infty} a_i z^i$. We will lower bound the dual activation $\check{\phi}$ and its derivative by the piecewise functions:

$$f_{\epsilon}(z) = \begin{cases} a_1 z & \text{if } z \in [0, \epsilon) \\ \check{\phi}(z) & \text{if } z \in [\epsilon, 1] \end{cases} \quad \text{and} \quad h_{\epsilon}(z) = \begin{cases} a_1 & \text{if } z \in [0, \epsilon) \\ \check{\phi}'(z) & \text{if } z \in [\epsilon, 1] \end{cases}.$$

The function $f_{\epsilon}(z)$ is visualized in Fig. C-1b. Now consider the function $k_{\epsilon}^{(L)}(z)$ defined as follows:

$$k_{\epsilon}^{(L)}(z) = k_{\epsilon}^{(L-1)}(z) h_{\epsilon}(f_{\epsilon}^{(L-1)}(z)) + f_{\epsilon}^{(L)}(z).$$

By definition, we have that $K^{(L)}(z) \geq k_{\epsilon}^{(L)}(z)$ for all $z \in [0, 1]$. We will now show that for any $\tilde{\epsilon}$, we can

select k_ϵ such that

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(z) - k_\epsilon^{(L)}(z)}{a_1^L(L+1)} < \tilde{\epsilon}. \quad (\text{C.4})$$

To prove (C.4), we first consider the updates for $L > L_0$ where L_0 is the largest integer such that $k^{(L_0)}(z) = K^{(L_0)}(z)$, $h_\epsilon(f_\epsilon^{(L_0+1)}(z)) = a_1$, and $f_\epsilon^{(L_0)}(z) = \check{\phi}^{(L_0)}(z)$. We will first prove inductively that for $T \in \mathbb{N}$:

$$\check{\phi}^{(L_0+T)}(z) - f_\epsilon^{(L_0+T)}(z) \leq C_1(z) \left(\sum_{i=0}^{T-1} a_1^{2L_0+T-1+i} \right), \quad (\text{C.5})$$

where $C_1(z)$ is a term independent of T . We begin with the base case of $T = 1$. Namely, we have for $z \in (\epsilon, 1)$:

$$\begin{aligned} \check{\phi}^{(L_0+1)}(z) - f_\epsilon^{(L_0+1)}(z) &\leq \sum_{i=1}^{\infty} a_i \left(\check{\phi}^{(L_0)}(z) \right)^i - a_1 f_\epsilon^{(L_0)}(z) \\ &= \sum_{i=2}^{\infty} a_i \left(\check{\phi}^{(L_0)}(z) \right)^i \quad \left(\text{since } f_\epsilon^{(L_0)}(z) = \check{\phi}^{(L_0)}(z) \right) \\ &\leq a_1^{2L_0} \tilde{C}_1(z) (1 - a_1) \quad \left(\text{where } \tilde{C}_1(z) = \left(\lim_{L \rightarrow \infty} \frac{\check{\phi}^{(L)}(z)}{a_1^L} \right)^2 \right) \\ &= C_1(z) a_1^{2L_0}, \end{aligned}$$

which concludes the base case. Now assume the statement is true for $T = T_0$. Then for $T = T_0 + 1$, we have:

$$\begin{aligned} \check{\phi}^{(L_0+T_0+1)}(z) - f_\epsilon^{(L_0+T_0+1)}(z) &= \sum_{i=1}^{\infty} a_i \left(\check{\phi}^{(L_0+T_0)}(z) \right)^i - a_1 f_\epsilon^{(L_0+T_0)}(z) \\ &\leq C_1(z) \left(\sum_{i=0}^{T_0-1} a_1^{2L_0+T_0+i} \right) + \sum_{i=2}^{\infty} a_i \left(\check{\phi}^{(L_0+T_0)}(z) \right)^i \\ &\leq C_1(z) \left(\sum_{i=0}^{T_0-1} a_1^{2L_0+T_0+i} \right) + C_1(z) a_1^{2L_0+2T_0} \\ &= C_1(z) \left(\sum_{i=0}^{T_0} a_1^{2L_0+T_0+i} \right), \end{aligned}$$

which concludes the proof by induction. We will next prove inductively that for $T \in \mathbb{N}$:

$$\begin{aligned} K^{(L_0+T)}(z) - k_\epsilon^{(L_0+T)}(z) &\leq C_1(z) \left(\sum_{i=0}^{T-1} (T-i) a_1^{2L_0+T-1+i} \right) \\ &\quad + C_2(z) \left(\sum_{i=0}^{T-2} (L_0+i+2) a_1^{2L_0+T+i} \right), \end{aligned} \quad (\text{C.6})$$

where $C_1(z), C_2(z)$ are terms independent of T . We begin with the base case of $T = 1$. Namely, we have for $z \in (\epsilon, 1)$:

$$\begin{aligned} K^{(L_0+1)}(z) - k_\epsilon^{(L_0+1)}(z) &\leq [K^{(L_0)}(z) \check{\phi}'(\check{\phi}^{(L_0)}(z)) - k_\epsilon^{(L_0)}(z) h_\epsilon(f_\epsilon^{(L_0)}(z))] + [\check{\phi}^{(L_0+1)}(z) - f_\epsilon^{(L_0+1)}(z)] \\ &= \check{\phi}^{(L_0+1)}(z) - f_\epsilon^{(L_0+1)}(z) \\ &\leq C_1(z) a_1^{2L_0} \quad (\text{by (C.5)}), \end{aligned}$$

which concludes the base case. Now, assume that (C.6) holds for $T = T_0$. Then for $T = T_0 + 1$, we have:

$$\begin{aligned} K^{(L_0+T_0+1)}(z) - k_\epsilon^{(L_0+T_0+1)}(z) &\leq [K^{(L_0+T_0)}(z)\check{\phi}'(\check{\phi}^{(L_0+T_0)}(z)) - k_\epsilon^{(L_0+T_0)}(z)h_\epsilon(f_\epsilon^{(L_0+T_0)}(z))] \\ &\quad + [\check{\phi}^{(L_0+T_0+1)}(z) - f_\epsilon^{(L_0+T_0+1)}(z)] \\ &= \left[K^{(L_0+T_0)}(z) \sum_{i=1}^{\infty} ia_i(\check{\phi}^{(L_0+T_0)}(z))^{i-1} - a_1 k_\epsilon^{(L_0+T_0)}(z) \right] \\ &\quad + [\check{\phi}^{(L_0+T_0+1)}(z) - f_\epsilon^{(L_0+T_0+1)}(z)]. \end{aligned}$$

Next we simplify each term in brackets via the inductive hypothesis. Let

$$S_1 = \left[K^{(L_0+T_0)}(z) \sum_{i=1}^{\infty} ia_i(\check{\phi}^{(L_0+T_0)}(z))^{i-1} - a_1 k_\epsilon^{(L_0+T_0)}(z) \right].$$

Then, given $\check{\phi}'(1) = C < \infty$, for

$$C_2(z) = (C - a_1) \lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)} \lim_{L \rightarrow \infty} \frac{\check{\phi}^{(L)}(z)}{a_1^L},$$

which is finite by Lemma 3, we have:

$$\begin{aligned} S_1 &\leq C_1(z) \left(\sum_{i=0}^{T_0-1} (T_0 - i) a_1^{2L_0+T_0+i} \right) + C_2(z) \left(\sum_{i=0}^{T_0-2} (L_0 + i + 2) a_1^{2L_0+T_0+1+i} \right) \\ &\quad + K^{(L_0+T_0)}(z) \sum_{i=2}^{\infty} ia_i(\check{\phi}^{(L_0+T_0)}(z))^{i-1} \\ &\leq C_1(z) \left(\sum_{i=0}^{T_0-1} (T_0 - i) a_1^{2L_0+T_0+i} \right) + C_2(z) \left(\sum_{i=0}^{T_0-2} (L_0 + i + 2) a_1^{2L_0+T_0+1+i} \right) \\ &\quad + C_2(z) a_1^{L_0+T_0} (L_0 + T_0 + 1) a_1^{L_0+T_0} \\ &\leq C_1(z) \left(\sum_{i=0}^{T_0-1} (T_0 - i) a_1^{2L_0+T_0+i} \right) + C_2(z) \left(\sum_{i=0}^{T_0-1} (L_0 + i + 2) a_1^{2L_0+T_0+1+i} \right). \end{aligned}$$

Next, let:

$$S_2 = [\check{\phi}^{(L_0+T_0+1)}(z) - f_\epsilon^{(L_0+T_0+1)}(z)].$$

Then, we have by (C.5) that

$$S_2 \leq C_1(z) \left(\sum_{i=0}^{T_0} a_1^{2L_0+T_0+i} \right).$$

Therefore, combining the bounds on S_1, S_2 , we conclude that

$$\begin{aligned}
K^{(L_0+T_0+1)}(z) - k_\epsilon^{(L_0+T_0+1)}(z) &\leq S_1 + S_2 \\
&\leq C_1(z) \left(\sum_{i=0}^{T_0} a_1^{2L_0+T_0+i} \right) + C_1(z) \left(\sum_{i=0}^{T_0-1} (T_0 - i) a_1^{2L_0+T_0+i} \right) \\
&\quad + C_2(z) \left(\sum_{i=0}^{T_0-1} (L_0 + i + 2) a_1^{2L_0+T_0+1+i} \right) \\
&= C_1(z) \left(\sum_{i=0}^{T_0} (T_0 + 1 - i) a_1^{2L_0+T_0+i} \right) \\
&\quad + C_2(z) \left(\sum_{i=0}^{T_0-1} (L_0 + i + 2) a_1^{2L_0+T_0+1+i} \right),
\end{aligned}$$

which concludes the proof by induction and establishes (C.6). Next, (C.6) implies:

$$\begin{aligned}
\frac{K^{(L_0+T)}(z) - k_\epsilon^{(L_0+T)}(z)}{a_1^{L_0+T}(L_0 + T + 1)} &\leq C_1(z) \left(\sum_{i=0}^{T-1} \frac{T - i}{T + L_0 + 1} a_1^{L_0-1+i} \right) \\
&\quad + C_2(z) \left(\sum_{i=0}^{T-2} \frac{L_0 + i + 2}{T + L_0 + 1} a_1^{L_0+i} \right) \\
&\leq \left(C_1(z) a_1^{L_0-1} + C_2(z) a_1^{L_0} \right) \frac{1}{1 - a_1}.
\end{aligned}$$

Hence, since the right-hand side does not depend on T , we conclude that

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(z) - k_\epsilon^{(L)}(z)}{a_1^L(L + 1)} \leq \left(C_1(z) a_1^{L_0-1} + C_2(z) a_1^{L_0} \right) \frac{1}{1 - a_1}.$$

Lastly, note that by selecting ϵ small enough, we can make $L_0(z)$ arbitrarily large. Hence, for any fixed $z \in [0, 1]$, we conclude that

$$\lim_{\epsilon \rightarrow 0} \lim_{L \rightarrow \infty} \frac{K^{(L)}(z) - k_\epsilon^{(L)}(z)}{a_1^L(L + 1)} = 0,$$

and as a consequence that

$$\begin{aligned}
\lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L + 1)} &= \lim_{\epsilon \rightarrow 0} \lim_{L \rightarrow \infty} \frac{K^{(L)}(z) - k_\epsilon^{(L)}(z)}{a_1^L(L + 1)} + \lim_{\epsilon \rightarrow 0} \lim_{L \rightarrow \infty} \frac{k_\epsilon^{(L)}(z)}{a_1^L(L + 1)} \\
&= \lim_{\epsilon \rightarrow 0} \lim_{L \rightarrow \infty} \frac{k_\epsilon^{(L)}(z)}{a_1^L(L + 1)}. \tag{C.7}
\end{aligned}$$

By uniformly bounding the right-hand side over ϵ , we will establish an upper bound on the pole order for the iterated, normalized NTK. To do this, we first show that the iterated, normalized k_ϵ and f_ϵ are equal for $z \in (\epsilon, 1)$. Let $\alpha(z) = k_\epsilon^{(L_0(z))}(z)$ and $\beta(z) = f_\epsilon^{(L_0(z))}(z)$ for $z \in (\epsilon, 1)$. We prove by induction for $T > 0$ that

$$k_\epsilon^{(L_0(z)+T)}(z) = a_1^T [\alpha(z) + T\beta(z)]. \tag{C.8}$$

The base case for $T = 1$ follows by

$$k_\epsilon^{(L_0(z)+1)}(z) = a_1[\alpha(z)] + a_1\beta(z).$$

Proceeding inductively, we assume that (C.8) holds for time T . Then at time $T + 1$, we have

$$\begin{aligned} k_\epsilon^{(L_0(z)+T+1)}(z) &= a_1 k_\epsilon^{L_0(z)+T} + f_\epsilon^{L_0+T+1}(z) \\ &= a_1^{T+1}[\alpha(z) + T\beta(z)] + a_1^{T+1}\beta(z) \\ &= a_1^{T+1}[\alpha(z) + (T+1)\beta(z)], \end{aligned}$$

which concludes the proof by induction. Thus, we obtain that

$$\begin{aligned} \lim_{L \rightarrow \infty} \frac{k_\epsilon^{(L)}(z)}{a_1^L(L+1)} &= \lim_{L \rightarrow \infty} \frac{k_\epsilon^{(L-L_0(z))}(\alpha(z))}{a_1^{L-L_0(z)}(L-L_0(z)+1)} \left(\frac{L-L_0(z)+1}{L+1} \right) a_1^{-L_0(z)} \\ &= \lim_{T \rightarrow \infty} \frac{a_1^T[\alpha(z) + T\beta(z)]}{a_1^T(T+1)} a_1^{-L_0(z)} \\ &= \beta(z) a_1^{-L_0(z)} \\ &= \lim_{L \rightarrow \infty} \frac{f_\epsilon^{(L)}(z)}{a_1^L}. \end{aligned}$$

Next, we will uniformly bound the iterated, normalized f_ϵ . In particular, since $\check{\phi} \geq f_\epsilon$ and the two functions have the same normalizing constant, we obtain

$$\lim_{L \rightarrow \infty} \frac{f_\epsilon^{(L)}(z)}{a_1^L} \leq \lim_{L \rightarrow \infty} \frac{\check{\phi}^{(L)}(z)}{a_1^L}.$$

Now, we have that for any $\hat{\epsilon}$, $\check{\phi}$ is upper bounded by the function:

$$\tilde{g}_{\hat{\epsilon}}(z) = \begin{cases} \check{\phi}(z) & \text{if } z \in [0, c) \\ 1 - (\check{\phi}'(1) - \hat{\epsilon})(1-z) & \text{if } z \in [c, 1] \end{cases},$$

where $z = c$ is the intersection of the secant line $1 - (\check{\phi}'(1) - \hat{\epsilon})(1-z)$ and $\check{\phi}$. We visualize $\tilde{g}_{\hat{\epsilon}}(z)$ in Fig. C-1c. By Lemma 4, we know that for $z \in (c, 1)$:

$$\lim_{L \rightarrow \infty} \frac{\tilde{g}_{\hat{\epsilon}}^{(L)}(z)}{a_1^L} = \frac{R_{\hat{\epsilon}}(z)}{(1-z)^{-\log_{\check{\phi}'(1)-\hat{\epsilon}} \check{\phi}'(0)}},$$

where $R_{\hat{\epsilon}}(z)$ is non-negative for $z \in [0, 1]$, bounded from above, and bounded away from 0 around $z = 1$. Since $\hat{\epsilon}$ is arbitrary, we conclude that for some ϵ'' , for $z \in (\epsilon'', 1)$:

$$\lim_{L \rightarrow \infty} \frac{f_\epsilon^{(L)}(z)}{a_1^L} \leq \lim_{L \rightarrow \infty} \frac{\check{\phi}^{(L)}(z)}{a_1^L} \leq \frac{R_1(z)}{(1-z)^{-\log_{\check{\phi}'(1)} \check{\phi}'(0)}}, \quad (\text{C.9})$$

where $R_1(z)$ is non-negative for $z \in [0, 1]$, bounded from above, and bounded away from 0 around $z = 1$. By substituting back the above inequalities into (C.7), we conclude that

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)} = \lim_{\epsilon \rightarrow 0} \lim_{L \rightarrow \infty} \frac{K^{(L)}(z) - k_\epsilon^{(L)}(z)}{a_1^L(L+1)} + \lim_{\epsilon \rightarrow 0} \lim_{L \rightarrow \infty} \frac{k_\epsilon^{(L)}(z)}{a_1^L(L+1)} \leq \frac{R_1(z)}{(1-z)^{-\log_{\check{\phi}'(1)} \check{\phi}'(0)}}. \quad (\text{C.10})$$

To conclude the proof, we need to establish a similar lower bound on the above limit. We will construct the lower bound by first establishing the order of the singularity of the iteration of $\check{\phi}$ and then showing that this order is a lower bound on the order of the singularity for the iterated, normalized NTK. Note that we have already established an upper bound on the order of the singularity of the iteration of $\check{\phi}$ in (C.9). Now,

we alternatively lower bound $\check{\phi}$ via the following function:

$$\tilde{f}(z) = \begin{cases} a_1 z & \text{if } x \in [0, c) \\ 1 - \check{\phi}'(1)(1 - z) & \text{if } x \in [c, 1] \end{cases},$$

where $z = c$ corresponds to the intersection of the tangent lines of $\check{\phi}$ at $z = 0$ and $z = 1$. We visualize $\tilde{f}(z)$ in Fig. C-1d. By Lemma 4, we have that for $z \in (c, 1)$:

$$\lim_{L \rightarrow \infty} \frac{\check{\phi}^{(L)}(z)}{a_1^L} \geq \lim_{L \rightarrow \infty} \frac{\tilde{f}^{(L)}(z)}{a_1^L} = \frac{Q(z)}{(1 - z)^{-\log_{\check{\phi}'(1)} \check{\phi}'(0)}},$$

where $Q(z)$ is non-negative for $z \in [0, 1]$, bounded from above, and bounded away from 0 around $z = 1$. Hence, we conclude that:

$$\lim_{L \rightarrow \infty} \frac{\check{\phi}^{(L)}(z)}{a_1^L} = \frac{R_2(z)}{(1 - z)^{-\log_{\check{\phi}'(1)} \check{\phi}'(0)}},$$

where $R_2(z)$ is non-negative for $z \in [0, 1]$, bounded from above, and bounded away from 0 around $z = 1$. Lastly, we utilize (C.2) to show that:

$$\lim_{L \rightarrow \infty} \frac{\check{\phi}^{(L)}(z)}{a_1^L} \leq \lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L (L + 1)}.$$

In particular, (C.2) states that

$$K^{(L)}(z) = \sum_{i=0}^L \check{\phi}^{(i)}(z) \prod_{k=i}^{L-1} \check{\phi}'(\check{\phi}^{(k)}(z)).$$

We next write $\check{\phi}^{(i)}(z)$ as a product and substitute the computed product back into (C.2). Namely, using the power series representation for $\check{\phi}$ and unrolling the iteration, we obtain:

$$\begin{aligned} \check{\phi}^{(i)}(z) &= \sum_{j=1}^{\infty} a_j \left(\check{\phi}^{(i-1)}(z) \right)^j \\ &= a_1 \check{\phi}^{(i-1)}(z) \left(1 + \sum_{j=2}^{\infty} \frac{a_j}{a_1} \left(\check{\phi}^{(i-1)}(z) \right)^j \right) \\ &= a_1^i z \prod_{k=0}^{i-1} \left(1 + \sum_{j=2}^{\infty} \frac{a_j}{a_1} \left(\check{\phi}^{(k)}(z) \right)^j \right). \end{aligned}$$

We similarly use the power series for $\check{\phi}'(z)$ to conclude that

$$\begin{aligned}
\check{\phi}'\left(\check{\phi}^{(k)}(z)\right) &= \sum_{j=1}^{\infty} j a_j \left(\check{\phi}^{(k)}(z)\right)^{j-1} \\
&\geq \sum_{j=1}^{\infty} a_j \left(\check{\phi}^{(k)}(z)\right)^{j-1} \\
&\geq a_1 \left(1 + \sum_{j=2}^{\infty} \frac{a_j}{a_1} \left(\check{\phi}^{(k)}(z)\right)^{j-1}\right) \\
&\geq a_1 \left(1 + \sum_{j=2}^{\infty} \frac{a_j}{a_1} \left(\check{\phi}^{(k)}(z)\right)^j\right) \quad (\text{as } \check{\phi}^{(k)}(z) \leq 1).
\end{aligned}$$

Therefore, we can simplify (C.2) as follows:

$$\begin{aligned}
K^{(L)}(z) &= \sum_{i=0}^L \check{\phi}^{(i)}(z) \prod_{k=i}^{L-1} \check{\phi}'\left(\check{\phi}^{(k)}(z)\right) \\
&\geq \sum_{i=0}^L a_1^i z \prod_{k=0}^{i-1} \left(1 + \sum_{j=2}^{\infty} \frac{a_j}{a_1} \left(\check{\phi}^{(k)}(z)\right)^j\right) \prod_{k'=i}^{L-1} a_1 \left(1 + \sum_{j=2}^{\infty} \frac{a_j}{a_1} \left(\check{\phi}^{(k')}(z)\right)^j\right) \\
&= \sum_{i=0}^L a_1^L z \prod_{k=0}^{L-1} \left(1 + \sum_{j=2}^{\infty} \frac{a_j}{a_1} \left(\check{\phi}^{(k)}(z)\right)^j\right) \\
&= (L+1) \check{\phi}^{(L)}(z),
\end{aligned}$$

and conclude that

$$\frac{K^{(L)}(z)}{a_1^L(L+1)} \geq \frac{\check{\phi}^{(L)}(z)}{a_1^L}.$$

As a consequence,

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)} \geq \lim_{L \rightarrow \infty} \frac{\check{\phi}^{(L)}(z)}{a_1^L} = \frac{R_2(z)}{(1-z)^{-\log_{\check{\phi}'(1)} \check{\phi}'(0)}}. \quad (\text{C.11})$$

Lastly, we combine (C.10) and (C.11) to conclude that there exists some ϵ such that for $z \in (\epsilon, 1)$:

$$\frac{R_2(z)}{(1-z)^{-\log_{\check{\phi}'(1)} \check{\phi}'(0)}} \leq \lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)} \leq \frac{R_1(z)}{(1-z)^{-\log_{\check{\phi}'(1)} \check{\phi}'(0)}}.$$

Thus, we conclude that

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)} = \frac{R(z)}{(1-z)^{-\log_{\check{\phi}'(1)} \check{\phi}'(0)}},$$

where $R(z)$ is non-negative for $z \in [0, 1]$, bounded from above, and bounded away from 0 around $z = 1$. This concludes the proof of Theorem 1. \square

To prove Theorem 2, we will use the result of Theorem 1 and that of [54], which analyzes the consistency of singular kernel smoothers. To connect infinitely wide and deep networks with kernel smoothers, we next prove that the infinite depth limit of the NTK corresponds to a kernel smoother under the conditions of Theorem 1.

Lemma 5. Let $\psi(z) = \lim_{L \rightarrow \infty} \frac{K^{(L)}(z)}{a_1^L(L+1)}$. Then under the setting of Theorem 2,

$$m_n(x) = \text{sign} \left(\sum_{i=1}^n y^{(i)} \psi(x^T x^{(i)}) \right),$$

assuming $\left| \sum_{i=1}^n y^{(i)} \psi(x^T x^{(i)}) \right| > 0$ for almost all $x \in \mathcal{S}_+^{d-1}$.

Proof. Let $m_n^{(L)}(x)$ be defined as follows:

$$m_n^{(L)}(x) = \text{sign} \left(y \left(K_n^{(L)} \right)^{-1} K^{(L)}(X, x) \right).$$

We first note that multiplying the argument to the sign function by a positive constant does not affect the value. Hence, we have:

$$\lim_{L \rightarrow \infty} m_n^{(L)}(x) = \lim_{L \rightarrow \infty} \text{sign} \left(y \left(K_n^{(L)} \right)^{-1} \frac{K^{(L)}(X, x)}{a_1^L(L+1)} \right).$$

Now we compare the argument of the sign function above to the corresponding kernel smoother. Namely, we have:

$$\left| y \left(K_n^{(L)} \right)^{-1} \frac{K^{(L)}(X, x)}{a_1^L(L+1)} - y \frac{K^{(L)}(X, x)}{a_1^L(L+1)} \right| \leq \|y\|_2 \left\| \left(K_n^{(L)} \right)^{-1} - I \right\|_2 \left\| \frac{K^{(L)}(X, x)}{a_1^L(L+1)} \right\|_2,$$

where the inequality follows from the Cauchy-Schwarz inequality and $\|Av\|_2 \leq \|A\|_2 \|v\|_2$ for $A \in \mathbb{R}^{n \times n}$, $v \in \mathbb{R}^n$. Now since 0 is an attractor for $\hat{\phi}$, then for any $h > 0$, there exists L_1 such that for $L > L_1$, the spectrum of $\hat{K}^{(L)}$ is contained in $[1 - hn^2, 1 + hn^2]$ by Weyl's inequalities. Hence, the spectrum of $\left(K_n^{(L)} \right)^{-1}$ is contained in $\left[\frac{1}{1+hn^2}, \frac{1}{1-hn^2} \right]$. Thus, we conclude that

$$\left\| \left(K_n^{(L)} \right)^{-1} - I \right\|_2 \leq \left(\frac{1}{1-hn^2} - 1 \right).$$

Hence by selecting h appropriately small, we conclude that for any ϵ_1 , there exists L_1 such that for $L > L_1$, $\left\| \left(K_n^{(L)} \right)^{-1} - I \right\|_2 < \epsilon_1$. Next, since $\lim_{L \rightarrow \infty} \frac{K^{(L)}(x^{(i)}, x)}{a_1^L(L+1)} = \psi(x^T x^{(i)})$, for any ϵ_2 , we can select L_2 such that for $L > L_2$,

$$\left| y \frac{K^{(L)}(X, x)}{a_1^L(L+1)} - \sum_{i=1}^n y^{(i)} \psi(x^T x^{(i)}) \right| < \epsilon_2.$$

Next under the assumption in the lemma, we may thus select ϵ_1, ϵ_2 small enough such that the argument of $m_n^{(L)}(x)$ is not exactly 0 for $L > \max(L_1, L_2)$. Thus we can interchange the limit and the sign function. As a consequence, for any $x \neq x^{(i)}$ for $i \in \{1, 2, \dots, n\}$ satisfying $\sum_{i=1}^n y^{(i)} \psi(x^T x^{(i)}) \neq 0$, we obtain that

$$\begin{aligned} \lim_{L \rightarrow \infty} m_n^{(L)}(x) &= \lim_{L \rightarrow \infty} \text{sign} \left(y \left(K_n^{(L)} \right)^{-1} \frac{K^{(L)}(X, x)}{a_1^L(L+1)} \right) \\ &= \text{sign} \left(\lim_{L \rightarrow \infty} y \left(K_n^{(L)} \right)^{-1} \frac{K^{(L)}(X, x)}{a_1^L(L+1)} \right) \\ &= \text{sign} \left(\sum_{i=1}^n y^{(i)} \psi(x^T x^{(i)}) \right). \end{aligned}$$

Lastly, if $x = x^{(i)}$ for some $i \in \{1, 2, \dots, n\}$, then since $\psi(z)$ has a singularity at $z = 1$,

$$\lim_{L \rightarrow \infty} m_n^{(L)}(x) = \text{sign} \left(\lim_{z \rightarrow 1} \sum_{i=1}^n \frac{1}{\psi(z)} y^{(i)} \psi(x^T x^{(i)}) \right) = \text{sign}(y^{(i)}),$$

which completes the proof. \square

We lastly utilize Theorem 1, Lemma 5, and the result of [54] to prove Theorem 2 (expressed below in terms of dual activations), which identifies infinitely wide and deep classifiers that achieve consistency.

Theorem. *Let m_n denote the classifier in Eq. (3) of the main text corresponding to training an infinitely wide and deep network with activation function ϕ on n training points. Let m denote the Bayes optimal classifier, i.e. $m(x) = \arg \max_{\tilde{y} \in \{-1, 1\}} \mathbb{P}(y = \tilde{y} | x)$. If the dual activation, $\check{\phi}$ satisfies:*

- 1) $\check{\phi}(0) = 0, \check{\phi}(1) = 1,$
- 2) $0 < \check{\phi}'(0) < 1$ and $\check{\phi}'(1) < \infty,$
- 3) $-\frac{\log(\check{\phi}'(0))}{\log(\check{\phi}'(1))} = \frac{d}{2},$

then m_n satisfies $\lim_{n \rightarrow \infty} \mathbb{P}_X (|m_n(x) - m(x)| > \epsilon) = 0$ for almost all $x \in \mathcal{S}_+^d$ and for any $\epsilon > 0$.

Proof. Thus far, we proved that under the conditions of Theorem 1, the classifier m_n corresponds to taking the sign of a kernel smoother using a singular kernel with singularity of order $-\frac{\log(\check{\phi}'(0))}{\log(\check{\phi}'(1))}$. For data with density in \mathbb{R}^d , kernel smoothers with singular kernels of the form $K_h(x, \tilde{x}) = \frac{1}{\|x - \tilde{x}\|^d}$ (i.e., the Hilbert estimate) converge to the Bayes optimal classifier in probability for almost all samples as $n \rightarrow \infty$ [54]. We note that multiplying $K_h(x, \tilde{x})$ by a non-negative function that is bounded away from 0 around 1 and bounded from above such that the kernel is still monotonically increasing also yields consistency in the same sense (see SI Appendix C). Returning to our setting, for any $x, \tilde{x} \in \mathcal{S}_+^d$, we can re-write the kernel $K_h(x, \tilde{x})$ as

$$K_h(x, \tilde{x}) = \frac{1}{\|x - \tilde{x}\|^d} = \frac{1}{2^{\frac{d}{2}} (1 - x^T \tilde{x})^{\frac{d}{2}}}.$$

The constant $\frac{1}{2^{\frac{d}{2}}}$ again does not affect the sign function. Lastly, assumption 3 selects the order of the singularity such that the limiting kernel from Theorem 1 can be written up to constant factors as a Hilbert estimate, which concludes the proof of Theorem 2. \square

C Extension of Hilbert estimate consistency

We utilize the following extension of the result from [54] to prove Theorem 2. In this section, we follow the notation from [54] in our statements and proofs.

Corollary. *For $x \in \mathcal{S}_+^d$, let $m(x)$ denote the Bayes optimal regressor. For $x, \tilde{x} \in \mathcal{S}_+^d$, let $K(x^T \tilde{x}) = \frac{R(x^T \tilde{x})}{2^{\frac{d}{2}} (1 - x^T \tilde{x})^{\frac{d}{2}}}$, where $R(z) \geq 0$ for $z \in [0, 1]$ is bounded from above, bounded away from 0 around $z = 1$, and $K(\cdot)$ is monotonically increasing in $[0, 1]$. Given a dataset $\{X_i, Y_i\}_{i=1}^n \subset \mathcal{S}_+^d \times \mathbb{R}$, let*

$$m_n(x) = \frac{\sum_{i=1}^n Y_i K(x^T X_i)}{\sum_{i=1}^n K(x^T X_i)}.$$

Let X have any density f on \mathcal{S}_+^d and let Y be bounded. Then, at almost all x with $f(x) > 0$, $m_n(x) \rightarrow m(x)$ in probability as $n \rightarrow \infty$.

Proof. The proof closely follows that of the theorem in [54] with the differences that (1) we map from densities on \mathcal{S}_+^d to densities on \mathbb{R}^d , and (2) we simply verify that the function $R(z)$ does not change the asymptotic analyses of the original proof. We begin by noting that the kernel K involves chordal distances on the sphere, i.e.,

$$K(x^T \tilde{x}) = \frac{R(x^T \tilde{x})}{\|x - \tilde{x}\|^d}.$$

We first define the random variable $W := \|P(x) - P(X)\|^d V_d$, where V_d is the volume of the unit sphere in d dimensions and $P : \mathcal{S}_+^d \rightarrow \mathbb{R}^d$ is the stereographic projection such that \mathcal{S}_+^d maps to a bounded region. We let f_P denote the density of the points $P(x)$ for $x \in \mathcal{S}_+^d$. We note that Euclidean distances after stereographic projection can be related to chordal distances, $\|x - X\|$, via the following formula (up to isometries of the sphere):

$$\|x - X\|^2 = \frac{\|P(x) - P(X)\|^2}{(1 + \|P(x)\|^2)(1 + \|P(X)\|^2)}.$$

Since we select the projection such that $\|P(x)\| < \infty$ for $x \in \mathcal{S}_+^d$, we have that $(1 + \|P(x)\|^2)(1 + \|P(X)\|^2)$ is bounded and nonzero, i.e., it is again a factor that simply scales the kernel function. We thus define

$$Q(x, \tilde{x}) = R(x^T \tilde{x})(1 + \|P(x)\|^2)^{\frac{d}{2}}(1 + \|P(\tilde{x})\|^2)^{\frac{d}{2}},$$

which is bounded away from zero for some $\epsilon > 0$ and x, \tilde{x} such that $x^T \tilde{x} > 1 - \epsilon$. Letting $W_i := V_d \|P(x) - P(X_i)\|^d$, the regressor $m_n(x)$ is given by

$$m_n(x) = \frac{\sum_{i=1}^n Y_i \frac{Q(x, X_i)}{W_i}}{\sum_{i=1}^n \frac{Q(x, X_i)}{W_i}}.$$

Hence, we can utilize the proof strategy of [54] for points $P(x)$ in \mathbb{R}^d . Namely as in [54], we analyze the term:

$$|m_n(x) - m(x)| \leq \left| \frac{\sum_{i=1}^n (Y_i - m(X_i)) \frac{Q(x, X_i)}{W_i}}{\sum_{i=1}^n \frac{Q(x, X_i)}{W_i}} \right| + \frac{\sum_{i=1}^n |m(X_i) - m(X)| \frac{Q(x, X_i)}{W_i}}{\sum_{i=1}^n \frac{Q(x, X_i)}{W_i}} := I + II.$$

To simplify notation, we let $Q_i = Q(x, X_i)$ and we let $\frac{Q_{(i)}}{W_{(i)}}$ denote the i^{th} order statistic ordered such that $W_{(1)} \leq W_{(2)} \dots \leq W_{(n)}$. Now, the proof strategy of [54] is to show that the terms I and II respectively converge to 0 in probability for almost all x as $n \rightarrow \infty$. To prove that I converges in this manner, following the proof of [54], we have that:

$$\mathbb{E}[I^2 | \{X_i\}_{i=1}^n] \leq C_1 \frac{\frac{1}{W_{(1)}}}{\sum_{j=1}^k \frac{Q_{(j)}}{W_{(j)}}} \leq C_2 \frac{\frac{1}{W_{(1)}}}{\sum_{j=1}^k \frac{1}{W_{(j)}}},$$

where k such that $W_{(k)} > V_d \delta^d$ for small δ , and $C_1, C_2 > 0$ are constants since $\{Q_{(j)}\}_{j=1}^k$ are non-negative and bounded away from 0. Hence, the convergence of I follows directly from the proof of [54]. To establish the convergence of II , we follow the proof of [54] and first establish that

$$A_n := \frac{\sum_{i \leq \theta n} \frac{Q_{(i)}}{W_{(i)}}}{\sum_{i=1}^n \frac{Q_{(i)}}{W_{(i)}}} \rightarrow 1$$

in probability as $n \rightarrow \infty$, for all θ fixed in $(0, 1)$. Let χ denote the indicator function, and following the notation of [54], let $U_{(i)}$ denote uniform order statistics. The work of [54] establishes that for any fixed

$\epsilon \in (0, 1)$ there exists δ such that for all $W_{(i)} \leq V_d \delta^d$:

$$(1 - \epsilon) f_P(P(x)) W_{(i)} \leq U_{(i)} \leq (1 + \epsilon) f_P(P(x)) W_{(i)}.$$

Hence, we consider the event $B = [W_{\lfloor \theta n \rfloor} \leq V_d \delta^d]$, and then as in the proof of [54], we obtain

$$A_n \chi_B \geq 1 - \frac{2\epsilon}{1 + \epsilon} - \frac{\frac{n C_3}{W_{\lfloor \theta n \rfloor}}}{f_P(P(x)) \sum_{i \leq \theta n} \frac{Q_{(i)}}{U_{(i)}}} \geq 1 - \frac{2\epsilon}{1 + \epsilon} - C_4 \frac{\frac{n}{W_{\lfloor \theta n \rfloor}}}{f_P(P(x)) \sum_{i \leq \theta n} \frac{1}{U_{(i)}}},$$

where $C_3, C_4 > 0$ are constants since $Q_{(i)}$ is bounded and positive for $i \leq \lfloor \theta n \rfloor$. The convergence of A_n then follows by continuing the proof from [54]. Next, again following the proof of [54], for any $\epsilon > 0$, we also select δ such that:

$$\sup_{r \leq \delta} \frac{\int_{S_{P(x), r}} |m(y) - m(x)| f_P(y) dy}{\int_{S_{P(x), r}} f_P(y) dy} \leq \epsilon,$$

where $S_{P(x), r}$ denotes the closed ball in \mathbb{R}^d of radius r centered at $P(x)$. Then as in [54], select $A = \{y : m(y) - m(x) > \epsilon\}$ and select $\theta \in (0, 1)$ small enough such that $\mathbb{P}(\|P(X_{\lfloor \theta n \rfloor}) - P(x)\| > \delta) \rightarrow 0$ as $n \rightarrow \infty$. Then, we have:

$$\begin{aligned} II &= \frac{\sum_{i=1}^n |m(X_i) - m(x)| \frac{Q_i}{W_i}}{\sum_{i=1}^n \frac{Q_i}{W_i}} \\ &\leq 2 \frac{\sum_{i > \theta n} \frac{Q_i}{W_i}}{\sum_{i=1}^n \frac{Q_i}{W_i}} + 2 \chi_{\|P(X_{\lfloor \theta n \rfloor}) - P(x)\| > \delta} + \epsilon + \frac{\sum_{i: P(X_i) \in S_{P(x), \delta} \cap A} \frac{Q_i}{W_i}}{\sum_{i=1}^n \frac{Q_i}{W_i}} \\ &:= V_1 + V_2 + V_3 + V_4. \end{aligned}$$

Now as in [54], we have that $V_1 \rightarrow 0$ in probability, as we showed $A_n \rightarrow 1$ in probability above. Then, $V_2 \rightarrow 0$ in probability and V_3 can be made as small as possible by the choice of ϵ . Lastly, $V_4 \rightarrow 0$ since, following the proof of [54]:

$$\frac{\sum_{i: P(X_i) \in S_{P(x), \delta} \cap A} \frac{Q_i}{W_i}}{\sum_{i=1}^n \frac{Q_i}{W_i}} \leq 2\epsilon + C_5 \frac{\frac{1}{W_{(1)}}}{\sum_{i=1}^n \frac{Q_{(i)}}{W_{(i)}}},$$

where $C_5 > 0$ is a constant. The above term goes to 0 in probability by the analysis of part *I* and the arbitrary choice of ϵ . This concludes the proof of this extension of the result of [54]. \square

D Proof of Corollary 1

For ease of reading, we repeat Corollary 1 below.

Corollary. *Let m_n denote the classifier in Eq. (3) of the main text corresponding to training an infinitely wide and deep network with activation function*

$$\phi(x) = \begin{cases} \frac{1}{\sqrt{2}} h_7(x) + \frac{1}{\sqrt{2}} x & \text{if } d = 1 \\ \frac{1}{2^{d/4}} \left(\frac{x^3 - 3x}{\sqrt{6}} \right) + \sqrt{1 - \frac{2}{2^{d/2}}} \left(\frac{x^2 - 1}{\sqrt{2}} \right) + \frac{1}{2^{d/4}} x & \text{if } d \geq 2 \end{cases},$$

where $h_7(x)$ is the 7th probabilist's Hermite polynomial.² Then the classifier m_n is Bayes optimal.

Proof. We need only check that $\check{\phi}(z)$ satisfies the conditions of Theorem 2. We first consider the case $d \geq 2$. In particular, since $\frac{x^2 - 1}{\sqrt{2}}$ is the 2nd normalized probabilist's Hermite polynomial and $\frac{x^3 - 3x}{\sqrt{6}}$ is the third

²For $d = 1$, this activation function can be written in closed form as $\frac{x^7 - 21x^5 + 105x^3 + (12\sqrt{35} - 105)x}{12\sqrt{70}}$.

normalized probabilist's Hermite polynomial, we have by [53, Lemma 11] that

$$\check{\phi}(z) = \frac{1}{2^{\frac{d}{2}}} z^3 + \left(1 - \frac{2}{2^{\frac{d}{2}}}\right) z^2 + \frac{1}{2^{\frac{d}{2}}} z.$$

We thus have by direct computation that

$$\check{\phi}'(1) = \frac{3}{2^{\frac{d}{2}}} + \left(2 - \frac{4}{2^{\frac{d}{2}}}\right) + \frac{1}{2^{\frac{d}{2}}} = 2 \quad ; \quad \check{\phi}'(0) = \frac{1}{2^{\frac{d}{2}}},$$

and so, the result follows from Theorem 2 since

$$-\log_{\check{\phi}'(1)} \check{\phi}'(0) = \log_2 2^{\frac{d}{2}} = \frac{d}{2}.$$

Now for the case of $d = 1$, we have again by [53, Lemma 11] that

$$\check{\phi}(z) = \frac{z^7}{2} + \frac{z}{2}.$$

By direct computation,

$$\check{\phi}'(1) = \frac{7}{2} + \frac{1}{2} = 4 \quad \text{and} \quad \check{\phi}'(0) = \frac{1}{2}.$$

Hence, the result follows from Theorem 2 since

$$-\log_{\check{\phi}'(1)} \check{\phi}'(0) = \frac{1}{2} = \frac{d}{2}.$$

□

E Proof of Theorem 3

We repeat Theorem 3 below in terms of dual activations.

Theorem. *Let m_n denote the classifier in Eq. (3) of the main text corresponding to training an infinitely wide and deep network with activation function $\phi(\cdot)$ on n training examples. If the dual activation, $\check{\phi}$, satisfies:*

- 1) $\check{\phi}(0) = 0, \check{\phi}(1) = 1,$
- 2) $\check{\phi}'(0) = 0, \check{\phi}'(1) < \infty,$

then $m_n(x)$ is the 1-NN classifier for $x \in \mathcal{S}_+^d$.

Proof. Let $m_n^{(L)}(x)$ be defined as follows:

$$m_n^{(L)}(x) = \text{sign} \left(y \left(K_n^{(L)} \right)^{-1} K^{(L)}(X, x) \right).$$

By the proof of Lemma 5, we analogously have that the Gram matrix converges to the identity matrix as depth approaches infinity, i.e. $\lim_{L \rightarrow \infty} K_n^{(L)} = I$. For $x, \tilde{x} \in \mathcal{S}_+^d$, let $z = x^T \tilde{x}$ and consider the radial kernel $K^{(L)}(z) = K^{(L)}(x, \tilde{x})$. Let $\check{\phi}(z) = \sum_{i=2}^{\infty} a_i z^i$ for $a_i \geq 0$, as given by (C.1). Without loss of generality, we assume $a_2 > 0$. The proof will follow by using induction to establish:

$$\check{\phi}^{(L)}(z) = z^{2^L} h_L(z) \quad \text{and} \quad K^{(L)}(z) = z^{2^L} g_L(z), \tag{C.12}$$

where h_L, g_L are positive, increasing functions on $(0, 1]$. The base case follows for $L = 0$ since $\check{\phi}^{(0)}(z) = K^{(0)}(z) = z$. Hence, we assume the statement is true for $L = T - 1$ and prove the statement for $L = T$. We

have

$$\check{\phi}^{(T)}(z) = \check{\phi} \left(\check{\phi}^{(T-1)}(z) \right) = \sum_{i=2}^{\infty} a_i \left(\check{\phi}^{(T-1)}(z) \right)^i = \left(\check{\phi}^{(T-1)}(z) \right)^2 \left[\sum_{i=2}^{\infty} a_i \left(\check{\phi}^{(T-1)}(z) \right)^{i-2} \right],$$

and hence using the inductive hypothesis, we can conclude that

$$\check{\phi}^{(T)}(z) = z^{2^T} h_{T-1}(z)^2 \left[\sum_{i=2}^{\infty} a_i \left(\check{\phi}^{(T-1)}(z) \right)^{i-2} \right] = z^{2^T} h_T(z),$$

where h_T is positive and increasing since h_{T-1} and the term in brackets are positive and increasing. We proceed similarly for $K^{(T)}$. Namely, we have:

$$\begin{aligned} K^{(T)}(z) &= K^{(T-1)}(z) \check{\phi}'(\check{\phi}^{(T-1)}(z)) + \check{\phi}^{(T)}(z) \\ &= z^{2^{T-1}} g_{T-1}(z) \left[\sum_{i=2}^{\infty} i a_i \left(\check{\phi}^{(T-1)}(z) \right)^{i-1} \right] + z^{2^T} h_T(z) \\ &= z^{2^{T-1}} \check{\phi}^{(T-1)}(z) g_{T-1}(z) \left[\sum_{i=2}^{\infty} i a_i \left(\check{\phi}^{(T-1)}(z) \right)^{i-2} \right] + z^{2^T} h_T(z) \\ &= z^{2^T} \left(h_{T-1}(z) g_{T-1}(z) \left[\sum_{i=2}^{\infty} i a_i \left(\check{\phi}^{(T-1)}(z) \right)^{i-2} \right] + h_T(z) \right) \\ &= z^{2^T} g_T(z), \end{aligned}$$

where $g_T(z)$ is positive and increasing since h_T, h_{T-1}, g_{T-1} and the term in brackets are positive and increasing, which completes the induction argument.

Now let $z_i = x^T x^{(i)}$ for $i \in \{1, 2, \dots, n\}$. Without loss of generality assume that $z_1 > z_j$ for all $j \neq 1$. To show that $\lim_{L \rightarrow \infty} m_n^{(L)}(x)$ is equivalent to the 1-NN classifier, we need only show that $\lim_{L \rightarrow \infty} m_n^{(L)}(x) = y^{(1)}$. By (C.12) for $j \neq 1$, we have that

$$\begin{aligned} \lim_{L \rightarrow \infty} \frac{K^{(L)}(z_j)}{K^{(L)}(z_1)} &= \lim_{L \rightarrow \infty} \frac{z_j^{2^L} g_L(z_j)}{z_1^{2^L} g_L(z_1)} \\ &\leq \lim_{L \rightarrow \infty} \frac{z_j^{2^L}}{z_1^{2^L}} \quad (\text{since } z_j < z_1 \text{ and } g_L \text{ are positive and increasing}) \\ &= 0. \end{aligned}$$

As a consequence, since $K^{(L)}(z_1) > 0$, we obtain that

$$\lim_{L \rightarrow \infty} m_n^{(L)}(x) = \lim_{L \rightarrow \infty} \text{sign} \left(y \left(K_n^{(L)} \right)^{-1} \frac{K^{(L)}(X, x)}{K^{(L)}(x^{(1)}, x)} \right) = y^{(1)},$$

which establishes that $\lim_{L \rightarrow \infty} m_n^{(L)}(x)$ converges to the 1-NN classifier, thereby completing the proof. \square

F Proof of Proposition 1

We repeat Proposition 1 below for ease of reading.

Proposition. *Let m_n denote the classifier in Eq. (3) of the main text corresponding to training an infinitely wide and deep network with activation function $\phi(\cdot)$ on n training examples. For $x, \tilde{x} \in \mathcal{S}_+^d$ with $x \neq \tilde{x}$, if*

the NTK $K^{(L)}$ satisfies

$$\lim_{L \rightarrow \infty} \frac{K^{(L)}(x, \tilde{x})}{C(L)} > C_1 \quad \text{and} \quad \lim_{L \rightarrow \infty} \frac{K^{(L)}(x, \tilde{x})}{C(L)} \neq \lim_{L \rightarrow \infty} \frac{K^{(L)}(x, x)}{C(L)} \quad (\text{C.13})$$

with $C_1 > 0$ and $0 < C(L) < \infty$ for any L , then m_n implements the majority vote classifier, i.e.,

$$m_n(x) = \text{sign} \left(\sum_{i=1}^n y^{(i)} \right).$$

Proof. Let $C_2 = \lim_{L \rightarrow \infty} \frac{K^{(L)}(x, x)}{C(L)}$. We consider two cases: (1) when $C_2 = \infty$, and (2) when $C_2 < \infty$. When $C_2 = \infty$, we have:

$$\begin{aligned} \lim_{L \rightarrow \infty} m_n^{(L)}(x) &= \lim_{L \rightarrow \infty} \text{sign} \left(y(K_n^{(L)})^{-1} K^{(L)}(X, x) \right) \\ &= \lim_{L \rightarrow \infty} \text{sign} \left(y \left(\frac{K_n^{(L)}}{K^{(L)}(x, x)} \right)^{-1} \frac{K^{(L)}(X, x)}{C(L)} \right) \\ &= \text{sign} \left(\sum_{i=1}^n y^{(i)} C_1 \right) \\ &= \text{sign} \left(\sum_{i=1}^n y^{(i)} \right), \end{aligned}$$

which corresponds to the majority vote classifier. When $C_2 < \infty$, we use the Sherman-Morrison formula to compute the inverse of the Gram matrix $\lim_{L \rightarrow \infty} (K_n^{(L)})^{-1}$. In particular, since the inverse is a continuous map on invertible matrices,

$$\lim_{L \rightarrow \infty} (K_n^{(L)})^{-1} = \frac{1}{(C_2 - C_1)} I - \frac{C_1}{(C_2 - C_1)(C_2 - C_1 + C_1 n)} J,$$

where I is the identity matrix and J is the all-ones matrix. Hence, we have that for $x \neq x^{(i)}$ for $i \in \{1, 2, \dots, n\}$:

$$\begin{aligned} \lim_{L \rightarrow \infty} y(K_n^{(L)})^{-1} \frac{K^{(L)}(X, x)}{C(L)} &= y \left(\frac{1}{(C_2 - C_1)} I - \frac{C_1}{(C_2 - C_1)(C_2 - C_1 + C_1 n)} J \right) C_1 \mathbf{1} \\ &= \frac{C_1}{C_2 - C_1 + C_1 n} \sum_{i=1}^n y^{(i)}, \end{aligned}$$

where $\mathbf{1} \in \mathbb{R}^n$ is the all-ones vector. Assuming that $\sum_{i=1}^n y^{(i)} \neq 0$, we can swap the limit and sign function to conclude that:

$$\begin{aligned} \lim_{L \rightarrow \infty} m_n^{(L)}(x) &= \text{sign} \left(\lim_{L \rightarrow \infty} y(K_n^{(L)})^{-1} K^{(L)}(X, x) \right) \\ &= \text{sign} \left(\frac{C_1}{C_2 - C_1 + C_1 n} \sum_{i=1}^n y^{(i)} \right) \\ &= \text{sign} \left(\sum_{i=1}^n y^{(i)} \right), \end{aligned}$$

which completes the proof. \square

G Proofs for when Infinitely Wide and Deep Networks are Majority Vote Classifiers

The following lemma implies that any activation function satisfying $\check{\phi}(0) > 0$ and $\check{\phi}'(1) > 1$ yields a NTK satisfying (C.13) and thus, the infinite depth classifier is the majority vote classifier by Proposition 1.

Lemma 6. *Let m_n denote the classifier in Eq. (3) of the main text corresponding to training an infinitely wide and deep network with activation function $\phi(\cdot)$ on n training examples. If $\check{\phi}$ satisfies:*

- 1) $\check{\phi}(0) > 0, \check{\phi}(1) = 1,$
- 2) $1 < \check{\phi}'(1) < \infty,$

then m_n is the majority vote classifier.

Proof. We show that the limiting kernel satisfies the properties of Proposition 1 with $C_2 = \infty$. Note that we must have $\check{\phi}'(0) < 1$ by Lemma 2. Now, since $\check{\phi}(0) < 1$ and $\check{\phi}(1) = 1$, by the intermediate value theorem, there exists some $c \in (0, 1)$ such that $\check{\phi}(c) = c$.

We claim that $\check{\phi}'(c) < 1$. Suppose for the sake of contradiction that $\check{\phi}'(c) \geq 1$. Then, since $\check{\phi}(z)$ can be written as a convergent power series with non-negative coefficients, we have that $\check{\phi}(z) \geq z$ for $z \in (c, 1]$. Hence either $\check{\phi}(z) = z$ on some subset of $(c, 1]$ or $\check{\phi}(z) > z$ for $z \in (c, 1]$. In the former case, analytic continuation implies that $\check{\phi}(z) = z$ on $[0, 1]$, and in the latter case, $\check{\phi}(1) > 1$. Thus, in either case we reach a contradiction and thus we can conclude that $\check{\phi}'(c) < 1$. Therefore, it follows that c is the unique fixed point attractor of $\check{\phi}(z)$.

Lastly, since $c \in (0, 1)$, we can conclude that the infinite depth NTK solves the equilibrium equation corresponding to the recursive formula for the NTK in Eq. (5) of the main text. Namely, for any $z \in (0, 1)$ and $K^*(z) := \lim_{L \rightarrow \infty} K^{(L)}(z)$:

$$K^*(z) = K^*(z)\check{\phi}'(c) + c \implies K^*(z) = \frac{c}{1 - \check{\phi}'(c)}.$$

Hence, for any $z \in (0, 1)$, it holds that $\lim_{L \rightarrow \infty} K^{(L)}(z) = \frac{c}{1 - \check{\phi}'(c)}$. Lastly, letting $a = \check{\phi}'(1)$, for $z = 1$, we have that

$$K^{(L)}(1) = \frac{a^L - 1}{a - 1},$$

and so $\lim_{L \rightarrow \infty} K^{(L)}(1) = \infty$. Thus, $\lim_{L \rightarrow \infty} K^{(L)}(x, \tilde{x})$ satisfies the conditions of Proposition 1, which concludes the proof of the lemma. \square

We next show that if $\check{\phi}$ falls under case 3 with $\check{\phi}'(1) < 1$, then under ridge regularization, the corresponding infinitely wide and deep classifier also implements majority vote classification.

Lemma 7. *Let $m_{n,\lambda}^{(L)}$ denote the ridge-regularized kernel machine with regularization term λ and with the NTK of a fully connected network with L hidden layers and activation function ϕ on n training points. If $\check{\phi}$ satisfies:*

- 1) $\check{\phi}(0) > 0, \check{\phi}(1) = 1,$
- 2) $\check{\phi}'(1) < 1,$

then $\lim_{\lambda \rightarrow 0^+} \lim_{L \rightarrow \infty} m_{n,\lambda}^{(L)}(x)$ is the majority vote classifier.

Proof. The proof follows that of Proposition 1. Since $\check{\phi}'(1) < 1$, $z = 1$ is the unique fixed point attractor of $\check{\phi}$. Then as in the proof of Lemma 6, for all $x, \tilde{x} \in \mathcal{S}_+^d$, it holds that

$$\lim_{L \rightarrow \infty} K^{(L)}(x, \tilde{x}) = \frac{1}{1 - \check{\phi}'(1)}.$$

Letting $c = \frac{1}{1-\phi'(1)}$, we obtain that

$$\begin{aligned}
\lim_{L \rightarrow \infty} m_{n,\lambda}^{(L)}(x) &= \text{sign} \left(y (K_n^{(L)} + \lambda I)^{-1} K^{(L)}(X, x) \right) \\
&= \text{sign} \left(y \left[\lim_{L \rightarrow \infty} (K_n^{(L)} + \lambda I)^{-1} \right] \left[\lim_{L \rightarrow \infty} K^{(L)}(X, x) \right] \right) \\
&= \text{sign} \left(y \left[\frac{1}{\lambda} I - \frac{c}{\lambda(\lambda + cn)} J \right] c \mathbf{1} \right) \\
&= \text{sign} \left(\frac{c}{\lambda + cn} \sum_{i=1}^n y^{(i)} \right),
\end{aligned}$$

where $J \in \mathbb{R}^{n \times n}$ is the all-ones matrix, $\mathbf{1} \in \mathbb{R}^n$ is the all-ones vector, and the third equality follows from the Sherman-Morrison formula. Hence, Proposition 1 implies that $\lim_{\lambda \rightarrow 0^+} \lim_{L \rightarrow \infty} m_{n,\lambda}^{(L)}(x)$ is again the majority vote classifier, thereby completing the proof. \square

H Infinitely Wide and Deep Networks with Standard Activation Functions

We now analyze where in our taxonomy lie infinitely wide and deep networks using standard activation functions. In particular, we consider the following commonly used activation functions: ReLU, sigmoid, swish [156], cosid [59], sine, tanh, and scaled hard tanh (Shtanh) [134]. For any activation function, we can follow the branches of our taxonomy by checking whether $A = \mathbb{E}[\phi(z)]$ and $A' = \mathbb{E}[\phi'(z)]$ are nonzero, where $z \sim \mathcal{N}(0, 1)$. If the activation function leads to a singular kernel classifier, we additionally derive its order of singularity, as given by Theorem 1.

We first outline the majority vote classifier cases below.

- ReLU ($\phi(x) = \max(x, 0)$): Since $A > 0$, ReLU leads to a majority vote classifier.
- Sigmoid ($\phi(x) = \frac{1}{1+e^{-x}}$): Since $A > 0$, sigmoid leads to a majority vote classifier.
- Swish ($\phi(x) = \frac{x}{1+e^{-x}}$): The expression for A has no closed form, but is approximated by $A \approx 0.207$ implying that swish leads to a majority vote classifier.
- Cosid ($\phi(x) = \cos(x) - x$): Since $A = \frac{1}{\sqrt{e}} > 0$, cosid leads to a majority vote classifier.

We next analyze the singular kernel classifier cases. We observe that sine, tanh, and hard tanh are all odd functions satisfying $A = 0$ and $A' \neq 0$. Hence, these all implement singular kernel classifiers, according to Theorem 1. For these activation functions, we compute the order of the singularity via the following steps:

- 1) Computing the normalization constant $C = \mathbb{E}[\phi(z)^2]$.
- 2) Computing $A' = \frac{1}{\sqrt{C}} \mathbb{E}[\phi'(z)]$.
- 3) Computing $B' = \frac{1}{C} \mathbb{E}[\phi'(z)^2]$.
- 4) Computing the order $\alpha = -\frac{\log(A'^2)}{\log(B')}$.

We list the order of singularity for each case below.

- Sine ($\phi(x) = \sin(ax)$): The normalization constant is given by $C = \frac{1}{\sinh(a^2)}$ and the order of singularity is given by:

$$\alpha = -\frac{\log \frac{a^2}{\sinh(a^2)}}{\log \frac{a^2 \cosh(a^2)}{\sinh(a^2)}}.$$

When a is chosen such that $\alpha = \frac{d}{2}$, the singular kernel is consistent for data with density on \mathcal{S}_+^d . We note that the function α is monotonically increasing in a and thus has solutions for any $d \geq 2$. This activation function is typically used with $a = 1$, and in this case leads to a singular kernel with order 0.593, which is near the consistent value of 0.5 for data on the unit circle.

- $\tanh(\phi(x) = \tanh(x))$: The normalization constant does not have a closed form, but is approximated by $C = 0.394294$. The corresponding order is given numerically by $\alpha \approx 0.44033$, which is near the consistent value of 0.5 for data on the unit circle.
- Shtanh: The activation function is given by:

$$\phi(x) = \begin{cases} kx & \text{if } |x| < a \\ ak \operatorname{sign}(x) & \text{if } |x| \geq a \end{cases},$$

with $a, k > 0$. The normalization constant is given by $C = k^2 D$, where $D = a^2 + (1 - a^2) \operatorname{erf}\left(\frac{a}{\sqrt{2}}\right) - a\sqrt{\frac{2}{\pi}} e^{-a^2/2}$. The order of singularity is given by:

$$\alpha = -\frac{\log\left(\frac{1}{D} \left(\operatorname{erf}\left(\frac{a}{\sqrt{2}}\right)\right)^2\right)}{\log\left(\frac{1}{D} \operatorname{erf}\left(\frac{a}{\sqrt{2}}\right)\right)}.$$

Interestingly, note that α does not depend on the scale factor k . Moreover, this function numerically appears to have maximum value bounded by 0.37 in the interval $[1, 2]$, implying that it is not consistent for any integral data dimension. For $a = 1$, the order is 0.364, which is close to the near-consistent value of 0.5 for data on the unit circle.

The results for all activation functions above are summarized in Fig. C-2.

I Experiments

Below, we corroborate our theoretical results using infinitely wide and deep networks on data sampled from a variety of distributions. As predicted by our theory, we show that standard activation functions such as ReLU, sine, and erf, lead to infinitely wide and deep networks that are not consistent classifiers for general

Activation	Infinitely Wide and Deep Classifier	Order of Singularity	Consistency
ReLU	Majority Vote	N.A.	Not consistent.
Sigmoid	Majority Vote	N.A.	Not consistent.
Swish	Majority Vote	N.A.	Not consistent.
Cosid	Majority Vote	N.A.	Not consistent.
Sine (a = 1)	Singular Kernel Classifier	0.593	Near consistent on unit circle.
Erf	Singular Kernel Classifier	0.444	Near consistent on unit circle.
Tanh	Singular Kernel Classifier	0.440	Near consistent on unit circle.
Hard Tanh (a = 1)	Singular Kernel Classifier	0.364	Near consistent on unit circle.

Figure C-2: Classifiers implemented by infinitely wide and deep networks with standard activation functions. ReLU and Sigmoid lead to classifiers that implement majority vote behavior since they satisfy the conditions of Proposition 1. On the other hand, sine, tanh, and hard tanh all lead to singular kernel classifiers with order of singularity near 0.5, which is the consistent pole order for data with density on the unit circle. Hence, these activation functions lead to infinitely wide and deep networks that are near-consistent on the unit circle, as is corroborated experimentally in Figure C-3a.

data distributions. On the other hand, given data dimension, custom activation functions satisfying Theorem 2 of our work lead to infinitely wide and deep networks that are consistent classifiers.

Experimental Setup. We consider the performance of classifiers on high dimensional synthetic data distributions from which we can generate an arbitrary number of samples and, importantly, can evaluate the performance of the Bayes optimal classifier. In particular, for labels $y \in \{-1, 1\}$ and data $x \in \mathcal{S}_+^{d-1}$, we select distributions $\mathbb{P}(x|y = -1)$ and $\mathbb{P}(x|y = 1)$. In addition, we vary the label prior, $p = \mathbb{P}(y = 1)$, to generate data with unbalanced label sets.

Distributions. We consider the following variety of low and high dimensional distributions.

- 1) Triangular distributions on the unit circle from [48]. For $\theta \in [0, \frac{\pi}{2}]$ and $x = (\cos \theta, \sin \theta)$:

$$\mathbb{P}(x|y = 1) = \frac{8\theta}{\pi^2}; \quad \mathbb{P}(x|y = -1) = \frac{4}{\pi} - \frac{8\theta}{\pi^2}.$$

- 2) Normalized Dirichlet distributions in arbitrary dimension. Given $\alpha^{(1)}, \alpha^{(2)} \in \mathbb{R}_+^d$, we sample \tilde{x} on the probability simplex in d dimensions (i.e. $x_i \geq 0$ and $\|x\|_1 = 1$) according to:

$$\mathbb{P}(\tilde{x}|y = 1) = \frac{1}{Z(\alpha^{(1)})} \prod_{i=1}^d \tilde{x}_i^{\alpha_i^{(1)}-1}; \quad \mathbb{P}(\tilde{x}|y = -1) = \frac{1}{Z(\alpha^{(2)})} \prod_{i=1}^d \tilde{x}_i^{\alpha_i^{(2)}-1},$$

where $Z(\alpha)$ is the normalizing (or partition) function. We then generate samples $x \in \mathcal{S}_+^{d-1}$ by projecting \tilde{x} onto the unit sphere via $x = \frac{\tilde{x}}{\|\tilde{x}\|_2}$.

Activation Functions. We consider the performance of infinitely wide and deep networks across the following activation functions, $\phi: \mathbb{R} \rightarrow \mathbb{R}$, that are normalized such that $\mathbb{E}_{z \sim \mathcal{N}(0,1)}[\phi(z)^2] = 1$:

- ReLU: $\phi(x) = \sqrt{2} \max(0, x)$;
- Erf: $\phi(x) = \sqrt{\frac{\pi}{2 \arcsin(\frac{2}{3})}} \text{erf}(x)$;
- Sine: $\phi(x) = \frac{1}{\sqrt{\sinh(a^2)}} \sin(ax)$;
- Cubic: $\phi(x) = \frac{1}{2^{d/4}} h_3(x) + \sqrt{1 - \frac{2}{2^{d/2}}} h_2(x) + \frac{1}{2^{d/4}} h_1(x)$, where $h_i(x)$ is the i^{th} Hermite polynomial.

We note that the erf function is similar to the tanh and hard tanh function but has a closed form for the dual activation. In our experiments, for the sine activation, we consider $a^2 = \{2.676, 6.135, 13.826\}$, which give rise to infinitely wide and deep networks that are near-consistent (i.e., they implement singular kernel classifiers with near-consistent pole order) for data with density in \mathcal{S}_+^{d-1} for $d = 3, 5, 9$. We similarly consider cubic activation functions according to Corollary 1 that give rise to infinitely wide and deep networks that are consistent for data with density in \mathcal{S}_+^{d-1} for $d = 2, 3, 5, 9$ (note that for $d = 2$, we use a 7th degree polynomial according to Corollary 1).

Summary of Results. Our results are summarized in Fig. C-3. We compare the performance of infinitely wide and deep classifiers with the following standard classifiers discussed in our taxonomy: (1) majority vote classifier; (2) 1-nearest neighbor (1-NN); and (3) consistent singular kernel classifier given by the Hilbert estimator [54]. Note that for a Hilbert estimator using a kernel of the form $K(x, \tilde{x}) = \frac{1}{\|x - \tilde{x}\|^\alpha}$, we select α to be the dimension of the data manifold (i.e., $d - 1$ for our data in \mathcal{S}_+^{d-1}). Our experiments strongly agree with our theoretical results. In particular, we find that selecting activation functions satisfying the conditions of Theorem 2 lead to infinitely wide and deep networks that are consistent, while standardly used activation functions lead to infinitely wide and deep networks that are not consistent for a variety of distributions. Consistent with these experimental findings, in Fig. C-2 we compute the infinite width and depth classifier corresponding to these standard activation functions, showing that none of them lead to

(a)

Distribution	Label Prior	Standard Activations			Our Activations								Traditional Classifiers			
		ReLU	Sine	Erf	Poly. 2D	Sine 3D	Cubic 3D	Sine 5D	Cubic 5D	Sine 9D	Cubic 9D	Majority Vote	1-NN	Hilbert Estimate	Bayes Optimal	
Triangular	0.2	79.79%	81.83%	83.39%	82.77%	78.53%	78.53%	77.24%	77.21%	77.04%	77.05%	79.95%	76.95%	82.77%	84.02%	
Triangular	0.4	60.70%	72.03%	74.49%	73.67%	69.59%	69.59%	68.51%	68.52%	68.24%	68.27%	59.94%	68.14%	73.71%	76.44%	
Dirichlet 2D	0.2	79.92%	81.2%	82.98%	82.46%	78.63%	78.63%	77.62%	77.64%	77.39%	77.4%	79.92%	77.34%	82.53%	83.64%	
Dirichlet 2D	0.4	61.50%	73.15%	75.52%	74.78%	69.75%	69.75%	68.76%	68.76%	68.51%	68.51%	60.46%	68.31%	74.77%	76.16%	
Dirichlet 3D	0.2	79.92%	94.73%	93.95%	93.95%	95.23%	95.22%	94.91%	94.91%	94.71%	94.71%	79.92%	94.5%	95.07%	96.22%	
Dirichlet 3D	0.4	60.54%	93.86%	93.6%	93.86%	94.34%	94.34%	93.29%	93.29%	92.85%	92.79%	60.46%	92.67%	94.38%	95.09%	
Dirichlet 5D	0.2	79.92%	82.66%	80.42%	80.07%	88.2%	88.17%	91.41%	91.41%	90.04%	90.04%	79.92%	89.35%	91.27%	92.96%	
Dirichlet 5D	0.4	60.50%	86.26%	84.85%	84.54%	88.59%	88.61%	89.31%	89.32%	86.52%	86.65%	60.46%	85.8%	89.31%	90.48%	
Dirichlet 9D	0.2	79.92%	79.92%	79.92%	79.92%	80.12%	80.1%	87.53%	87.64%	91.8%	91.56%	79.92%	89.55%	91.49%	94.25%	
Dirichlet 9D	0.4	60.46%	75.84%	73.06%	70.85%	81.7%	81.64%	87.86%	87.86%	89.53%	89.26%	60.46%	86.91%	89.27%	92.13%	

(b)

Activation:	ReLU	Sine	Erf	Poly. 2D	Sine 3D	Cubic 3D	Sine 5D	Cubic 5D	Sine 9D	Cubic 9D
Depth:	50k	200	100	100	100	100	50	100	50	100

(c)

	$\alpha^{(1)}$	$\alpha^{(2)}$
Dirichlet 2D	[1, 2]	[2, 1]
Dirichlet 3D	[.5, .5, 2.5]	[2.5, .1, .5]
Dirichlet 5D	[3, 1, 2, 1, 1]	[1, 2, 2, 2, 3]
Dirichlet 9D	[1, 2, 1, 2, 1, 1, 1, 1, 1]	[1, 1, 1, 2, 3, 4, 2, 2, 1]

Figure C-3: Comparison of infinitely wide and deep neural network classifiers with varying activation function and traditional classifiers from our taxonomy. All datasets and activation functions are detailed in the Supplementary. (a) We report the test accuracy of infinitely wide and deep classifiers across a variety of distributions and activation functions. The naming convention for our activation functions indicates the dimension for which these activation functions are consistent (e.g. Poly. 2D is consistent for data on the unit circle). Green boxes highlight activation functions that lead to networks that perform within 0.1% or better than the Hilbert estimate, which is known to be consistent [54]. Note that the best performing activation functions for each distribution satisfy the conditions from Theorem 2 in the main text. Additionally, note that in higher dimensions, $d = 5, 9$, our theoretically derived activation functions outperform standard activation functions by over 10%. (b) A list of depths used for our experiments. (c) Parameters of the Dirichlet distributions used in our experiments.

infinitely wide and deep classifiers that are universally consistent. However, we identify several that are near-consistent (order of singularity near 0.5) for data on the unit circle.

Experimental Details. In all experiments using Dirichlet distributions, we select 10000 training samples and 10000 test samples. For all experiments with triangular distributions, we use 5000 training samples and 10000 test samples to avoid numerical issues arising from sampling two unequal points that have a dot product of 1. There is no validation set since we simply interpolate the training data with kernel ridgeless regression. The depths selected for each activation are given in Fig. C-3b. We list all choices of $\alpha^{(1)}, \alpha^{(2)}$ considered in our experiments in Fig. C-3c. When using activation functions such as erf, sine, or cubics, we consider the performance of deep Neural Network Gaussian Processes (NNGPs) instead of the NTK since we found that deep NTKs with these activation functions lead to numerical issues, i.e. rounding kernel values to 0 at large depths. For the NNGPs, we also ensure that the Gram matrix has 1’s along the diagonal regardless of the depth. We found that considering depths much larger than those given in Fig. C-3b led to numerical issues for several activation functions, i.e. kernel values were rounded to 0. Lastly, for ReLU activation, we use 1000 training samples for $d = 5, 9$ and 100 for $d = 2, 3$ since we find that the sublinear rate of convergence to infinite depth behavior requires greater than 10^5 depth (i.e. the kernel value with a test sample is far from $\frac{1}{4}$, which is the infinite depth value established by [90]).

Code and Hardware. Code for experiments is available at https://github.com/uhlerlab/inf_depth_ntks. Experiments were run using 1 Titan Xp containing 12GB of memory.

J Effectiveness of finite-depth NTKs using our derived activation functions on a variety of benchmarking classification tasks

While our optimality results hold for infinitely deep NTKs and as the sample size approaches infinity, we now show that our derived activation functions result in competitive classifiers even in the small-sample setting and when used with finite-depth NTKs. In particular, [61] described a variety of datasets in the small-sample regime (90 classification tasks with at most 5000 samples) that have been widely used for benchmarking. Recent work [10] demonstrated that support vector machines (SVMs) using ReLU NTKs achieved the best performance on these classification tasks, thereby outperforming 179 other methods from [61]. Performance on these tasks was compared using the following metrics:

1. Friedman rank. The average ranking of a classifier across all datasets, where fractional rankings are considered, i.e., if two models are tied in their ranks, then they both receive the mean of their rankings. Lower is better.
2. Average accuracy. The average accuracy of a classifier across all dataset. Higher is better.
3. P90/P95. The average number of datasets in which the classifier achieved within 90%/95% of the accuracy of the best classifier. Higher is better.
4. Percentage Maximum Accuracy (PMA). The average percentage of the best classifier’s accuracy achieved by the given classifier. Higher is better.

In Fig. C-4a, we demonstrate that grid searching over NTKs using the activation functions obtained in this work leads to improved performance over the best models described so far across all metrics. In particular, we grid search over NTKs of depth $\{1, 2, 3, 4, 5\}$ and the following groups of activation functions obtained in this work: (1) sinusoid activations that yield Bayes optimal infinitely deep NTKs in dimensions 1, 2, and 4 and (2) cubic polynomial activations that yield Bayes optimal infinitely deep NTKs in dimensions 1, 2, and 4. Since our derived activation functions are implemented for data on the unit sphere, we normalize all data to the unit sphere accordingly. We then select between our best NTK and the best ReLU NTK based on validation accuracy. For ReLU NTKs, we use the code from [10]. NTKs with our activation functions were selected for 31 out of the 90 tasks based on validation accuracy.

While the above experiments consider the small-sample regime, we next demonstrate the effectiveness of using NTKs with our activation functions also on small-dimensional datasets with many samples. Among the datasets provided in [61], we consider three datasets that have over 10,000 samples with fewer than 15 features.³ These are “chess king-rook vs. king”, “adult”, and “nursery”. Note that while the number of features is small, we do not a priori know the ambient dimension of the data. We thus use the same grid search over activation functions as described above. Interestingly, while for “chess king-rook vs. king” and “nursery”, the cubic polynomial that yields Bayes optimal infinitely deep NTKs in dimension 4 was selected in the grid search, for “adult”, the cubic polynomial that yields Bayes optimal infinitely deep NTKs for dimension 8 was selected. In the experiments, we also compare with finite width networks. For this, we grid search over networks of width 128 and depth between 1 hidden layer and 5 hidden layers, with and without batch normalization [94], and over learning rates 0.01 and 0.1. We use mean squared error for training all networks. The results are shown in Fig. C-4b. In all cases, we find that the activation functions obtained in this work outperform both the standard ReLU NTK and finite width ReLU neural networks.

These extensive benchmarking experiments demonstrate that while our theoretical results provide activation functions that yield provably Bayes optimal infinite depth NTKs as the sample size approaches infinity, our derived activation functions also result in competitive classifiers when used with finite-depth NTKs in the small-sample or small-dimensional regime. Given the simplicity of these models (training an infinite width network in the NTK regime involves solving a linear system of equations), these models should be added to the standard toolkit of simple classifiers like SVM, random forest, or other tree-based models.

³There are only four such datasets in this benchmark. We chose not to include *statlog-shuttle*, which contains 58,000 training samples with 9 features and 7 classes, since all ReLU NTK and neural networks achieve greater than 99.9% test accuracy, thereby making it difficult to make meaningful comparisons.

(a) Performance across 90 classification tasks from Fernández-Delgado et al. *JMLR* 2014

Model	Friedman Rank	Average Accuracy	P90	P95	PMA
Ours	25.74	82.09%	88.89%	75.56%	95.98% ± 5.09%
ReLU NTK [Arora et al. <i>ICLR</i> 2019]	28.34	81.95%	88.89%	72.22%	95.72% ± 5.17%
Random Forest* [Arora et al. <i>ICLR</i> 2019]	33.51	81.56%	85.56%	67.78%	95.25% ± 5.30%
Neural Net [Arora et al. <i>ICLR</i> 2019]	38.06	81.02%	85.56%	60.00%	94.55% ± 5.17%

* Best of 179 models from Fernández-Delgado et al. *JMLR* 2014.(b) Performance comparison on low dimensional classification tasks from Fernández-Delgado et al. *JMLR* 2014

Dataset: Chess King-Rook vs. King Dataset size: 28,056 Data dimension: 6 Number of Classes: 18		Dataset: Adult Dataset size: 48,842 Data dimension: 14 Number of Classes: 2		Dataset: Nursery Dataset size: 12,960 Data dimension: 8 Number of Classes: 5	
Model	Test Accuracy	Model	Test Accuracy	Model	Test Accuracy
Ours	80.37%	Ours	84.92%	Ours	99.62%
ReLU NTK	77.37%	ReLU NTK	84.72%	ReLU NTK	99.55%
Neural Net	52.14%	Neural Net	84.87%	Neural Net	98.86%

Figure C-4: Comparison between finite depth NTKs using the activation functions derived in our work and a broad spectrum of 180 models including fully connected ReLU networks and ReLU NTKs on a variety of classification tasks that are widely used for benchmarking (90 classification tasks from [61]). While our models are provably optimal in the infinite-depth setting as the sample size approaches infinity, these experiments demonstrate that our activation functions also result in competitive results in the small-sample setting as well as the small-dimensional large-sample setting. All metrics and grid search parameters are presented in SI Appendix J. (a) Grid searching over finite depth NTKs using our derived activation functions leads to the best performance across all metrics over 180 models on all 90 small-sample classification tasks (with fewer than 5000 training examples in [61]), thereby outperforming the ReLU NTK, which was previously found to outperform all other methods [10]. (b) Finite depth NTKs using the activation functions derived in our work also outperform ReLU NTK and neural networks on small-dimensional large-sample classification tasks (with fewer than 15 features and greater than 5000 training examples in [61]).

Appendix D

Chapter 5 Supplementary

A Theoretical evidence for Deep Neural Feature Ansatz

We present the proof of Proposition 5 below.

Proof. Gradient descent with learning rate η proceeds as follows:

$$B^{(t+1)} = B^{(t)} + \eta \nabla g(B^{(t)}x)(y - g(B^{(t)}x))x^T .$$

If $B^{(0)} = \mathbf{0}$, then by induction $B^{(t)} = \alpha^{(t)}x^T$ for all time steps t where $\alpha^{(t)} \in \mathbb{R}^k$. Then, we have that

$$\begin{aligned} \nabla f_t(z)\nabla f_t(z)^T &= B^{(t)T} \nabla g(B^{(t)}z)g(B^{(t)}z)^T B^{(t)} \\ &= x\alpha^{(t)T} \nabla g(B^{(t)}z)g(B^{(t)}z)^T \alpha^{(t)}x^T \\ &= (xx^T)(\alpha^{(t)T} \nabla g(B^{(t)}z)g(B^{(t)}z)^T \alpha^{(t)}) \\ &\propto xx^T . \end{aligned}$$

Similarly, we have that

$$B^{(t)T} B^{(t)} = x\alpha^{(t)T} \alpha^{(t)}x^T = (xx^T)(\alpha^{(t)T} \alpha^{(t)}) \propto xx^T .$$

□

We now extend the proposition above to the setting where we have multiple training samples, and we train for one step of gradient descent.

Proposition 9. Let $f(z) = g(Bz)$ with $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and $g : \mathbb{R}^k \rightarrow \mathbb{R}$. Assume $g(0) = 0$ and $\nabla g(0) \neq \mathbf{0}$. Given n training samples $\{(x_i, y_i)\}_{i=1}^n$, suppose that f is trained to minimize $\frac{1}{2} \sum_{i=1}^n (y_i - f(x_i))^2$ using gradient descent. If $B^{(0)} = \mathbf{0}$ and $f_1(z) := g(B^{(1)}z)$, then

$$\nabla f_1(z)\nabla f_1(z)^T \propto B^{(1)T} B^{(1)} .$$

Proof. Gradient descent proceeds as follows:

$$B^{(t+1)} = B^{(t)} + \eta \sum_{i=1}^n \nabla g(B^{(t)}x_i)(y_i - g(B^{(t)}x_i))x_i^T .$$

If $B^{(0)} = \mathbf{0}$, then $B^{(1)} = \eta \nabla g(0) \sum_{i=1}^n y_i x_i^T$. Thus, we have that

$$\begin{aligned} \nabla f_1(z) \nabla f_1(z)^T &= B^{(1)T} \nabla g(B^{(1)}z) g(B^{(1)}z)^T B^{(1)} \\ &= \eta^2 \left(\sum_{i=1}^n y_i x_i \nabla g(0)^T \right) \nabla g(B^{(1)}z) g(B^{(1)}z)^T \left(\nabla g(0) \sum_{i=1}^n y_i x_i^T \right) \\ &= \eta^2 \left(\sum_{i=1}^n y_i x_i \right) \left(\sum_{i=1}^n y_i x_i^T \right) \left(\nabla g(0)^T \nabla g(B^{(1)}z) g(B^{(1)}z)^T \nabla g(0) \right) \\ &\propto \left(\sum_{i=1}^n y_i x_i \right) \left(\sum_{i=1}^n y_i x_i^T \right). \end{aligned}$$

Similarly, we have:

$$\begin{aligned} B^{(1)T} B^{(1)} &= \eta^2 \left(\sum_{i=1}^n y_i x_i \nabla g(0)^T \right) \left(\nabla g(0) \sum_{i=1}^n y_i x_i^T \right) \\ &= \eta^2 \left(\sum_{i=1}^n y_i x_i \right) \left(\sum_{i=1}^n y_i x_i^T \right) \|\nabla g(0)\|_2 \\ &\propto \left(\sum_{i=1}^n y_i x_i \right) \left(\sum_{i=1}^n y_i x_i^T \right). \end{aligned}$$

□

Following a similar argument to that of Proposition 5, we now prove the ansatz for 1-hidden layer linear neural networks.

Proposition 10. *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ denote a two layer neural network of the form*

$$f(x) = A \frac{1}{\sqrt{k}} Bx;$$

where $A \in \mathbb{R}^{1 \times k}$, $B \in \mathbb{R}^{k \times d}$. Suppose only B is trainable. Let $B^{(t)}$ and $f^{(t)}$ denote updated weights after t steps of gradient descent on the dataset $(X, y) \in \mathbb{R}^{d \times n} \times \mathbb{R}^{1 \times n}$ with constant learning rate $\eta > 0$. If $\{A_i^{(0)}\}_{i=1}^k$ are i.i.d. random variables $\mathbb{E}[A_i^2] = 1$ and $B^{(0)} = \mathbf{0}$,

$$\lim_{k \rightarrow \infty} B^{(t)\top} B^{(t)} = \lim_{k \rightarrow \infty} \nabla f^{(t)} \nabla f^{(t)\top};$$

where $\nabla f^{(t)}$ is the gradient of $f^{(t)}$.¹

Proof of Proposition 10. The gradient descent updates proceed as follows:

$$B^{(t+1)} = B^{(t)} + \frac{\eta}{\sqrt{k}} A^{(0)\top} \left(y - A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X \right) X^\top.$$

We provide a proof by induction. We begin with the base case with $t = 1$. The base case follows from the

¹Note that since $f^{(t)}$ is linear, the gradient is constant and independent of the point at which it is taken.

fact that $\lim_{k \rightarrow \infty} \frac{1}{k} A^{(0)} A^{(0)\top} = 1$ and $B^{(1)} = \frac{\eta}{\sqrt{k}} A^{(0)\top} y X^\top$ and thus,

$$\begin{aligned} \lim_{k \rightarrow \infty} B^{(1)\top} B^{(1)} &= \lim_{k \rightarrow \infty} \frac{\eta^2}{k} X y^\top A^{(0)} A^{(0)\top} y X^\top = \eta^2 X y^\top y X^\top ; \\ \lim_{k \rightarrow \infty} \nabla f_1 \nabla f_1^\top &= \lim_{k \rightarrow \infty} B^{(1)\top} A^{(0)\top} \frac{1}{k} A^{(0)} B^{(1)} \\ &= \lim_{k \rightarrow \infty} \eta^2 X y^\top \left(A^{(0)} \frac{1}{k} A^{(0)\top} \right) \left(A^{(0)} \frac{1}{k} A^{(0)\top} \right) y X^\top = \eta^2 X y^\top y X^\top . \end{aligned}$$

Thus, we now assume the inductive hypothesis that

$$\lim_{k \rightarrow \infty} B^{(t)\top} B^{(t)} = \lim_{k \rightarrow \infty} \nabla f^{(t)} \nabla f^{(t)\top}$$

and analyze the case for timestep $t + 1$. We first have:

$$\begin{aligned} B^{(t+1)\top} B^{(t+1)} &= \left[B^{(t)} + \frac{\eta}{\sqrt{k}} A^{(0)\top} \left(y - A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X \right) X^\top \right]^\top \left[B^{(t)} + \frac{\eta}{\sqrt{k}} A^{(0)\top} \left(y - A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X \right) X^\top \right] \\ &= B^{(t)\top} B^{(t)} + B^{(t)\top} \frac{\eta}{\sqrt{k}} A^{(0)\top} y X^\top - B^{(t)\top} \frac{\eta}{k} A^{(0)\top} A^{(0)} B^{(t)} X X^\top \\ &\quad + \frac{\eta}{\sqrt{k}} X y^\top A^{(0)} B^{(t)} + \frac{\eta^2}{k} X y^\top A^{(0)} A^{(0)\top} y X^\top - \frac{\eta^2}{k} X y^\top A^{(0)} A^{(0)\top} A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X X^\top \\ &\quad - \frac{\eta}{k} X X^\top B^{(t)\top} A^{(0)\top} A^{(0)} B^{(t)} - \frac{\eta^2}{k} X X^\top B^{(t)\top} \frac{1}{\sqrt{k}} A^{(0)\top} A^{(0)} A^{(0)\top} y X^\top \\ &\quad + \frac{\eta^2}{k^2} X X^\top B^{(t)\top} A^{(0)\top} A^{(0)} A^{(0)\top} A^{(0)} B^{(t)} X X^\top . \end{aligned}$$

To simplify notation, we let

$$Z = \lim_{k \rightarrow \infty} A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} \quad ; \quad M = \lim_{k \rightarrow \infty} B^{(t)\top} B^{(t)},$$

noting that for $x \in \mathbb{R}^d$, Zx converges in distribution to a standard normal random variable by the central limit theorem. Taking the limit as $k \rightarrow \infty$, applying the inductive hypothesis and the fact that $\lim_{k \rightarrow \infty} \frac{1}{k} A^{(0)} A^{(0)\top} = 1$, we reduce the above to

$$\begin{aligned} \lim_{k \rightarrow \infty} B^{(t+1)\top} B^{(t+1)} &= M + \eta Z^\top y X^\top - \eta M X X^\top \\ &\quad + \eta X y^\top Z + \eta^2 X y^\top y X^\top - \eta^2 X y^\top Z X X^\top \\ &\quad - \eta X X^\top M - \eta^2 X X^\top Z^\top y X^\top + \eta^2 X X^\top M X X^\top . \end{aligned}$$

We will now show that $\lim_{k \rightarrow \infty} \nabla f^{(t+1)} \nabla f^{(t+1)^\top}$ is of the same form. Namely, we have

$$\begin{aligned}
\nabla f^{(t+1)} \nabla f^{(t+1)^\top} &= B^{(t+1)^\top} A^{(0)\top} \frac{1}{k} A^{(0)} B^{(t+1)} \\
&= B^{(t)^\top} A^{(0)\top} \frac{1}{k} A^{(0)} B^{(t)} + B^{(t)^\top} A^{(0)\top} \frac{1}{k} A^{(0)} \frac{\eta}{\sqrt{k}} A^{(0)\top} y X^\top \\
&\quad - B^{(t)^\top} A^{(0)\top} \frac{1}{k} A^{(0)} \frac{\eta}{k} A^{(0)\top} A^{(0)} B^{(t)} X X^\top \\
&\quad + \frac{\eta}{\sqrt{k}} X y^\top A^{(0)} A^{(0)\top} \frac{1}{k} A^{(0)} B^{(t)} + \frac{\eta^2}{k} X y^\top A^{(0)} A^{(0)\top} \frac{1}{k} A^{(0)} A^{(0)\top} y X^\top \\
&\quad - \frac{\eta^2}{k} X y^\top A^{(0)} A^{(0)\top} \frac{1}{k} A^{(0)} A^{(0)\top} A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X X^\top \\
&\quad - \frac{\eta}{\sqrt{k}} X X^\top B^{(t)^\top} A^{(0)\top} A^{(0)} A^{(0)\top} \frac{1}{k} A^{(0)} B^{(t)} \\
&\quad - \frac{\eta^2}{k} X X^\top B^{(t)^\top} \frac{1}{\sqrt{k}} A^{(0)\top} A^{(0)} A^{(0)\top} \frac{1}{k} A^{(0)} A^{(0)\top} y X^\top \\
&\quad + \frac{\eta^2}{k^2} X X^\top B^{(t)^\top} A^{(0)\top} A^{(0)} A^{(0)\top} \frac{1}{k} A^{(0)} A^{(0)\top} A^{(0)} B^{(t)} X X^\top .
\end{aligned}$$

Now taking the limit as $k \rightarrow \infty$, we reduce the above to

$$\begin{aligned}
\lim_{k \rightarrow \infty} \nabla f^{(t+1)} \nabla f^{(t+1)^\top} &= M + \eta Z^\top y X^\top - \eta M X X^\top \\
&\quad + \eta X y^\top Z + \eta^2 X y^\top y X^\top - \eta^2 X y^\top Z X X^\top \\
&\quad - \eta X X^\top M - \eta^2 X X^\top Z^\top y X^\top + \eta^2 X X^\top M X X^\top .
\end{aligned}$$

Hence, we conclude

$$\lim_{k \rightarrow \infty} B^{(t+1)^\top} B^{(t+1)} = \lim_{k \rightarrow \infty} \nabla f^{(t+1)} \nabla f^{(t+1)^\top},$$

which completes the proof by induction. \square

In the following proposition, we extend the previous analysis to the case of two layer linear neural networks where both layers are trained for two steps of gradient descent.

Proposition 11. *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ denote a two layer neural network of the form*

$$f(x) = A \frac{1}{\sqrt{k}} B x;$$

where $A \in \mathbb{R}^{1 \times k}$, $B \in \mathbb{R}^{k \times d}$. Let $A^{(t)}$, $B^{(t)}$ and $f^{(t)}$ denote updated weights after t steps of gradient descent on the dataset $(X, y) \in \mathbb{R}^{d \times n} \times \mathbb{R}^{1 \times n}$ with constant learning rate $\eta > 0$. If $\{A_i^{(0)}\}_{i=1}^k$ are i.i.d. random variables $\mathbb{E}[A_i^{(0)2}] = 1$ and $B^{(0)} = \mathbf{0}$,

$$\lim_{k \rightarrow \infty} B^{(2)^\top} B^{(2)} = \lim_{k \rightarrow \infty} \nabla f^{(2)} \nabla f^{(2)^\top};$$

where $\nabla f^{(2)}$ is the gradient of $f^{(2)}$.

Proof. We prove the statement directly. The gradient descent updates proceed as follows:

$$\begin{aligned} A^{(t+1)} &= A^{(t)} + \frac{\eta}{\sqrt{k}} \left(y - A^{(t)} \frac{1}{\sqrt{k}} B^{(t)} X \right) X^\top B^{(t)\top}, \\ B^{(t+1)} &= B^{(t)} + \frac{\eta}{\sqrt{k}} A^{(t)\top} \left(y - A^{(t)} \frac{1}{\sqrt{k}} B^{(t)} X \right) X^\top. \end{aligned}$$

Thus, after 1 step of gradient descent, we have

$$A^{(1)} = A^{(0)} \quad ; \quad B^{(1)} = \frac{\eta}{\sqrt{k}} A^{(0)\top} y X^\top.$$

From the proof of Proposition 10, we have that

$$\lim_{k \rightarrow \infty} B^{(1)\top} B^{(1)} = \lim_{k \rightarrow \infty} B^{(1)\top} \frac{1}{k} A^{(0)\top} A^{(0)} B^{(1)},$$

and so, we define the matrix M to be:

$$M := \lim_{k \rightarrow \infty} B^{(1)\top} B^{(1)}.$$

Next, after 2 steps of gradient descent, we have:

$$\begin{aligned} A^{(2)} &= A^{(1)} + \frac{\eta}{\sqrt{k}} \left(y - A^{(1)} \frac{1}{\sqrt{k}} B^{(1)} X \right) X^\top B^{(1)\top} \\ &= A^{(0)} + \frac{\eta}{\sqrt{k}} y X^\top B^{(1)\top} - \frac{\eta}{k} A^{(0)} B^{(1)} X X^\top B^{(1)\top} \\ &= A^{(0)} + \frac{\eta^2}{k} y X^\top X y^\top A^{(0)} - \frac{\eta^3}{k^2} A^{(0)} A^{(0)\top} y X^\top X X^\top X y^\top A^{(0)}; \end{aligned}$$

and

$$\begin{aligned} B^{(2)} &= B^{(1)} + \frac{\eta}{\sqrt{k}} A^{(1)\top} \left(y - A^{(1)} \frac{1}{\sqrt{k}} B^{(1)} X \right) X^\top \\ &= \frac{2\eta}{\sqrt{k}} A^{(0)\top} y X^\top - \frac{\eta}{k} A^{(0)\top} A^{(0)} B^{(1)} X X^\top \\ &= \frac{2\eta}{\sqrt{k}} B^{(1)} - \frac{\eta}{k} A^{(0)\top} A^{(0)} B^{(1)} X X^\top. \end{aligned}$$

Thus, we simplify $B^{(2)\top} B^{(2)}$ as follows:

$$\begin{aligned} B^{(2)\top} B^{(2)} &= \frac{4\eta^2}{k} B^{(1)\top} B^{(1)} - \frac{2\eta^2}{k\sqrt{k}} B^{(1)\top} A^{(0)\top} A^{(0)} B^{(1)} X X^\top \\ &\quad - \frac{2\eta^2}{k\sqrt{k}} X X^\top B^{(1)\top} A^{(0)\top} A^{(0)} B^{(1)} + \frac{\eta^2}{k^2} X X^\top B^{(1)\top} A^{(0)\top} A^{(0)} A^{(0)\top} A^{(0)} B^{(1)} X X^\top. \end{aligned}$$

Taking the limit as $k \rightarrow \infty$, we simplify the above expression to

$$\lim_{k \rightarrow \infty} B^{(2)\top} B^{(2)} = 4\eta^2 M + \eta^2 X X^\top M X X^\top.$$

A key observation is that as $k \rightarrow \infty$, the $O\left(\frac{1}{k}\right)$ and $O\left(\frac{1}{k^2}\right)$ terms in $A^{(2)}$ will vanish in the evaluation of $\lim_{k \rightarrow \infty} \nabla f^{(2)} \nabla f^{(2)\top}$ since the gradient also contains an extra $\frac{1}{\sqrt{k}}$ term from f . Hence only the $O(1)$ terms

given by $A^{(0)}$ will remain in the evaluation of $\lim_{k \rightarrow \infty} \nabla f^{(2)} \nabla f^{(2)\top}$. Using this observation, we have:

$$\lim_{k \rightarrow \infty} \nabla f^{(2)} \nabla f^{(2)\top} = \lim_{k \rightarrow \infty} B^{(2)\top} \frac{1}{k} A^{(2)\top} A^{(2)} B^{(2)} = \lim_{k \rightarrow \infty} B^{(2)\top} \frac{1}{k} A^{(0)\top} A^{(0)} B^{(2)},$$

which by the expansion of $B^{(2)\top} B^{(2)}$ and the proof of Proposition 10, is equivalent to $4\eta^2 M + \eta^2 X X^\top M X X^\top$. \square

The above results demonstrate that the ansatz holds for one-hidden-layer neural networks trained in isolation. We now prove the ansatz in the more general setting of deep, nonlinear fully connected networks by ensembling, or averaging over infinitely many networks. We present the proof of Theorem 10 below.

Proof. For a matrix $A_\alpha \in \mathbb{R}^{c \times d}$, we denote its p -th row by $A_{\alpha,p}$. For a vector $v \in \mathbb{R}^d$, we denote its i -th element by $v(i)$. To simplify notation, we drop the subscript t if it is irrelevant (e.g., fixed) in an expression. We consider the right hand side of the desired equation. The gradient with respect to the input is given by

$$\begin{aligned} \nabla_x f(x) &= \frac{c_\phi^L}{\prod_{\ell=1}^L k_\ell} \sum_{k'_L} \sum_{k'_{L-1}} \dots \sum_{k'_1} a_{k'_L} \phi'(W_{L,k'_L}^\top \phi_{L-1}(z)) \\ &\quad \cdot W_{L,k'_L}(k'_{L-1}) \phi'(W_{L-1,k'_{L-1}}^\top \phi_{L-2}(x)) \dots W_{3,k'_3}(k'_2) \cdot \phi'(W_{2,k'_2}^\top \phi_1(x)) \\ &\quad \cdot W_{2,k'_2}(k'_1) \phi'(W_{1,k'_1}^\top x) \cdot W_{1,k'_1}. \end{aligned}$$

Then,

$$\begin{aligned} \nabla_x f(x) \nabla_x f(x)^\top &= \frac{c_\phi^L}{\prod_{\ell=1}^L k_\ell} \\ &\quad \cdot \sum_{k'_L, k''_L} \sum_{k'_{L-1}, k''_{L-1}} \dots \sum_{k'_1, k''_1} \phi'(W_{L,k'_L}^\top \phi_{L-1}(x)) \dots \phi'(W_{2,k'_2}^\top \phi(W_1 x)) \phi'((W_{1,k'_1})^\top x) \\ &\quad \cdot \phi'(W_{L,k''_L}^\top \phi_{L-1}(x)) \dots \phi'(W_{2,k''_2}^\top \phi(W_1 x)) \phi'((W_{1,k''_1})^\top x) \\ &\quad \cdot a_{k'_L} W_{L,k'_L}(k'_{L-1}) \dots W_{3,k'_3}(k'_2) W_{2,k'_2}(k'_1) \\ &\quad \cdot a_{k''_L} W_{L,k''_L}(k''_{L-1}) \dots W_{3,k''_3}(k''_2) W_{2,k''_2}(k''_1) \\ &\quad \cdot W_{1,k'_1} W_{1,k''_1}^\top. \end{aligned}$$

By the gradient independence ansatz, we can generate new samples $\widetilde{W}_L, \dots, \widetilde{W}_2$,

$$\begin{aligned} \lim_{k_2, \dots, k_L \rightarrow \infty} \mathbb{E}_a \left[\nabla_x f(x) \nabla_x f(x)^\top \right] &= \lim_{k_2, \dots, k_L \rightarrow \infty} \mathbb{E}_a \frac{c_\phi^L}{\prod_{\ell=1}^L k_\ell} \sum_{k'_L, k''_L} \sum_{k'_{L-1}, k''_{L-1}} \dots \sum_{k'_1, k''_1} \\ &\quad \cdot \phi'(W_{L,k'_L}^\top \phi_{L-1}(x)) \dots \phi'(W_{2,k'_2}^\top \phi(W_1 x)) \phi'((W_{1,k'_1})^\top x) \\ &\quad \cdot \phi'(W_{L,k''_L}^\top \phi_{L-1}(x)) \dots \phi'(W_{2,k''_2}^\top \phi(W_1 x)) \phi'((W_{1,k''_1})^\top x) \\ &\quad \cdot a_{k'_L} \widetilde{W}_{L,k'_L}(k'_{L-1}) \dots \widetilde{W}_{3,k'_3}(k'_2) \widetilde{W}_{2,k'_2}(k'_1) \\ &\quad \cdot a_{k''_L} \widetilde{W}_{L,k''_L}(k''_{L-1}) \dots \widetilde{W}_{3,k''_3}(k''_2) \widetilde{W}_{2,k''_2}(k''_1) \\ &\quad \cdot W_{1,k'_1} W_{1,k''_1}^\top. \end{aligned}$$

Pulling factors outside of the limit,

$$\begin{aligned}
\lim_{k_2, \dots, k_L \rightarrow \infty} \mathbb{E}_a \left[\nabla_x f(x) \nabla_x f(x)^\top \right] &= \lim_{k_2, \dots, k_{L-1} \rightarrow \infty} \frac{c_\phi^L}{\prod_{\ell=1}^{L-1} k_\ell} \\
&\lim_{k_L \rightarrow \infty} \frac{1}{k_L} \sum_{k'_L} \left(\phi' \left(W_{L, k'_L}^\top \phi_{L-1}(x) \right) \right)^2 \\
&\sum_{k'_{L-1}, k''_{L-1}} \widetilde{W}_{L, k'_L}(k'_{L-1}) \widetilde{W}_{L, k'_L}(k'_{L-1}) \\
&\cdot \sum_{k'_{L-2}, k''_{L-2}} \dots \sum_{k'_1, k''_1} \phi' \left(W_{L-1, k'_{L-1}}^\top \phi_{L-2}(x) \right) \dots \phi' \left((W_{1, k'_1})^\top x \right) \\
&\cdot \phi' \left(W_{L-1, k''_{L-1}}^\top \phi_{L-2}(x) \right) \dots \phi' \left((W_{1, k''_1})^\top x \right) \\
&\cdot \widetilde{W}_{L-1, k'_{L-1}}(k'_{L-2}) \dots \widetilde{W}_{3, k'_3}(k'_2) \widetilde{W}_{2, k'_2}(k'_1) \\
&\cdot \widetilde{W}_{L-1, k''_{L-1}}(k''_{L-2}) \dots \widetilde{W}_{3, k''_3}(k''_2) \widetilde{W}_{2, k''_2}(k''_1) \\
&\cdot W_{1, k'_1} W_{1, k''_1}^\top.
\end{aligned}$$

Note that by re-sampling, $\left(\phi' \left(W_{L, k'_L}^\top \phi_{L-1}(x) \right) \right)^2$ is independent of the remaining terms, and so we can apply the law of large numbers as $k_L \rightarrow \infty$ and split the expectation as follows.

$$\begin{aligned}
\lim_{k_2, \dots, k_L \rightarrow \infty} \mathbb{E}_a \left[\nabla_x f(x) \nabla_x f(x)^\top \right] &= \lim_{k_2, \dots, k_{L-1} \rightarrow \infty} \frac{c_\phi^L}{\prod_{\ell=1}^{L-1} k_\ell} \\
&\mathbb{E}_{k'_L} \left[\left(\phi' \left(W_{L, k'_L}^\top \phi_{L-1}(x) \right) \right)^2 \right] \\
&\mathbb{E}_{k'_L} \left[\sum_{k'_{L-1}, k''_{L-1}} \widetilde{W}_{L, k'_L}(k'_{L-1}) \widetilde{W}_{L, k'_L}(k'_{L-1}) \right. \\
&\cdot \sum_{k'_{L-2}, k''_{L-2}} \dots \sum_{k'_1, k''_1} \phi' \left(W_{L-1, k'_{L-1}}^\top \phi_{L-2}(x) \right) \dots \phi' \left((W_{1, k'_1})^\top x \right) \\
&\cdot \phi' \left(W_{L-1, k''_{L-1}}^\top \phi_{L-2}(x) \right) \dots \phi' \left((W_{1, k''_1})^\top x \right) \\
&\cdot \widetilde{W}_{L-1, k'_{L-1}}(k'_{L-2}) \dots \widetilde{W}_{3, k'_3}(k'_2) \widetilde{W}_{2, k'_2}(k'_1) \\
&\cdot \widetilde{W}_{L-1, k''_{L-1}}(k''_{L-2}) \dots \widetilde{W}_{3, k''_3}(k''_2) \widetilde{W}_{2, k''_2}(k''_1) \\
&\left. \cdot W_{1, k'_1} W_{1, k''_1}^\top \right].
\end{aligned}$$

Evaluating the expectations above, we conclude:

$$\begin{aligned}
\lim_{k_2, \dots, k_L \rightarrow \infty} \mathbb{E}_a \left[\nabla_x f(x) \nabla_x f(x)^\top \right] &= \lim_{k_2, \dots, k_{L-1} \rightarrow \infty} \frac{c_\phi^{L-1}}{\prod_{\ell=1}^{L-1} k_\ell} \\
&\cdot \sum_{k'_{L-1}} \sum_{k'_{L-2}, k''_{L-2}} \dots \sum_{k'_1, k''_1} \phi' \left(W_{L-1, k'_{L-1}}^\top \phi_{L-2}(x) \right) \dots \phi' \left((W_{1, k'_1})^\top x \right) \\
&\cdot \phi' \left(W_{L-1, k'_{L-1}}^\top \phi_{L-2}(x) \right) \dots \phi' \left((W_{1, k''_1})^\top x \right) \\
&\cdot \widetilde{W}_{L-1, k'_{L-1}}(k'_{L-2}) \dots \widetilde{W}_{3, k'_3}(k'_2) \widetilde{W}_{2, k'_2}(k'_1) \\
&\cdot \widetilde{W}_{L-1, k''_{L-1}}(k''_{L-2}) \dots \widetilde{W}_{3, k''_3}(k''_2) \widetilde{W}_{2, k''_2}(k''_1) \\
&\cdot W_{1, k'_1} W_{1, k''_1}^\top .
\end{aligned}$$

Recursively applying this procedure yields

$$\lim_{k_2, \dots, k_L \rightarrow \infty} \mathbb{E}_a \left[\nabla_x f(x) \nabla_x f(x)^\top \right] = \frac{c_\phi}{k_1} \sum_{k'_1} \phi' \left((W_{1, k'_1})^\top x \right)^2 W_{1, k'_1} W_{1, k'_1}^\top .$$

Taking the expectation with respect to x ,

$$\mathbb{E}_x \left[\lim_{k_2, \dots, k_L \rightarrow \infty} \mathbb{E}_a \left[\nabla_x f(x) \nabla_x f(x)^\top \right] \right] = \frac{1}{k_1} \sum_{k'_1} W_{1, k'_1} W_{1, k'_1}^\top .$$

□

B Methods

Below, we provide a description of all datasets, models, and training methodology considered in this work.

Validating the Deep Neural Feature Ansatz All neural networks in Fig. 2A have 5 hidden layers with 1024 hidden units per layer and ReLU activation. We use minibatch gradient descent with a learning rate of 0.1 and batch size 128 for 500 epochs and initialize the first layer weights according to a Gaussian distribution with mean 0 and standard deviation 10^{-4} . All networks used in Fig. 2B and Supplementary Fig. D-1 have 5 hidden layers with 64 hidden units per layer and ReLU activation. We use minibatch gradient descent with a learning rate of 0.2 and batch size of 128 for 500 epochs and initialize the first layer weights according to a Gaussian distribution with mean 0 and standard deviation of 10^{-6} .

Spurious features. For experiment in Fig. 5-3A, we constructed a training set of size 50000 concatenated CIFAR-10 and MNIST digits, and a corresponding test set of 10000 test images. The training and test data were generated from data loaders provided by PyTorch. We used 20% of the training samples were used for validation. For Fig. 5-3A and B, we trained a five-hidden-layer fully connected ReLU network with 64 hidden units per layer using SGD with a learning rate of 0.2 and a mini-batch size of 128. We initialized first layer weights from a Gaussian with mean zero and standard deviation 10^{-4} . For Fig. 5-3C, we trained a two-hidden-layer fully connected ReLU network with 5 hidden units per layer using SGD for 500 epochs with a learning rate of 0.1 and a mini-batch size of 128. For all experiments, we train using the mean squared error (MSE) with one-hot labels for each of the classes.

Grokking. The total number of training and validation samples used is 553 with 500 examples of airplanes and 53 examples of trucks. We use 800 examples per class from the PyTorch test set as test data. We set a small stars of pixels (8 pixels tall, 7 pixels wide) in the upper left corner to yellow (all 1s in the green and red channel) if the image is a truck and all 0s if the image is a plane. We use 80% of the 553 samples for training and 20% for validation. We train a two hidden layer fully connected ReLU network using Adam with a learning rate of 10^{-4} and batch size equal to dataset size. We initialize the weights of the first layer

of the ReLU network according to a normal distribution with standard deviation 5×10^{-3} . We train RFMs updating only the diagonals of the feature matrix for three iterations with ridge regularization of 2.5×10^{-3} and using the Laplace kernel as the base kernel function with a bandwidth of 10. We used ridge regularization to slow down training of RFMs to visualize how the feature matrix changes through iteration. We note that without regularization, the RFM gets 100% test accuracy within 1 iteration.

Lottery Tickets. For all binary classification tasks on CelebA, we normalize all images to be on the unit sphere. We train 2-hidden layer ReLU networks with 1024 hidden units per layer using stochastic gradient descent (SGD) for 500 epochs with a learning rate of 0.1 and a mini-batch size of 128. We train using the mean squared error (MSE) with one-hot labels for each of the classes. Accuracy is reported as the argmax across classes. We split available training data into 80% training and 20% validation for hyper-parameter selection. We report accuracy on a held out test set provided by PyTorch [142]. In addition, since there can be large class imbalances in this data, we ensure that the training set and test set are balanced by limiting the number of majority class samples to the same number of minority class samples. Given that these are higher resolution images, we limit the total number of training and validation examples per experiment to 50000 (25000 per class). When re-training networks after masking to the top 2% of pixels with highest intensity in the diagonal of the first layer NFM, we re-initialize networks of the same architecture using the default PyTorch initialization scheme.

RFMs trained on CelebA. For the CelebA tasks in Appendix Fig. D-5, we train RFMs for 1 iteration, use a ridge regularization term of 10^{-3} , and average the gradient outer product of at most 20000 examples. All RFMs use Laplace kernels as the base kernel and use a bandwidth parameter of $L = 10$. We solve kernel ridge regression exactly via the solve function in numpy [186]. We use the same training, validation, and test splits considered in the lottery ticket experiments.

RFMs, neural networks, NTK, and Laplace kernels on SVHN. In Appendix Fig. D-6A, we train 2-hidden layer ReLU networks with 1024 hidden units per layer using stochastic gradient descent (SGD) for 500 epochs with a learning rate of 0.05 and a mini-batch size of 100. We train using the mean squared error (MSE) with one-hot labels for each of the classes. Accuracy is reported as the argmax across classes. We train RFMs for 5 iterations and average the gradient outer product of at most 20000 examples. We also center gradients during computation of RFMs by subtracting the mean of the gradients before computing the average gradient outer product. RFMs and Laplace kernels used all have a bandwidth parameter of 10. We compare with the NTK of a 2-hidden layer ReLU network. For all kernels, we solve kernel ridge regression with ridge term of 10^{-3} via the solve function in numpy [186]. The test accuracy for RFMs in Fig. D-6A is given by training a 1-hidden layer NTK with ridge regularization of 10^{-2} on the feature matrix selected from the last iteration of training, which resulted in the best validation accuracy.

RFMs, neural networks, NTK, and Laplace kernels on low rank polynomial regression. In Appendix Fig. D-6B and C, we consider the low rank polynomials from [189] and [52]. We use 1000 examples for training and 10000 samples for testing. Following the setup of [189], we sample training inputs from a Rademacher distribution in 30 dimensions and add random noise (see Appendix Fig. D-6B). The labels are generated by the product of the first two coordinates of the inputs without noise. We train a 1 hidden layer neural network for 1000 epochs using full batch gradient descent with a learning rate of .1 and initialize the first layer with standard deviation 10^{-3} so as to mitigate the effect of the initialization. We train RFMs with no ridge term and set the base kernel function as the Laplace kernel with bandwidth 10. We note the neural network was able to interpolate the training data and achieved a training R^2 of 1.

For the second low rank experiment in Appendix Fig. D-6C, we sample inputs, x , according to a 10 dimensional isotropic Gaussian distribution and sample a fixed vector, u , on the unit sphere in 10 dimensions. The targets are given by $g(u^T x)$ where $g(z) = \text{He}_2(z) + \text{He}_4(z)$ where He_2, He_4 are the second and fourth probabilist’s Hermite polynomials. We train a 1 hidden layer neural network using full batch Adam [101] with a learning rate of 10^{-2} and use the default PyTorch initialization. We train RFMs with no ridge term and set the base kernel function as the Laplace kernel with bandwidth 10. We note the neural network was able to nearly interpolate the training data within 1000 epochs and achieved a training R^2 of 0.971.

121 datasets from [61]. We first describe the experiments for 120 of the 121 datasets with fewer than 130000 examples since we used EigenPro [125] to train kernels on the largest dataset. For all kernel methods (RFMs, Laplace kernel and NTK), we grid search over ridge regularization from the set $\{10, 1, .1, .01, .001\}$. We grid search over 5 iterations for RFMs and used a bandwidth of 10 for all Laplace kernels. For NTK ridge regression experiments, we grid search over NTKs corresponding to ReLU networks with between 1 and 5 hidden layers. For the dataset with 130000 samples, we use EigenPro to train all kernel methods and RFMs. We run EigenPro for at most 50 iterations and select the iteration with best validation accuracy for reporting test accuracy. For small datasets (i.e., those with fewer than 5000 samples), we grid search over updating just the diagonals of M and updating the entire matrix M . Lastly, for all kernel methods and RFMs, we grid search over normalizing the data to the unit sphere. We note that there is one dataset (balance-scale), which had a data point with norm 0, and so we did not grid search over normalization for this dataset.

Tabular data benchmark from [70]. We used the repository from [70] at <https://github.com/LeoGrin/tabular-benchmark>, modifying the code as needed to incorporate our method. On all datasets, we grid search over 5 iterations of RFM with the Laplace kernel, solving kernel regression in closed form at all steps. This benchmark consists of 20 medium regression datasets (without categorical variables), 3 large regression datasets (without categorical variables), 15 medium classification datasets (without categorical variables), 4 large classification datasets (without categorical variables), 13 medium classification datasets (with categorical variables), 5 large regression datasets (with categorical variables), 7 medium classification datasets (with categorical variables), and 2 large classification datasets (with categorical variables). Following the terminology from [70], “medium” refers to datasets with at most 10000 training examples and “large” refers to those with more than 10000 training examples. In general, we grid-searched over ridge regularization parameters in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ with fixed bandwidth $L = 10$. For regression, we centered the labels and scaled their variance to 1. On large regression datasets, we also optimized for bandwidths over $\{1, 5, 10, 15, 20\}$. We searched over two target transformations - the log transform ($\hat{y} = |y| \log(1 + |y|)$) and `sklearn.preprocessing.QuantileTransformer`. In both cases, we inverted the transform before testing. We also searched over data transformations - `sklearn.preprocessing.StandardScaler` and `sklearn.preprocessing.QuantileTransformer`. We also optimized for the use of centering/not centering the gradients in our computation, and extracting just the diagonal of the feature matrix. For non-kernel methods, we compare to the metrics reported in [70]. For classification, we report the average accuracy across the random iterations in each sweep (including random train/val/test splits). For regression, the average R^2 is reported. The reported test score is the average performance of the model with the highest average validation performance.

We next provide a description of all metrics considered in the tabular benchmarks.

Friedman Rank. To compute Friedman rank, we rank classifiers in order of performance (e.g. the top performer gets rank 1) for each dataset and then average the ranks. In the original results of [61], certain classifiers were missing performance values. To compute the Friedman rank, the authors of [61] impute such missing entries via the average classifier performance for this data. We provide code for computing the Friedman rank that replicates the ranks provided in the original work of [61].

Average Accuracy. Average accuracy is just the average over all available accuracies across datasets. In this case, missing accuracies are not imputed for the average and are simply dropped.

Percentage of Maximum Accuracy (PMA). An average over the percentage of the best classifier accuracy achieved by a given model across all datasets.

P90/P95. An average over all datasets for which a classifier achieves within 90%/95% of the accuracy of the best model.

Average Distance to Minimum (ADTM). This metric normalizes for variance in the hardness of different datasets. Let x_{ij} be the performance of method j for dataset i , the ADTM for method j is defined as $\text{ADTM}_j = \text{Avg}_i \left(\frac{x_{ij} - \min_j x_{ij}}{\max_j x_{ij}} \right)$. Note $\text{ADTM} \in [0, 1]$, with 1 indicating a method is the best among all methods in the collection, and 0 indicating a method is the worst.

C Background on Kernel Ridge Regression

We here provide a brief review of kernel ridge regression [166]. Given a dataset $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$ and a Hilbert space, \mathcal{H} , kernel ridge regression constructs a non-parametric estimator given by

$$\hat{f}_{n,\lambda} = \underset{f \in \mathcal{H}}{\text{argmin}} \sum_{i=1}^n (f(x_i) - y_i)^2 + \lambda \|f\|_{\mathcal{H}}^2; \quad (\text{D.1})$$

where $\lambda \geq 0$ is referred to as the ridge regularization parameter. Note this is an infinite dimensional optimization problem in a Reproducing Kernel Hilbert Space, \mathcal{H} , corresponding to a positive semi-definite kernel function K . By virtue of the Representer theorem [190], this problem has a unique solution in the span of the data given by

$$\hat{f}_{n,\lambda} = \sum_{i=1}^n \hat{\alpha}_i K(x, x_i) \quad \text{where} \quad \hat{\alpha} = y(K(X, X) + \lambda I_n)^{-1}; \quad (\text{D.2})$$

where $K(X, X)_{ij} = K(x_i, x_j)$ and $y \in \mathbb{R}^{1 \times n}$. Naively, this involves solving a $n \times n$ linear system, which can be typically solved in closed form for $n \leq 100,000$. For $n > 100,000$, we apply the EigenPro solver [125] to approximately solve kernel regression via early-stopped, preconditioned-SGD that can run on the GPU. For $\lambda \rightarrow 0^+$, we recover the pseudo-inverse solution $\hat{\alpha} = yK(X, X)^\dagger$. For multi-class and multi-variate problems, y_i are vector valued and we consider each class/target variable as a separate problem.

D Kernel alignment and gradient outer product

To improve kernel selection for supervised learning, a line of research [47, 49, 172] considered selecting a kernel or a combination of kernels to maximize alignment with the following, *ideal* kernel, function.

Definition 12. Suppose data $(x, f(x)) \in \mathbb{R}^d \times \mathbb{R}$ are generated by a target function $f(x)$. Then, the *ideal kernel* is $K^*(x, z) = f(x)f(z)$.

If one knows the target function f beforehand, then the ideal feature map is $\psi(x) = f(x)$, as the predictor $1^T \psi(x)$ will recover the target value exactly (assuming no label noise). Further, in the Bayesian setting, the ideal kernel averaged over the distribution of target functions will be optimal [97]. We now showcase a benefit of the expected gradient outer product, M , by demonstrating that regression with a Mahalanobis kernel using M will recover the ideal kernel when the target function is linear.

Proposition 12. Let $x \in \mathbb{R}^d$ have density ρ , let $\beta \in \mathbb{R}^d$, and consider the linear model, i.e., $f(x) = \beta^T x$. For $z, z' \in \mathbb{R}$, let $K_M(z, z') = z^T M z'$ with $M = \mathbb{E}_x[\nabla f(x)\nabla f(x)^T]$. Then, $K_M = K^*$.

Proof. Note $\nabla f(x) = \beta$ for all x . Hence, $M = \beta\beta^T$, and $K_M(z, z') = z^T \beta\beta^T z' = f(z)f(z') = K^*(z, z')$. \square

Moreover, the expected gradient outer product will provably reduce the sample complexity when the target function depends on only a few relevant directions in the data, as implied by the following proposition.

Proposition 13. Let $x \in \mathbb{R}^d$ have density ρ and let the target function $f^* : \mathbb{R}^d \rightarrow \mathbb{R}$ be a polynomial with degree p and rank r , i.e., $f(x) = g(Ux)$ where $U \in \mathbb{R}^{r \times d}$ and $g : \mathbb{R}^r \rightarrow \mathbb{R}$. Let $M = \mathbb{E}_x[\nabla f^*(x)\nabla f^*(x)^T] \in \mathbb{R}^{d \times d}$. Then, there exists a fixed polynomial kernel such that kernel ridge regression on the transformed data $(M^{\frac{1}{2}}X, y)$ has the minimax sample dependence on rank, $O(r^p)$.

Proof of Proposition 13. The gradient of the target function in directions orthogonal to the target subspace U is 0, as the function does not vary in these directions. Thus, $\nabla f^*(x)$ is in the span of U . Hence, for any $x' \in \mathbb{R}^d$, as

$$Mx' = \mathbb{E}_{x \in X_N} [\nabla f^*(x) \nabla f^*(x)^T x'] ,$$

we have that Mx' is also in the span of U . Therefore, the transformed data $M^{\frac{1}{2}}X$ lies in an r -dimensional subspace and has an equivalent representation in an r -dimensional coordinate space. Namely, for all $i, j \in [d]$, there exists $\alpha_i, \alpha_j \in \mathbb{R}^r$ such that $\|x_i - x_j\| = \|\alpha_i - \alpha_j\|$. Further, the degree of the target function does not change under linear transformation or rotation. The final bound follows from the generalization error bound of linear regression for kernel ridge regression with a polynomial kernel of degree r . \square

Remarks. This result is in contrast to using a fixed kernel for which $\Omega(d^p)$ samples are required to achieve better error than the trivial 0-function by kernel ridge regression [66]. While the above propositions assume we have knowledge of the expected gradient outer product of the target function, we note that related algorithms are optimal, even when the expected gradient outer product has not been estimated exactly. For example, kernel ridge regression using a Mahalanobis kernel with M set to the neural feature matrix after 1 step of gradient descent gives the optimal dependence on the rank r under certain conditions on the target function [52]. We note that a related iterative procedure using kernel smoothers to simultaneously estimate a predictor and gradients achieves minimax optimality for low-rank function estimation [88].

A Correlation between Average gradient outer product and Trained NFM

Layer	Smiling	Glasses	Rosy Cheeks	5 o'clock Shadow	Necktie	SVHN
1	0.9977	0.9892	0.9907	0.9926	0.9704	0.9519
2	0.9891	0.9837	0.9920	0.9584	0.9199	0.8642
3	0.9415	0.6936	0.8734	0.8784	0.9429	0.7739
4	0.9723	0.7316	0.7765	0.9225	0.9012	0.7624
5	0.9805	0.9927	0.9781	0.9618	0.8170	0.5952

B Correlation between Initial NFM and Trained NFM

Layer	Smiling	Glasses	Rosy Cheeks	5 o'clock Shadow	Necktie	SVHN
1	0.0003	0.0004	0.0004	0.0004	0.0006	0.0095
2	0.1662	0.1614	0.1398	0.1702	0.2426	0.3324
3	0.2069	0.1555	0.1800	0.1814	0.1890	0.3868
4	0.2028	0.2011	0.2226	0.1962	0.1802	0.4316
5	0.1793	0.1665	0.1896	0.1939	0.2016	0.3787

Figure D-1: **(A)** The correlation between the average gradient outer product and the trained NFM for each layer in five-hidden-layer ReLU fully connected networks trained on 6 image classification tasks from CelebA and SVHN. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix. **(B)** The correlation between initial NFM and trained NFM for each layer in five-hidden-layer ReLU fully connected networks trained on 6 image classification tasks from CelebA and SVHN.

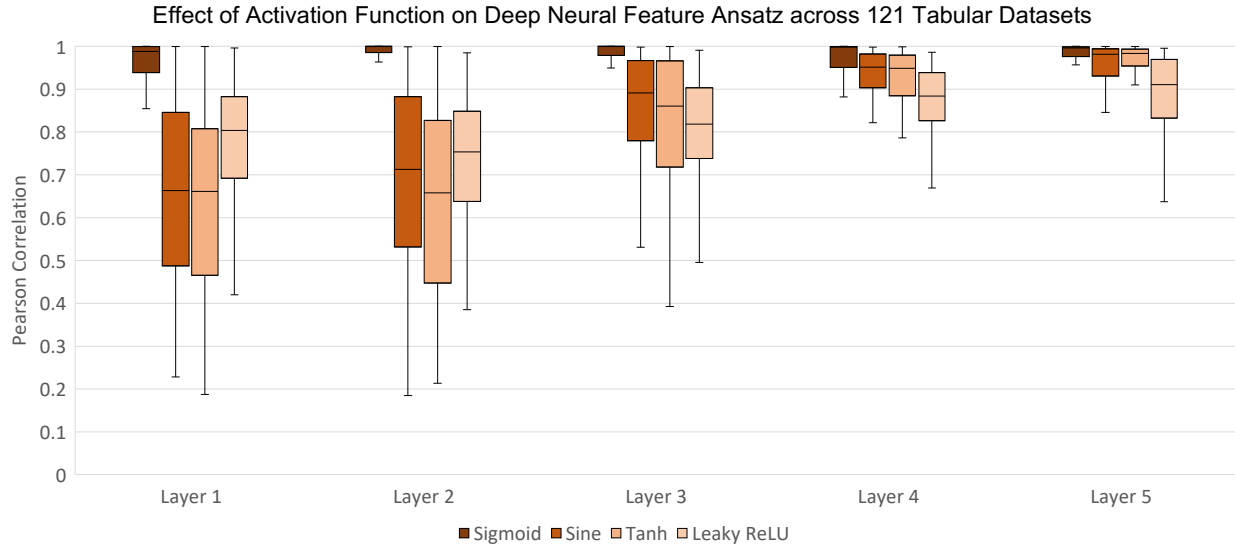


Figure D-2: The correlation between the average gradient outer product and the trained NFM for each layer in five-hidden-layer, width 64 fully connected networks trained on 121 tabular classification tasks using the Adam optimizer with learning rate of 10^{-4} and default initialization scheme from PyTorch across sigmoid, sine, hyperbolic tangent, and leaky ReLU activation. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix. For all activations across all layers, median correlation is above .65.

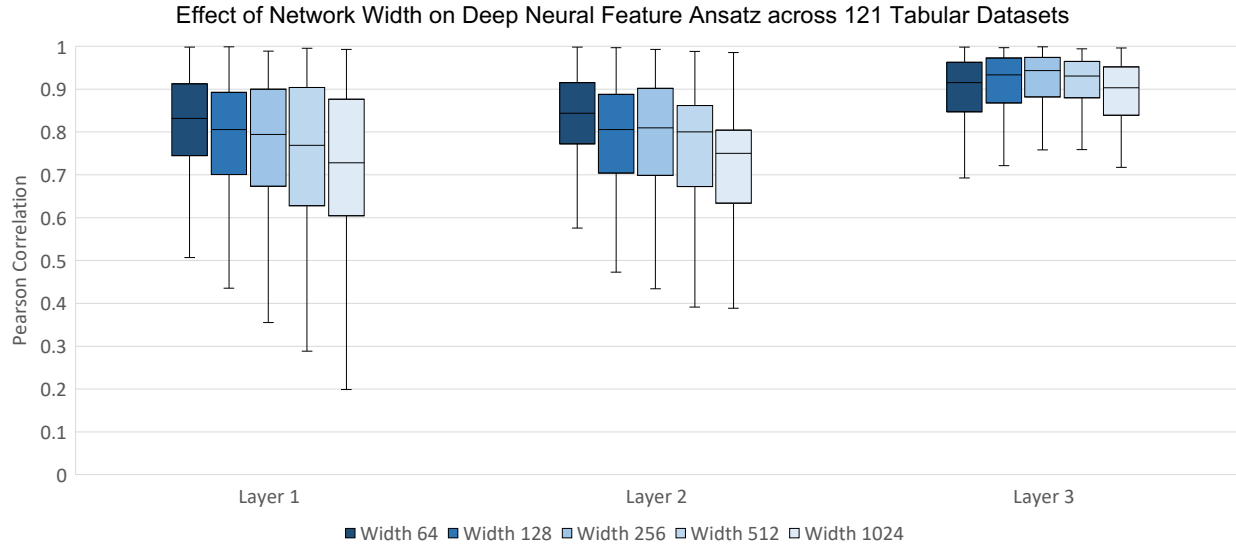


Figure D-3: The correlation between the average gradient outer product and the trained NFM for each layer in five-hidden-layer ReLU fully connected networks trained on 121 tabular classification tasks using the Adam optimizer with learning rate of 10^{-4} and default initialization scheme from PyTorch across widths in the range $\{64, 128, 256, 512, 1024\}$. We observe that lower width leads to higher correlation. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix.

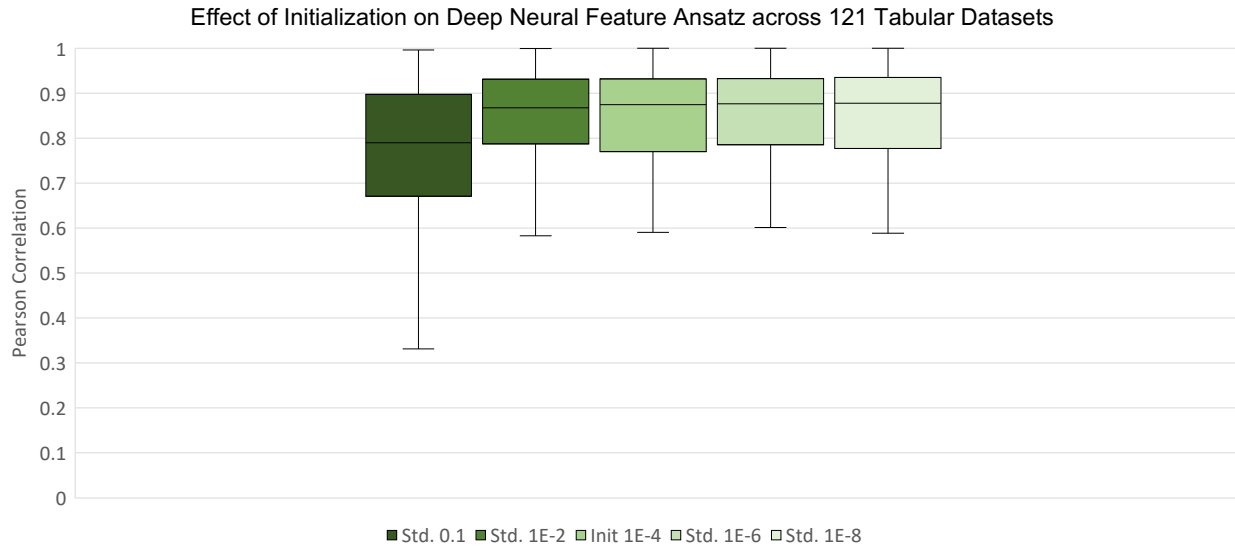


Figure D-4: The correlation between the average gradient outer product and the trained NFM for the first layer in 1-hidden-layer, width 256 ReLU fully connected networks trained on 121 tabular classification tasks using the Adam optimizer with learning rate of 10^{-4} using an initialization scheme of Gaussian with mean 0 and standard deviation in the range $\{10^{-1}, 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}\}$. To compute the trained NFM, we subtract the layer weights at initialization from the final weights before computing the Gram matrix. We observe that lower initialization scheme leads to higher correlation.

Task	Lipstick	Eyebrows	5 o'clock shadow	Necktie	Smiling	Rosy Cheeks
First Layer NFM (Top Eigenvector)						
RFM Feature Matrix:						
Correlation	0.999	0.999	0.999	0.999	0.999	0.999

Task	Glasses	Mustache	Goatee	Hat	Blonde	Male
First Layer NFM (Top Eigenvector)						
RFM Feature Matrix (Top Eigenvector)						
Correlation	0.999	0.999	0.999	0.995	0.997	0.999

Figure D-5: The top eigenvector of the first layer NFM from a two-hidden-layer, 1024 width ReLU network and from RFM feature matrices visually highlight similar features across 12 CelebA classification tasks. These top eigenvectors are highly correlated (Pearson correlation greater than 0.99).

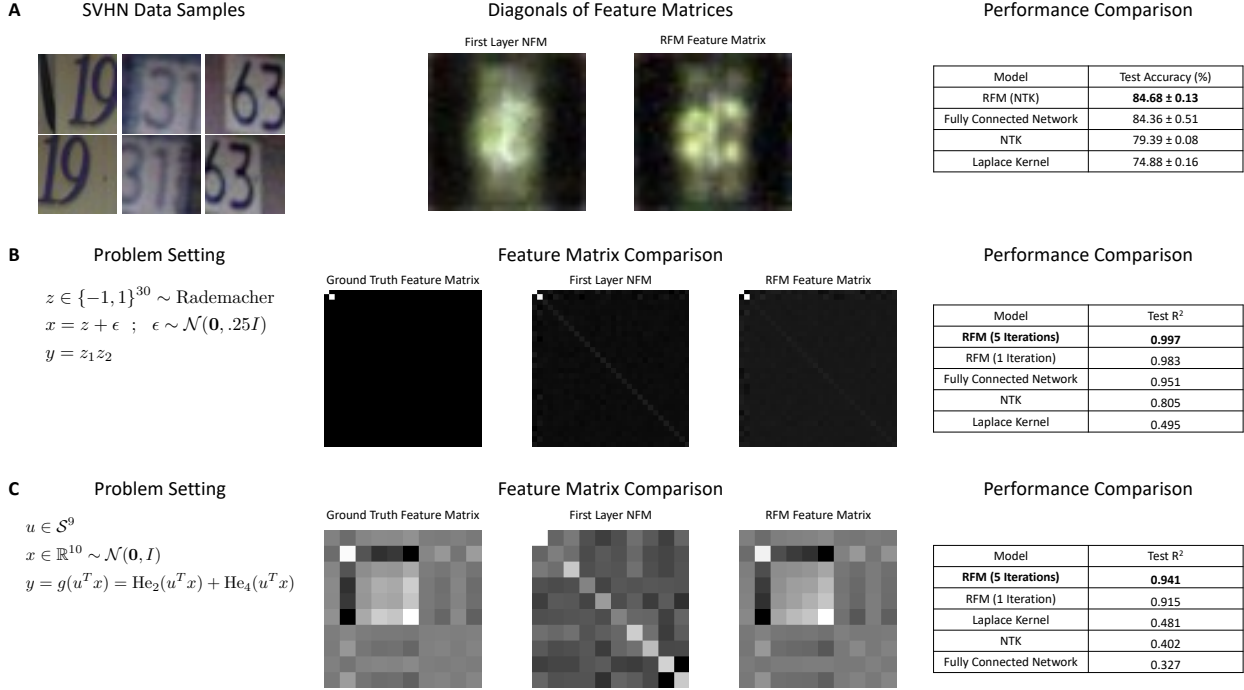


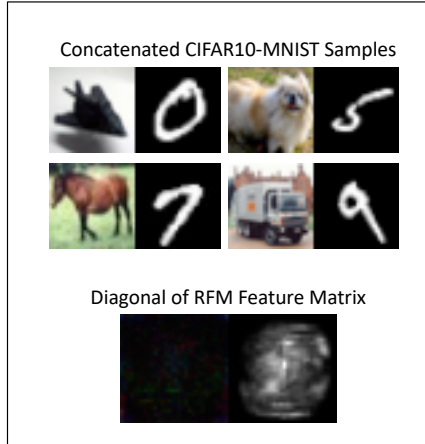
Figure D-6: Comparison of first layer NFM features and RFM features and performance between two-hidden-layer, 1024 width ReLU fully connected networks and RFMs. **(A)** (Left) Samples from the SVHN dataset in which the goal is to identify the center digit from a view of potentially many digits. (Center) Upon visualizing the diagonals of the feature matrices of RFMs and deep networks, we observe that these models learn to select the center digit. (Right) By selecting the center digit, RFMs and deep networks provide a 10% increase in test accuracy over Laplace kernels and a 5% increase in test accuracy over NTKs. **(B)** (Left) We consider the low rank setup from [189] in which the targets, y , are generated as a product of the first two coordinates of Rademacher random variables z . (Center) Since we know the target function, we can compare the ground truth feature matrix against the first layer NFM of a 1 hidden layer ReLU fully connected network and the RFM feature matrix. We observe that both models learn to select the top two coordinates. (Right) The performance of RFMs and neural networks far exceeds of NTKs and Laplace kernels since these methods learn to select relevant coordinates for prediction. **(C)** (Left) The low rank setup from [52] in which the targets, y are generated as a function of a projection of the input x onto a 1 dimensional subspace. Here, u is on the unit sphere in 10 dimensions and He_2, He_4 denote the second and fourth probabilist's Hermite polynomials. (Center) While RFMs learn to accurately approximate the ground truth gradient outer product, fully connected networks require additional training modifications, as discussed in [52]. (Right) As they learn the relevant subspace, RFMs far outperform 1 hidden layer ReLU fully connected networks, NTKs, and the Laplace kernel.

Lottery Tickets and RFMs

Task	Glasses	Mustache	Goatee	Eyebrows	Rosy Cheeks	Smiling
Thresholded RFM Features						
Percent of Pixels Remaining	1.42%	1.66%	3.40%	1.44%	2.43%	1.76%
RFM Accuracy	90.92%	88.15%	89.40%	74.36%	86.66%	89.62%
RFM-T Accuracy	94.06%	91.32%	91.19%	78.11%	88.72%	91.24%

Figure D-7: Connections between lottery tickets in deep networks and RFMs. The diagonals of the feature matrices of RFMs trained on CelebA are sparse, thereby indicating that only a subset of coordinates is used for prediction. Such sparsity suggests that we can threshold to very few pixels while still minimally affecting performance. Indeed, re-training RFMs upon thresholding to less than 3.5% of total pixels in CelebA tasks consistently improves performance for these tasks.

A CIFAR10-MNIST classification with RFMs



B Lipstick classification with RFMs

Diagonal of RFM Feature Matrix:			
Corruption	Mask	Corrupted Samples	Test Acc.
None			90.71%
Lips (1260 Pixels)			84.69%
Eyes (477 Pixels)			51.65%

Figure D-8: RFMs accurately capture simplicity biases of deep fully connected networks. **(A)** We train RFMs on a dataset similar to that from [168] in which we concatenate images of CIFAR-10 objects with unique digits from MNIST. Upon visualizing the diagonals of the feature matrices of RFMs, we observe that the model learns to mask the CIFAR-10 image and focus on the MNIST digit for prediction. **(B)** The diagonal of the feature matrix for an RFM trained on lipstick classification unusually indicates that eyes are used as a key feature. We thus construct a mask based on the top RFM features and replace the eyes of all test samples with those of a single individual. The trained RFM does 39.06% worse on these corrupted samples. On the other hand, replacing the lips of all test samples with those from the same individual leads to only a minor, 6.02%, decrease in accuracy.

A Accuracy (%) across large classification datasets from Grinsztajin et al. 2022							
Dataset	RFM	XG Boost	Gradient Boosting Tree	Resnet	SAINT	MLP	FT Transformer
MiniBooNE	94.97	94.47	94.14	94.61	94.32	94.32	94.56
HIGGS	72.44	72.87	72.55	72.39	72.75	71.15	73.13
Coverttype	94.10	89.74	89.76	89.39	89.59	87.43	90.69
Jannis	80.68	79.56	79.47	78.59	79.77	76.45	79.74

B R ² across large regression datasets from Grinsztajin et al. 2022						
Dataset	RFM	XGBoost	Gradient Boosting Tree	Resnet	SAINT	FT Transformer
Diamonds	0.948	0.948	0.947	0.941	0.945	0.945
NYC Taxi Green Dec. 2016	0.569	0.629	0.624	0.247	0.534	0.120
Year	0.334	0.307	0.307	0.119	0.289	0.117

C Metrics across all classification datasets from Grinsztajin et al. 2022								
Metric	RFM	XG Boost	Gradient Boosting Tree	Random Forest	Resnet	SAINT	MLP	FT Transformer
Average Accuracy (%)	83.84	83.63	83.28	83.31	81.81	82.43	81.13	82.62
ADTM	0.810	0.762	0.641	0.555	0.286	0.458	0.089	0.549
PMA	0.991	0.990	0.986	0.986	0.967	0.975	0.959	0.977
Friedman rank	± 0.017	± 0.014	± 0.013	± 0.010	± 0.029	± 0.026	± 0.030	± 0.026
Friedman rank	2.39	2.63	3.95	4.24	6.05	5.13	7.37	4.24

D Metrics across all regression datasets from Grinsztajin et al. 2022							
Metric	RFM	XGBoost	Gradient Boosting Tree	Random Forest	Resnet	SAINT	FT Transformer
Average R ²	0.776	0.780	0.771	0.758	0.749	0.770	0.744
ADTM	0.701	0.823	0.534	0.322	0.489	0.572	0.504
PMA	0.984	0.988	0.974	0.949	0.928	0.970	0.919
Friedman rank	± 0.031	± 0.023	± 0.037	± 0.083	± 0.173	± 0.042	± 0.203
Friedman rank	3.00	2.13	4.00	5.52	4.52	4.30	4.52

Figure D-9: Performance of RFMs, XGBoost, Gradient Boosting Trees, Random Forests, ResNets, Transformers (SAINT and FT), and fully connected networks (MLPs) from [70]. Our method used 3600 hours in total, while all other methods used 20000 hours for tuning, as reported in [70]. Results from all models other than RFMs are reported from the tables provided by [70]. All metrics and training details are outlined in Methods. Large tasks have 50000 training examples except for Jannis (40306 examples) and Diamonds (37758 examples). The medium tasks have at most 10000 samples. We show (A) average accuracy on large classification tasks and (B) average R^2 on large regression tasks. We compare model performance through commonly used metrics across all datasets for (C) classification, and (D) regression.

n-train	data-keyword	FT Transformer	GradientBoostingTree	RandomForest	Resnet	SAINT	XGBoost	ours	best method
4547.0	wine-quality	-	0.458	0.504	-	-	0.498	0.497	RandomForest
5457.0	isolet	-	-	-	-	-	-	0.868	ours
5734.0	cpu-act	-	0.985	0.983	-	-	0.986	0.986	XGBoost
7056.0	sulfur	-	0.806	0.859	-	-	0.865	0.913	ours
7484.0	Brazilian-houses	-	0.996	0.993	-	-	0.998	0.938	XGBoost
9625.0	Ailerons	-	0.843	0.839	-	-	0.844	0.841	XGBoost
9752.0	MiamiHousing2016	-	0.924	0.924	-	-	0.936	0.934	XGBoost
10000.0	Bike-Sharing-Demand	-	0.69	0.687	-	-	0.695	0.689	XGBoost
10000.0	california	-	0.846	0.83	-	-	0.853	0.867	ours
10000.0	diamonds	-	0.945	0.945	-	-	0.946	0.945	XGBoost
10000.0	elevators	-	0.863	0.841	-	-	0.908	0.923	ours
10000.0	fifa	-	0.663	0.655	-	-	0.668	0.653	XGBoost
10000.0	house-16H	-	0.541	0.486	-	-	0.548	0.489	XGBoost
10000.0	house-sales	-	0.884	0.871	-	-	0.887	0.883	XGBoost
10000.0	houses	-	0.84	0.829	-	-	0.852	0.864	ours
10000.0	medical-charges	-	0.979	0.979	-	-	0.979	0.979	GradientBoostingTree
10000.0	nyc-taxi-green-dec-2016	-	0.554	0.563	-	-	0.553	0.532	RandomForest
10000.0	pol	-	0.99	0.989	-	-	0.99	0.991	ours
10000.0	superconduct	-	0.905	0.909	-	-	0.911	0.911	ours
10000.0	year	-	0.271	0.241	-	-	0.282	0.303	ours
37758.0	diamonds	0.945	0.947	-	0.941	0.945	0.948	0.948	ours
50000.0	nyc-taxi-green-dec-2016	0.12	0.624	-	0.247	0.534	0.629	0.569	XGBoost
50000.0	year	0.117	0.307	-	0.119	0.289	0.307	0.334	ours

Table D.1: Regression R^2 without categorical variables.

n-train	data-keyword	FT Transformer	GradientBoostingTree	MLP	RandomForest	Resnet	SAINT	XGBoost	ours	best method
1787.0	wine	77.54	78.77	77.62	78.96	77.95	77.09	79.85	80.67	ours
2220.0	phoneme	85.28	86.78	84.95	88.55	86.21	85.58	86.48	88.16	RandomForest
3631.0	kdd-ipums-la-97-small	88.96	88.38	87.95	87.95	88.07	89.02	88.26	88.62	SAINT
5325.0	eye-movements	58.62	63.75	56.89	65.04	57.41	58.93	65.54	61.14	XGBoost
7057.0	pol	98.47	97.94	94.27	98.21	94.81	98.14	98.05	98.33	FT Transformer
7404.0	bank-marketing	80.42	80.27	79.18	79.82	79.37	79.09	80.44	79.73	XGBoost
9363.0	MagicTelescope	85.09	85.88	84.7	85.6	85.78	85.1	85.92	86.5	ours
9441.0	house-16H	88.16	88.2	87.78	87.8	87.5	88.16	88.83	87.78	XGBoost
10000.0	Higgs	70.62	71.08	68.86	70.76	69.44	70.72	71.42	70.73	XGBoost
10000.0	MiniBooNE	93.74	93.19	93.54	92.65	93.68	93.52	93.62	93.93	ours
10000.0	california	88.62	89.82	86.0	89.21	87.57	89.07	90.17	90.29	ours
10000.0	covertime	81.32	81.87	78.89	82.73	80.26	80.31	81.9	85.95	ours
10000.0	credit	76.53	77.22	75.99	77.28	76.1	75.99	77.38	77.66	ours
10000.0	electricity	82.0	86.16	81.04	86.14	80.86	81.77	86.83	82.93	XGBoost
10000.0	jannis	76.55	77.02	74.57	77.27	74.6	77.26	77.78	78.28	ours
40306.0	jannis	79.74	79.47	76.45	78.85	78.59	79.77	79.56	80.68	ours
50000.0	Higgs	73.13	72.55	71.15	71.98	72.39	72.75	72.83	72.44	FT Transformer
50000.0	MiniBooNE	94.34	94.14	94.32	93.53	94.44	94.32	94.43	94.97	ours
50000.0	covertime	90.69	89.76	87.43	90.59	89.39	89.53	89.74	94.1	ours

Table D.2: Classification accuracy without categorical variables.

n-train	data-keyword	FT Transformer	GradientBoostingTree	HistGradientBoostingTree	RandomForest	Resnet	XGBoost	ours	best method
2836.0	analcata-supreme	0.977	0.981	0.982	0.981	0.978	0.983	0.987	ours
2946.0	Mercedes-Benz-Greener-Manufacturing	0.548	0.578	0.576	0.575	0.545	0.578	0.575	GradientBoostingTree
6048.0	visualizing-soil	0.998	1.0	1.0	1.0	0.998	1.0	1.0	RandomForest
6219.0	yprop-4-1	0.037	0.056	0.063	0.095	0.021	0.08	0.072	RandomForest
7484.0	Brazilian-houses	0.883	0.995	0.993	0.993	0.878	0.998	0.906	XGBoost
9999.0	OnlineNewsPopularity	0.143	0.153	0.156	0.149	0.13	0.162	0.139	XGBoost
10000.0	Bike-Sharing-Demand	0.937	0.942	0.942	0.938	0.936	0.946	0.933	XGBoost
10000.0	SGEMM-GPU-kernel-performance	1.0	1.0	1.0	1.0	1.0	1.0	1.0	ours
10000.0	black-friday	0.379	0.615	0.616	0.609	0.36	0.619	0.612	XGBoost
10000.0	diamonds	0.99	0.99	0.991	0.988	0.989	0.991	0.991	XGBoost
10000.0	house-sales	0.891	0.891	0.89	0.875	0.881	0.896	0.891	XGBoost
10000.0	nyc-taxi-green-dec-2016	0.511	0.573	0.539	0.585	0.451	0.578	0.549	RandomForest
10000.0	particulate-matter-ukair-2017	0.673	0.683	0.69	0.674	0.658	0.691	0.662	XGBoost
37758.0	diamonds	-	0.992	0.993	-	-	0.993	0.993	ours
50000.0	SGEMM-GPU-kernel-performance	-	1.0	1.0	-	-	1.0	1.0	XGBoost
50000.0	black-friday	-	0.631	0.636	-	-	0.639	0.623	XGBoost
50000.0	nyc-taxi-green-dec-2016	-	0.636	0.585	-	-	0.648	0.589	XGBoost
50000.0	particulate-matter-ukair-2017	-	0.706	0.71	-	-	0.712	0.69	XGBoost

Table D.3: Regression R^2 with categorical variables.

n-train	data-keyword	FT Transformer	GradientBoostingTree	HistGradientBoostingTree	RandomForest	Resnet	SAINT	XGBoost	ours	best method
3479.0	rl	70.31	77.62	76.05	79.79	70.56	68.2	77.01	70.47	RandomForest
3522.0	KDDCup09-upselling	78.05	-	-	-	76.98	77.8	-	-	FT Transformer
3589.0	KDDCup09-upselling	-	80.36	80.61	80.02	-	-	79.56	77.24	HistGradientBoostingTree
5325.0	eye-movements	59.83	63.94	63.58	65.73	57.93	58.54	66.77	62.75	XGBoost
10000.0	compass	75.34	74.09	75.15	79.28	74.46	71.87	76.91	77.5	RandomForest
10000.0	covertime	86.74	85.56	84.47	85.89	85.27	84.95	86.42	89.5	ours
10000.0	electricity	84.16	87.97	88.15	87.76	82.64	83.35	88.69	85.33	XGBoost
10000.0	road-safety	76.74	76.21	76.45	75.88	76.08	76.43	76.69	75.48	FT Transformer
50000.0	covertime	93.61	93.22	92.09	93.35	92.24	92.58	93.31	95.58	ours
50000.0	road-safety	78.95	78.73	78.85	78.13	78.41	77.96	80.29	77.56	XGBoost

Table D.4: Classification accuracy with categorical variables.

Bibliography

- [1] Netflix prize rules, 2009. <https://www.netflixprize.com/assets/rules.pdf>.
- [2] E. Abbe, E. Boix-Adsera, and T. Misiakiewicz. The merged-staircase property: a necessary and nearly sufficient condition for sgd learning of sparse functions on two-layer neural networks. In *Conference on Learning Theory*, pages 4782–4887. PMLR, 2022.
- [3] C. C. Aggarwal. *Recommender Systems: The Textbook*. Springer Publishing Company, Incorporated, 1st edition, 2016.
- [4] G. Alain and Y. Bengio. What regularized auto-encoders learn from the data-generating distribution. *Journal of Machine Learning Research*, 15:3743–3773, 2014.
- [5] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.
- [6] S. Arora, N. Cohen, N. Golowich, and W. Hu. A convergence analysis of gradient descent for deep linear neural networks. In *International Conference on Learning Representations*, 2019.
- [7] S. Arora, N. Cohen, and E. Hazan. On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization. In *International Conference on Machine Learning (ICML)*, 2018.
- [8] S. Arora, N. Cohen, W. Hu, and Y. Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, 2019.
- [9] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, 2019.
- [10] S. Arora, S. S. Du, Z. Li, R. Salakhutdinov, R. Wang, and D. Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In *International Conference on Learning Representations*, 2020.
- [11] D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien. A Closer Look at Memorization in Deep Networks. In *International Conference on Machine Learning*, 2017.
- [12] A. Atanasov, B. Bordelon, and C. Pehlevan. Neural networks as kernel learners: The silent alignment effect. In *International Conference on Learning Representations*, 2022.
- [13] J. Ba, M. A. Erdogdu, T. Suzuki, Z. Wang, D. Wu, and G. Yang. High-dimensional asymptotics of feature learning: How one gradient step improves the representation. *arXiv preprint arXiv:2205.01445*, 2022.
- [14] Y. Bai and J. D. Lee. Beyond linearization: On quadratic and higher-order approximation of wide neural networks. In *International Conference on Learning Representations*, 2019.
- [15] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In *International Conference on Machine Learning (ICML)*, 2012.
- [16] P. L. Bartlett, P. M. Long, G. Lugosi, and A. Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070, 2020.

- [17] S. Bartunov, J. Rae, S. Osindero, and T. Lillicrap. Meta-learning deep energy-based memory models. In *International Conference on Learning Representation*, 2020.
- [18] D. Beaglehole, M. Belkin, and P. Pandit. Kernel ridgeless regression is inconsistent in low dimensions. *arXiv:2205.13525*, 2022.
- [19] T. Becker, K. Yang, J. C. Caicedo, B. K. Wagner, V. Dancik, P. Clemons, S. Singh, and A. E. Carpenter. Predicting compound activity from phenotypic profiles and chemical structures. *bioRxiv*, <https://doi.org/10.1101/2020.12.15.422887>, 2020.
- [20] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):711–720, 1997.
- [21] M. Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numerica*, 30:203–248, 2021.
- [22] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.
- [23] M. Belkin, D. Hsu, and P. Mitra. Overfitting or perfect fitting? Risk bounds for classification and regression rules that interpolate. In *Advances in Neural Information Processing Systems*, 2018.
- [24] M. Belkin, D. Hsu, and J. Xu. Two models of double descent for weak features. *Society for Industrial and Applied Mathematics Journal on Mathematics of Data Science*, 2(4):1167–1180, 2020.
- [25] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.
- [26] M. Belkin and P. Niyogi. Semi-supervised learning on riemannian manifolds. *Machine Learning*, 56:209–239, 2004.
- [27] M. Belkin, L. Rademacher, and J. Voss. Eigenvectors of Orthogonally Decomposable Functions. *Society for Industrial and Applied Mathematics Journal on Computing*, 47(2):547–615, 2018.
- [28] M. Belkin, A. Rakhlin, and A. Tsybakov. Does data interpolation contradict statistical optimality? In *International Conference on Artificial Intelligence and Statistics*, 2018.
- [29] A. Bellet, A. Habrard, and M. Sebban. Metric learning. *Synthesis lectures on artificial intelligence and machine learning*, 9(1):1–151, 2015.
- [30] A. Belyaeva, L. Cammarata, A. Radhakrishnan, C. Squires, K. Yang, G. Shivashankar, and C. Uhler. Causal network models of SARS-CoV-2 expression and aging to identify candidates for drug repurposing. *Nature Communications*, 12(1024), 2021.
- [31] A. Bietti, J. Bruna, C. Sanford, and M. J. Song. Learning single-index models with shallow neural networks. *arXiv preprint arXiv:2210.15651*, 2022.
- [32] A. Bietti and J. Mairal. On the inductive bias of neural tangent kernels. In *Advances in Neural Information Processing Systems*, 2019.
- [33] B. Bordelon and C. Pehlevan. Self-consistent dynamical field theory of kernel evolution in wide neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 32240–32256. Curran Associates, Inc., 2022.
- [34] R. Bouckaert and E. Frank. Evaluating the replicability of significance tests for comparing learning algorithms. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2004.

- [35] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.
- [36] J. Buhmann and K. Schulten. *Neural Computers*, chapter Storing Sequences of Biased Patterns in Neural Networks with Stochastic Dynamics. Springer-Verlag, 1989.
- [37] E. Candès and B. Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2012.
- [38] E. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 56(5):2053–2080, 2010.
- [39] L. Chen and S. Xu. Deep Neural Tangent Kernel and Laplace kernel have the same RKHS. In *International Conference on Learning Representations*, 2021.
- [40] M. Chen, J. Pennington, and S. Schoenholz. Dynamical isometry and a mean field theory of rnns: Gating enables signal propagation in recurrent neural networks. In *International Conference on Machine Learning*, pages 873–882. PMLR, 2018.
- [41] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.
- [42] Z. Cheng, M. Gadelha, S. Maji, and D. Sheldon. A Bayesian perspective on the deep image prior. In *Computer Vision and Pattern Recognition*, 2019.
- [43] Y. Cho and L. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, 2009.
- [44] A. Christmann and I. Steinwart. *Support Vector Machines*. Springer, 01 2008.
- [45] A. Coates, H. Lee, and A. Y. Ng. An analysis of single layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.
- [46] P. G. Constantine. *Active subspaces: Emerging ideas for dimension reduction in parameter studies*. SIAM, 2015.
- [47] C. Cortes, M. Mohri, and A. Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13:795–828, 2012.
- [48] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.
- [49] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. *Advances in Neural Information Processing Systems*, 14, 2001.
- [50] G. Cybenko. Approximation by superposition of sigmoidale function. *Mathematics of Control Signal and Systems*, 2:304–314, 01 1989.
- [51] S. B. Damelin and N. S. Hoang. On surface completion and image inpainting by biharmonic functions: Numerical aspects. *International Journal of Mathematics and Mathematical Sciences*, 2018.
- [52] A. Damian, J. Lee, and M. Soltanolkotabi. Neural networks can learn representations with gradient descent. In *Conference on Learning Theory*, pages 5413–5452. PMLR, 2022.
- [53] A. Daniely, R. F. Frostig, and Y. Singer. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances in Neural Information Processing Systems*, 2016.

- [54] L. Devroye, L. Györfi, and A. Krzyżak. The Hilbert kernel regression estimate. *Journal of Multivariate Analysis*, 65:209–227, 1998.
- [55] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*, volume 31. Springer Verlag, 1996.
- [56] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019.
- [57] S. S. Du, X. Zhai, B. Póczos, and A. Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019.
- [58] Y. Du and I. Mordatch. Implicit generation and generalization in energy-based models, 2019. arXiv:1903.08689.
- [59] S. Eger, P. Youssef, and I. Gurevych. Is it time to Swish? Comparing deep learning activation functions across NLP tasks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.
- [60] A. Faragó and G. Lugosi. Strong universal consistency of neural network classifiers. *IEEE Transactions on Information Theory*, 39(4), 1993.
- [61] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181, 2014.
- [62] R. A. Fisher. The statistical utilization of multiple measurements. *Annals of eugenics*, 8(4):376–386, 1938.
- [63] S. Fort, G. K. Dziugaite, M. Paul, S. Kharaghani, D. M. Roy, and S. Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:5850–5861, 2020.
- [64] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.
- [65] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.
- [66] B. Ghorbani, S. Mei, T. Misiakiewicz, and A. Montanari. Linearized two-layers neural networks in high dimension. *The Annals of Statistics*, 49(2):1029–1054, 2021.
- [67] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.
- [68] R. Gonzalez and R. Woods. *Digital Image Processing*, volume 4. Pearson, 2018.
- [69] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, volume 1. MIT Press, 2016.
- [70] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Neural Information Processing Systems Datasets and Benchmarks*, 2022.
- [71] S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11:23–63, 1987.
- [72] S. Gunasekar, B. E. Woodworth, S. Bhojanapalli, B. Neyshabur, and N. Srebro. Implicit regularization in matrix factorization. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6151–6159. Curran Associates, Inc., 2017.
- [73] B. Hanin and M. Nica. Finite depth and width corrections to the Neural Tangent Kernel. In *International Conference on Learning Representations*, 2020.

- [74] W. Härdle and T. M. Stoker. Investigating smooth multiple regression by the method of average derivatives. *Journal of the American statistical Association*, 84(408):986–995, 1989.
- [75] T. Hastie, A. Montanari, S. Rosset, and R. J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *arXiv:1903.08560*, 2019.
- [76] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning. *Aug, Springer*, 1, 01 2001.
- [77] S. Hayou, A. Doucet, and J. Rousseau. On the impact of the activation function on deep neural networks training. In *International Conference on Machine Learning*, 2019.
- [78] S. Hayou, A. Doucet, and J. Rousseau. Mean-field behaviour of Neural Tangent Kernel for deep neural networks. *arXiv:1905.13654*, 2021.
- [79] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *International Conference on Computer Vision*, 2015.
- [80] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*. Institute of Electrical and Electronics Engineers, 2016.
- [81] K. Hermann and A. Lampinen. What shapes feature representations? exploring datasets, architectures, and training. *Advances in Neural Information Processing Systems*, 33:9995–10006, 2020.
- [82] G. E. Hinton. *A practical guide to training restricted Boltzmann machines*, chapter 24, pages 599–619. Springer, 2012.
- [83] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.
- [84] R. Hodos, P. Zhang, H.-C. Lee, Q. Duan, Z. Wang, N. R. Clark, A. Ma’ayan, F. Wang, B. Kidd, J. Hu, D. Sontag, and J. Dudley. Cell-specific prediction and application of drug-induced gene expression profiles. *Pacific Symposium on Biocomputing*, 23:32–43, 2018.
- [85] J. J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.
- [86] J. J. Hopfield, D. I. Feinstein, and R. G. Palmer. ‘Unlearning’ has a stabilizing effect in collective memories. *Nature*, 304(5922):158–159, 1983.
- [87] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [88] M. Hristache, A. Juditsky, J. Polzehl, and V. Spokoiny. Structure adaptive approach for dimension reduction. *Annals of Statistics*, pages 1537–1566, 2001.
- [89] J. Huang and H.-T. Yau. Dynamics of deep neural networks and neural tangent hierarchy. In *International Conference on Machine Learning*, pages 4542–4551. PMLR, 2020.
- [90] K. Huang, Y. Wang, M. Tao, and T. Zhao. Why do deep residual networks generalize better than deep feedforward networks? — A Neural Tangent Kernel perspective. In *Advances in Neural Information Processing Systems*, 2020.
- [91] M. Huh, H. Mobahi, R. Zhang, B. Cheung, P. Agrawal, and P. Isola. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.
- [92] A. Hyvärinen and E. Oja. A Fast Fixed-Point Algorithm for Independent Component Analysis. *Neural Computation*, 9(7):1483–1492, 1997.
- [93] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, 2019.

- [94] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, 2015.
- [95] M. Iwata, L. Yuan, Q. Zhao, Y. Tabei, F. Berenger, R. Sawada, S. Akiyoshi, M. Hamano, and Y. Yamanishi. Predicting drug-induced transcriptome responses of a wide range of human cell lines by a novel tensor-train decomposition algorithm. *Bioinformatics*, 35(14):191–199, 2019.
- [96] A. Jacot, F. Gabriel, and C. Hongler. Neural Tangent Kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.
- [97] A. Jacot, B. Simsek, F. Spadaro, C. Hongler, and F. Gabriel. Kernel alignment risk estimator: Risk prediction from training data. *Advances in Neural Information Processing Systems*, 33:15568–15578, 2020.
- [98] J. Janková and S. van de Geer. De-biased sparse PCA: Inference for eigenstructure of large covariance matrices. *IEEE Transactions on Information Theory*, 67(4):2507–2527, 2021.
- [99] D. Kalimeris, G. Kaplun, P. Nakkiran, B. Edelman, T. Yang, B. Barak, and H. Zhang. Sgd on neural networks learns functions of increasing complexity. *Advances in neural information processing systems*, 32, 2019.
- [100] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.
- [101] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.
- [102] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, 2017.
- [103] B. Kosko. Bidirectional associative memories. *Institute of Electrical and Electronics Engineers Transactions on Systems, Man, and Cybernetics*, 18(1):49–60, 1988.
- [104] S. Kpotufe, A. Boularias, T. Schultz, and K. Kim. Gradients weights improve regression and classification. *Journal of Machine Learning Research*, 2016.
- [105] A. Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, University of Toronto, 2009.
- [106] O. Kwon, H. Lee, H.-J. Kong, E.-J. Kwon, J. Park, W. Lee, S. Kang, M. Kim, W. Kim, and H.-J. Cha. Connectivity map-based drug repositioning of bortezomib to reverse the metastatic effect of galnt14 in lung cancer. *Oncogene*, 39:1–14, 06 2020.
- [107] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the Institute of Electrical and Electronics Engineers*, 86(11):2278–2324, 1998.
- [108] J. Lee, Y. Bahri, R. Novak, S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as Gaussian processes. In *International Conference on Learning Representations*, 2017.
- [109] J. Lee, S. Schoenholz, J. Pennington, B. Adlam, L. Xiao, R. Novak, and J. Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. In *Advances in Neural Information Processing Systems*, 2020.
- [110] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, J. Sohl-Dickstein, and J. Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*, 2019.
- [111] A. Lewkowycz, Y. Bahri, E. Dyer, J. Sohl-Dickstein, and G. Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020.

- [112] B. Li. *Sufficient dimension reduction: Methods and applications with R*. Chapman and Hall/CRC, 2018.
- [113] K.-C. Li. On principal hessian directions for data visualization and dimension reduction: Another application of stein’s lemma. *Journal of the American Statistical Association*, 87(420):1025–1039, 1992.
- [114] Y. Li, C. Wei, and T. Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.
- [115] Z. Li, Z.-Q. J. Xu, T. Luo, and H. Wang. A regularized deep matrix factorized model of matrix completion for image restoration. *arXiv:2007.14581*, 2020.
- [116] T. Liang and H. Tran-Bach. Mehler’s formula, branching process, and compositional kernels of deep neural networks. *Journal of the American Statistical Association*, pages 1–35, 11 2020.
- [117] W. Little. The existence of persistent states in the brain. *Mathematical Biosciences*, 19(1):101–120, 1974.
- [118] C. Liu, L. Zhu, and M. Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. In *Advances in Neural Information Processing Systems*, 2020.
- [119] C. Liu, L. Zhu, and M. Belkin. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *arXiv:2003.00307*, 2020.
- [120] C. Liu, L. Zhu, and M. Belkin. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. *Applied and Computational Harmonic Analysis*, 01 2022.
- [121] J. Liu, P. Musialski, P. Wonka, and J. Ye. Tensor completion for estimating missing values in visual data. *Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence*, 35(1):208–220, 2013.
- [122] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.
- [123] P. M. Long. Properties of the after kernel. *arXiv preprint arXiv:2105.10585*, 2021.
- [124] S. Ma and M. Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems*, 2017.
- [125] S. Ma and M. Belkin. Kernel machines that adapt to GPUs for effective large batch training. In *Conference on Machine Learning and Systems*, 2019.
- [126] P. C. Mahalanobis. On the generalized distance in statistics. In *Proceedings of the National Institute of Science of India*, 1936.
- [127] R. McEliece, E. Posner, E. R. Rodemich, and S. S. Venkatesh. The Capacity of the Hopfield Associative Memory. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 33(4):461–482, 1987.
- [128] G. Meanti, L. Carratino, L. Rosasco, and A. Rudi. Kernel methods through the roof: handling billions of points efficiently. In *Advances in Neural Information Processing Systems*, 2020.
- [129] P. P. Mitra and C. Sire. Parameter-free statistically consistent interpolation: Dimension-independent convergence rates for Hilbert kernel regression. *arXiv:2106.03354*, 2021.
- [130] T. K. Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

- [131] A. Mousavi Hosseini, S. Park, M. Girotti, I. Mitliagkas, and M. A. Erdogu. Neural networks efficiently learn low-dimensional representations with sgd. In *International Conference on Learning Representations*, 2023.
- [132] S. Mukherjee and Q. Wu. Estimation of gradients and coordinate covariation in classification. *The Journal of Machine Learning Research*, 7:2481–2514, 2006.
- [133] S. Mukherjee, D.-X. Zhou, and J. Shawe-Taylor. Learning coordinate covariances via gradients. *Journal of Machine Learning Research*, 7(3), 2006.
- [134] M. Murray, V. Abrol, and J. Tanner. Activation function design for deep networks: Linearity and effective initialisation. *arXiv:2105.07741*, 2021.
- [135] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference in Learning Representations*, 2020.
- [136] R. Neal. *Bayesian Learning for Neural Networks*, volume 1. Springer, 1996.
- [137] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. *Advances in Neural Information Processing Systems (NIPS)*, 2011.
- [138] E. Nichani, A. Radhakrishnan, and C. Uhler. Increasing depth leads to U-shaped test risk in over-parameterized convolutional networks. In *International Conference on Machine Learning Workshop on Over-parameterization: Pitfalls and Opportunities*, 2021.
- [139] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. Schoenholz. Neural Tangents: Fast and easy infinite neural networks in Python. In *International Conference on Learning Representations*, 2020.
- [140] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.
- [141] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuglu. Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2016.
- [142] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.
- [143] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research British Machine Vision Association*, 12:2825–2830, 2011.
- [144] J. Pennington, S. Schoenholz, and S. Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. *Advances in neural information processing systems*, 30, 2017.
- [145] M. Pezeshki, O. Kaba, Y. Bengio, A. C. Courville, D. Precup, and G. Lajoie. Gradient starvation: A learning proclivity in neural networks. *Advances in Neural Information Processing Systems*, 34:1256–1272, 2021.
- [146] T.-H. Phan, Y. Qui, J. Zeng, L. Xie, and P. Zhang. A deep learning framework for high-throughput mechanism-driven phenotype compound screening and its application to COVID-19 drug repurposing. *Nature Machine Intelligence*, 3:247–257, 2021.
- [147] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

- [148] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in Neural Information Processing Systems*, 2016.
- [149] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. In *International Conference on Learning Representations Mathematical Reasoning in General Artificial Intelligence Workshop*, 2022.
- [150] S. Pushpakom, F. Iorio, P. A. Eyers, K. J. Escott, S. Hopper, A. Wells, A. Doig, J. Guillems, T. Latimer, C. McNamee, A. Norris, P. Sanseau, D. Cavalla, and M. Pirmohamed. Drug repurposing: progress, challenges and recommendations. *Nature Reviews Drug Discovery*, 18(1):41–58, 2019.
- [151] A. Radhakrishnan, D. Beaglehole, P. Pandit, and M. Belkin. Feature learning in neural networks and kernel machines that recursively learn features. *arXiv preprint arXiv:2212.13881*, 2022.
- [152] A. Radhakrishnan, M. Belkin, and C. Uhler. Overparameterized neural networks implement associative memory. *Proceedings of the National Academy of Sciences*, 117(44):27162–27170, 2020.
- [153] A. Radhakrishnan, M. Belkin, and C. Uhler. Wide and deep neural networks achieve consistency for classification. *Proceedings of the National Academy of Sciences*, 120(14):e2208779120, 2023.
- [154] A. Radhakrishnan, G. Stefanakis, M. Belkin, and C. Uhler. Simple, fast, and flexible framework for matrix completion with infinite width neural networks. *Proceedings of the National Academy of Sciences*, 119(16):e2115064119, 2022.
- [155] A. Rakhlin and X. Zhai. Consistency of interpolation with Laplace kernels is a high-dimensional phenomenon. In *Conference on Learning Theory*, 2019.
- [156] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. In *International Conference on Learning Representations*, 2017.
- [157] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, 2021.
- [158] B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *Society for Industrial and Applied Mathematics Review*, 52(3):471–501, 2010.
- [159] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *International Conference on Machine Learning (ICML)*, 2011.
- [160] D. A. Roberts, S. Yaida, and B. Hanin. *The Principles of Deep Learning Theory: An Effective Theory Approach to Understanding Neural Networks*. Cambridge University Press, 2022.
- [161] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention*, 2015.
- [162] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
- [163] I. Rubachev, A. Alekberov, Y. Gorishniy, and A. Babenko. Revisiting pretraining objectives for tabular deep learning. *arXiv preprint arXiv:2207.03208*, 2022.
- [164] R. Salakhutdinov and H. Larochelle. Efficient learning of deep Boltzmann machines. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.
- [165] S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein. Deep information propagation. *arXiv preprint arXiv:1611.01232*, 2016.
- [166] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

- [167] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *International Conference on Computer Vision*, pages 618–626, 2017.
- [168] H. Shah, K. Tamuly, A. Raghunathan, P. Jain, and P. Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.
- [169] Z. Shi, J. Wei, and Y. Lian. A theoretical analysis on feature learning in neural networks: Emergence from inputs and advantage over fixed features. In *International Conference on Learning Representations*, 2022.
- [170] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, 2017.
- [171] S. Singla and S. Feizi. Salient ImageNet: How to discover spurious features in deep learning? In *International Conference on Learning Representations*, 2022.
- [172] A. Sinha and J. C. Duchi. Learning kernels with random features. *Advances in Neural Information Processing Systems*, 29, 2016.
- [173] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *JOSA A*, 4(3):519–524, 1987.
- [174] G. Somepalli, M. Goldblum, A. Schwarzschild, C. B. Brusa, and T. Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.
- [175] C. Squires, D. Shen, A. Agarwal, D. Shah, and C. Uhler. Causal imputation via synthetic interventions. *arXiv:2011.03127*, 2020.
- [176] E. M. Stein and R. Shakarchi. *Complex Analysis*, volume II. Princeton University Press, 2003.
- [177] S. Strogatz. *Nonlinear Dynamics and Chaos*, volume 2. Westview Press, 2015.
- [178] A. Subramanian, R. Narayan, S. M. Corsello, et al. A next generation connectivity map: L1000 platform and the first 1,000,000 profiles. *Cell*, 171(6):1437–1452, 2017.
- [179] J. Tachella, J. Tang, and M. Davies. The neural tangent link between cnn denoisers and non-local filters. *arXiv:2006.02379*, 2020.
- [180] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- [181] S. Trivedi, J. Wang, S. Kpotufe, and G. Shakhnarovich. A consistent estimator of the expected gradient outerproduct. In *UAI*, pages 819–828, 2014.
- [182] R. Tsuchida, F. Roosta-Khorasani, and M. Gallagher. Invariance of weight distributions in rectified MLPs. In *International Conference on Machine Learning*, 2018.
- [183] K. Tunyasuvunakool, J. Adler, Z. Wu, T. Green, M. Zielinski, A. Židek, A. Bridgland, A. Cowie, C. Meyer, A. Laydon, S. Velankar, G. Kleywegt, A. Bateman, R. Evans, A. Pritzel, M. Figurnov, O. Ronneberger, R. Bates, S. Kohl, and D. Hassabis. Highly accurate protein structure prediction for the human proteome. *Nature*, 596:1–9, 2021.
- [184] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Conference on Computer Vision and Pattern Recognition*, pages 586–587. IEEE Computer Society, 1991.
- [185] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep Image Prior. In *Conference on Computer Vision and Pattern Recognition*, pages 9446–9454. Institute of Electrical and Electronics Engineers, 2018.

- [186] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.
- [187] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu. scikit-image: image processing in Python. *PeerJ*, 2:e453, 2014.
- [188] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*, 2008.
- [189] N. Vyas, Y. Bansal, and P. Nakkiran. Limitations of the ntk for understanding generalization in deep learning. *arXiv preprint arXiv:2206.10012*, 2022.
- [190] G. Wahba. *Spline models for observational data*. SIAM, 1990.
- [191] T. Wang, S. Buchanan, D. Gilboa, and J. Wright. Deep networks provably classify data on curves. In *Advances in Neural Information Processing Systems*, 2021.
- [192] T. Wang, D. Zhao, and S. Tian. An overview of kernel alignment and its applications. *Artificial Intelligence Review*, 43:179–192, 2015.
- [193] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, 2000.
- [194] G. Williams, A. Gatt, E. Clarke, J. Corcoran, P. Doherty, D. Chambers, and C. Ballard. Drug repurposing for Alzheimer’s disease based on transcriptional profiling of human iPSC-derived cortical neurons. *Translational Psychiatry*, 9, 2019.
- [195] J. K. Winkler, C. Fink, F. Toberer, A. Enk, T. Deinlein, R. Hofmann-Wellenhof, L. Thomas, A. Lallas, A. Blum, W. Stolz, et al. Association between surgical skin markings in dermoscopic images and diagnostic performance of a deep learning convolutional neural network for melanoma recognition. *JAMA dermatology*, 155(10):1135–1141, 2019.
- [196] X. Wu, S. S. Du, and R. Ward. Global convergence of adaptive gradient methods for an over-parameterized neural network. *arXiv:1902.07111*, 2019.
- [197] Y. Xia, H. Tong, W. K. Li, and L.-X. Zhu. An adaptive estimation of dimension reduction space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(3):363–410, 2002.
- [198] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, 2018.
- [199] L. Xiao, J. Pennington, and S. Schoenholz. Disentangling trainability and generalization in deep learning. In *International Conference on Machine Learning*, 2019.
- [200] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolution network, 2015. arXiv:1505.00853.
- [201] H. Xue, S. Zhang, and D. Cai. Depth image inpainting: Improving low rank matrix completion with low gradient regularization. *Institute of Electrical and Electronics Engineers Transactions on Image Processing*, 26:4311–4320, 2017.
- [202] G. Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019.
- [203] G. Yang and E. Hu. Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, 2020.
- [204] G. Yang and S. Schoenholz. Mean field residual networks: On the edge of chaos. In *Advances in Neural Information Processing Systems*, 2017.

- [205] G. Yang and S. Schoenholz. Deep mean field theory: Layerwise variance and width variation as methods to control gradient explosion. In *International Conference on Learning Representations*, 2018.
- [206] S. Zagoruyko and N. Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016.
- [207] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.
- [208] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.
- [209] C. Zhang, S. Bengio, M. Hardt, and Y. Singer. Identity crisis: Memorization and generalization under extreme overparameterization. In *International Conference on Learning Representations*, 2020.
- [210] C.-H. Zhang and S. S. Zhang. Confidence intervals for low dimensional parameters in high dimensional linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1):217–242, 2014.
- [211] L. Zhu, C. Liu, A. Radhakrishnan, and M. Belkin. Quadratic models for understanding neural network dynamics. *arXiv preprint arXiv:2205.11787*, 2022.